VxWorks 653 For Application Developers Lab Guide

Copyright © Wind River Systems, Inc.

Version 2.2 A, May 2008

All rights reserved. No part of this publication may be reproduced in any form, by photocopy, microfilm, retrieval system, or by any other means now known or hereafter invented without the prior written permission of Wind River System, Inc.

This document is designed to support Wind River training.

For information about the Wind River Systems training program, see:

https://education.windriver.com

Education Services
Wind River Systems, Inc.
500 Wind River Way
Alameda, CA 94501
510-748-4100 (phone)
510-749-2454 (fax)
Email: training@windriver.com

For training in Europe, Asia and Africa, please contact your local sales office.

Wind River, the Wind River logo, and VxWorks are registered trademarks of Wind River Systems, Inc. Any third party trademarks referenced are the property of their respective owners. For further information regarding Wind River trademarks, please see www.windriver.com/company/terms/trademark.html

Table of Contents

Labs

1. Lab Setup

2. Getting Started Lab – Hardware Target

3. Getting Started Lab – Simulator Target

4. System Concepts Lab

5. Configuration and Build (Network Image) Lab

6. Configuration and Build (RAM Payload) Lab

7. Debugging Lab

8. Visualization Lab

9. ARINC653 API Lab

10. I/O Partition Lab

© 2008 Wind River Systems, Inc.

Lab Setup

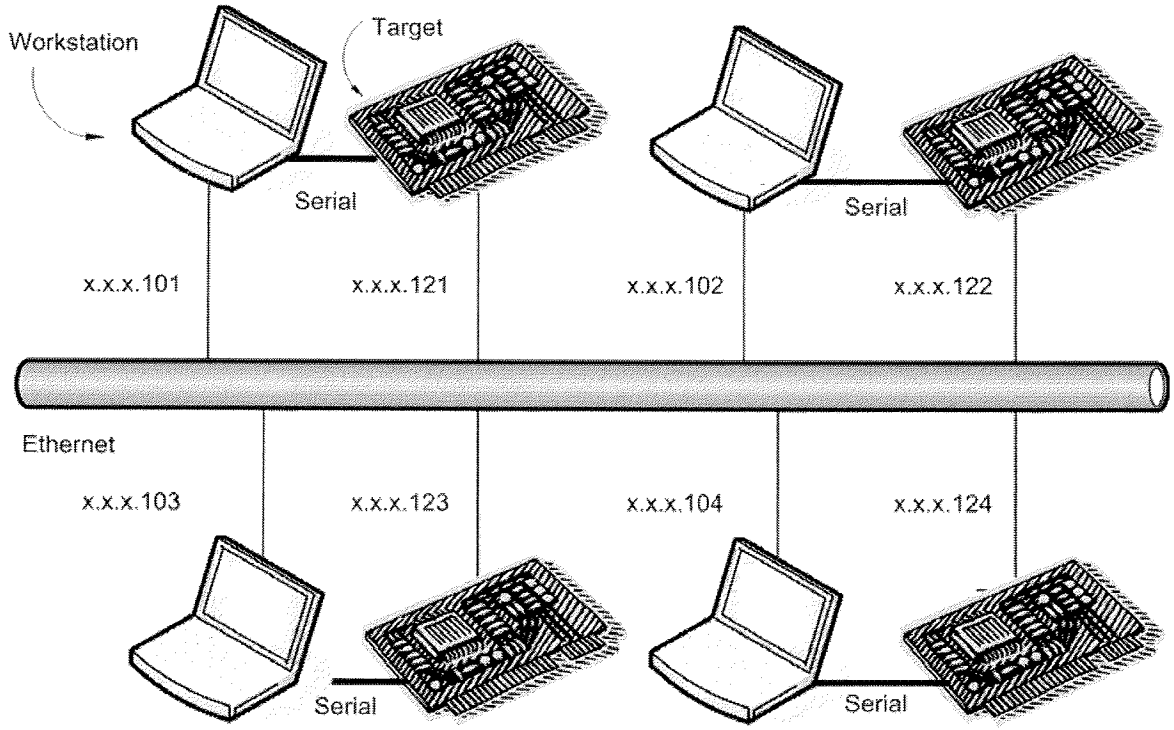
© 2008 Wind River Systems, Inc.

WIND RIVER

Lab Setup

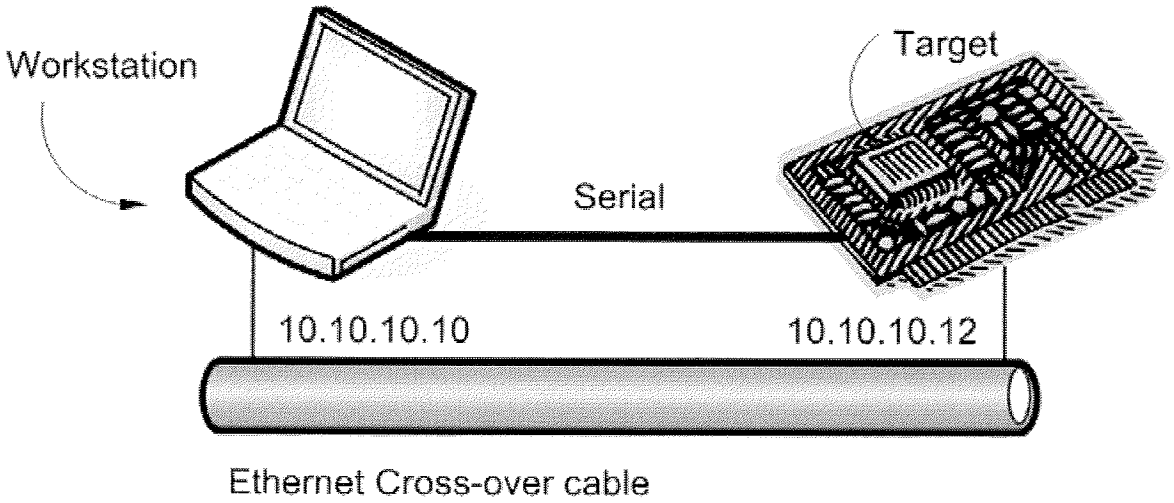
Lab Environment (Multiple User Network)

Each team works at a workstation and has a separate account.



Lab Environment (Single User Network)

Each team works at a standalone workstation.



In some training facilities hardware targets are supported. If hardware targets are not available, labs can be done on simulators. The user name and password configuration are shown in the table below.

User name	Password	Home directory
Training	Training	C:\

Check with your instructor for your particular lab environment setup.

When logging in for the first time, if asked to reboot the machine because new hardware was found, do so.

Each computer should have VxWorks 653 Platform, 2.2 installed on it.

Labs are written for use on a Windows host and using either the simulator or a real hardware target.

The following locations are useful to know:

C:\WindRiver Installation directory for VxWorks 653 Platform, 2.2

C:\VxWorks653_Labs Folder containing original copies of labs

C:\myLabs Placeholder for your labs copying, modifications and build

Configuration for real hardware targets

Boot parameters for the VxWorks targets are set over a RS-232 serial line.

Summary

As some of the information may be volatile, we suggest you fill out the following table with the values for your team.

Item	Lab Reference	Your team's values
Home Lab directory	*C:\myLabs*	
VxWorks 653 Root Directory	*C:\WindRiver*	
Target name	*Target*	
Target IP address	*TargetIP*	
Boot host name	*BootHost*	
Boot host IP address	*BootHostIP*	
FTP Server User ID	*FTP Username*	
FTP Server Password	*FTP Password*	

Getting Started Lab

Hardware Target

© 2008 Wind River Systems, Inc.

WIND RIVER

Getting Started Lab
(Hardware Target)

Objective: In this lab, you will *use the online help pages to access information about VxWorks 653 routines and libraries, boot VxWorks 653 and use the Target Manager to configure and manage target servers.*

Note: This lab should take approximately 35 minutes.

Lab Overview

Using Workbench:

- *Launching the online documentation*
- *Using the help information*
- *Building indices for online manuals*

Using Workbench with a target:

- *Launching a default VxWorks image*
- *Configuring and Managing target connections*
- *Using the Host Shell*
- *Launching Target Tools via the Workbench IDE*

Using the Online documentation

1. Select **Start → All Programs → Wind River → Workbench 2.6 → Wind River Workbench 2.6**.

2. In the Workspace Launcher window keep the default **workspace** and click **OK**.

3. In the **Welcome** screen select **Workbench**.
   The **Application Development** perspective is shown.

4. Select **Help → Help Contents** from the Wind River Workbench menu bar.
   The help browser will open in your web browser, with contents (an index of documents) in the gutter at the left.
   Let's start with the **VxWorks 653 Core OS API Reference** manual.
   This reference contains information on VxWorks 653 libraries and routines.

© 2008 Wind River Systems, Inc.

5. In the left gutter, click on **Wind River Documentation → References → Operating System → VxWorks 653 Core OS API Reference**.
   Begin by looking at information on a library.

6. From the list of references in the right pane, navigate your way to **apexPortLib**.

7. Open up this page and look at the documentation on the APEX port library.

   Now you will look at the documentation for the routine **read()**.

8. Use the fact that **read()** belongs to **ioLib**, and click on **VxWorks 653 Core OS API Reference**.

9. From the right pane, click on **ioLib → read( )**.
   Occasionally, however, you won't know which library a routine belongs to.
   In these instances, the 'index' may be used.
   The 'index' is only available for certain manuals.

10. Use the 'Routines Index' directly, by clicking on **Wind River Documentation → References → Operating System → VxWorks 653 Core OS API Reference → Routines Index**, and then selecting **read** directly.

Next you are going to take a look at some of the Wind River Workbench 2.6 documentation.

11. Select **Wind River Documentation → Getting Started → Wind River Workbench Tutorial 2.6 VxWorks Version**.

12. Click on the **home** icon at the top of the right pane to see a table of contents for the Wind River Workbench tools.

13. Click on **Wind River Workbench Tutorials** then **Starting Wind River Workbench** to see this chapter on the right hand side of the browser.

14. Go back to the contents gutter, and click on **Wind River Documentation → Guides → Operating System → VxWorks 653 Programmer's Guide, 2.2**.

15. Click on the **Search Scope** link at the top left of the window.

16. Select **Search only the following topics**, and **New...**.
    A new window will open.

17. Select **Wind River Documentation → Guides**.

18. In the **List name** field at the top, type **VxWorks 653**, press **OK**, and then press **OK** again.

© *2008* Wind River Systems, Inc.

19. Type **read** into the search text box and click on the **GO** button to the right of the Search box.

20. Search results will be displayed in the gutter on the left.

The indexing may take a few minutes if this is the first time that this operation has been performed since the Workbench installation.
Next, suppose you want to get information about booting a VxWorks 653 target.

21. Click on the **Search** box on the upper left hand corner of the help browser, and enter **booting** in the text box. Click **GO** (or 'Enter').
    This should give you a list of topics on 'booting'.

22. Click on the first topic, **Boot**, to open the corresponding chapter in your browser.

23. Close the web browser.

Booting

1. Start a serial session between your PC and target by selecting the **Terminal** tab in the lower middle pane of Workbench.

2. Click on the connect icon (the two green balls connected by a green "N" shape).

3. Power-on the target.

4. When you see the 'auto boot' countdown, hit any key.
   Note that the CPU type and version number of the BSP is printed (very useful if you aren't sure of the type of target board you are using).
   The targets have been set up with most of the boot parameters cleared.
   To boot VxWorks 653, you need to know:
   - Internet address of host computer (BootHostIP)
   - Internet address for the target (TargetIP)
   - Name of the host (BootHost)
   - Pathname of the text file containing information about the system you wish to boot, for example `C:\WindRiver\vxworks653-2.2\target\proj\bspName_default\integration\boot.txt`
   - Name of your target (Target)
   - Boot device (network interface used to download boot file).
   - FTP User name and password (LoginName)

We will be using FTP to download the VxWorks 653 image to the target.

5. Start the FTP server by by selecting **Start → All Programs → Wind River → VxWorks 653 2.2 for PowerPC → FTP Server**.

6. Minimize, but do not close, the FTP Server window (**WFTPD**).

7. In the Workench **Terminal** pane, type *c* at the boot prompt to define, or change, the boot parameters.

```
[VxWorks 653 Boot]: c
```

   To help in modifying the boot parameters, use the following keys:

Key	Description
<backspace>	Change last character typed (or whatever key maps to a <ctlr>H)
.	(at start of line) Clear current entry
-	Go back to last parameter
<ctrl>D	Done modifying parameters, returns to boot parameter prompt
<ctrl>U	Erase current line

8. Enter the following boot parameters:

```
boot device : mv
processor number : 0
host name :
file name: C:\WindRiver\vxworks653-
2.2\target\proj\wrSbc750gx_default\integration\boot.txt
inet on ethernet (e) : 10.10.10.1
inet on backplane (b) :
host inet (h) : 10.10.10.10
gateway inet (g):
user (u) : training
ftp password (pw) (blank = use rsh): training
flags (f) : 0x0
target name (tn) : vx653
startup script (s) :
other (o):
```

9. To verify that the settings are correct, type the ***p*** command (print) at the boot prompt:
   `[VxWorks 653 Boot]: p`

```
boot device          : mv
unit number          : 0
processor number     : 0
host name            :
file name            :\WindRiver\vxworks653-
2.2\target\proj\wrSbc750gx_default\integration\boot.txt
inet on ethernet (e) : 10.10.10.1
host inet (h)        : 10.10.10.10
user (u)             : training
ftp password (pw)    : training
flags (f)            : 0x0
target name (tn)     : vx653
```

   The only parameters displayed are those which have been defined.

10. To boot VxWorks 653, type @ at the boot prompt:
    `[VxWorks 653 Boot]: @`

```
boot device : mv
unit number : 0
processor number : 0
file name : C:\WindRiver\vxworks653-
2.2\target\proj\wrSbc750gx_default\integration\boot.txt
inet on ethernet (e) : 10.10.10.1
host inet (h) : 10.10.10.10
user (u) : training
ftp password (pw) : training
flags (f) : 0x0
target name (tn) : vx653

Attached TCP/IP interface to mv0.
Warning: netmask value is 0.
Attaching interface lo0...done
Loading : C:\WindRiver\vxworks563-
2.2\target\proj\wrSbc750gx_default\integration\boot.txt

sm0=C:\WindRiver\vxworks653-
2.2\target\proj\wrSbc750gx_default\integration\configRecord.reloc
0x00001b6c + (0x000fe494)

sm1=C:\WindRiver\vxworks653-
2.2\target\proj\wrSbc750gx_default\integration\coreOS.sm
0x000d0944 + 0x0000f958 + 0x00004fdc + 0x00021cf4

sm2=C:\WindRiver\vxworks653-
2.2\target\proj\wrSbc750gx_default\integration\vxSysLib.sm
0x000312e4 + 0x00002af4 + 0x00005b50 + 0x00001c28
Warning: the number of loaded sections (4) is less than the number of configRecord
black boxes (6).

sm3=C:\WindRiver\vxworks653-
2.2\target\proj\wrSbc750gx_default\integration\wrSbc750gx_part1.sm
0x0000315c + 0x0000002c + 0x00000010 + 0x00000008
Warning: the number of loaded sections (4) is less than the number of configRecord
black boxes (6).

Starting at 0x100000...

Attached TCP/IP interface to mv unit 0
Attaching interface lo0...done


            VxWorks 653
```

```
Copyright (c) 1984-2007 Wind River Systems, Inc.

           CPU: wrSbc750gx IBM Power PC
   Runtime Name: VxWorks 653
Runtime Version: 2.2
    BSP version: 1.4/5
        Created: Nov  3 2007, 21:14:27
            WDB: Ready.
```

The target has now successfully booted!

Configuring the Target Server

Introduction

When using a real hardware target
1. The hardware target is booted independently of Wind River Workbench.
2. A target server must be started which will allow the (host) tools to connect to the target.

Using the Target Manager

1. Click on the **Target Manager** tab in the lower left pane of Wind River Workbench. Functionality from within the **Target Manager** view allows you to create and save new target server configurations as well as start target servers from new or saved configurations.

2. From within the **Target Manager** tab, add a new target to your local host by left-clicking on the **Create A New Target Connection** icon .
This will launch the **New Connection** wizard.

3. In the window that pops up, select **Wind River VxWorks 653 Target Server Connection**.

4. Click on **Next**.

5. Choose **wdbrpc** as the backend, since we will be using ethernet to communicate between the host and the target.

6. In the **Target Name / IP Address** field, enter the following IP address **10.10.10.1**, and click on **Next**.

7. Review the information on the **Object Path Mappings** pane, and click **Next**.

8. Click **Next** through the **Target State Refresh** and **Default Breakpoint Options** panes.

9. In the **Connection Summary** dialog, change the **Connection name:** to **my-session** and uncheck the checkbox **Immediately connect to target if possible**.

10. Click on **Finish**.
In the **Target Manger** pane will be the newly created target connection displayed as **my-session (VxWorks 653 2.2)**.

    We will next explore and change some of the properties of the target server connection **my-session**. We will configure an option into the target server that will

allow one of the host tools (System Viewer) to upload data collected on the target to the host.

11. Right-click on **my-session(VxWorks 653 2.2)**, select **Properties**.

12. In the **Target Server Options** tab, go down to the **Advanced target server options** pane, and click on **Edit...**.

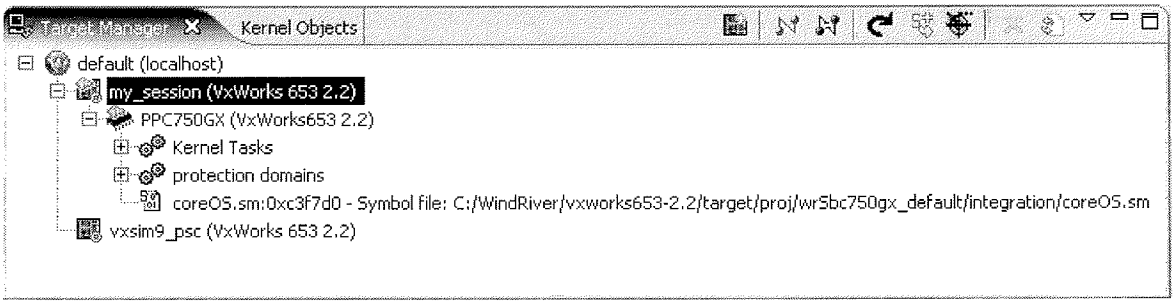
13. In the **Common** tab, and **Target Server File System** properties, make sure that the **Enable target server file system** is checked and set the **Root** to **C:\temp\tsfs**.

14. Also, ensure the **Make Target Server File System writable** checkbox is selected.

15. Explore other categories if you wish, then click on **OK**.
    The **Memory** tab allows one to control the size of the target server memory cache.
    The **Logging** tab contains controls to enable and set the locations of log files.
    The **Symbols** tab allows you to select what kernel symbols are loaded by the target server.

16. Hit **OK**, and **OK** again to apply the changes made.

17. To start our custom configured target server, right-click **my-session (VxWorks 653 2.2)** and choose **Connect 'my-session'** from the context menu (make sure that your target is still running!)
    Several things should happen:
    - If you open **my-session(VxWorks 653 2.2)** (click on the **plus** icon) in the **Target Manager**, you will see information about the active session, and some information about the target to which we have just connected.
    - The connection icon (to the left of **my-session**) should now have a target-server icon to indicate that a target server has been started (a target session is active) for this particular target and should have a check-mark as well to indicate it's connected .
    - A target server icon should appear in the taskbar system tray in the bottom right-hand corner of your screen. (To open the target server log window and see information about the connection status, double click on this icon).



You can now start any of the available tools from the Wind River Workbench IDE.

Starting Tools from the Workbench IDE

1. If not already connected, connect a target server by right clicking on **my_session (VxWorks 653 2.2)** and selecting **Connect 'my_session'**.

   To connect a tool to a target requires first highlighting the target connection and then selecting the tool from the **Target** menu of the Wind River Workbench development environment.

   The Wind River Workbench IDE supports working with multiple targets.
   You can add more than one target to your local host.

   As soon as you create a target connection, you would see another connection button in the **Target Manager** pane.

   To connect a tool to this target, click on the target connection, and then select the tool that you want to connect from the **Target** menu.

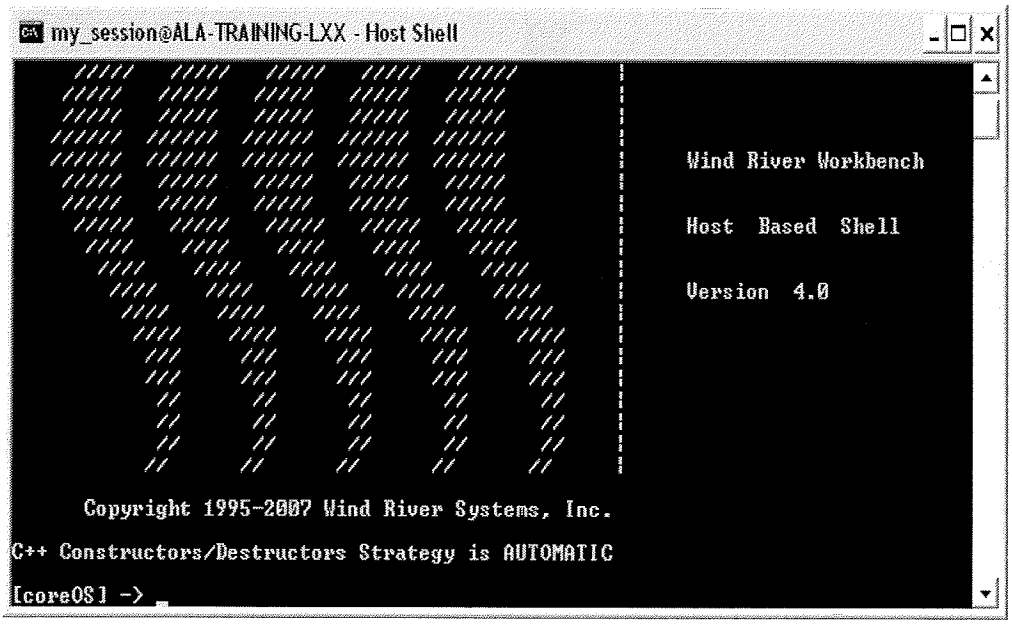
   This allows you to work simultaneously on multiple targets, for example a simulator target and a real target.

2. Start a Host shell by clicking on the **my_session (VxWorks 653 2.2)** and then selecting **Target → Host Shell** from the menu bar.

   A **Start Host Shell** window appears.

3. Set any options you desire, and press **OK**.

   A **Host Shell** window opens outside the Wind River Workbench IDE with a logo and version information.



The Host Shell may also be opened by right-clicking on the connection in the **Target Manager**, then selecting **Target Tools → Host Shell**.

4. To terminate the Host Shell session, simply type **quit** at the Host Shell prompt:
   ```
   [coreOS] -> quit
   ```

   or just close the **Host Shell** window.

   Other tools may now also be launched.

5. Close Workbench.

This concludes the lab. Do not continue.

Getting Started Lab
Simulator

© 2008 Wind River Systems, Inc.

WIND RIVER

Getting Started Lab
(Simulator)

Objective: In this lab, you will *use the online help pages to access information about VxWorks 653 routines and libraries, boot the VxWorks 653 simulator and use the Target Manager to configure and manage target servers.*

Note: This lab should take approximately 35 minutes.

Lab Overview

Using Workbench:

- *Launching the online documentation*
- *Using the help information*
- *Building indices for online manuals*

Using Workbench with a target:

- *Launching a simulator*
- *Configuring and Managing target connections*
- *Using the Host Shell*
- *Launching Target Tools via the Workbench IDE*

Using the Online Documentation

1. Select **Start → All Programs → Wind River → Workbench 2.6 → Wind River Workbench 2.6**.

2. In the Workspace Launcher window keep the default **workspace** and click **OK**.

3. In the **Welcome** screen select **Workbench**.
   The **Application Development** perspective is shown.

4. Select **Help → Help Contents** from the Wind River Workbench menu bar.
   The help browser will open in your web browser, with contents (an index of documents) in the gutter at the left.

Let's start with the **VxWorks 653 Core OS API Reference** manual.
This reference contains information on VxWorks 653 libraries and routines.

© 2008 Wind River Systems, Inc.

5. In the left gutter, click on **Wind River Documentation → References → Operating System → VxWorks 653 Core OS API Reference**.
   Begin by looking at information on a library.

6. From the list of references in the right pane, navigate your way to **apexPortLib**.

7. Open up this page and look at the documentation on the APEX port library.

Now you will look at the documentation for the routine **read()**.

8. Use the fact that **read()** belongs to **ioLib**, and click on **VxWorks 653 Core OS API Reference**.

9. From the right pane, click on **ioLib → read( )**.
   Occasionally, however, you won't know which library a routine belongs to.
   In these instances, the 'index' may be used.
   The 'index' is only available for certain manuals

10. Use the 'Routines Index' directly, by clicking on **Wind River Documentation → References → Operating System → VxWorks 653 Core OS API Reference → Routines Index**, and then selecting **read** directly.

Next you are going to take a look at some of the Wind River Workbench 2.6 documentation.

11. Select **Wind River Documentation → Getting Started → Wind River Workbench Tutorial 2.6 VxWorks Version**.

12. Click on the **home** icon at the top of the right pane to see a table of contents for the Wind River Workbench tools.

13. Click on **Wind River Workbench Tutorials** then **Starting Wind River Workbench** to see this chapter on the right hand side of the browser.

14. Go back to the contents gutter, and click on **Wind River Documentation → Guides → Operating System → VxWorks 653 Programmer's Guide, 2.2**.

15. Click on the **Search Scope** link at the top left of the window.

16. Select **Search only the following topics**, and **New...**.
    A new window will open.

17. Select **Wind River Documentation → Guides**.

18. In the **List name** field at the top, type **VxWorks 653**, press **OK**, and then press **OK** again.

© *2008* Wind River Systems, Inc.

19. Type **read** into the search text box and click on the **GO** button to the right of the Search box.
    Search results will be displayed in the gutter on the left.
    The indexing may take a few minutes if this is the first time that this operation has been performed since the Workbench installation.
    Next, suppose you want to get information about booting a VxWorks 653 target.

20. Click on the **Search** box on the upper left hand corner of the help browser, and enter **booting** in the text box. Click **GO** (or 'Enter').
    This should give you a list of topics on 'booting'.

21. Click on the first topic, **Boot**, to open the corresponding chapter in your browser.

22. Close the web browser

© *2008* Wind River Systems, Inc.

Launching the Simulator and the Tools

Booting the Simulator via Workbench

1. Start the Wind River Workbench development environment (if not already started) by selecting **Start → All Programs → Wind River → Workbench 2.6 → Wind River Workbench 2.6**

2. In the **Target Manager** pane, create a new simulator connection by clicking on the **Create a New Target Connection,** .
   Functionality from within the **Target Manager** view allows you to create and save new target server configurations as well as start target servers from new or saved configurations.
   In the case of a simulator target, the simulator is launched automatically before the target server, by default.

3. In the **Connection Type** pane, select **Wind River VxWorks 653 Simulator Connection** and click on **Next.**

4. In the **VxWorks 653 Boot Parameters** pane, select **Custom simulator:** and **Browse...** to the following **boot.txt**: **C:\WindRiver\vxworks653-2.2\target\proj\ simpc_default\ integration\boot.txt**.

5. Click on **Open**, and then **Next**.

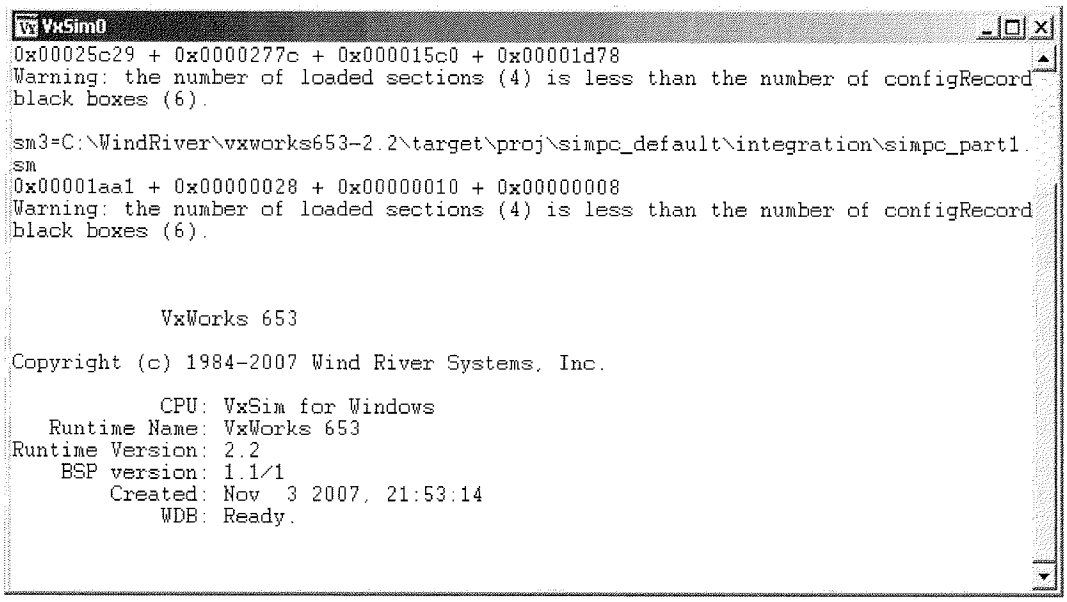
6. In the **Target Sever Options** pane accept the default 'Backend' setting, **wdbpipe**, and click on **Next**.

7. In the **Object Path Mappings** pane, verify that **Load module symbols to debug server automatically if possible** is enabled, and click on **Next**.

8. Click **Next** through the **Target State Refresh** and **Default Breakpoint Options** panes.

9. In the **Connection Summary** pane, set the 'Connection name:' to **Lab-vxsim1**, select **Immediately connect to target if possible**, and then click **Finish**.

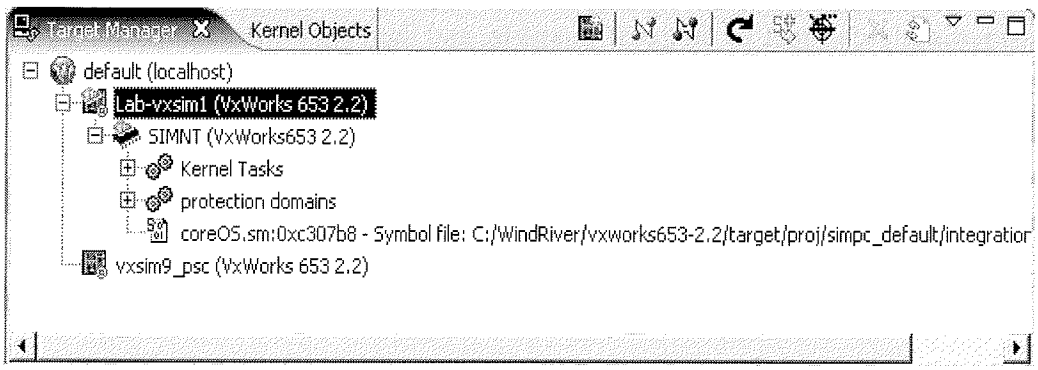
The following window opens outside the Workbench IDE, which is the simulator target:



Do **not** close this window; doing so will shut down the simulator as well!

The default simulator is only configured to run the Core OS and a single partition. Several other things should happen:

- The target icon (to the left of **Lab-vxsim1**) should now be changed to indicate that a target server has been started (a target session is active) for this particular target, and the target is attached
- You can see information about the active session in the **Target Manager**, including the CPU type **SIMNT** and the OS **VxWork653 2.2**.
- A target server icon should appear in the taskbar system tray in the bottom right-hand corner of your screen. (To open the target server log window and see information about the connection status, double click on this icon).
- In the **Target Manager** pane, you may now open sub-items on your connection to view more information about the running system.



Any of the available debug tools can now be started from the Workbench IDE.

10. Close the simulator. This can be done in one of the following ways:
    a. Right click on **Lab-vxsim1 (VxWorks 653 2.2)** and select **Disconnect 'Lab-vxsim1';**
    b. Select **Lab-vxsim1 (VxWorks 653 2.2)** and click on the **Disconnect Lab-vxsim1** icon;
    c. Close the simulator console window.

11. Shut down the target server by right clicking on the Target Server icon in the system tray, and selecting 'Shutdown'.

Booting the Simulator from the Command Line

The simulator can be started from the command line.

1. Open the VxWorks 653 2.2 'Development Shell' by clicking **Start → All Programs → Wind River → VxWorks 653 2.2 for PowerPC → VxWorks 653 2.2 Development Shell.**

2. Change the current directory to the project directory that contains **boot.txt.**

   → `cd C:\WindRiver\vxworks653-2.2\target\proj\simpc_default\integration`

3. Type **vxsim.**

   → `vxsim`

The simulator target opens. Note however that it is stand alone; there is no target server attached to it.

4. To get vxsim options, from the **VxWorks 653 2.2 Development Shell**, type

   → `vxsim -help`

Starting Tools from the Workbench IDE

1. Connect a target server to the previous custom built simulator by right clicking on **Lab-vxsim1 (VxWorks 653 2.2)** and selecting **Connect 'Lab-vxsim1'**.

To connect a tool to a target requires first highlighting the target connection and then selecting the tool from the **Target** menu of the Wind River Workbench development environment.

The Wind River Workbench IDE supports working with multiple targets.
You can add more than one target to your local host.

As soon as you create a target connection, you would see another connection button in the **Target Manager** pane.

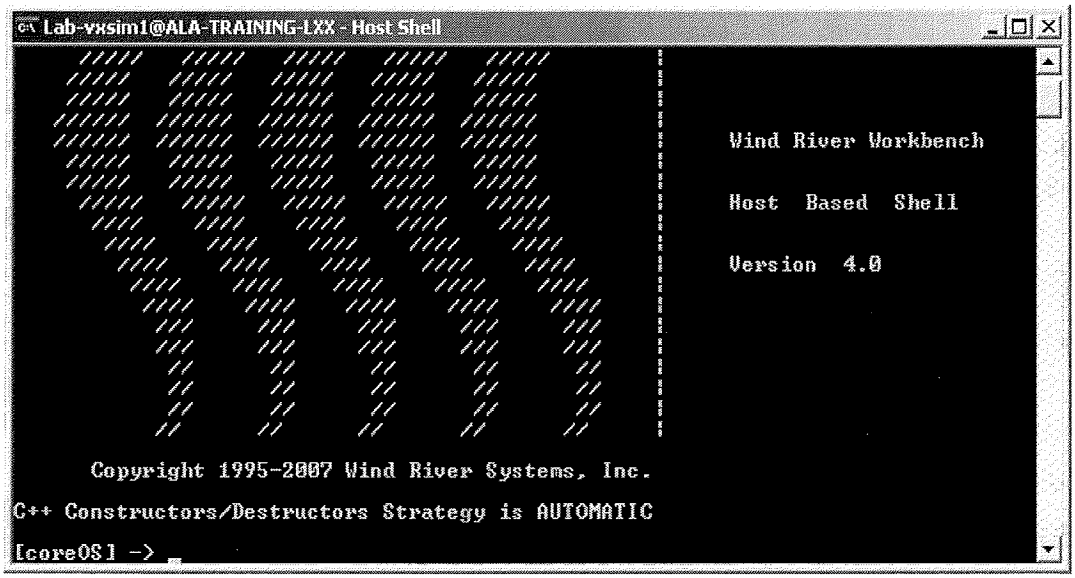
To connect a tool to this target, click on the target connection, and then select the tool that you want to connect from the **Target** menu.

This allows you to work simultaneously on multiple targets, for example a simulator target and a real target.

2. Start a Host shell by clicking on the **Lab-vxsim1 (VxWorks 653 2.2)** and then selecting **Target → Host Shell.**
   A **Start Host Shell** window appears.

3. Set any options you desire, and press **OK**.

   A **Host Shell** window opens outside the Wind River Workbench IDE with a logo and version information.



   The Host Shell may also be opened by right-clicking on the connection in the **Target**

Manager, then selecting **Target Tools → Host Shell**.

4. To terminate the Host Shell session, simply type **quit** at the Host Shell prompt:
   ```
   [coreOS] -> quit
   ```

   or just close the **Host Shell** window.

Other tools may now also be launched.

Configuring the Target Server

Introduction

So far, we launched a simulator and a target server was launched *automatically* for this simulator target.

When using a real hardware target, the process is a little bit different:

1. Boot the hardware target independently of Workbench.
2. Start a target server which will allow the tools to connect to the target.

In the following lab we will configure and launch a new, custom target communications session, as you would do with a hardware target.

Using the Target Manager

1. Check that you still have the standard simulator running (the simulator window still exists).

2. Kill the target server by right clicking on the session name in the **Target Manager** pane, and selecting **Disconnect**, or left-click on the session-name and click on the **Disconnect** icon.
   The communications channel will be shut down, and the simulator window will close.

3. In the **Target Manager** pane, right click on the **Lab-vxsim1 (VxWorks 653 2.2)** session and select **New... → Connection....**
   This opens the **New Connection** wizard, allowing us to customize a new connection.

4. Select **Wind River VxWorks 653 Simulator Connection** and click **Next**.

5. Select **Custom Simulator** and **Browse...** to the following **boot.txt**:
   C:\WindRiver\vxworks653-2.2\target\proj\ simpc_default\ integration\boot.txt

6. Set the **Processor Number** to **2**, and click **Next**.

7. Set the **Backend** to **wdbpipe** (if not already set), and in the **Advanced target server options** box, click on **Edit....**
   This will open the **Edit Target Server Options** pane.

8. In the **Common** tab, make sure that **Enable target server file system** is selected, and set the **Root:** to **C:\temp\tsfs**.

9. Enable **Make Target Server File System writable** if not already selected.
   These target server options are configured to allow the System Viewer tool to upload data collected on the target to the host.

10. In the **Symbols** tab, select **Load global and local symbols.**
    The **Logging** tab enables output, backend, and WTX logging.

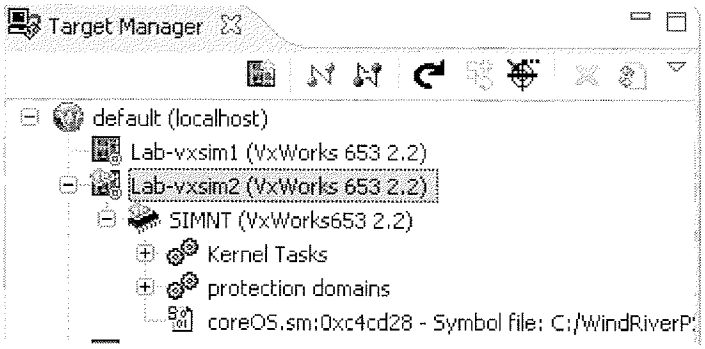
11. Click on **OK** to save the changes, and then click on **Next**.

12. Review the mappings in the **Object Path Mappings** pane, make sure that the **Load module symbols to debug server automatically if possible** is checked, and click on **Next**.

13. Accept the defaults in the **Target State Refresh** and **Default breakpoint Options** pane by clicking on **Next**.

14. In the **Connection Summary** pane, set the name to **Lab- vxsim2**.

15. Make sure that **Immediately connect to target if possible** option is enabled, and click on **Finish**.
    In the **Target Manager** pane, the following is displayed**:**



The **Finish** button in the **New Connection** wizard launches the simulator session, starts the target server, and connects the target server to the Simulator automatically.
Any of the available tools from the Workbench IDE can now be started; the only difference now is that the target server has been customized.

This concludes the lab. Do not continue.

System Concepts Lab

© 2008 Wind River Systems, Inc.

WIND RIVER

System Concepts Lab

Objective: *In this lab, you will become familiar with the reference build process to build a complete system.*

Note: This lab should take approximately 30 minutes.

Lab Overview

As part of the standard installation, the reference process provides working examples on the configuration and build process in VxWorks 653.
It can be used as a means to learn about the VxWorks 653 build process or as a starting point for creating your own build system for your projects.

The reference system you will build in this lab contains one partition and one application. This lab contains the configuration steps necessary for building a VxWorks ARINC 653 system: a partition OS for the application to run on; and a core OS to manage the module and schedule a single partition application to run.

There are four parts to be build:

1. *module OS build;*
2. *partition OS build;*
3. *application build;*
4. *integration build.*

1. Determine the target system (either the simulator or a real hardware target) you wish to build and identify its appropriate BSP files in **C:\WindRiver\vxworks653-2.2\target\config**.

   For example, the BSP would be:
   Simulator: simpc
   Target: wrSbc750gx

2. In the BSP directory for your chosen target, open the default XML BSP file, which is named ***BSP_Name*_default.xml** (for example, **wrSbc750gx_default.xml**).

3. Note the value of the attribute **VirtualAddress** of the **SharedLibraryDescription** element. For example,

   simpc: `0x50000000`
   wrSbc750gx: `0x10000000`

   This value is the default virtual address for the partition OS for your target.

4. Open the file **C:\WindRiver\vxworks653-2.2\target\reference\helloWorld\introduction\hello-full\pos\hello-pos.xml**

5. Right click on the file, select **Properties**, and deselect the **Read-only** Attribute. Click **OK**.

6. Locate the attribute **VirtualAddress** of the **SharedLibraryDescription** element. Its current value is `$(SSLADDR)`.

7. Replace this value with the partition OS virtual address for your particular BSP. Save the file, and close it.

8. Open the **Makefile** in the BSP directory and find the value of your **CPU** variable. For example,

   simpc: `SIMNT`
   wrSbc750gx: `PPC604`

9. In the BSP directory for your chosen BSP, open the BSP XML description file, which is named ***BSP_Name*.xml** (for example, **wrSbc750gx.xml**), containing the CoreOSDescription.

10. Note the value of the attribute **partitionVirtualAddress** of the **KernelConfiguration** element. For example,

    simpc: `0x28000000`
    wrSbc750gx: `0x40000000`

    This value is the default partition virtual address, `PARTADDR`, for your target.

11. Open the VxWorks 653 Development Shell by selecting **Start → All Programs → Wind River → VxWorks 653 2.2 for Power PC → VxWorks 653 2.2 Development Shell**.

12. Change your current directory to **C:\WindRiver\vxworks653-2.2\target\reference\helloWorld\introduction\hello-full**.

13. Run **make**, specifying the values for ***CPU***, ***BSP*** and ***PARTADDR***:

    Simulator:
    → `make BSP=simpc CPU=SIMNT PARTADDR=0x28000000`

    wrSbc750gx Target:
    → `make BSP=wrSbc750gx CPU=PPC604 PARTADDR=0x40000000`

The **Makefile** will run the makefiles for each part of the build.

This will create a network-loadable system image (**boot.txt**) in the directory **C:\WindRiver\vxworks653-2.2\target\reference\helloWorld\introduction\hello-full\int\demo**.

14. Reboot your target with this system image, and describe what happens. What is the partition schedule?

Exercise: System Reconfiguration

Modify the system that you just built to print "bye" every 2 seconds.

Hints:

a. Utilize **sysClkRateGet** to determine your system clock rate.
b. Use the native VxWorks API **taskDelay**
c. Include the vthreads library in the Shared Library API XML description, **pos-api.xml**

```
<Interface>
      <Version Name="hello"/>
            <xi:include
href="$(WIND_BASE)/target/vThreads/config/comps/xml/vThreads.xml"/>
```

This concludes the lab. Do not continue

Configuration and Build Lab

Network Loadable System Image

© 2008 Wind River Systems, Inc.

WIND RIVER

Configuration and Build Lab (Network Loadable System Image)

Objective: In this lab, you will *build a preconfigured VxWorks ARINC 653 system.*

Note: This lab should take approximately 30 minutes.

Lab Overview

The system you will build in this lab contains one partition and one application. This lab contains the configuration steps necessary for building a VxWorks ARINC 653 system: a partition OS for the application to run on; and a core OS to manage the module and schedule a single partition application to run.

There are four parts to be build:

1. *module OS build;*
2. *partition OS build;*
3. *application build;*
4. *integration build.*

Module OS Configuration

In a VxWorks ARINC 653 system, the module OS is provided by the VXWorks 653 core OS.
The module OS for this single partition example consists of the default core OS configuration provided by VxWorks 653 and the default set of core OS components.
We will use the default **CoreOSDescription** document from the following BSP:

Simulator: simpc

Target: wrSbc750gx

The module OS contains only a **Makefile**.
You will manually edit the **Makefile** using Workbench.

1. Copy the lab sources to the appropriate directory:
   a. ***Simulator***: Copy the **C:\VxWorks653_Labs\Configuration_Building_Labs\simulator\SinglePartition_full** folder to **C:\myLabs\Configuration_Building.**

   b. ***Hardware Target***: Copy the **C:\VxWorks653_Labs\Configuration_Building_Labs\wrSbc750gx\SinglePartition_full** folder to **C:\myLabs\Configuration_Building**.

2. If not already started, open Workbench by clicking **Start → All Programs → Wind River → Workbench 2.6 → Wind River Workbench 2.6.**

3. From Workbench, open the following file by clicking **File → Open File…**: **C:\myLabs\Configuration_Building\SinglePartition_full\mos\Makefile**. The **Makefile** contains the following lines:

```
WIND_BASE := $(subst \,/,$(WIND_BASE))
PRJEXE := `which prj.sh`
```

The first line is a definition for **WIND_BASE** that makes the converts \ characters. The second line creates the basic project structure.

The following includes the appropriate version of **Makefile.vars** in your **Makefile**:

```
include $(WIND_BASE)/target/config/make/Makefile.vars
```

The following creates a core OS project:

```
      prjCreate -domtype kernel -prj SinglePartition/ -
bsp $(BSP) -name coreOS
```

Notice the separator.

`prjcreate`	Creates a coreOS project.
`–domtype`	Specifies that we are creating a kernel project
`-prj`	Specifies the location in which the core OS project is created
`-bsp`	Specifies the BSP to build for. (This value will be supplied on the command line).
`-name`	Specifies the name of the coreOS.

The following copies the correct **CoreOSDescription** document for your target into the project directory:

```
      cp ./$(BSP).xml SinglePartition/bsp.xml
```

The following generates the linker script for the coreOS project:

```
      xmlgen --ldScript --arch $(TOOLARCH) -o
SinglePartition/coreOS.lds SinglePartition/bsp.xml
```

The following adds the **targetTools.ddf** and **windview.ddf** components:

```
	@sh $(PRJEXE) domComponentBundleAdd -p
./SinglePartition
$(WIND_BASE)/target/config/comps/vxWorks/sysTemplates/vxK
ernel/targetTools.ddf
	@sh $(PRJEXE) domComponentBundleAdd -p
./SinglePartition
$(WIND_BASE)/target/config/comps/vxWorks/sysTemplates/vxK
ernel/windview.ddf
```

The following calls the **ADD_NEEDED** target to ensure that all component dependencies in the core OS configuration are accounted for, and that components that are needed are added:

```
	make -C SinglePartition ADD_NEEDED
```

The following builds the MOS:

```
	make -C SinglePartition
```

The **clean** target removes the **SinglePartition** directory and its contents:

```
clean:
	rm -rf $ SinglePartition
```

4. Save the file and close it.

Building the Module OS

1. Open the 'VxWorks Development Shell' by clicking **Start → All Programs → WindRiver → VxWorks 653 2.2 for Power PC → VxWorks 653 2.2 Development Shell.**

2. Change the current directory to **C:\myLabs\Configuration_Building\SinglePartition_full\mos.**

3. Run **make**, specifying the values of the variables ***$(CPU)*** and ***$(BSP)***:

   a. *Hardware Target*:
   → `make BSP=wrSbc750gx CPU=PPC604`

   b. *Simulator*:
   → `make BSP=simpc CPU=SIMNT`

The build files will be placed in a directory named for the build spec used.
This directory will be located under the project directory.
For instance, if you specified the **SIMNT** as the value of the ***$(CPU)*** variable on the command line, the built files will be found in the directory **..mos\SinglePartition\SIMNTgnu.debug**.

Partition OS Configuration

The partition OS configuration is contained in a system shared library.
It includes the following files:

pos.xml	The **SharedLibraryDescription** document that contains configuration information for the library that will form part of the system configuration record.
pos-api.xml	Contains information that is used to build entry-point tables to enable applications to link to the routines in the shared library.
Makefile.pos	Contains the commands required to build the partition OS.
Makefile	Contains rules to establish a project to build the partition OS.

Shared Library Description Document

1. In Workbench, open the **pos.xml** file from **C:\myLabs\Configuration_Building\SinglePartition_full\pos** by clicking **File → Open File...**

Notice the line `SystemSharedLibrary="true"`.
This specifies that the library is a system shared library rather than a regular shared library.

Locate the line `VirtualAddress="$(SSLADDR)"`.
This line specifies the virtual address of the system shared library within the module.

To build the example, you will need to substitute an appropriate value for your module in place of **"$(SSLADDR)"**.

2. Replace the **VirtualAddress** attribute value **$(SSLADDR)** with the following addresses:
   a. ***Hardware Target***:
      0x10000000

   b. ***Simulator***:
      0x50000000

These values can be found in the default BSP Module configuration file (**BSP_default.xml**) located in the BSP directory for your BSP.

3. Save the file and close it.

Partition OS Configuration

1. In Workbench, open the **Makefile.pos** file from **C:\myLabs\Configuration_Building\SinglePartition_full\pos** by clicking **File → Open File..**.

The partition OS **Makefile.pos** contains the commands required to build the partition OS. It contains the following lines:

```
all: pos.sm pos-stubs.o
```

The partition OS build process produces two outputs:

pos.sm	The partition OS system module file
pos-stubs.o	The stub file for the shared library

It includes the appropriate version of **Makefile.vars** into your **Makefile**:

```
include
$(WIND_BASE)/target/vThreads/config/make/Makefile.vars
```

It adds the following to establish the path to the object files supplied with VxWorks 653 that will be built into the system shared library:

```
vpath %.c $(WIND_BASE)/target/vThreads/config/comps/src
vpath %.o $(WIND_BASE)/target/vThreads/lib/obj$(CPU)gnuvx
vpath %.xml ../
```

It also adds the following to establish the shared library object file dependencies:

```
pos.sm: sslMain.o vThreadsComponent.o pos-ept.o pos.lds
```

sslMain.o	Provides initialization for system shared libraries
vThreadsComponent.o	vThreads component file, supplied with VxWorks 653
pos-ept.o	Entry-point table
pos.lds	Linker script

It calls the linker:

```
    $(LD) $(LDFLAGS) -T pos.lds -o $@ $(filter %.o,$^)
```

It ensures that all **.o** rules are implemented:

```
%.o: %.c
	$(CC) $(CFLAGS) -c -o $@ $<
```

It generates the entry point table from information in the Shared Library API document, **pos-api.xml**:

```
pos-ept.c: pos-api.xml
	xmlgen --linkage --output-entrypoints $@ $<
```

It generates the stubs file using the information from the Shared Library API document:

```
pos-stubs.c: pos-api.xml
	xmlgen --linkage --arch $(TOOLARCH) --output-stubs
$@ $<
```

It generates the linker script:

```
pos.lds: pos.xml
	xmlgen --ldScript --arch $(TOOLARCH) -o $@ $<
```

It includes the **clean** target:

```
clean:
	rm -f *.lds
	rm -f *.sm
	rm -f *.o
```

2. Close the file.

Partition OS Project Makefile

1. Open the **Makefile** from **C:\myLabs\Configuration_Building\SinglePartition_full\pos**.

This file will contain the rules to establish a project to build the partition OS. The **all** rule must copy all the required files to the **SinglePartition** directory.

The following creates the partition OS project:

```
mkdir -p SinglePartition
cp Makefile.pos SinglePartition/Makefile
cp pos.xml SinglePartition
cp pos-api.xml SinglePartition
```

The following builds the partition OS by adding a **make** call in the **SinglePartition** directory:

```
make -C SinglePartition
```

The clean rule removes the **SinglePartition** directory:

```
clean:
	rm -rf SinglePartition
```

2. Close the file.

Building the Partition OS

1. In the VxWorks 653 Development Shell change your current directory to **C:\myLabs\Configuration_Building\SinglePartition_full\pos.**

2. Run **make**, specifying the values of the variables *$(CPU)* and *$(BSP)*:

   Simulator:
   → `make BSP=simpc CPU=SIMNT`

   Hardware Target:
   → `make BSP=wrSbc750gx CPU=PPC604`

3. Check that **pos.sm** and **pos-stubs.o** are created in the **pos/SinglePartition** directory.

Application Configuration and Build

Application code for the **SinglePartition** system is contained in the directory **C:\myLabs\Configuration_Building\SinglePartition_full\app.**

It must contain the following files:

testApplication.c	Application source file
Application1.xml	Configuration information for the library that will form part of the system configuration record.
Makefile.app1	Commands needed to build the application
Makefile	Rules to establish a project to build the partition OS.

Application Configuration

1. Open **Makefile.app1** from
   C:\myLabs\Configuration_Building\SinglePartition_full\app.

The application makefile needs to access files from the partition OS build.
The variable **$(POS_DIR)** identifies the location of the partition OS build files:

```
POS_DIR = ../../pos/SinglePartition
```

The path to the vThreads code is established:

```
vpath %.c $(WIND_BASE)/target/vThreads/config/comps/src
```

The following provides the path to the directory where the partition OS files reside and to the .xml files.

```
vpath %.o $(POS_DIR)
vpath %.xml ../
```

The following links the required modules to create the application system module:

```
part1.sm: vxMain.o testApplication.o pos-stubs.o
part1.lds
```

vxMain.o	VxWorks653 application initialization code
testApplication.o	Application object file
pos-stubs.o	Partition OS stub file
part1.lds	Application linker script

The call to the linker to create the application module file:

```
	$(LD) $(LDFLAGS) -T part1.lds -o $@ $(filter %.o,$^)
```

The following ensures that all **.o** rules are implemented:

```
%.o: %.c
	$(CC) $(CFLAGS) -c -o $@ $<
```

The following creates the application linker script using information from the Application Description document:

```
part1.lds: Application1.xml
     xmlgen  --ldScript  --arch  $(TOOLARCH)  --address
$(PARTADDR) -o $@ $<
```

The following adds the clean rule:

```
clean:
     rm -f *.o
     rm -f *.sm
     rm -f *.lds
```

2. Close the file.

Application Configuration Makefile

1. Open the **Makefile** from **C:\myLabs\Configuration_Building_Lab\SinglePartition_full\app**. This file will contain rules to establish a project to build the application.

The **all** rule must copy all the required files to the **SinglePartition** directory.

The following creates the application project:

```
     mkdir -p SinglePartition
     cp Makefile.app1 SinglePartition/Makefile
     cp Application1.xml SinglePartition
     cp testApplication.c SinglePartition
```

The following builds the application by calling **make** in the **SinglePartition** directory:

```
     make -C SinglePartition
```

The following **clean** rule removes the **SinglePartition** directory:

```
clean:
     rm -rf SinglePartition
```

2. Close the file.

Building the Application

1. Open the **Configuration_Building\ SinglePartition_full\mos*bsp*.xml**

2. Determine the value of the attribute **partitionVirtualAddress** (`PARTADDR`) in the element **CoreOSDescription → Kernel Configuration**.

   For example,
   Simulator:
   `partitionVirtualAddress=0x28000000`

   Hardware Target:
   `partitionVirtualAddress=0x40000000`

3. From the VxWorks Development Shell, change your current directory to **C:\myLabs\Configuration_Building\SinglePartition_full \app.**

4. Run **make**, specifying the values for **CPU**, **BSP** and **PARTADDR**.
   Simulator:
   `→make BSP=simpc CPU=SIMNT PARTADDR=0x28000000`

   Hardware Target:
   `→make BSP=wrSbc750gx CPU=PPC604 PARTADDR=0x40000000`

5. Check that the application system module, **part1.sm** is created in **C:\myLabs\Configuration_Building_Labs\SinglePartition_full \app\SinglePartition.**

Integration Build

The integration build is the final step in creating a VxWorks 653 module.
The integration build creates the system configuration record and the system image which you load onto the target and execute.
The integration build files are located in the directory:
C:\myLabs\Configuration_Building_Labs\SinglePartition_full\int.

They include:

Module_SinglePartition.xml
: Contains configuration information for the module that will form part of the system configuration record.

Partition1.xml
: Describes the configuration of the partition in which the application will reside.

Makefile.int
: Contains the commands required to combine the various elements of the module together to form a payload that can be transferred to a target and executed.
It also builds the system configuration record.

Makefile
: Creates the project directory, copies files from the various sources to the project directory and runs the integration **Makefile**.

Integration Configuration

1. Open **Makefile.int** from **C:\myLabs\Configuration_Building\SinglePartition_full\int**.

The following calls XMLGen to create a list of system module files and a list of symbol files:

```
SM_FILES = $(shell $(XMLGEN_FILES) $(XML_FILE))
SYM_FILES = $(shell $(XMLGEN_FILES) --sym $(XML_FILE))
```

It adds the variables that will point to the location of the module OS, partition OS and application build files:

```
MOS_DIR = ../../mos/SinglePartition
POS_DIR = ../../pos/SinglePartition
APP_DIR = ../../app/SinglePartition
```

It includes the **Makefile.rules**:

```
include $(WIND_BASE)/target/config/make/Makefile.rules
```

It copies the system module files for the core OS, system shared library and application to the current directory:

```
coreOS.sm: $(MOS_DIR)/$(CPU)gnu.debug/coreOS.sm; cp $< $@
pos.sm: $(POS_DIR)/pos.sm ; cp $< $@
part1.sm: $(APP_DIR)/part1.sm ; cp $< $@
```

It specifies the creation of symbol tables:

```
net: $(SYM_FILES)
```

It adds the rule to check that the size of the ELF sections of the system module files fits into the black boxes specified in the configuration documents:

```
checkSize: $(SM_FILES)
     xmlgen --bbCheck $(addprefix -j ,$(basename $^))
configRecord.xml
```

It finally adds the clean rule:

```
clean:
	rm -f *.sm
	rm -f *.sym
	rm -f *.xml
	rm -f *.o
	rm -f boot.txt
	rm -f configRecord.reloc
```

2. Close the file.

Project Integration Configuration Makefile

1. Open the **Makefile** in **C:\myLabs\Configuration_Building\SinglePartition_full\int**.

The project directory is created and files copied from the various sources to the project directory:

```
	mkdir -p SinglePartition
	cp Makefile.int SinglePartition/Makefile
	cp Module_SinglePartition.xml SinglePartition
	cp Partition1.xml SinglePartition
	cp ../mos/SinglePartition/bsp.xml SinglePartition
	cp ../pos/SinglePartition/pos.xml SinglePartition
	cp ../app/SinglePartition/Application1.xml
SinglePartition
```

The following builds the integration:

```
	make -C SinglePartition
```

It adds the following **clean** rule:

```
clean:
	rm -rf SinglePartition
```

2. Close the file.

Building the System Image

1. In the VxWorks Development Shell change the working directory to **C:\myLabs\Configuration_Building\SinglePartition_full\int.**

2. Run **make**, specifying the values for ***CPU***, ***BSP*** and ***PARTADDR***: ***Simulator***:
   → `make BSP=simpc CPU=SIMNT PARTADDR=0x28000000`

Hardware Target:
→ `make BSP=wrSbc750gx CPU=PPC604 PARTADDR=0x40000000`

The network loadable system image (**boot.txt**) for the **SinglePartition** system is created in **C:\myLabs\Configuration_Building\SinglePartition_full\int\SinglePartition**.

3. Reboot your target with this system image, and describe what happens.

Module Build

1. Open the **Makefile** in **C:\myLabs\Configuration_Building_Labs\SinglePartition_full**

The build rules are added:

```
build:
	make -C mos
	make -C pos
	make -C app
	make -C int
```

The **clean** rule is added that will remove all built components:

```
clean:
	make -C mos clean
	make -C pos clean
	make -C app clean
	make -C int clean
```

Finally, the export rule is added that will export all module components into the **export** directory.

```
export:
	cp ./mos/SinglePartition/$(CPU)gnu.debug/coreOS.sm
../exports
	cp -f ./mos/SinglePartition/bsp.xml ../exports
	cp -f ./pos/SinglePartition/pos.sm ../exports
	cp -f ./pos/SinglePartition/pos.xml ../exports
	cp -f ./app/SinglePartition/part1.sm ../exports
	cp -f ./app/SinglePartition/Application1.xml
../exports
$(EXPORT_DIR):
	mkdir $(EXPORT_DIR)
```

2. In the VxWorks Development Shell, change the working directory to **C:\myLabs\Configuration_Building\SinglePartition_full**

3. If using a real hardware target, shut down the FTP server.

4. Run **make**, specifying the values for ***CPU***, ***BSP*** and ***PARTADDR*** and ***TOOL***.

   Simulator:
   ```
   → make BSP=simpc CPU=SIMNT PARTADDR=0x28000000 TOOL=gnu
   clean all
   ```

   Hardware Target:
   ```
   → make BSP=wrSbc750gx CPU=PPC604 PARTADDR=0x40000000
   TOOL=gnu clean all
   ```

5. If using a real hardware target, restart the FTP server.

6. Reboot the target with this system image. Is it the same system ?

Exercise: Application Configuration and Build Lab

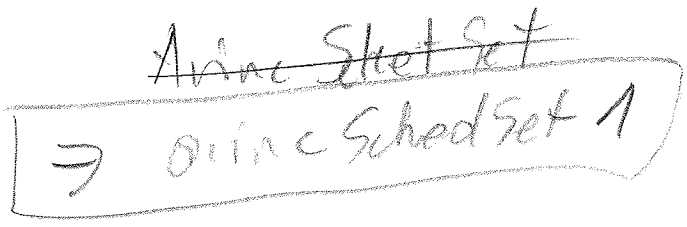
Objective: *In this lab, you will reinforce your system concepts from an application developer's perspective, and use the configuration and build process to verify your understanding.*

Note: This lab should take approximately 30 minutes.

---

1. For the **SinglePartition_full** system that you just built, what is the system schedule?
2. Change the system schedule so that only the Module OS is scheduled to run.
3. Modify **testApplication.c** so that the application partition prints "Hello" every 2 seconds.
4. Rebuild and reboot your target. Explain what happens? Does the application run as expected?
5. Create another schedule, such that the Module OS has a minor frame of 2 seconds, and the application partition has a minor frame of 1 second.
6. Rebuild and reboot your target. Explain what happens. Does the application partition run at every 2 second interval?
7. Use **arincSchedSet** to change the schedule. Now what happens? How often does the application partition print "Hello"? Explain.

This concludes the lab. Do not continue.



Configuration and Build Lab
RAM Payload System Image

© 2008 Wind River Systems, Inc.

WIND RIVER

Configuration and Build Lab
(RAM Payload System Image)

Objective: In this lab, you will *build a RAM Payload System Image.*

Note: This lab should take approximately 40 minutes.

Lab Overview

The RAM Payload System Image system is identical to the Network Loadable System Image except for:

payloadObjs_ram.o
The boot loader for a RAM payload system image.

boot.txt
Not used anymore to load the system image. It is still required to enable the target server to connect to the target for the debugging purposes.

CoreOSDescription Document
A RAM payload region must be allocated in the configuration of the core OS. This requires that you edit the CoreOSDescription document for your platform.

Boot loader
Include a RAM payload boot loader in the RAM payload.

Note: This lab can be done only for **wrSbc750gx**.

You can do this lab only after you have completed the **Build_Configuration_NET** lab.

Module Rebuild

1. Copy the **C:\VxWorks653_Labs\Configuration_Building_labs\wrSbc750gx\SinglePartition_ram** folder to **C:\myLabs\Configuration_Building**.

2. Open the project **Makefile** in **C:\myLabs\Configuration_Building\SinglePartition_full\mos**, and include **payloadObjs**. Insert the following lines on the **all** rule:

```
ifneq ($(CPU),SIMNT)
      make -C SinglePartition payloadObjs
endif
```

3. Save the file and close it.

4. Open the **Makefile** in **C:\myLabs\Configuration_Building\SinglePartition_full**, and copy the **payloadObjs** generated by the mos build into the export directory by adding the following to **export** rule (before the **$(EXPORT_DIR):**) :

```
ifneq ($(CPU),SIMNT)
      cp
./mos/SinglePartition/$(CPU)gnu.debug/payloadObjs_ram.o
../exports
      cp
./mos/SinglePartition/$(CPU)gnu.debug/payloadObjs_rom.o
../exports
endif
```

5. Save the file and close it.

6. Open **wrSbc750gx.xml** in **C:\myLabs\Configuration_Building\SinglePartition_full\mos**.

7. Check that the values for the **Size** and **Base_Address** attributes of the **ramPayloadRegion** element are as follows:

```
ramPayloadRegion Size="0x01000000"
Base_Address="0x0f000000"
```

8. Save the file if you performed any modifications and close it.

   You will now rebuild and export the **SinglePartition** module.

9. Open the VxWorks Development Shell and change your current directory to **C:\myLabs\Configuration_Building\SinglePartition_full**.

10. If using a hardware target, make sure that the FTP server is shut down.

11. Run **make**, specifying the values for ***CPU***, ***BSP*** and ***PARTADDR*** and ***TOOL***.

    Simulator:
    ```
    → make BSP=simpc CPU=SIMNT PARTADDR=0x28000000 TOOL=gnu
    clean all
    ```

    Hardware Target:
    ```
    → make BSP=wrSbc750gx CPU=PPC604 PARTADDR=0x40000000
    TOOL=gnu clean all
    ```

Integration Configuration

1. Open **Makefile.int** in **C:\myLabs\Configuration_Building\SinglePartition_ram\int**, and add the ram target:

```
all: ram checkSize
```

2. Create a list of binary files that will be created by the "ram" target:

```
BIN_FILES = $(shell $(XMLGEN_FILES) --bin $(XML_FILE))
```

3. Add the variables that will point to the location of the module OS, partition OS and application build files:

```
MOS_DIR = ../../../exports
POS_DIR = ../../../exports
APP_DIR = ../../../exports
```

4. Copy the payload loader for the RAM payload system image from the **exports** directory:

```
payloadObjs_ram.o:   ../../../exports/payloadObjs_ram.o   ;
cp $< $@
```

5. Copy the system module files for the core OS, system shared library, and application to the current directory:

```
coreOS.sm: $(MOS_DIR)/coreOS.sm ; cp $< $@
pos.sm: $(POS_DIR)/pos.sm ; cp $< $@
part1.sm: $(APP_DIR)/part1.sm ; cp $< $@
mos.sm: coreOs.sm
```

6. Pad out the binary files that constitute the RAM payload so that they align on the boundaries specified in their respective black boxes:

```
BINFLAGS_EXTRA = $(shell $(XMLGEN_BINFLAGS) -j $*
configRecord.xml)
$(BIN_FILES): configRecord.xml
```

7. Save the file and close it.

Project Configuration

1. Open the **Makefile** in **C:\myLabs\Configuration_Building\SinglePartition_ram\int**.

2. Create the project directory and copy files from the various sources to the project directory:

```
mkdir -p SinglePartition
cp Makefile.int SinglePartition/Makefile
cp Module_SinglePartition.xml SinglePartition
cp Partition1.xml SinglePartition
cp ../../exports/bsp.xml SinglePartition
cp ../../exports/pos.xml SinglePartition
cp ../../exports/Application1.xml SinglePartition
```

3. Add the line that builds the integration:

```
make -C SinglePartition
```

4. Add the **clean** rule:

```
clean:
	rm -rf SinglePartition
```

5. Save the file and exit.

Building the System Image

1. Open the VxWorks Development Shell, and change the working directory to **C:\myLabs\Configuration_Building\SinglePartition_ram.**

2. Run **make**, specifying the values for ***BSP***, ***CPU*** and ***RAM_ADDR***:
   → `make BSP=wrSbc750gx CPU=PPC604 RAM_ADDR=0x0f000000`

The RAM Payload system image for **SinglePartition** is created in **C:\myLabs\Configuration_Building\SinglePartition_ram\int\SinglePartition**.

Booting the RAM Payload System Image

Verify that the RAM Payload System Image works by rebooting your target with the newly created RAM Payload System.

1. Change the 'file name' parameter of the bootApp to load **sms_ramPayload** instead of **boot.txt**, start the FTP server, and reboot.

What do you see ?

This concludes the lab. Do not continue.

© 2008 Wind River Systems, Inc.

Debugging Lab

© 2008 Wind River Systems, Inc.

WIND RIVER

Debugging Lab

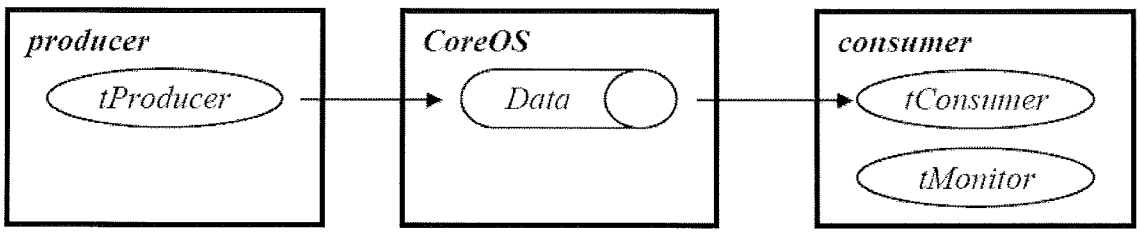
Objective: In this lab, you will *learn how to use the basic debugger features to debug a multi-partition, multi-process application.*

Note: This lab should take approximately 90 minutes.

Lab Overview

The system we will debug in the first part of the lab consists of two partitions, consisting of three applications:

- *"tProducer", running in "firstAppPart", periodically writes some data (integers) into a structure Data which is sent to a queuing port.*
- *The consumer process "tConsumer", running in "secondAppPart", reads in the structure from the queuing port, processes the data and then waits for the next structure to arrive on the queuing port.*
- *The monitor process "tMonitor", also running in "secondAppPart", checks the sign of the data received by "tConsumer" and reports any changes of the sign of the data to the console.*



Building and launching the application

1. Copy the lab sources to **C:\myLabs\Debugging** directory:
   a) If using the simulator, copy the **C:\VxWorks653_Labs\Debug_Labs\simulator** folder contents to **C:\myLabs\Debugging**

   b) If using a real hardware target, copy the contents of the **C:\VxWorks653_Labs\Debug_Labs\wrSbc750gx** folder to **C:\myLabs\Debugging**.

2. Start the VxWorks Development Shell from the **Start** menu:

   Start → All Programs → Wind River → VxWorks 653 2.2 for PowerPC → VxWorks 653 2.2 Development Shell.

3. In the VxWorks 653 2.2 Development Shell, change to the **C:\mylabs\Debugging** directory:

   → `cd C:\mylabs\Debugging`

4. Browse the contents of this directory.
   In particular, examine the contents of **xml\Lab.xml**, and **Makefile**.

5. Build the system:

   Simulator:
   → `make BSP=simpc CPU=SIMNT PARTADDR=0x28000000`

   Hardware Target:
   → `make BSP=wrSbc750gx CPU=PPC604 PARTADDR=0x40000000`

This will build the module OS, system shared library, configuration record and the application partitions.

If using the simulator, the final binaries are created in **C:\mylabs\Debugging\ simpc_Lab\integration.**

If using a hardware target, the final binaries are created in **C:\mylabs\Debugging\ wrSbc750gx_Lab \integration.**

6. When the build completes, start the target to run this newly created system:

   a) ***If using the simulator***:
   Kill any simulators that may still be running.
   In the Workbench IDE, change the session settings of the simulator in the **Target Manager**.
   Right click on **Lab-vxsim1** and select **Properties**.
   Check the Custom simulator radio button and specify the **boot.txt** file that we just build: **C:\mylabs\Debugging\simpc_Lab\integration\boot.txt**
   In the **Target Manager**, right click **Lab-vxsim1** and select **Connect** to start the simulator.

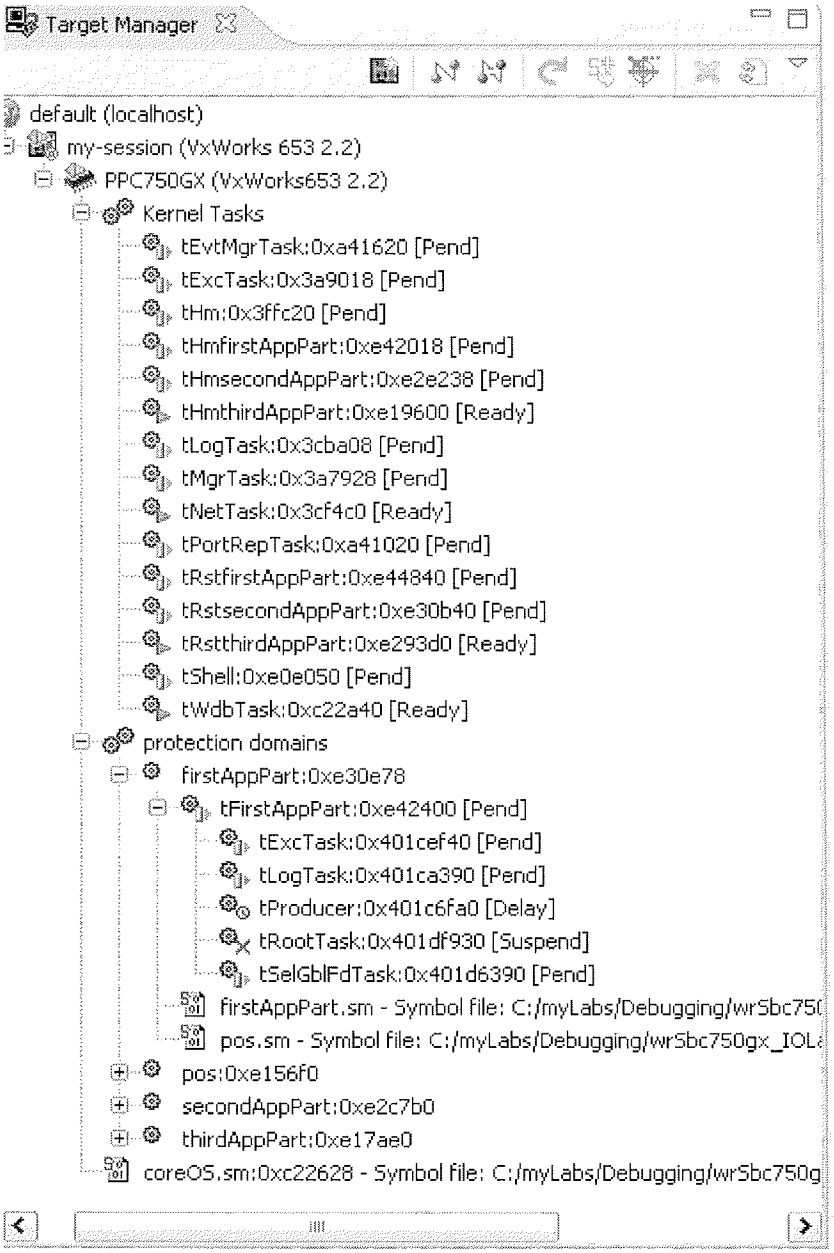
© 2008 Wind River Systems, Inc.

b) ***If using a hardware target***:
Start the FTP server by clicking **Start→All Programs→Wind River→VxWorks 653 2.2 for PowerPC→FTP Server**.
Apply power to the target, and in the Workbench **Terminal** window interrupt the boot.
Type ***c*** at the boot prompt to change the boot parameter "file name" to the location of our system **boot.txt** file which we just built, for example **C:\mylabs\Debugging\wrSbc750gx_Lab\integration\boot.txt.**
Reboot the target.

7. Open the **Device Debug** Perspective by selecting from Workbench **Window → Open Perspective → Device Debug.**

8. If not done already, setup and connect a target server.

   After you have connected to the target, in the IDE, the **Target Manager** window will show the target connected, and displays the tasks and partitions on the target.



Note: The image above is for a real hardware target. If using a simulator, the image will be slightly different.

A "refresh" may be necessary to illustrate all of the elements by clicking on the **Refresh View or Selection** icon in the **Target Manager** window. This window shows the module **coreOS.sm** containing the Module OS code, and a list of Module OS (kernel) tasks that have been spawned at boot time. Also shown are the partitions **firstAppPart**, **secondAppPart, thirdAppPart**, the system shared library **pos**, and their associated static modules (**.sm**). Associated with each partition is a task which runs all the processes (threads) of the partition.

9. In the target console (i.e. Terminal view), you should get regularly printed messages

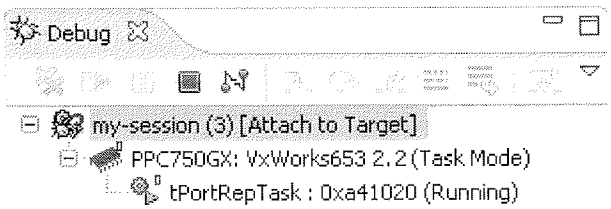
giving the state of **intBeat**.

Attaching a Kernel Task (Task mode)

Workbench allows debugging an existing kernel task or spawning and debugging a new kernel task.

1. In order to debug an existing task, you must attach the debugger to the running task. To do this, expand the **Kernel Tasks** item in the **Target Manager** and select **tPortRepTask** (the port reporting task for port monitoring).

2. Select **Target → Attach to Kernel Task (task mode)** from the main menu. The debugger attaches to the task and the task is displayed in the **Debug** view.



The image above is for a real hardware target. If using a simulator the image will be slightly different.

3. Select **tPortRepTask** in the Debug view.

4. Click the **Suspend** button on the **Debug** view toolbar. The task is stopped. Its stack trace is displayed in the **Debug** view, and the **Assembly** view opens showing the current execution point.

5. Click the **Resume** button on the **Debug** view toolbar.

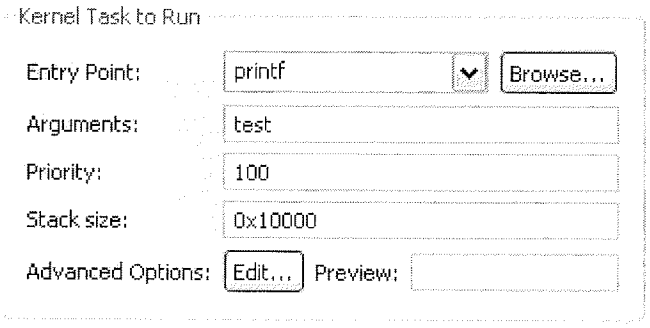
6. Click the **Disconnect** button on the **Debug** view toolbar. The debugger connection to the task is terminated.

7. Click the **Remove All Terminated** button in the **Debug** view. This clears the **Debug** view.

Spawning and Attaching a Kernel Task (task mode)

To spawn and debug a kernel task:

1. In the **Target Manager**, select **Kernel Tasks**.
2. Select **Target → Debug → Debug Kernel Task...**. The **Debug** dialog is displayed.
3. On the **Main** tab, enter "printf" in the **Entry Point** field and "test" in the **Arguments:** field.

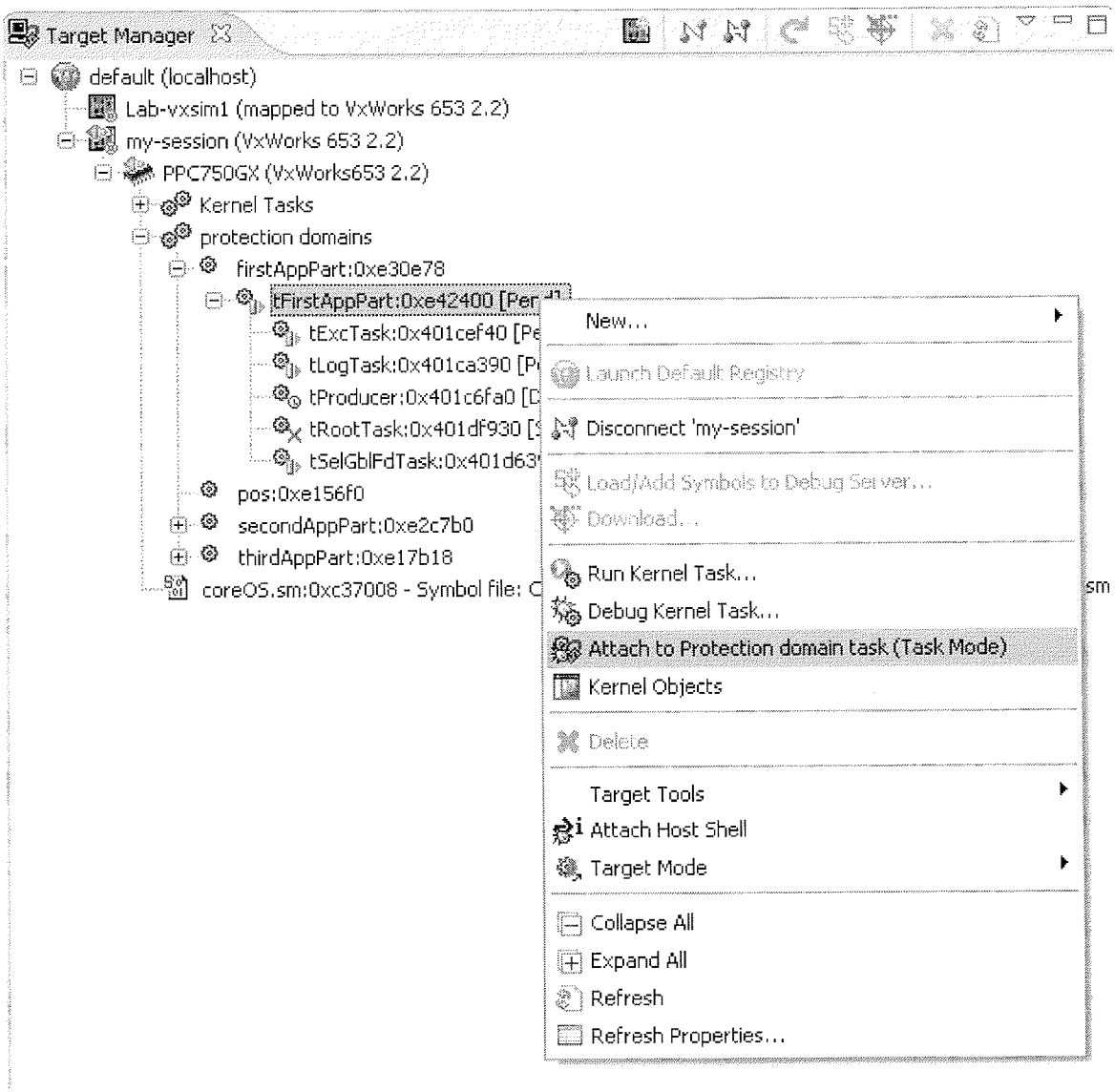


4. In the **Debug Options** tab, make sure that **Break on Entry** and **Automatically attach spawned Kernel Tasks** are checked.
5. Click **Debug**. The task is added to the **Debug** view and the assembly code is shown in the **Assembly** view.
6. Use the step controls on the **Debug** view toolbar to step through the code. Observe that the recently executed lines are shaded in progressively lighter tones as you step.
7. Click the **Resume** button on the **Debug** view toolbar. The string ***test*** is printed in the simulator console and the task exits. The debugger connection to the task is terminated.
8. Click the **Remove All Terminated** button in the Debug view. This clears the **Debug** view.

© *2008* Wind River Systems, Inc.

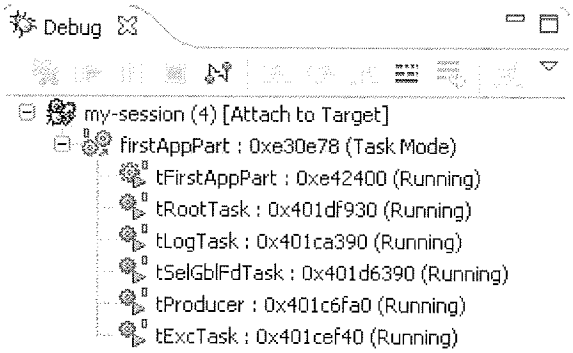
Attaching to the Partition

1. In the **Target Manager** window, expand the **firstAppPart** protection domain to view the running tasks (processes). You will see listed first the Partition OS task **tFirstAppPart** followed by the threads executing, as seen here.



2. Attach to **tFirstAppPart** by selecting the **Attach to Protection domain task (Task Mode)**, as shown above.

The debug window of the IDE will then display the list of threads executing as threads of the **tFirstAppPart** task.
The **tFirstAppPart** task is shown at the same level as the threads running within the task. It is the first item in the list and is distinguished by a different icon.



© *2008* Wind River Systems, Inc.

Setting a Breakpoint from Source

You can now open the source code file for this task and associate it with the debug session. Depending on how your project is configured, Workbench may be able to associate the source code files with the project automatically.

The procedure below assures that the association is made.

1. Open the source file **producer.c** in the Sources directory of your lab, using **File → Open File...**

   Navigate to **C:\mylabs\Debugging\wrSbc750gx_Lab\firstApp\producer.c** if using a real target, or to **C:\mylabs\Debugging\simpc_Lab\firstApp\producer.c** if using the simulator.

   It will be displayed in the source window of the IDE.

2. Select **firstAppPart** from **Debug** window and then highlight the first instruction of the **SineAllocate** function from the **producer.c** file.

   Right click in the gutter next to that line and select **Breakpoints → Add Breakpoint...** The **Line Breakpoint Properties** panel is displayed.

3. In the **Source Lookup** tab click the **Edit Source Lookup...** button. **The Edit Source Lookup Dialog** is displayed.

4. Click on the **Add...** button and select **File System Directory** and click **OK**. The **Directory Selection** dialog is displayed.

5. Select **C:\mylabs\Debugging\wrSbc750gx_Lab\firstApp** if using a real hardware target or **C:\mylabs\Debugging\simpc_Lab\firstApp** if using the simulator, and click **OK.**

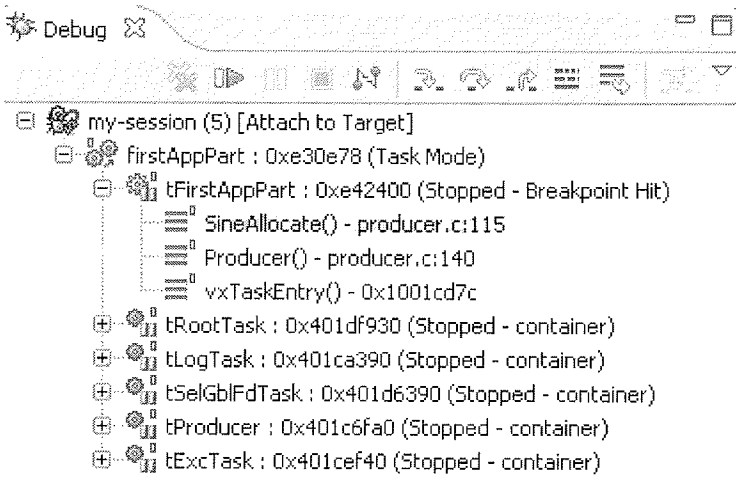
6. Click OK to close the **Edit Source Lookup Path** dialog.

7. Click OK to close the **Line Breakpoint Properties** dialog. The breakpoint is set.

8. In the **Breakpoints** window you will see the breakpoint listed with a checkmark indicating it is enabled. Since this function is being called as the application executes the breakpoint should be encountered shortly after you set it.

© *2008* Wind River Systems, Inc.

9. The current location points to the source line where the breakpoint was set; the **Debug** Window displays the current stack, and the console display has stopped printing, indicating that the producer is no longer running.



Try looking at **producer.c** displayed in Assembly mode by clicking the **Toggle Assembly Stepping Mode** button in the **Debug** window tool bar.

You will see each source line displayed with its corresponding assembly language. You can toggle this mode on and off as desired. Experiment with the step controls.

10. When you have finished experimenting, remove the breakpoint by selecting **Run → Remove All Breakpoints**, and clicking **Yes.**

11. Resume the thread by clicking the **Resume** button on the **Debug** view toolbar. Close the **producer.c** file in the **Editor** view.

De-activate the **Toogle Assemply Stepping Mode** clicking on the button from the **Debug** toolbar.

Setting an Expression Breakpoint

1. Ensure that **tFirstAppPart** is selected in the **Debug** view.

2. Select **Run → Breakpoints → Add Expression Breakpoint...**. The **Expression Breakpoint Properties** dialog is displayed.

3. In the **Location Expression** field of the **General** tab, enter the function name "SineAllocate"

4. Select the **Scope** tab. Ensure that **Enable restricting of breakpoint scope** is checked and that **Enable scoping to debug targets and threads** is not checked.

5. Ensure the **Show only active launch configurations** is checked. The list of launch configurations will be restricted to those active in the debugger.

6. Expand the **firstAppPart** partition and ensure that the **tFirstAppPart** partition task is selected.

7. Click **OK**. The breakpoint is set.

8. Wait for the system to hit the breakpoint. This should happen momentarily.

9. When the breakpoint is hit, Workbench will open the source file and display the line where the breakpoint was hit. (If Workbench cannot locate the file, it will prompt you to specify its location.)

10. Remove the breakpoint, resume the partition, and close the source file.

Using the Symbol Browser

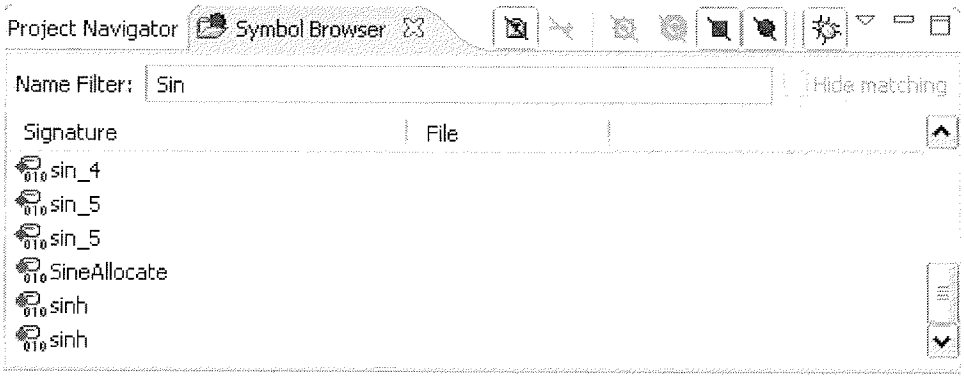
You can use the symbol browser to navigate the symbols in the tasks that are attached to the debugger.

1. Locate the **Symbol Browser** view. If it is hidden behind another view, click its tab to bring it forward. If it is not visible, select it from **Window → Show View**.

2. Click the **Show the symbols loaded by the debugger** button on the **Symbol Browser** tool bar. (It is the one at the right with a bug on it.) The view is populated with a long list of symbols.

3. In the **Name Filter**: field of the **Symbol Browser** toolbar, enter **Sin.** The list of symbols is restricted to those whose names start with the string entered.

© 2008 Wind River Systems, Inc.



4. Double-click on the **SineAllocate** symbol. Workbench opens the source file in the **Edit** view and displays the function definition for **SineAllocate**.

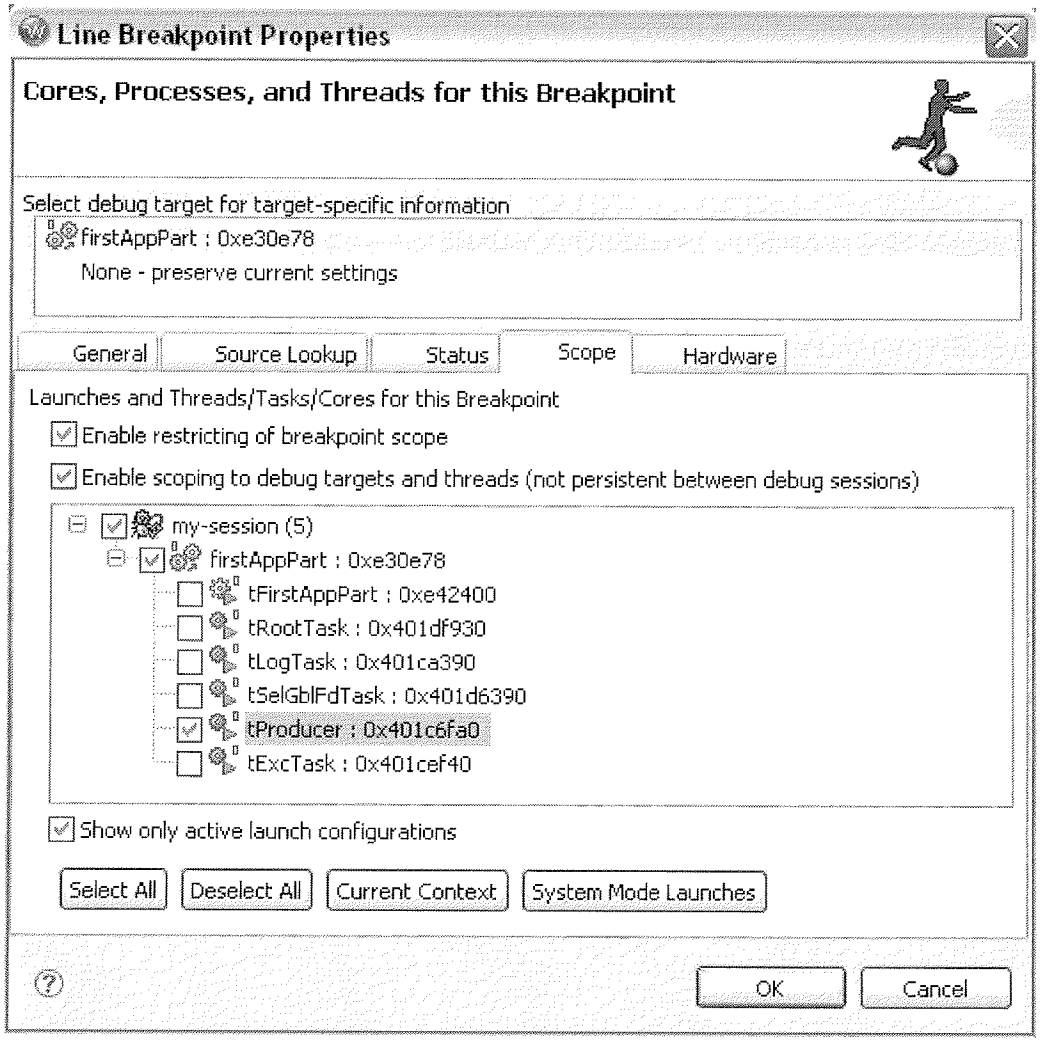
Setting a Breakpoint in a Thread

While it is not possible to step through a thread, it is possible to set a breakpoint in a thread. (This means that the breakpoint will be hit only when the code is executing in the thread specified, not when that code is executed by another thread.)

1. Locate the ***SineAllocate*** function in the source code file.
2. Locate the first executable line of the function.
3. Right click in the gutter next to that line (not on the line itself, but in the gutter) and select **Breakpoints → Add Breakpoint....**
4. Select the **Source Lookup** tab and confirm that the source lookup settings are correct.
5. Select the **Scope** tab.
6. Check **Enable scoping to debug targets and threads**. This allows you to specify which threads the breakpoint applies to.

© *2008* Wind River Systems, Inc.

7. Check thread **tProducer** and clear all the other threads.



8. Click **OK**. The breakpoint is set.

9. Wait for the system to hit the breakpoint. This should happen momentarily. You cannot use the step controls because stepping through a thread is not supported.

10. Remove all breakpoints by right clicking and selecting **Remove All**.

11. Click the **Resume** button on the **Debug** view toolbar.

12. Click the **Disconnect** button on the **Debug** view toolbar. The debugger connection to the task is terminated.

13. Click the **Remove All Terminated** button on the **Debug** view toolbar. This clears the **Debug** view.

Debugging a Partition in System Mode

1. In the **Target Manager**, select **SIMNT** if using a simulator, or **PPC750** if using a real target, under the connection name

2. Select **Target → Attach to Kernel (System Mode)**. Click Yes in the launching dialog. The **Debug** view will show the system.

3. In the **Target Manager**, select the **tFirstAppPart** task.

4. Select **Target → Attach to Protection domain task (System Mode)**.
   If the menu shows **Attach to Protection domain task (TaskMode)**, select **Target → Target Mode → System**, then select **Target → Attach to Protection domain task (System Mode)**.
   The **Debug** view will show the partition.

5. In the **Debug** view, highlight the system debug scope.

6. You can now set a breakpoint in the partition. Select **Run → Breakpoints → Add Expression Breakpoint**.
   The **Expression Breakpoint Properties** dialog is displayed.

7. In the **Location Expression** field of the **General** tab, enter the expression **SineAllocate**. The information panel of the dialog will show the location of the symbol.

8. On the **Scope** tab of the dialog, ensure that **Enable restricting of breakpoint scope** is checked.

9. Ensure that **tFirstAppPart** is part of the selected scope (the scope cannot be limited to **tFirstAppPart** alone).

10. Click **OK**. The breakpoint is set.

11. Wait for the breakpoint to be hit. This should happen momentarily. The **Debug** view will show that the system is stopped and will display a stack trace.

12. Set another breakpoint

13. Resume the task. The task breaks at the next breakpoint. In system mode you can only resume the system. You cannot step through **tFirstAppPart**. Experiment with resuming and viewing variables.

14. When you are finished, resume the system and disconnect the debugger.

This concludes the lab. Do not continue.

© 2008 Wind River Systems, Inc.

Visualization Lab

© 2008 Wind River Systems, Inc.

WIND RIVER

System Viewer Lab

Objective: In this lab, you will *collect, examine and interpret System Viewer event logs.*

Note: This lab should take approximately 45 minutes.

Lab Overview

The debugging lab includes a third partition that contains the application that we will use System Viewer to visualize.

All we need to do is change the system schedule to the second predefined schedule that allocates time only to partition three.

Configuring System Viewer

1. ***If not built already***, build the Debugging Application:

   a. Copy the lab sources to **C:\myLabs\Debugging** directory:
      1. If using the simulator, copy the **C:\VxWorks653_Labs\Debug_Labs\simulator** folder contents to **C:\myLabs\Debugging**
      2. If using a real hardware target, copy the contents of the **C:\VxWorks653_Labs\Debug_Labs\wrSbc750gx** folder to **C:\myLabs\Debugging**.

   b. Start the VxWorks Development Shell from the **Start** menu:
      Start → All Programs → Wind River → VxWorks 653 2.2 for PowerPC → VxWorks 653 2.2 Development Shell.

   c. From the VxWorks 653 2.2 Development Shell, change to the **C:\mylabs\Debugging** directory:
      → `cd C:\mylabs\Debugging`

   d. Build the system:
      Simulator:
      → `make BSP=simpc CPU=SIMNT PARTADDR=0x28000000`
      Hardware Target:
      → `make BSP=wrSbc750gx CPU=PPC604 PARTADDR=0x40000000`

      If using the simulator, the final binaries are created in **C:\mylabs\Debugging\ simpc_Lab\integration.**
      If using a hardware target, the final binaries are created in **C:\mylabs\Debugging\ wrSbc750gx_Lab \integration.**

© 2008 Wind River Systems, Inc.

2. If any breakpoints are set, clear them.

3. Reboot the target, and restart the target server connection.

4. Change the schedule, from the target shell or host shell:
   → `arincSchedSet 1`

5. In the **Target Manager** window, right click on the target server connection name and click **Refresh**.

6. Expand the **tThirdAppPart** task (**protection domains → thirdAppPart → tThirdAppPart**).
   You should see the **tRead1**, **tRead2**, and **tPeriod** tasks in the task list for the third partition.

7. Start **System Viewer Configuration** by right clicking the target connection in the target manager window and selecting **Target Tools → System Viewer Configuration**.
   The **System Viewer Configuration** menu appears.

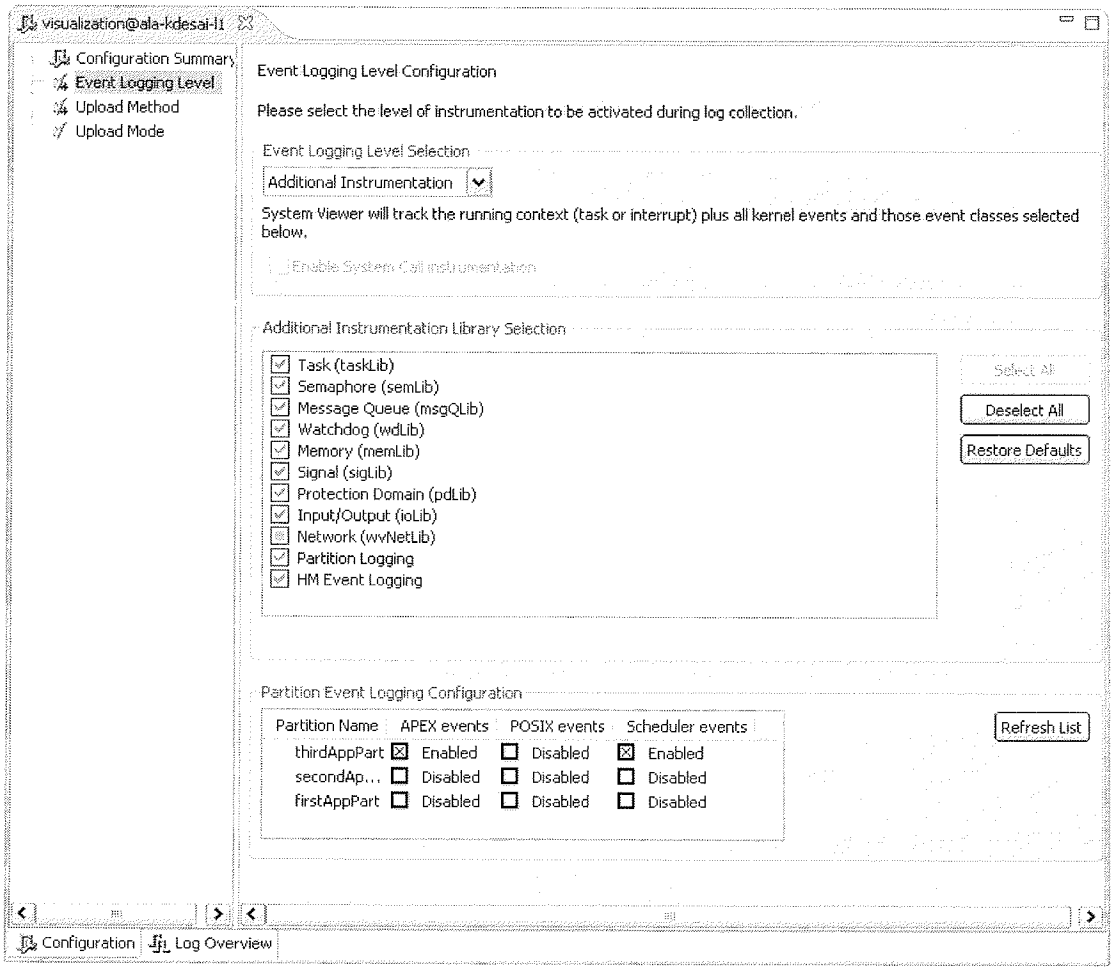
8. In the **Configuration** window, select **Event Logging Level**.
   This opens the **Event Logging Level Configuration** pane.

9. Select **Additional Instrumentation** logging level from the drop-down list and make sure that all possible libraries are checked.

10. Click on the **Partition Logging** checkbox.
    The **Partition Event Logging Configuration** group box will be displayed at the bottom of the panel.

11. Enable the **APEX** and **Scheduler** events only for the third partition by clicking the checkboxes as shown below:

© *2008* Wind River Systems, Inc.



This will give the maximum amount of logged event information and the most interesting display (albeit at the expense of relatively larger overhead and rapidly growing, sometimes cluttered event logs.)

12. Configure the upload method and mode by selecting **Upload Method** in the left pane.

13. Specify **Socket via TSFS** from the drop down list.
    If the **Upload Method** gives an error, replace the name of **Host:** with the IP address `10.10.10.10` in the **Socket via TSFS configuration**.

14. Click **Apply**.

15. Select **Upload Mode** in the left pane, and choose **Deferred Upload** from the "Upload Mode Selection" drop down.

16. Click on the **Advanced** >> button to see the buffer configuration panel.

17. Set the "Max. Buffer Count:" to `20`, and change the "Buffer Size:" to `256 KB`.

18. Click **OK** to close this panel, and complete the System Viewer configuration.

Collecting and Uploading a Short Event log

1. With the system still running, click the **Start System Viewer logging** icon on the System Viewer Tool bar located on the main tool bar.

2. Check the **Refresh every 1 second** box to get updates about buffer usage during data collection (Since we are collecting all possible events, the buffer will fill very quickly, and the data collection will stop).

3. Reclick the **Start System Viewer logging** icon click to start a new log. If desired, click on the **Stop System Viewer logging** icon.

4. Click the **Upload Event Log** icon.

System Viewer opens a log window and sends the necessary commands to the target to start the upload process.
When the upload is finished, **System Viewer** parses the uploaded log and displays it in the graph Window.

You are ready to start analysis of the log.

Examining an Event Log

1. Select the **Event Graph** tab, if not already selected.

2. Click on the "Zoom 100%" icon, on the System Viewer toolbar (of the main WindView IDE window).

This expands the scale of the graph to show the whole event log. (Be aware that System Viewer will draw all events in the log at this scale, and redrawing operations when the System Viewer graph is resized or uncovered may take some time for large event logs.)

3. The task and interrupt contexts in the log are listed on the left of the graph: the container tree.
   At the top are the interrupt levels in numerical order.
   Then, for each domain, the tasks are listed in priority order with the Module OS first.
   At the bottom is the **Idle** state: the state when no tasks or interrupts are executing.
   Some events which occur in interrupts are sometimes shown in this state.

You should see a fairly narrow cluster of events involving the tasks which run in the Module OS and in **thirdAppPart**.

4. To focus on our application running in **thirdAppPart**, close the domain folder containing the Core OS in the container tree (left hand side of the event graph), **firstAppPart**, and **secondAppPart**.

5. Zoom in on a time of interest that contains activity. You may do this by dragging the mouse across the events of interest, while pressing the `[Shift]` key at the same

time.

6. Alternatively, after dragging the mouse across the events, right-click and select "Zoom In" from the context menu or click the "Zoom in" icon on the System Viewer toolbar.

7. You should now see more clearly the activity in **thirdAppPart**. The tasks of particular interest are **tPeriod**, **tRead1**, and **tRead2**.

8. Notice that when these tasks are running, the **tThirdAppPart** task of the partition is also running.

9. You should see purple events marked **fd** in **tThirdAppPart**. These events are the I/O **write** which correspond to the **printf** done in the processes. The **write** is handled by **tThirdAppPart**, which accesses the I/O device directly. Note that if the device were busy, the entire partition would block. No other module OS tasks are involved in the write process. How can one prevent the entire partition from being blocked?

10. Play with the zoom buttons and with the cursor to navigate through the log.

11. Move the mouse over the **apexEventSet** event in the context of the **tPeriod** process.

A description of the event appears in the status bar at the bottom of the graph window.

12. Double-click on the event, and the **Event Search** dialog appears.

13. In the **Event Search** dialog, click the **Lock** check boxes appropriately so that the **Container** is locked, but the **Event** and **Object** fields are not.

14. Then click the **Next** button.

This goes to the closest next event in the **tPeriod** context.
You can navigate through all the events of the log which occur in the **tPeriod** process.

15. Choose the **tThirdAppPart** process directly in the **Event Search** dialog box's Container pull down menu and navigate through the calls done by this task. Note how the task state stipple can change for example after a **semTake**.

16. Right click on this state stipple line and select **Context State** from the context menu that appears. This gives you the state of the task.

You can also use the **Legend** from the System Viewer **Help** menu.

17. Now click on the **Event Table** tab on the top of the event graph. In the container tree you can select one or more process (using the CTRL key) and list the events which occur in these processes. Try the tasks in **tThirdAppPart**.

© *2008* Wind River Systems, Inc.

18. Now, experiment with the use of System Viewer as a tool to understand what happens in your system. Try to explain how the **TIMED_WAIT** (**apexTimedWait**) call in **tRead2** is informed that the delay has expired.

Make use of the two user events (10 and 20) that have been inserted in the process code. You may need to zoom out to see these events.

© *2008* Wind River Systems, Inc.

Monitoring Tools Lab

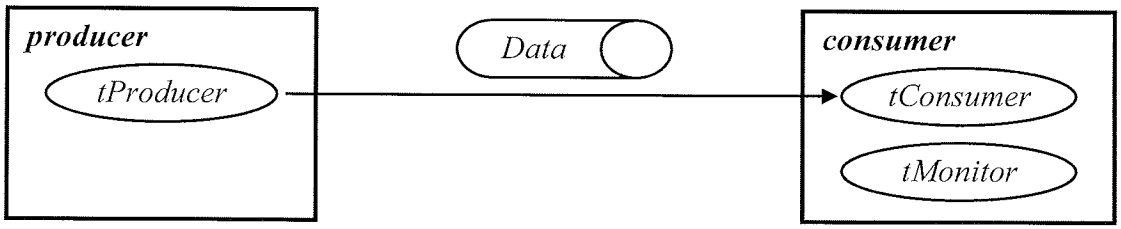
Objective: *In this lab you will learn how to use the monitoring tools to debug a multi-partition, multi-process applications.*

Building the Application

1. Reboot your target, and reconnect your target server.

The two partition system containing the producer/consumer/monitor processes runs, and the target console prints the state of **intBeat**:

- "tProducer", running in "firstPartition", periodically writes some data (integers) into a structure **Data** which is sent to a queuing port.
- The consumer process "tConsumer", running in "secondPartition", reads in the structure from the queuing port, processes the data and then waits for the next structure to arrive on the queuing port.
- The monitor process "tMonitor", checks the sign of the data received by "tConsumer" and reports any changes of the sign of the data to the console.



Time Monitoring

1. Start a host shell, and edit its viewing size: Right click on the menu bar of the Shell and select **Properties**.

2. In the Layout tab, change the Window Size Width setting to `132`. Click **OK**, and then **OK** again.

3. From the shell, begin CPU monitoring:
   → `timeMonitorStop`

   → `timeMonitorStart`

4. Display CPU usage of the APEX calls in the first partition
   → `:firstAppPart`
   → `timeMonitorAPEXShow`

5. Get all the CPU usage in the first partition
   → `timeMonitorShow`

   Which partition uses most of its CPU ? Why?

Memory Usage Monitoring

From the host shell:

→ `memoryUsageHeapShow`

→ `memoryUsageStackShow`

→ `memoryUsageHmShow`

→ `memoryUsagePortShow`

→ `memoryUsageAllShow`

Port Monitoring

1. From the Host Shell, and (re)-enable system monitoring
   → `portMonitorDisable`

2. Setup the channel to monitor. For example, to monitor the send port write and the destination port read of channel `1`
   → `portMonitorStart 1, 0x5`

3. Start the "Target Console" (Virtual I/O) by right clicking on the target server connection, and selecting **Target Tools → Target Console**.

4. Enable the **Set the VIO channel number** box, and set it to `99`.

5. Keep the other default settings, and click **OK**.
   The target console appears.

6. From the host shell, enable the port monitoring,
   → `portMonitorEnable 1,10`

   What Happens ?

This concludes the lab. Do not continue.

© 2008 Wind River Systems, Inc.

ARINC 653 API Lab

© 2008 Wind River Systems, Inc.

WIND RIVER

ARINC 653 API Lab

Objective: In this lab, you will *learn how process scheduling, process synchronization, and process health monitoring works inside a partition; how to manage exclusive access to shared resources, and understand the process communication within a partition and between partitions.*

Note: This lab should take approximately 150 minutes.

Lab Overview

The lab contains several pre-configured systems, each with 1, 2 or 3 partitions. Each section of the lab concentrates on specific API uses for the configuration and control of system and application resources. Each partition is structured to take you through the steps in building an ARINC 653 multi partition system build. To help automate the build process for the lab much of the configuration effort has already been completed in the lab software section. These can be reviewed and inspected in both the ***Makefile*** *and the* ***Lab.xml*** *file.*

1. Copy the lab sources to the **C:\myLabs\API** folder.

Simulator: Copy the **C:\VxWorks653_Labs\API_Labs\simulator** folder content to **C:\myLabs\API**.

Hardware Target: Copy the **C:\VxWorks653_Labs\API_Labs\wrSbc750gx** folder contents to **C:\myLabs\API**.

2. Open the VxWorks Development Shell by clicking **Start → All Programs → WindRiver → VxWorks 653 2.2 for Power PC → VxWorks 653 2.2 Development Shell.**

© 2008 Wind River Systems, Inc.

Process Scheduling

Objective: To understand ARINC 653 process scheduling inside a partition.

1. From the VxWorks Development Shell, change to the **C:\mylabs\API\Process_Sched** directory.

   ```
   → cd C:\mylabs\API\Process_Sched
   ```

2. Browse the contents of this directory. In particular, examine the contents of **xml\Lab.xml**, and **Makefile**.

3. Build the system by typing '**make**'.

   Simulator:
   ```
   → make BSP=simpc CPU=SIMNT PARTADDR=0x28000000
   ```

   Hardware target:
   ```
   → make BSP=wrSbc750gx CPU=PPC604 PARTADDR=0x40000000
   ```

This will build the module OS, system shared library, configuration record and the application partitions.

If you are using the simulator the final binaries are created in the directory **C:\myLabs\API\Process_Sched\simpc_Process_Sched\integration.**

If you are using the hardware target the final binaries are created in the directory **C:\myLabs\API\Process_Sched\wrSbc750gx__Process_Sched\integration.**

4. When the build completes, start the target to run this newly created system.

   a. ***If using the simulator***, kill any simulators that may still be running.

      In the Workbench IDE, change the session settings of the simulator in the **Target Manager**.

      Right click on **Lab-vxsim1** and select **Properties**.

      In the dialogue box, check the Custom simulator and browse for the boot file: **C:\myLabs\API\Process_Sched\simpc_Process_Sched\integration\boot.txt**

      Click **OK**.

      In the **Target Manager**, right click **Lab-xvsim1** and select **Connect** to start the simulator.

© 2008 Wind River Systems, Inc.

b. ***If using a hardware target***, start the FTP server, power on the target, and in the **Terminal** window of the Workbench IDE, interrupt the boot.

   Type ***c*** at the boot prompt to change the boot parameter "file name" to the location of our system **boot.txt** file which we just built, for example **C:\myLabs\API\Process_Sched\wrSbc750gx_Process_Sched\integration\boot.txt.**

   Reboot the target.

5. Your system should now run. In each partition, the periodic process increments a counter each time a period occurs.

6. This counter is printed in the console.

7. The default case is the following one:

Partition	Part1	Part2
Partition Window	2s	1s
Process period	8s	8s
Process Time capacity	8s	8s

8. Explain what you see in the console.

9. Now change the configuration of your system to the following one. You must modify **period.c** to change the values in the process table to the new period and time capacity.

Partition	Part1	Part2
Partition Window	2s	1s
Process period	200ms	200ms
Process Time capacity	200ms	200ms

10. If using the hardware target, shut down the FTP server.

11. Rebuild the project by typing the following commands:

    Simulator:
    ```
    → make BSP=simpc CPU=SIMNT PARTADDR=0x28000000 clean all
    ```

    Hardware target:
    ```
    → make BSP=wrSbc750gx CPU=PPC604 PARTADDR=0x40000000
    clean all
    ```

12. Reboot the target.

13. What are the differences with the previous case?

Since VxWorks 653 is ARINC653, Part 1, Supplement 2 compliant, the beginning of a period of a periodic process is synchronized with the beginning of the partition window.

But as you can see, the Health Monitor (use the **hmLogEntriesShow** command for "firstPartition" and "secondPartition") enforces the deadline action for each of the processes.

14. From the host shell, verify the HM logs
    → **hmLogEntriesShow 1**
    ...
    → **hmLogEntriesShow 2**

Since the process period is 200 milliseconds and the partition durations are 2 seconds and 1 second respectively, the processes are both suspended after the first partition window due to the fact they were to become active outside of their partition window.

15. During development you can disable the Health Monitor so that the system will continue to run despite the timing violation.

To disable the default health monitor actions, but to allow logging, remove the error action associated with the hm event, **HME _DEFAULT** for the partition HM.

16. In "Lab.xml" remove the entry "hmDefaultHandler" in the partition hm table, "part1Hm"

```
<ErrorIDAction
     ErrorIdentifier="HME_DEFAULT"
     ErrorAction="hmDefaultHandler"/>
```

    so that it looks like

```
<ErrorIDAction
     ErrorIdentifier="HME_DEFAULT"
     ErrorAction=""/>
```

This allows the errors to be logged but no action will be taken during debugging.

17. Rebuild, and reboot your target. Explain what you see. How many HM logs do you have now ?

Exercise: Background Process Configuration

1. Remove "secondPartition" from the system.
2. Set the period and deadline of the periodic process back to the default values of case 1.
3. Add a second, aperiodic process.
4. This aperiodic process will suspend itself after doing a **TIMED_WAIT** of 10 seconds. (It should be resumed by the periodic process at every two occurrences of the periodic process).

© 2008 Wind River Systems, Inc.

5. Now set a value to the deadline of this aperiodic process. What value must you give to this deadline?
6. What call must you add in your aperiodic process to have this deadline work?
7. For the partition HM, reset the **HME_DEFAULT** handler to its original setting, "`hmDefaultHandler`".
8. Rebuild and reboot your target. Explain what you see.

Background Processes

1. Remove **secondPartition** from the system: In **Lab.xml** set the **Duration** attribute in the <**PartitionWindow**> element for **secondPartition** in the <**Schedule**> element to `0`.

2. We are going to rebuild the system using the source file **background.c** instead of **period.c**. Rename **period.c** to **period.c.old**, and **background.c** to **period.c.** This will create one periodic process and one background process, which never blocks.

3. Reset the initialization routine in **Makefile.firstApp** and **Makefile.secondApp**. Set the value of the **USER_APPL_INIT** to **InitBackground**.

4. Rebuild the system, by typing the following commands in the VxWorks Development Shell:

   Simulator:
   → `make BSP=simpc CPU=SIMNT PARTADDR=0x28000000 clean all`

   Hardware target:
   → `make BSP=wrSbc750gx CPU=PPC604 PARTADDR=0x40000000 clean all`

5. Reboot the target. What happens?

6. Remember that the Health Monitor can affect how the system executes. Find the bug in your system and solve it.

7. Now create a third process in **background.c**. This process should be a second background process of the same priority of the existing process, **tBackground**.

8. Add the appropriate calls to create and start the processes to ensure that the two processes will be able to get the CPU.

Process Synchronization

Objective: To understand ARINC 653 process synchronization inside a partition.
To compare the use of events and semaphores for synchronization.

Use of events

1. Change to the **C:\myLabs\API\Sync** directory in the VxWorks development Shell.

   → `cd C:\myLabs\API\Sync`

Sources\synchro_event.c contains the code of three processes and their creation.

The first process is periodic and will be used to synchronize the execution of the two other, aperiodic, processes.

An event is also created that is used for starting the two aperiodic processes at each period of the periodic process.

2. Build the system by typing te following command in the VxWorks Development Shell:

   Simulator:
   → `make BSP=simpc CPU=SIMNT PARTADDR=0x28000000`

   Hardware target:
   → `make BSP=wrSbc750gx CPU=PPC604 PARTADDR=0x40000000`

This will build the module OS, system shared library, configuration record and the application partitions.

If you are using the simulator, the final binaries are created in the directory **C:\myLabs\API\Sync\simpc_Sync\integration.**

If you are using a hardware target, the final binaries are created in the directory **C:\myLabs\API\Sync\wrSbc750gx_Sync\integration**.

5. When the build completes, change the boot parameters accordingly, and reboot the target to run this newly created system.

Where is the best place to call **RESET_EVENT**?

Compare the behavior of the processes if you use this call in the periodic process or if you use it in the aperiodic process.

Is there a difference between the two solutions?

© 2008 Wind River Systems, Inc.

Is there a way to synchronize one process at a time (at each period only one periodic process is unblocked)?

Use of semaphores

Instead of events, semaphores can be used to synchronize processes.

The second partition contains the processes using semaphores for synchronization. **Sources\synchro_semaphore.c** contains the synchronization code using a single semaphore.

The two aperiodic processes will wait on the same semaphore.

The semaphore is signaled once at each period.

So, one aperiodic process will be awakened at each period as the semaphore is not a broadcasting mechanism like an event.

How many schedules have been defined? Why?

1. From the Workbench **Target Manager** view, connect to the target, start a Host Shell, and change the schedule so that only partition 2 runs.

   → `arincSchedSet 1`

When you create the semaphore, you need to choose 3 parameters: the initial value, the maximum value, and the queuing discipline.
What is the maximum value?

For our synchronization case what initial value must be set?

If you want both processes to be awakened successively and fairly, do you chose **FIFO** or **PRIORITY** for the queuing discipline?

2. Now suppose you want to awaken more than one process at each period.
   Signal the semaphore twice at each period.

3. Now try to signal the semaphore three times at each period, and then four times.
   What do you see?

Is there any difference if you change the queuing discipline from **FIFO** to **PRIORITY**? Why?

You should still have the maximum value of your semaphore set to 1.

How should this value be changed to solve the problem met when you signaled the semaphore four times in a period?

© 2008 Wind River Systems, Inc.

Managing Process Exclusive Access to Shared Resources

Objective: To understand the ARINC 653 exclusive access mechanism inside a partition. To compare the use of semaphores and the preemption lock mechanisms to obtain exclusive access to a resource.

Use of preemption lock

1. Change to the **C:\myLabs\API\Mutex** directory.
   → `cd C:\myLabs\API\Mutex`

Sources\mutex_preemptionlock.c contains the code of two processes and their creation. The first process is periodic and the second process is aperiodic, of lower priority.

2. Build the system by typing:

   Simulator:
   → `make BSP=simpc CPU=SIMNT PARTADDR=0x28000000`

   Hardware target:
   → `make BSP=wrSbc750gx CPU=PPC604 PARTADDR=0x40000000`

3. When the build completes, change the boot parameters accordingly, and reboot the target to run this newly created system.
   The preemption lock mechanism protects the critical section of code.

Check that you don't have any more race conditions when running the code.

In our example, what can be the consequence of using preemption lock rather than using semaphores to protect the critical section?
How can we see that?

Use of semaphores

1. From the Workbench **Terminal** view, or the Workbench Host Shell, change the schedule so that only partition 2 runs.

   → `arincSchedSet 1`

2. Read the source file **Sources\mutex_semaphore.c**
   Try to find the shared resource used by the periodic process and the background process.

How do you explain the messages which are printed at the console?

© 2008 Wind River Systems, Inc.

What are the semaphore parameters (maximum and initial value) used in its creation?

Notice the "wait" and "signal" semaphore functions used in the code to protect the critical section of code.

What is used as the queuing policy for the semaphore: **FIFO** or **PRIORITY**?

Does this have any influence in our example? In which case is this choice important?

© *2008* Wind River Systems, Inc.

Process Communication within a Partition

Objective: To understand the ARINC 653 communication mechanisms inside a partition.
To compare the use of buffers and blackboards for communication.

Use of blackboards

Use a blackboard for communication between the periodic process and the two other processes.

1. Change to the **C:\myLabs\API\Intrapartition** directory:

   → `cd C:\myLabs\API\Intrapartition`

Sources\intracomm_blackboard.c contains code for three processes.

The highest priority one is periodic, produces data, and sends it to two other, lower priority processes.

What value is given to the parameter "message size" of the blackboard?

2. Build the system by typing:

   Simulator:
   → `make BSP=simpc CPU=SIMNT PARTADDR=0x28000000`

   Hardware target:
   → `make BSP=wrSbc750gx CPU=PPC604 PARTADDR=0x40000000`

3. When the build completes, change the boot parameters accordingly, and reboot the target to run this newly created system.

   What do you see?

The result obtained is going to depend on whether or not you clear the blackboard and where it is cleared.

Use of buffers

1. From the Workbench **Terminal** view, or the Workbench Host Shell, change the schedule so that only partition 2 runs:

   → `arincSchedSet 1`

Read the source file **Sources\intracomm_buffer.c**.

What are the parameters (message size, buffer size) for the buffer?
Try the two queuing disciplines **FIFO** and **PRIORITY** and explain the two different behaviors.

Is the buffer suitable for broadcasting messages to multiple processes?

© 2008 Wind River Systems, Inc.

Process Communication Between Partitions

Objective: To understand ARINC 653 communication mechanisms between partitions.
To compare the use of queuing port and sampling port mechanisms for communication.

Use of queuing ports

1. Change to the **C:\myLabs\API\QPorts** directory.

```
→ cd C:\myLabs\API\QPorts
```

The first partition, "intercomm1", will create a periodic process that will send data on a channel, through a queuing port.

The second and third partitions, "intercomm2", each create two aperiodic processes that wait for data on this channel, through a queuing port, and print a message when they receive data.

2. Build the system by typing:

Simulator:
```
→ make BSP=simpc CPU=SIMNT PARTADDR=0x28000000
```

Hardware target:
```
→ make BSP=wrSbc750gx CPU=PPC604 PARTADDR=0x40000000
```

3. When the build completes, change the boot parameters accordingly, and reboot the target to run this newly created system.

You should see that the messages sent to the port are broadcasted to all partitions but in a partition they are received as for a **FIFO** queue.

Use of sampling ports

We will now repeat the same steps as for the queuing ports but using sampling ports instead.

As reading on a sampling port doesn't block, the **intercomm2_queuing.c** file we used in the previous steps is not suitable.

The file **intercomm2_samplingport.c** will create two periodic processes that will poll the sampling port, rather than two processes which wait for data.

© 2008 Wind River Systems, Inc.

1. Change to the **C:\myLabs\API\SampPorts** directory.
   → `cd C:\myLabs\API\SampPorts`

2. Build the system by typing:

   Simulator:
   → `make BSP=simpc CPU=SIMNT PARTADDR=0x28000000`

   Hardware target:
   → `make BSP=wrSbc750gx CPU=PPC604 PARTADDR=0x40000000`

3. When the build completes, change the boot parameters accordingly, and reboot the target to run this newly created system.

To configure sampling ports, the **protocol** field and the **maxMsgs** field are not required. The new field is the **refresh rate**.

What should the refresh rate be if you want that the messages are valid if they are read in the first occurrence of the partition window which follows the partition window they were produced in?

© 2008 Wind River Systems, Inc.

Process Health Monitoring

Objective: To understand ARINC 653 process health monitoring inside a partition.

1. Change to the **C:\mylabs\API\Process_HM** directory.

   → `cd C:\mylabs\API\Process_HM`

This system consists of two partitions, with each partition creating a periodic process. Each partition will also register a template Process Level Health Management routine.

2. Build the system by typing **make**.

   Simulator:
   → `make BSP=simpc CPU=SIMNT PARTADDR=0x28000000`

   Hardware target:
   → `make BSP=wrSbc750gx CPU=PPC604 PARTADDR=0x40000000`

3. When the build completes, change the boot parameters accordingly, and reboot the target to run this newly created system.

Your system should now run. The default case proposed is the following one:

Partition	Part1	Part2
Partition Window	2s	1s
Process period	8s	8s
Process Time capacity	8s	8s

Explain what you see in the console.

4. Now change the configuration of your system to the following one.

Partition	Part1	Part2
Partition Window	2s	1s
Process period	200ms	200ms
Process Time capacity	200ms	200ms

You must modify **procHm.c** to change the values in the process table to the new period and time capacity.

5. Rebuild the system, by typing the following commands in the VxWorks Development shell:

Simulator:

```
→ make BSP=simpc CPU=SIMNT PARTADDR=0x28000000 clean all
```

Hardware target:

```
→ make BSP=wrSbc750gx CPU=PPC604 PARTADDR=0x40000000
  clean all
```

What are the differences with the previous case?

In the **Process Scheduling** example, the **Partition Level Health Management** was invoked since there was no **Process Level Manager** registered.

This template provides a framework for implementing application specific actions in each partition.

Partition Level I/O

Objective: To perform I/O to a physical device from within a partition.

Note: This lab only works with the wrSbc750gx hardware target. It will NOT work on the simulator.

1. Change to the **C:\mylabs\API\PartitionHwIO** directory.
   → `cd C:\mylabs\API\PartitionHwIO`

2. Browse the contents of this directory. In particular, examine the contents of **Lab.xml**, **wrSbc750gx.xml**, **Sources\periodLed.c**, and the **Makefile**.

3. Build the system by typing:
   → `make BSP=wrSbc750gx CPU=PPC604 PARTADDR=0x40000000`

This builds a single partition system containing a periodic process.

The final binary is created in the directory
C:\mylabs\API\PartitionHwIO\wrSbc750gx_PartitionHwIO\integration.

Note that this configuration is specific to the wrSbc750gx.

4. When the build completes, modify the boot parameters accordingly, and reboot the target to run this newly created system

Your system should now run. In the partition, the periodic process toggles the LED each time a period occurs.

5. Modify the default case and observe.
What happens if multiple partitions mapped and access the LED register?

This concludes the lab. Do not continue.

© 2008 Wind River Systems, Inc.

I/O Partition Lab

© 2008 Wind River Systems, Inc

WIND RIVER

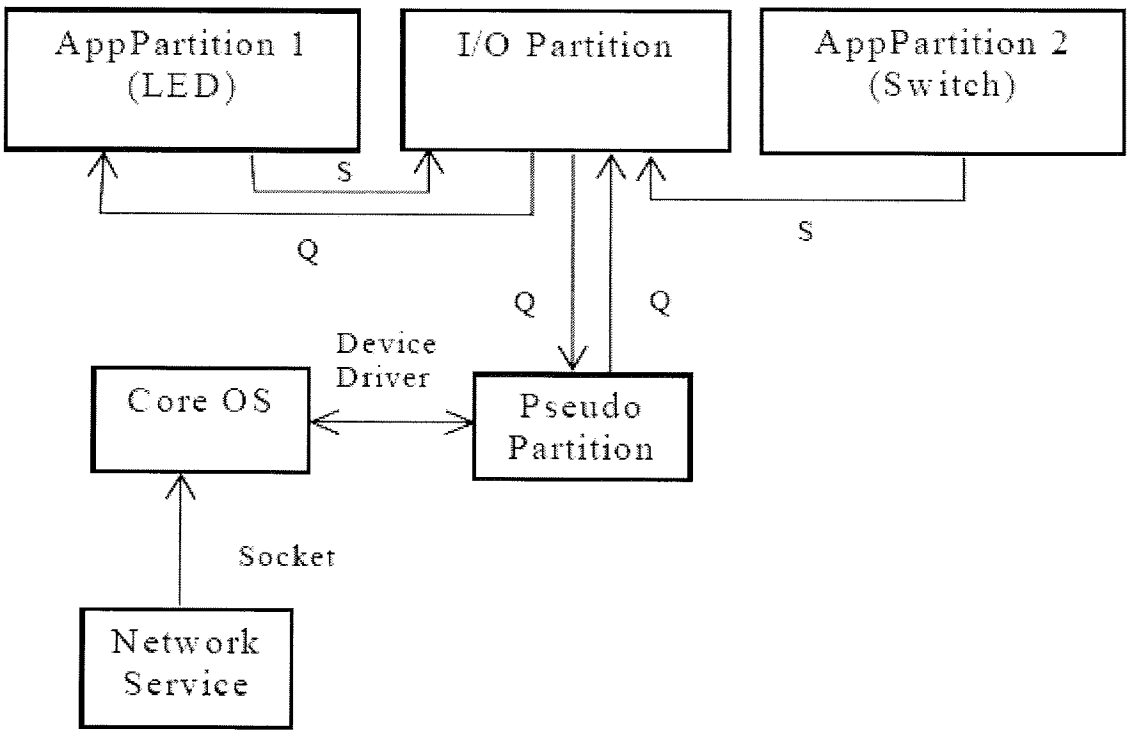
I/O Partition Lab

Objective: In this lab, you will *implement a system containing a basic I/O partition that communicates to a network based control program to accept and send status of hardware controlled LED's and switches.*

Note: This lab should take approximately 100 minutes and can be performed only using a ***hardware target.***

Lab Overview

The lab is based on a 3-partition configuration.



Q and S denote queuing and sampling ports, respectively.

The framework supplied includes incomplete application source, the I/O partition source, along with complete system properties defined in the configuration record and region file.

The system is described in **Table 1**.

Table 1 – System Description

Partition Name	Function
ioPartition	The I/O partition receives requests via an ARINC port connected to network services in the core OS. The messages received are requests for status or state changes for the devices controlled in the other two partitions. The I/O partition then decodes these requests and communicates those requests to the appropriate partition for disposition. Any status changes from the application partitions are communicated back via a separate ARINC port back to the network
AppPartition1	This partition has the register for control of the 4 user LED's mapped to its virtual address space. It receives requests from the I/O partition. Those requests are: 1. Set LED **n** on 2. Clear LED **n** 3. Report LED status The partition process is responsible for acting on the requests and then replying with the action taken.
AppPartition2	This partition has the register for reading the status of the 4 user switches mapped to its virtual address space. The Switch process will periodically sample the state of the switches and when the status changes, update that status to the I/O partition.

The template source files provide comments to assist in correctly implementing each function for the associated partition and connecting it to the correct ARINC port to communicate to/from the I/O partition.

The configuration record supplied defines a number of channels to implement communication between the application partitions and the I/O partition using ARINC ports.

Table 2 describes the I/O channels and their function. This information is used to implement the required ARINC port creation and communications calls to accomplish the data exchange.
The connections between the I/O partition and the core OS network stack have already been provided.

© 2008 Wind River Systems, Inc.

Table 2 – Port Descriptions

ARINC Port Name	Channel Number	Partition End Point	Function
dest_coreos sender_coreos	1 2	Pseudo Partition	These ports handle the communication to/from the core OS for the network (create calls supplied in template). The channels connect to the I/O partition
dest_iopart sender_iopart	2 1	I/O partition	These ports handle the communication to/from the I/O partition for the network (create calls supplied in template). The channels connect to the core OS.
led_setdata_in led_readdata_out	3 4	AppPartition1	These ports receive and send LED data from/to the I/O partition
led_readdata_in led_setdata_out	4 3	I/O Partition	These ports receive and send LED data to/from AppPartition1
sw_data_out	5	AppPartition2	This port send switch status updates to the I/O partition
sw_data_in	5	I/O Partition	This port receives switch status updates from AppPartition2

© 2008 Wind River Systems, Inc.

Building the Application

1. Copy **C:\VxWorks653_Labs\IO_Lab\wrSbc750gx** folder contents to **C:\myLabs\IO**.

2. Change to the **C:\myLabs\IO** directory.

   ```
   →cd C:\myLabs\IO\xml
   ```

3. Browse the contents of this directory.

4. In particular, examine the contents of **IOLab.xml.**

5. Then, browse the **Sources** subdirectory.

6. The address initialization for the shared data regions is supplied as part of the template application source files, **LedApp.c** and **SwrApp.c**.

7. All source files can be found in **C:\myLabs\IO\Sources**.

8. The template source code provides the framework to implement each of the applications as well as the I/O partition communications.

9. The source file **ioPartLab.h** contains the definition of the message format and the descriptions of the control fields for the messages.

10. These are used to implement each of the applications for controlling the LED's and checking the switch status and reporting these back to the I/O partition.

11. Edit the template application source files **LedApp.c** and **SwrApp.c** with the correct ARINC 653 function calls so that the system behaves accordingly. (The locations where code needs to be added is denoted by /* ?? */.)

Look at **C:\mylabs\IOLab\Sources\APPIOMdl.c** for examples of the API usage for sampling and queuing ports.

12. Build the system by typing the following commands in the VxWorks 653 2.2 Development Shell:

    ```
    →cd C:\myLabs\IO

    →make BSP=wrSbc750gx
    ```

13. The final binary is created in **C:\mylabs\IO\wrSbc750gx_IOLab\integration**.

System Verification – I/O Simulator Tool

1. Reboot the target. Do not forget to open the FTP server.

2. Test the system by using the ***Iosimulator*** supplied for use on the Host Platform.

   Run the provided **Iosimulator** tool, **iopltool**, by executing the following command from the DOS command:

   ```
   →cd C:\myLabs\IO\host_tool

   →iopltool 10.10.10.1 5000 0
   ```

10.10.10.1	ip_address_of_target
5000	port number that the target looks for the connection on. It can be changed by changing the value used in the Init routine for the "TxRxProc" component.
0	the mode parameter that tells the tool to use TCP

3. Operation of the tool is self explanatory once you bring it up.
   The result desired is that the selected action takes place and the status is reported back to the host.

   This **Iosimulator** will allow the sending and receiving of messages from the target board.

4. Should **firstAppPart** be Cold or Warm restarted?

5. What affect on communications occurs should the port protocol be changed in one or more of the channels?

6. How does the underlying partition schedule interact with **secondAppPart** and the periodic process checking the switch status?

© 2008 Wind River Systems, Inc.

System Verification – Local Loopback Option

Test the application by building the **TxRxProc_debug.c** with the local loopback option defined.

1. In **TxRxProc_debug.c**, change the line before
   #define LOCAL_LOOPBACK /* define for local target testing */

   from

   ```
   #if 0
   ```

   to

   ```
   #if 1
   ```

2. Shut down the FTP server.

3. Rebuild the system by typing the following commands in The VxWorks 653 2.2 Development Shell:

   ```
   →cd C:\myLabs\IO

   →make BSP=wrSbc750gx remove

   →make BSP=wrSbc750gx
   ```

4. Boot the target with the newly created image. (Don't forget to start the FTP server).

5. When the system is up and running, change the user switches to verify that indeed the local loopback option is working.

   The result desired is to have the changes in switch settings be reflected back in the LEDs.

6. Power off the hardware target and close the FTP server.

© 2008 Wind River Systems, Inc.

Importing and building the IOLab Project into Workbench

1. Clean the project by typing the following commands in the VxWorks Development Shell:

   ```
   →cd C:\myLabs\IO

   →make BSP=wrSbc750gx remove
   ```

2. Build the project by typing the following command:

   ```
   →make BSP=wrSbc750gx wbimport
   ```

3. From the Workbench **File** pull down, select **Import...** then select **General → Existing Projects into Workspace**.

4. Click on **Next**.

5. In the **Import Projects** screen use the **Browse...** option to select the **C:\myLabs\IO\wrSbc750gx_IOLab** directory as the root directory.

6. Click **OK**, and then **Finish**.
   You should see the **IO_Lab** project appearing in the projects window.

7. Once imported, select the **Makefile.wr** file from the **Project Navigator** and double click to edit the file.

8. Uncomment the following line:
   ```
   BSP             = wrSbc750gx
   ```

9. Finally, **select IO_Lab** in the **Project Navigator**, right click and select **Build Project.**

This concludes the lab. Do not continue.

© 2008 Wind River Systems, Inc.

Printed in the UK

VxWorks 653 For Application Developers Lecture Guide

Wind River VxWorks 653 Platform 2.2

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Copyright © Wind River Systems, Inc.

Version 1.0 A, May 2008

All rights reserved. No part of this publication may be reproduced in any form, by photocopy, microfilm, retrieval system, or by any other means now known or hereafter invented without the prior written permission of Wind River Systems, Inc. This document is designed to support the *VxWorks 653 Platform 2.2* course. It is not designed as a stand-alone document, nor is it intended as a substitute for documentation that accompanies Tornado, VxWorks, or Wind River Workbench or any other Wind River Systems, Inc. software or hardware product. For information about Wind River training courses, see:

http://education.windriver.com
Education Services Department
Wind River Systems, Inc.
500 Wind River Way, Alameda, CA 94501
510-748-4100 (phone) 510-749-2454 (fax)
E-mail: training@windriver.com

For training outside of North America, please contact your local sales office.

Wind River®, VxWorks®, Tornado®, Wind®, Wind View®, VxSim® are registered trademarks of Wind River. All other trademarks are trademarks of Wind River Systems, Inc., or are the properties of their respective owners. For further information regarding Wind River trademarks, please see http://www.windriver.com/company/terms/trademark.html.

Slide 2

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Course Prerequisites

- General prerequisites
  - User level familiarity with the host operating system on which the Wind River® VxWorks 653 Platform 2.2 will be installed and utilized
    - Graphical and command-line user interfaces, file systems, and standard editors
  - C programming experience: familiarity with basic control structures, pointers, structures, preprocessor commands
  - Basic understanding of operating systems and debugging techniques
- Preferred prerequisites
  - Basic understanding of ARINC 653 concepts
  - Familiarity with VxWorks
  - Familiarity with **make** and **Makefile**'s

Slide 3

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Objectives

By the end of this course you will be able to:

- Understand the components of ARINC 653
  - Partitions
  - Processes
  - Partition, task, and module OS scheduling
  - Intra-partition communication and synchronization
  - Inter-partition communication using ARINC ports
- Understand how ARINC 653 components are implemented in Wind River® VxWorks 653 Platform 2.2
  - I/O models

Slide 4

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Objectives (Continued)

By the end of this course you will be able to:

Configure applications and partitions
- Partition schedules
- Effective use of ports and health monitoring

Use Workbench to perform cross development and debugging
- Monitoring Tools

Describe the ARINC 653 API

Configure an application for final deployment

Slide 5

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Wind River® VxWorks 653 Platform 2.2 Course

	Chapter
Introduction to ARINC 653	1
Getting Started	2
System Concepts	3
Configuration and Build Partitioning	4
Debugging	5
Visualization and Monitoring	6
ARINC 653 API	7
ARINC 653 API Lab Overview	8
I/O Partition	9

Slide 6

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Introduction to ARINC 653

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Objectives

By the end of this chapter you will be able to understand:

- ARINC Standards
- Module and Partition OS
- Partitioning
- Process management, control and scheduling
- Intra-Partition Communication
- Inter-Partition Communication
- Health Monitoring

Slide 2

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Aeronautical Radio, Incorporated (ARINC)

- Corporation in which US airlines are principal stockholders
  - Other stockholders include a variety of other air transport companies, aircraft manufacturers and non-U.S. airlines
- Sponsors Airlines Electronic Engineering Committee (AEEC)
  - AEEC formulates standards for electronic equipment and systems for airlines
    - Principal function to establish equipment characteristics
  - AEEC is composed of airline technical personnel

Slide 3

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- Other ARINC activities include
  - The operation of an extensive system of domestic and overseas aeronautical land radio stations
  - The fulfillment of systems requirements to accomplish ground and airborne compatibility
  - The allocation and assignment of frequencies to meet those needs
  - The coordination incident to standard airborne communications and electronics systems
  - The exchange of technical information

ARINC Standards

Three categories of ARINC Standards

- ARINC Characteristics
  - Define the avionics equipment's form, fit, function and interfaces
- ARINC Specifications
  - Define physical packaging, or mounting of avionics equipment, or data communications standards, or a high-level computer language
- ARINC Reports
  - Provide guidelines or general information that airlines find to be good practices, often related to avionics maintenance and support

Slide 4

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- ARINC Characteristics are divided into two series
  - The ARINC 700 series for digital avionics
  - The ARINC 500 series for analog avionics
- ARINC Specifications and ARINC Reports related to
  - Digital avionics are included in the ARINC 600 series
  - Older, analog avionics are contained in the ARINC 400 series
- Benefits of ARINC Standards include
  - Promoting competition
  - Providing interchangeability and standardized maintenance procedures
  - A starting point for avionics development
  - Allowing aircraft manufacturers to pre-wire aircraft to ensure that cost-effective avionics and new installations are compatible

© 2008 Wind River Systems, Inc.

Basic Concepts

ARINC 653 - ***ARINC Specification 653, Avionics Application Software Standard Interface, Supplement 2***

- Defines a general-purpose APEX (APplication/EXecutive) interface between the OS of an avionics computer and application software
- Specifies a list of services that allow application software to control the scheduling, communication, and status information of its internal processing elements
- Allows application to be abstracted from underlying OS
- Permits independently developed applications to execute together on the same hardware

Slide 5

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

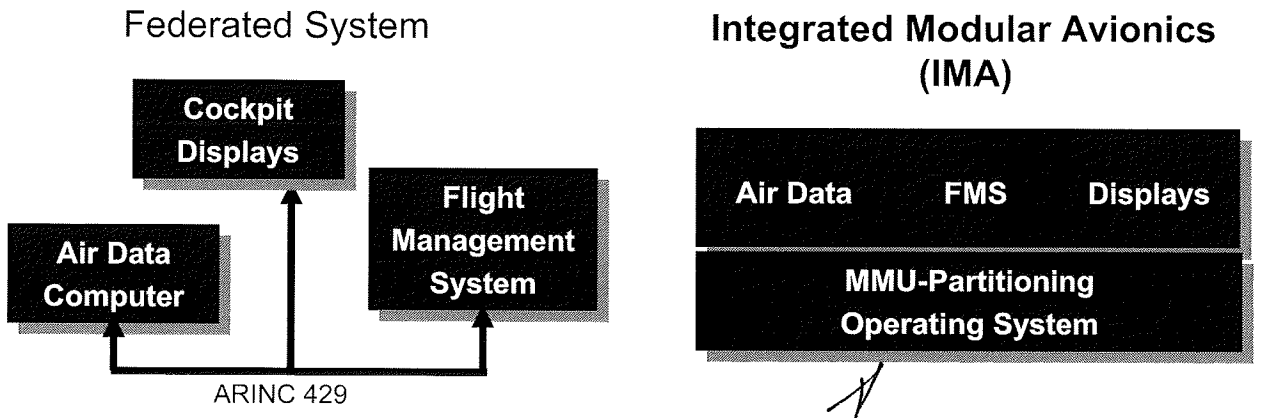
- The ARINC specification is available for purchase from https://www.arinc.com/

3 → Test procedure

VxWorks support 1+2 mit 3

IMA and ARINC 653

- ARINC 653 is a specification for an application executive used for integrating avionics systems on modern aircraft
- It is an API of 51 routines: time and space (memory) partitioning, health monitoring (error detection and reporting), communications via "ports"
- ARINC 653 OS and applications are typically certified per DO-178B; different partitions can be certified to different DO-178B "levels"



© 2008 Wind River Systems, Inc.

WIND RIVER

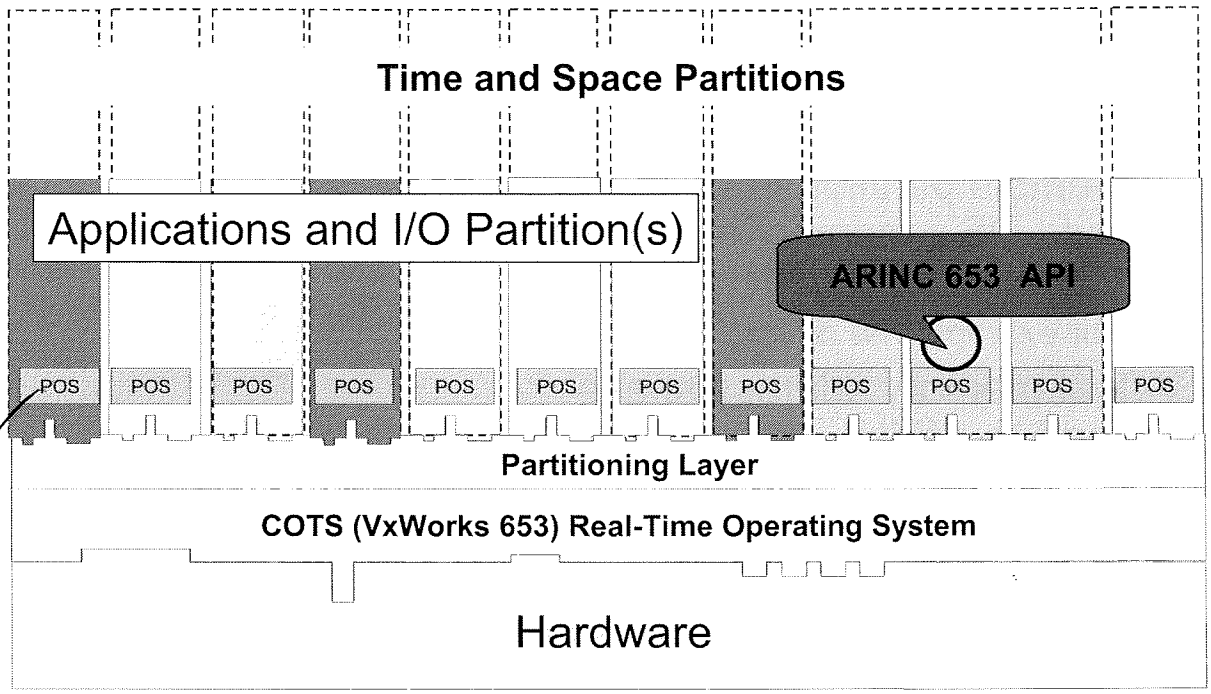
Student Notes

DO-178B → zły standard ale być może dobry

Vx → A

A + S

Module and Partition OS



Slide 7

© 2008 Wind River Systems, Inc.

WIND RIVER

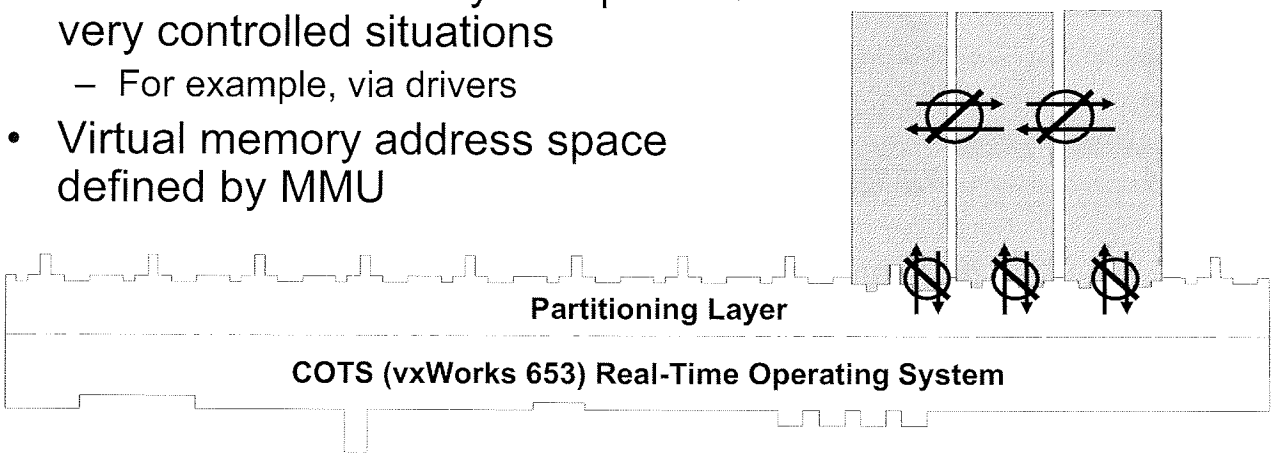
Student Notes

Often standard (with POS as standard)

MMU -> memory management unit.

Spatial Partitioning = Memory Partition

- Ensures that partition memory and control objects are not accessible outside their partition
- Prevents a partition from accessing Module OS's memory except under very controlled situations
  - For example, via drivers
- Virtual memory address space defined by MMU



Slide 8

© 2008 Wind River Systems, Inc.

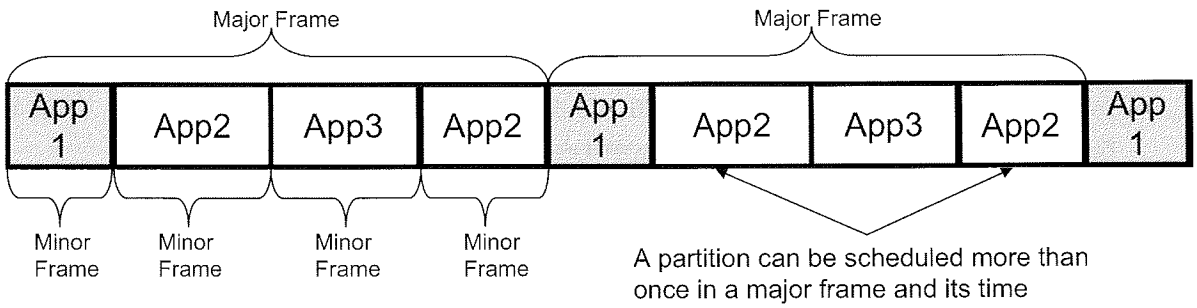
WIND RIVER

Student Notes

- Each partition has its own
- memory space
- to OS runs in (Module OS)

Temporal Partitioning

- Applications are independent of other applications on the same hardware



Each partition is scheduled for a specific duration for each minor frame

- Partitions are assigned slices of time, during which only that partition can execute
  - Slices can be assigned to a spare partition for later expansion
- Each partition's timing is predictable regardless of the presence or absence of other partitions

Slide 9

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Robust Partition DO178?

Replenish

Temporal Partitioning (Continued)

- Operations performed within one partition must not prevent another partition from having access to its entire time period
- Examples of restricted operations include
  - A process in partition A starts an asynchronous I/O operation that will generate an interrupt upon completion
    - Interrupt not guaranteed to occur during partition A's execution time
  - Near the end of partition B's allocated execution time, a process in partition B starts a Direct Memory Access (DMA) operation, which utilizes the processor bus, which is not guaranteed to complete before partition B's execution time expires

Slide 10

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Space time → dedicated vs the Module OS ?

© 2008 Wind River Systems, Inc.

Processes

- Programming unit contained within a partition
  - Executes concurrently with other processes of the same partition
- Multiple processes are supported within a partition
- Processes may be periodic or aperiodic
- Partition is responsible for the behavior of its internal processes

Priority Based Scheduling / Preemption ?

Slide 11

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- Memory for a process comes entirely from the partition in which it resides

Intra-Partition Communication

- Communication and synchronization between processes within the same partition
- General inter-process communications mechanisms use
  - Buffers
  - Blackboards
- Inter-process synchronization mechanisms
  - Semaphores → counting semaphores (ARINC)
  - Events
- Required memory is allocated from partition OS's memory pool

Slide 12

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Buffers and Blackboards

- Communication of a single message type between multiple source and destination processes
- Required memory is allocated from partition's memory pool
- Buffers allow queuing of messages
  - Each new instance of a message may carry uniquely different data
  - Stored in the message queue in first-in/first-out (FIFO) order
- Blackboards don't allow queuing
  - Single overwritten message
  - Message remains until it is either cleared or overwritten by a new instance of the message

Slide 13

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Semaphores and Events

- Used for inter-process synchronization
- Required memory is allocated from partition's memory pool
- Semaphores
  - Provide controlled access to resources
  - Counting semaphores only (a semaphore's value indicates the number of currently available resources)
- Events
  - Supports control flow between processes by notifying the occurrences of conditions to awaiting processes
  - Composed of a bi-valued state variable (states called "up" and "down") and a set of waiting processes (initially empty)

Slide 14

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Inter-Partition Communication

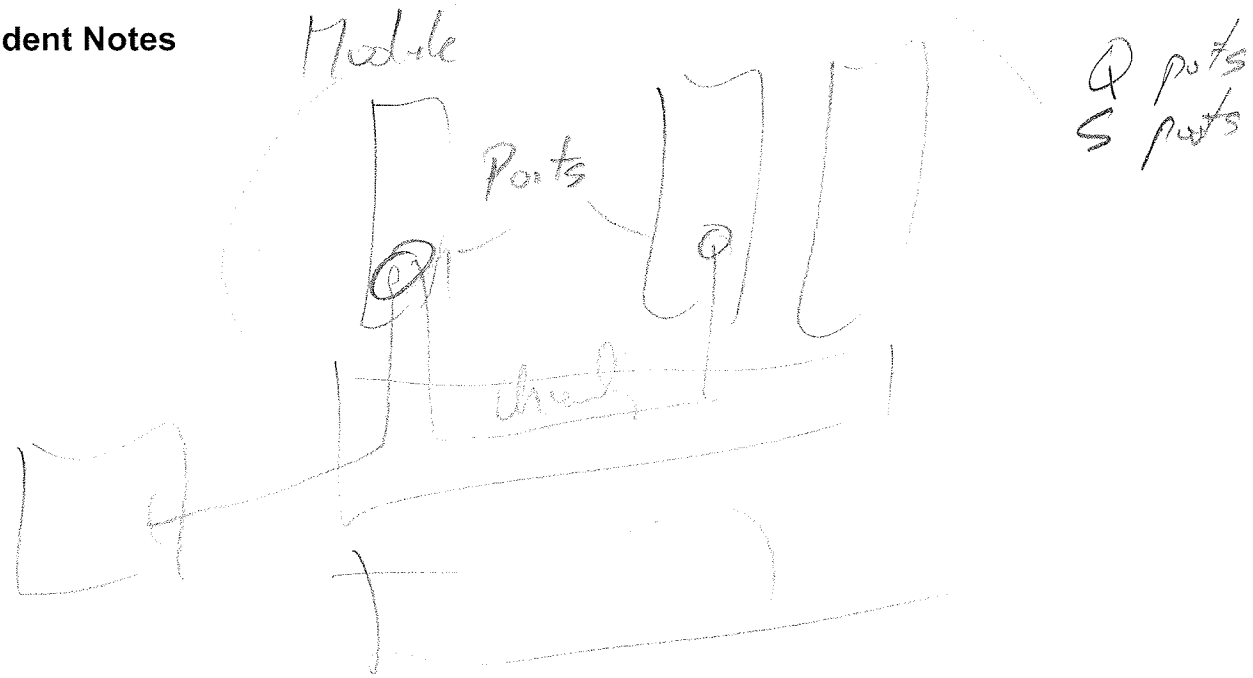
- Communication between two or more partitions executing either on the same core module or on different core modules
- All inter-partition communication is conducted via messages sent through ports
- A message is defined as a continuous block of data of finite length
- A message is sent from a single source to one or more destinations
- The destination of a message is a partition, and not a process within a partition
- A channel is a connection of a source and destination
- A channel is facilitated by a transport task running in the module OS
- Two types of ports: Sampling and Queuing
- Required port memory is allocated from the Partition OS's memory pool
  - Channel memory is allocated by the module OS

Slide 15

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes



Sampling Ports

- Successive fixed length messages carry identical but updated data
- No queuing is performed
  - Messages are overwritten
- A message remains in the source port until it is transmitted by the channel, or it is overwritten by a new occurrence of the message, whichever occurs first

Slide 16

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

W obu portach trzeba przechowywać czas na transport 000

Moduł OS → przetransportuje

Queuing Ports

- Each new instance of a message may carry uniquely different data, including variable length messages
- Multiple messages are buffered in message queues
- Transmit messages are stored in a FIFO message queue of the source port until they are transmitted
- Receive messages are stored in a FIFO message queue of the destination port until they are received by the destination partition
- Application software is responsible for handling overflow when the queuing port is full

Slide 17

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Health Monitoring (HM)

- Provides a framework to raise and handle alarms in system
- Functions at 3 levels
  - Process
  - Partition
  - Module
- Table-driven fault responses and recovery actions occur at partition and module level
- Application driven responses and recovery actions occur at process level

Slide 18

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Health Monitoring - Notification

- Partition or module level handlers can communicate information to other partitions by notifying them of given events
  - For instance, one partition handler can tell another about an event that caused it to restart the partition
- Notification can be configured so that a partition will only accept notification of certain events and from certain partitions
  - All partitions will accept notification from a module handler unless notification feature is not enabled
- Notification is configured and enabled on per partition basis

Slide 19

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Questions

1. There are three categories of ARINC Standards. True or False?
2. What does ARINC 653 allow applications to be abstracted from?
3. Only one process is supported within a partition. True or False?
4. Which ARINC standards categories define the avionics equipment's form, fit, function, and interfaces?
5. Spatial partitioning ensures that partition memory and control objects are always accessible outside their partition. True or False?

Slide 20

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

1. True: ARINC Characteristics, ARINC Specifications and ARINC Reports.
2. Underlying OS.
3. False. A partition supports multiple processes.
4. ARINC Characteristics
5. False

Review

In this chapter you learned that ARINC 653 specifies:

- Module and Partition OS
- Partitioning
  - Spatial
  - Temporal
- Process management, control, and scheduling
- Intra-Partition Communication
  - Buffers and Blackboards
  - Semaphores and Events
- Inter-Partition Communication
  - Ports
- Health Monitoring

Slide 21

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Getting Started

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Objectives

By the end of this chapter you will be able to:

- Use the tools provided by Wind River Workbench
- Use the available help
- Setup a target hardware
- Use the target server
- Configure the WDB agent

Slide 2

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Getting Started

Introduction
Getting Help
Setting Up the Host
Booting the Target
Connecting the Host and Target
Simulator

Slide 3

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Wind River® VxWorks 653 Platform 2.2

- Complete, integrated environment to accelerate software development for ARINC 653 compliant safety critical systems
- Key features include
  - ARINC 653 compliant operating system
    - Full ARINC 653 conformance
      - Health Management
      - APEX API
      - Time and Space Partitioning
  - ANSI Standard C Language Interface
  - Optional Ada 95 compiler for Ada application development
  - Certifiable to DO-178B Level A
  - Strong host tools and debug support

Slide 4

© 2008 Wind River Systems, Inc.

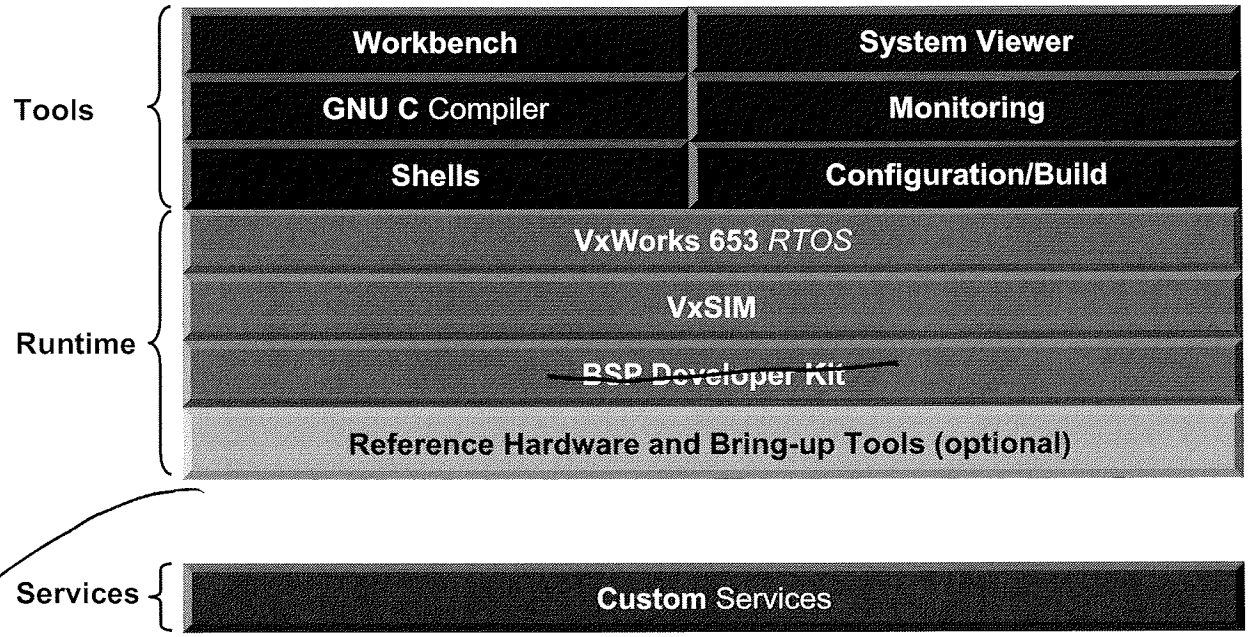
WIND RIVER

Student Notes

- Wind River's implementation of the ARINC653 specification as well as suitable development tools is called Wind River® VxWorks 653 Platform 2.2. In addition to the fully ARINC653 compliant operating system, the platform includes:
  - Workbench—an IDE common to all Wind River platforms suitable for setting up editing and debug sessions
  - Build Utilities—the GNU compiler, Tcl configurators and builds, an XML parser
  - Debugging and Analysis Tools—the Wind Shell, Debugger, Kernel Object Viewer, System Viewer
  - DO-178B Level A Certification for some OS components

© 2008 Wind River Systems, Inc.

Wind River® VxWorks 653 Platform 2.2 Components



Slide 5

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

OCD → dodatkowe narzędzia do debugowania

Wind River® VxWorks 653 Platform 2.2 Component Delivery

- DVD for product components
  - Workbench tools, which run on the host
  - VxWorks 653 operating system, which runs on the target
  - Wind River supplied Board Support Package
  - Board-specific boot applications (to download VxWorks from host)
  - Compiler Toolkit
- Online HTML manuals on operating system and tools
- Online Support (OLS)
  - Tech Tips
  - Application Notes
  - Any mandatory patches
- 'What's New' in Workbench

Slide 6

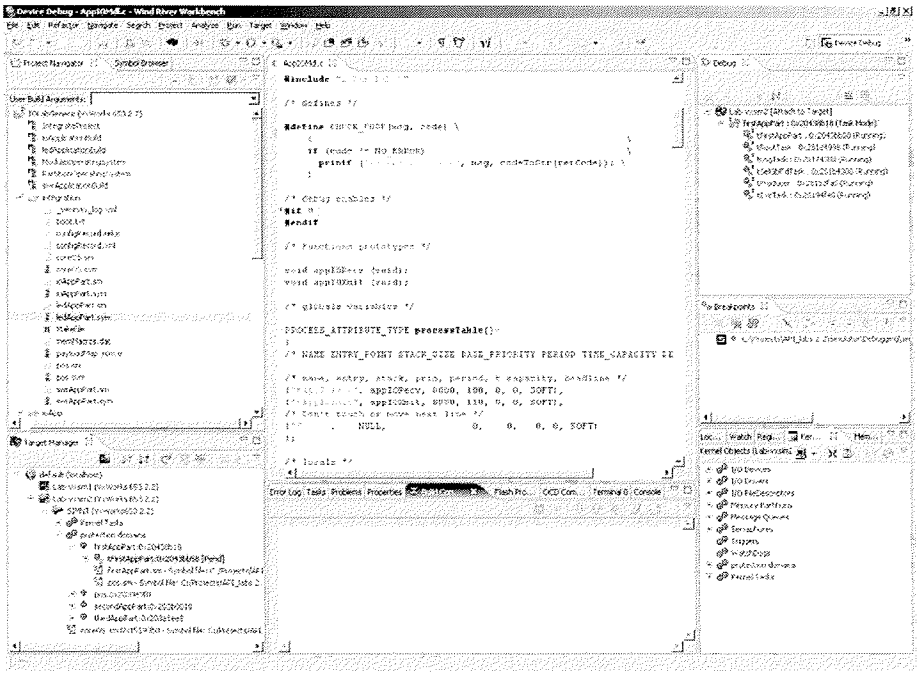
© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- The GNU ToolKit family of compilers, linkers, etc., is automatically installed in the Tornado directory tree
  - In Wind River® VxWorks 653 Platform 2.2 the C compiler is GCC version 3.32

Wind River Workbench



Slide 7

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- Device Debug perspective is shown above

Eclipse

- Eclipse is an IDE for anything, and for nothing in particular.[1]
- The Eclipse Platform is designed and built to meet the following requirements:
  - Support the construction of a variety of tools for application development.
  - Support an unrestricted set of tool providers, including independent software vendors (ISVs).
  - Support tools to manipulate arbitrary content types (such as HTML, Java, C, JSP, EJB, XML, and GIF).
  - Facilitate seamless integration of tools within and across different content types and tool providers.
  - Support both GUI and non-GUI-based application development environments.
  - Run on a wide range of operating systems, including Windows®.
  - Capitalize on the popularity of the Java programming language for writing tools.

Slide 8

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- The goal of Eclipse is not to provide developers with all of the tools that are necessary for product design and development
  - Instead, the goal is to enable to cooperation and interaction of tools in a diverse development environment.
- This closely aligns with Wind River's goal of providing a highly portable, highly integrated, and scalable development environment for our customers, hence our selection of Eclipse as the framework for Wind River Workbench.
- [1]Eclipse White Paper, eclipse.org

Was; Eclips 3.2

Eclipse Platform

- The Eclipse Platform UI paradigm is based on **editors**, **views** and **perspectives**
- **Editors** allow the user to open, edit, and save objects
- **Views** provide information about some object that the user is working with in the workbench
  - A view may augment other views by providing information about the currently object being edited
- Workbench can have several separate **perspectives** defined, only one of which is visible at any given moment
  - Each perspective defines its own set of editors and views to be displayed and arranged
  - Several different types of views and editors can be open at the same time within a perspective
  - Once an editor or view is open, its instance is shared among all perspectives defining it
  - The user can quickly switch perspectives to work on a different task, and can easily rearrange and customize perspectives to better suite a particular task

Slide 9

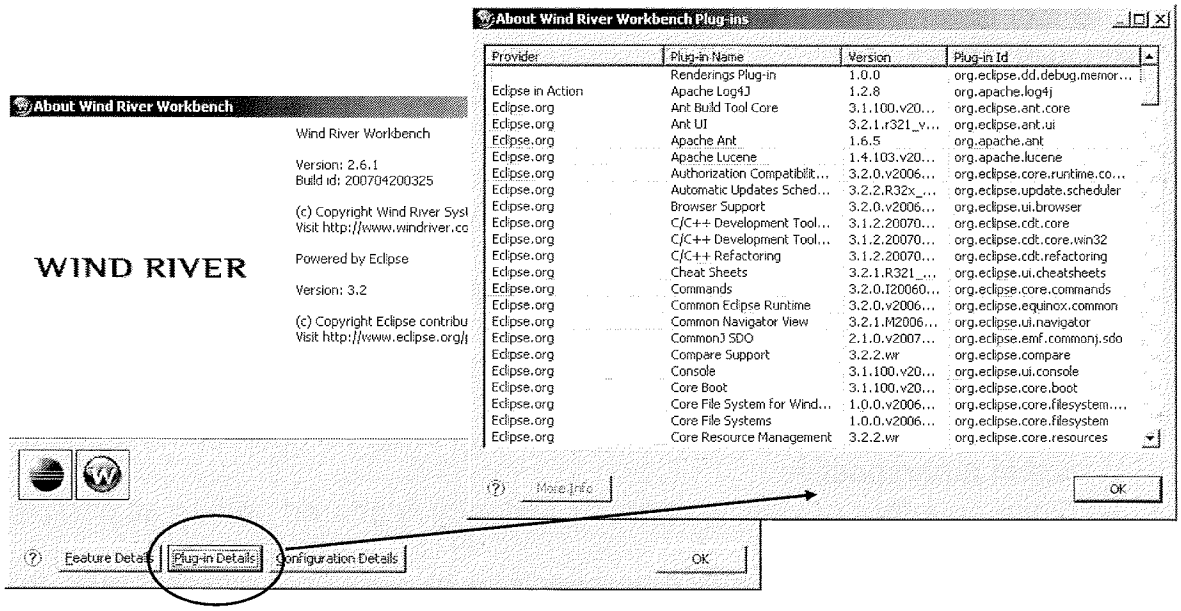
© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- Eclipse introduces new language for some elements that you will need to become familiar with:
  - Views instead of Windows
  - Perspectives instead of Desktops

Wind River Workbench Plug-ins



Slide 10

© 2008 Wind River Systems, Inc.

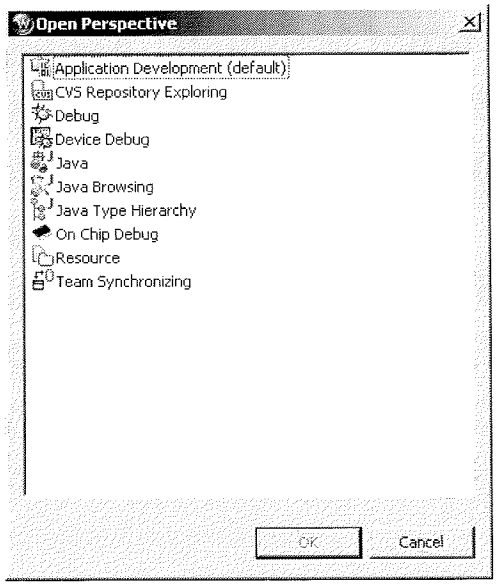
WIND RIVER

Student Notes

- Plug-ins code is stored under the workbench-2.6/wrwb/2.6/plugins subdirectory

Wind River Workbench Perspectives

- Wind River Workbench functionalities are by default grouped in different Perspectives
  - Technology developed by Wind River are organized and presented in following Perspectives:
    - Application Development
    - Embedded Debug
    - MemScope
    - ProfileScope



Slide 11

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Development Tools

- **Simulator:** Simulates a VxWorks target
- **Target Manager:** Manages connections with a target
- **Shell:** Accesses a target interactively
- **Debugger:** Enables source-level, multi-task debugging
- **System Viewer:** Analyzes a multitasking application
- **Triggering:** Associates actions with events
- **Configuration:** Configure and build system

Slide 12

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Monitoring Tools

- Performance: Monitors and reports CPU usage either system-wide or within a partition (per thread)
  - Idle time
  - Time spent processing interrupts
  - Time spent for application processing
- Memory Usage: Monitoring system memory usage for heaps, stacks, ports, and health monitoring
  - Reports usage on a per partition basis
- Port: Logging and reporting of all port activity in a system
  - Includes both port control data and message contents
  - Monitored data may be streamed to the host or collected in a log and retrieved later
- Target side monitoring is permanent, minimally intrusive and is included in the software
- Each monitoring tool has a host-side component, which formats and reports data on command

Slide 13

© 2008 Wind River Systems, Inc.

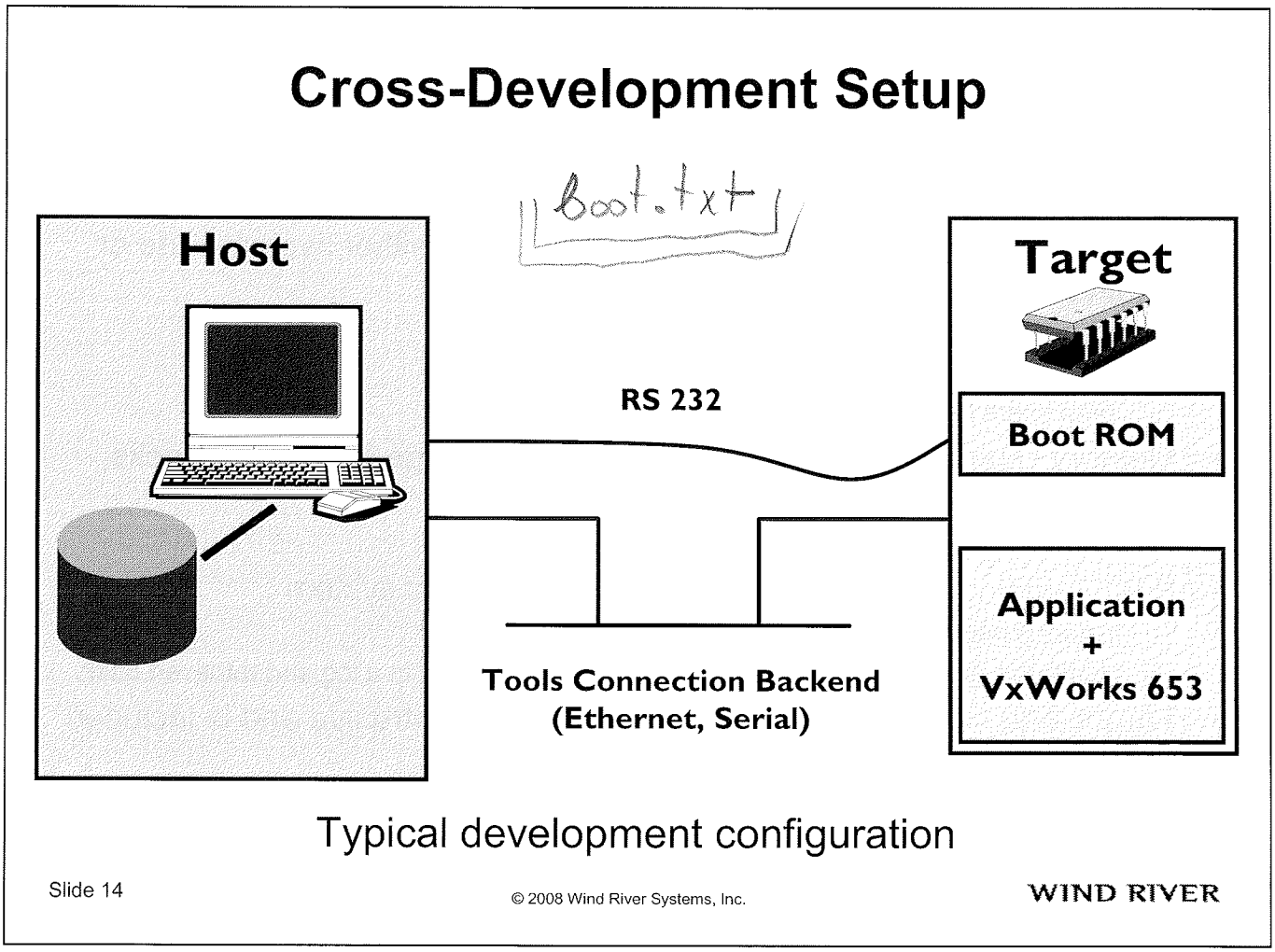
WIND RIVER

Student Notes

Tools are qualified = you can use that tools in order to demonstrate application for certification

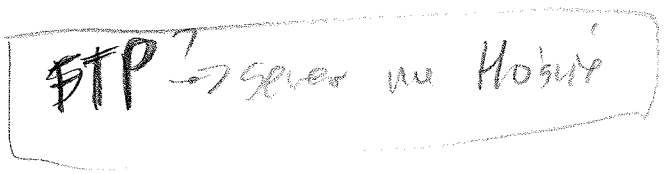
↳ test

Sys – NO
User – NO
– NO
Kernel mod Build – yes
debug – NO



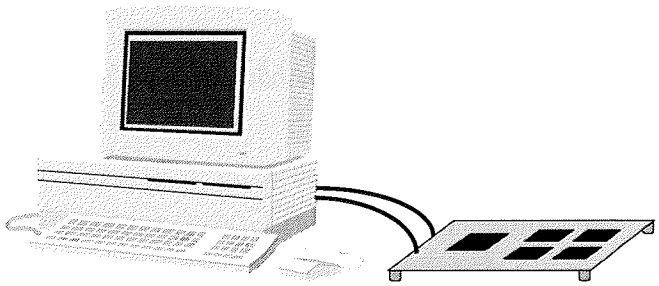
Student Notes

- Coding and building is done on the host, using tools for
  - Editing
  - Compiling and linking
  - Tracking source code
- Testing and debugging is done on target, using tools for
  - Loading
  - Execution
  - Debugging
  - Monitoring performance



Setting up for Cross-Development

- Set up target hardware
- Set up host
  - Install Platform
  - Configure networking
  - Configure Licensing
  - Specify the target boot parameters and boot the target
  - Start the registry, if not configured to automatically start
  - Configure and launch a target server
    - Target servers manage the host tools' interaction with a specific target
    - Registry manages target servers
  - Build application system
  - Boot target



Slide 15

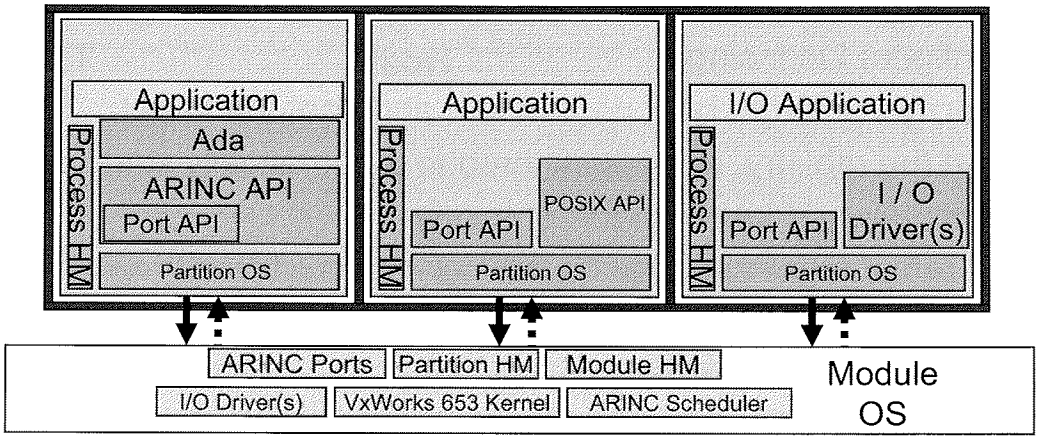
© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

VxWorks 653

- Module O/S is configured to support
  - Hardware
  - I/O (split mode or Module OS)
  - Development
  - Partition Scheduling
  - Intra Partition Port Communication
- Partition O/S is configured to support an API type
  - ARINC
  - POSIX
  - Partition OS (default)
  - I/O (partition based)



Slide 16

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- HM – Health Monitor

ARINC 653 Specifications

- VxWorks 653 meets ARINC 653 specifications by providing
  - Strict spatial and temporal partitioning
    - Communication between applications within discrete partitions is strictly controlled by the Module OS
    - Partitioned applications interacting with the Module OS are well bounded so as to prevent them from affecting the entire system
    - Erroneous behavior within an application is confined to its partition
  - Full support of the ARINC 653 APEX API, including both inter and intra partition communication
  - Health Monitoring
    - Alarm detection, logging and response
  - Partition restart capability
  - Module restart capability

Slide 17

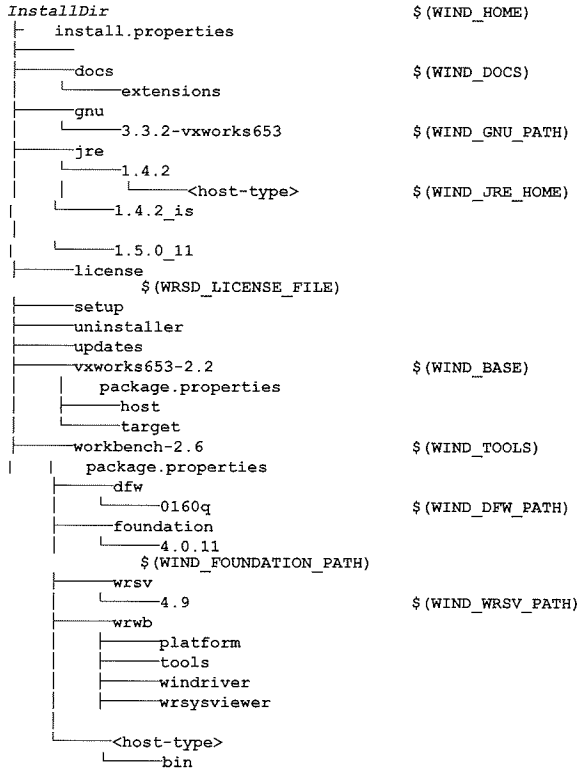
© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- Runtime Enhancements
  - Separate memory pools for ARINC port channels and HM logs
    - These are isolated from the Module OS memory pool so it is no longer required to rebuild/resize the Module OS when these regions are resized
  - Floating point exceptions in partitions
    - Allows the developer to selectively support floating point exceptions on a partition basis
  - Separate stack for system call handlers that run in the context of the Module OS memory rather than partition OS memory
    - Provides for greater robustness and security
  - Multiple Partition OSs
    - Different types of partition OSs are now possible in a single system. This implies multiple system shared libraries. Combinations include full and certifiable subsets of the Partition OS
  - ARINC priority preemptive scheduler (APPS) scheduling
    - System-wide scheduling scheme of partitions that adds a global priority-preemptive scheme during the idle time within an ARINC 653 time-preemptive schedule. Allows for global priority preemptive scheduling to occur within the ARINC653 scheduler
  - Module OS Interface Library (COIL)
    - Exposes the API's needed to implement a user defined partition operating system. A library of routines, independent of the Partition OS, that supports the management of interrupts and exceptions, device I/O, inter-partition messaging, and injection of health monitoring events

Relevant VxWorks 653 Directories



- ***WIND_**** variables specify location of tools and run-time
  - In an installation most variables can be derived from ***WIND_HOME***
- **package.properties** file describes the package, environment and dependencies to other packages
- **install.properties** created at install time by aggregating all **package.properties** files
- ***InstallDir*** is specified when installing the CDs

Slide18

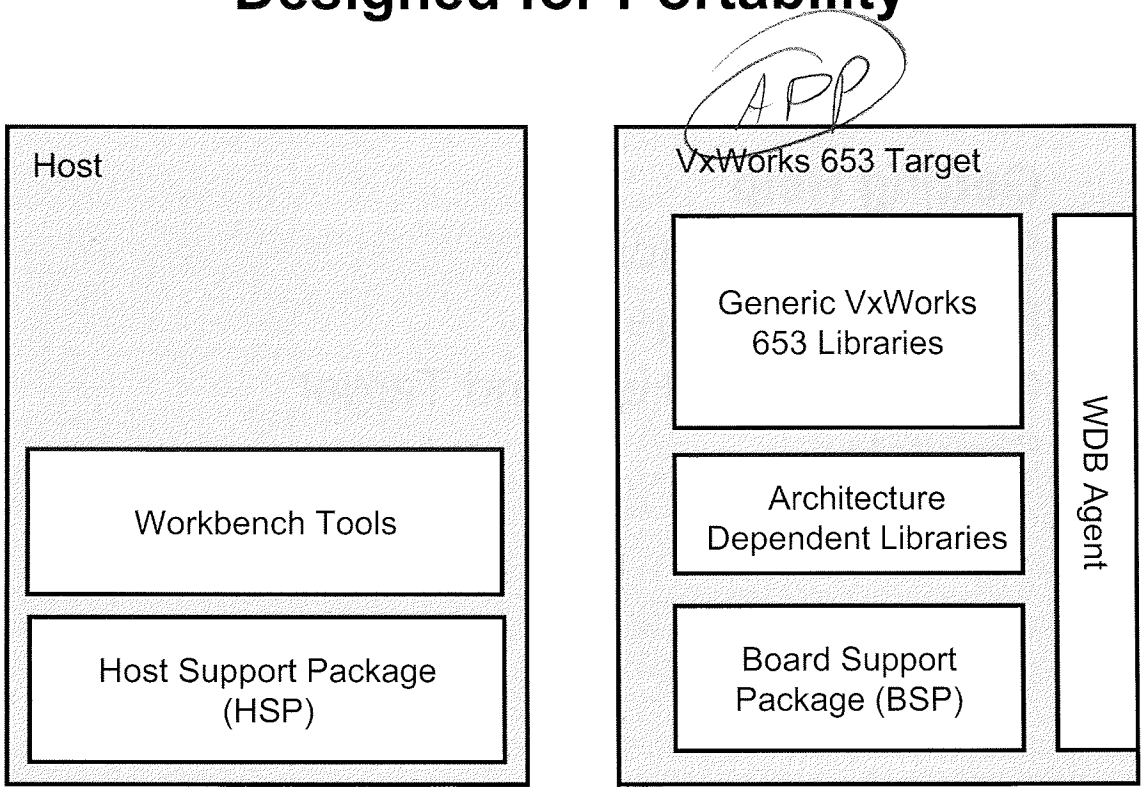
© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- **setup.log** is contained in ***WIND_HOME***
- Recommended NOT to have spaces in the ***InstallDir*** directory name

Designed for Portability



Slide 19

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- Most of the operating system is machine independent
- The machine-dependent portions of VxWorks 653 are
  - Architecture specific (initially, PPC, MVME)
  - Board specific
    - For example, hardware initialization, bus interface, clock/timer functions encapsulated in a Board Support Package (BSP)
  - Host specific
    - Cross-compilers, tornado tools & target server encapsulated in a Host Support Package (HSP)
      - For example, Windows NT, Windows 2000, Solaris 2.7, and host simulator.
  - The WDB debug agent provides target agency
    - Because it is independent of the operating system, it supports system-level debugging with the remote source-level debugger

Getting Started

Introduction
Getting Help
Setting Up the Host
Booting the Target
Connecting the Host and Target
Simulator

Slide 20

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Product Manuals

Manuals provided include:

On the host tools

- ***Wind River Workbench User's Guide 2.6, VxWorks 653 Version***
- ***Wind River System Viewer User's Guide***
- ***Wind River Workbench Migration Guide, 2.5***

On the operating system

- ***VxWorks 653 Configuration and Build Guide***
- ***VxWorks 653 Programmer's Guide***
- ***VxWorks 653 Partition OS API Reference***
- ***VxWorks 653 Module OS API Reference***

On-chip debugging

- ***Wind River Workbench On-Chip Debugging User Tutorials***

Connectivity

- ***Wind River DO-178B Network Stack for VxWorks 653 Guide***

Slide 21

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- Other manuals include
  - ***Wind River VxWorks 653 Platform 2.2 Release Notes***
  - ***GNU Toolkit User's Guide*** (HTML only)
  - ***GNU Make User's Guide*** (HTML only)
- All Workbench and VxWorks manuals are available in HTML format
  - Manuals can be accessed using the "Help" menu

Other Resources

- Online Support self-help web pages
  - Search engine, FAQ, patches, document updates, known problem list, online chat forum
- Training Courses
  - VxWorks 653 BSP/DD
  - Real-Time Programming for Embedded Systems
- Technical Support
  - Installation problems and media errors
  - Wind River software, documentation, and service errors
  - Understanding Wind River product functionality
- Usenet newsgroups such as comp.os.vxworks
- Tutorials bundled with Wind River® VxWorks 653 Platform 2.2
- ARINC spec document

Slide 22

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- Additional newsgroups of interest include
  - **comp.os.realtime** for general real-time topics
  - **comp.arch.bus.vmebus**
  - **comp.arch.embedded**
- Technical Support is available if purchased
  - In the USA, call 1-800-USA-4WRS
    - Hours are from 5 A.M. to 5 p.m. Pacific Time
    - Outside of the USA, contact your local Wind River office
- Online Support may be reached from the Wind River Systems home page
  - **www.windriver.com**
  - Access Online Support from Customer Support
  - Online Support access requires registration for a password and user ID

© 2008 Wind River Systems, Inc.

Getting Started

Introduction
Getting Help
Setting Up the Host
Booting the Target
Connecting the Host and Target
Simulator

Slide 23

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Installing Wind River® VxWorks 653 Platform 2.2 on Windows Host Systems

- Starting Workbench from the program group, the environment is automatically set
  - Otherwise start **wrenv** to use the tools from the command line
  - Environment variables can also be set in login scripts
- Registry startup procedure is defined when installing Workbench
  - The registry may be installed
    - In the Startup folder
    - As a service
  - The registry may also be started manually
- After installation is complete, specify the registry host to be used from the "Tools → Options → Target Manager" page
- Consult Windows documentation to configure TCP/IP

Slide 24

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Installing Wind River® VxWorks 653 Platform 2.2 on UNIX Systems

- Set environment variables required by Workbench in the host **.cshrc** (or **.profile**) file
  - The script **wrenv.csh** (or **wrenv.sh**) may be sourced to set these variables
- Modify the registry host's start-up files to start the registry daemon, **wtxregd**, when the host boots
- Configure network access information
  - Obtain target IP addresses
  - Assign host names to targets (optional)
  - If booting with RSH, list the target's name in **~/.rhosts**
  - If necessary, modify routing tables

Slide 25

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Workbench License Management

- Running a target server and System Viewer requires a license
- License is configured as part of Platform installation
- Installation program allows
  - Specifying licensing method
    - "Unique user" licenses require a network and a license server running on one host
      - License server allocates licenses to clients, on a first come first serve basis
    - Node locked licenses are valid for a single user on a single machine
      - No license server is required
  - Obtaining a license by
    - Contacting a Wind River database server
      - Requires an active Internet connection
    - Sending an email to license@windriver.com
  - Installing and configuring a license server

Slide 26

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- Configuring a license server should be done by a system administrator
- To configure license management later, use the "SETUP –L" tool on the installation CD
- On Windows, the license server is started automatically, when the server host reboots (unless specified otherwise during the license setup)
  - The FLEXlm License Manager control panel may be used to manually start or reconfigure the license server
- On UNIX, the license server is started manually with the "lmgrd [options]" command
  - This command can be added to the server host's start-up files
- The number and type of licenses are determined when the product is purchased
- See the *Wind River VxWorks 653 Platform 2.2 Getting Started* guide for license setup information
- Workbench uses the FLEXlm license management system
  - For more information about FLEXlm, visit http://www.globetrotter.com

Starting Wind River Workbench

- Workbench development suite perspectives can be used to manage a cross-development session:
  - Configure and manage target - host connections
  - Start Host tools
  - Invoke online manuals
- To start Wind River Workbench (Windows):
  - Select ***Wind River → Wind River Workbench 2.6 → Wind River Workbench 2.6*** from the ***Start*** menu in Windows
- To start Wind River Workbench (UNIX):

```
> startWorkbench.sh
```

Slide 27

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

wrenv Utility

- **wrenv** utility is responsible for constructing development environment settings for all tools
- **wrenv** is a shell independent replacement for "**Vars**" scripts
  - Replaces the need for ***.bat**, ***.sh** and ***.csh** script variants
- **wrenv** provides environment that is portable between multiple machines
  - Shell scripts previously contained absolute paths
- **wrenv** can be used to provide information for any tools like IDE, configurators, make, and so forth.
  - Tools no longer need hard code paths or symbols– just rely on **wrenv**
- Default functionality of **wrenv** creates a command shell with a preloaded environment
  - **wrenv** can also simply dump the environment settings to standard output in a format that can be sourced by plain, *.bat, *.sh, *.csh or *.tcl utilities
  - **wrenv** can be requested to dump dependency information useful to a variety of tools

Slide 28

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

wrenv Usage

wrenv [options] [env=value] [command [args]]

Options are:

-v	verbose debug mode
-e	existing environment variables have precedence
-f fmt	select format of environment (plain, sh, csh, bat, tcl. Default is plain)
-i path	specify absolute install.properties path
-o type	select operation (run, print_env, print_varsprint_compatible, print_packages, print_package_name. Default is run)
-p package	selects the package by name
-r root	specify root path for package

Slide 29

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Getting Started

Introduction
Getting Help
Setting Up the Host
Booting the Target
Connecting the Host and Target
Simulator

Slide 30

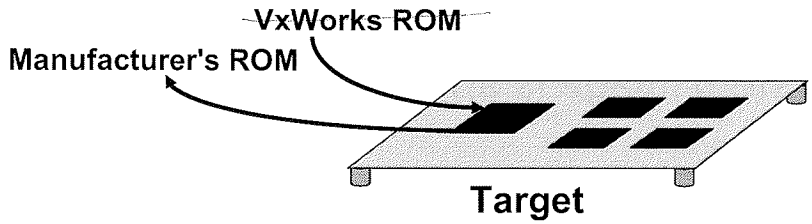
© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Target Hardware Configuration

- Reprogram Flash with VxWorks bootable image or replace the manufacturer's ROM with VxWorks ROM



- Connect host and target (serial, ethernet cables, and such)
- See your BSP Reference for additional hardware setup considerations

Slide 31

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- Before you boot your target, you may first need to:
  - replace the manufacturer's ROM's with VxWorks ROM's
  - put the VxWorks boot ROM into flash memory
  - insert the VxWorks boot floppy into the floppy drive
  - jumper your target board
  - reprogram the flash with the VxWorks Boot Loader image

Boot ROM

- Target boot ROM's code executes on power up
- Boot ROM's do not contain the operating system under which your application will run
- VxWorks system image is an object module on the host
- Boot ROM code:
  - typically loaded/decompressed into RAM
  - allows setting of boot parameters
  - downloads operating system (and applications) into target memory via the network
  - starts executing VxWorks

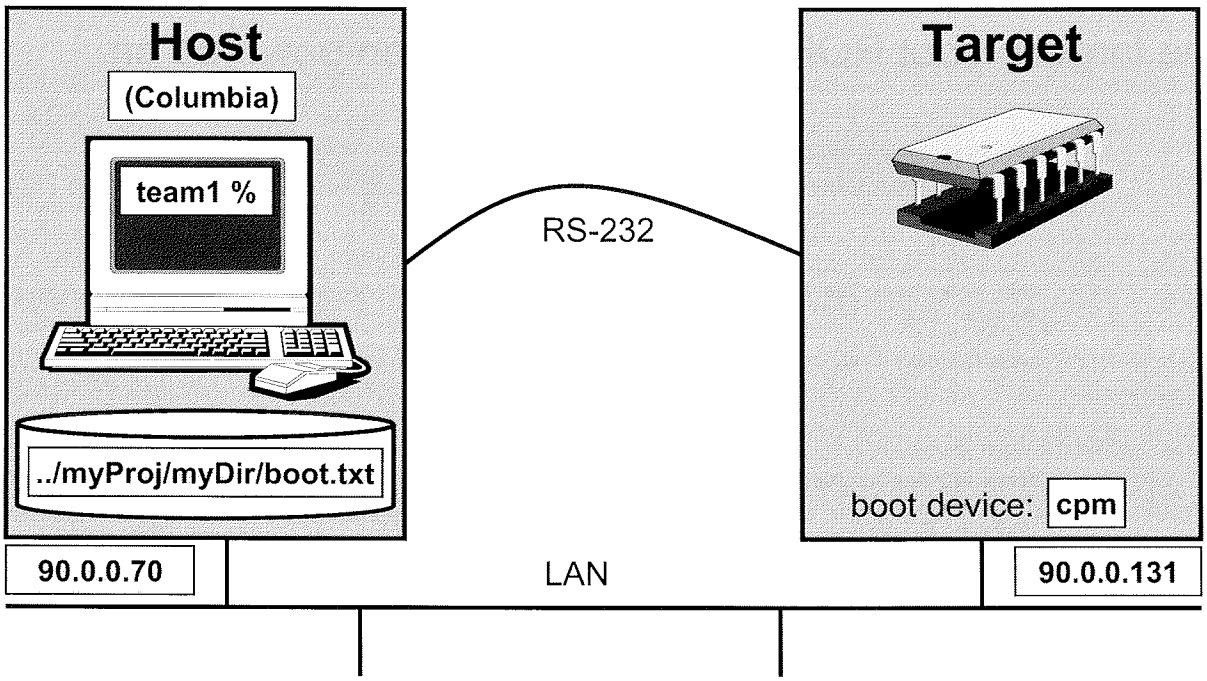
Slide 32

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Boot Configuration



Slide 33

© 2008 Wind River Systems, Inc.

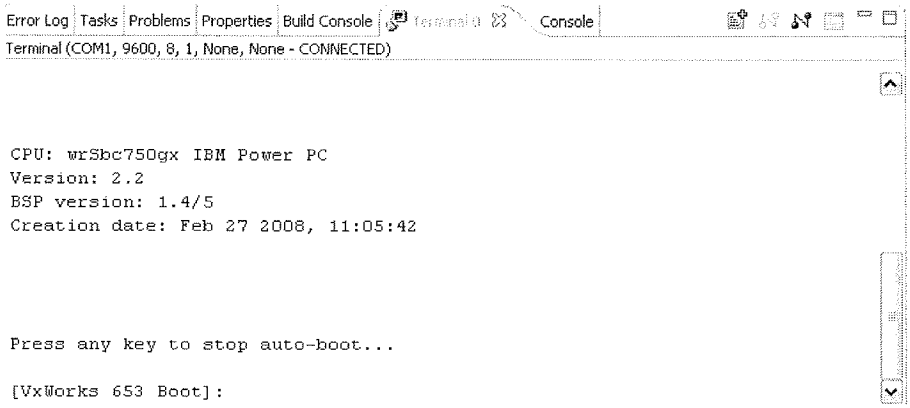
WIND RIVER

Student Notes

- By default
  - Boot parameters are set via a serial link
  - VxWorks is booted through the ethernet (which is much faster)
- Key boot parameters
  - Boot device — Network interface to use
  - Host internet address — Address of host from which to boot
  - Path of VxWorks image — File to transfer over network
  - User name — For file access purposes
  - Target internet address — Defines the target's address

Powering the Target

The target's boot ROM code executes when powered on.



Slide 34

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- On power-on, the boot ROM's I/O is through a serial port
- One may type 'h' at the boot prompt to see a list of commands, for example

```
[VxWorks 653 Boot]: h
 ?               - print this list
 @               - boot (load and go)
 p               - print boot params
 c               - change boot params
 l               - load boot file
 g adrs          - go to adrs
 d adrs[,n]      - display memory
 m adrs          - modify memory
 f adrs, nbytes, value - fill memory
 t adrs, adrs, nbytes  - copy memory
 e               - print fatal exception
 $dev(0,procnum)host:/file h=# e=# b=# g=# u=usr [pw=passwd] f=#
                   tn=targetname s=script o=other
Boot flags:
  0x02  - load local system symbols
Available Boot Devices:
Enhanced Network Devices: mv mv mv
Other Devices:
```

Boot Program Commands

```
[VxWorks Boot]: h

 ?                      - print this list
 @                      - boot (load and go)
 p                      - print boot params
 c                      - change boot params
 l                      - load boot file
 g adrs                 - go to adrs
 d adrs[,n]             - display memory
 m adrs                 - modify memory
 f adrs, nbytes, value  - fill memory
 t adrs, adrs, nbytes   - copy memory
 e                      - print fatal exception
 v                      - print boot logo with version
 n netif                - print network interface device address
 N                      - set ethernet address
 $dev(0,procnum)host:/file h=# e=# b=# g=# u=usr [pw=passwd] f=#
                           tn=targetname s=script o=other
 Boot flags:
   0x02  - load local system symbols
   0x04  - don't autoboot
   0x08  - quick autoboot (no countdown)
   0x20  - disable login security
   0x40  - use bootp to get boot parameters
   0x80  - use tftp to get boot image
   0x100 - use proxy arp
Available boot devices:
Enhanced Network Devices: mv mv mv
Other Devices
```

Slide 35

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Changing Boot Parameters

```
[VxWorks Boot]: c

'.' = clear field;  '-' = go to previous field;  ^D = quit

boot device          : mv0
processor number     : 0
host name            : training
file name            : C:\WindRiver\vxworks653-2.2\target\proj\
wrSbc750gx_default\integration\boot.txt
inet on ethernet (e) : 10.10.10.1
inet on backplane (b):
host inet (h)        : 10.10.10.10
gateway inet (g)     :
user (u)             : training
ftp password (pw) (blank = use rsh): training
flags (f)            : 0x0
target name (tn)     : WR750GX
startup script (s)   :
other (o)            :
```

Target boards with NVRAM retain changes after power is cycled.

Slide 36

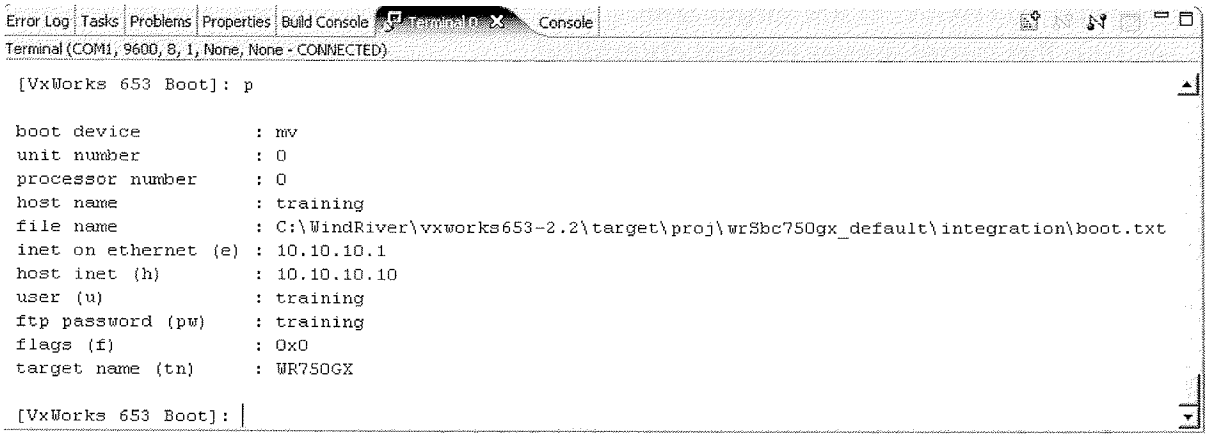
© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- Alternatively, can type boot parameters on a single line at the boot prompt:
  - $motfcc(0,0)training:`C:\WindRiver\vxworks563-2.2\target\proj\wrSbc750gx_default\integration\boot.txt` h=10.10.10.10 e=10.10.10.1 u=training pw=training tn=vx653
- Keeping this line in a file on the host machine allows cut and paste (in host's windowing system) for boot
- Can also make new boot ROM's with correct default parameters by specifying the value of DEFAULT_BOOT_LINE in ../target/config/<bspName>config.h file

Boot Parameters



Unset boot parameters are not displayed by "p"

- For example, gateway inet ("g") startup script ("s") and other ("o") parameters are not shown

Slide 37

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- Use the "h" command to determine available boot devices
- For file access permission to download the VxWorks system image and other application modules
  - Specify a user name, and
  - Edit ~/.rhosts file and leave the password parameter blank (for rsh protocol) or specify a password (for ftp protocol)

Rebooting

- To generate a warm reboot
  - Type ***reboot( )*** at a shell prompt, or
  - Type "CTRL-X" in the target shell
  - Use the target server ***reboot*** command
- To generate a cold reboot
  - Turn power off and on
  - Press the abort button on the board
  - Press the reset button on the chassis
- On a cold reboot
  - RAM is zeroed out to avoid parity errors reading uninitialized memory
- In a typical VME environment
  - Abort generates a warm reboot
  - Reset causes a cold reboot for all CPUs on the bus.
- Health Management can also generate a cold/warm reboot

Slide 38

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Alternative Booting Schemes

- RAM Payload Images
  - RAM payload images are used for development systems where restart capability is required
  - The image consists of a single binary file containing all the components of the system, as well as the boot code for the image
- ROM Payload Images
  - ROM payload images are used for deployed systems
  - The image resides permanently on the target in ROM, so no network connection to the host is required to boot the system
  - When the target is booted, the image is copied from ROM to RAM and executed
  - Restart is supported because the original image is available in ROM

Slide 39

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- Some boot mechanisms will require rebuilding the boot ROM
- Downloaded Image is the only image type supported by the simulator.
- Because two copies of a RAM payload image reside in RAM at the same time, the use of a RAM payload requires approximately twice the memory of other image types. A RAM payload image is transferred from the host to the target RAM over the network by the boot ROM. The image is installed in a section of RAM configured as a RAM payload area. Once the RAM payload is installed in RAM, a copy program is initiated that copies the image into the area of RAM where it will be executed, and then executes it. Because a copy of the image is maintained in the payload area of RAM, it is possible to perform a warm or cold restart of the system without rebooting the target.
- The ROM payload image is intended to be burned into ROM on the target. The ROM payload image includes a boot ROM generated specifically to boot that image. It replaces the default boot ROM that was used to download other image types over the network. The composition of the image is configurable by the user and may consist of one or more binary files containing different components of the system, and a separate file containing the boot code for the image

Alternative Booting Schemes (Continued)

- Download (net boot) Images
  - Consists of a separate binary file for each component and a boot file that tells the system how to boot the image
  - Transferred from the host to the target RAM over the network using the boot ROM
  - Partition restart is not available with a downloadable image
  - The only way to restart the system is to reboot, which will download the system image again

Slide 40

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Getting Started

Introduction
Getting Help
Setting Up the Host
Booting the Target
Connecting the Host and Target
Simulator

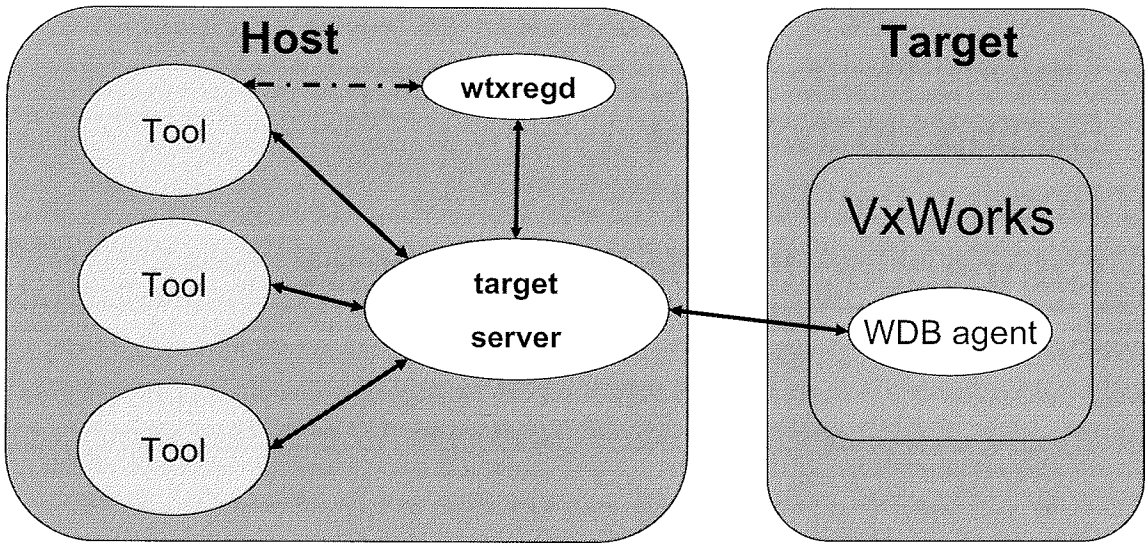
Slide 41

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Tools Communication Architecture



- wtxregd must be started before target server and tools
- Tools, registry, and target server can run on different hosts

Slide 42

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- wtxregd, the Wind Registry Daemon, manages the information a tool needs to connect to a target server. Tools initially contact the registry to find out how to contact a particular target server
- Workbench automatically starts the Wind Registry Daemon and the Target Server when the tools are run on the local host.

Target Server

- After booting a target, you must start a target server to access the target using the host tools
- Target server provides host-based management of target resources needed by development tools:
  - Communication with debug agent on target
  - Dynamic module loading and unloading
  - Host-resident symbol table for target
  - Allocation of memory on target for host tools
  - Cache of target program text segment memory
  - Virtual I/O facilities
- Flexible scheme:
  - Minimizes tool's impact on target
  - Tools independent of type of communication link

Slide 43

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- The target server satisfies a tool's WTX request by:
  - manipulating information it has cached
  - sending requests to the WDB agent on the target

WDB Agent Configuration

- WDB agent on target serves host's target server:
  - Reads and modifies memory
  - Sets and clears breakpoints
  - Creates, starts, stops, and deletes tasks
  - Calls functions
  - Gathers system object information
- WDB agent can be configured:
  - Specify task, system, or dual debug mode
  - Select communication strategy consistent with target server back end

Slide 44

© 2008 Wind River Systems, Inc.

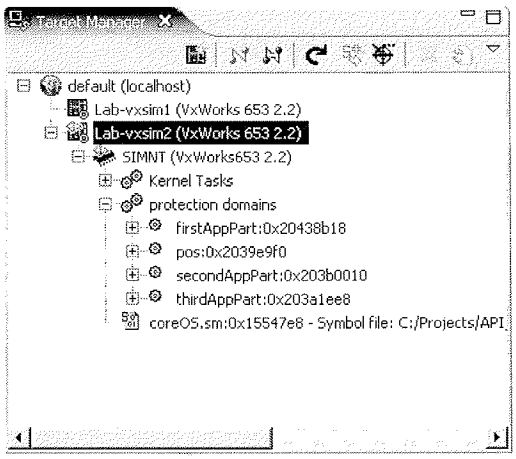
WIND RIVER

Student Notes

- The target server and target agent communicate via the Wind Debug (WDB) Protocol.
- The agent uses target memory to satisfy tools' requests.
- Using a dedicated pool of memory for the agent, host tools' intrusion on the target is minimized.
- To obtain information on target system objects, a tool communicates via the target server with the target agent to gather and return the requested information.

Target Manager

- Target Manager is used to configure, start and manage host - target connections



Slide 45

© 2008 Wind River Systems, Inc.

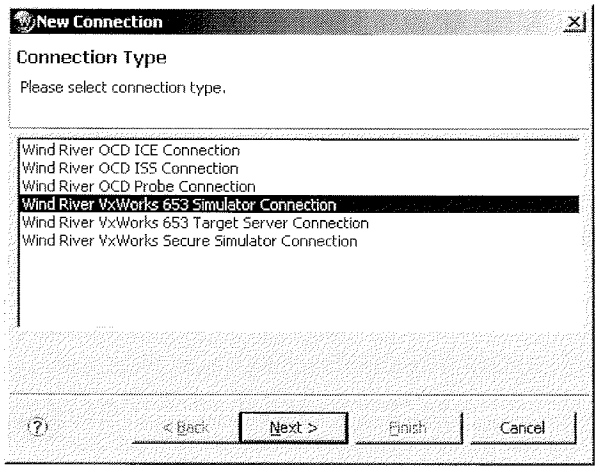
WIND RIVER

Student Notes

- Default view in both the Application Development and Embedded Debug perspectives
- To bring up the target manager window select Window->Show View->Target Manager

Configuring a Host-Target Connection (Step1)

- Performed using Target Manager



Slide 46

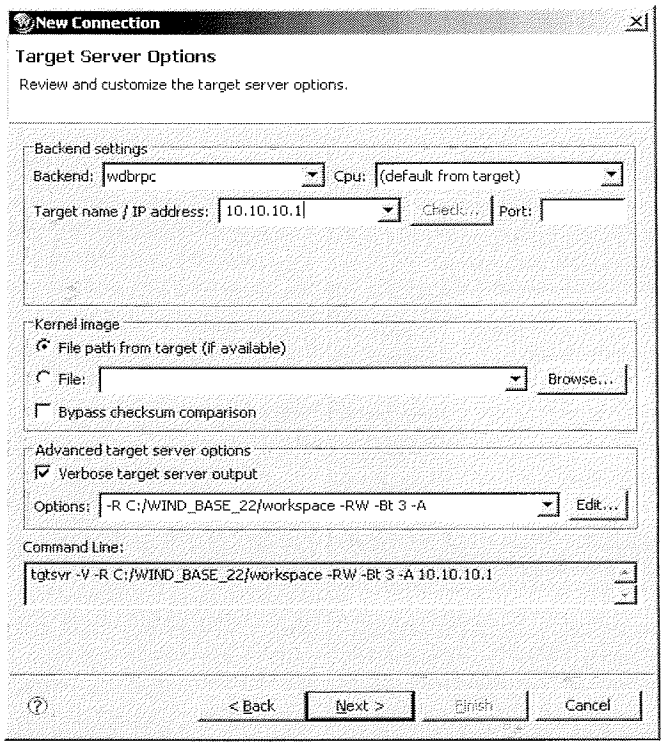
© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- Right-click on **default(localhost)** and select **New->Connection**.. Or click on the **'Create a New Target Connection'** button
- In the **"New Connection"** window select the appropriate connection, for example:
  - '**Wind River VxWorks 653 Target Server Connection**" for hardware targets
  - **"Wind River VxWorks 653 Simulator Connection"** for the simulator

Configuring a Host-Target Connection (Step 2)



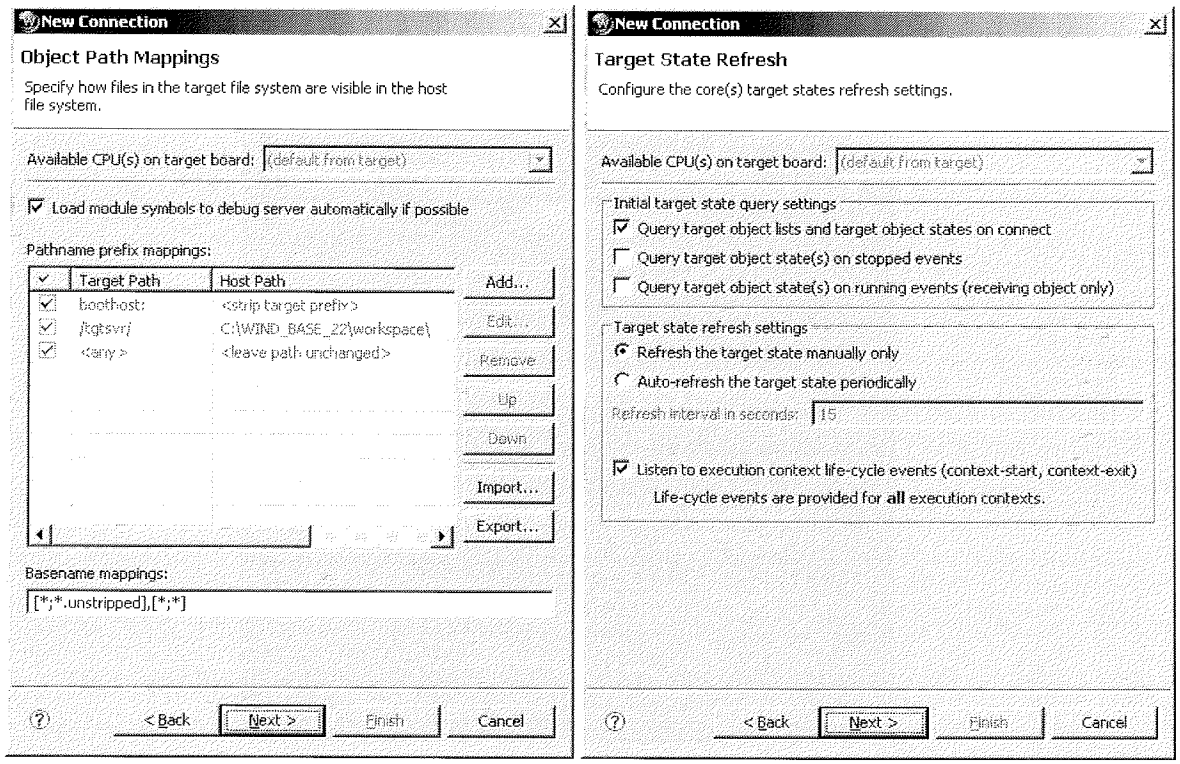
© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- TSFS properties are specified in the “Other Options” above
- Some of the options that can be entered in the “Other Options” field:
  - -A Export all symbols
  - -Bt Timeout for backend
  - -Br Retry number for backend
  - -R Root directory for target server file system
  - -RW Make target server file system read/write
- By Selecting the **Edit...**, other options can be set from the GUI

Configuring a Host-Target Connection (Step 3)



Slide 48

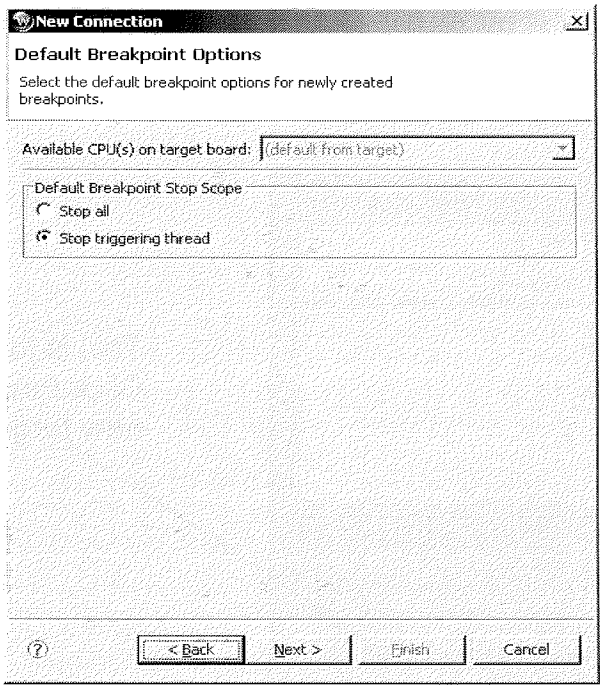
© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- Review and/or add object path mappings

Configuring a Host-Target Connection (Step 4)



Slide 49

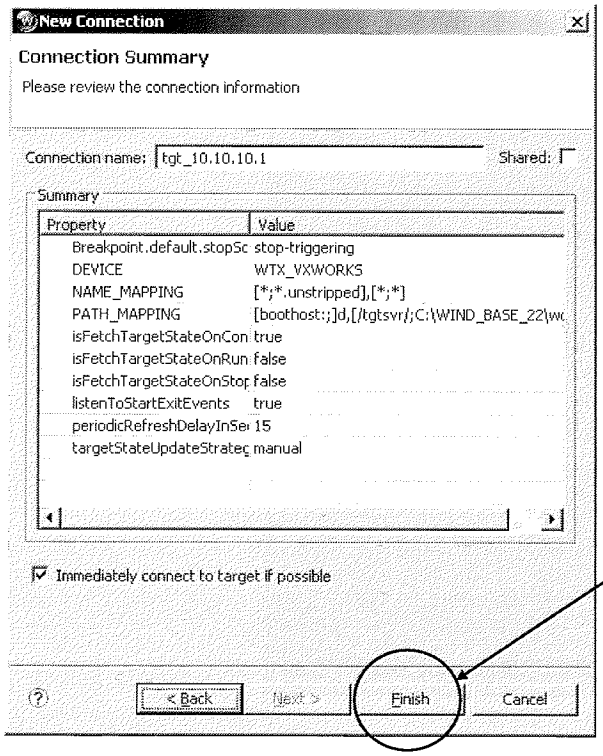
© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- Set the default breakpoint options

Configuring a Host-Target Connection (Step 4)



This is just a review step. Click Finish to setup the new target connection.

© 2008 Wind River Systems, Inc.

WIND RIVER

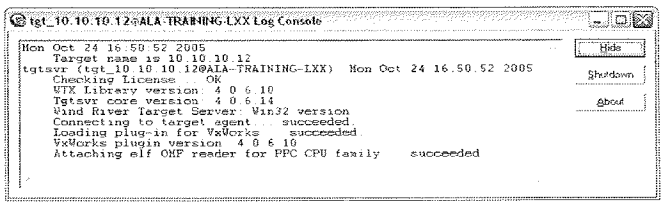
Student Notes

- "Shared" allows you to make your target definition visible to others (other than your user-id)
- Target servers are normally automatically "killed" on a disconnect, "shared" keeps the target server alive for others to use

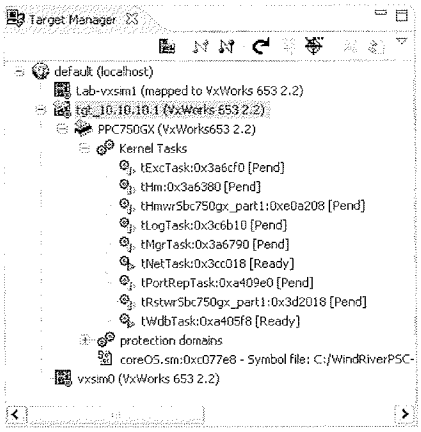
© 2008 Wind River Systems, Inc.

Target Connection Information

- Click on target server icon in taskbar system tray (Windows)



- Expand active session in target manager

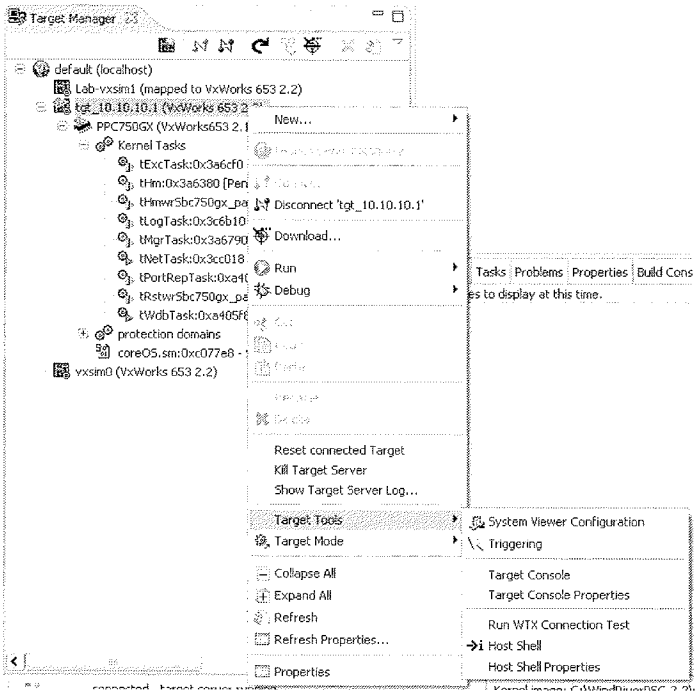


© 2008 Wind River Systems, Inc.
WIND RIVER

Student Notes

Connecting Tools to the Target

- To connect tool to target, select it from the context menu



Slide 52

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Getting Started

Introduction
Getting Help
Setting Up the Host
Booting the Target
Connecting the Host and Target
Simulator

Slide 53

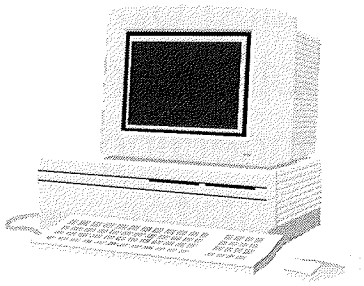
© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

VxWorks Simulator

- Wind River® VxWorks 653 Platform 2.2 includes a target simulator
  - Allows development prior to hardware availability
  - Supports most VxWorks facilities
- Set up development host
  - Define environment variables
  - Network configuration
  - Start the registry, wtxregd



Host (Tools and Target Simulator)

- Start Target Manager and launch target server
- Connect to target from Target Manager

Slide 54

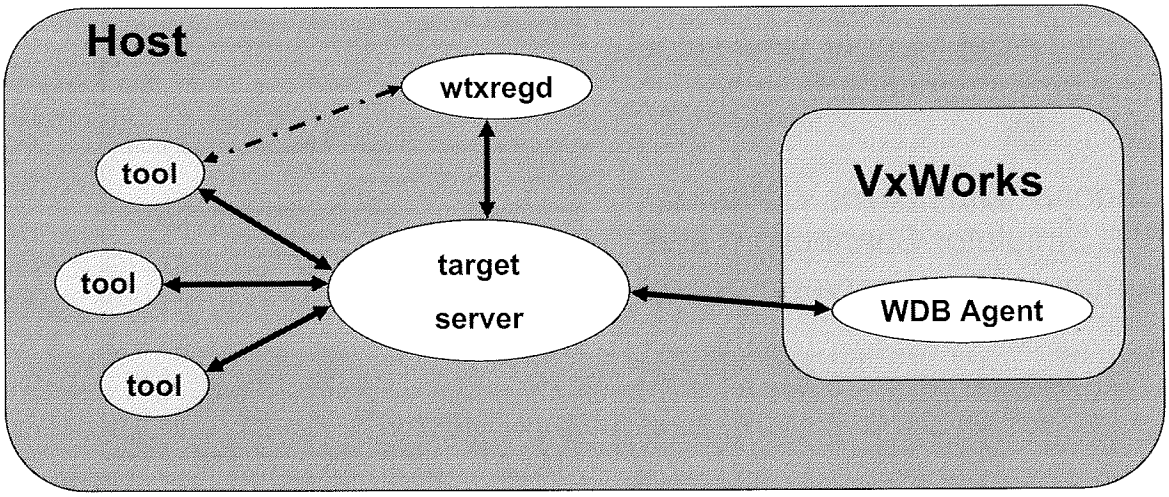
© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- For Windows, Registry startup is selected while installing Workbench.

Tools Communication Architecture - Simulator Target



- VxWorks runs as a process under host OS
- Host tool-target communication is via target server and WDB agent

Slide 55

© 2008 Wind River Systems, Inc.

WIND RIVER

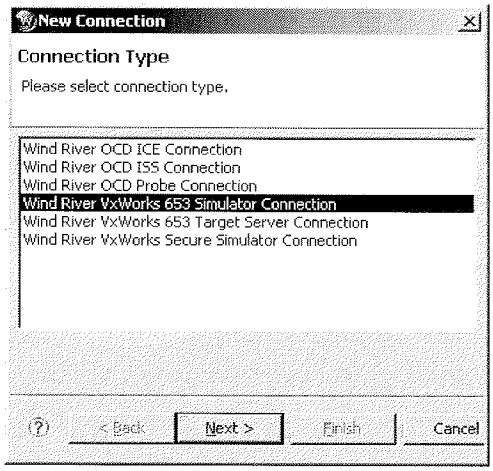
Student Notes

- VxWorks Simulator is a "BSP" for the host OS
- Hardware interrupts are simulated using signals (UNIX) or Windows messages (Windows)

Starting a Simulator Session (Step 1)

- From Target Manger view select new target connection option

- Choose "Wind River VxWorks 653 Simulator Connection" and configure target server properties



Slide 56

© 2008 Wind River Systems, Inc.

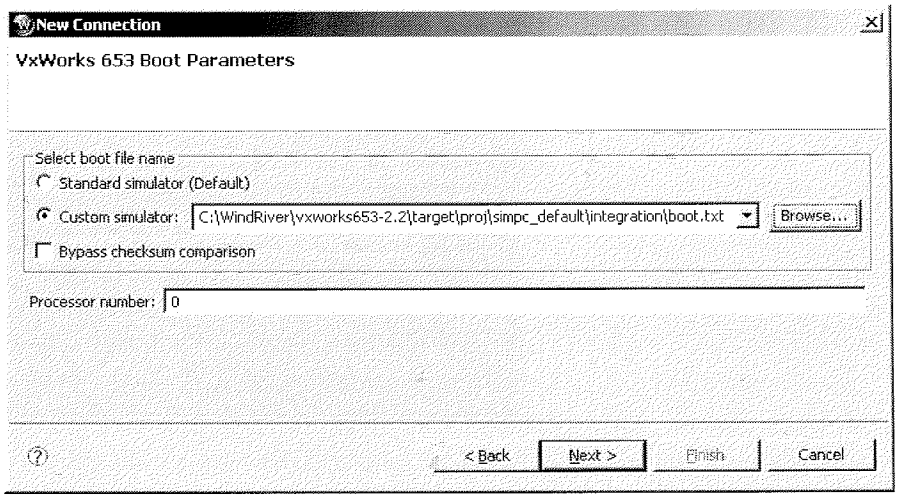
WIND RIVER

Student Notes

- Configuring a simulator connection is a five step process

Starting a Simulator Session (Step 2)

- Choose the simulator file to use
- Configure (optionally) boot parameters



Slide 57

© 2008 Wind River Systems, Inc.

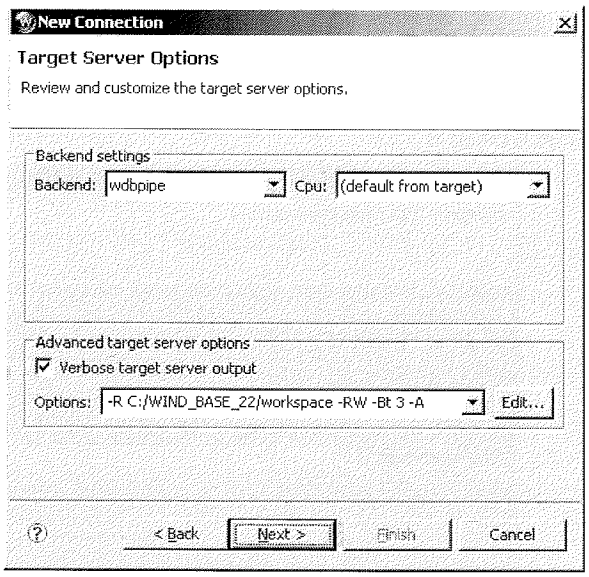
WIND RIVER

Student Notes

- Standard simulator BSP is under ../target/config/simpc directory

Starting a Simulator Session (Step 3)

- Configure options



Slide 58

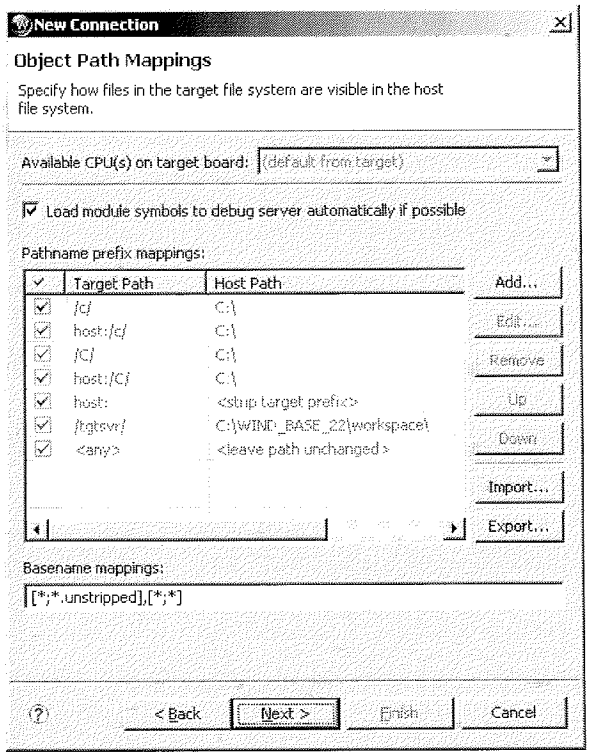
© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Starting a Simulator Session (Step 4)

- Set object path mappings



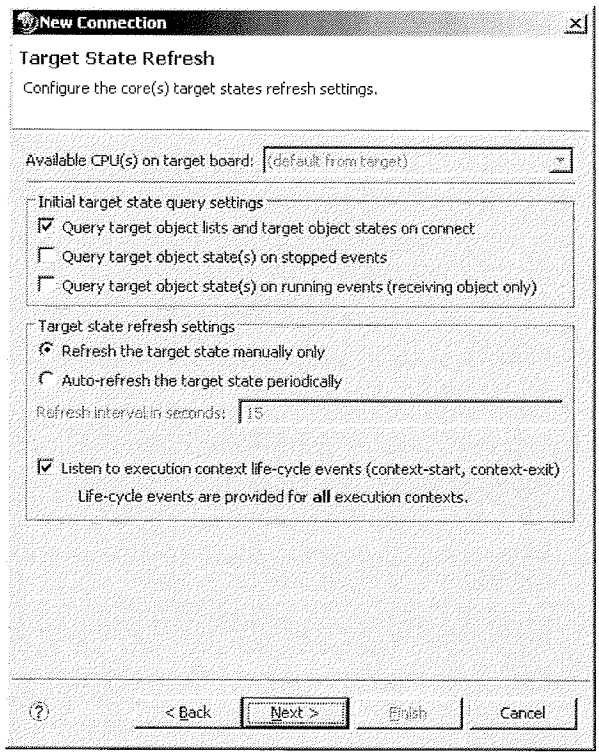
© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Starting a Simulator Session (Step 5)

- Configure the Target Refresh



Slide 60

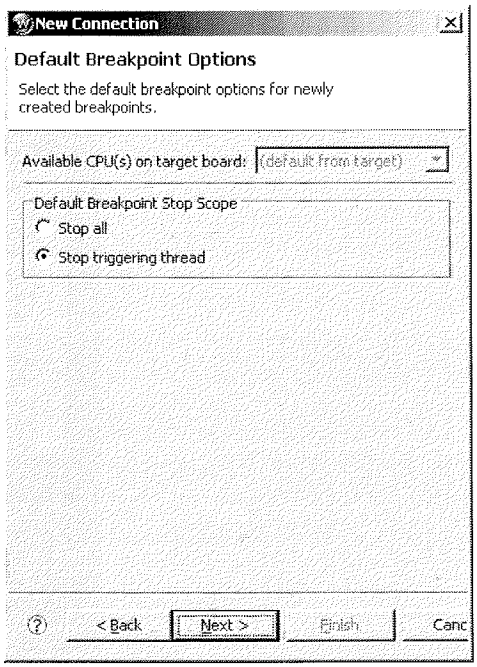
© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Starting a Simulator Session (Step 6)

- Configure the Default Breakpoint options



Slide 61

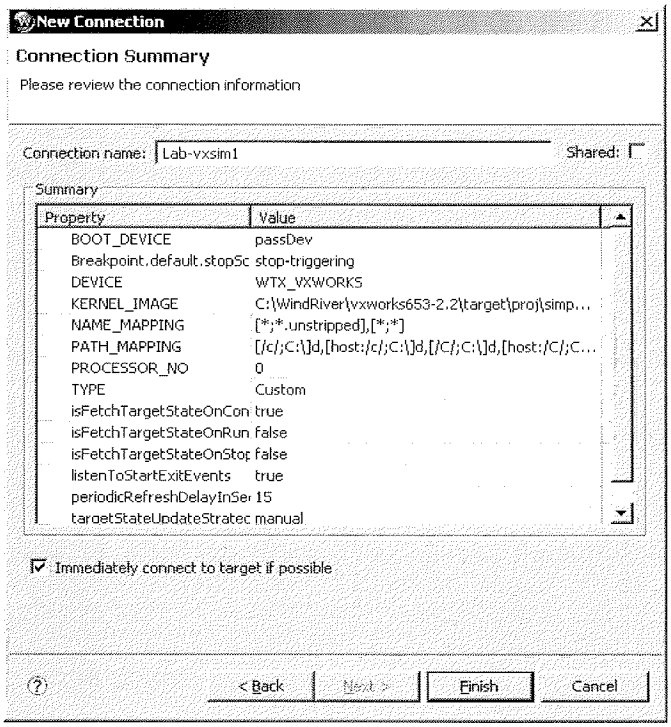
© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Starting a Simulator Session (Step 7)

- Confirm and finish



Slide 62

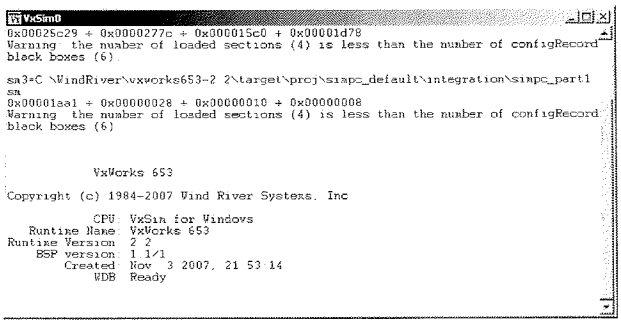
© 2008 Wind River Systems, Inc.

WIND RIVER

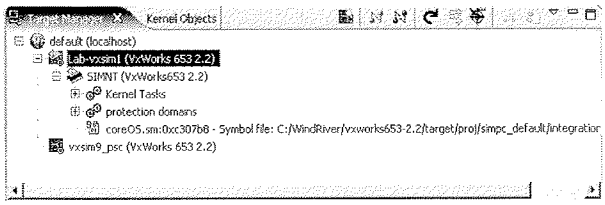
Student Notes

Starting a Simulator Session (Step 7)

- Start the Simulator by selecting the **Connect** button



- A console window for the I/O of your tasks will open, closing this window will terminate the simulator!



Slide 63

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- In the above console window for the simulator, prompt and the banner shown is for the "Kernel Shell"
- When a connection to a simulator is made, target server for it is also started

Questions

1. Wind River Workbench functionalities are by default grouped in different Perspectives. True or False?
2. VxWorks 653 meets ARINC 653 specifications by providing:
   a) Health Monitoring
   b) Partition restart capability
   c) Module restart capability
   d) All of the above
3. RAM payload images are used for development systems where _____ is required.
4. Which development tool is used to access the target interactively?
5. The module OS is not configured to support partition scheduling. True or False?

Slide 64

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

1. True.
2. D
3. restart capability
4. Shell
5. false

Review

In this chapter you learned:

- Wind River® VxWorks 653 Platform 2.2 provides Wind River Workbench 2.6 Development Environment and VxWorks 653 Real-time Operating System
- Workbench provides the following Tools
  - Project Navigator
  - Host Shell
  - Target Shell
  - Debugger
  - System Viewer
  - VxWorks Simulator
  - Triggering
- VxWorks 653
  - ARINC 653 Compliant Operating System

Slide 65

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Review (Continued)

In this chapter you learned:

- Help available:
  - Online documentation
  - Customer Support and Online Support
- Target hardware setup is described in BSP documentation
- Invoke a target server for each target.
- Target server manages:
  - Target resources used by host tools
  - Target information
  - Communication with target
  - Tool-target interactions
- Configure WDB Agent:
  - Debug mode
  - Communication strategy

Slide 66

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Do ćwiczenia

System Concepts

windows.hosted-training.com

8008117001

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Objectives

By the end of this chapter you will be able to:

- Use the virtual memory management
- Understand the MMU page attributes
- Schedule a partition
- Share the code among multiple partitions
- Understand how the Health Management is reporting and recovers the faults that occur at the process, partition or module level

Slide 2

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

System Concepts

Module OS and Partition Scheduling
Partition OS and Process Scheduling
Shared Libraries
ARINC Port Inter-Partition Communications
Health Monitor

Slide 3

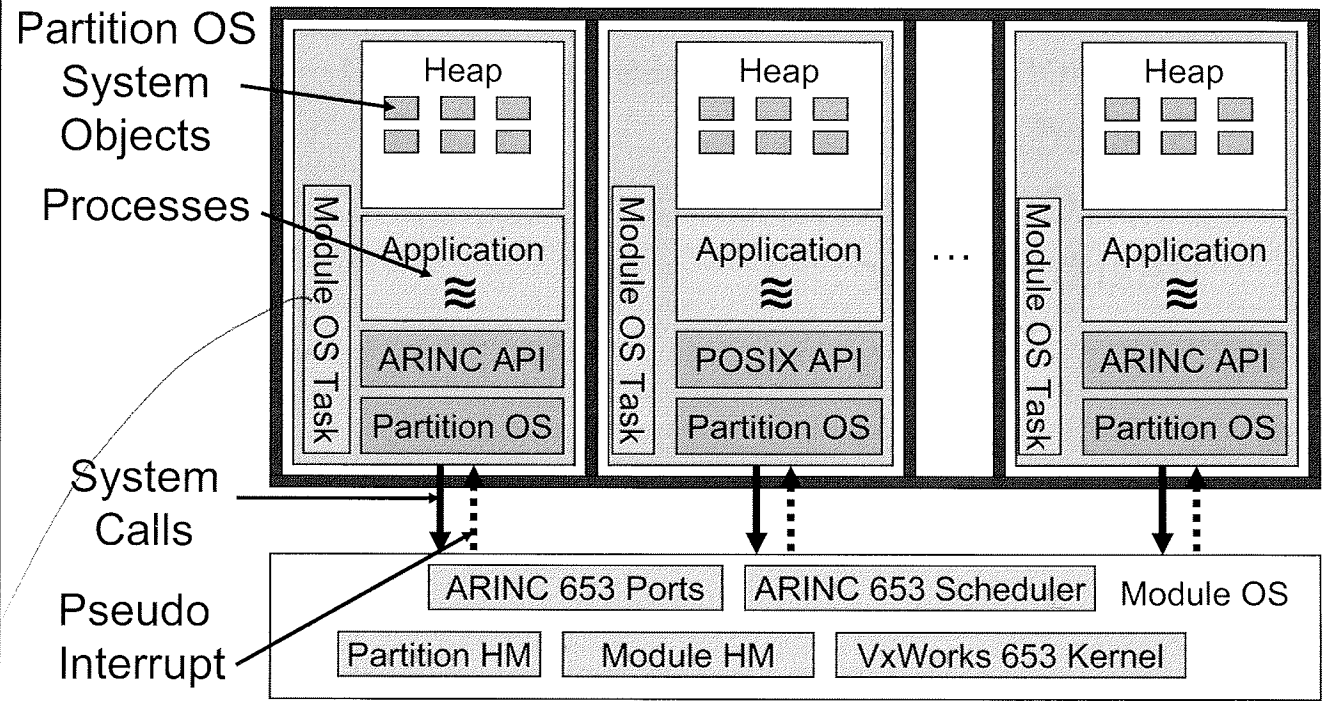
© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Memory

VxWorks 653 Architecture



Slide 4

© 2008 Wind River Systems, Inc.

WIND RIVER

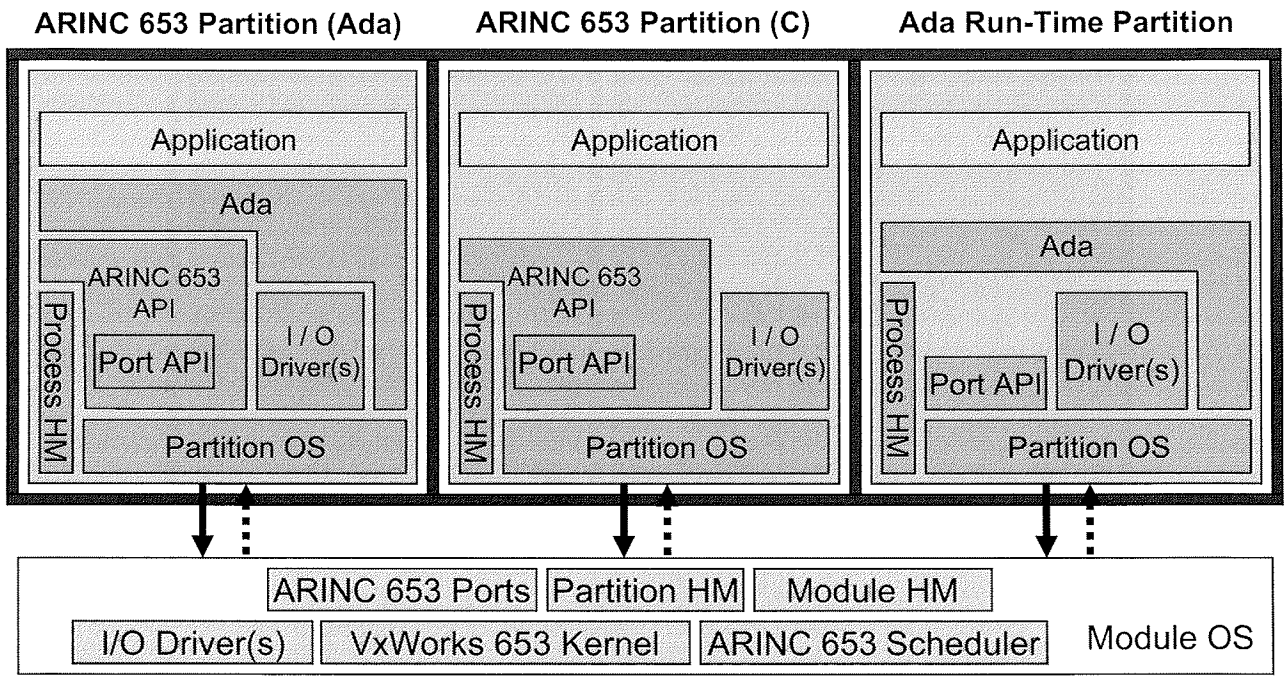
Student Notes

Partition scheduler

Pseudo-interrupts

VxWorks 653 Architecture Details

+POSIX
+VXWORKS



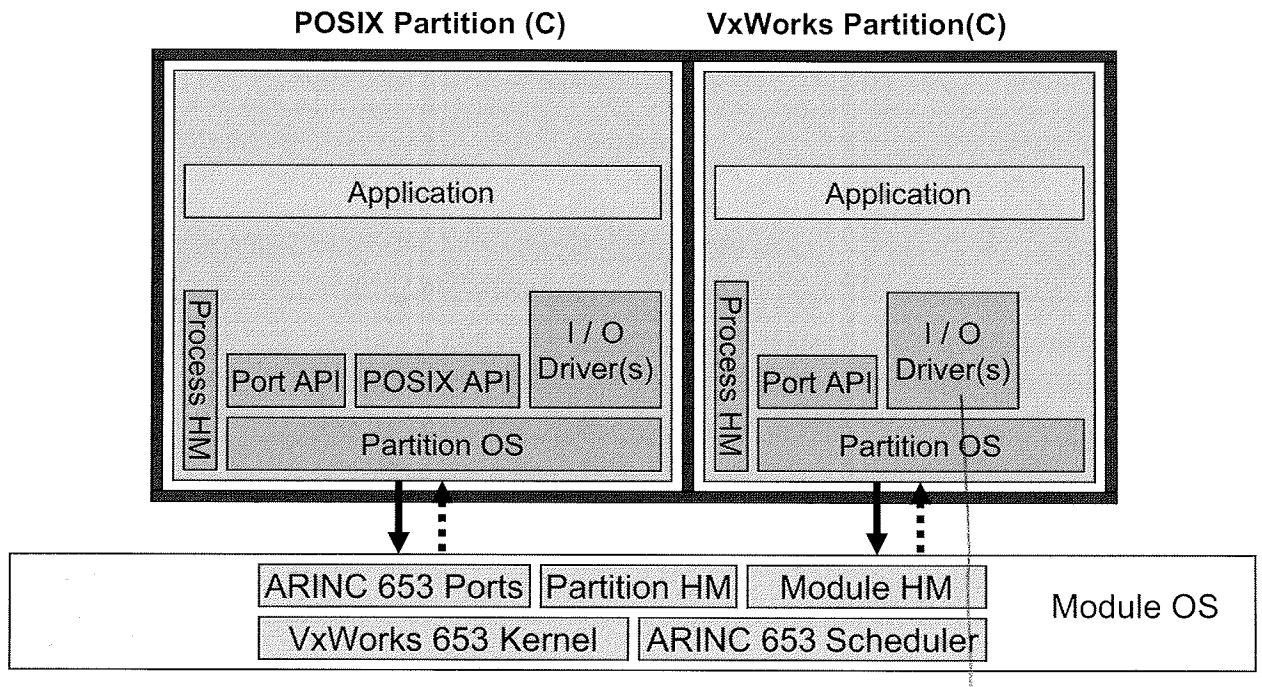
Slide 5

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

VxWorks 653 Architecture Details



Slide 6

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

VxWorks 653 Module OS

- Partition management (Creation, Restart, Scheduling, etc.)
- Partition boundary definition and enforcement (via MMU)
- Trapping and execution of system calls on behalf of the Partition OS
- System resource allocation
- System clock (time since power up)
- Message passing between partitions (IPC)
- Module/Partition level Health Monitoring (HM)
- Loading and containing configuration records
- Debug agent (requested by the host tools)
- Management of tasks that execute the partition OS
- Handling of interrupts (timers) and exceptions

Slide 7

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

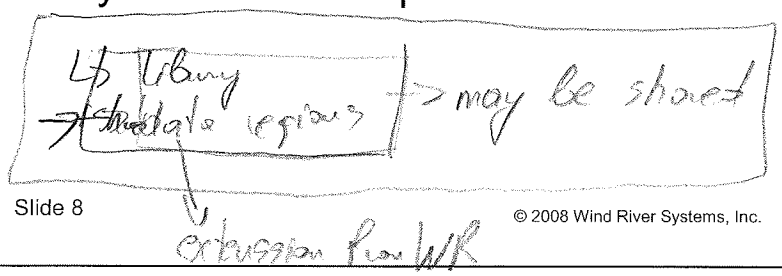
- The ModuleOS is the foundation upon which all of the partition instances are based
- The ModuleOS is the only true operating system involved in VxWorks 653—partition instances are implemented as ModuleOS tasks
- The ModuleOS manages all resources a normal operating system manages:

Protected Memory Model

- VxWorks 653 uses virtual memory management
- Supports multiple address spaces for the system
- Using multiple virtual memory protection domains can
  - Isolate applications from the OS kernel
  - Prevent errant code from corrupting other applications
  - Meet performance requirements of real-time systems

"Nothing is shared..."
...unless explicitly shared

- Systems can be partitioned to achieve greater reliability



Slide 8

© 2008 Wind River Systems, Inc.

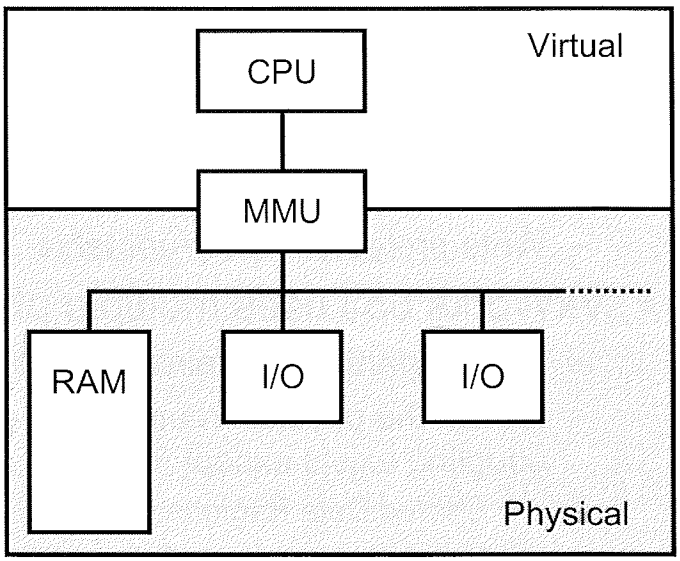
WIND RIVER

Student Notes

- The Module OS uses the MMU to define individual virtual address spaces for each partition
- As each partition task sees only addresses related to the partition it manages, protection of the ModuleOS data structures and other partitions is achieved
- The Module OS and the ARINC 653 specification rigidly define cross partition visibility through a mechanism (the MMU) or a rule (the specification)

Memory Management Unit (MMU)

- MMU hardware
  - Translates virtual to physical addresses
  - Supports definition of access rights such as "unmapped", "read only", "supervisor mode only"
  - Supports definition of multiple virtual memory contexts
- A process attempting to access protected memory
  - Causes the MMU to generate an exception
    - HM policy controls response to error condition



Slide 9

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

VxWorks 653 and the MMU

- VxWorks 653 uses the MMU page attributes to
  - Write protect text and read only data segments for loaded object modules or system images
  - Provide stack overflow detection
  - Protect the Module OS from errant partition code
  - Write protect the interrupt table
- VxWorks 653 uses multiple physical page pools to
  - Protect partitions against each other
  - Isolate the physical memory for each partition so no sharing or coupling exists except for
    - Shared Data Regions (space is shared among attached partitions)
    - Read only sections of shared libraries

Slide 10

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Module OS Responsibilities

- Creates and manages all partitions, shared libraries and shared data regions
- Allocates Module OS portions of ARINC port connections used for inter-partition communications
- Tasks perform Partition and Module level Health Management
- Tasks manage warm and cold restart of partitions
- In development, Module OS provides debug agent support

Slide 11

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Module OS Limitations

- In VxWorks 653 partitions cannot be dynamically created at run time. However, VxWorks 653 supports the concept of an online loaded partition (allowing for the addition of 3rd party applications within their own partition space but space is reserved at build time)
- Object modules cannot be download into a partition dynamically

To be deterministic and certifiable...
Resources must be defined and instantiated at startup since allocations and frees are disabled after startup

Slide 12

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Partition Scheduling

- In Module OS, scheduling unit is a partition not a process within a partition
- Schedule for partitions is predetermined via XML Configuration File
- Module OS tasks associated with a partition execute in partition's time frame
- Time can be scheduled just for Module OS
  - Define a schedule for **SPARE** partition
- Up to 16 separate schedules can be defined
  - Schedules can be switched programmatically (Mode-based scheduling)

Slide 13

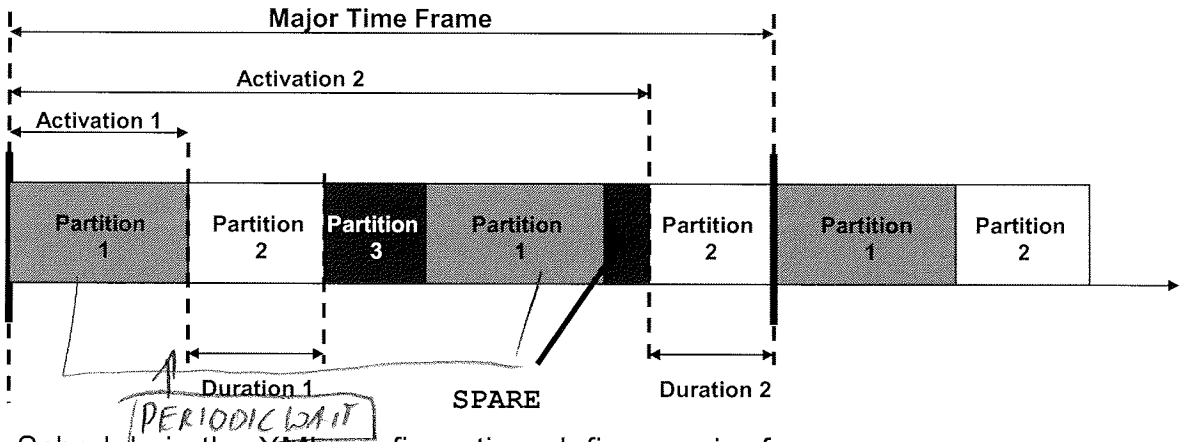
© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- Module OS tasks, not associated with a partition and of sufficient priority can pre-empt a partition task
  - Debug only

Partition Scheduling



- Schedule in the XML configuration, defines major frame
- Each partition's time is referred to as a minor frame
  - Minimum minor frame is .25 milliseconds
  - Partitions do not have a priority
- Partitions can be scheduled in any order and multiple times within a major frame
- Health Monitor monitors for any temporal violations (jitter and deadlines)

Slide 14

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

System Concepts

Module OS and Partition Scheduling
Partition OS and Process Scheduling
Shared Libraries
ARINC Port Inter-Partition Communications
Health Monitor

Slide 15

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Partition OS Overview

- Based on VxWorks 5.5 but runs at user-level not supervisor
- Provides application level OS capabilities
  - Threading (preemptive, priority based)
  - System objects (Semaphores, Messages Queue, events, ...)
  - Heap Memory Management (***malloc***) if not disabled
  - Runtime libraries (ANSI, math, stdio, APEX, POSIX)
    - APEX and POSIX are mutually exclusive in a partition
  - Software watch dog timer
  - Process Health Monitoring
  - I/O layer and device drivers
- All memory allocations are made within partition space
  - Insures space separation between partitions/applications
- All allocations are made at initialization
- Interrupts and exceptions are synthesized by Module OS, using SW signaling mechanism

Slide 16

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- The Partition OS environment is provided by a ModuleOS task running code based on VxWorks 5.5.
  - This environment provides all normal operating facilities in several APIs, including:
    - Thread creation and scheduling
    - Kernel primitives (semaphores, message queues and the like)
- All memory requests from processes within a partition are satisfied by memory pre-allocated for the partition.
- Only timer (i.e., system clock) interrupts are reported to the partition from the Module
  - Exceptions generated by processes within a partition can be managed wholly from within the partition.

Partition OS Memory Management

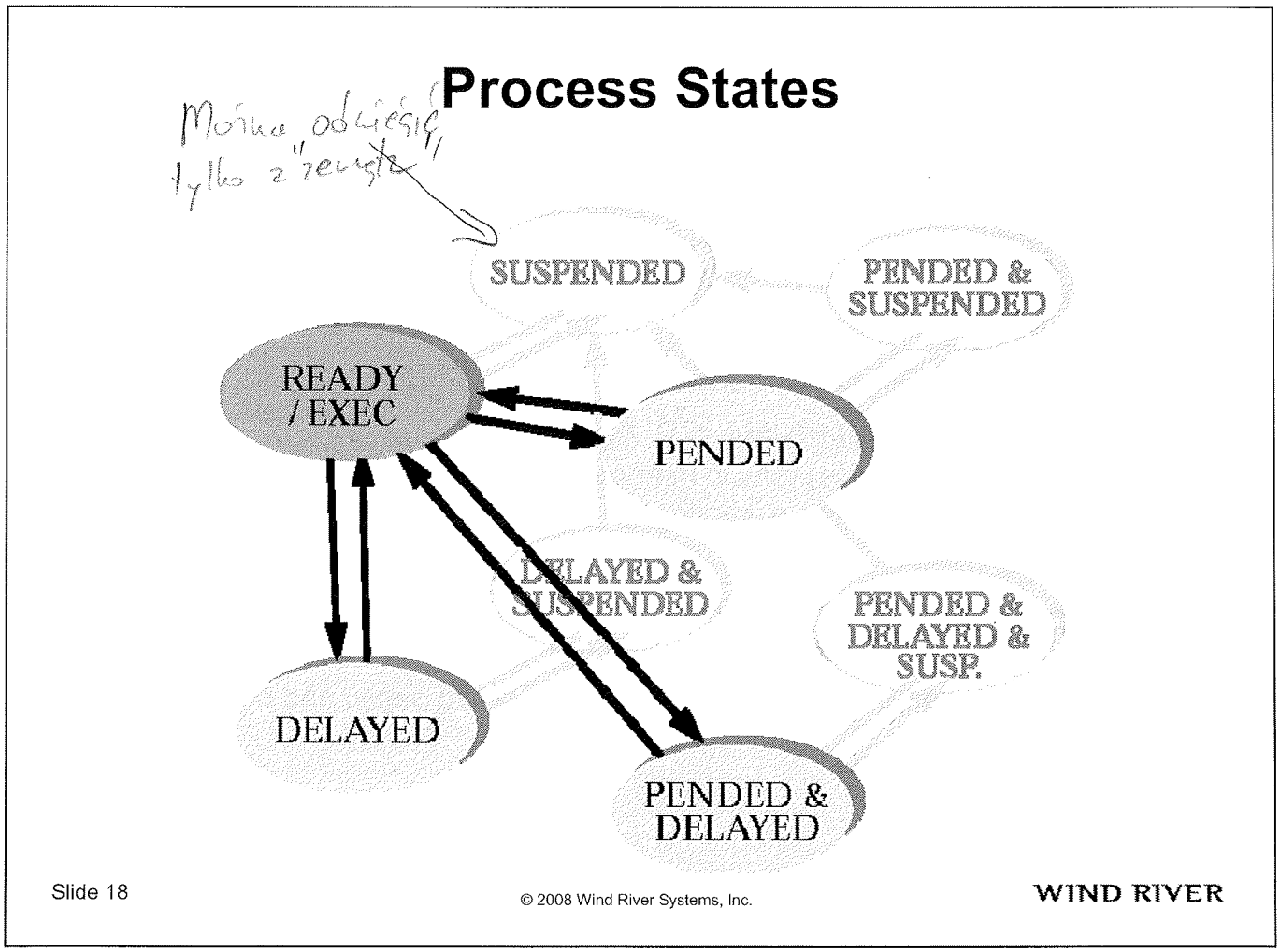
- Stack overflow protection implemented with use of guard page(s) at top of the process stack
- Number of guard pages used is set in the XML configuration and is on a partition basis
  - Default for the PowerPC is one
- Guard page(s) are supervisor R/W access only
  - Since thread (process) runs in user mode, any access to guard page area results in a data access exception
- Limitations
  - Stack underflow is not detected
  - If an overflow occurs during a system call, system will execute a cold restart
  - Huge stack frame overflows may step over guard pages and not be caught

Slide 17

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

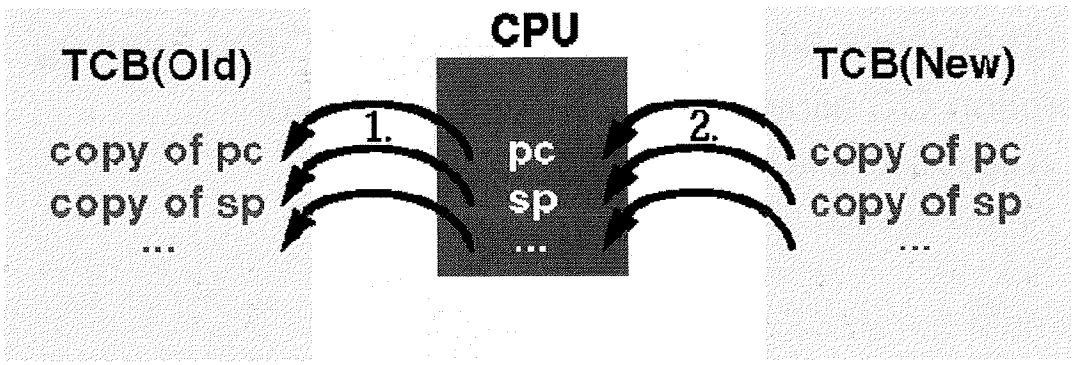


Student Notes

- Non-suspended processes may be in one of four states
  - Ready
    - processes that are eligible to execute
    - The actually executing process has the highest priority among all ready processes
    - The other processes in the ready queue wait for the CPU
  - Delayed
    - processes that are waiting for an amount of time to pass
  - Pended
    - processes that are waiting for an object event to become available
  - Pended and Delayed
    - processes waiting for a object event, but with a timeout specified
- Processes may, additionally, be suspended
  - Suspended processes will not execute
  - processes that are delayed or pended can be suspended
  - When resumed they will return to the appropriate state

Context Switch

- A ***context switch*** occurs when one process stops executing and a new process starts



1. Context of currently executing process is saved into its TCB
2. Context of next process is copied from its TCB into CPU

- An OS call or pseudo-interrupt can cause a context switch

© 2008 Wind River Systems, Inc.

Student Notes

© 2008 Wind River Systems, Inc.

Multiprocessing

- Separate processes are created to perform different system requirements
  - For example, data acquisition and data computation
- Each process alternates between "ready" and "waiting"
- Partition OS allows for a process to wait for
  - A specified time delay (Delay)
  - An event such as an pseudo-interrupt (Pend)

Slide 20

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- Multiprocessing advantages
  - Using different wait commands for each process permits manipulating the components at different rates
  - Using a priority-based scheduler permits running the most important process in the case when two processes are in the "ready" state at the same time
  - Using a preemptive scheduler permits an important task which enters the "ready" state to start running right away, preempting the less important task previously running
  - Permits easier addition of new components

Multiprocessing Kernel

- The ***wind*** kernel is that part of the VxWorks Module OS that directly manages processes
- Allocates CPU to processes according to VxWorks scheduling algorithm
- Uses **Task Control Blocks** (**TCB**s) to keep track of processes
  - One per process
  - Declared as ***WIND_TCB*** data structure in **taskLib.h**
  - O.S. control information — state, process priority, delay timer, breakpoint list, error status, I/O redirections, etc.
  - CPU Context Information — PC, SP, CPU registers, FPU registers

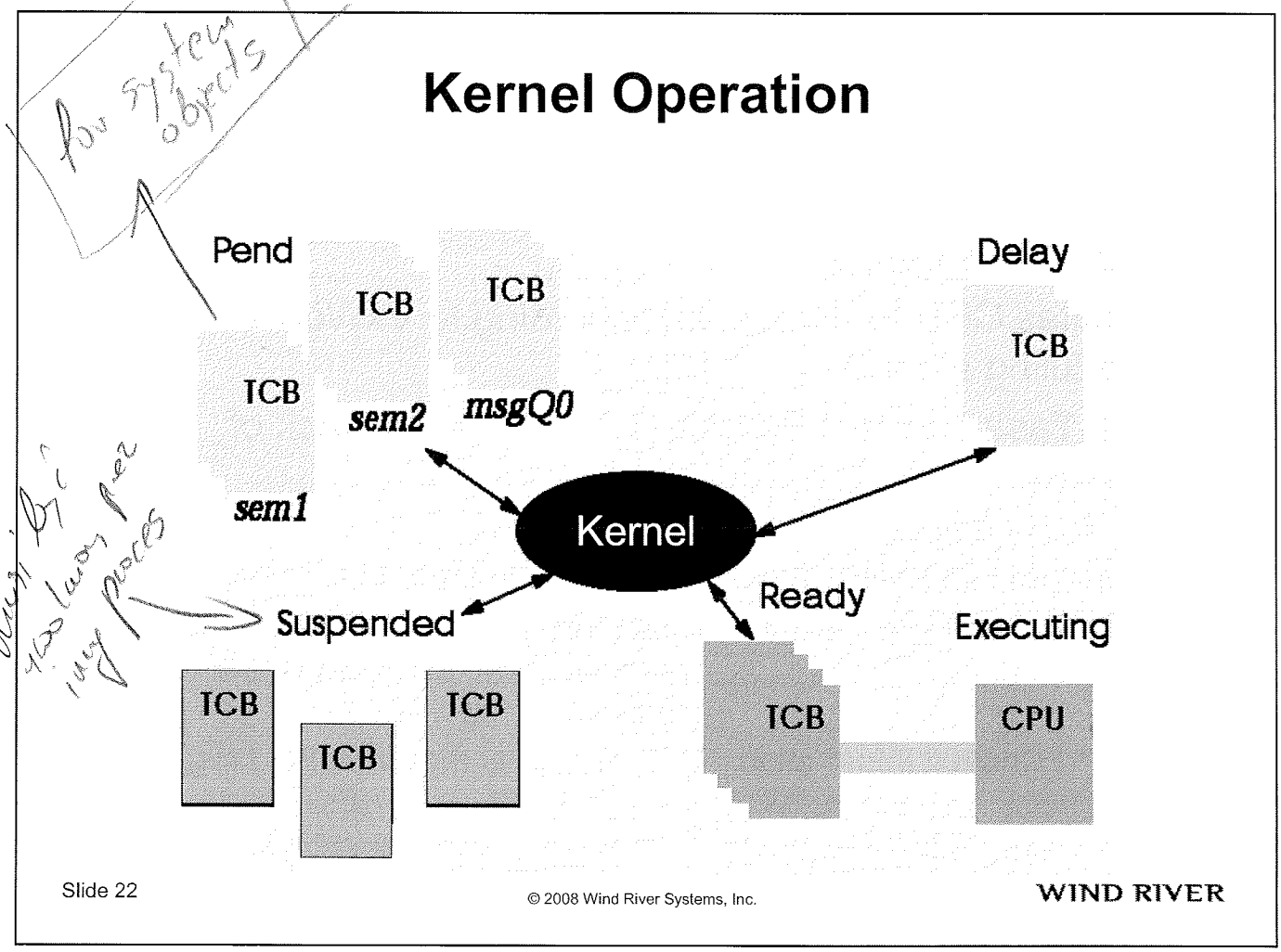
Slide 21

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- The TCB actually contains a pointer to the floating point register set storage location for those processes which use floating point
- There are a few spare fields at the end of the TCB which may be used by an application to store additional information specific to a process
  - See the declaration in **taskLib.h**



Student Notes

- The kernel manages the processes, moving them from state to state based on kernel operations invoked
- A separate pend queue exists for each kernel object (semaphore, etc..)
  - For example, an event could be a particular semaphore's being given, or a message arriving across a particular message queue
- A process in the PENDED + DELAYED state is linked into both a pend queue and the delay queue
  - The link nodes are part of the TCB
- The ready queue is not a simple linked list
  - It is a bit-mapped priority queue, so that insertion and deletion are done in constant time
- Whenever any of the above queues is manipulated, kernel state is entered
  - Upon exiting kernel state, code executes to check whether a reschedule (context switch) should occur

© 2008 Wind River Systems, Inc.

Priority Scheduling

- Different application jobs may have different precedence, which should be observed when allocating CPU
- Preemptive scheduling is based on process priorities chosen to reflect job precedence
- Highest priority process ready to run (not pended or delayed) is allocated the CPU
- Rescheduling can occur at any time as a result of a kernel call or on a process exit
- Context switch is not delayed until next system clock tick

Slide 23

© 2008 Wind River Systems, Inc.

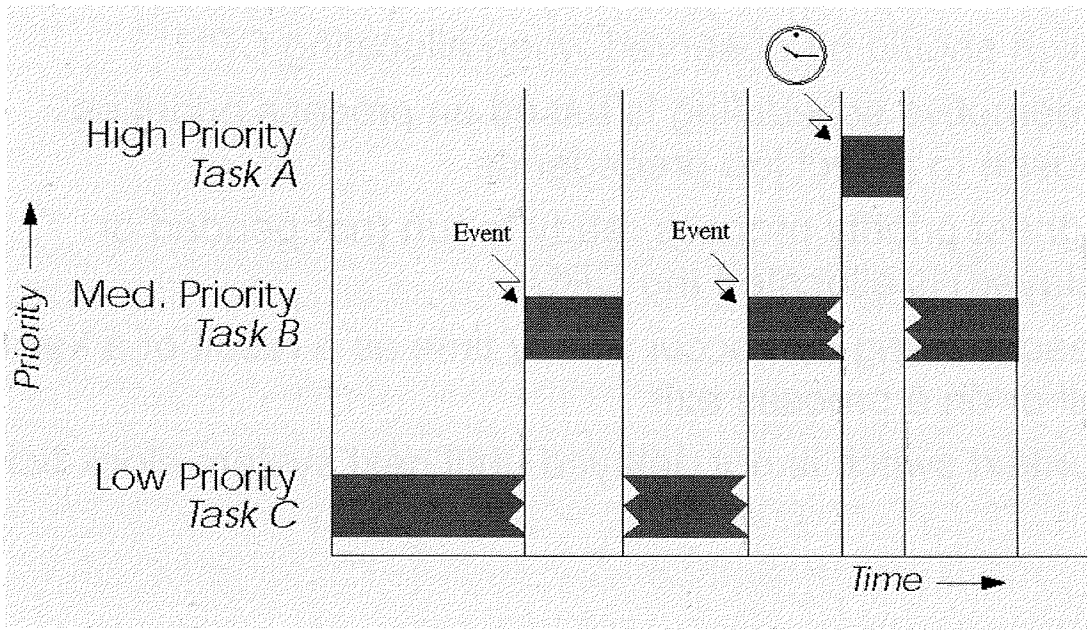
WIND RIVER

Student Notes

- Reschedules do not occur merely because a process has been executing for a long time
  - The exception is if round-robin scheduling is enabled
- When an interrupt causes a reschedule, it is because the interrupt service routine called a kernel function
  - A special case of this is the processing of timeouts and delays which occurs on every system clock tick interrupt
- The precedence of a job may be more determined by timing requirements than by the job's "intrinsic importance," which is often nebulous

© 2008 Wind River Systems, Inc.

Priority Based Preemption



Equal priority processes won't preempt each other (by default)

Slide 24

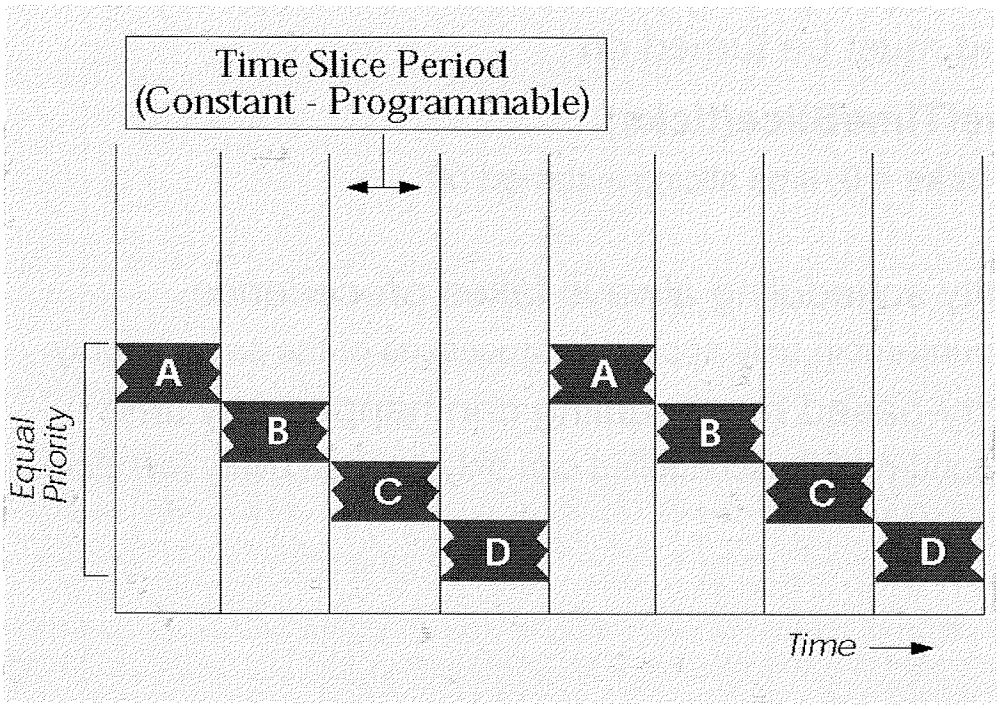
© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- Rescheduling occurs if
  - The currently running process pends or delays
  - A process of higher priority becomes ready
- When a process of higher priority takes the CPU away from the current process, it is preempting that running process
- If process B never blocks or pends, then process C will never run

Round-Robin Scheduling



Slide 25

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- Processes A to D (same priority) share the CPU on an equal basis
- Round-robin mode makes all processes of the same priority share the CPU fairly

Time-Slicing

- Allows equal priority processes to preempt each other, time slicing must be turned on

  kernelTimeSlice(ticks)
  - If **ticks** = 0, time slicing is turned off

- Priority scheduling always takes precedence
  - Round-robin only applies to processes of the same priority
- Priority-based rescheduling can happen any time
  - Round-robin rescheduling can only happen every few clock ticks

Slide 26

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- When time-slicing is turned on, it is turned on globally for processes of all priorities
  - If the processes of priority 100 use round robin to share the CPU equally amongst themselves, then the processes of priority 120 must also use round robin to share the CPU equally amongst themselves
- Ticks is specified in system clock ticks
  - To get the clock frequency (in Hz),

    → clockRate = sysClkRateGet( )

System Concepts

Module OS and Partition Scheduling
Partition OS and Process Scheduling
Shared Libraries
ARINC Port Inter-Partition Communications
Health Monitor

Slide 27

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Shared Libraries

- Allow sharing of code (text) among multiple partitions
- Configured like any other VxWorks 653 system feature: through an XML specification
- VxWorks 653 must contain a system shared library (Partition OS)
  - WR implementation requirement
- Partition OS provides access to Module OS for partitions requiring Module OS services
- Use XML to specify which libraries are accessed by each partition
- Partitions share read-only sections (text, rodata) of library but have private copies of writable data

Slide 28

© 2008 Wind River Systems, Inc.

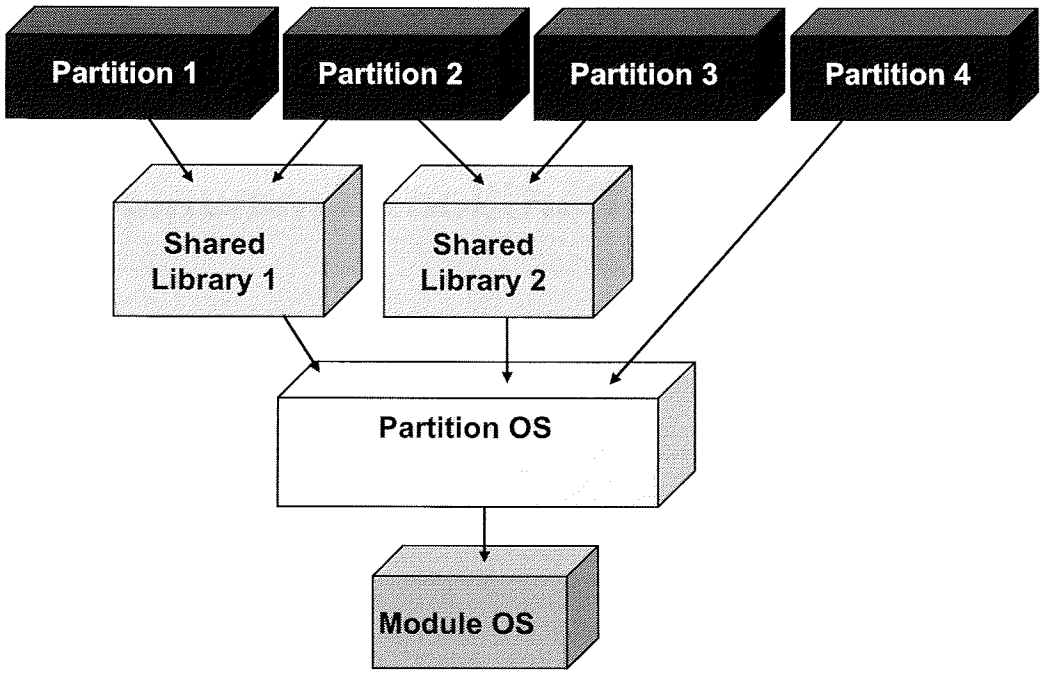
WIND RIVER

Student Notes

- My-ssl.xml

partition OS -> shared library

Shared Libraries Layout



Slide 29

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

System Concepts

Module OS and Partition Scheduling
Partition OS and Process Scheduling
Shared Libraries
ARINC Port Inter-Partition Communications
Health Monitor

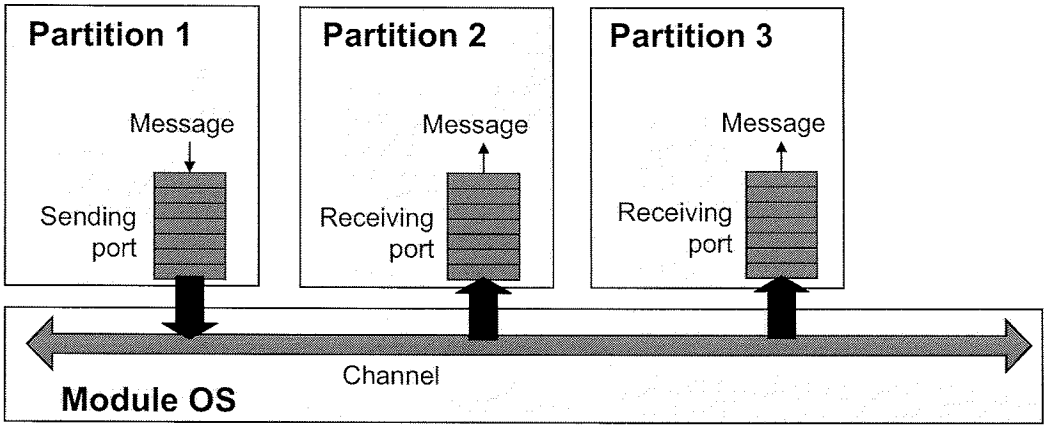
Slide 30

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Inter-Partition Communication Overview



- Inter-partition communication is made of Messages, Ports and Channels
- This is the only inter-partition communication method in ARINC spec
- Partition with partition, partition with Module, partition with other module
- Message is sent from a single source port to one or more destination ports
- Configuration is done in the XML Configuration File

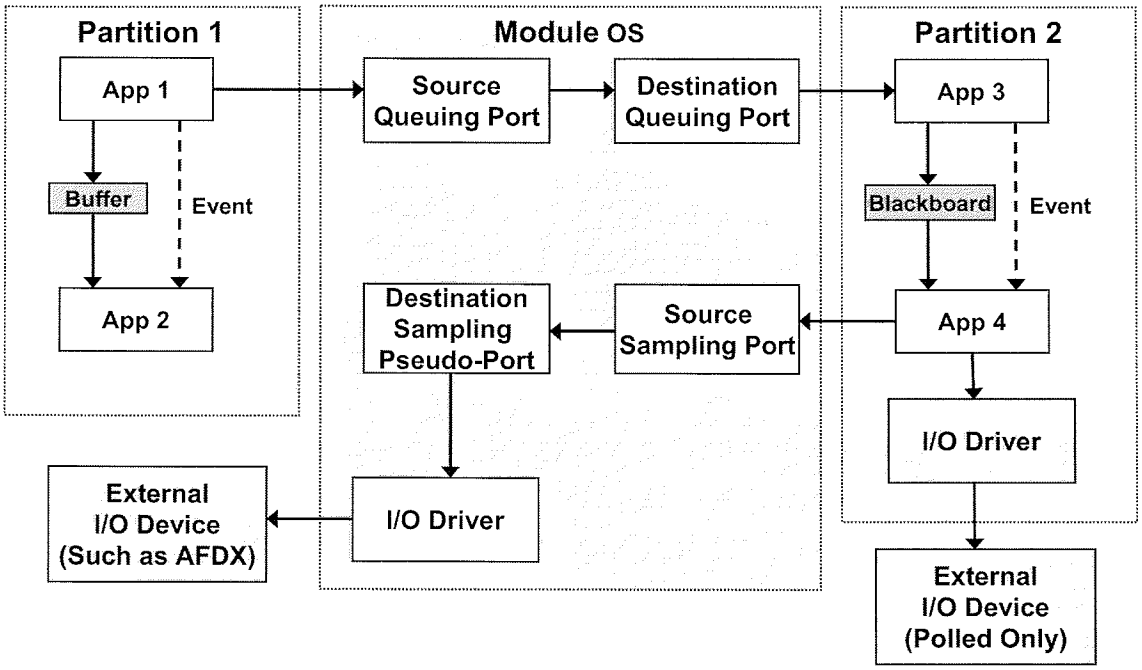
Slide 31

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

ARINC 653 Example Data Flows



Slide 32

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- Graphic shows example data and control flows, intra and inter-partition
- Data moved within a partition (intra-partition) from on process to another via buffers or blackboards
- Intra-partition (inter-process) synchronization via Events or Semaphores
- Data moved inter-partition via source and destination ports (queuing or sampling)
- Data can be moved from a partition to an I/O device via ports
- IO devices can also be mapped into partition space. Only polled mode is allowed since App4 is in user mode

Communication Modes (2 Available)

- Queuing Mode
  - Ports behave like message queues
  - Messages are queued in FIFO order
  - A receiver waits if its destination queue is empty
  - If source port is configured as "SENDER_BLOCK", the sender waits if its source queue is full
- Sampling Mode
  - Ports behave like shared memory with mutual exclusion protection
  - A reader never waits to read a message
  - Writer always overwrites previous message
  - Data coherency is guaranteed

Slide 33

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- RECEIVER_DISCARD – vxWorks 653 extension
  - If one of the destination ports of the channel is full, the source port discards the message for this port
  - The message could be lost if all destination ports are full

Inter-Partition Communication

- XML configuration defines channel(s) that are created within Module OS at startup
- A channel defines a logical connection between a source and one or more destination ports
  - Ports are well defined access points
- Partitions then attach to these channels during their respective startups
- Ensures that all system resources are allocated and then static at system run time
- Operation of queuing ports is based on VxWorks message queue objects

Slide 34

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- The ARINC 653 Specification states a channel contains one and only one Source and one and only one Destination
  - The ability to have multiple Destination Ports connected to a channel is an extension of VxWorks 653 ARINC 653
- A potential problem may occur by having multiple destination ports on a channel
  - Depending on the port configuration and usage, one destination port can cause the entire channel to block
    - This allows one partition to affect the execution of another partition, violating ARINC 653 temporal partitioning rules.

System Concepts

Module OS and Partition Scheduling
Partition OS and Task Scheduling
Shared Libraries
ARINC Port Inter-Partition Communications
Health Monitor

Slide 35

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Health Monitor (HM)

- Framework provided in VxWorks 653 for reporting and recovery of faults that can occur at process, partition or module level (also used for reporting messages)
- Framework is hierarchical -- Errors occurring at one level that cannot be handled are propagated up to next level
- Process level HM is contained within a partition
  - Developer's responsibility to create process level error handler and to register it
  - Optional part of the HM
- Partition and module level HM reside within Module OS and are table driven
  - Defined in XML configuration by system integrator
  - Alarm codes are mapped to specific handlers

Slide 36

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

HMTypes.h -> definitions formerly HM

usr/HM.c

fm/src/os/hmdefaultHHandlers.c

(HmPolicyHandlers.c)

(comps -> components (for platform provider)

Health Monitoring

- Provides a framework to raise and handle alarms in a system
- Functions at 3 levels
  - Process
  - Partition
  - Module
- Table-driven fault responses and recovery actions occur at partition and module level
- Application driven responses and recovery actions occur at process level

Slide 37

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

HM Levels

- Process HM
  - Also known as the error handler process, it is present within a partition for the handling of process related errors and events
  - Handler is installed (by application) by calling ***hmErrorHandlerCreate( )*** or ***CREATE_ERROR_HANDLER( )***
- Partition HM
  - Present in parallel to Partition OS for handling of partition errors and events that may affect operation of Partition OS within the partition
  - Memory and task context belong to Module OS but scheduling occurs within partition window
- Module HM
  - Present in parallel to all partitions and hence all partition HMs
  - Not part of any partition window and has priority to execute over all partitions, partition HMs and all other Module OS tasks
- Both Partition HM and Module HM are table-driven and directly relate code with an appropriate handler

Slide 38

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Health Monitoring - Notification

- Partition or module level handlers can communicate information to other partitions by notifying them of given events
  - For instance, one partition handler can tell another about an event that caused it to restart the partition
- Notification can be configured such that a partition will only accept notification of certain events and from certain partitions
  - All partitions will accept notification from a module handler unless notification feature is not enabled
- Notification is configured and enabled by system integrator on a per partition basis

Slide 39

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

HM Definitions

- Event
  - Can be either an Alarm or a Message
  - An event is injected by calling
    - ***HM_EVENT_INJECT( )***, or
    - ARINC 653 API
      - ***RAISE_APPLICATION_ERROR( )***, or
      - ***REPORT_APPLICATION_MESSAGE( )***
- Alarm
  - A fault occurring in a system that needs attention from system
    - Faults include hardware-generated exceptions, error paths in the code, thresholds crossed, etc.
- Message
  - An event that has its code parameter set to **HM_MSG**

Slide 40

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Alarms and Messages

- Alarms are generated by the code that detects fault enabling application to provide some recovery mechanism
- Alarms are logged only if a handler is assigned or auto logging is on
- Messages sent from within a partition are handled by partition HM
- Messages sent from outside the partition are handled by module HM
- A default handler is provided to log message into either partition or module HM log depending upon level from which it was injected
  - System integrator is free to replace these default handlers by modifying XML configuration

Slide 41

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Questions

1. The VxWorks Module OS manages:
   a. System resource allocation
   b. The debug agent
   c. Partition boundary definition and enforcement
   d. All of the above
2. What does a process attempting to access protected memory cause the MMU to generate ?
3. In VxWorks 653, partitions cannot be dynamically created at run time. True or False ?
4. Within a major frame, partitions can be scheduled:
   a. Multiple times
   b. In a fixed order
   c. In any order
   d. All of the above
5. All memory allocations in a Partition OS are made at initialization. True or False ?

Slide 42

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

1. D
2. Exception
3. True
4. D
5. True, only if configured that way

malloc() može biti, dopušteno za zauzimanje pri startupu.

Review

In this chapter you learned that:

- VxWorks 653 uses a virtual memory management that supports multiple address spaces for the system
- VxWorks 653 uses the MMU page attributes to protect the Module OS from errant partition code
- Module OS creates and manages all partitions, shared libraries and shared data regions
- In the Module OS, the scheduling unit is a partition not a task and the partition schedule is predetermined in the XML configuration

Slide 43

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Review (Continued)

In this chapter you learned that:

- Partition OS provides application level OS capabilities
- Shared libraries allow sharing of code among multiple partitions (VxWorks 653 must contain a Partition OS)
- Inter-partition communication uses messages, ports and channels
- Health Management is a framework for the reporting and recovery of faults that can occur at the process, partition or module level

Slide 44

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Configuration and Build Partitioning

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Objectives

By the end of this chapter you will be able to:

- Understand the basic principles of the configuration and development processes
- Understand the reference process
- Understand the basic XML schema
- Learn to build:
  - Module OS
  - ACE
  - Partition OS
  - Applications
  - Configuration Record
  - Payloads

Slide 2

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Configuration and Build Partitioning

Configuration Overview

XML Schema

Configuration and Build Partitioning

- MOS and ACE
- Partitions
- Integration

Slide 3

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Lessons Learned

- IMA systems are *extremely* complex:
  - Large number of applications: 10+
  - Large application: 2,000,000+ lines of code, 4-8 MBytes
  - Large configuration data: 40,000+ configuration entries
- Complexity must be *managed* to be successful
  - Role and responsibility have to be clearly defined
  - Roles and activities have to be decoupled
- Development cycles are *shorter* and shorter
- Cost of Change must be very *low*
  - Introducing a change *must* have a low impact even during the certification cycle

- *Solution*: Configuration & Build Partitioning based on DO-297/ED-~~12B~~ 124

Slide 4

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

So what is DO-297/ED-12B?

"Integrated Modular Avionics (IMA) Development Guidance and Certification Considerations"

- *Purpose:*

"..provides guidance for IMA developers, integrators, applicants, and those involved in the approval and continued airworthiness of IMA systems. It provides specific guidance for the assurance of IMA systems as differentiated from traditional federated avionics"

- Results of joint US/EU Study *RTCA SC-200* and *EUROCAE WG-60*
- Defines roles and responsibilities – Certification applicant, Systems Integrator, Platform Provider, Application Developer
- References RTCA DO-178B (EUROCAE ED-12B) and ARINC 653

Slide 5

© 2008 Wind River Systems, Inc.

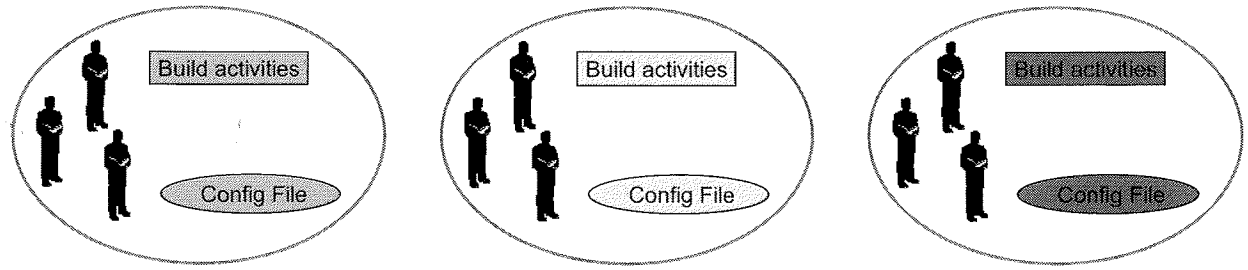
WIND RIVER

Student Notes

- RTCA - Radio Technical Commission for Aeronautics

DO-297/ED-12B Stakeholders

- *Separate* and *organize* configuration data and build activities per IMA roles:
  - *System Integrator* (SI) ,
  - *Platform Provider* (PP) and
  - *Application Developers* (AP)
- Each *role* has its *own* configuration data and set of build activities
- Each activity is *independent* of every other



Slide 6

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

DO-297/ED-12B Certification Stakeholders

Certification Authority (FAA, EASA, JAA)
- Organization that grants approval on behalf of the state(s) responsible for the aircraft/engine certification

Certification Applicant
- Responsible for demonstrating compliance to applicable aviation regulations
- Seeking TC, Amended TC, Supplemental TC or Amended STC

System Integrator
- Integrating the "platform" and "applications" to produce "IMA System"
- System Configuration, Resource allocation, IMA V&V

Platform Provider
- Provide processing hardware and software resources (including the core software)
- Specify interfaces, shared resources, configuration tables
- Platform V&V

Application Developer
- Develops "Hosted" applications and verifies on "platform"
- Specifies external interfaces and resource requirements of application

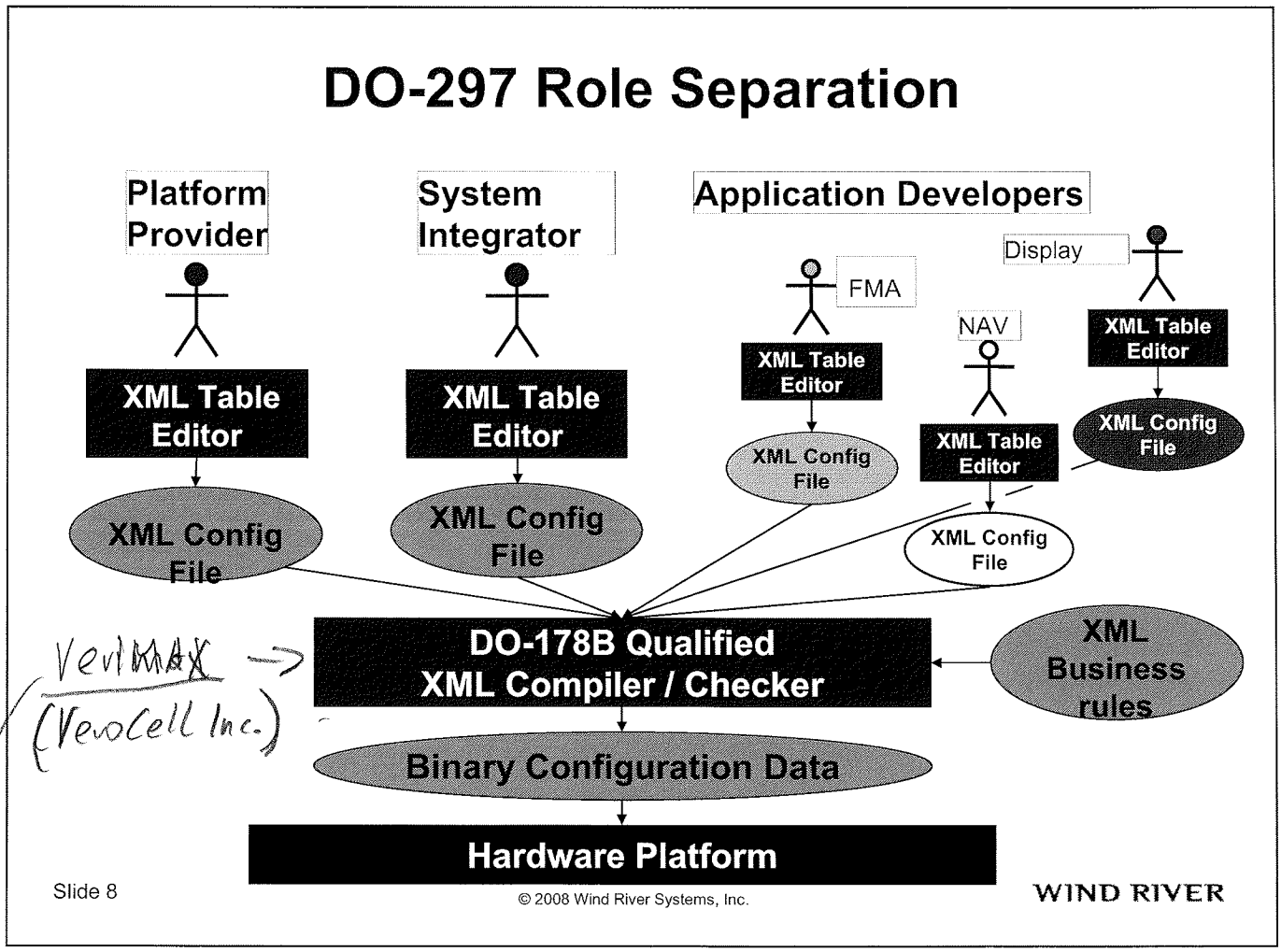
Slide 7

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- TC – Type Certificate STC – Supplemental Type Certificate Mention Certification Authority
- DO-297 recommends
  - Assignment of roles and responsibilities
  - Coordinate with Certification Authority as early as possible
  - Clear and well-defined communications paths
- EASA – European Aviation Safety Agency
- JAA – Joint Aviation Authorities
  - EASA will take over full responsibilities from JAA beginning in 2008
- DO 254 discusses the Validation and Verification Process



VerifMAX (VeroCell Inc.)

Student Notes

can be used for certification

Certification of an IMA System

From DO-297 :

"Six tasks define the incremental acceptance of IMA systems in the certification process:"

- Task 1: Module acceptance
- Task 2: Application software or hardware acceptance
- Task 3: IMA system acceptance
- Task 4: Aircraft integration of IMA system – including Validation and Verification (V&V)
- Task 5: *Change* of modules or applications
- Task 6: *Reuse* of modules or applications

Key implementation and certification challenges:-

- How to *change* application or configuration entities without affecting the entire system?
  - Without requiring re-testing or re-certification of other independent entities
- How to *reuse* applications from one IMA project on the next IMA project?
  - Without having to re-write and re-test the entire application

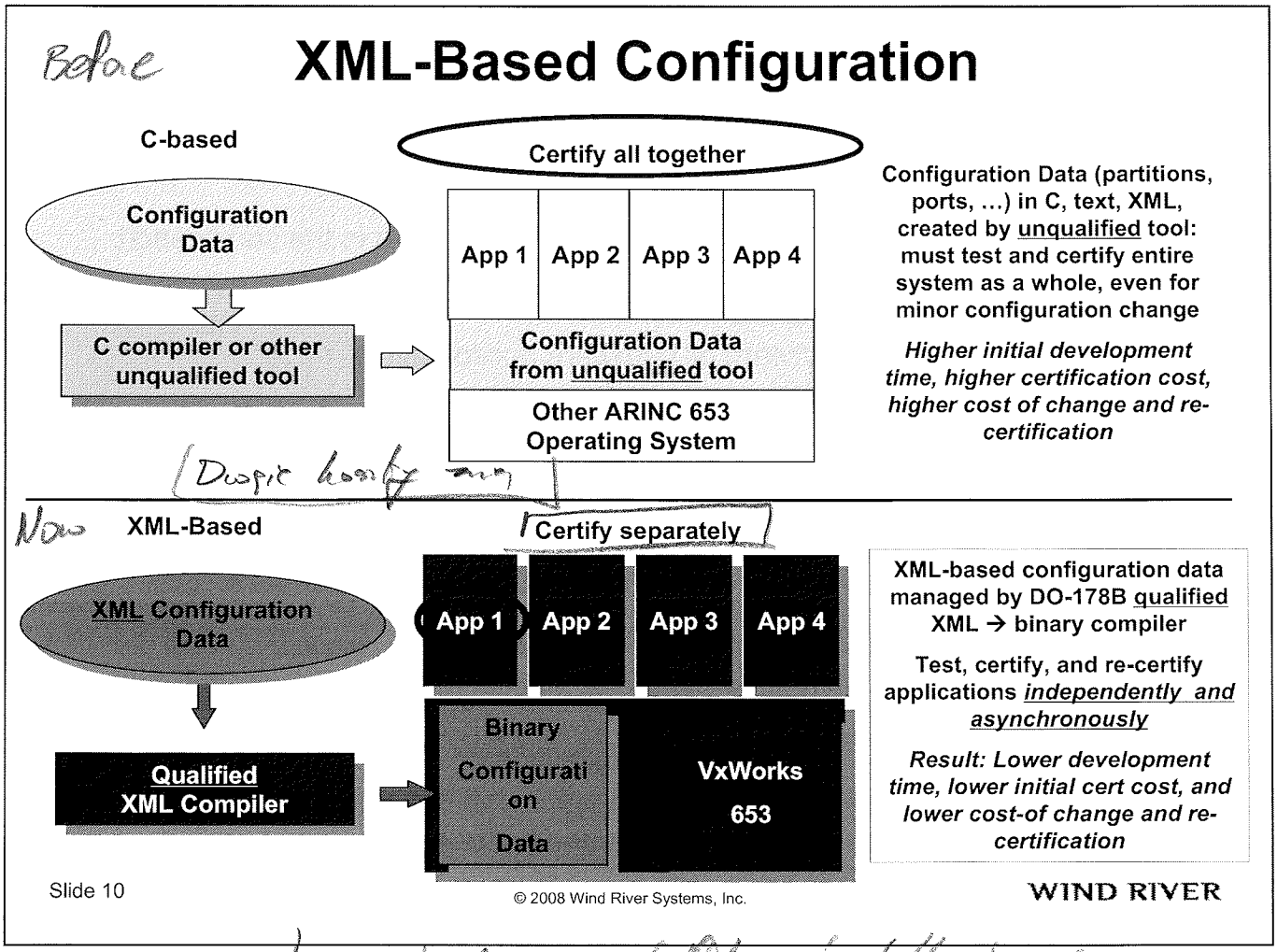
Slide 9

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

→ jeśli Application uzyska certyfikat → może być użyta na innej platformie

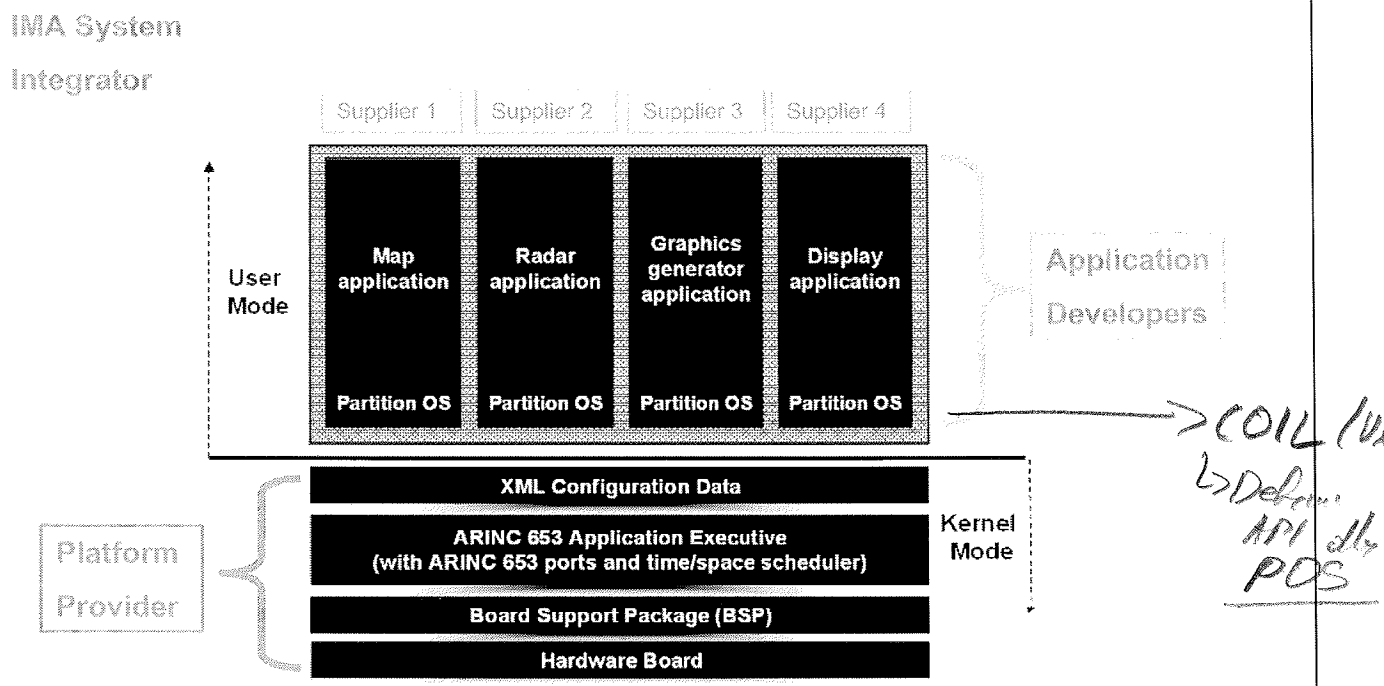


Student Notes

- New XML schema with heritage from ARINC 653 Supplement 1
- Proposing for Supplement 3 at the June 2005 ARINC meeting in Lisbon
- Boeing's Mike McNichols (original schema author): Supportivel, "Addresses known concerns"
- Used on the Boeing 787 Dreamliner, 767 Tanker, C-130 AMP, and others
- Wind River competitor: "Because used in the 787, new schema is a de facto standard"

© 2008 Wind River Systems, Inc.

ARINC 653 Certification



Slide 11

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Benefits

- Clearly defines *responsibility* and *ownership* of configuration data
- Enables each configuration entity to be submitted *independently*
- *Incremental* changes can be introduced *without* impacting the entire program
- Preserves *confidentiality* between parties since configuration data sharing is not required
- Establish the notion of *contract* between roles
- *Minimize* "cost of change"
- Create *manageable* configuration data set

Slide 12

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Independent Build, Link & Load

IBLL

Build Activities

- Each build activity provided by WR defines a *reference process*
  - Set of steps to build a system
- Highly *adaptable* to the development / certification process of each program
- Each entity is *independently* loadable on the target
- Based on *well known* tools
  - Compiler, linker, Makefiles
  - Only a few utilities builds in TCL
  - GUI are not qualifiable
- Repeatability and transparency is key for certification

Slide 13

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Reference Process

- The reference process is a tutorial supplied as part of the installation
- Located in
  `$(WIND_BASE)/target/reference/helloWorld/`
- The Reference Process is designed to provide detailed examples
- Documentation
  - `VxWorks 653 Configuration and Build Guide, version 2.2`

Slide 14

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

XML Configuration

- Based on XML schema outlined in the ARINC 653 specification
- Designed to meet the *usability* and *flexibility* of a commercial product
  - Cooperative effort between Verocel, Wind River and key customer programs
- WindRiver is a *leading* member of the ARINC 653 sub-committee in charge of proposing the *next generation* of XML schema
  - The proposal currently discussed by the sub-committee is based on WR XML schema

Slide 15

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- No conformance test for the Schema

Configuration and Build Partitioning

Configuration Overview

XML Schema

Configuration and Build Partitioning

- MOS and ACE
- Partitions
- Integration

Slide 16

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

XML Schema Overview

- An XML schema is a description of which elements are permitted in an XML document
  - The purpose of an XML schema is to define the legal building blocks of an XML document
- An XML schema defines
  - Elements that can appear in a document
  - Attributes that can appear in a document
  - Default and fixed values for elements and attributes
- An XML document is an instantiation of an XML schema

Slide 17

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- C++ Analogy
  - An XML schema is like a class
  - An XML document is like an object

XML Schema Reference

- Full documentation provided in the Online Help section of Workbench
  - Wind River Documentation → References → Operating System
- Contained in the ***VxWorks 653 Configuration and Build Guide, 2.2 Schema Reference***

Slide 18

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

XML Documents

- An XML document is a collection of information organized using elements
- Elements must begin with a start tag
  - For example,

    <Module>
- Elements must end with an end tag
  - For example,

    </Module>
- If there is no information between the start and end tag, the start and end tags can be combined
  - For example,

    <Module/> ← empty element

Slide 19

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

XML Elements

- XML elements can contain other XML elements between the start and end tags
  - For example,

```
<Module>
      <CoreOS>
      ...
      </CoreOS>
</Module>
```

Slide 20

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

XML Attributes

- XML elements can contain attributes in the start tag
  - For example,

```
<Module Name="vxWorks">
      <CoreOS>
            <CoreOSDescription
                  KernelName="coreOS"
                  BSPName="wrSbc750gx"
                  />
      </CoreOS>
</Module>
```

Slide 21

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

VxWorks 653 2.2 Schema

- Located in:
  - `target/config/xml/schema` (contains VerIMAx annotations)
  - `target/config/xml/cleanschema`
- Root element in a VxWorks 653 2.2 XML document is always **`<Module>`**

```
<Module>
   <CoreOS/>
   <Applications/>
   <SharedDataRegions/>
   <SharedLibraryRegions/>
   <Partitions/>
   <Schedules/>
   <Connections/>
   <HealthMonitor/>
</Module>
```


© 2008 Wind River Systems, Inc.

Student Notes

- The schema in the **`cleanschema`** directory does not contain annotations for use by VerIMAx.

© 2008 Wind River Systems, Inc.

System Configuration

- The **<Module>** element has two attributes
  - Name
  - Type

```
<Module Name="Module">
   <CoreOS/>
   <Applications/>
   <SharedDataRegions/>
   <SharedLibraryRegions/>
   <Partitions/>
   <Schedules/>
   <Connections/>
   <HealthMonitor/>
</Module>
```

Slide 23

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Module (Core) OS Configuration

- An Application Developer (AD) would not likely have ability to configure Module OS or Partition OS library
  - The MOS and POS are typically under the Platform Provider's (PP) control
  - This would then be imported into the application project
- Import Module OS definition, by using an **<xi:include>** tag
  - The **<xi:include>** tag must point to an XML file with a **<CoreOSDescription>** element as its root
  - Typically the same name as the BSP directory in target/config

```
<Module>
   <CoreOS>
      <xi:include
   href="$(DEV_BASE)/Platforms/$(PLATFORM)/bsp.xml" />
   </CoreOS>
</Module>
```

Slide 24

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- XML editors cannot recognize macros
  - Expand macros to use with editor
- External macros (eg **$WIND_BASE**) are allowed in XML, as in above and in " <xi:include href="$(WIND_BASE)/target/config/wrSbc750gx/wrSbc750gx.xml"/>"

Application Configuration

- **<Applications>** contains a set of **<Application>** elements
- **<Application>** has one attribute:
  - **Name**
    - Names the application so it can be referenced by other XML elements
    - Application name is not accessible on target

```
<Module>
   <Applications>
    <Application
     Name="firstPartition"/>
    <Application
     Name="secondPartition"/>
   </Applications>
</Module>
```

Slide 25

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- The Application description is defined in a document that is outside the module (for reuse).
  - It gets "included" into a module
  - Shows the separation of the schema

<ApplicationDescription> Element

- An **<Application>** element contains an **<ApplicationDescription>** element
- Any **<ApplicationDescription>** can be replaced by an **<xi:include>** tag that points to an XML document with an **<ApplicationDescription>** element as its root

```
<Module>
   <Applications>
        <Application Name="firstPartition">
              <ApplicationDescription/>
        </Application>
        <Application Name="secondPartition">
              <xi:include href="app2.xml"/>
        </Application>
   </Applications>
</Module>
```


© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- To programmatically get a partition name/ID

```
PARTITION_STATUS_TYPE partStatus;
 GET_PARTITION_STATUS(&partStatus, &retCode );
if (retCode != NO_ERROR)
 {
        printf("Recv InitComm: Unable to determine partition number, init failed\n");
        return;
    }
    else
    {
    nb_part = partStatus.IDENTIFIER;
    }
....
```

- Using the native vxWorks call:

```
STATUS configRecordFieldGet
    (
    int    field,            /* field selector */
    void * rValue            /* return value */
    )

field                   rValue type
PARTITION_NUMBER  int*
```

<ApplicationDescription> Attributes

- The **<ApplicationDescription>** element has these ~~attributes:~~
  - **MemorySize**
  - **Ports**

- The MemorySize elements are the Black Box definitions for the Application static memory usage
  - Must be multiples of the MMU page size used on the target

```
<Module>
   <Applications>
      <Application
       Name="firstPartition">
      <ApplicationDescription
       MemorySizeBss="0x1000"
       MemorySizeText="0x4000"
       MemorySizeData="0x1000"
       MemorySizeRoData="0x1000"
       />
      </Application>
   </Applications>
</Module>
```

Slide 27

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- Split mode drivers do not require memory mapped IO in the partition since that portion typically exists in the Module OS in that model.
  - The partition based user mode driver maps the device registers into the partition space using a shared data region for access

<ApplicationDescription> Elements

- The <ApplicationDescription> element has these elements:
  - **MemorySize**
  - **Ports**

```
<Module>
   <Applications>
      <Application Name="firstPartition">
         <ApplicationDescription>
            <MemorySize/>
            <Ports/>
         </ApplicationDescription>
      </Application>
   </Applications>
</Module>
```

Slide 28

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Application Black Box

- The **<MemorySize>** element defines "black box" for application
  - **MemorySizeBss**
  - **MemorySizeText**
  - **MemorySizeData**
  - **MemorySizeRoData**
  - **MemorySizePersistentData**
  - **MemorySizePersistentBss**

```
<Module>
    <Applications>
        <Application Name="firstPartition">
            <ApplicationDescription>
                <MemorySize
                MemorySizeBss="0x1000"
                MemorySizeText="0x4000"
                MemorySizeData="0x1000"
                MemorySizeRoData="0x1000"
                MemorySizePersistentData="0x800"
                MemorySizePersistentBss="0">
                </MemorySize>
            </ApplicationDescription>
        </Application>
    </Applications>
</Module>
```


© 2008 Wind River Systems, Inc.

Student Notes

- To determine BLACK BOX memory size use
  - objdumpppc --section-headers <filename>
  - readelfppc

Additional Memory Sections

- The **<MemorySize>** element can define black box sizes for more sections using the **<AdditionalSection>** element
  - Allowable types:
    - TEXT
    - RODATA
    - DATA
    - BSS
    - PERSISTENT_DATA
    - PERSISTENT_BSS

```
<Module>
   <Applications>
      <Application
   Name="firstPartition">
         <ApplicationDescription>
           <MemorySize
         MemorySizeBss="0x1000"
         MemorySizeText="0x4000"
         MemorySizeData="0x1000"
         MemorySizeRoData="0x1000"

   MemorySizePersistentData="0x1000"
         MemorySizePersistentBss="0">
        <AdditionalSection Name="myadd"
           Size="0x2000" Type="DATA"/>
          </MemorySize>
        </ApplicationDescription>
      </Application>
   </Applications>
</Module>
```

Slide 30

© 2008 Wind River Systems, Inc.

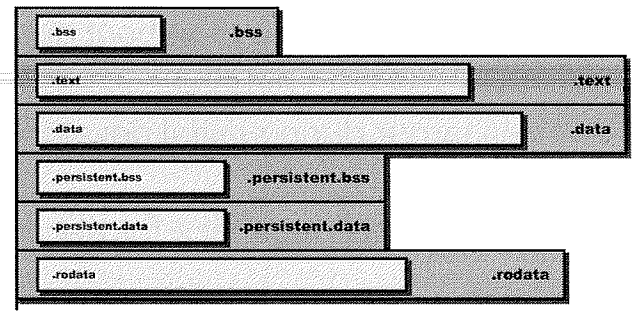
WIND RIVER

Student Notes

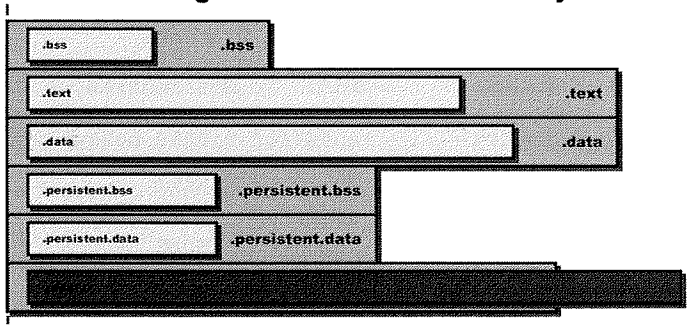
- Additional Section must also be defined programmatically, or via linker script

Black Box Configuration

Valid Configuration



Invalid Configuration – must redefine your black box



→ Dos/and configuration error

Slide 31

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- Graphical example of correct and incorrect parameter specifications for blackbox config in XML file

Determining Black Box Parameters

- Use **`readelfppc --section-headers`** ***`filename.sm`***
  - ***`filename.sm`*** is the file that is loaded by the boot application via the ***`boot.txt`*** file for a given application partition
  - It is best to make the sizes larger during initial development and debug and then adjust downward
  - All sizes must be modulo the MMU page size for the architecture (PPC is 4096 [0x1000])

```
> readelfppc --section-headers AppPartition1.sm
There are 18 section headers, starting at offset 0x1d448:
Section Headers:
  [Nr] Name              Type            Addr     Off    Size
  [ 0]                   NULL            00000000 000000 000000
  [ 1] .persistent.data  PROGBITS        40007000 016000 000000
  [ 2] .persistent.bss   PROGBITS        40008000 016000 000000
  [ 3] .text             PROGBITS        40000000 010000 000898
  [ 4] .rodata           PROGBITS        40004000 014000 000204
  [ 5] .data             PROGBITS        40005000 015000 0000c8
  [ 6] .bss              NOBITS          40006000 016000 000014
  [ 7] .comment          PROGBITS        00000000 016000 0000c9
```


© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- Tool for getting the correct values to use in the Black box configuration

© 2008 Wind River Systems, Inc.

Application Ports

- The **<Ports>** element configures application ports

```
<Module>
   <Applications>
      <Application Name="firstPartition">
         <ApplicationDescription>
            <MemorySize/>
            <Ports/>
         </ApplicationDescription>
      </Application>
   </Applications>
</Module>
```


© 2008 Wind River Systems, Inc.

Student Notes

© 2008 Wind River Systems, Inc.

<Ports> Element

- The **<Ports>** element has no attributes
- It contains a set of **<SamplingPort>** and **<QueuingPort>** elements

```
<Module>
   <Applications>
      <Application Name="myApp">
         <ApplicationDescription>
            <Ports>
               <SamplingPort/>
               <QueuingPort/>
               <QueuingPort/>
            </Ports>
         </ApplicationDescription>
      </Application>
   </Applications>
</Module>
```

Slide 34

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

<SamplingPort> Attributes

- A **<SamplingPort>** has these attributes:
  - **Name**
  - **Direction**
    - **SOURCE** or **DESTINATION**
  - **MessageSize**
    - In bytes
  - **RefreshRate**
    - In seconds

```
<Module>
   <Applications>
      <Application Name="myApp">
       <ApplicationDescription>
          <Ports>
             <SamplingPort Name="sampling"
            Direction="SOURCE"
            MessageSize="200"
            RefreshRate="0.01"/>
          </Ports>
       </ApplicationDescription>
      </Application>
   </Applications>
</Module>
```

Slide 35

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

<QueuingPort> Attributes

- A **<QueuingPort>** has these attributes:
  - **Name**
  - **Direction**
    - **SOURCE** or **DESTINATION**
  - **MessageSize**
    - In bytes
  - **QueueLength**
  - **Protocol**
    - **SENDER_BLOCK**, **RECEIVER_DISCARD** (source port only) or **NOT_APPLICABLE**

```
<Module>
  <Applications>
    <Application Name="myApp">
      <ApplicationDescription>
        <Ports>
          <QueuingPort Name="queuing"
          Direction="DESTINATION"
          MessageSize="1024"
          QueueLength="100"
          Protocol="NOT_APPLICABLE"/>
        </Ports>
      </ApplicationDescription>
    </Application>
  </Applications>
</Module>
```

Slide 36

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- Must have a "Protocol" attribute

Shared Data Configuration

- The **<SharedDataRegions>** element contains a set of **<SharedData>** elements
- **<SharedData>** has one attribute:
  - **Name**
    - Names the shared data region so it can be referenced by other XML elements
    - This will also be the shared data region's run time name

```
<Module>
   <SharedDataRegions>
      <SharedData
       Name="sdRgn1"/>
      <SharedData
       Name="sdRgn2"/>
   </SharedDataRegions>
</Module>
```

Slide 37

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

<SharedDataDescription> Element

- A **<SharedData>** element contains a **<SharedDataDescription>** element
- As with applications and shared libraries, this element can be replaced by an **<xi:include>** tag

```
<Module>
  <SharedDataRegions>
    <SharedData Name="sdRgn">
      <SharedDataDescription/>
    </SharedData>
    <SharedData
      Name="anotherRegion">
      <xi:include href="rgn.xml"/>
    </SharedData>
  </SharedDataRegions>
</Module>
```

Slide 38

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

<SharedDataDescription> Attributes

- A <SharedDataDescription> element has these attributes:
  - Size
  - VirtualAddress
    - A zero value allows the configuration system to place the SD region in virtual space automatically
  - CachePolicy
    - OFF, DEFAULT, COPY_BACK, WRITE_THRU, IO, COHERENCY
  - SystemAccess (MOS)
    - NONE, READ_ONLY, WRITE_ONLY, READ_WRITE
  - DataType

```
<Module>
  <SharedDataRegions>
    <SharedData Name="sdRgn">
     <SharedDataDescription
      Size="0x20000"
      VirtualAddress="0"
      CachePolicy="DEFAULT"
      SystemAccess="READ_WRITE"
      >
     </SharedDataDescription>
    </SharedData>
  </SharedDataRegions>
</Module>
```

Slide 39

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Partition Configuration

- **<Partitions>** lists the partitions in the system
- Each partition is a collection of:
  - An Application
  - One or more libraries
    - A partition must have a POS specified
  - Shared data regions

```
<Module>
    <Ace/>
    <CoreOS/>
    <Applications/>
    <SharedDataRegions/>
    <SharedLibraryRegions/>
    <Partitions/>
    <Schedules/>
    <Payloads/>
    <Connections/>
    <HealthMonitor/>
</Module>
```

Slide 40

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

<Partition> Attributes

- **<Partitions>** contains a set of **<Partition>** elements
- **<Partition>** has these attributes:
  - **Id**
    - The partition number (1 to 255)
    - 0 (zero) is reserved for the MOS
  - **Name**
    - Names the partition so it can be referenced by other XML elements
    - This will also be the application project's directory name, and the partition's run time name

```
<Module>
   <Partitions>
      <Partition Id="1"
       Name="firstPartition"
       />
      <Partition Id="2"
       Name="secondPartition"
       />
   </Partitions>
</Module>
```

Slide 41

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

<Partition> Element

- A **<Partition>** element contains a **<PartitionDescription>** element
- **<PartitionDescription>** has these elements
  - Application
  - SharedDataRegion
  - SharedLibraryRegion
  - Settings
- **<PartitionDescription>** elements can also be replaced by an **<xi:include>** tag
- **<PartitionDescription>** has no attributes

```
<Module>
   <Partitions>
      <Partition Name="firstPartition">
         <PartitionDescription>
            <Application/>
            <SharedDataRegion/>
            <SharedLibraryRegion/>
            <Settings/>
         </PartitionDescription>
      </Partition>
      <Partition Name="secondPartition">
         <xi:include href="part2.xml"/>
      </Partition>
   </Partitions>
</Module>
```

Slide 42

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

<PartitionDescription> Elements

- **<Application>** and **<SharedLibraryRegion>** each have one attribute
  - **NameRef**
- **<SharedDataRegion> has two attributes**
  - **NameRef**
  - **UserAccess** defines the partition's access permissions to the region
    - **NONE, READ_ONLY, WRITE_ONLY, READ_WRITE**
- These elements list domains that compose partition
- Must be exactly one application
- Can be zero or more shared data regions

```
<Module>
 <Partitions>
    <Partition Name="firstPartition">
     <PartitionDescription>
      <Application NameRef="hello"/>
      <SharedDataRegion
           UserAccess="READ_WRITE"
           NameRef="mySD"/>
      <SharedLibraryRegion NameRef="pos"/>
     </PartitionDescription>
    </Partition>
   </Partitions>
</Module>
```

Slide 43

© 2008 Wind River Systems, Inc.

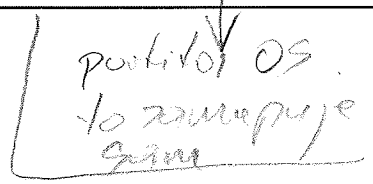
WIND RIVER

Student Notes

<Settings> Element

- **<Settings>** has multiple attributes that allow configuration of the run time variables used to configure the partition OS
  - An all ones value means to use the default value supplied by the system
  - **RequiredMemorySize** defines the overall partition container size and must be large enough to contain all the black boxes used PLUS allow sufficient heap

```
<Module>
   <Partitions>
      <Partition Name="firstPartition">
         <PartitionDescription>
            <Settings
                  RequiredMemorySize="0x200000"
                  PartitionHMTable="part1Hm"
                  numFiles="0xffffffff"
                  numDrivers="0xffffffff"
                  isrStackSize="0xffffffff"
            />
         </PartitionDescription>
      </Partition>
   </Partitions>
</Module>
```



Slide 44

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- Refer to the Schema Reference for a complete list of available attributes and their possible values

Schedule Configuration

- Partition schedules are defined in the **<Schedules>** element
- The **<Schedules>** element contains a sequence of **<Schedule>** elements
  - One **<Schedule>** element per schedule

```
<Module>
   <CoreOS/>
   <Applications/>
   <SharedDataRegions/>
   <SharedLibraryRegions/>
   <Partitions/>
   <Schedules/>
        <Schedule/>
        <Schedule/>
   <Connections/>
   <HealthMonitor/>
</Module>
```

Slide 45

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- Naming inconsistencies are found during the VerIMAx checking process rather than at run time.

<Schedule> Attributes

- A **<Schedule>** element has these attributes
  - **Id**
    - The schedule number from 0 to 15
- Must have schedule 0
  - Schedule 0 **always** runs on power up
  - 1-15 reserved for System Integrator
  - Multiple schedules can be used to performed phased initialization of a system at power up

```
<Module>
   <Schedules>
      <Schedule/>
      <Schedule
        Id="0"
        >
      </Schedule>
   </Schedules>
</Module>
```

Slide 46

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

<PartitionWindow> Element

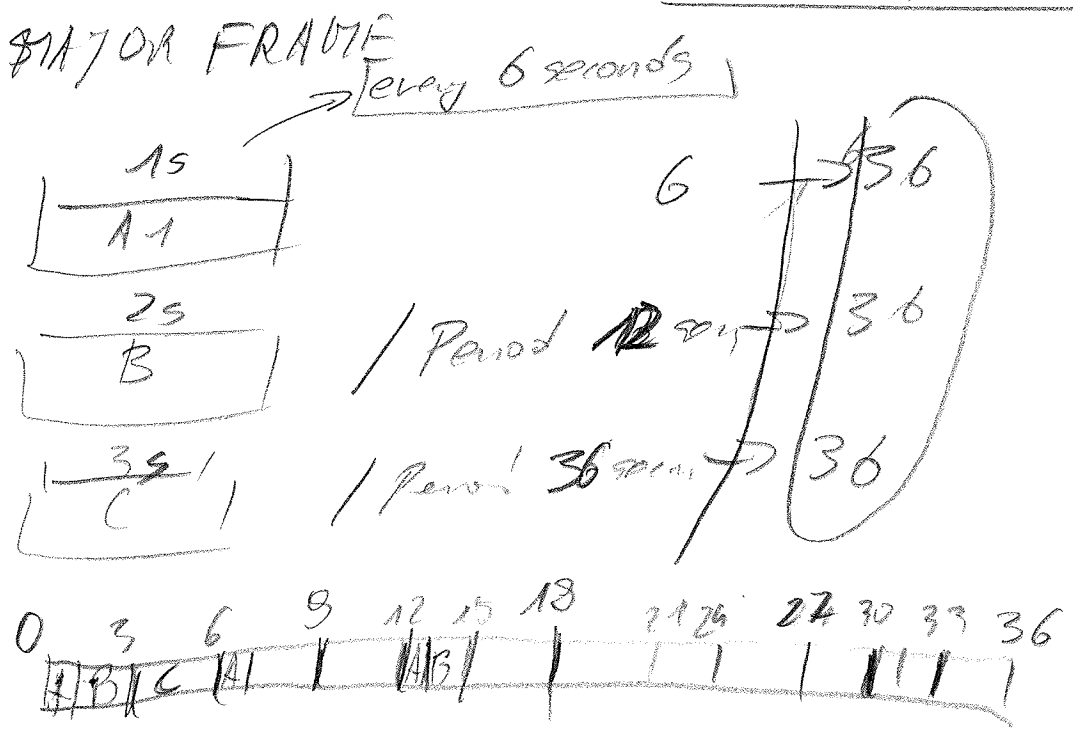
- A **<Schedule>** element contains a sequence of **<PartitionWindow>** elements
- A **<PartitionWindow>** element describes a minor frame
  - **PartitionNameRef**
    - Partition Name
    - **SPARE** which provides time only for the MOS
      - Primarily used as a way to debug partition initialization
  - **Duration**
    - In seconds
    - Must include leading zero
  - **ReleasePoint**
    - For periodic processes, it must be set to "true" for at least one partition window in the major frame

```
<Module>
  <Schedules>
    <Schedule Id="0">
      <PartitionWindow/>
      <PartitionWindow
        PartitionNameRef="firstPartition"
        Duration="0.05"
        ReleasePoint="true"
        />
    </Schedule>
    <Schedule Id="1">
      <PartitionWindow/>
    </Schedule>
  </Schedules>
</Module>
```


© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes



Debug Schedule Example

```
<Module>
   <Schedules>
      <Schedule Id="0">
        <PartitionWindow
        PartitionNameRef="SPARE"
        Duration="0.5"
        ReleasePoint="true"
        />
        <PartitionWindow
        PartitionNameRef="secondPartition"
        Duration="0.25"
        ReleasePoint="true"
        />
      </Schedule>
   </Schedules>
</Module>
```

Slide 48

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Channel Configuration

- The **<Connections>** element connects port elements together using **<Channel>** elements
- **<Connections>** has no attributes
- Each **<Channel>** element has:
  - **Id** attribute
  - A **<Source>** element
  - A **<Destination>** element
- The **<Source>** and **<Destination>** elements each have two attributes:
  - **PartitionNameRef**
    - The name of the partition whose application defines the port
  - **PortNameRef**
    - The name of the port

```
<Module>
   <Connections>
     <Channel/>
     <Channel Id="2">
       <Source
       PartitionNameRef="firstPartition"
       PortNameRef="part1Port"/>
       <Destination
       PartitionNameRef="secondPartition"
       PortNameRef="part2Port"/>
      </Channel>
      <Channel/>
   </Connections>
</Module>
```

Slide 49

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Inter-Partition Configuration

- Port records are configured using the **<Ports>** elements in the **<ApplicationDescription>** element
- Port channels (port records with matching channel ID's) are configured using **<Connections>**
- Definitions must be consistent between the two elements

```
<Module>
   <Applications>
    <Application>
     <ApplicationDescription>
       <Ports/>
     </ApplicationDescription>
    </Application>
   </Applications>
   <Connections/>
</Module>
```

Slide 50

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Health Monitor Configuration

- Defined in **<HealthMonitor>** element
- <HealthMonitor> for the Module has these elements
  - SystemHMTable
  - ModuleHMTable
  - One or more PartitionHMTable
    - Partitions can share a common table for HM handling or each partition can specify its own unique HM table
- Responsibility for the **<HealthMonitor>** element is typically shared between the Platform Provider (PP) and System Integrator (SI)

```
<Module>
   <CoreOS/>
   <Applications/>
   <SharedDataRegions/>
   <SharedLibraryRegions/>
   <Partitions/>
   <Schedules/>
   <Connections/>
   <HealthMonitor>
      <SystemHMTable/>
      <ModuleHMTable/>
      <PartitionHMTable/>
   </HealthMonitor>
</Module>
```

Slide 51

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- Application Developer typically does not configure HM

<SystemHMTable> Element

- Defined in **<SystemHMTable>** element
- The **<SystemHMTable>** for the Module has these elements
  - **Name**
  - **SystemState**
- The **<SystemHMTable>** element should contain a **SystemState** element for each of the three system states, **HM_MODULE_MODE**, **HM_PARTITION_MODE**, and **HM_PROCESS_MODE**

```
<Module>
    <HealthMonitor>
      <SystemHMTable Name="systemHm">
       <SystemState SystemState="HM_PARTITION_MODE">
          <ErrorIDLevel ErrorIdentifier="HME_DEFAULT"
          ErrorLevel="HM_PARTITION_LVL"/>
       </SystemState>
       <SystemState SystemState="HM_MODULE_MODE">
          <ErrorIDLevel ErrorIdentifier="HME_DEFAULT"
          ErrorLevel="HM_MODULE_LVL"/>
       </SystemState>
       <SystemState SystemState="HM_PROCESS_MODE">
          <ErrorIDLevel ErrorIdentifier="HME_DEFAULT"
          ErrorLevel="HM_PROCESS_LVL"/>
       </SystemState>
      </SystemHMTable>
      <ModuleHMTable/>
      <PartitionHMTable/>
    </HealthMonitor>
</Module>
```

Slide 52

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- The Application developer does not typically configure HM

<SystemState> Element

- Each **SystemState** element contains a list of **ErrorIDLevel** elements for each of the **SystemState** in order to map specific Errors to the HM level that will process it
- The **ErrorIDLevel** element identifies the health monitor level to be invoked when a particular type of error occurs

```
<Module>
    <HealthMonitor>
      <SystemHMTable Name="systemHm">
       <SystemState SystemState="HM_PARTITION_MODE">
          <ErrorIDLevel ErrorIdentifier="HME_DEFAULT"
          ErrorLevel="HM_PARTITION_LVL"/>
       </SystemState>
       <SystemState SystemState="HM_MODULE_MODE">
          <ErrorIDLevel ErrorIdentifier="HME_DEFAULT"
          ErrorLevel="HM_MODULE_LVL"/>
       </SystemState>
       <SystemState SystemState="HM_PROCESS_MODE">
          <ErrorIDLevel ErrorIdentifier="HME_DEFAULT"
          ErrorLevel="HM_PROCESS_LVL"/>
       </SystemState>
      </SystemHMTable>
      <ModuleHMTable/>
      <PartitionHMTable/>
    </HealthMonitor>
</Module>
```

Slide 53

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

<ModuleHMTable> Element

- The **ModuleHMTable** element contains a Module level health monitor table
- The **<ModuleHMTable>** for the Module has these Attributes
  - **Name**
  - **ModuleCallback** (optional)

```
<Module>
    <HealthMonitor>
      <SystemHMTable/>
      <ModuleHMTable Name="moduleHm"
        ModuleCallback="">
          <SystemState>
            <ErrorIDAction
                    ErrorIdentifier="HME_UNKNOWN"
                    ErrorAction="hmDefaultHandler"/>
            <ErrorIDAction
                    ErrorIdentifier="HME_NUMERIC_ERROR"
                    ErrorAction=""/>
          </SystemState>
          <Settings
                    maxQueueDepth="34"
                    queueThreshold="32"
                    stackSize="16384"
                    maxLogEntries="100"
                    logEntriesThreshold="98"
                    attributeMask="0x00000001"
                    eventFilterMask="0xFFFFFFFF"
                    maxErrorHandlerQueueDepth="128"
                    errorHandlerQueueThreshold="126">
          </Settings>
      </ModuleHMTable>
      <PartitionHMTable/>
    </HealthMonitor>
</Module>
```

Slide 54

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

<PartitionHMTable> Element

- The **PartitionHMTable** element contains a partition health monitor table
- At least one must be present
  - Partitions can share a common table or have a unique one assigned to each partition
- The **<PartitionHMTable>** for the Module has these Attributes
  - **Name**
    - Used by partitions to reference the desired table for operation
  - **PartitionCallback** (optional)

```
<Module>
    <HealthMonitor>
      <SystemHMTable/>
      <ModuleHMTable/>
      <PartitionHMTable Name="part1Hm"
        PartitionCallback="">
          <SystemState>
            <ErrorIDAction
                    ErrorIdentifier="HME_UNKNOWN"
                    ErrorAction="hmDefaultHandler"/>
            <ErrorIDAction
                    ErrorIdentifier="HME_NUMERIC_ERROR"
                    ErrorAction=""/>
          </SystemState>
          <Settings
                    maxQueueDepth="34"
                    queueThreshold="32"
                    stackSize="16384"
                    maxLogEntries="100"
                    logEntriesThreshold="98"
                    attributeMask="0x00000001"
                    eventFilterMask="0xFFFFFFFF"
                    maxErrorHandlerQueueDepth="128"
                    errorHandlerQueueThreshold="126">
          </Settings>
    </PartitionHMTable>
    </HealthMonitor>
</Module>
```

Slide 55

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

target/sucl

usrHm.c → declare handlers

hmDefaultHandlers.c

hmTypes.h

Process-level → error create handler ...

voice application errors

→ user defined errors

<SystemState> Element

- Each **SystemState** element contains these elements
  - **ErrorIdentifier**
  - **ErrorAction**
- The **SystemState** maps a specific error code (identifier) with a handler (function) that will be called in order to perform the **ErrorAction**
- The **SystemState** is typically a long list of errors that need to be handled by the Module Level HM

```
<Module>
    <HealthMonitor>
      <SystemHMTable/>
      <ModuleHMTable/>
      <PartitionHMTable Name="part1Hm"
        PartitionCallback="">
          <SystemState>
            <ErrorIDAction
                    ErrorIdentifier="HME_UNKNOWN"
                    ErrorAction="hmDefaultHandler"/>
            <ErrorIDAction
                    ErrorIdentifier="HME_NUMERIC_ERROR"
                    ErrorAction=""/>
          </SystemState>
          <Settings
                    maxQueueDepth="34"
                    queueThreshold="32"
                    stackSize="16384"
                    maxLogEntries="100"
                    logEntriesThreshold="98"
                    attributeMask="0x00000001"
                    eventFilterMask="0xFFFFFFFF"
                    maxErrorHandlerQueueDepth="128"
                    errorHandlerQueueThreshold="126">
          </Settings>
    </PartitionHMTable>
    </HealthMonitor>
</Module>
```

Slide 56

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

<Settings> Element

- Each **Settings** element defines the run-time behavior of the partition health monitor

```
<Module>
    <HealthMonitor>
      <SystemHMTable/>
      <ModuleHMTable/>
      <PartitionHMTable Name="part1Hm"
        PartitionCallback="">
          <SystemState>
            <ErrorIDAction
                    ErrorIdentifier="HME_UNKNOWN"
                    ErrorAction="hmDefaultHandler"/>
            <ErrorIDAction
                    ErrorIdentifier="HME_NUMERIC_ERROR"
                    ErrorAction=""/>
          </SystemState>
          <Settings
                    maxQueueDepth="34"
                    queueThreshold="32"
                    stackSize="16384"
                    maxLogEntries="100"
                    logEntriesThreshold="98"
                    attributeMask="0x00000001"
                    eventFilterMask="0xFFFFFFFF"
                    maxErrorHandlerQueueDepth="128"
                    errorHandlerQueueThreshold="126">
          </Settings>
    </PartitionHMTable>
    </HealthMonitor>
</Module>
```

Slide 57

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- Application Developer typically does not configure HM
- Full details of all elements are in the Schema Reference

© 2008 Wind River Systems, Inc.

Configuration and Build Partitioning

Configuration Overview
XML Schema
Configuration and Build Partitioning
- MOS and ACE
- Partitions
- Integration

Slide 58

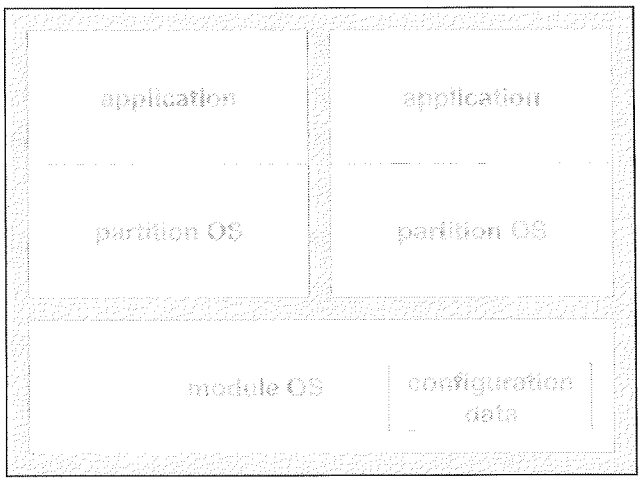
© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Configuration and Build Partitioning

- A VxWorks 653 system consists of at least four components:
  - A module OS (MOS)
  - Configuration data
  - At least one partition OS (POS)
  - At least one application partition



Slide 59

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Supporting DO-297/ED-12B Role Separation

- Configuration and Build Partitioning means:
  - Independent Build
    - Don't need the entire source to build one piece
    - No more "system" project that builds everything
  - Independent Link
    - Don't need OS binaries to link an application
  - Independent Load
    - Binaries can be loaded/updated (flashed) separately
- Result:
  - Development Roles can be accomplished independently per DO-297/ED-12B

Slide 60

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Build Process Overview

- The build process appears to be more complex
- Independence has many advantages:
  - The process is very modular and very flexible
    - Easily adapted to existing customer environment
  - Independence reduces the impact of change at all stages of development
    - Including during test for credit for certification
  - Hidden complexity creates problems for our customers.
    - Transparency is the key to reducing costs and schedules.
  - The process makes it easier for multiple suppliers to work together
    - Defined interface points in process insure expectations are met

```
make -C moduleOS
make -C partitionOS
make -C application
make -C integration
```

Slide 61

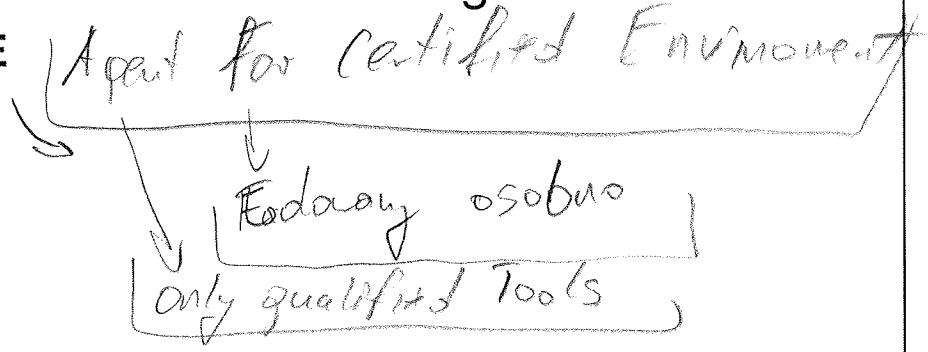
© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Configuration and Build Partitioning

Configuration Overview
XML Schema
Configuration and Build Partitioning
MOS and ACE
Partitions
Integration



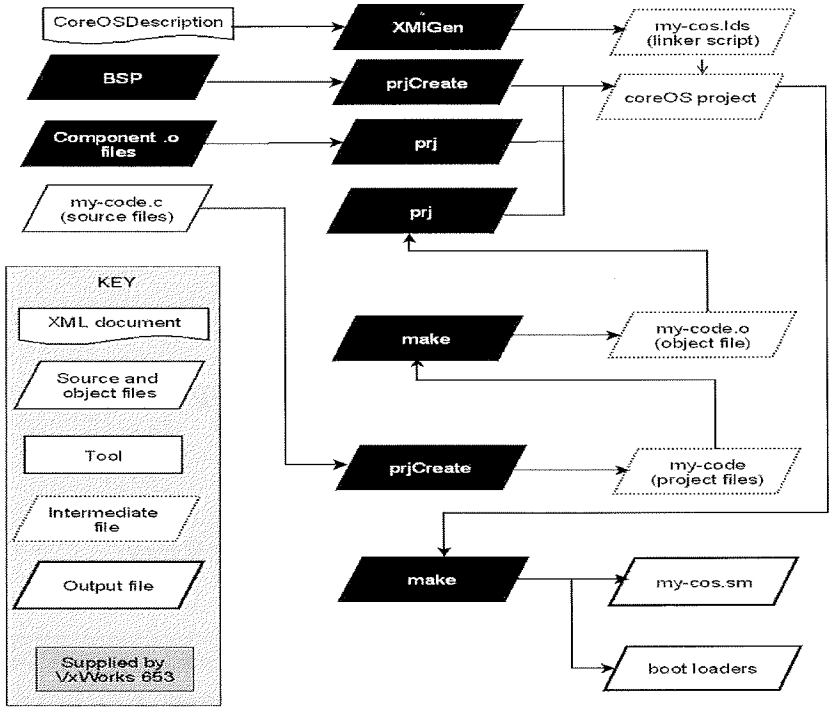
Slide 62

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Core OS Configuration and Build Process



Slide 63

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

+ external module to link
-> make file

Module OS Configuration

- MOS is configured in two steps:
  - Create a MOS project and makefile, **`prjCreate -domType kernel`**
    - Specify a BSP
      `-bsp bspName`
    - Specify the MOS directory
      `-prj mosdirName`
    - Specify the MOS name
      `-name mosName`
  - Generate a linker script, **`xmlgen --ldScript`**
    - Specify the architecture, BSP xml description
      `-arch archName`
      `bsp bspName.xml`
    - Output the linker script file (`.lds`) to the MOS directory
      `-o mosdirName/mosName.lds`

```
prjCreate -domType kernel -bsp wrSbc8349 -prj mosdir -name mos
xmlgen --ldScript --arch ppc -o mosdir/mos.lds bsp.xml
```

Slide 64

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- The MOS name must match the **`@KernelName`** attribute in the XML configuration

Module OS Build

- Once the MOS is created, "make" will build it

```
prjCreate -domType kernel -bsp wrSbc8349 -prj mosdir -name mos

xmlgen --ldScript --arch ppc -o mosdir/mos.lds bsp.xml

make
```

Slide 65

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Configuring a Cert MOS

- To create a cert MOS
  - Use the "**-ddf**" option to select the cert MOS composition
  - Use "**prj**" to select the cert build specification

```
prjCreate -domType kernel -ddf certKernel -bsp wrSbc8349 -prj mosdir
   -name mos
prj projBuildSet -prj mosdir PPC603gnu.cert

xmlgen --ldScript --arch ppc -o mosdir/mos.lds bsp.xml
```

Slide 66

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Configuring Module OS Components

- To change the MOS composition
  - Use "**prj**" to add / remove / configure MOS components
  - Use "**prj**" to set MOS component values

```
prjCreate -domType kernel -bsp wrSbc8349 -prj mosdir -name mos
prj domComponentBundleAdd -prj mosdir targetTools.ddf
prj domComponentRemove    -prj mosdir INCLUDE_SHELL_VI_MODE
prj domComponentAdd       -prj mosdir INCLUDE_SHELL_EMACS_MODE
prj domComponentValueSet  -prj mosdir SHELL_STACK_SIZE 0x8000

xmlgen --ldScript --arch ppc -o mosdir/mos.lds bsp.xml
```

Slide 67

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Adding Source Files to Module OS

- To add source files to the MOS, create a source component and add it to the MOS

```
prjCreate -domType kernel -bsp wrSbc8349 -prj mosdir -name mos
prjCreate -type kernelComponent -prj compdir -build PPC603gnu.debug
prj compFileAdd       -prj compdir "myfile.c"
prj compAttributeSet -prj compdir INIT_RTN "myComponentInit();"
prj domComponentAdd  -prj mosdir compdir INCLUDE_COMPDIR

xmlgen --ldScript --arch ppc -o mosdir/mos.lds bsp.xml
```

Slide 68

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Creating an ACE Project

- To create an ACE project
  - Create the project and makefile
    `prjCreate -domType ace`
  - Generate the linker script
    `xmlgen --ldScript`
  - Include the serial and/or Ethernet components
    `INCLUDE_WDB_COMM_SERIAL, INCLUDE_WDB_COMM_MINI_UDP`

```
prjCreate -domType ace -build PPC603gnu.cert -prj acedir -name ace
prj projBuildTagValueSet -prj acedir COREOS_DIR mosdir/PPC603gnu.cert
prj projBuildTagValueSet -prj acedir COREOS_NAME mos.sm
prj domComponentAdd      -prj acedir INCLUDE_WDB_COMM_SERIAL
prj domComponentAdd      -prj acedir INCLUDE_WDB_COMM_MINI_UDP
prj domParameterValueSet -prj acedir WDB_TTY_CHANNEL 0

xmlgen --ldScript --arch ppc -j ace -o acedir/ace.lds module.xml
```

Slide 69

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Configuration and Build Partitioning

Configuration Overview
XML Schema
Configuration and Build Partitioning
- MOS and ACE
- **Partitions**
- Integration

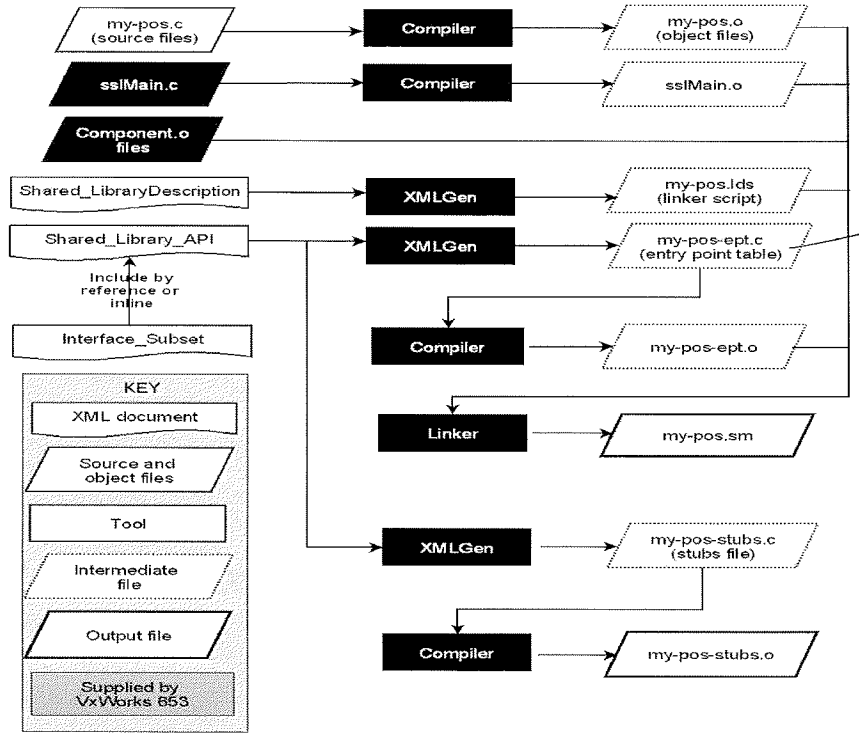
Slide 70

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Partition OS Configuration and Build



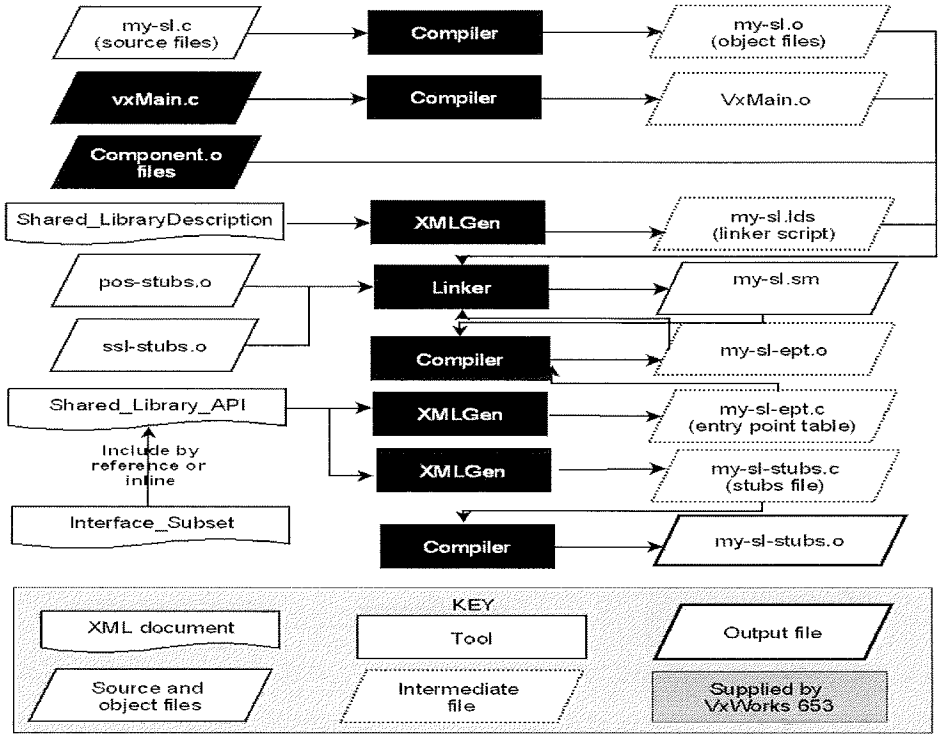
Slide 71

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Shared Library Configuration



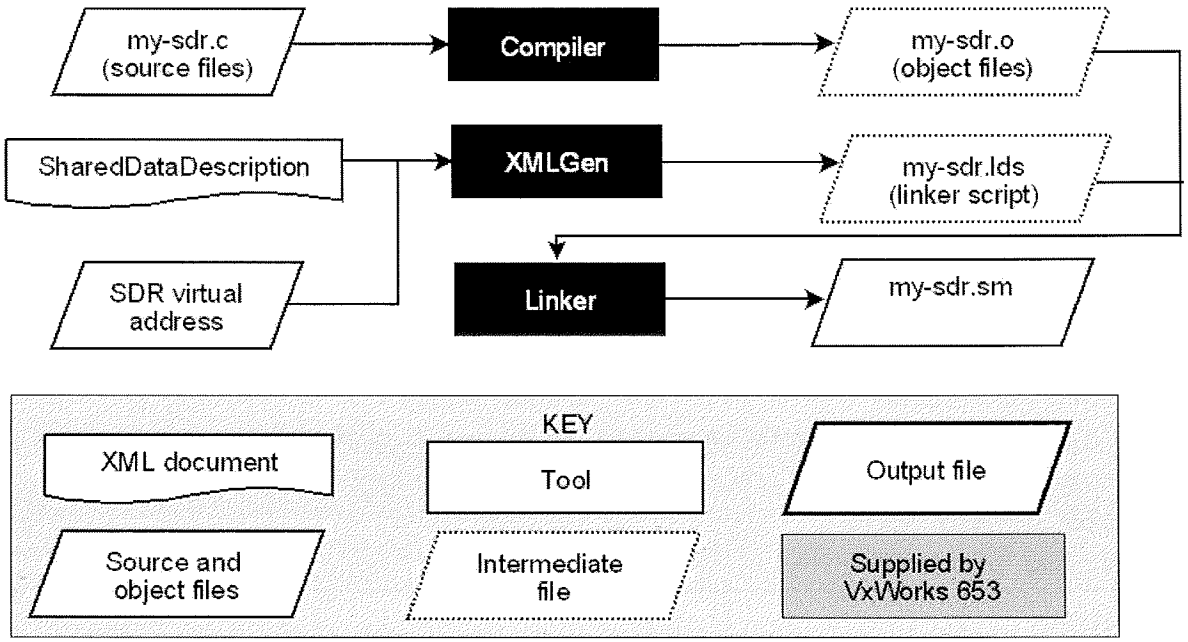
Slide 72

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Shared Data Region Configuration and Build



Slide 73

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Partition OS Makefile

- Write a **`Makefile`** to configure the partition OS
  - There is no "project creation" step

```
all: pos.sm pos-stubs.o

include $(WIND_BASE)/target/vThreads/config/make/Makefile.vars
include $(WIND_BASE)/target/vThreads/config/make/Makefile.rules

pos.sm: sslMain.o vThreadsComponent.o pos-ept.o pos.lds
pos-ept.c: api.xml
pos-stubs.c: api.xml
pos.lds: my-pos.xml
```

Slide 74

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Partition OS Configuration

- **`Makefile.vars`** and **`Makefile.rules`** define rules for building everything
  - **`Makefile`** only needs to provide dependencies
- The partition OS developer has two deliverables
  - The partition OS binary (**`pos.sm`**)
    - **`pos.sm`** depends on
      - **`sslMain.o`** (This file *must* be first)
      - vThreads (**`vThreadsComponent.o`**) COIL, APEX
      - The entry point table, **`pos-ept.o`** for the application stubs to attach to
        - » The entry point table is generated from the SL API XML document
      - The linker script, **`pos.lds`**, generated from the POS black box
  - The stubs file, **`pos-stubs.o`**
    - The stubs are generated from the same SL API XML document as the entry point table

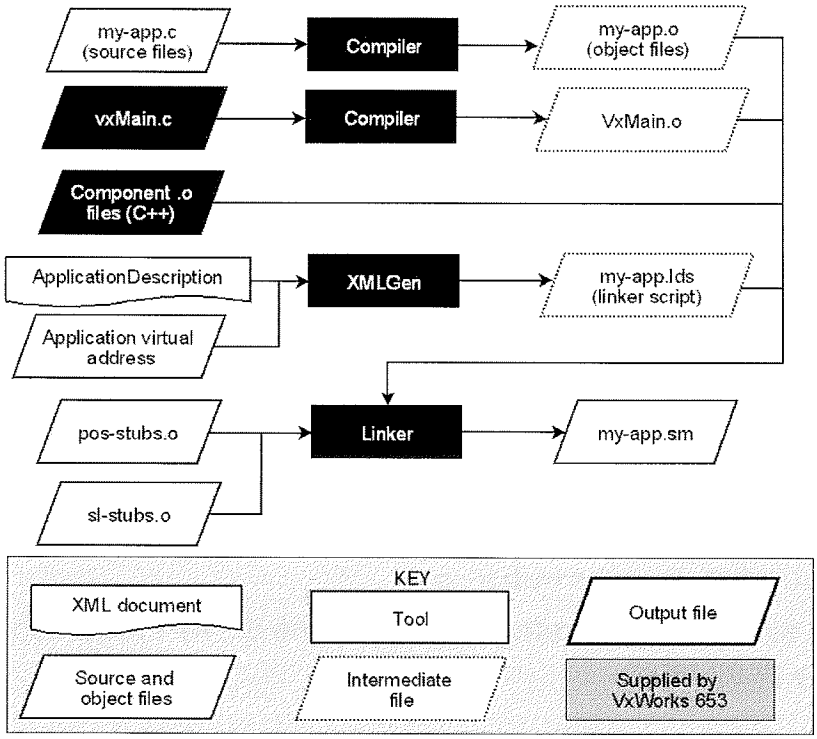
Slide 75

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Application Configuration and Build



Slide 76

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Adding Components to Partition OS

- Include additional components as dependencies for the partition OS binary:

```
all: pos.sm pos-stubs.o

include $(WIND_BASE)/target/vThreads/config/make/Makefile.vars
include $(WIND_BASE)/target/vThreads/config/make/Makefile.rules

pos.sm: sslMain.o vThreadsComponent.o apexComponent.o myComponent.o pos-
    ept.o pos.lds
pos-ept.c: api.xml
pos-stubs.c: api.xml
pos.lds: my-pos.xml
```

Slide 77

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Application Configuration

- The application **`Makefile`** is similar to the partition OS **`Makefile`**
- The application developer has one deliverable, the application binary, **`app.sm`**
  - **`app.sm`** depends on
    - **`vxMain.o`** (This file *must* be first)
    - POS stubs, **`pos-stubs.o`**, provided by the POS developer
    - The linker script, **`app.lds`**, generated from the application black box

```
all: app.sm

include $(WIND_BASE)/target/vThreads/config/make/Makefile.vars
include $(WIND_BASE)/target/vThreads/config/make/Makefile.rules

app.sm: vxMain.o pos-stubs.o app.lds
app.lds: my-app.xml
```

Slide 78

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- Without **`pos-stubs.o`**, the application will not be able to call any POS functions

Adding Application Source

- To add source files to the application, add the associated object files to the dependency list for the application binary

```
all: app.sm

include $(WIND_BASE)/target/vThreads/config/make/Makefile.vars
include $(WIND_BASE)/target/vThreads/config/make/Makefile.rules

app.sm: vxMain.o myApplication.o pos-stubs.o app.lds
app.lds: my-app.xml
```

Slide 79

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Configuration and Build Partitioning

Configuration Overview
XML Schema
Configuration and Build Partitioning
- MOS and ACE
- Partitions
- **Integration**

Slide 80

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Integration

- Write a **`Makefile`** to build the configuration record and payloads
- Targets defined in **`Makefile.rules`** :
  - **`net`** rule builds a netboot image
    - Requires a list of .sm files
  - **`ram / rom`** builds a RAM/ROM payload
    - Requires a list of .bin files instead of .sm files

```
all: net ram rom

XML_FILE = module.xml
SM_FILES = $(shell xmlgen --fileList $(XML_FILE))
BIN_FILES = $(shell xmlgen --fileList --bin $(XML_FILE))

include $(WIND_BASE)/target/config/make/Makefile.vars
include $(WIND_BASE)/target/config/make/Makefile.rules
```

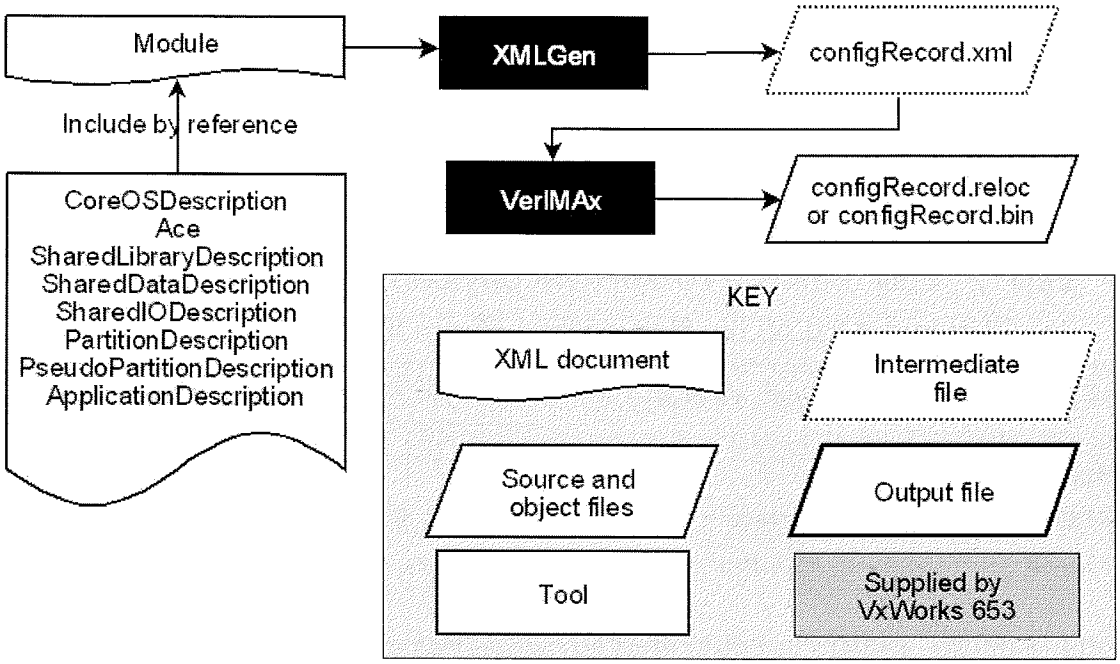
Slide 81

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Building the Configuration Record



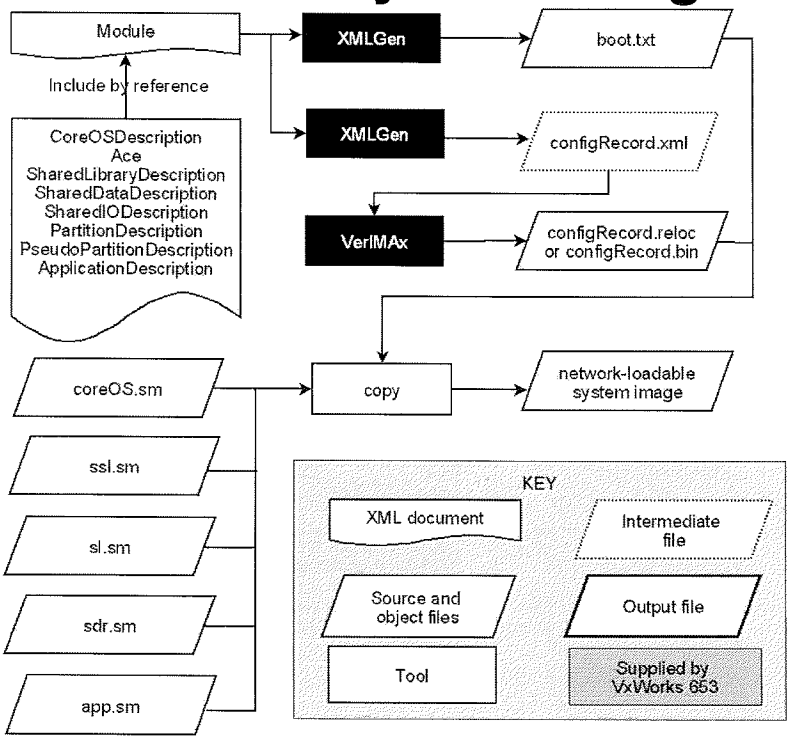
Slide 82

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Configuring and Building a Network Loadable System Image



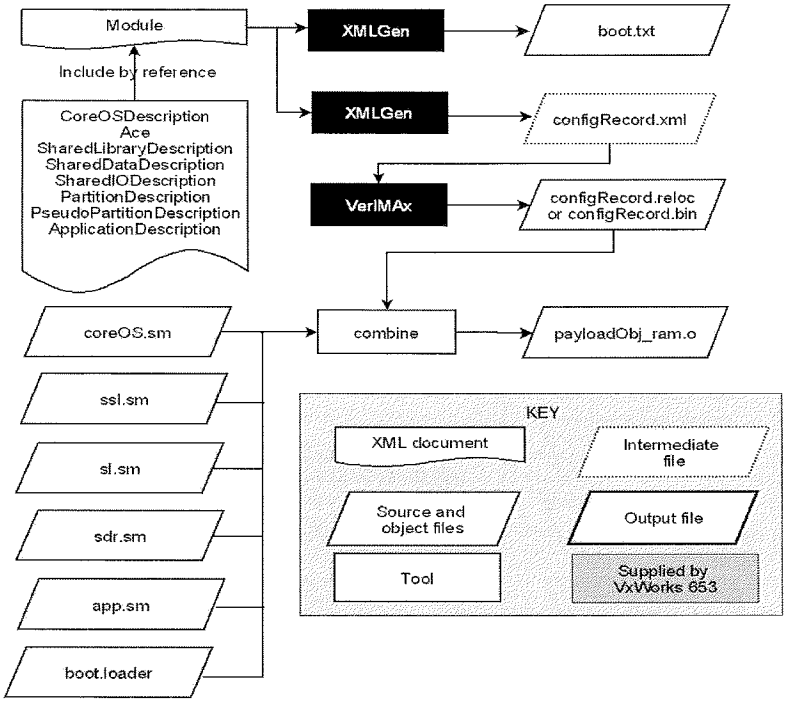
Slide 83

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Configuring and Building a RAM Payload System Image



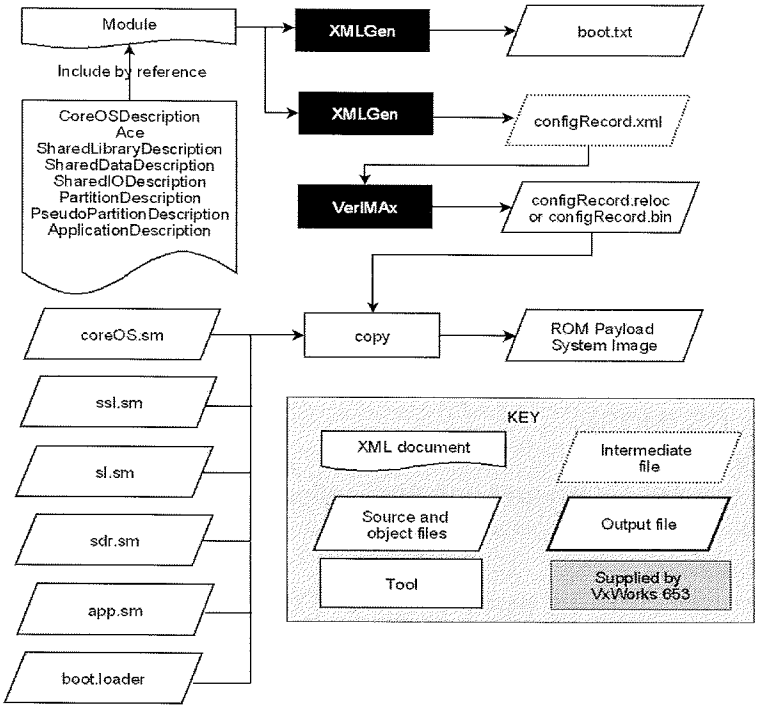
Slide 84

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Configuring and Building a ROM Payload System Image



Slide 85

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Questions

1. Partitioning means that the configuration data and build activities are separated and organized per IMA roles. True or False?
2. The configuration and build of VxWorks 653 systems involves a collection of:
   a. XML documents:
   b. source code files;
   c. binary components;
   d. command-line tools.
   e. All of the above
3. The "net" rule builds a _____
4. The mandatory components of a VxWorks 653 system are:
   a. A module OS (MOS)
   b. Configuration data
   c. One partition OS (POS) and one application
   d. Payloads
   e. All of the above
5. Adding components to the Partition OS does not require including the components as dependencies for the Partition OS binary. True or False?

Slide 86

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

1. True
2. E
3. netboot image
4. A, B and C
5. False

Review

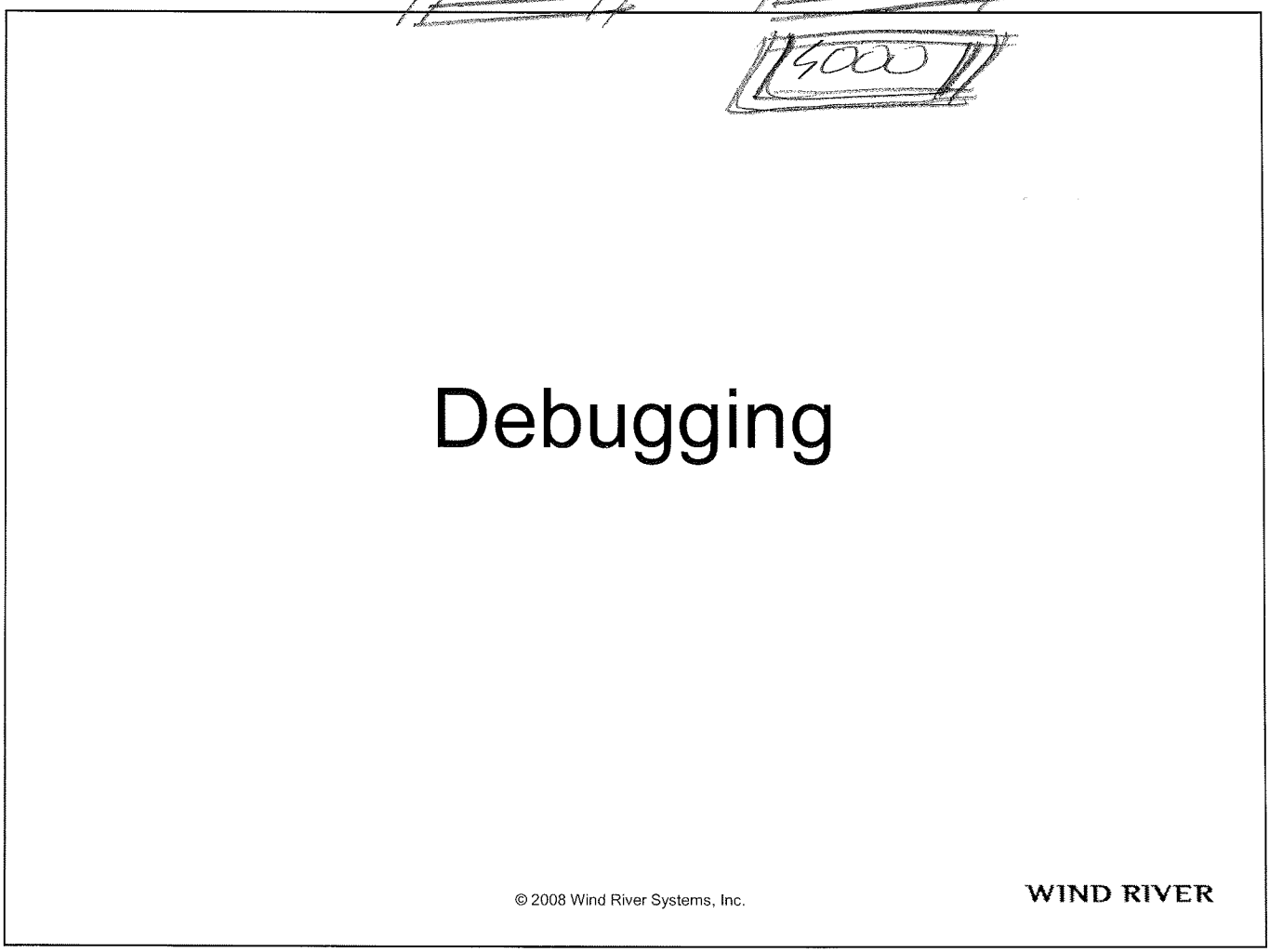
- In this chapter you learned:
  - Configuration and development processes are the key factors for successful certification
    - Time and integration are where the challenges are, not the applications, the OS or the hardware
  - Special emphasis should be put on these two areas from the start of a program
  - Both areas needs to be carefully designed
  - Examined the build process for all parts of a VxWorks 653 system
    - Use prjCreate and prj to create Module OS and ACE projects
    - Write Makefiles to build Partition OS and application projects
    - Write an integration Makefile to build the configuration record and payloads
  - Reference process examples are under `$(WIND_BASE)/target/reference/helloWorld/`
  - ***VxWorks 653 Configuration and Build Guide***, version 2.2 provides details of the reference process as well as possible variations

Slide 87

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes



Student Notes

Objectives

By the end of this chapter you will be able to:

- Use the Host-Based Shell
- Debug the code by using Wind River Workbench source-code debugger
- Perform system and task level debugging

Slide 2

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Debugging

- **Host Shell**
  - Basic Features
  - Advanced Features
- Kernel Objects Viewer
- Source Level Debugger
  - Task Mode Debugging
  - System Mode Debugging

Slide 3

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

The Host-Based Shell

- Interactive C expression interpreter
  - Call target functions in the Module OS
  - Examine and modify target variables
  - Create, modify and get information about system objects, such as partitions, tasks, semaphores and message queues
- Interactive Tcl interpreter
  - Flexible shell scripting and automation
  - Access WTX protocol
  - Create convenience commands
- To start the Host Shell, select target server and then
  - Select "Host Shell" from the "Target" menu
- Qualified Verification tool
  - List of qualified commands

Slide 4

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- Tcl is the Tool Command Language, a powerful scripting language
  - See Ousterhout's *Tcl and the Tk ToolKit* for information on Tcl.
- Must be knowledgeable user
- One can run multiple shells per VxWorks target
- The Host Shell can be started from the command line by executing *windsh*
- Qualified verification tool
  - Certifiable subset of the API
  - Subset is deterministic and testable

Partition Selection

- Within host shell, concept of a "current working partition" exists
  - Not applicable to the target shell
- Shell prompt indicates current working partition
- Colon operator is used to switch current working partition
- Current working partition can be any type
- Some shell commands are only valid within a partition

```
[coreOS] -> :firstPartition
[firstPartition] ->
```

Slide 5

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- The default current working PD is Kernel PD
- The colon operator without an argument can be used to switch to the kernel PD
- The colon operator does PD name completion
  - For example, if there are two partitions named **partFma** and **partNav**, then to select **partFma** one only needs to type **:partF**
- In the target shell, one can switch contexts, but there are no commands available

How the Shell Interprets Commands

The Host Shell uses target server's system symbol table and these rules to interpret names in expressions:

1. Use a Host Shell built-in command in preference to a target symbol of the same name.
2. Look for "symbol" in the symbol table.
3. Look for "_symbol" in the symbol table.
4. Convert a task name to a task ID.

```
[coreOS] -> retVal = printf ("Foo is %d\n", foo)
[coreOS] -> i (tWdbTask)
```

Slide 6

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- Built-in functions are implemented in the shell and execute at least partly on the host
  - The shell command i( ) is a built-in function which prints out information about VxWorks tasks
- A task ID is 32-bit number which uniquely identifies a task
- When a symbol must be created, the target server allocates memory on the target from the protection domain and stores the new symbol in the host-based symbol table
- For some target architectures, the C compiler prepends an underscore to the names of symbols in program code
  - This is the reason for rule #3
  - However, the Host Shell will match xx with _xx even when the target is not one of those for which the compiler prepends an underscore

Symbol Table Look Up

- Display symbols matching a regular expression with ***lkup( )***

```
[coreOS] -> lkup ("^task")
Symbol Table for firstPartition (PD ID 0xc7058)
taskName      0x200000e0 text  link  ---> coreOS
taskIdSelf    0x200001a0 text  link  ---> coreOS
taskDelay     0x200000c0 text  link  ---> coreOS
value = 0 = 0x0
[coreOS] -> lkup ("^x$")
Symbol Table for firstPartition (PD ID 0xc7058)
x             0x200053b0 bss              (shell)
value = 0 = 0x0
```

- Possible to list symbols of a partition
  - ***lkup( )*** must be used in Shared Library Domain whose name is name of partition suffixed by SI
- Global symbols and entry points for that partition

Slide 7

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- Some C compilers prepend an "_" to symbols declared in the source
- Use *lkAddr( )* to display symbols located near a particular address
- The Tcl WTX function *wtxSymListGet* can return a list of symbols matching a regular expression, and defined within a particular module
  - See the ***Tornado API Programmer's Guide*** or online documentation for details

Shell as a C Interpreter

Shell command syntax generally same as C programming language

```
[coreOS] -> x = (8 * 6) / 4
new symbol "X" added to "coreOS" symbol table.
X = 0x349680: value = 12 = 0x0c = _vx_offset_EXC_INFO_xer +
0x1 (PD NAME: coreOS)

[coreOS] -> Nom = "Nelson"
new symbol "Nom" added to "coreOS" symbol table.
Nom = 0x349660: value = 3446384 = 0x349670 = Nom + 0x10 (PD
NAME: coreOS)
[firstPartition] -> &x
value = 3446400 = 0x349680 = X
```

Slide 8

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- Any global variable can be manipulated
  - Examine the values of a program's variables
  - Change the values of variables
  - Create new variables
- By default, standard output from a target function called directly from the Host Shell appears in the Host Shell window

Shell as a C Interpreter

- Dynamic allocations are disabled within partition tasks:

```
[firstPartition] -> X= 2+4*8
Cannot allocate memory for new symbol in PD: 0xda81b0
can't add 'X' to "firstPartition" symbol table - error =
0x0.
[firstPartition] -> lkup "printf"
Symbol Table for firstPartition (PD ID 0xda81b0)
printf                    0x40000738 text
value = 0 = 0x0
[firstPartition] -> printf "%s\n", "Hello World\n"
Cannot allocate memory for new string in PD: 0xda81b0
```

Slide 9

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Task Information

- To display task information, use ***i(taskNameOrId)***

```
[coreOS] -> i (tNetTask)
  NAME       ENTRY       TID    PRI  STATUS      PC      ERRNO    PD ID
 --------  ---------  --------  ---  -------  --------  -------  --------
 tNetTask   _netTask   0xd2638   50   READY    0x5e554       0    0xb8c58
 value = 0 = 0x0
```

- If **taskNameOrId** = 0, display information for all tasks
- Use ***ti( )*** for detailed task information
- Other functions exist for viewing this information within partitions other than Module OS
- Partition specific
  - **partitionHelp**
  - **partitionTaskInfoShow**

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Executing Target Functions

- To execute functions on the target, either
  - Call function from the shell or
  - Use shell command ***sp***
- In either case, a new task gets spawned by WDB agent to run function
- Only possible in Module OS
  - In partitions, calls to target functions are not available unless a partition-based target shell is being used
  - Dynamic allocation is prohibited in partitions

Slide 11

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Syntax Quirks

- Target functions are always called (spawned) with 10 integer arguments
  - For example,

    `[coreOS] -> 2 * foo (&mac, 27)` is really

    `[coreOS] -> 2 * foo (&mac, 27, 0,0,0,0,0,0,0,0)`

- If the entire expression is a function call, with the function name first on the line, parentheses may be omitted
  - For example,

    `[coreOS] -> foo &mac, 27` but not

    `[coreOS] -> 2 * foo &mac, 27`

Slide 12

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

ARINC 653 and POSIX Specific Commands

These show commands give information on ARINC 653 and POSIX objects:

- ***pdShow***
- ***partitionShow (Module OS only)***
- ***arincSchedShow***
- ***apexBlackboardShow***
- ***apexEventShow***
- ***apexSemShow***
- ***apexBufferShow***
- ***apexProcessShow***
- ***apexQueuingPortShow***
- ***apexSamplingPortShow***
- ***partitionPosixSemShow***
- ***partitionPosixMqShow***
- ***partitionPosixTimerShow***

Slide 13

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- `partitionShow` is a target based command when the show routines are included
  - See **`target/src/os/pd/partitionShow.c`**

Partition Specific Commands

- These show commands give information on tasks (threads) and memory usage within each partition:
  - ***partitionMemPartShow***
  - ***partitionMemShow***
  - ***partitionTaskCheckStack***
  - ***partitionTaskInfoShow***
  - ***partitionTt***
  - ***partitionW***
- Commands allow for in-depth partition visibility without need for a partition based target shell
- Qualified tools described in the documentation
  - **windsh653Qualified** library

Slide 14

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

HM Specific Commands

- Health Monitor can also be displayed using shell

```
[coreOS] -> hmLogEntriesShow 1
Partition number: 1
Event: 1
-----------------
Code                = PARTITION_MODE_SET (13)
subCode             = (0)
historicalCode      = 0
addInfo             = 1
addr                = 0x89dbc
level               = HM_PARTITION_LVL (3)
timestamp (ticks)   = 0x7a5f72
sysStatus           = HM_SYSCALL_STATUS (128)
partNumber          = 1
taskName            = tRstfirstPartition
taskId              = 0x237018
message             = Initial partition COLD start
```

- Parameter in ***hmLogEntriesShow*** represents partition number in configuration record
  - 0 for the Module OS HM log

Slide 15

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Debugging

- **Host Shell**
  - Basic Features
  - **Advanced Features**
- Kernel Objects Viewer
- Source Level Debugger
  - Task Mode Debugging
  - System Mode Debugging

Slide 16

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Advanced Features: Examining Target Variables

- Shell interprets all variables as 32-bit integers unless specified otherwise
  - Casts must be used to tell shell how to interpret non-integer types

```
[coreOS] -> z = (short) 0x90
z = 0x20005360: value = 144 = 0x90 (PD NAME:coreOS)
[coreOS] -> (short) z
z = 0x20005360: value = 144 = 0x90 (PD NAME:coreOS)
[coreOS] -> z
z = 0x20005360: value = 9437184 = 0x900000 (PD NAME:coreOS)
```

- All variables created in the Host Shell are allocated 8 bytes of target memory, to preserve space in case the variable's value is changed to a double

Slide 17

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- Floating-point values are stored as doubles

  [firstPartition] → x = 9.1

  x = 0x20005360: value = 9.1

  [firstPartition] → (double) x

  x = 0x20005360: value = 9.1
- Host Shell casts are used to assign type information to an expression
  - In C, casts are used to convert one type to another
- The type information specified in a Host Shell cast is not retained
- Shell built-in functions screen target symbols of the same name
  - To access the target symbol rather than the built-in, prefix its name with '@'

  [firstPartition] → @i = @i + 3

Caveats

- Shell does not recognize symbolic macros defined for C preprocessor with #define directives
  - Examine header files for structure and symbol definitions
- Shell does not recognize data structures
  - Use a source-level debugger to view and manipulate structures
- Use ***d( )*** to display memory
  - Useful for examining shared data (virtual address) regions

```
[myApp] -> d pMyBuf
Dump memory for myApp (PD ID 0xc0478)
0x200053b0:  4865 6c6c 6f20 7374 7564 656e 7473 2100   *Hello students!.*
0x200053c0:  0000 1008 0000 ec30 2000 4010 2000 4010   *.......0 .@. .@.*
0x200053d0:  0000 0000 0000 0000 0000 0000 0000 0000   *................*
0x200053e0:  0000 0000 0000 0000 0000 0000 0000 0000   *................*
0x200053f0:  0000 0000 0000 0000 0000 0000 0000 0000   *................*
0x20005400:  0000 0000 0000 0000 0000 0000 0000 0000   *................*
```


© 2008 Wind River Systems, Inc.

Student Notes

- The *d( )* shell built-in can display data in 1, 2, 4 or 8 byte units
  - The data is displayed conventionally, with the most significant byte of a unit displayed on the unit's left
  - On little-endian machines, this means that for units larger than 1 byte, the order in which the bytes are displayed left-to-right in the memory dump is not the same as the order in which they occur in memory
    - There is a reflection of the bytes in each unit
- Use *m( )* to modify memory starting at a particular address

© 2008 Wind River Systems, Inc.

Shell Line Editing

- Shell provides two styles of line editing, vi or emacs
- To switch to emacs-style line editing, type

```
[coreOS] -> ?shConfig LINE_EDIT_MODE emacs
```

- Type ***h*** to see a list of previous commands

- vi-style
  - Use "Esc" to enter line editing mode, then command
  - For example,
    - `k (j)` to get previous (next) shell command
    - `h (l)` to go left (right) one character
- emacs-style
  - Use up (down) arrow key to get previous (next) shell command
  - Left (right) arrow key to go left (right) one character

Slide 19

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- By default, there is space in the history list for 20 commands
  - To increase this, call ***h( )*** with a parameter
  - For instance,
  - **[firstPartition] -> h 500**
  - increases the history capacity to 500 commands
- A complete list of Host Shell line editing commands is available in the *Tornado User's Reference*

Shell Special Characters

Character	Description
^C (UNIX) ^-Break (PC)	Terminate a function call and return shell prompt
^X	Reboot VxWorks target
^D, TAB	Symbol and filename completion; command synopses
^W	Show HTML help for command
Esc	Enter Vi-like editing mode from input mode (if vi-style is selected)
^U	Delete ("kill") current line, return to normal input mode

Slide 20

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- Use the command or filename completion and synopsis help keys as follows

```
[firstPartition} -> semT                                    type ^D
_semTake      _semTakeTbl     _semTerminate
[firstPartition] -> semTa                          type TAB
[firstPartition] -> _semTake                       type <space> ^D
semTake() - take a semaphore

STATUS semTake
 (
    SEM_ID semId,  /* semaphore ID to take */
    int    timeout /* timeout in ticks */
   )
[firstPartition] -> _semTake (mySem,               type ^W to see documentation
```

- Pathname completion is similar
  - It is only attempted if no target symbol or built-in matches the string, so it is a good idea to start relative paths with "./"

Shell Primitives and Target Resources

Shell built-in commands execute as C-interpreter primitives on the host when possible

- For example,
  - Built-in ***lkup( )*** executes entirely on host
  - No communication with the target is required

Slide 21

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Shell Basic Level Debugging

- Use ***b( )*** to set a task breakpoint
- Use ***s( )*** / ***c( )*** to step/continue a task
- If a task halts unexpectedly, ***tt( )*** can be used to find out which routine failed

```
[myPartSl] -> b (printf)
value = 0 = 0x0
[myPartSl] ->
Break at 0x0002cb10: printf Task: 0x000d6018 (s1u0)
                              PD   : 0x000c6798 (myPartSl)
[myPartSl] -> tt()
0x20014034 stuff          +24 : printf ([])
value = 0 = 0x0
[myPartSl] -> c
value = 0 = 0x0
```

Slide 22

© 2008 Wind River Systems, Inc.

WIND RIVER

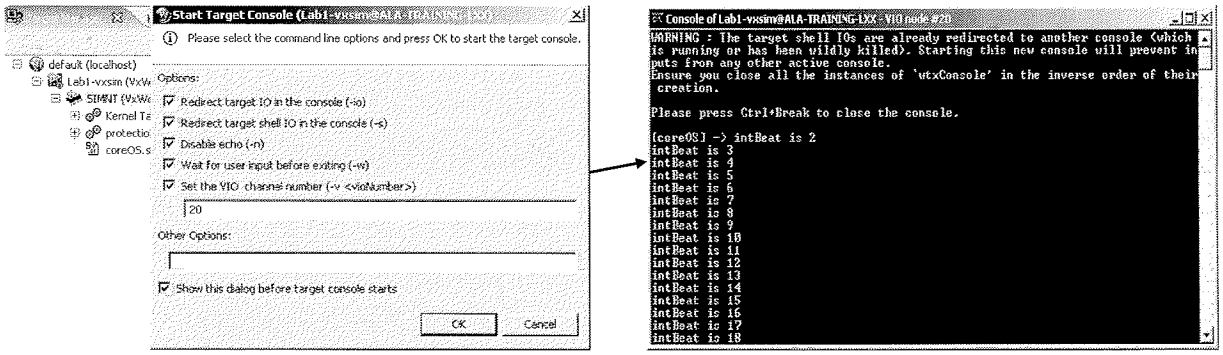
Student Notes

- Use the Workbench Debugger for full source-level debugging
- The output from *tt( )*
  - The value on the left is the address of the current position in the subroutine
  - The + value after the function name is the offset from the beginning of the subroutine (in hexadecimal)
  - A hexadecimal dump of the subroutine arguments is shown

Function	Description
b	Set breakpoint/list breakpoints
bd	Delete a breakpoint
bdall	Delete all breakpoints
s	Single step
l	Disassemble code
tt	Stack trace
c	Resume execution from breakpoint

Virtual I/O

- By default, a target's standard input, output, and error are directed to console used to set boot parameters (usually connected to the serial port on board)
- Target I/O can be redirected to a virtual console using target server - WDB agent communication link (useful if target lacks a serial I/O channel or if developing remotely)



Slide 23

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- Once a virtual console has been started, it can be accessed from the target like a normal character (stream) I/O device
- Used as part of the Port Monitoring tool

Shell Host I/O Redirection

- Shell input can be redirected to come from a file

```
[coreOS] ->  < script
```

- Shell output can be redirected to a file

```
[coreOS] -> h( ) > hfile
```

- Caveat: I/O can only be redirected to files accessible to shell process running on development host, not to local devices on target

Slide 24

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- File names after a shell redirection symbol (< or >) are not quoted
- Expressions involving the relational < and > operators may have to be enclosed in parentheses so that the shell does not consider them to be I/O redirections
- When redirecting input to the shell from a script, or when cutting and pasting multiple lines into the Host Shell, *SH_GET_TASK_IO* should be off

Debugging

- Host Shell
  - Basic Features
  - Advanced Features
- **Kernel Objects Viewer**
- Source Level Debugger
  - Task Mode Debugging
  - System Mode Debugging

Slide 25

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Kernel Objects Viewer Overview

- Graphical tool for displaying target information
  - Kernel objects
  - Kernel tasks
  - APEX objects such as partitions, processes, semaphores, queues, ...
  - Drivers, File Descriptors, and Device objects
- To use Kernel Object Viewer
  - Select Target Connection in Target manager
    - Right click on CPU, and select "Kernel Objects" or
    - Select "Kernel Objects" tab in lower middle pane

Slide 26

© 2008 Wind River Systems, Inc.

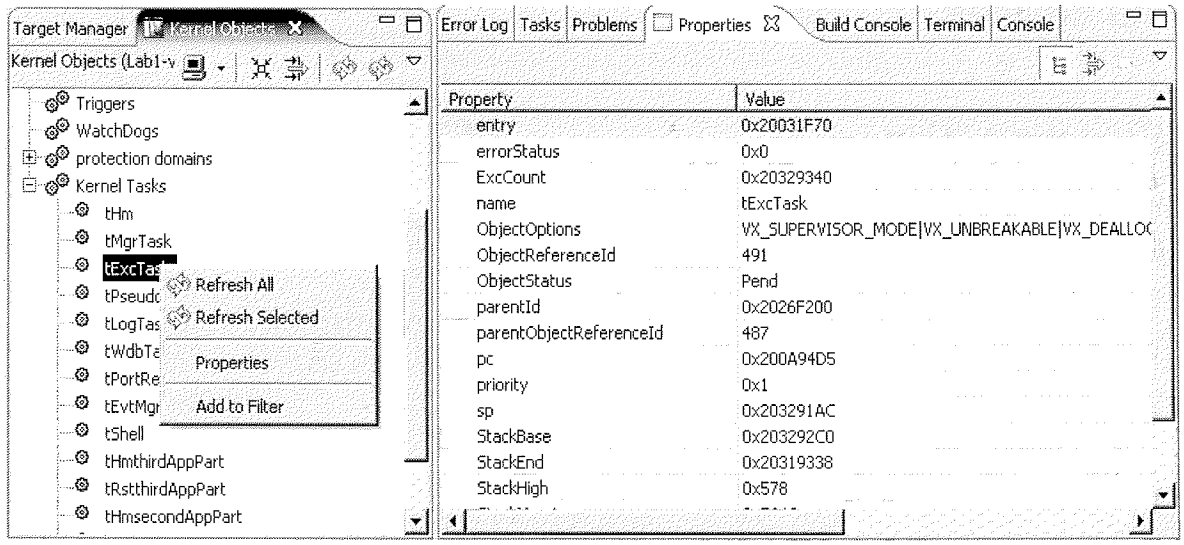
WIND RIVER

Student Notes

Object Properties

To inspect the properties of an object

1. Click on the object.
2. Select "Properties" from the context menu for the instance.



Slide 27

© 2008 Wind River Systems, Inc.

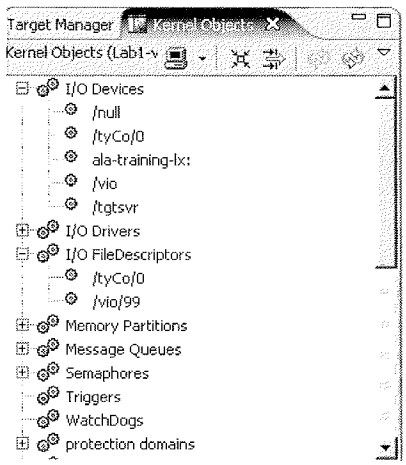
WIND RIVER

Student Notes

- Opening an object in the Tree View also shows all owned objects in a hierarchical tree

I/O System Information

- Selecting "I/O Devices" shows all known system I/O devices
- Selecting "File Descriptors" shows all open file descriptors



Slide 28

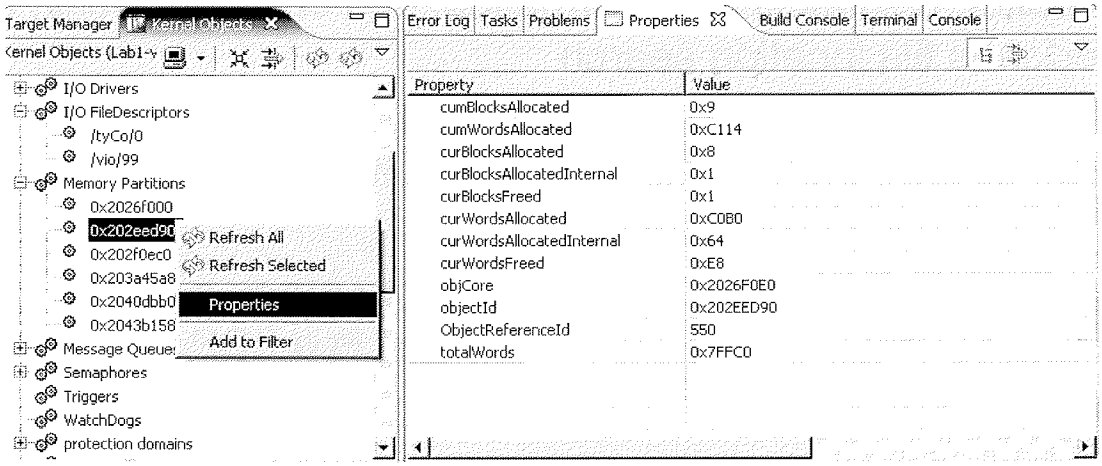
© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Memory Partition Information

1. Click to select a memory partition.
2. Select it's properties to see the statistics and information about the partition.



Slide 29

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Browsing Symbol Information

- Symbol browsing is able to display where symbol is defined in editor, displays type information, and calls hierarchy, and performs file location search
- Functions available from Navigate menu tab (select the icons pictured above, or use the "Symbol Browser" tab next to the "Project Navigator")
- Browser will only have information for source code in current active project. For example, if project is a BSP build, browser will not have information on internals of network stack.

Slide 30

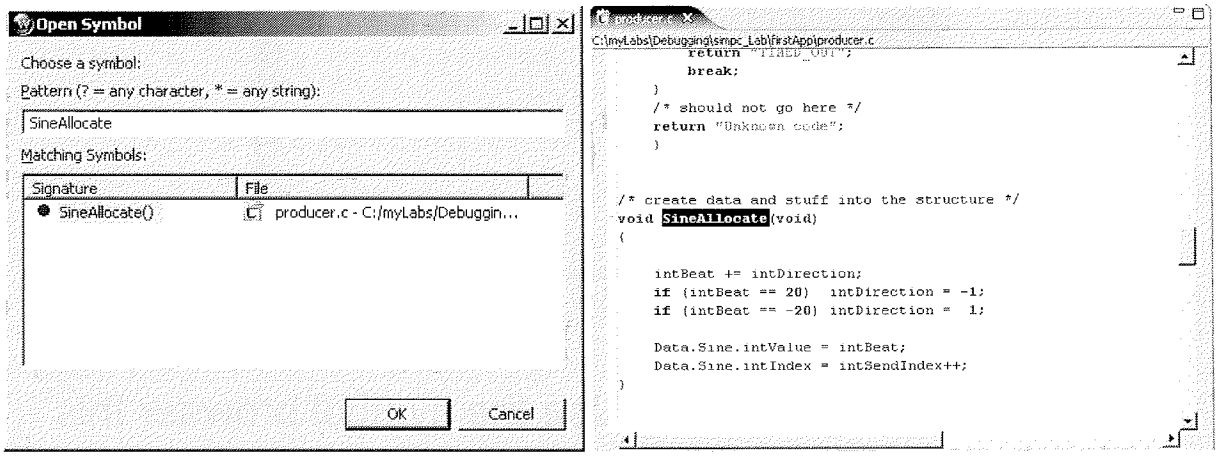
© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Symbol Browser – Edit Symbol

- Open the "open symbol" window from the Navigate menu or the "Open Symbol" icon. Enter the symbol you want to edit and click OK. An editor window will open on the symbol.



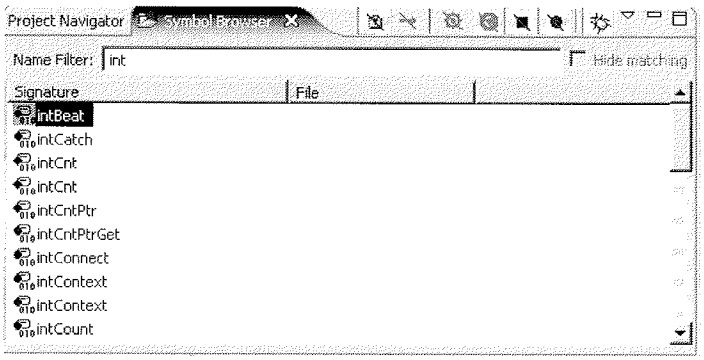
© 2008 Wind River Systems, Inc.

Student Notes

© 2008 Wind River Systems, Inc.

Symbol Browser tab

- Select the symbol browser tab. Look for "int":



- Double-click on "intCount". Use the symbol browser this way to find symbols to set breakpoints on.

© 2008 Wind River Systems, Inc.

Student Notes

© 2008 Wind River Systems, Inc.

Symbol Browser – Type Hierarchy

- To display the type hierarchy of a given type, in the Navigate menu select "open type hierarchy" or the icon as pictured above.
- Enter or select the type of interest, click ok.
- In the lower right corner, the type hierarchy pane will open.

Slide 33

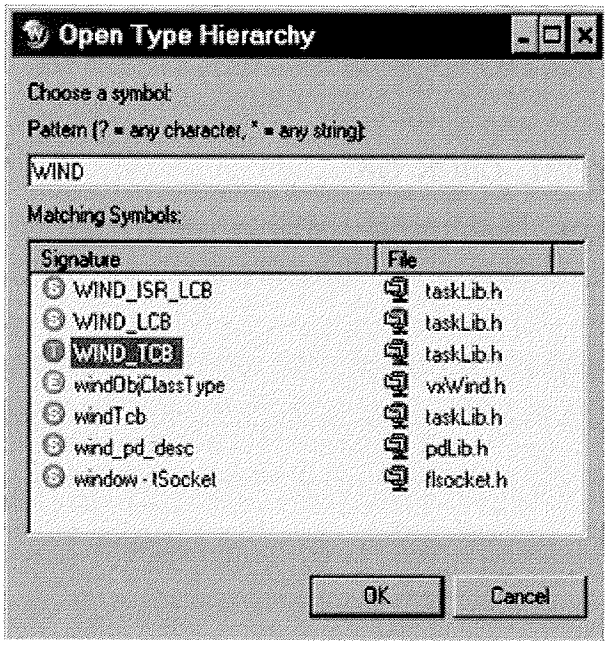
© 2008 Wind River Systems, Inc.

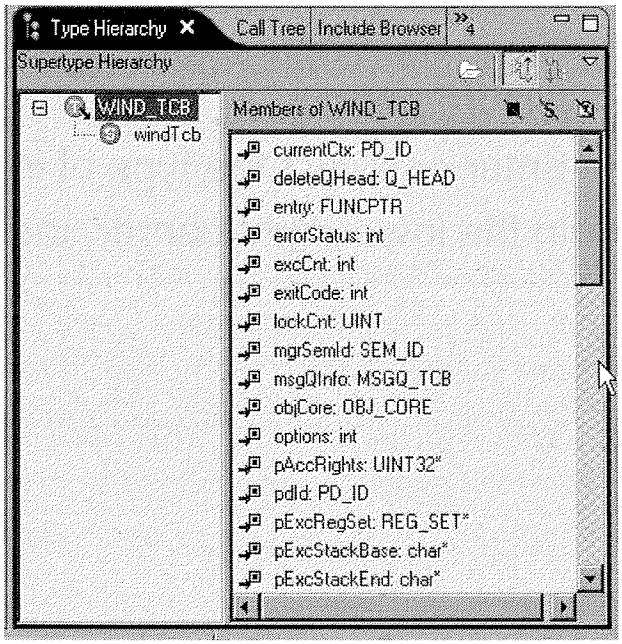
WIND RIVER

Student Notes

- VeroStack, from Verocel, will build out Static Analysis

Symbol Browser – Type Hierarchy (Continued)





Slide 34

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- VeroStack, from Verocel, will build out Static Analysis

Symbol Browser – Call Tree

- To display a call tree, select "open call tree" from the Navigate menu, or click the "open call tree" icon.
- The "open call tree" dialog will open.
- Select a symbol, then click OK.
- In the lower left corner the Call Tree pane will open with several display options in it's tool bar.

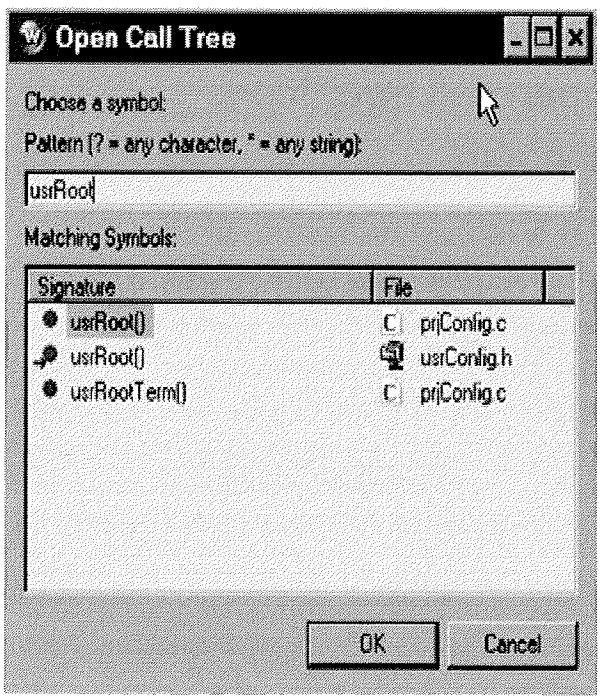
Slide 35

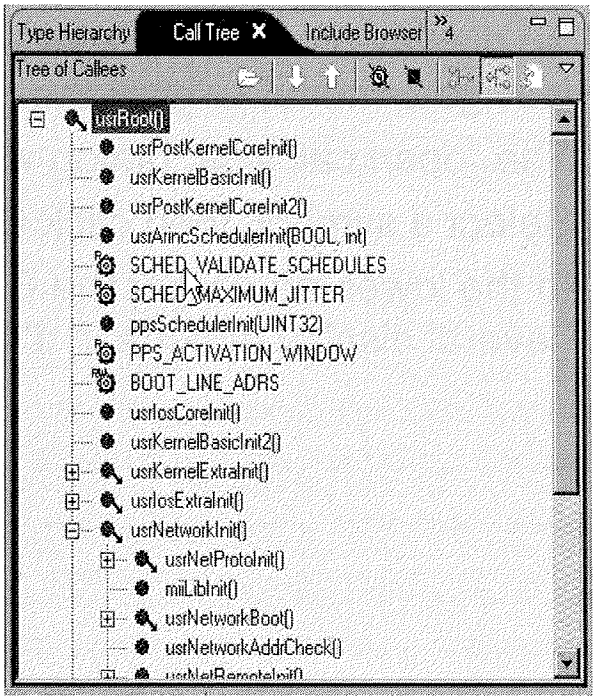
© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Symbol Browser – Call Tree (Continued)





Slide 36

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Debugging

- Host Shell
  - Basic Features
  - Advanced Features
- Kernel Objects Viewer
- **Source Level Debugger**
  - Task Mode Debugging
  - System Mode Debugging

Slide 37

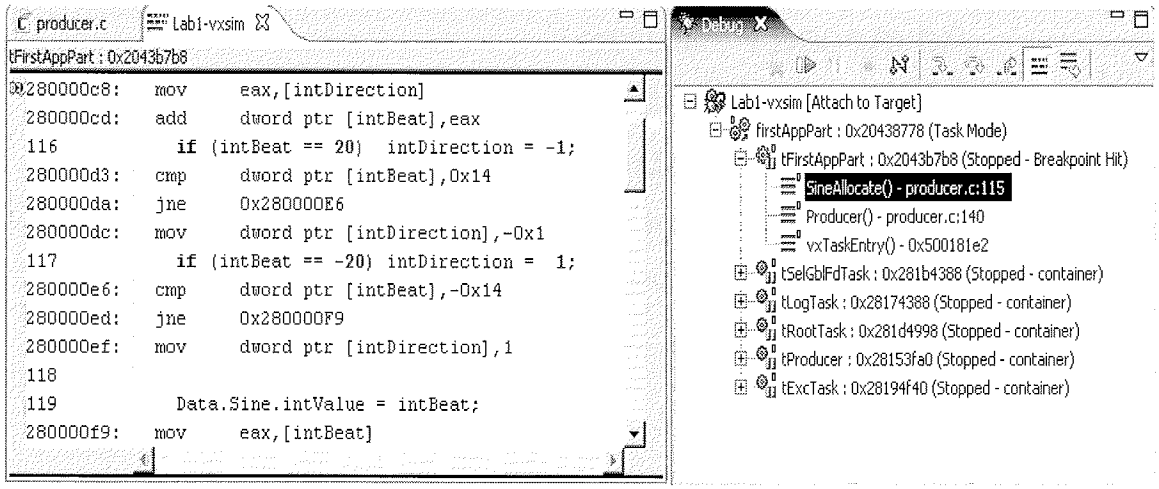
© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Source Level Debugger Overview

- Debugger enables source and assembly level debugging
- Simultaneous debug of multiple tasks - graphical user interface (Debugger Control and additional Views)



- Debugger uses WTX protocol to communicate with target server
- GUI interface starts automatically when connected to target
  - Debugger attaches to the partition, not the task/process

Slide 38

© 2008 Wind River Systems, Inc.

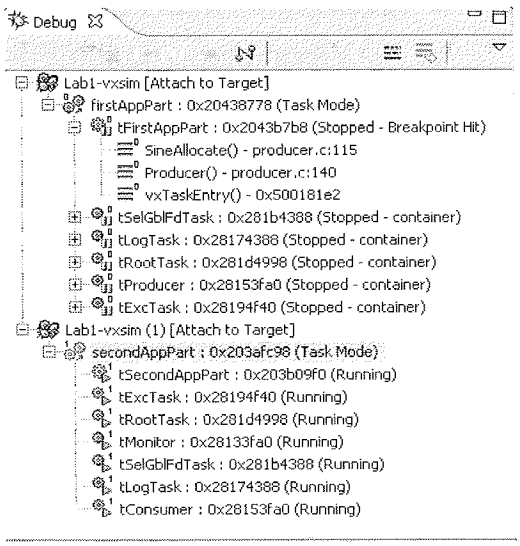
WIND RIVER

Student Notes

- WTX: Wind Tool eXchange

Debugger Control View

- Used to set up and control applications to be debugged
- Shows partitions, tasks and status (attached, running, and so on)
- Supports context sensitive actions
- Presents color-coded tasks
- Possible actions include
  - Read in debug/symbol info
  - Set breakpoints
  - Stop and continue task--or partition
  - Open source code
  - Dump memory
  - Display function names



Slide 39

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- Tasks to be debugged can be spawned with any tool capable of spawning tasks
- Task properties (e.g. stack size, priority, options) can be set using the graphical interface debugger
- Color-Coding is used in the watch windows

Task and System Level Debugging

- Task mode debugging
  - Can debug tasks only, not ISRs
  - Breakpoints affect only one task or subset of tasks
    - When stopping the Partition OS, all tasks within that partition are also stopped
  - With tasks stopped, other tasks and ISRs continue executing
  - Communication with WDB agent is interrupt driven
  - Effective only when debugging a single partition
- System mode debugging
  - Can debug tasks, ISRs, and pre-kernel execution
  - Default breakpoints stop the whole system
    - Task-specific breakpoints can be set
  - With system stopped
    - External WDB agent runs with interrupts locked
    - Communication with the WDB agent is in polled mode

Slide 40

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- To debug at system mode over Ethernet requires an END network interface driver, which supports polled mode communication
- System mode is the preferred mode
  - No 'Hole in Time'

Debugging the Partition OS

- From debugger GUI only one task (***tPartitionName***) in each partition is seen (this task represents partition OS running within partition space)
- This task runs all processes of the partition
  - Includes ARINC processes, POSIX threads or native Partition OS (vxWorks) tasks
  - Side effect is that a task mode breakpoint will stop entire partition
- Processes of a partition can be debugged by stopping ***tPartitionName*** and/or setting breakpoints in ***tPartitionName***
- Breakpoints can be set at specific function entry/exit points or at statements within a function that is in partition
- You can directly view processes (threads) running within partition OS using Debugger GUI

Slide 41

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- 3 tasks get created in the MOS for each partition
  - **tPartitionName**
  - **tHmPartitionName**
  - **tRstPartitionName**

Debugging Within a Partition

- To debug a partition application from startup:
  1. Start system with a schedule that does not allot time to the partition
     - A debug configuration is set within schedule in configuration record that only allots time for Module OS (partition 0) for first schedule entry)
  2. Place breakpoints at function entry/exit point or within a function in partition and then select either Task Mode or System Mode
  3. Switch to normal schedule used for the system, using **arincSchedSet( ).** Breakpoints that are then encountered will be displayed along with source code
- All the normal debugging operations such as step, step to end of function, or display variables can then be performed

Slide 42

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Debugging with Health Monitoring

- When debugging, interactions can occur with HM system that can cause unexpected results such as reboots or tasks not running (being suspended)
- Typical issues include:
  - In Task mode, stopping Partition OS task can result in task being suspended after Debugger has resumed execution
    - Can be caused by having ARINC timeout errors (such as deadlines) occurring in partition
    - Partition OS task can be resumed by using ***taskResume( )*** from host shell or replacing Partition HM handler for debugging
  - In System mode, system can reboot due to lost ticks being reported
    - Avoid this by replacing Lost Tick Handlers with a `NULL` handler for debugging
- Other interactions possible based on system configuration

Slide 43

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Debugging

Host Shell
- Basic Features
- Advanced Features

Kernel Objects Viewer

Source Level Debugger

Task Mode Debugging

System Mode Debugging

Slide 44

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Task Mode Debugging: Starting a Debug Session

- To generate debug symbols
  - Compile code with the **-g** option specified
  - Specifying **-O0** is also recommended
- To invoke the GUI based debugger
  - Select a target in the Target Manager
  - Right-click a kernel task to debug, select **Attach to Kernel Task**
  - Right-click on a partition, select Attach to Protection domain task
- Spawn a new task
  - Right click a kernel task, select Run Kernel Task
- To exit the debugger
  - Click on the "disconnect debugger" icon

Slide 45

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- To start the debugger from the command line, use `tcdb tgtsvr-name`
- The GNU compiler allows specifying both debugging info (`-g`) and optimization (`-O` flags)
  - Compiling with **-g** and no **-O** flags is equivalent to compiling with **-g -O0**
  - Compiling with aggressive optimization (**-O2** or higher) when debugging can result in confusion, both to the debugger program and the debugger user
  - Except in rare cases when timing changes from compiler optimizations are involved in the problems seen, it is best to debug code with no optimization
- The debugger uses ELF/DWARF II file and debug format readers

Debugging Tools

- Examples of debugger functionality
  - Set/delete breakpoints
  - Step to next line of code
  - Step over a function call
  - Continue program execution
  - Finish current subroutine
  - Move up or down stack frames
  - Monitor variables or other expressions
  - De-reference pointers
- For more information about debugging tools, see online help, Wind River Documentation → Guides → Host Tools → Wind River Workbench User's Guide 2.6.1, VxWorks 653 Version, Chapter 5

Slide 46

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Breakpoints

- Each breakpoint has an associated scope and action
- Scope defines which task can hit the breakpoint
  - One specific task (for the kernel domain)
  - All tasks in a particular domain, since Partition OS is also stopped
  - All breakable tasks (for kernel domain)
- Action defines what happens when a breakpoint is hit
  - Task encountering breakpoint will halt
  - All tasks in a partition of the task hitting the breakpoint will halt
  - All breakable tasks in Module OS will halt (if breakpoint is in Module OS)
  - A function will be executed (Module OS only)

Slide 47

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Wind River ICE

Specialized Breakpoints

- Hardware breakpoints on some architectures
  - Halts on instruction or various sorts of data access to specified address
    - Limited in number
  - Only available with hardware support: PPC 603, 603e, 604, etc.
- Temporary breakpoints
  - Effective once, then deleted (default) or disabled
- Conditional breakpoints
  - Effective if specified expression is true, otherwise task resumes
- Function entry (exit) breakpoints
  - Can be set before tasks are started
  - Are fixed in scope (partition) and action (halt specific task)

Slide 48

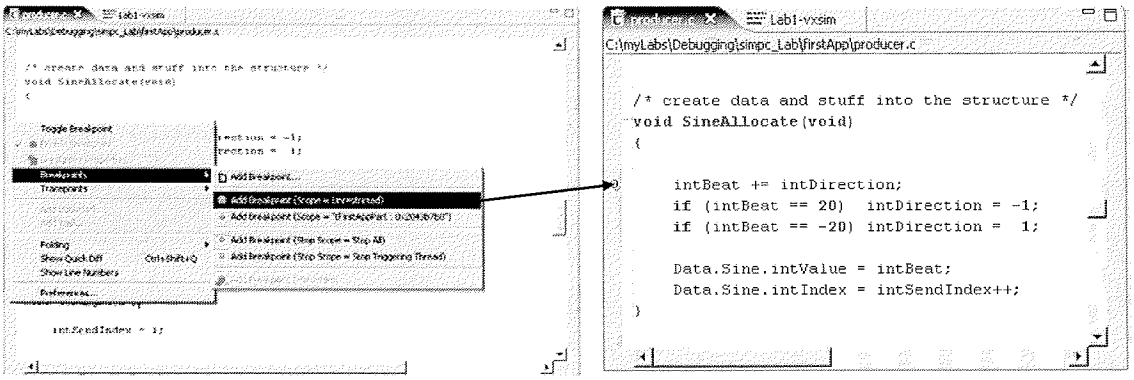
© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Setting Breakpoints

1. Select a target session in the Target Manager. Select a task in kernel partition – right click on it, and select **Attach to Kernel Task (Task Mode)**.
2. With the source open in the edit pane, in the left gutter, right-click where you want the breakpoint set:



© 2008 Wind River Systems, Inc.

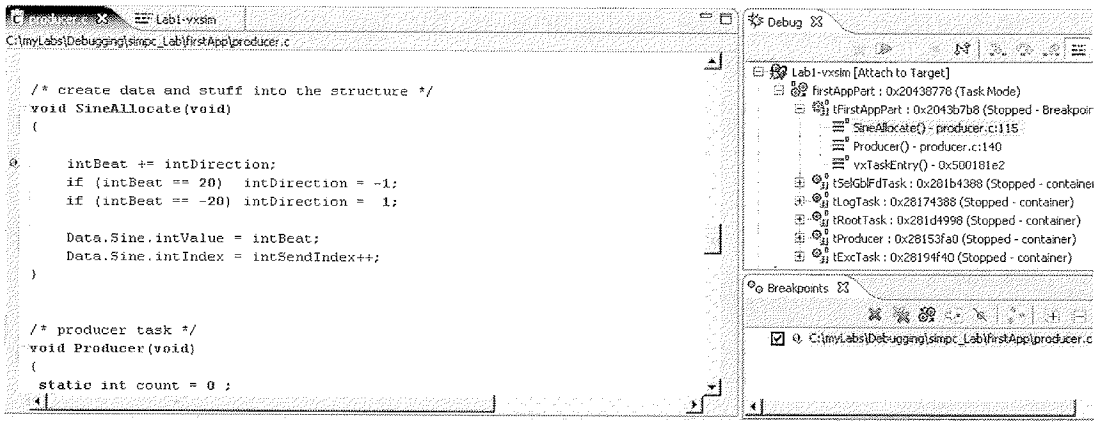
Student Notes

- To open the source view for a task, right-click on the task in the "Debugger Control" view and choose "Task Source" from the context menu
- Function entry and exit breakpoints can also be set in the function view

© 2008 Wind River Systems, Inc.

Breakpoints

- If the task is running, the breakpoint will be hit
  - The task has stopped, and it's stack trace is displayed



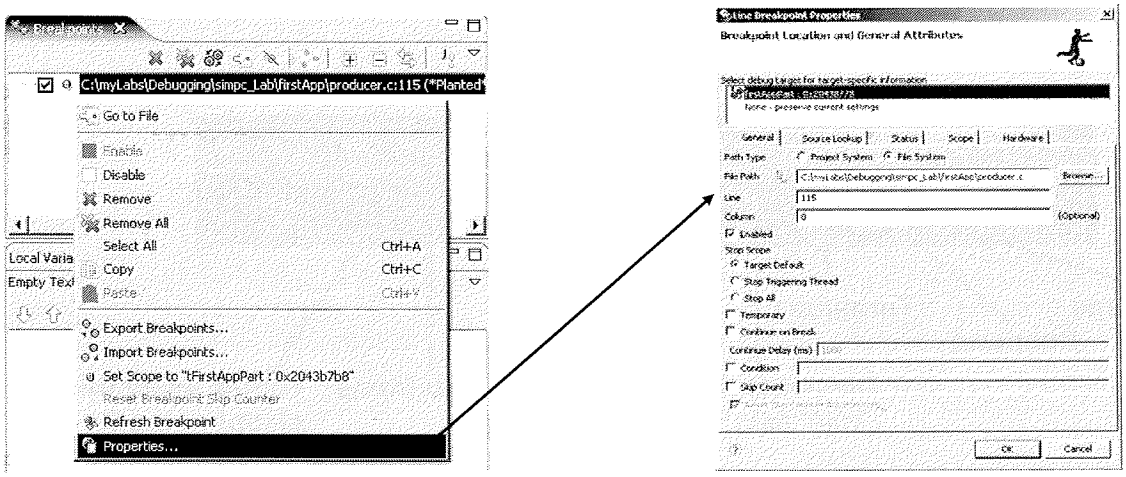
© 2008 Wind River Systems, Inc.

Student Notes

© 2008 Wind River Systems, Inc.

Listing and Modifying Breakpoints

- To list all breakpoints open the **Breakpoints** pane
- To display and modify breakpoint properties right-click on the breakpoint, and then choose "Properties"



© 2008 Wind River Systems, Inc.

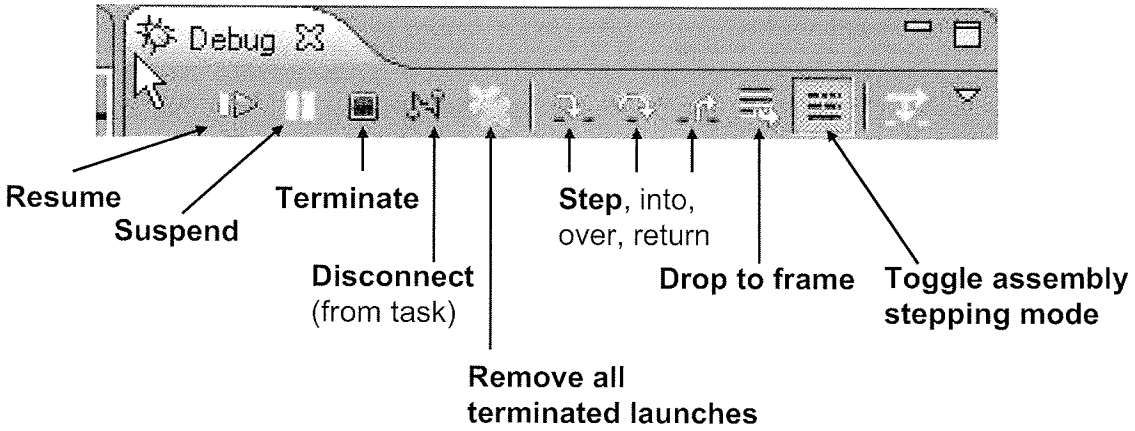
WIND RIVER

Student Notes

- In the command line debugger, use "info breakpoints"

Controlling Task Flow

- A task stopped by debugger is in "break" state
- To control task flow, use
  - "Debugger Control" view for basic control
  - "Source" view for advanced control



Slide 52

© 2008 Wind River Systems, Inc.

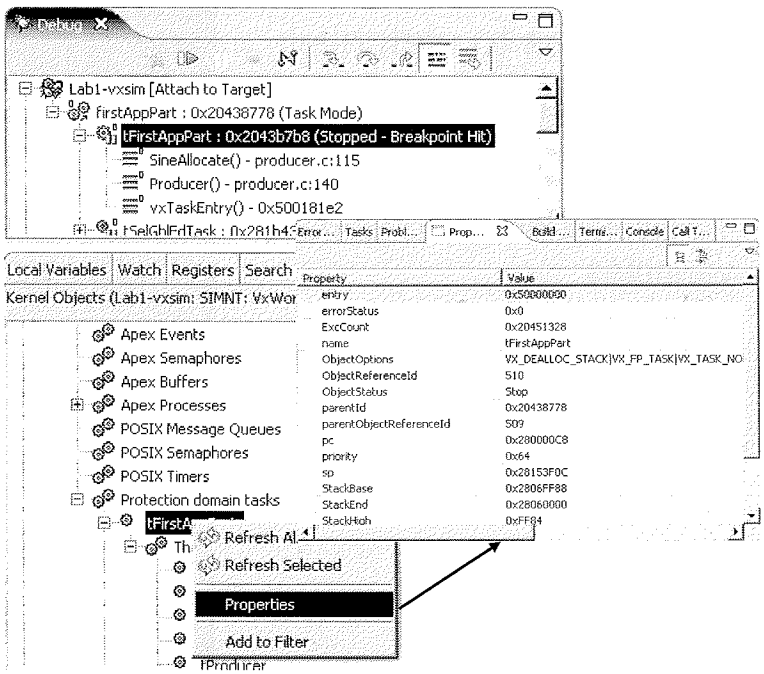
WIND RIVER

Student Notes

- In the command line debugger, use "halt" to stop a task, "continue" to continue a task or domain, and "step" to single-step
- The "next" command is commonly called "step over", since it steps over a function call
- The "step" and "next" buttons step by assembly instruction when assembly is shown (mixed or assembly display mode, or when there is no source)
  - At the command line, use "stepi" for stepping a single assembly instruction
- "Continue" continues execution until either a breakpoint is hit, the stop button is used, or the task exits
- "Finish" continues until execution leaves the selected stack frame, or until a breakpoint is hit
- The "stop" button can be used to stop the highlighted task or PD (task mode), or the entire system (system mode) if it is executing without hitting any breakpoints

Examining Task Stacks

- Click on task in debugger view to view stack trace (task must be suspended)
- Use kernel objects view to obtain stack information, and so forth



© 2008 Wind River Systems, Inc.

WIND RIVER

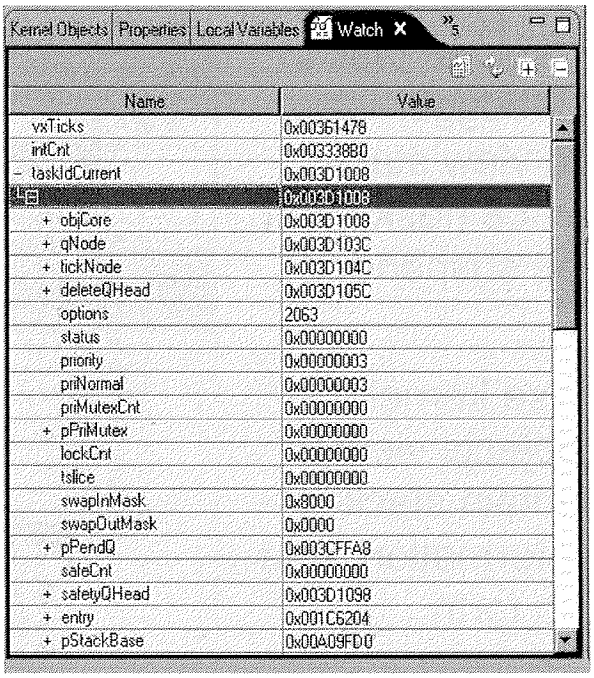
Student Notes

- The command line debugger provides a variety of stack manipulation routines

`print-stack`	Trace the stack of the current task
`print-frame`	Show the current stack frame
`up` ***n***	Move up ***n*** frames
`down` ***n***	Move down ***n*** frames

Monitoring Variables and Expressions

- Add variables to watchlist
- Set refresh mode
- Open other windows
  - To display local and global variables
  - To display memory contents



© 2008 Wind River Systems, Inc.

Student Notes

- Common information commands for the command line debugger

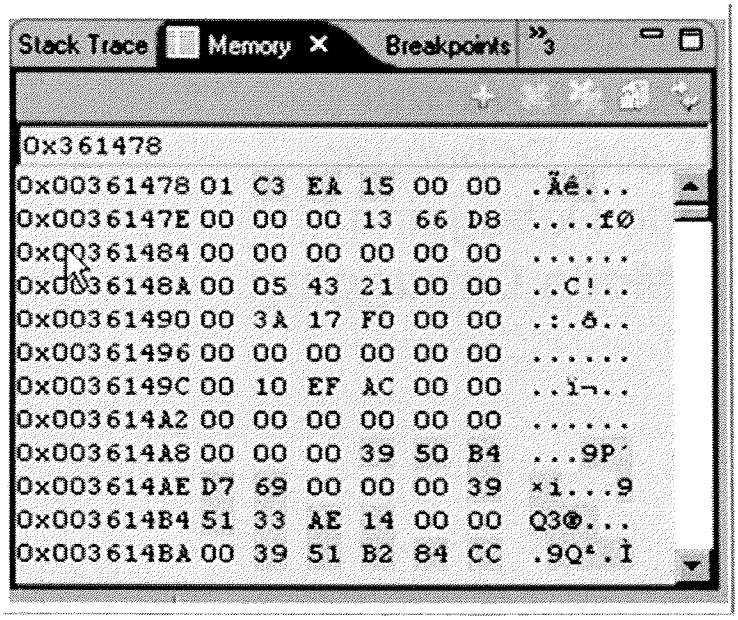
`print expr`	Display current value of ***expr***
`display expr`	Display value of ***expr*** each time task is stopped
`info variables`	Show global variables
`print-stack`	Stack frame listing
`display-regs`	Display registers each time task is stopped
`info breakpoints`	List breakpoints and their status
`info task`	List of tasks in current PD
`info pds`	List of PDs

- Use the "help" command for more information

© 2008 Wind River Systems, Inc.

Additional Views

- Use **Window → Show View** to open other windows such as (virtual) memory display



© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- The Memory View allows one to display and modify memory
  - All addresses are virtual addresses.
- The Registers View shows the value of the registers for the selected task
- The Functions View lists all functions that have been downloaded into the specified domain and allows one to set breakpoints or open the source code for a selected function
- The Module View displays all modules downloaded to the selected domain and allows one to set breakpoints or edit functions
- The Class List View lists all classes and allows one to open relevant source files
- The Class Hierarchy View graphically displays inheritance relationships of classes
- The Function Call History View records the (member) functions that have been invoked
- The Object Relationships View displays the relationship between objects, such as reference, creation, instantiation and creation order

© 2008 Wind River Systems, Inc.

Debugging

Host Shell
- Basic Features
- Advanced Features

Kernel Objects Viewer

Source Level Debugger

Task Mode Debugging

System Mode Debugging

Slide 56

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

System Mode Debugging

- System mode debugging supports
  - Debugging ISRs
  - Debugging before kernel is started
  - Multitask debugging
- In system mode, VxWorks 653 and all application tasks and ISRs stop when a breakpoint is hit
- System-Mode debugging SAVES and RESTORES system time, eliminating "hole-in-time" problem when debugging
- External WDB agent runs with interrupts locked when VxWorks 653 is stopped
- VxWorks 653 configuration must include WDB system debugging component

Slide 57

© 2008 Wind River Systems, Inc.

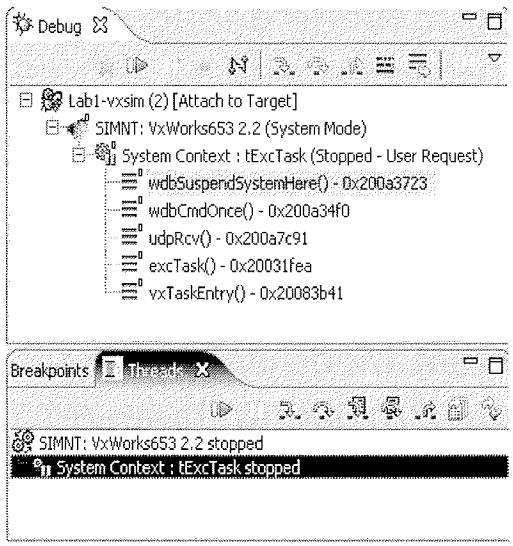
WIND RIVER

Student Notes

- System level debugging requires a communication path which supports polled-mode communication by the external WDB agent
- To debug over ethernet requires an END driver, which supports this mode
- If a serial connection is being used, the highest possible data rate supported by both host and target should be used
- Unless debugging before the kernel is started, the WDB task debugging component can also be included
- Hole-in-time – if the system time were not saved and restored while the system was suspended, from a system point of view, there would be a pause in processing – a "hole in time" where no processing was done.

Switching to System Mode

- To enter system mode, right click on the target in the **Target Manager**. From the menu, select **Attach to kernel, system mode**.
- Click ⏸ in system mode to halt the entire system
- Task trace displayed for halted system



Slide 58

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- In the Command Line Debugger, use "set system on"
- The Host Shell commands *sysSuspend( )* and *sysResume()* can be used to suspend or resume the system, transferring control to or from the external WDB agent
  - One should be cautious using the Host Shell to start or stop the system this way under circumstances in which the debugger might think itself in control of the system state

Debugging in System Mode

- Use normal debugger features to debug applications in system mode.
- Single-step through code
  - When system continues, VxWorks controls scheduling
  - Same tasks may not resume after a step, next or finish command
    - Such commands may set breakpoints and resume system
  - System time is saved and restored, so system isn't affected by pause in processing (no "hole-in-time" problem)
- Add locations to watch windows
- Open debugger windows
  - Stack, memory, registers

Slide 59

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Breakpoints in System Mode

- Various types of breakpoints are available in system mode
- Scope of breakpoints in system mode is selectable
  - One task
  - All tasks in the Module OS
  - All tasks system wide
- Action of breakpoints in system mode is fixed
  - Halt system and/or log an expression

Slide 60

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Questions

1. The current working partition within the shell can be any type. True or False ?
2. The Kernel Objects Viewer is used to display
   a. Kernel tasks
   b. APEX objects such as partitions, processes, semaphores, queues, ...
   c. Drivers, File Descriptors, and Device objects
   d. All of the above
3. In system mode debugging, default breakpoints stop only one task. True or False?
4. In task mode debugging, one can debug;
   a. Tasks
   b. ISRs
   c. Pre-kernel execution
   d. None of the above
5. When debugging within a partition, how can the schedule be changed?

Slide 61

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

1. True.
2. D
3. False
4. A
5. arincSchedSet()

Review

In this chapter you learned:

- Wind River Workbench source-code debugger
  - Executes on host while code executes on target
- Code must be compiled with `-g` option.
  - For easiest debugging, also specify `-O0` (or no `-O` flags)
- System-level debugging
  - Useful for debugging ISRs and BSPs
  - Useful for multitask debugging
  - VxWorks 653 and application freeze when breakpoint is hit
  - WDB agent uses polled-mode communication with the system suspended

Slide 62

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Visualization and Monitoring

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Objectives

By the end of this chapter you will be able to:

- Configure the System Viewer
- Use the System Viewer
- Use the Monitoring tools to monitor performance, memory usage and ports.

Slide 2

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Visualization and Monitoring

Wind River System Viewer
- **Introduction**
- Configuring System Viewer
- Using System Viewer

Monitoring Tools

Slide 3

© 2008 Wind River Systems, Inc.

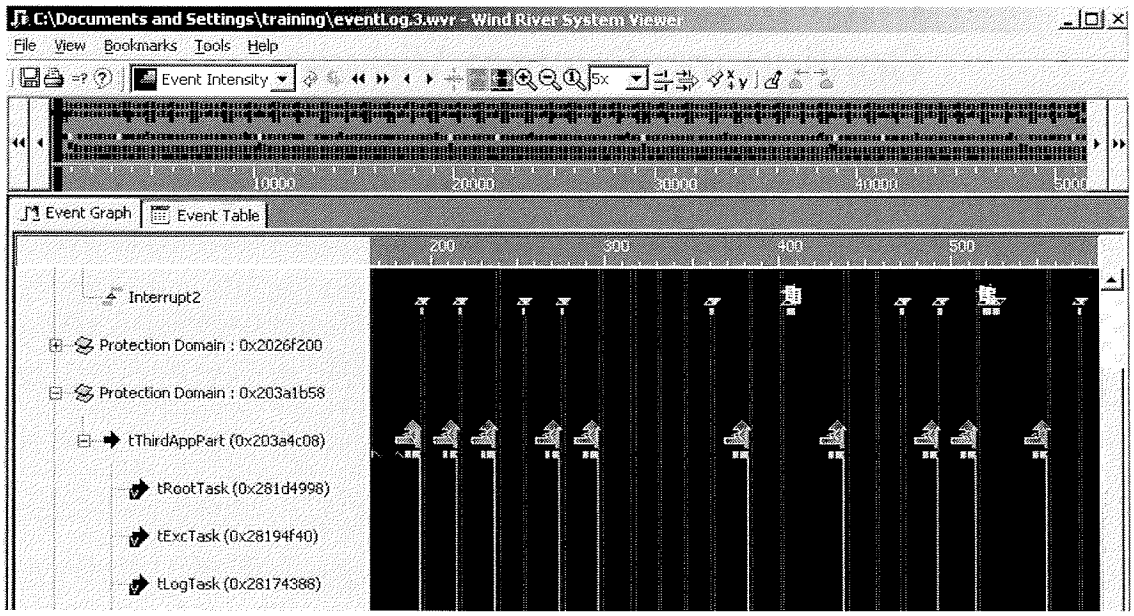
WIND RIVER

Student Notes

Not qualified

Wind River System Viewer Introduction

- A software 'run-time analyzer'
- Helps diagnose race conditions and timing problems



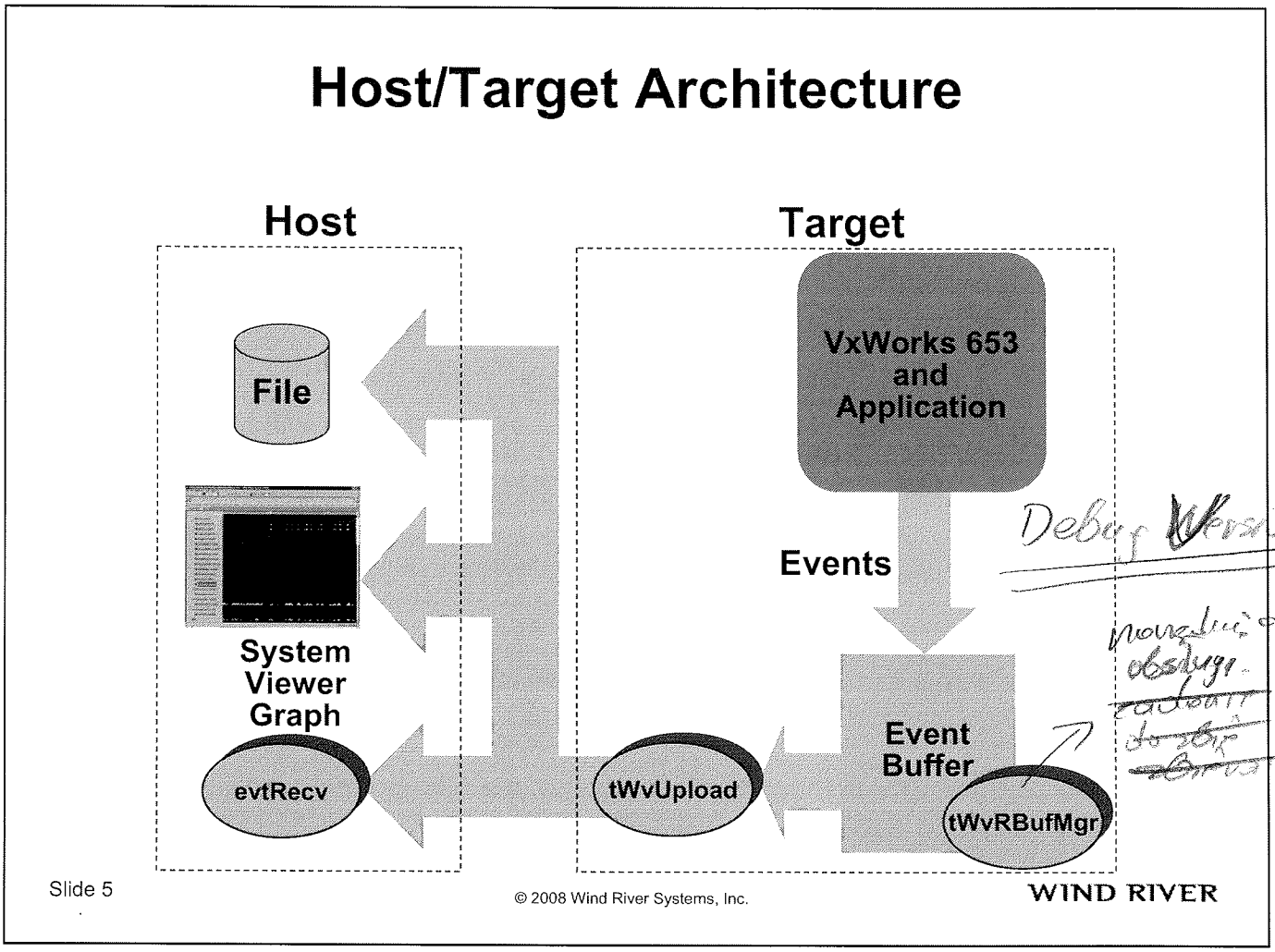
Slide 4

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- Data logging can be triggered to focus on events of interest



Student Notes

- When event logging is enabled, event data are placed in a buffer in target memory as system instrumentation points and user events are encountered
  - This buffer can be dynamically extended or contracted by the task "tWvRBuffMgr"
- The task "tWVUpload" uploads event data to the host
  - When and how this upload occurs can be configured
- Events can be uploaded to a listening System Viewer graph, to the "evtRecv" tool, or to a host file, by various different upload methods
- Event logging, event upload, and event buffer management can be configured either programmatically or via the Tornado GUI
- The concept of System Viewer events does not refer to VxWorks 653 events used for interprocess communication
- System calls are expensive

Visualization and Monitoring

Wind River System Viewer

- Introduction
- **Configuring System Viewer**
- Using System Viewer

Monitoring Tools

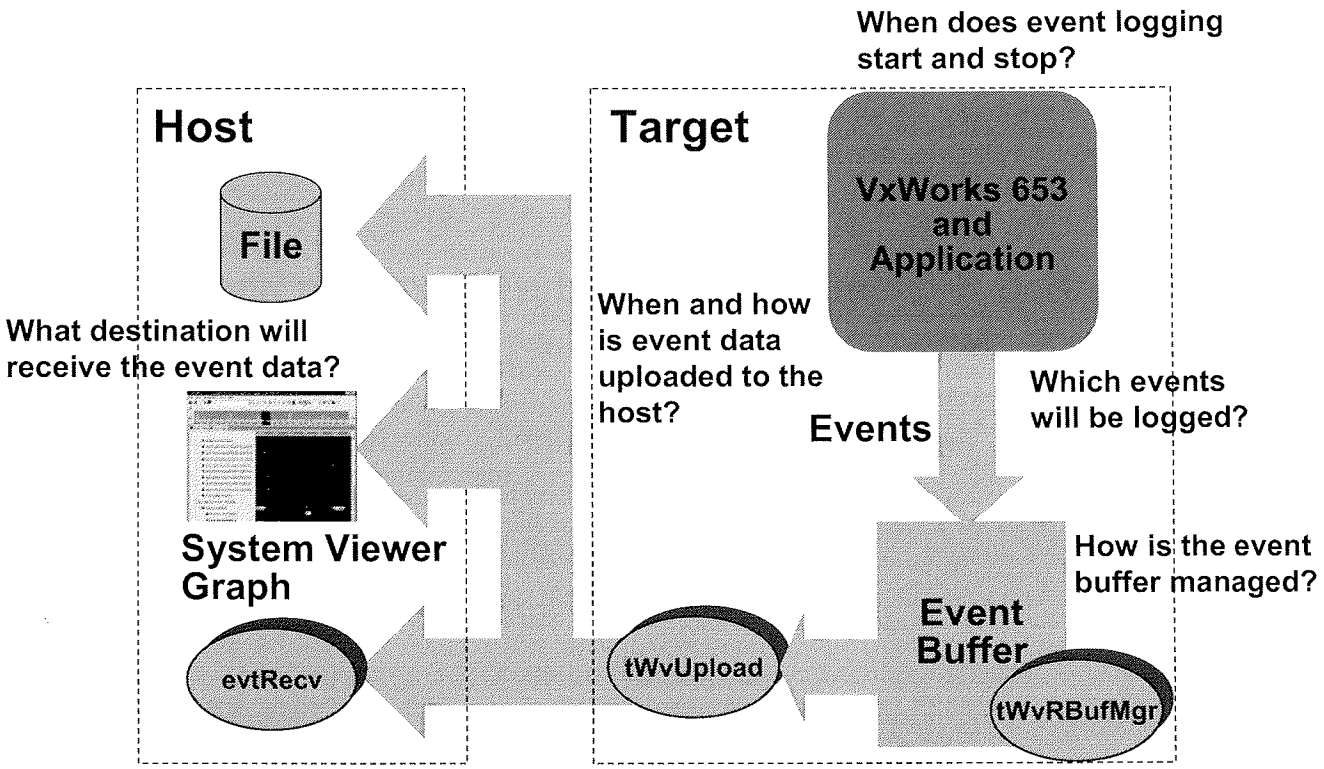
Slide 6

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Configuring System Viewer



Slide 7

© 2008 Wind River Systems, Inc.

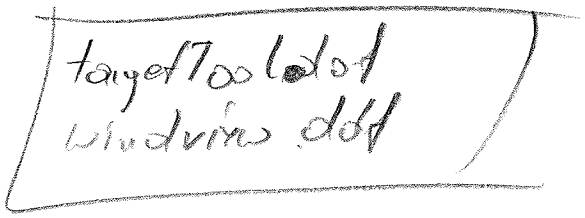
WIND RIVER

Student Notes

System Viewer Configuration

Configuring System Viewer involves:

- Configuring VxWorks 653 (Module OS and System Shared Library)
- Configuring the Target Server
- Configuring System Viewer and the Triggering host tools



Slide 8

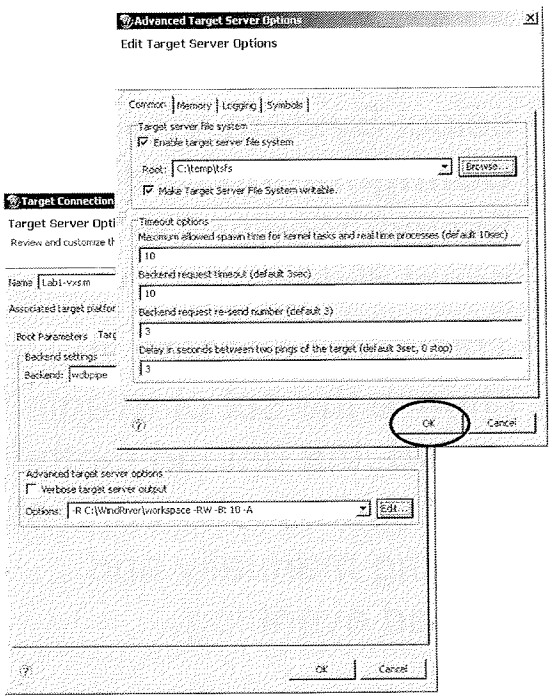
© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Target Server File System (TSFS)

- WDB agent TSFS component must be included
- Provides remote access to host filesystem
- Can open TCP socket connections on the host on behalf of the target
- Specify the host file system root directory and read/write access
  - From the target connection's properties, click "edit" on Advanced Target Server Options



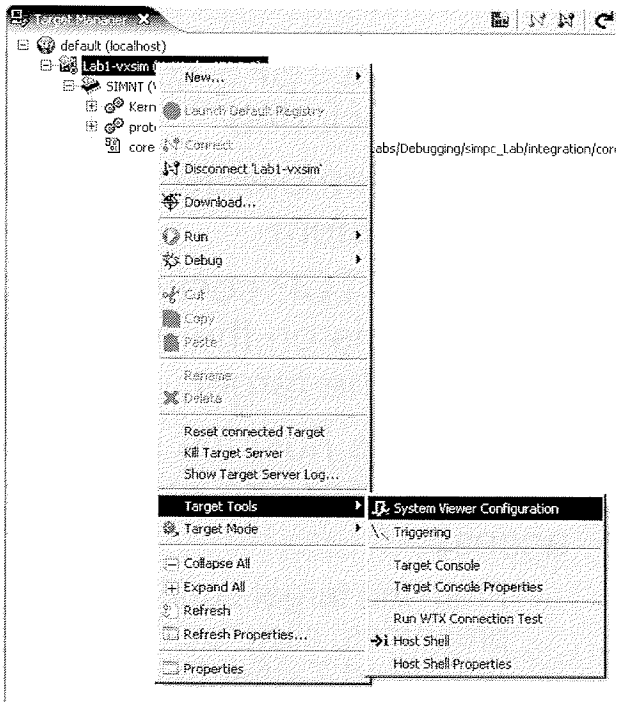
© 2008 Wind River Systems, Inc.

Student Notes

- The TSFS facilities can be used even on targets without networking support (such as VxSim) so long as there is a communication channel between the target server and the WDB agent
- When the TSFS component of the WDB agent is included in VxWorks 653, an I/O system device named "/tgtsvr" is created on the target
  - This device allows access to TSFS facilities
    - To open a file **foo/bar** on the host in the TSFS root directory, one opens **/tgtsvr/foo/bar** on the target
      - Operations on the returned file descriptor are actually handled by the target server, via the WDB agent
    - To create a TCP socket connection to port number 6164 on host "fido", one opens the "file" **/tgtsvr/TCP:fido:6164** on the target, obtaining a file descriptor which is for most purposes equivalent to a socket
      - The actual socket, however, is created by the target server on the host
- See the reference pages for ***wdbTsfsDrv*** for more information

System Viewer Configuration and Control

1. From the target Menu, select **System Viewer Configuration** to view the System Viewer Control panel dialog.
2. This panel enables you to choose the set of events to record, the upload method and mode, to start and stop logging manually and upload event log.



Slide 10

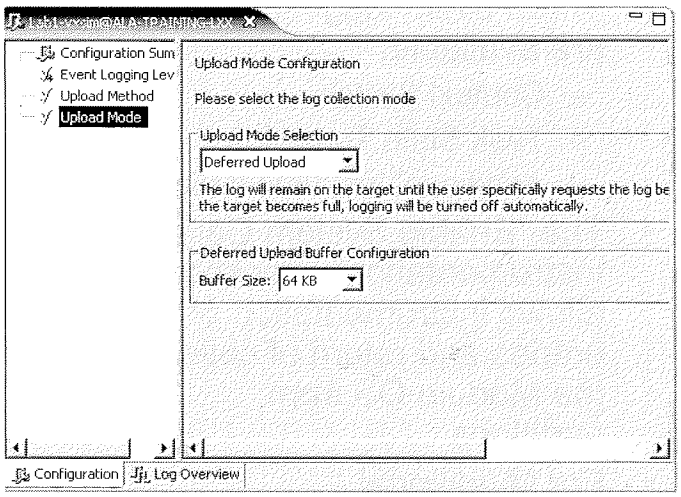
© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

System Viewer Configuration and Control (Continued)

3. Configuration tab opens the Config & Control dialog.



Slide 11

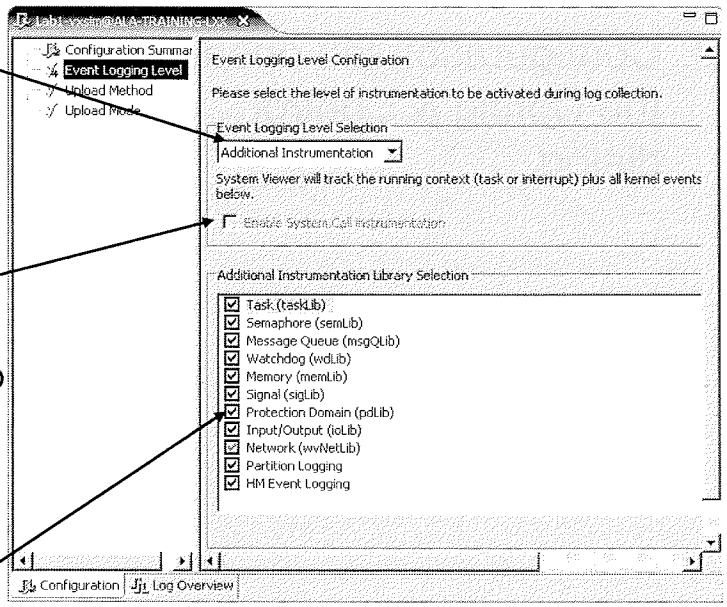
© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Configuring the Event Logging Level

- Three event logging levels are available:
  - Context Switch
  - Task State Transition
  - Additional Instrumentation
- The "Enable System Call Instrumentation" checkbox allows instrumentation of entries and return of calls to the Module OS from partitions
- For "Additional Instrumentation" level, select the libraries logged
  - For example, "Protection Domain" to partition processes and events



Slide 12

© 2008 Wind River Systems, Inc.

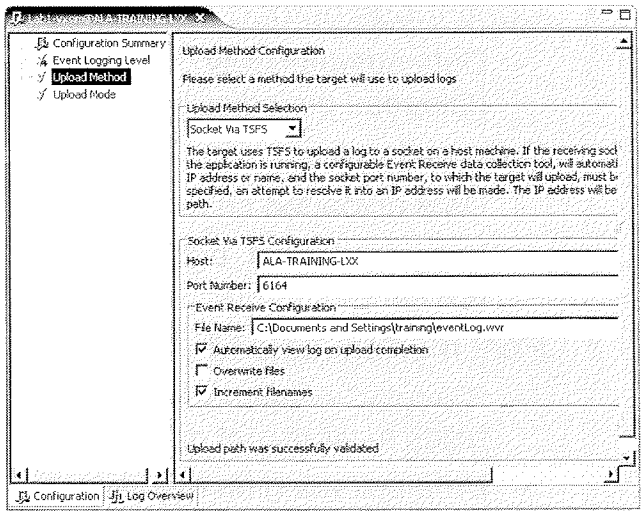
WIND RIVER

Student Notes

- When logging at the Additional Instrumentation level, selecting a library for logging means that all objects of that library will log events
  - For instance, selecting "semLib" means that all object events involving any semaphore in VxWorks 653 or the application will be logged
- To instrument only some objects of a given type requires that **wvLib** routines be called in the application (or from the shell)
- See the reference entries for ***wvObjInst( )*** and ***wvObjInstModeSet( )*** for more information on logging events corresponding to particular objects or sets of objects

Configuring the Upload Method

- Upload via a socket lets you upload directly to a System Viewer graph or the "Event Receive" tool
- Upload to a file lets you store the log in a file on a host
- Four types of links with the host can be used
  - TSFS (WDB link)
  - TCP/IP socket
  - NFS
  - netDrv (FTP)



© 2008 Wind River Systems, Inc.

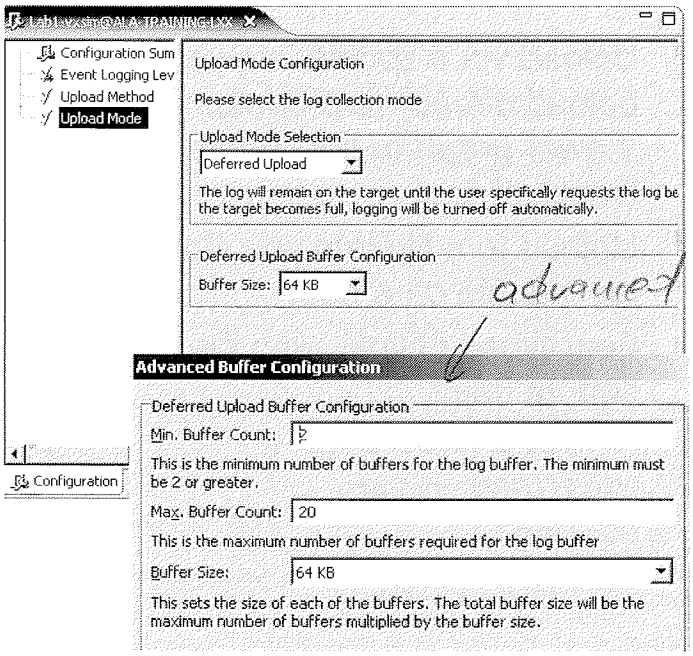
Student Notes

- For upload via TSFS socket, one must specify the target server device path for the destination host and port number
  - For example "/tgtsvr/TCP:myhost:6164"
- For upload via TCP/IP socket, one must specify the destination host and port number
  - For example, "myhost:6164".
- For upload to a file via TSFS, one must specify the TSFS path to the file on the host
  - For example, "/tgtsvr/evtLog.wvr";
  - One must also configure the target server specifying the root of the TSFS file system, and enabling read/write TSFS access.
- Using "File via NFS," one specifies any file writable by the target via the NFS client on a NFS server
- Using "File via netDrv," one specifies any file writable by the target via the netDrv driver on a host with a TCP server

© 2008 Wind River Systems, Inc.

Configuring the Upload Mode

- Continuous upload
  - Data automatically uploaded periodically
- Deferred upload
  - Data uploaded on request
  - Logging stops if buffers are full
- Post mortem upload
  - "Circular" event buffer holds most recent data
  - Data uploaded on request after a warm reboot
- Advanced button allows more control over buffers



© 2008 Wind River Systems, Inc.
WIND RIVER

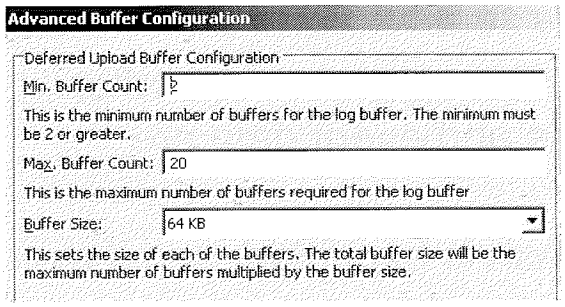
Student Notes

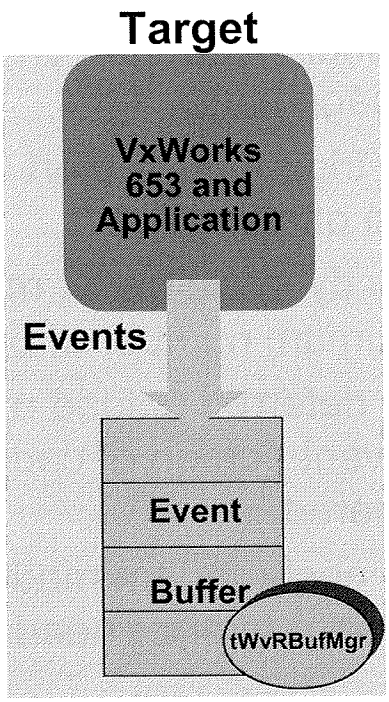
- In deferred mode, collection continues until either it is explicitly stopped or the event buffer fills
  - The data can be uploaded after collection stops
  - This mode has the advantage that upload activity does not appear in the uploaded log
  - This is the default mode
- In continuous mode, upload occurs periodically during collection as the event buffer fills
  - The event buffer can be dynamically expanded or contracted during upload
  - This mode has the advantage that very large logs can be collected, but upload activity will occur in the log
- In post-mortem mode, a circular buffer is created in the area defined in the buffer management window
  - During code execution events will be logged continuously with older data being overwritten by new events
  - Event data is uploaded after a system crash followed by a warm reboot

© 2008 Wind River Systems, Inc.

Buffer Size and Management

- Variable size event buffer requires an additional task to manage the buffer
- Fixed size buffers do not require buffer management





Slide 15

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- Buffer thrashing, a condition in which the ring buffer is repeatedly extended and contracted as the amount of data crosses a threshold, can sometimes occur during continuous upload mode
  - Often it can be avoided by increasing the minimum buffer count beyond the thrashing point
- ***wvUploadTaskConfig* (stackSize, priority)** can be used to modify the priority of the upload task for the next iteration

Intrusiveness Issues

- Uploading intrudes on running of application, as does buffer management
- Use triggering to start System Viewer, focusing on area of code of interest
- Stop dynamic buffer management by defining minimum and maximum number of buffers as same
- Use deferred upload mode so that upload occurs after relevant portion of code is done and event data are gathered
- Use post-mortem mode if a crash is expected, and if events just prior to error need to be observed

Slide 16

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Visualization and Monitoring

Wind River System Viewer

- Introduction
- Configuring System Viewer
- **Using System Viewer**

Monitoring Tools

Slide 17

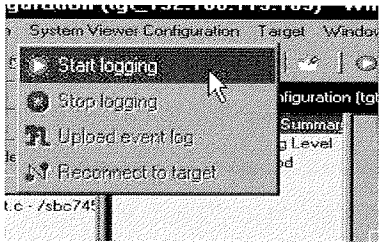
© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Controlling Data Collection and Upload

- Start/Stop Data Collection
  - "Start logging" tab from "System Viewer" menu



  - Start and Stop buttons on System Viewer Config & Control panel
  - Programmatic control from target
  - Triggering facility
- Data Upload
  - Button on System Viewer Config & Control panel
  - Programmatic control from target

Slide 18

© 2008 Wind River Systems, Inc.

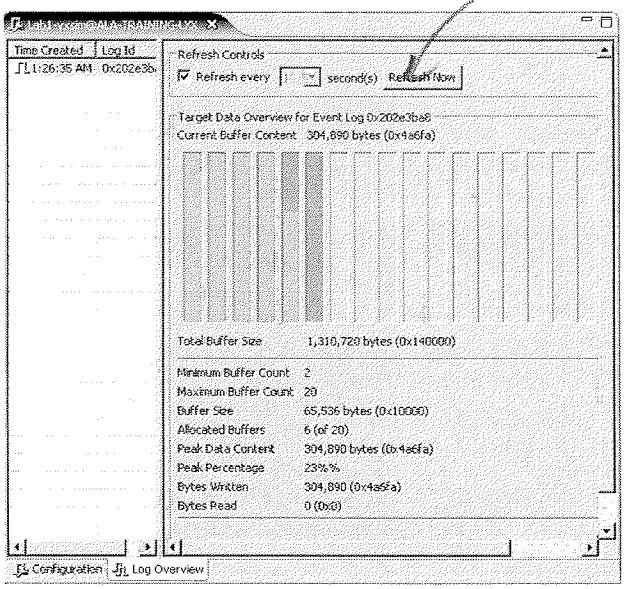
WIND RIVER

Student Notes

- The usrSystem Viewer module defines the routines wvOn( ) and wvOff( ) which start and stop a typical System Viewer logging and upload session
  - The upload mechanism used is the last one of
    - TCP/IP socket upload path
    - TSFS socket upload path
    - File upload path which has been included in the VxWorks 653 image
  - usrSystem Viewer documentation is in the System Viewer User's Guide

Log Overview During Logging

- **System Viewer Configuration** panel monitors buffer filling during continuous collection

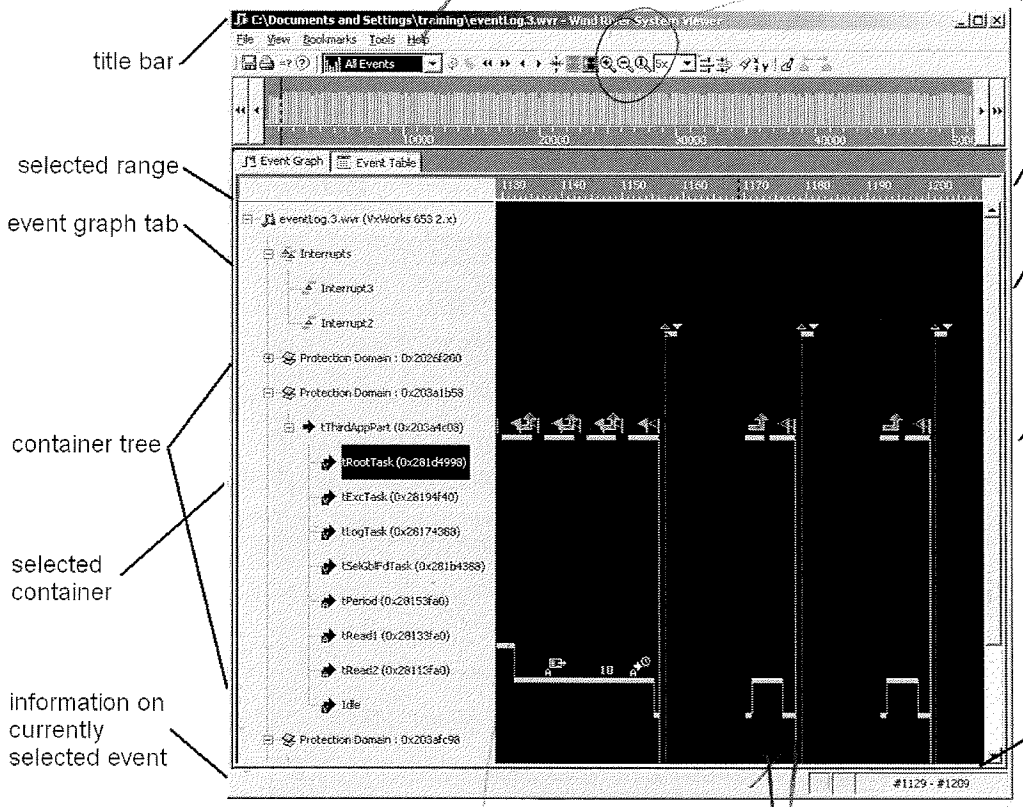


© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

The Event Graph View



Slide 20

© 2008 Wind River Systems, Inc.

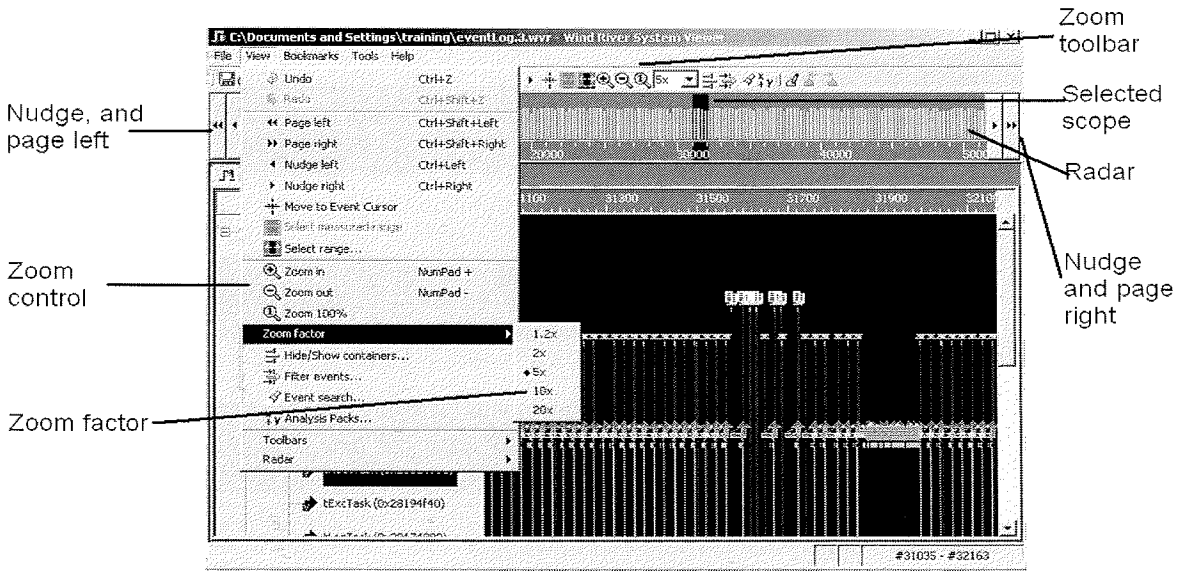
WIND RIVER

Student Notes

- Can distinguish ARINC processes

Navigation and Zooming

- To select an area, simultaneously press the "Shift" key and the left mouse button, and then drag the mouse over the area of the Radar you want to inspect



Slide 21

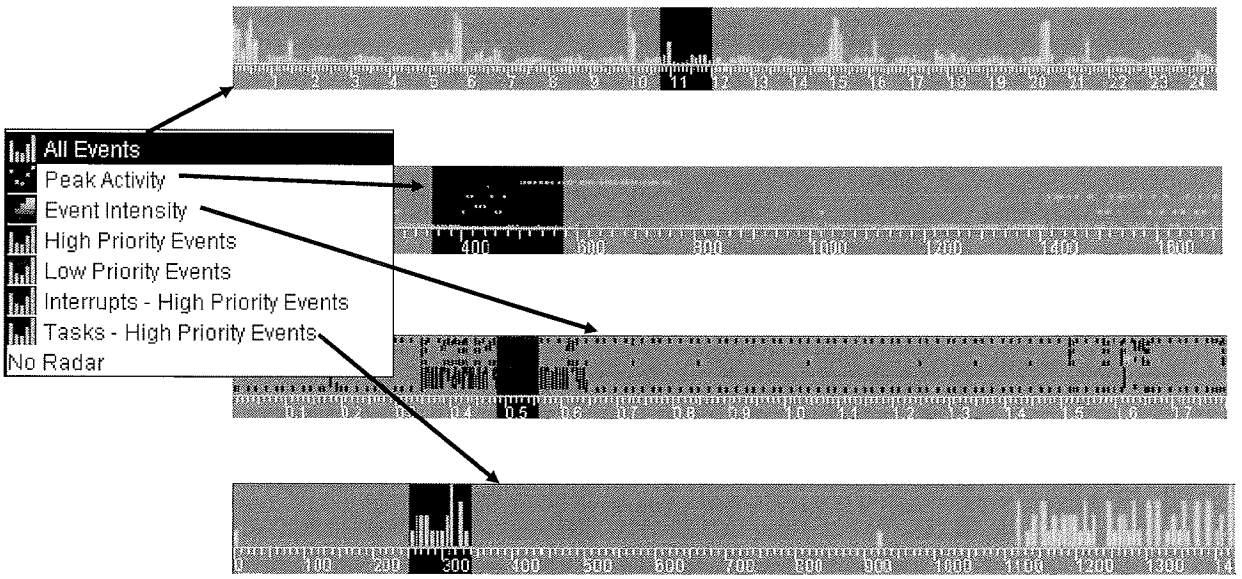
© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Using Radar

- The radar helps in rapidly focusing on an area of interest.



Slide 22

© 2008 Wind River Systems, Inc.

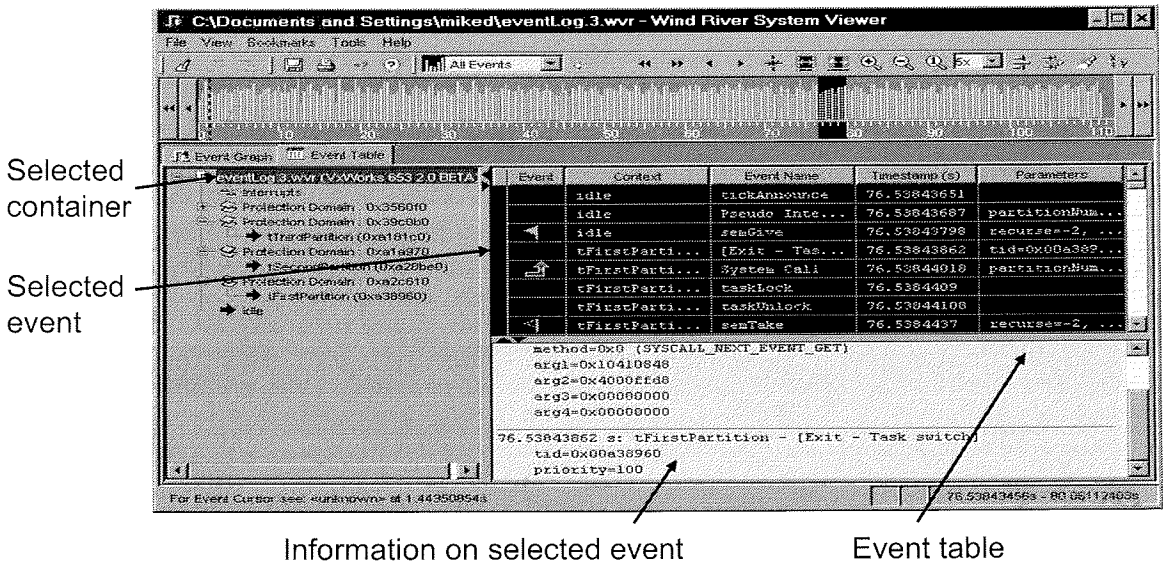
WIND RIVER

Student Notes

- The radar always represent the whole log range, with X-axis representing time
  - "All Events": All events are given the same weighting
    - The Y-axis represents the level of activity
  - "Peak Activity": A higher weighting is given to the container with highest activity
    - The Y-axis represents the vertical location of a container in the container tree (left side of event graph)
  - "Event Intensity": The color intensity is proportional to the activity in the container
    - The Y-axis represents the vertical location of a container in the tree
  - "High Priority Events": Events occurring at high priority in tasks or interrupts are given the highest weighting
    - The Y-axis represents the priority at which the activity occurs
  - "Low Priority Events": Same as "High Priority Events" but for low priority events
  - "Interrupts - High Priority Events": Same as "High Priority Events" but only for events occurring in interrupts
  - "Tasks - High Priority Events": Same as "High Priority Events" but only for events occurring in Tasks

Event Table View

- A sorted list of events, which occur in the domains and tasks selected in the left hand container tree



Slide 23

© 2008 Wind River Systems, Inc.

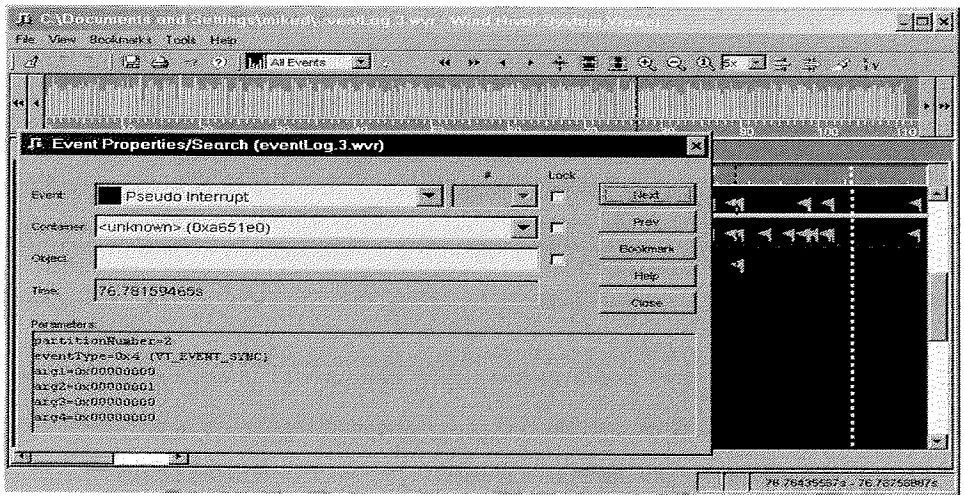
WIND RIVER

Student Notes

- The context menu on the "Event Table" view allows adding, hiding columns, changing the column order, saving displayed events in a text file, and setting the cursor or a bookmark on a given event

Searching for Events

1. Place your cursor over the event and a description will appear in the status bar.
2. Double-click, and the "Event Properties" dialog will appear.



Slide 24

© 2008 Wind River Systems, Inc.

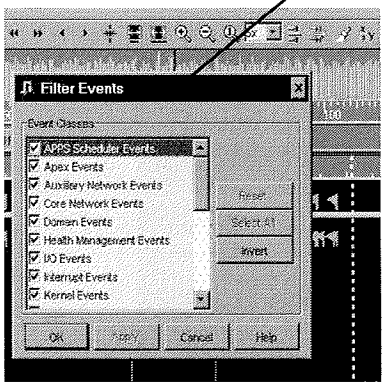
WIND RIVER

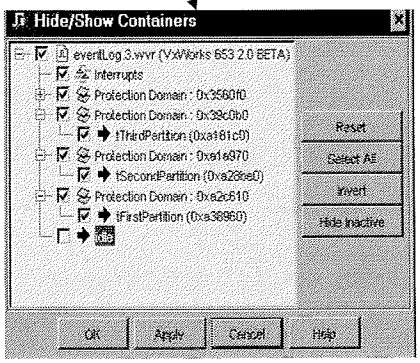
Student Notes

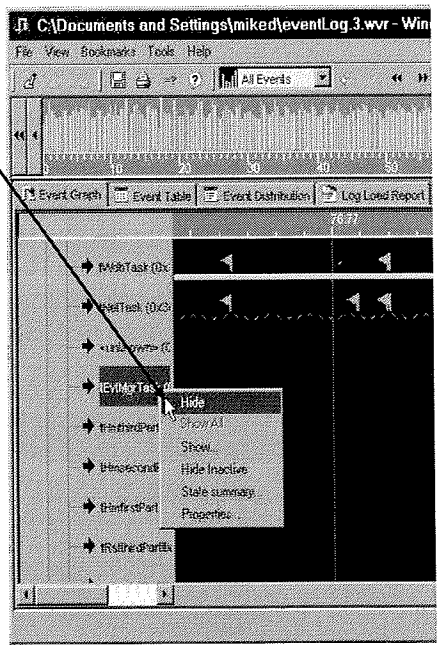
- Double-clicking an event, or right-clicking and selecting "Show Event..." from the context menu brings up the "Show Event" dialog
- The "Show Event" dialog allows one to search forwards or backwards in time for events matching particular criteria
  - The type of event
  - The task or interrupt context in which the event occurs
  - The VxWorks 653 object (semaphore, message queue, watchdog, etc.) involved, if any
- Select or type in the appropriate values, and lock the fields of interest, then click the "Prev" or "Next" buttons
  - For example, type `user19` in the event type field and lock only that field to search for occurrences of user event `19`
  - Double-click on a ***semTake( )*** event, lock only the object ID field, and search forwards or backwards in time for a corresponding ***semGive( )***

Too Many Events?

- Use the left context menu to hide domains, tasks, processes
- Use the System Viewer toolbar to
  - Hide/Show containers
  - Filter events







Slide 25

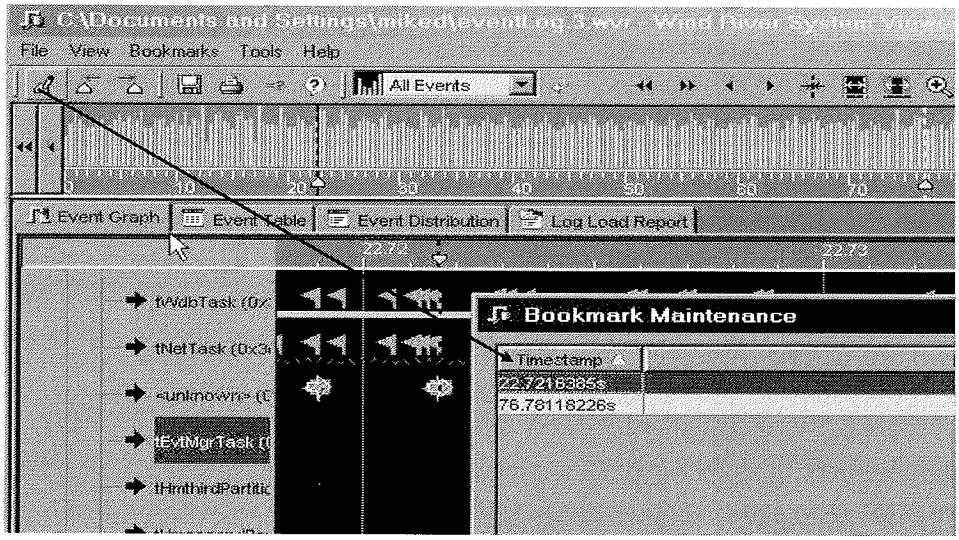
© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Using Bookmarks

- Bookmarks identify interesting locations in the event log.
  - You can defined them at any timestamp.



Slide 26

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Triggering

- Triggering associates actions with System Viewer events
- Actions can
  - Start or stop System Viewer logging
  - Call a function
  - Optionally, disable a trigger and/or enable another
- A trigger fires when enabled and an event matches criteria
  - Any combination of event type, context, and involved system object ID
  - Optional conditional expression involving integral global variable or ISR-safe function of no argument
  - User trigger events, distinct from user System Viewer events, are generated using ***trgEvent( )***
- Cannot trigger in POS, because of system call mechanism

Slide 27

© 2008 Wind River Systems, Inc.

WIND RIVER

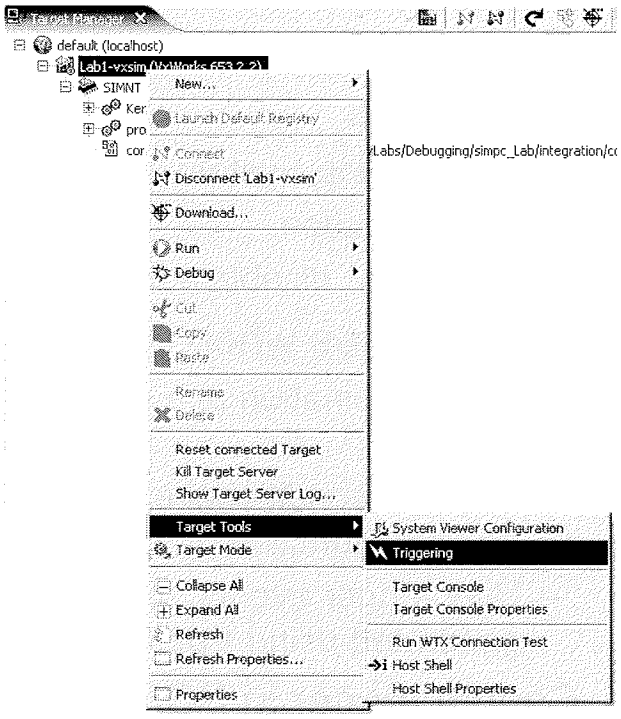
Student Notes

- Using Triggering to start and stop System Viewer is a good way to collect short event logs focused on the events of interest
- Starting or stopping System Viewer, or calling a function which is not explicitly identified as safe to call from an ISR, occurs in the context of a task "trgActDef" spawned specifically for this purpose
- In the case of a trigger starting System Viewer, this means that logging is not started immediately upon the event which fires the trigger, but only when the "trgActDef" task gets a chance to run
- For this purpose, then, it is best if the trigger is initiated on an event which occurs somewhat before the events of interest
  - Stopping System Viewer after an event of interest is an alternative
- The triggering facility is documented in the ***System Viewer User's Guide***
  - Triggering can be controlled from the target application using the routines in **trgLib**
- See the reference entry for this library for more information

Launching the Triggering User Interface

1. Click the ***Triggering*** button on target connection's properties menu to open **Triggering** control panel.

2. Trigger managing panes will open in the editor/ source debug pane. A new "Triggering" menu tab will also appear.



Slide 28

© 2008 Wind River Systems, Inc.

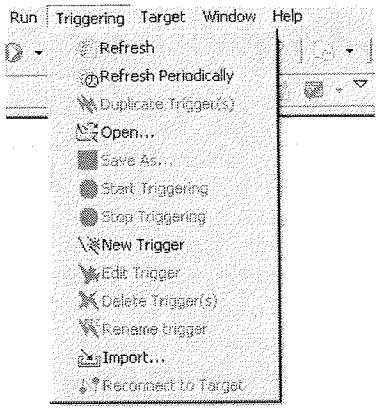
WIND RIVER

Student Notes

- For more information on the Triggering user interface, see the ***System Viewer User's Guide***

Triggering User Interface

3. Edit trigger firing conditions and actions, and add new trigger points by selecting actions under the "triggering" menu.
   1. A toolbar is also available for these functions



Slide 29

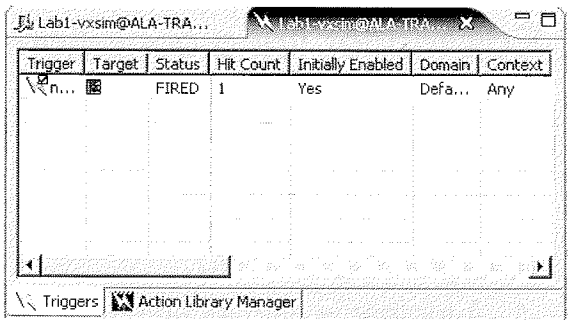
© 2008 Wind River Systems, Inc.

WIND RIVER

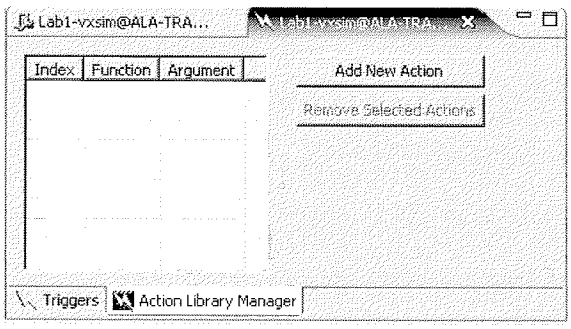
Student Notes

Starting and Monitoring Triggers

1. Click to start Triggering.



2. Use the action library manager to add new trigger actions.



Slide 30

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Adding New Events

- Programmatically, you can also
  - Instrument particular objects
  - Log user events with ***wvEvent( )*** and ***e( )***
- For information about programmatic control, see
  - **wvLib**
  - ***Wind River System Viewer User's Guide***
  - ***Wind River System Viewer API Reference***

Slide 31

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

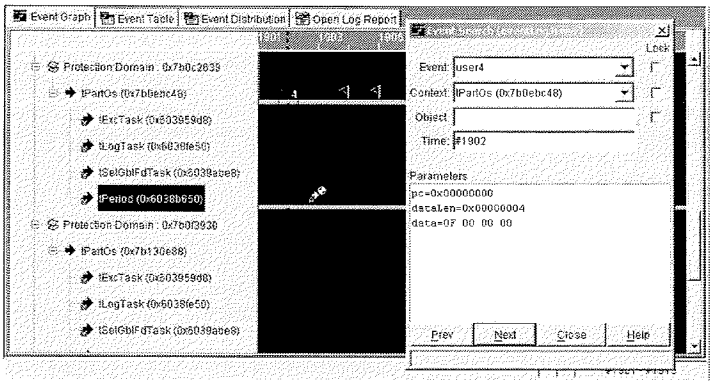
- Logging larger classes of events yields more information, but imposes greater overhead
  - The amount of overhead which System Viewer imposes on the target system is also affected by
    - The amount of instrumented (kernel) work in the application compared to non-instrumented work
    - The upload strategy
    - The event buffer configuration
- Most of the programmatic routines used for controlling instrumentation and logging are implemented in **wvLib**
  - See the ***VxWorks 653 Core OS Reference : Libraries*** entry for this library for more information

User-Defined Events

- STATUS wvEvent (eventNo, buffer, bufsize)

eventNo	ID number for the event, which will appear on the viewgraph
buffer	Address of user supplied data. ***NULL*** if none
bufSize	Size of user supplied data, in bytes

- Buffer can contain user data that is passed up to System Viewer on upload
  - This is displayed in a hex-dump format



Slide 32

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- A useful technique is to create macros for instrumentation functions such as ***wvEvent()***
  - After development completes the macros can be turned into no-ops so as to avoid instrumentation in the production code

User Eventpoints

STATUS e (addr, eventNo, taskNameOrId, evtRtn, arg)

addr	Address to set eventpoint
eventNo	An event number for display
taskNameOrId	The task ID of the task in which to raise the eventpoint, 0 = all tasks
evtRtn	Function to call when event point is hit. Event is logged if **evtRtn**'s return value is zero
arg	An integer argument to pass to **evtRtn** above

Slide 33

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

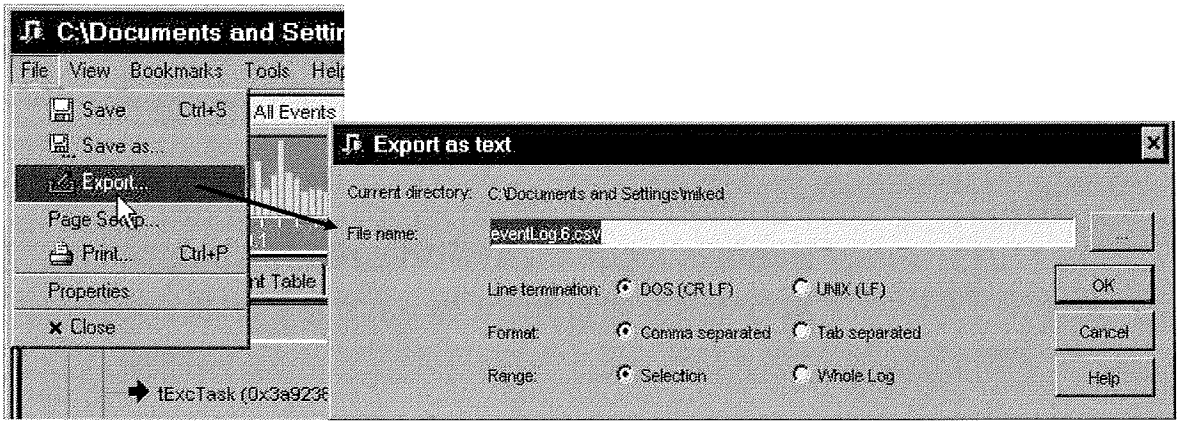
- An eventpoint is like a breakpoint, except that instead of stopping the task which hits it, a System Viewer event can be logged and a function can be called
- The ***e( )*** function is a WindSh built-in, which also exists as a target function in **dbgLib**
- For example

  → `e sin, 10, 0, 0, 0`

  sets an eventpoint with an ID of `10` at the address of ***sin( )*** for all tasks
  - Whenever any task calls the ***sin( )*** function, System Viewer user event `10` would be logged
- If the **evtRtn** argument is used, it should return an integer and take a four-byte argument (passed as an integer)
  - If the returned value is not zero, the event will not be logged

Exporting Events to Text Files

- The "Save as text" option exports an event selection or an entire log to a text file.
  - This can be further analyzed using Excel, or other tools.



© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Review: System Viewer

- Diagnose race conditions and timing problems
- Configure through VxWorks 653 configuration and System Viewer Config & Control panel
- Use Radar, Event Graph, and Event Table views to navigate through logged events
- Use triggering to focus on areas of interest
- Add custom User Events for additional application details

Slide 35

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Visualization and Monitoring

Wind River System Viewer

Monitoring Tools

- Memory Usage
- Performance
- Port

Slide 36

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Monitoring Tools

- Target side monitoring is permanent, minimally intrusive and is included in the software
  - Each monitoring tool has a host-side component, which formats and reports data on command
- Performance
  - Monitors and reports CPU usage either system-wide or within a partition (per thread)
    - Idle time
    - Time spent processing interrupts
    - Time spent for application processing

Slide 37

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Monitoring Tools (Continued)

- Memory Usage
  - Monitors system memory usage for heaps, stacks, ports, and health monitoring
  - Reports usage on a per partition basis
- Port
  - Logging and reporting of all port activity in a system
    - Includes both port control data and message contents
  - Monitored data can be streamed to the host or collected in a log and retrieved later

Slide 38

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Visualization and Monitoring

Wind River System Viewer
Monitoring Tools
- **Memory Usage**
- Performance
- Port

Slide 39

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Monitoring Memory Usage

- Reports memory used by heaps, stacks, ports, and health monitoring
  - Usage reported on a per partition basis
- Monitor runs as part of Host Shell
  - No additional configuration necessary
  - No change to system performance
    - Any changes in performance are due exclusively to use of Host Shell

Slide 40

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- Memory usage monitoring is not supported in COIL-based partitions

Monitoring Application Heaps:

memoryUsageHeapShow "*pID*"

- Displays heap summary, including
  - base address
  - size
  - number of allocated bytes
  - maximum number of allocated bytes
  - margin
- **pID** is the name or ID of the Module OS or application for which the heap usage is desired
  - If **pID** is not specified (or **pID** = 0), the heap summary for Module OS and all applications is displayed

Slide 41

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Monitoring Task Stacks

memoryUsageStackShow "*pID*" "*tID*"

- Displays stack summary, including
  - base address
  - size
  - current number of bytes used
  - maximum number of bytes used
  - margin
- **pID** specifies the application (or Module OS) to search in
- **tID** is name or ID of task or thread for which stack usage is desired
  - If **tID** is not specified (or **tID** = 0), stack usage for all tasks in specified application is displayed

Slide 42

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- Verification mechanism for certification

Examples

```
[coreOS] -> memoryUsageHeapShow ""
     App Name    Heap Base    Heap Size    Current(%)       Peak(%)        Margin
 --------------- ----------  ---------- -------------- -------------- ----------
 coreOS          0x00000000   17192832  10911200(63%)  11191888(65%)    6000944
 thirdPartition  0x40000000          0        0(100%)        0(100%)          0
 secondPartit >  0x40000000    1953792     70864( 3%)     70864( 3%)    1882928
 firstPartition  0x40000000    1953792     60720( 3%)     60720( 3%)    1893072
```

```
[coreOS] -> memoryUsageStackShow "coreOS"
    App Name      Task Name     Stk Base     Size      Current(%)      Peak(%)       Margin
 --------------- ------------ ---------- -------- -------------- -------------- ---------
 coreOS          tHm           0x1503fc0   16320       656( 4%)       752( 4%)     15568
 coreOS          tMgrTask      0x1506fc0   12224       464( 3%)       544( 4%)     11680
 coreOS          tExcTask      0x1508fc0    8128       336( 4%)       784( 9%)      7344
...
 coreOS          tRstfirstP >  0x1529fc0    4032       864(21%)      1872(46%)      2160
```


© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Monitoring the Health Monitor

memoryUsageHmShow "*pID*"

- Displays health monitor memory usage, including
  - Size of the HM log
  - Current number of logged items
  - Maximum number of logged items
  - Margin
- **pID** specifies application (or Module OS) for which health monitor memory usage is desired
  - If **pID** is not specified (or **pID** = 0), health monitor memory usage summaries for Module OS and all application partitions are displayed

Slide 44

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Monitoring Port Memory

memoryUsagePortShow

- Displays memory used by ports, including
  - Name of the port memory pool
  - Total size
  - Current memory use
  - Margin

Slide 45

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Monitoring all Memory Usage

memoryUsageAllShow "*pID*"

- Displays memory used by heap, stack, health monitor, and ports
  - memoryUsageHeapShow
  - memoryUsageStackShow
  - memoryUsageHmShow
  - memoryUsagePortShow
- **pID** specifies application (or Module OS) for which memory usage is desired
  - If **pID** is not specified (or **pID** = 0), summaries for Module OS and all application partitions are displayed

Slide 46

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Visualization and Monitoring

Wind River System Viewer
Monitoring Tools
- Memory Usage
- **Performance**
- Port

Slide 47

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Performance Monitor Overview

- Monitors CPU use either in Module OS or partition OS and can run in a certified environment
- Data is acquired by default
  - Component cannot be removed
  - Monitoring starts when Module OS starts, or when a partition is initially activated
  - Uses accumulators that run within the system
- Data is viewed by including **INCLUDE_TIME_MONITOR_SHOW** binary component
  - Component is included automatically
  - Data is read and fed to the host for display
- Tool is designed to be non-intrusive for CPU consumption
  - Runs permanently and takes a fixed, minimal overhead during partition window in updating a few fields in Partition OS structures
    - Since reporting task uses a communication media (such as a network stack and device), it does utilize CPU time

Slide 48

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Monitoring the Module OS

- Three elements are monitored in Module OS
  - Task Time
    - Cumulative time spent executing tasks
    - Includes `tPartitionName` that runs partitions
    - One time accumulator per task
  - Idle Time
    - Cumulative time that system is idle
    - Monitoring stops when it exits the idle loop or when an interrupt occurs
  - Interrupt Time
    - Time that system spends in an interrupt
    - As with tasks, there is one time accumulator for all interrupts

Slide 49

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- All elements begin when the time monitoring starts is recorded
- A task can become inactive when another task becomes active or when an interrupt occurs
- accumulator – a counter of elapsed time

Monitoring the Partition OS

- Four elements are monitored in partition OS
  - Thread Time
    - Cumulative time spent executing threads from time that monitoring starts is recorded
    - One time accumulator per thread
    - Monitoring stops when a pseudo-interrupt occurs or when partition is scheduled out
      - If a thread stops because of a pseudo-interrupt, it will start again when pseudo-interrupt finishes executing
      - Monitoring is not stopped when partition OS task is preempted by a Module OS task
  - Idle Time
    - Cumulative time that system has been idle since monitoring started
    - Monitoring stops when exiting idle loop, when a pseudo-interrupt occurs, or when partition is scheduled out

Slide 50

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Monitoring the Partition OS (Continued)

- Pseudo-Interrupt Time
  - Time that system spends in a pseudo-interrupt once monitoring is started
  - As with tasks, there is one accumulator per pseudo-interrupt
  - Monitoring stops when pseudo-interrupt execution exits or when partition is scheduled out
- Partition Window
  - Amount of time that a partition is active since monitoring started

Slide 51

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Time Monitor Configuration

- To enable time monitor at startup, set parameter **TIME_MONITOR_ENABLE** to TRUE
  - Equivalent to calling **timeMonitorStart()**
  - **TIME_MONITOR_ENABLE** is specified in components **TIME_MONITOR2** (Module OS) or Partition OS
- Time accumulator consists of a cumulative time counter and time when measurement was started
- Free list of accumulators is created at start up time containing all accumulators
  - Number of accumulators is defined by parameter **MAX_NUMBER_OF_ACCU** in **TIME_MONITOR** (Module OS) or Partition OS component

Slide 52

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

TIME_MONITOR_STOP
TIME_MONITOR_CLEAR
TIME_MONITOR_START

Time Monitor Configuration (Continued)

- For each task created, one time accumulator is created
- When start function is called, each time accumulator pointer points to its respective accumulators
- Each time a task is created, an accumulator is removed from the free list, and added to the TCB of the task
- When a task is deleted, task's accumulator is added to free list

Slide 53

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Time Monitor API

- **timeMonitorLibInit( )**
  - Initializes the time monitoring library
  - Creates global pool of accumulators
- **timeMonitorClear( )**
  - Clears all monitor values
- **timeMonitorStart( )**
  - Starts the time monitoring
- **timeMonitorStop( )**
  - Stops the time monitoring
  - Updates all accumulators

Slide 54

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Time Monitor API (Continued)

- **timeMonitorTaskGet( )**
  - Retrieves the CPU time used by this task
- **timeMonitorSysGet( )**
  - Retrieves the system time values
- **timeMonitorShow( )**
  - Displays all of the acquired data
- **timeMonitorAPEXShow( )**
  - Displays all of the acquired data for any APEX processes
  - Filters **timeMonitorShow** to only display APEX processes

Slide 55

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Example: timeMonitorShow

```
[coreOS] -> timeMonitorShow 0
-----------+----------------------------------------+--------------------+---------+
Id         |                name                    | CPU H:M:S,nS       | used %  |
-----------+----------------------------------------+--------------------+---------+
---------- | CPU time monitoring period             | 00:00:35,395400000 | 100 %
---------- | CPU time monitoring reference          | 00:00:35,395400000 | 100 %
---------- | CPU time used by IDLE                  | 00:00:30,338511000 | 85 %
---------- | CPU time used by ISR                   | 00:00:04,987712000 | 14 %
0x0024d970 | tHm                                    | 00:00:00,000000000 | 0 %
0x0024dc88 | tMgrTask                               | 00:00:00,000000000 | 0 %
0x0024f238 | tExcTask                               | 00:00:00,000000000 | 0 %
0x00257848 | drv_01                                 | 00:00:00,000013000 | < 1 %
0x002c4f20 | tLogTask                               | 00:00:00,000000000 | 0 %
0x002f82b8 | tNetTask                               | 00:00:00,017307000 | < 1 %
0x00334248 | tWdbTask                               | 00:00:00,000213000 | < 1 %
0x00c8e608 | tShell                                 | 00:00:00,002074000 | < 1 %
0x00c90d20 | tEvtMgrTask                            | 00:00:00,000000000 | 0 %
0x00c93380 | t10                                    | 00:00:00,000000000 | 0 %
0x00c956b0 | t40                                    | 00:00:00,000000000 | 0 %
0x00c979e0 | t25                                    | 00:00:00,000000000 | 0 %
0x00c99d10 | tsync                                  | 00:00:00,000000000 | 0 %
0x003aa7c0 | tHmwrSbc750gx_pa                       | 00:00:00,000032000 | < 1 %
0x003aaaf8 | tWrSbc750gx_part1                      | 00:00:00,009004000 | < 1 %
0x003ac018 | tRstwrSbc750gx_part1                   | 00:00:00,000225000 | < 1 %
0x003c04b8 | tHmwrSbc750gx_pa                       | 00:00:00,000023000 | < 1 %
0x003c06e8 | tWrSbc750gx_part2                      | 00:00:00,006435000 | < 1 %
0x003aeac0 | tRstwrSbc750gx_part2                   | 00:00:00,000372000 | < 1 %
0x003b0a10 | tHmwrSbc750gx_pa                       | 00:00:00,000027000 | < 1 %
0x003b0d48 | tWrSbc750gx_part3                      | 00:00:00,015316000 | < 1 %
0x003c2620 | tRstwrSbc750gx_part3                   | 00:00:00,000328000 | < 1 %
0x003e8900 | tHmwrSbc750gx_pa                       | 00:00:00,000031000 | < 1 %
0x003e8c38 | tWrSbc750gx_part4                      | 00:00:00,017480000 | < 1 %
0x003d61b8 | tRstwrSbc750gx_part4                   | 00:00:00,000289000 | < 1 %
```

Slide 56

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Visualization and Monitoring

Wind River System Viewer

Monitoring Tools

- Memory Usage
- Performance
- **Port**

Slide 57

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Port Monitoring Overview

- Port monitoring tool is composed of three components
  - Low-level instrumentation
    - Embedded in sampling and queuing port libraries
    - Collects data during port operation and sends it to monitoring component
    - Cannot be removed
  - Monitoring
    - Receives collected data and stores it in a buffer (buffer is allocated to port specific memory region)
    - Can be added or removed from system as required
    - Interaction between instrumentation component occurs when monitoring is first enabled
  - Reporting
    - Provided as a configlette **usrPortMonitor.c**
    - Code reads collected data in buffer and asynchronously pushes it to host for display

Slide 58

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- The tool can run in a certified environment

+ Port Rep

ti tasks port proces

"Monitoring" Component

- Message data monitoring consists of identifying each message and logging its identifier
- If reporting is enabled, message data is shown if message is still in port buffer at time of reporting
- Monitored data is stored in a ring buffer
  - One buffer for whole system
  - Stopping port monitoring does not clear data collected in monitoring buffer
- Buffer size is specified at system startup during call to **portMonitorLibInit( )**
- A sender point, a distribution point, and a receiver point of each channel are monitored

Slide 59

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Channel Points

- A sender point, a distribution point, and a receiver point of each channel are monitored
  - Sender Point
    - All messages that are sent by a port and put into a source port are monitored
  - Distribution Point
    - All messages taken from the sender port and sent to the destination port are monitored
      - Sampling ports use a zero-copy algorithm, so there is no distribution point for sampling channels
      - When using the **RECEIVER_DISCARD** function, the event of discarding a message when the destination buffer is full is recorded at the distribution point
  - Receiver Point
    - All messages read from a destination port are monitored

Slide 60

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Channel Points (Continued)

- Information captured about each message at each point includes:
  - Port type
  - Timestamp
  - Message length
  - Source port identifier (sender, distribution points only)
  - Destination port identifier (distribution, receiver points only)
  - Message identifier

Slide 61

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

"Reporting" Component

- "Reporting" component collects data from monitoring buffer and writes it to virtual I/O (VIO) channel of WDB agent
  - VIO channel provides fast, asynchronous communication between target and host
- Using this channel lets reporting code rely exclusively on WDB back end without requiring configuration for a specific communication media
- Reporting task is spawned at configlette initialization by **usrPortMonitorInit( )**
- If monitoring is disabled, reporting task suspends
- Configlette code **usrPortMonitor.c** can be modified as required, or removed from the system entirely
  - For example, a platform integrator might want to modify the code to avoid using VIO channel and send a UDP broadcast packet through a socket instead
- If bandwidth of media used for streaming is not large enough, messages can be lost because monitoring buffer wraps
  - Any loss of messages is indicated in report

Slide 62

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Port Monitor Use

- Monitoring
  - Include binary component **INCLUDE_PORT_MONITOR** in Module OS
  - From host or target shell, or programmed into Module OS, enable or disable monitoring of a specified channel using
    - **portMonitorStart**
    - **portMonitorStop**
- Reporting
  - Include binary component **INCLUDE_PORT_REPORT** in Module OS
  - From host or target shell, enable or disable reporting using
    - **portMonitorEnable** (monitoring buffer cleared when reporting enabled)
    - **portMonitorDisable**
  - Display data with **wtxConsole** or use **portMonitorLib.tcl** application
  - **portMonitorShow**
    - Displays monitored data for a specific channel or port
    - Convenience routine when using target shell on a standalone system
    - Not meant to be certified

Slide 63

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- Reporting routines cannot be programmed into the Module OS because the reporting function is part of the tools, and tools can be included or removed from the system with no impact on the kernel domain

Example: Port Monitor

```
[coreOS] -> portMonitorEnable 0, 0
Time (sec/usec) Event    Type     Port Name        part MsgLen Dest. Port Name Dest. Part
00345:123789123 SENDER   Queuing "source port 5" 5    21
00354:665789403 DISTRI   Queuing "source port 5" 5    21     "destin port 5" 6
00384:123789794 RECEIVER Queuing "destin port 5" 6    21
00396:123789689 SENDER   Queuing "source port 5" 5    21
```

```
[coreOS] -> portMonitorEnable 1, 100
SENDER:
-----------------
Type                = QUEUING
channel                         = 2
port name                       = "source port 5"
port id                         = 0x20ccf128
parent channel                  = 2
partition                       = 1
module                          = 1
timestamp (sec/usec)            = 151 / 459151790
msgLen (bytes)                  = 21
displayed message length        = 21
00:                                           6d 65 73   * mes*
10: 73 61 67 65 20 70 61 79 6c 6f 61 64 20 31 30      * sage payload 10*
```

Slide 64

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Review

In this chapter you learned that:

- Target side monitoring is permanent, and minimally intrusive
  - Host-side components format and report data on command
- Performance - monitors and reports CPU usage either system-wide or within a partition (per thread)
- Memory Usage - monitors system memory usage for heaps, stacks, ports, and health monitoring
- Port - logs and reports all port activity in a system

Slide 65

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Questions

1. On continuous upload, data is uploaded on request after a warm reboot. True or False?
2. Deferred upload means that
   a. Data is uploaded on request
   b. "Circular" event buffer holds most recent data
   c. Data uploaded on request after a warm reboot
   d. Logging stops if the buffers get full
   e. All of the above
3. Triggering associates actions with _____
4. Target side monitoring is:
   a. Temporary
   b. Minimally intrusive
   c. Included in the software
   d. All of the above
5. The Partition Window is the amount of time that a partition is active since monitoring started. True or False?

Slide 66

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

1. False. On continuous upload the data is automatically uploaded, periodically.
2. A and D
3. System Viewer events.
4. B and C
5. True

ARINC 653 API

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Objectives

By the end of this chapter you will be able to:

- Understand the APEX interface and ARINC functionalities in VxWorks 653.
- Manage the partitions, processes and time
- Understand the Inter and Intra partition communication
- Use Health monitoring and Configuration tables

Slide 2

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

ARINC 653 API

Introduction

Partition Management

Process Management

Time Management

Inter Partition Communication

Intra Partition Communication

Health Monitoring

Slide 3

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

ARINC 653 Specification

- Primary objective of ARINC specification is to define a general purpose APplication/EXecutive (APEX) interface between operating system of an avionics computer resource and the application software
- Included within this specification are
  - Interface requirements between the application software and the OS
  - List of services that allow the application software to control scheduling, communication, and status information of its internal processing elements

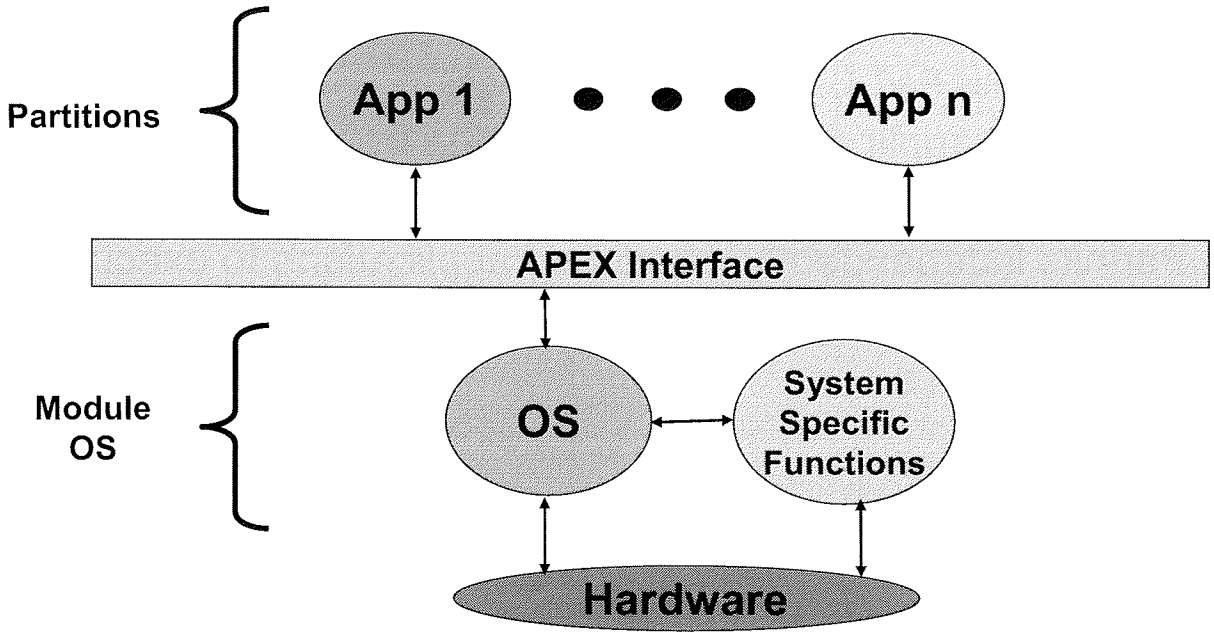
Slide 4

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

APEX Interface



Slide 5

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Overview

- Wind River VxWorks 653 Paltform 2.2 provides a full ARINC 653 APEX interface
  - It includes Part 1, Supplement 2 changes
- Allows the application to be abstracted from underlying OS implementation
- Permits independently developed applications to execute together on same hardware
- Set of 51 API's

Slide 6

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

ARINC 653 Advantages

- Portability
  - APEX interface facilitates portability of software applications
- Reusability
  - APEX interface allows production of reusable application code for systems
- Modularity
  - By removing hardware and software dependencies, APEX interface reduces impact on application software from modifications to overall system
- Integration of software of Multiple Criticalities
  - Each application uses a virtual CPU
  - Supports DO-178B Levels A - E on same processor, safely

Slide 7

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

ARINC Functionalities in VxWorks 653

- Partition Management
  - Based on a VxWorks Protection Domains/Shared Libraries
- Process Management
  - Based on a VxWorks task (also referred to as a thread)
- Time Management
  - Based on ***taskDelay*** and VxWorks's watchdogs
- Inter and Intra partition Communication
  - Extensions of message queues as well as new Wind objects
- Health Monitoring
  - Provides an infrastructure to handle error reporting and response
- Configuration Tables
  - Used at initialization time to create and allocate resources

Slide 8

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

ARINC 653 API

Introduction
Partition Management
Process Management
Time Management
Inter Partition Communication
Intra Partition Communication
Health Monitoring

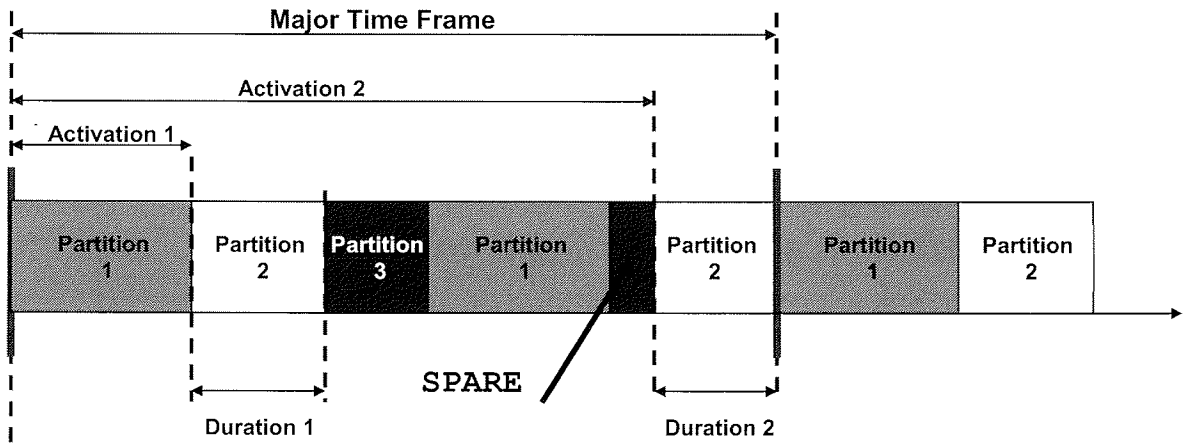
Slide 9

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Partition Management



- A major time frame is defined by schedule set in configRecord for Module OS
- Up to 16 major frames can be defined and switched via shell, or programmatically for different operating modes

Slide 10

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- Part 2 Implementation: Extended Services
  - Actions associated with schedule changes

Partition Management API

- ***GET_PARTITION_STATUS***
  - Obtains the status of the current partition
- ***SET_PARTITION_MODE***
  - Sets the operating mode of the current partition
    - Valid modes are ***NORMAL***, ***IDLE*** (shutdown), ***COLD_START*** and ***WARM_START***
  - Partitions must be set to ***NORMAL*** to begin operation after creation
- ***SET_SCHEDULE_MODE*** (extension)
  - Selects one schedule among 16 possible schedules
    - The time of the transition is selected by the application (next major frame, end of current window, next tick)
  - Native API: ***arincSchedSet(*int *index,* int *mode)***

Slide 11

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

ARINC 653 API

Introduction
Partition Management
Process Management
Time Management
Inter Partition Communication
Intra Partition Communication
Health Monitoring

Slide 12

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Process Management API

- These are the ARINC process APIs:

GET_PROCESS_ID	***GET_PROCESS_STATUS***
CREATE_PROCESS	***SET_PRIORITY***
SUSPEND	***SUSPEND_SELF***
STOP	***STOP_SELF***
START *	***DELAYED_START***
LOCK_PREEMPTION	***UNLOCK_PREEMPTION***
RESUME	***GET_MY_ID (extension)***

- ARINC 653 provides for either a hard or soft deadline for a process
  - A hard deadline forces remedial action such as ***WARM_START***
  - A soft deadline records failure and continues
  - Deadline actions are performed by Health Management

Slide 13

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- * START is synchronized with the partition for which the release point is set to `TRUE`.

Process Attributes

```
PROCESS_ATTRIBUTE_TYPE processTable[]=
{
/* NAME ENTRY_POINT STACK_SIZE BASE_PRIORITY PERIOD TIME_CAPACITY DEADLINE */

/* name,            entry,              stack, prio, period,          t capacity,deadline */
{"apexMinimalTest", apexMinimalTestRun, 4000, 100, TIME_4sec,        TIME_2sec,      SOFT},
{"apexListener",    apexListener,       4000, 110, ZERO_TIME_VALUE,  TIME_6sec,      SOFT},
{"forwarder",       apexForward,        4000, 110, ZERO_TIME_VALUE,  ZERO_TIME_VALUE,SOFT}
```

- Most ARINC process attributes map directly to a vxWorks task attribute
- Period defines time when each activation of process is to occur
  - If period is zero (aperiodic), then normal priority based pre-emptive scheduling will occur
- Time capacity is elapsed time allowed for process to complete its execution before invoking deadline action or needs to be replenished

Slide 14

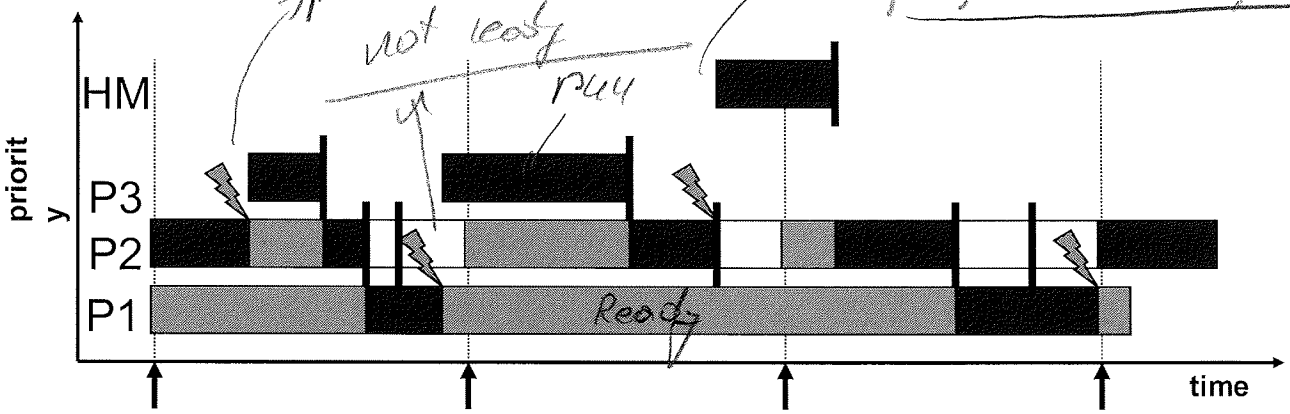
© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Process Management Example

3 Processes plus a Health Monitor thread



- P2 is a periodic process scheduled to start at points specified on horizontal axis
  - Health Monitor kicks in when P2 overruns deadline (second orange bar)
  - In this case a "soft" deadline has been hit, logged, and the system is allowed to continue

Slide 15

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- Red – Executing
- Green – Ready
- Yellow – Not Ready

ARINC 653 API

Introduction
Partition Management
Process Management
Time Management
Inter Partition Communication
Intra Partition Communication
Health Monitoring

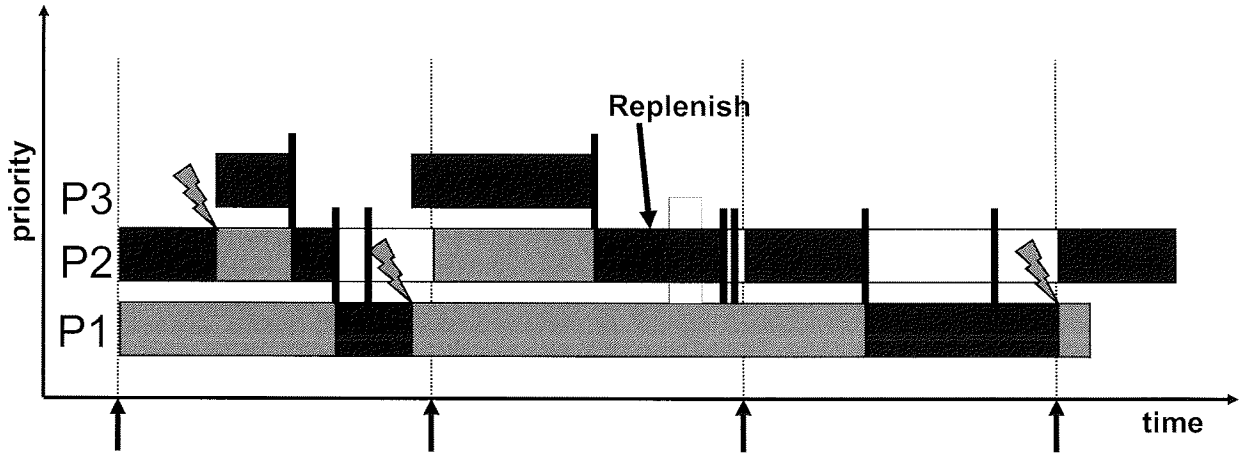
Slide 16

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Time Management Example



- Same example as before, except P2 realizes it needs more time and sends a replenish to move the deadline
- If deadline is reached while partition is not running, it will be deferred until next activation of partition and then dealt with

Slide 17

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- Red – Executing
- Green – Ready
- Yellow – Not Ready

Time Management API

- Support for periodic and aperiodic processes
  - ***TIMED_WAIT*** ***PERIODIC_WAIT***
  - ***GET_TIME*** ***REPLENISH***

- Underlying partition scheduling is transparent to application
- Deadlines can be adjusted programmatically using ***REPLENISH***
  - E.g awaiting I/O

Slide 18

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

ARINC 653 API

Introduction
Partition Management
Process Management
Time Management
Inter Partition Communication
Intra Partition Communication
Health Monitoring

Slide 19

© 2008 Wind River Systems, Inc.

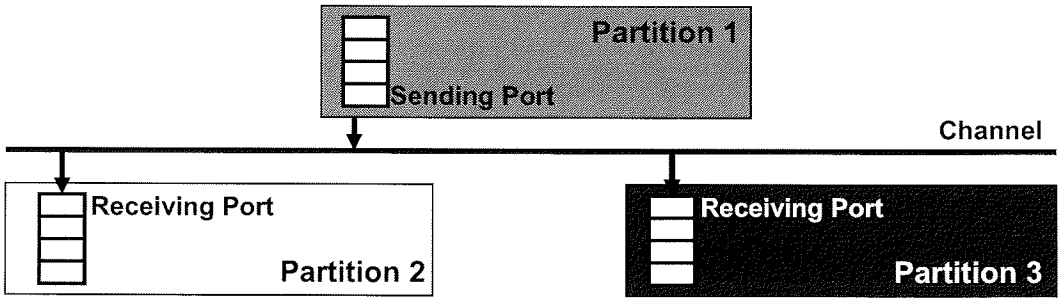
WIND RIVER

Student Notes

Sampling Ports API

- Overwrites, fixed length, time stamped (delivery time, validity time)
- API

CREATE_SAMPLING_PORT ***WRITE_SAMPLING_PORT***
READ_SAMPLING_PORT ***GET_SAMPLING_PORT_ID***
GET_SAMPLING_PORT_STATUS



Slide 20

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- Communication between two or more partitions executing either on the same core module or on different core modules
- All inter-partition communication is conducted via messages sent through ports
- A message is defined as a continuous block of data of finite length
- A message is sent from a single source to one or more destinations
- The destination of a message is a partition, and not a process within a partition
- A channel is a connection of a source and destination
- The system integrator (not the application developer) configures the channel connections
- Two types of ports: Sampling and Queuing
- Required memory is allocated from the Module OS's memory pool

Queuing Ports API

- Queuing ports properties
  - Stored messages, variable length, FIFO or PRIORITY
- API

  CREATE_QUEUING_PORT
  SEND_ QUEUING _~~PORT~~ MESSAGE
  RECEIVE_ QUEUING _~~PORT~~ MESSAGE
  GET_ QUEUING _PORT_ID
  GET_ QUEUING _PORT_STATUS

Slide 21

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

ARINC 653 API

Introduction
Partition Management
Process Management
Time Management
Inter Partition Communication
Intra Partition Communication
Health Monitoring

Slide 22

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

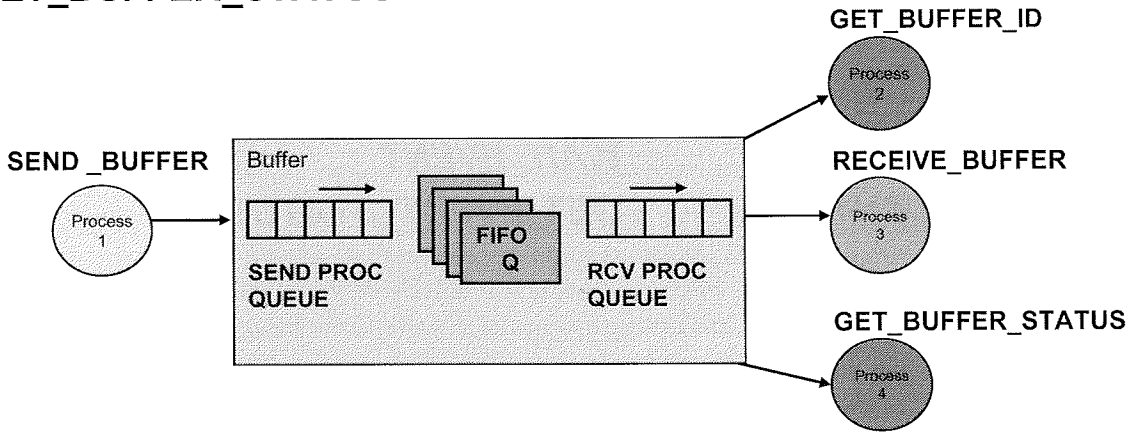
Buffers

Buffers consist of queued messages

CREATE_BUFFER ***SEND_BUFFER***

RECEIVE_BUFFER ***GET_BUFFER_ID***

GET_BUFFER_STATUS



Slide 23

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Semaphores and Events

- Counting semaphores are used for synchronization
  - *CREATE_SEMAPHORE*
  - *WAIT_SEMAPHORE*
  - *SIGNAL_SEMAPHORE*
  - *GET_SEMPAHORE_ID*
  - *GET_SEMAPHORE_STATUS*
- Events are used for notification of occurrence of a condition
  - *CREATE_EVENT*
  - *SET_EVENT*
  - *RESET_EVENT*
  - *WAIT_EVENT*
  - *GET_EVENT_ID*
  - *GET_EVENT_STATUS*

Slide 24

© 2008 Wind River Systems, Inc.

WIND RIVER

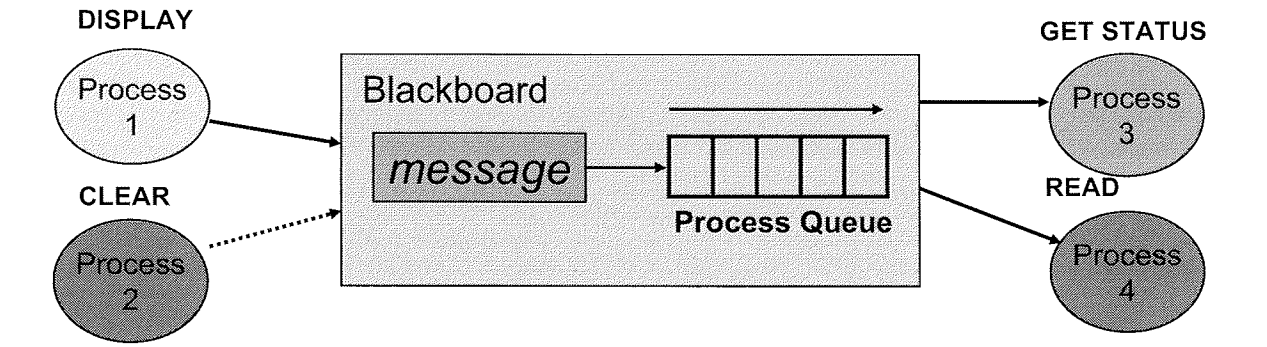
Student Notes

Blackboards

On a blackboard, a message is overwritten by the next message, or it can be cleared

- Processes can queue for a message but process reads latest version of data on blackboard
- API

CREATE_BLACKBOARD	*CLEAR_ BLACKBOARD*
GET_BLACKBOARD_STATUS	*READ_BLACKBOARD*
GET_BLACKBOARD_ID	*DISPLAY_BLACKBOARD*



Slide 25

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

ARINC 653 API

Introduction
Partition Management
Process Management
Time Management
Inter Partition Communication
Intra Partition Communication
Health Monitoring

Slide 26

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Process Level Health Monitoring API

- ***CREATE_ERROR_HANDLER***
  - Creates an error handler for the current partition
- ***GET_ERROR_STATUS***
  - Gets the error status of the first process in the error list
- ***RAISE_APPLICATION_ERROR***
  - Invokes the error handler process for a specific error
    - Can be invoked to have the handler just report a message
- ***REPORT_APPLICATION_MESSAGE***
  - Transmits a message to the HM function

Users uses time

© 2008 Wind River Systems, Inc.

Student Notes

Invokes the schedule 0 → modify to ~~Process~~ Partitions.

© 2008 Wind River Systems, Inc.

Health Monitoring Error Codes

These error codes can be raised by the function:
RAISE_APPLICATION_ERROR

DEADLINE_MISSED	***APPLICATION_ERROR***
NUMERIC_ERROR	***ILLEGAL_REQUEST***
STACK_OVERFLOW	***MEMORY_VIOLATION***
HARDWARE_FAULT	***POWER_FAIL***

4.10.2 → Γιατί είναι χρήσιμη η συνάρτηση

Slide 28

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Review

In this chapter you learned:

- The ARINC653 API allows the underlying implementation to be abstracted away so an application developer has increased portability
- It provides a mechanism to facilitate the reuse of software components across multiple systems while alleviating porting issues
- It permits independently developed applications to execute together on the same hardware
- VxWorks 653 fully supports this API, including Supplement 2 changes

Slide 29

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Questions

1. ARINC 653 specification includes interface requirements between the application software and the OS. True or False?
2. Time Management:
   a. Is based on taskDelay
   b. Provides an infrastructure to handle error reporting and response.
   c. Is based on watchdogs
   d. Is based on a VxWorks task
   e. All of the above
3. Partitions must be set to _____ to begin operation after creation.
4. The ***REPORT_APPLICATION_MESSAGE*** function
   a. Invokes the error handler process for a specific error
   b. Gets the error status of the first process in the error list
   c. Transmits a message to the HM function
   d. All of the above
5. Counting semaphores are used for _____

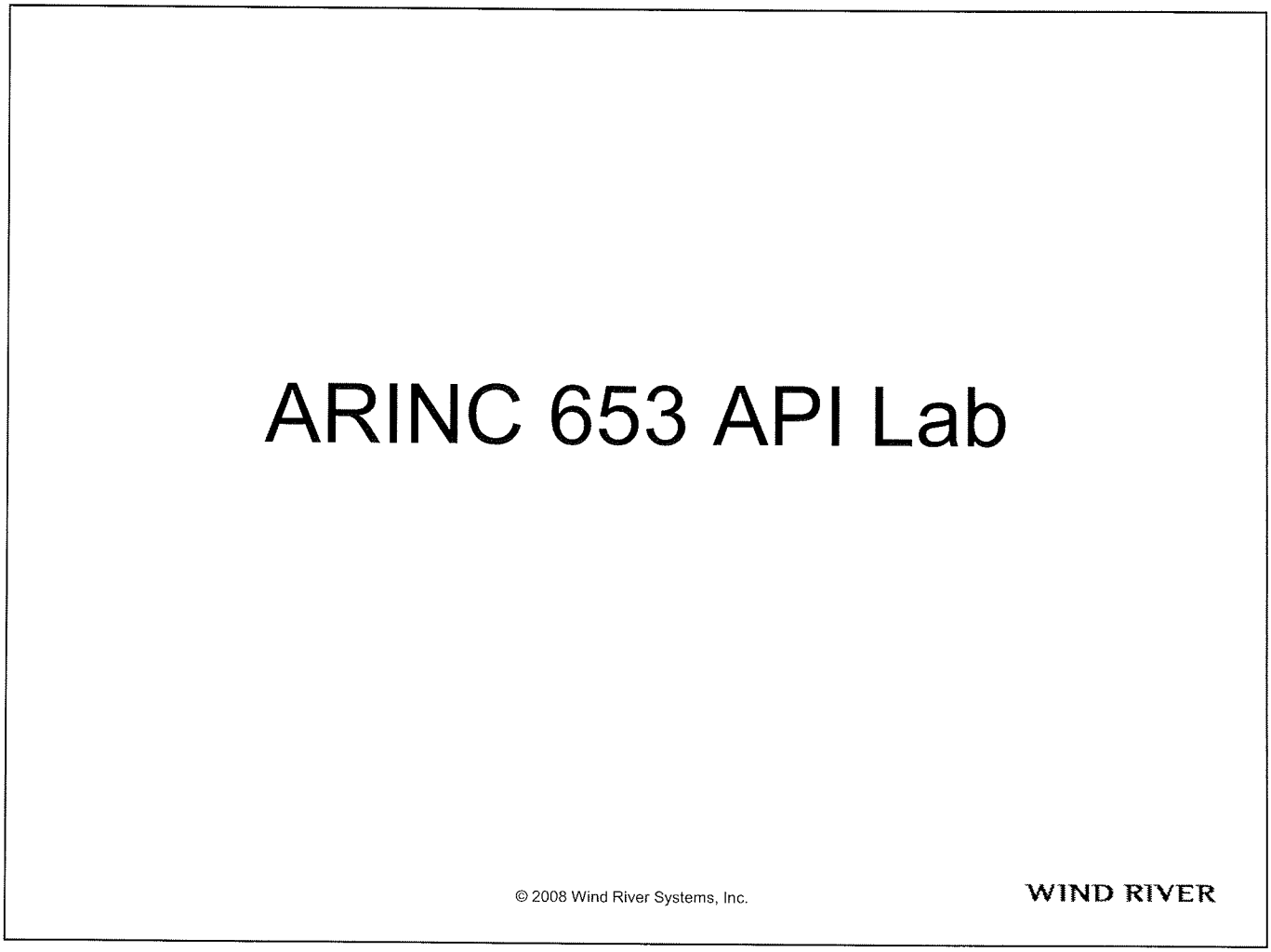
Slide 30

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

1. True
2. A and C
3. Normal
4. C
5. synchronization



Student Notes

© 2008 Wind River Systems, Inc.

Objectives

By the end of this chapter you will be able to understand:

- Process Scheduling
- Process Synchronization
- Protecting Shared Resources
- Intra Partition Communication
- Inter Partition Communication
- Process Health Monitoring

Slide 2

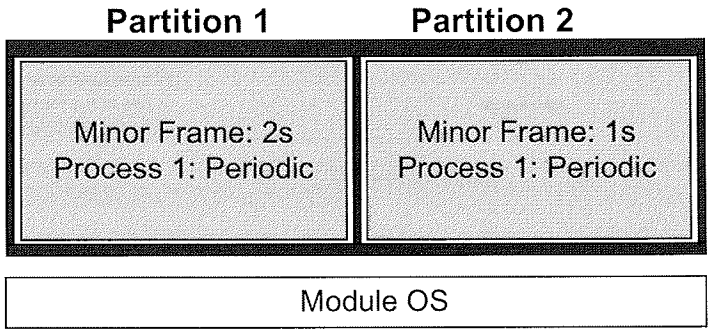
© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Process Scheduling: Overview

- ARINC 653 process scheduling in a partition
- Initial system is composed of 2 partitions, each containing the same single periodic process with attributes
  - Period, 8s
  - Time capacity, 8s



Slide 3

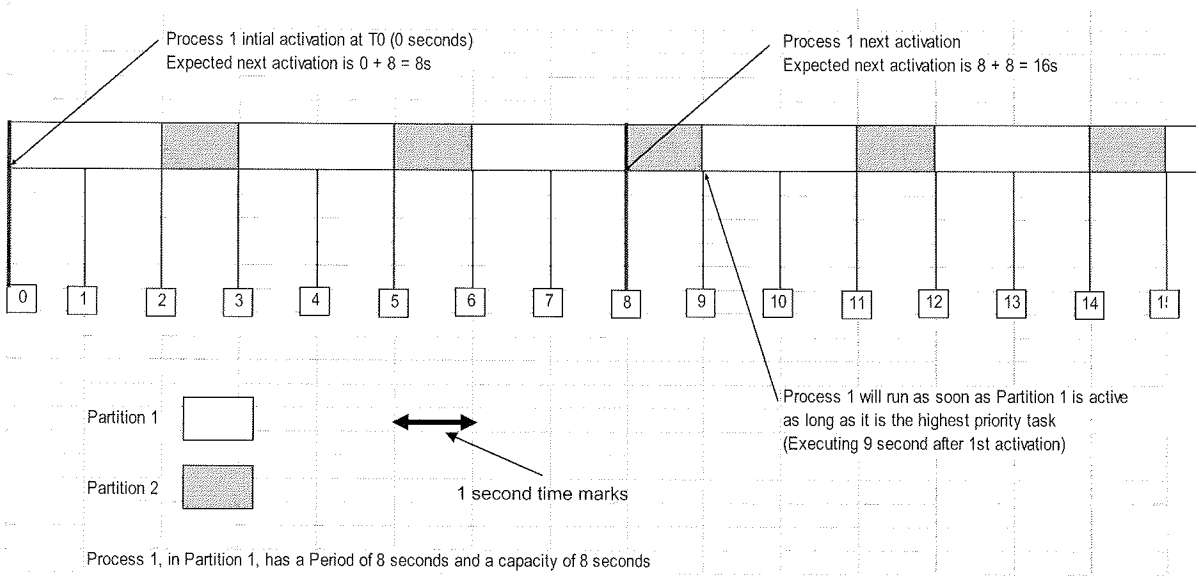
© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- Periodic processes use Absolute time, i.e. wall clock time

Periodic Process Example



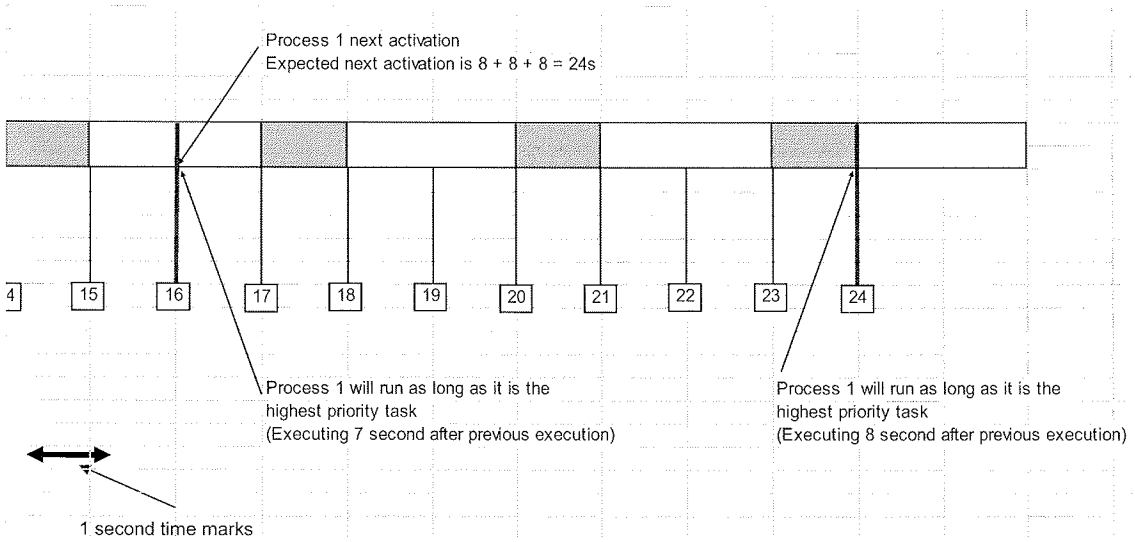
Slide 4

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Periodic Process Example



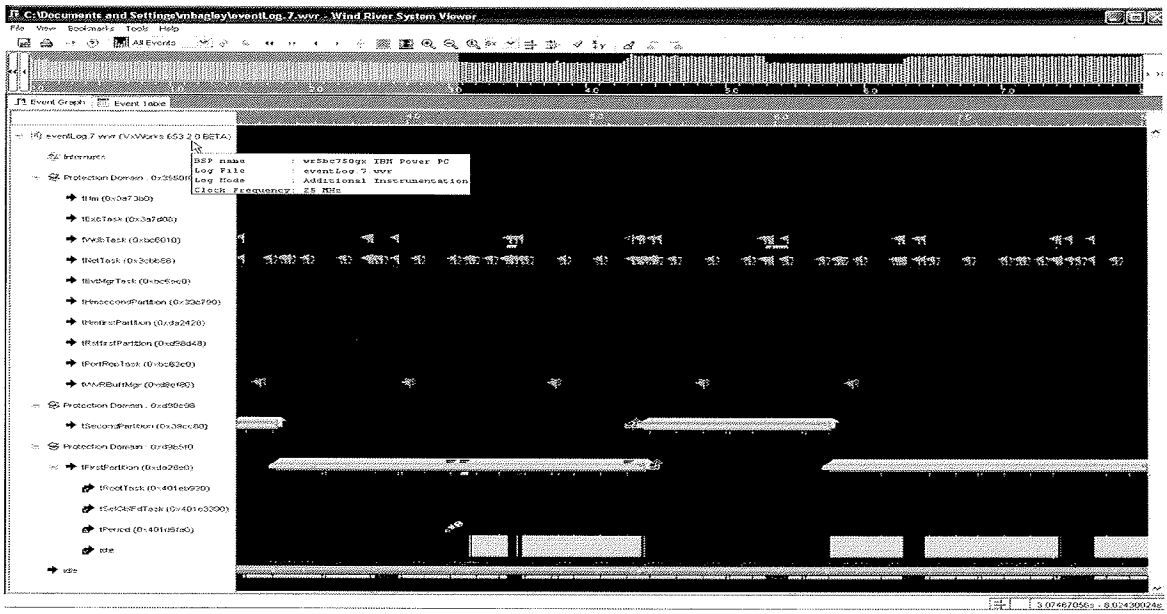
Slide 5

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

System Viewer Display of Periodic Processes



© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Relevant API

- period.c
  - CREATE_PROCESS
  - START
  - SET_PARTITION_MODE
  - PERIODIC_WAIT
- background.c
  - CREATE_PROCESS
  - START

Slide 7

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- **CREATE_PROCESS**
  - Creates a process
  - Returns an identifier that denotes the created process
- **START**
  - Initializes all attributes of a process to their default values,
  - Resets the runtime stack of the process
- **SET_PARTITION_MODE**
  - Sets the operating mode of the current partition to normal after the application portion of the initialization of the partition is complete
  - This routine is also expected to be used for setting the partition back to IDLE (partition shutdown), and to COLD_START/WARM_START, when a serious fault is detected and processed
- **PERIODIC_WAIT**
  - Suspends the execution of the requesting process until the next release point in the processor time line that corresponds to the period of the process

ARINC 653 API Lab

Process Scheduling
Process Synchronization
Protecting Shared Resources
Intra Partition Communication
Inter Partition Communication
Process Health Monitoring

Slide 8

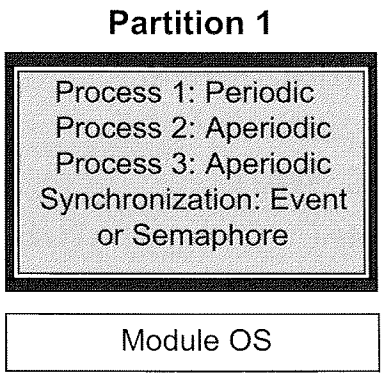
© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Process Synchronization: Overview

- ARINC 653 process synchronization within a partition
  - Using events
  - Using semaphores
- System is composed of a single partition containing
  - A periodic process
  - Two aperiodic processes



© 2008 Wind River Systems, Inc.

Student Notes

© 2008 Wind River Systems, Inc.

Relevant API

- synchro_event.c
  - CREATE_EVENT
  - PERIODIC_WAIT
  - SET_EVENT
  - RESET_EVENT
  - WAIT_EVENT
- synchro_semaphore.c
  - CREATE_SEMAPHORE
  - PERIODIC_WAIT
  - GET_SEMAPHORE_STATUS
  - SIGNAL_SEMAPHORE
  - WAIT_SEMAPHORE

Slide 10

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- CREATE_PROCESS, START, and SET_PARTITION_MODE are assumed

ARINC 653 API Lab

Process Scheduling
Process Synchronization
Protecting Shared Resources
Intra Partition Communication
Inter Partition Communication
Process Health Monitoring

Slide 11

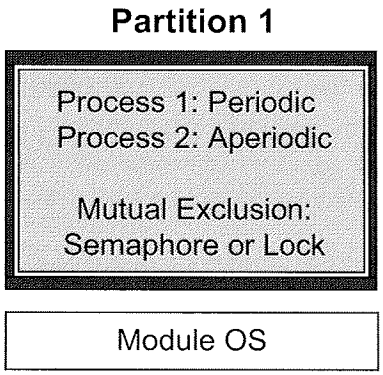
© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Protecting Shared Resources: Overview

- ARINC 653 exclusive access mechanisms within a partition
  - Using semaphores
  - Using preemption locks
- System is composed of a single partition containing
  - A periodic process
  - An aperiodic processes



Slide 12

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Relevant API

- mutex_semaphore.c
  - CREATE_SEMAPHORE
  - PERIODIC_WAIT
  - SIGNAL_SEMAPHORE
  - WAIT_SEMAPHORE
- mutex_preemptionlock.c
  - PERIODIC_WAIT
  - LOCK_PREEMPTION
  - UNLOCK_PREEMPTION

Slide 13

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- CREATE_PROCESS, START, and SET_PARTITION_MODE are assumed

ARINC 653 API Lab

Process Scheduling
Process Synchronization
Protecting Shared Resources
Intra Partition Communication
Inter Partition Communication
Process Health Monitoring

Slide 14

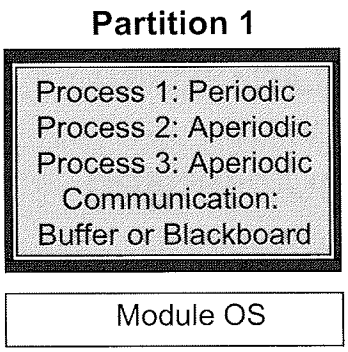
© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Intra Partition Communication: Overview

- ARINC 653 communication mechanisms within a partition
  - Using buffers
  - Using blackboards
- System is composed of a single partition containing
  - A periodic process
  - Two aperiodic processes



© 2008 Wind River Systems, Inc.

Student Notes

© 2008 Wind River Systems, Inc.

Relevant API

- intracomm_buffer.c
  - CREATE_BUFFER
  - PERIODIC_WAIT
  - SEND_BUFFER
  - RECEIVE_BUFFER
- intracomm_blackboard.c
  - CREATE_BLACKBOARD
  - PERIODIC_WAIT
  - DISPLAY_BLACKBOARD
  - READ_BLACKBOARD
  - TIMED_WAIT
  - CLEAR_BLACKBOARD

Slide 16

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- CREATE_PROCESS, START, and SET_PARTITION_MODE are assumed

ARINC 653 API Lab

Process Scheduling
Process Synchronization
Protecting Shared Resources
Intra Partition Communication
Inter Partition Communication
Process Health Monitoring

Slide 17

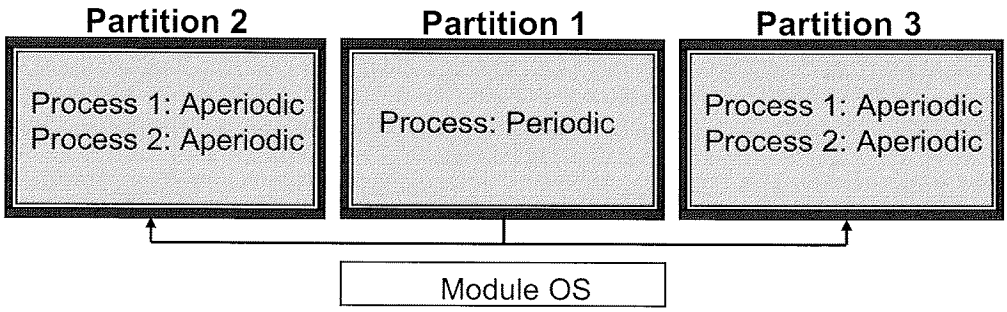
© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Inter Partition Communication: Overview

- ARINC 653 communication mechanisms between partitions
  - Using queuing ports
  - Using sampling ports
- System composed of 3 partitions: Partition 1 communicates with partitions 2 and 3 using either queuing or sampling ports



Slide 18

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Relevant API

- intercomm1_queuing.c
  - CREATE_QUEUING_PORT
  - PERIODIC_WAIT
  - SEND_QUEUING_MESSAGE
- intercomm2_queuing.c
  - CREATE_QUEUING_PORT
  - RECEIVE_QUEUING_MESSAGE
- intercomm1_samplingport.c
  - CREATE_SAMPLING_PORT
  - PERIODIC_WAIT
  - WRITE_SAMPLING_MESSAGE
- intercomm2_samplingport.c
  - CREATE_SAMPLING_PORT
  - PERIODIC_WAIT
  - READ_SAMPLING_MESSAGE
  - SUSPEND_SELF

Slide 19

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- CREATE_PROCESS, START, and SET_PARTITION_MODE are assumed

ARINC 653 API Lab

Process Scheduling
Process Synchronization
Protecting Shared Resources
Intra Partition Communication
Inter Partition Communication
Process Health Monitoring

Slide 20

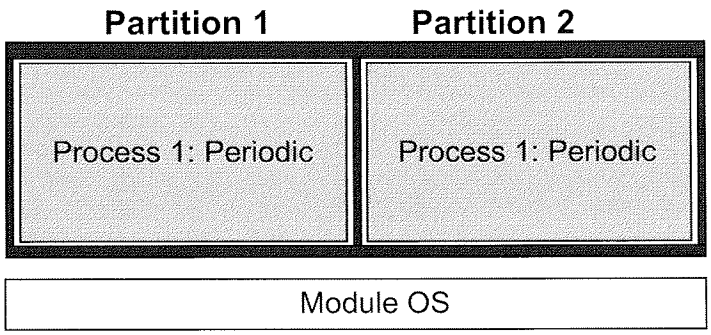
© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Process Health Monitoring: Overview

- ARINC 653 process health monitoring in a partition
- System composed of 2 partitions, each containing the same single periodic process
  - A process level health monitor routine is also registered in each partition



Slide 21

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Relevant API

procHm.c
- CREATE_ERROR_HANDLER
- PERIODIC_WAIT
- GET_ERROR_STATUS
- STOP_SELF

Slide 22

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

- CREATE_PROCESS, START, and SET_PARTITION_MODE are assumed

Review

In this chapter you learned about:

- Process Scheduling
- Process Synchronization
- Protecting Shared Resources
- Intra Partition Communication
- Inter Partition Communication
- Process Health Monitoring

Slide 23

© 2008 Wind River Systems, Inc.

WIND RIVER

Student Notes

Input/Output Partition

Slide 1

© 2008 Wind River

WIND RIVER

Student Notes

© 2008 Wind River Systems, Inc.

Objectives

By the end of this chapter you will be able to:

- Understand the use of the Device Drivers with the system using three different models
- Use the ARINC ports and Device Drivers

Slide 2

© 2008 Wind River

WIND RIVER

Student Notes

Input/Output (I/O) Partition

I/O Device Driver Models

ARINC Ports and Device Drivers

I/O Partition Lab

Slide 3

© 2008 Wind River

WIND RIVER

Student Notes

Device Driver Models

Device drivers can be configured into the system using three different models:

- Partition Level device driver
  - Device driver is located entirely in a partition
    - Typically an I/O partition → inno partyon
- Split device driver
  - Device driver is split between a partition
    - Typically between an I/O partition and Module OS
- Module OS device driver
  - Device driver is located entirely in Module OS

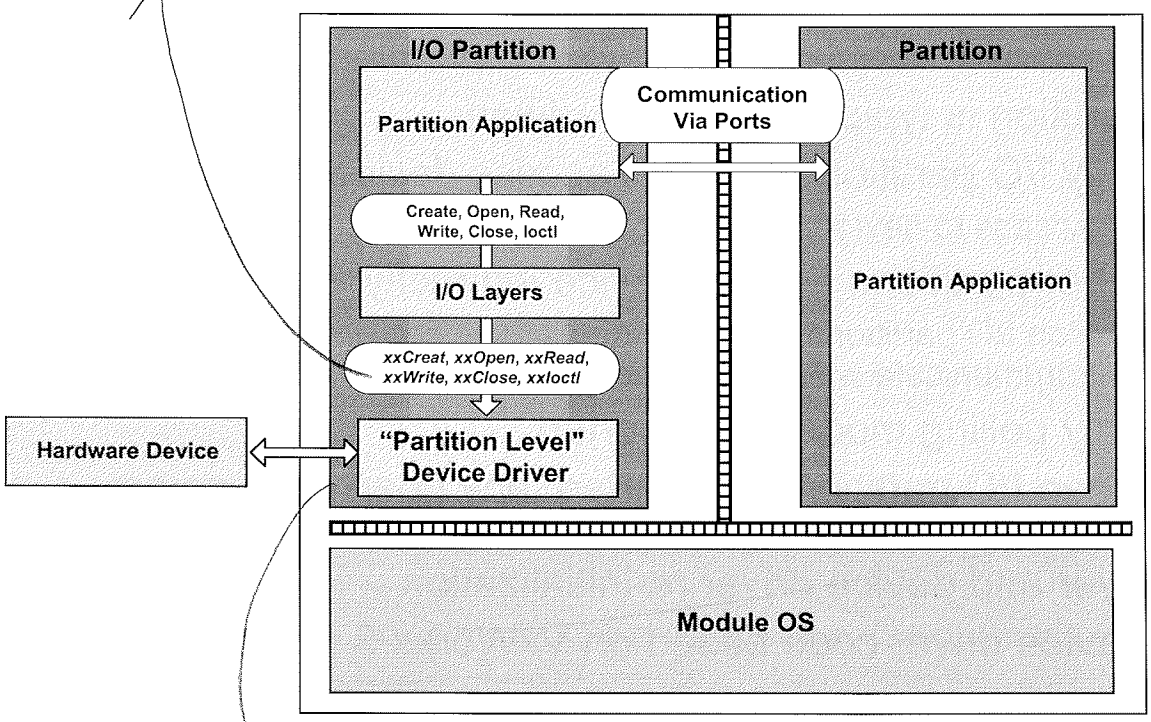
Slide 4

© 2008 Wind River

WIND RIVER

Student Notes

Partition Level Device Driver



Slide 5

© 2008 Wind River

WIND RIVER

Student Notes

- Standard VxWorks device driver model

Partition Level Device Driver Model

Device driver is entirely located in a partition

- Partition is typically an I/O partition
- Only I/O partition code can access device directly
  - Access to device does not generate system calls to Module OS
- Other partitions access device indirectly via ports
  - Non I/O partitions rely on a local I/O device driver that uses ports to communicate with I/O partition
- Only polling mode is supported
- When I/O partition is restarted, any open devices are automatically closed, I/O partition is restarted including I/O layer and device driver re-initialization
- Device driver code based on VxWorks 5.x code

Slide 6

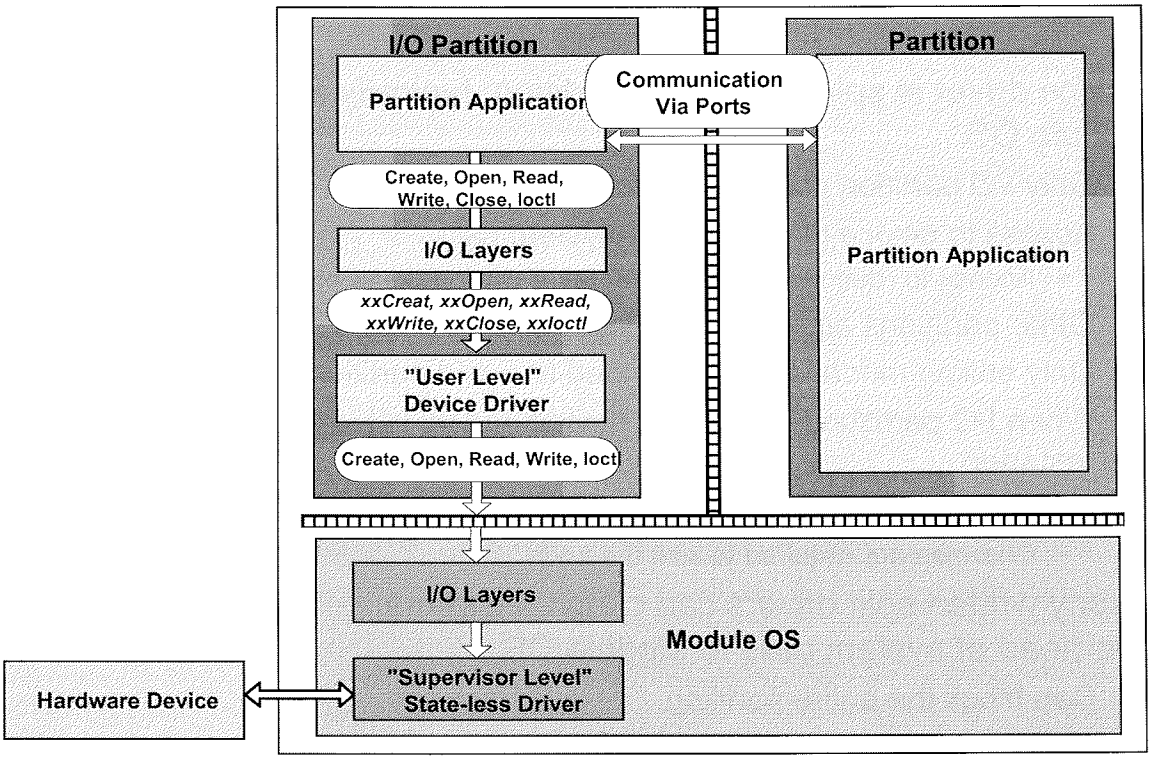
© 2008 Wind River

WIND RIVER

Student Notes

Advantage:
-> no system calls
(no swedish between user and supervisor)

Split Device Driver



Slide 7

© 2008 Wind River

WIND RIVER

Student Notes

- A stateless driver cannot keep the notion of partition state, especially regarding restarts

- Tylko partycja I/O ma dostęp do sterowników

Split Device Driver Model

- Device driver is split between a partition, typically an I/O partition, and Module OS
  - Only I/O partition code can access driver directly
  - Access to device may generate system calls
    - Access to routines in 'supervisor level' device driver portion
  - Other partitions access device indirectly via ports
  - Only polling mode is recommended to avoid partitioning issues
  - When I/O partition is restarted, device and its I/O layers are automatically restarted
  - Device driver code based on VxWorks 5.x code
    - Changes are required to split the code
- Poorly written split drivers can cause performance problems due to overhead of system calls to Module OS
  - Can block the entire partition

Slide 8

© 2008 Wind River

WIND RIVER

Student Notes

- Issues: Violation of partitioning

We may use interrupts, although pooling should be use.

© 2008 Wind River Systems, Inc.

Split Device Driver Model Considerations

- Reason to split a driver
  - Hardware control registers are not isolated on a separate MMU page from other devices not controlled by the same partition
  - Registers are only accessible by the Module OS
- 'Supervisor level' device driver should contain specific code
  - Executing supervisor CPU instructions
  - Posing a security issue for whole system
    - For example, access to device control registers, DMA control registers, etc.
- Module OS device driver must not call any blocking core OS routines
  - For example, ***semTake***, ***malloc***
- E.g. graphics drivers, initiate DMA's at the beginning of the window

Slide 9

© 2008 Wind River

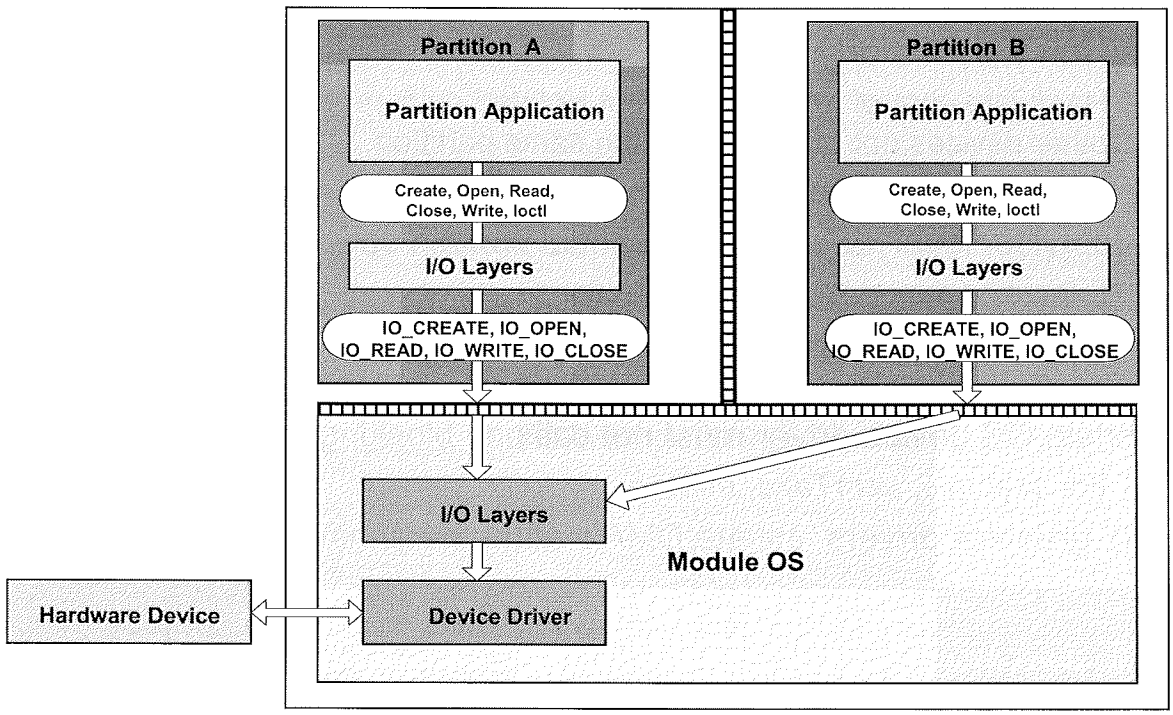
WIND RIVER

Student Notes

- Per ARINC 653, spatial partitioning rules says application code in one partition cannot affect the operation or memory of another partition
  - Most target hardware sets contain sets of hardware device control and status registers, registers for difference devices (Serial and Ethernet for example) and / or different channels on the same device type (Serial channel 0, Serial channel 1) are not separated on different MMU pages
  - Therefore, if a partition is allowed access to one set of registers, generally it ends up with access to all registers
    - That partition can then read and write registers for devices it is not suppose to have access to
    - This breaks ARINC 653 spatial partitioning rules

App. Prog. → 9.2.2
Guide

Module OS Device Driver



Slide 10

© 2008 Wind River

WIND RIVER

Student Notes

© 2008 Wind River Systems, Inc.

Module OS Device Driver Model

- Device driver located entirely in Module OS
  - All partitions can access device via ***open***, ***read***, ***write***, ***close***, ***ioctl*** calls
  - Interrupt and/or polling modes are supported
    - Interrupt mode should only be used if interrupts are deterministic
  - Device driver code is based on VxWorks 5.x code
- Reasons for a Module OS driver
  - Hardware control registers are not isolated on a separate MMU page from other devices not controlled by same partition
  - Other processing is a very small percent of processing time
    - System call overhead would be high percent of processing
- Partition Restart
  - Module OS forces any pending I/O system calls for this partition to complete by issuing a specific IOCTL code (**FIORESET**) to file descriptor being accessed
  - All Module OS file descriptors owned by partition are closed
- System WARM Restart
  - Module OS device drivers (except system clock) must be re-initialized in **HME_POWER_FAIL** handler
  - System integrator must ensure that the state of Module OS I/O layers is re-synchronized with new state of these device drivers

Slide 11

© 2008 Wind River

WIND RIVER

Student Notes

- A ~~split driver~~ uses the I/O system calls (**open, close, read, write, ioctl**) to access the core driver
  - This access is costly (about 6 microseconds per call on a 400 Mhz processor)
    - A poorly written split driver can end up spending more time in the I/O layer than in the actual driver doing useful work
    - If a driver is I/O intensive, then by nature, most of its work must be done in the core level driver (due to the MMU restrictions)
      - Therefore, the cost of development of a split driver may be higher than any benefit derived from it

I/O Partition

I/O Device Driver Models

ARINC Ports and Device Drivers

I/O Partition Lab

Slide 12

© 2008 Wind River

WIND RIVER

Student Notes

ARINC Ports and Device Driver

- VxWorks 653 offers multiple methods to connect standard ARINC ports (sampling, queuing) to device drivers
  - Local (partition to partition) ports – no driver
  - Pseudo ports
    - Described in the ARINC 653 specification as a method to implement off module communications
    - Allows connection to an implementation specific physical device driver
  - Direct Access ports
    - Partition level pseudo-ports
    - Module OS level

WR implementation

- Allow developers to decide which requirements fit from a portability and performance aspect

Slide 13

© 2008 Wind River

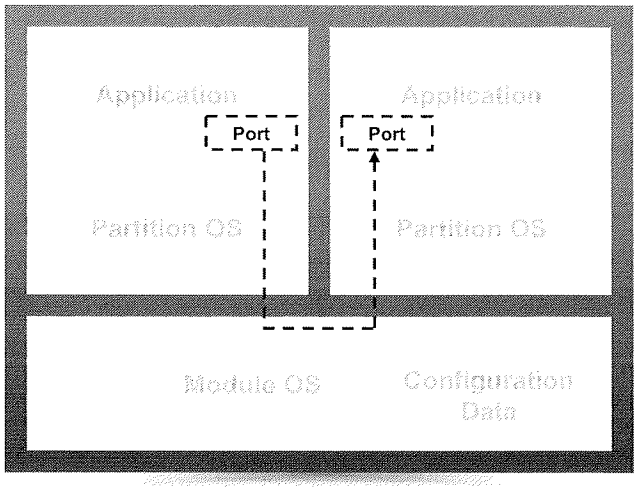
WIND RIVER

Student Notes

- Defined in the specification

Local Ports

- Local ports allow communication between partitions.
  - ARINC standard implementation
  - The connection is buffered.
  - The connection is configurable.
  - The connection can be one-to many
  - It is slower than direct access ports.
  - It is comprised of one source, at least one destination and a channel.



Slide 14

© 2008 Wind River

WIND RIVER

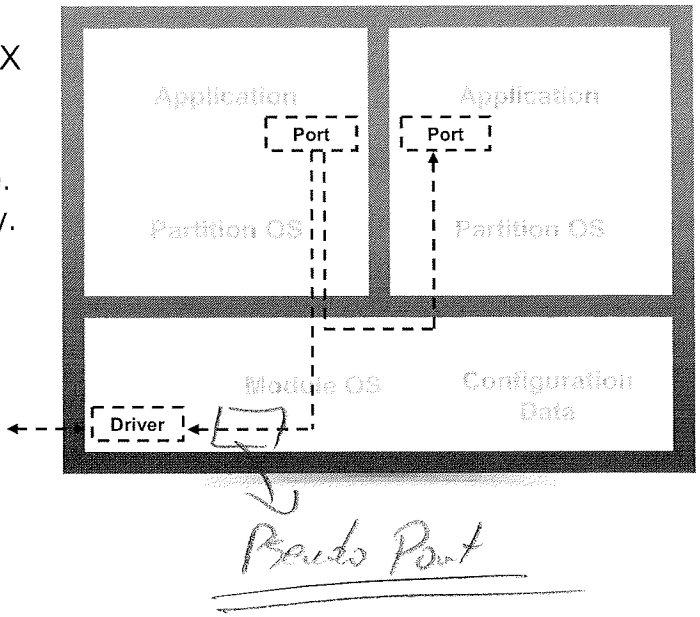
Student Notes

- Local ports are the ARINC standard port mechanism as defined in the 653P1-2 specification
- No externally supplied device driver needed.
- Channels are configured in the Module XML file, source and destination in the application XML file.
- Applications then call create routines during initialization to establish communication path

Proj. Guido 7.15.5

Pseudo Ports

- Pseudo ports allow communication with other modules over a physical transport mechanism such as an AFDX network
  - The connection is buffered.
  - The connection is configurable.
  - The connection is one-to-many.
  - Only one of the destination connections can be a pseudo-port
  - It is slower than direct access ports.
  - It uses a standard VxWorks device driver model.



Slide 15

© 2008 Wind River

WIND RIVER

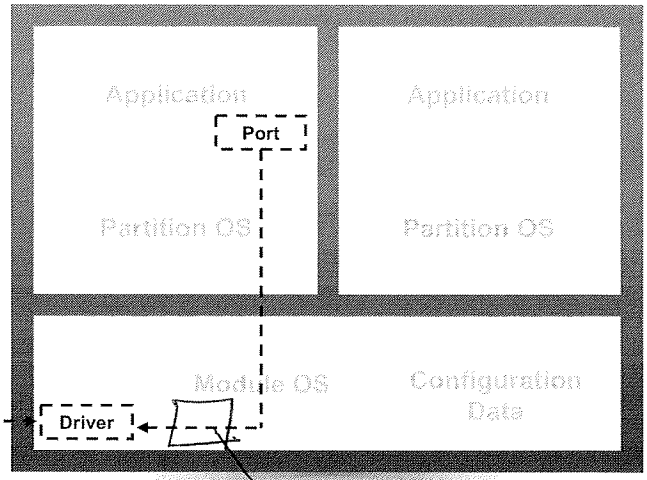
Student Notes

- This mechanism employs a Module OS resident device driver to provide interface to the external module
- VerIMAx enforces the one pseudo-port destination at build time → configuration check
- Configuration of device driver to port done in Module XML
- Other configuration the same as local ports (previous slide).

→ Do komunikacji między modułami

Direct Access Ports

- Direct access ports also allow communication with other targets over a physical transport.
  - There is no message buffering in the module OS.
  - It is faster than pseudo ports.
  - Communication must be one-to-one.
  - Communication must be hard-coded.



Slide 16

© 2008 Wind River

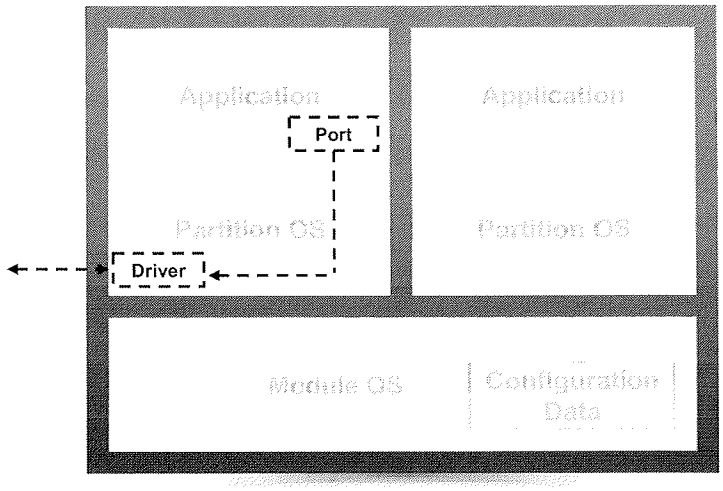
WIND RIVER

Student Notes

- This mechanism employs a Module OS resident device driver to provide interface to the external module
- Any buffering must be performed by either the device itself or the driver
  - The OS does not provide any (except in the case of defined alignment needs).
  - To do alignment requires copying
- XML attribute ***DIRECT_ACCESS*** is a Wind River extension
- Not covered in 653 specification

Partition Level Pseudo Ports

- Partition pseudo ports are direct access ports with a user-mode driver.
  - There is no message buffering
  - It is faster than pseudo ports.
  - Communication must be one-to-one.
  - Communication must be hard-coded.
  - There is polling mode only.
- Benefit: No system calls



Slide 17

© 2008 Wind River

WIND RIVER

Student Notes

- Configuration defined in the application XML
- Faster than pseudo-port in the Module OS since no system call takes place
- Polling mode only, since it is done at the user level

© 2008 Wind River Systems, Inc.

I/O Partition

I/O Device Driver Models
ARINC Ports and Device Drivers
I/O Partition Lab

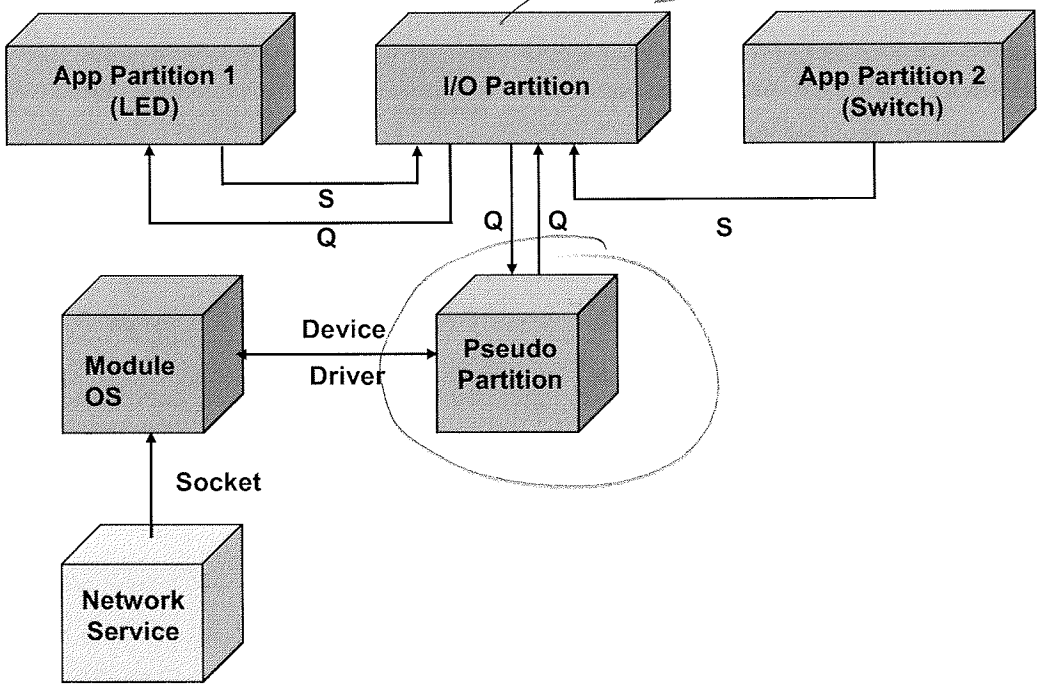
Slide 18

© 2008 Wind River

WIND RIVER

Student Notes

Overview



Slide 19

© 2008 Wind River

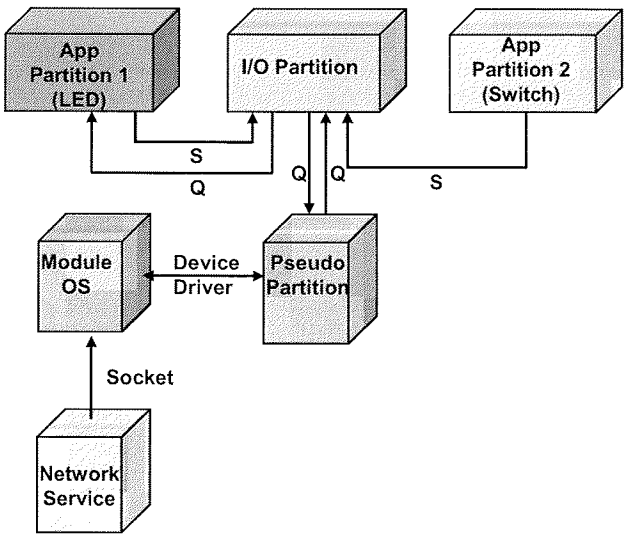
WIND RIVER

Student Notes

© 2008 Wind River Systems, Inc.

LED Partition

- 4 user LEDs are mapped to LED partition's virtual address space
- It receives requests from I/O partition, using a queuing port
- Those requests are:
  - Set LED n on
  - Clear LED n
  - Report LED status
- Process in partition acts on these requests
- It then replies with action taken, using a sampling port
- Application code is contained in **LedApp.c**



Slide 20

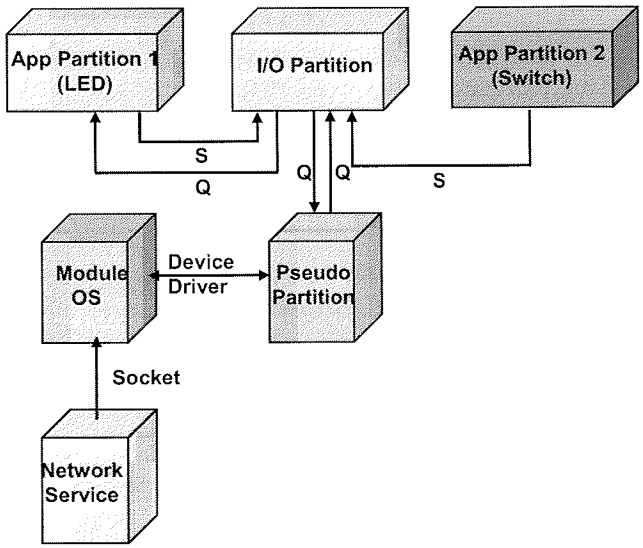
© 2008 Wind River

WIND RIVER

Student Notes

Switch Partition

- 4 user switches are mapped to the Switch partition's virtual address space
- Process within partition periodically samples state of switches
- When switch status changes, it updates status to I/O partition using a sampling port
- Application code is contained in **SwrApp.c**



Slide 21

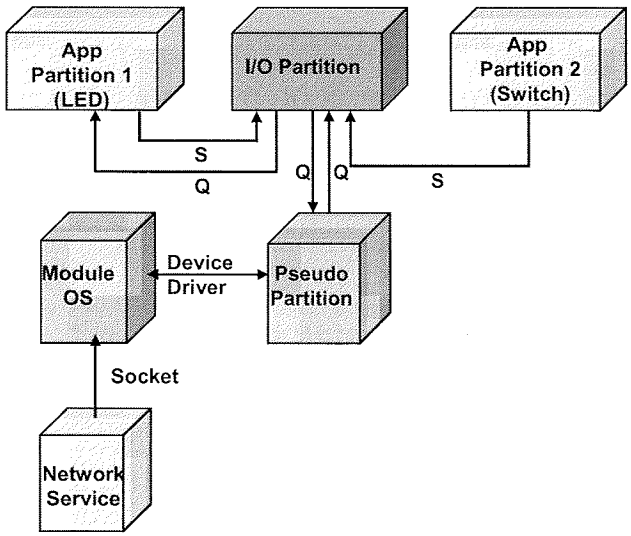
© 2008 Wind River

WIND RIVER

Student Notes

I/O Partition

- I/O partition receives requests from a queuing port connected to network services in the core OS
- Messages received are requests for status or state changes for devices controlled in the other two partitions
- It decodes these requests and communicates those requests to appropriate partition for disposition
- Status changes from the application partitions are communicated back via a separate queuing port back to network service



Slide 22

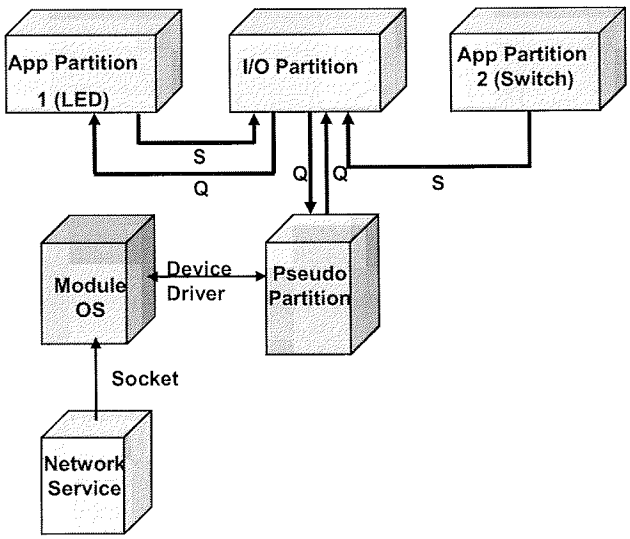
© 2008 Wind River

WIND RIVER

Student Notes

Channel Configuration

- Channel 1
  - sender_iopart → dest_coreos
- Channel 2
  - sender_coreos → dest_iopart
- Channel 3
  - led_setdata_in → led_setdataout
- Channel 4
  - led_readdata_out → led_readdata_in
- Channel 5
  - sw_data_out → sw_data_in



Slide 23

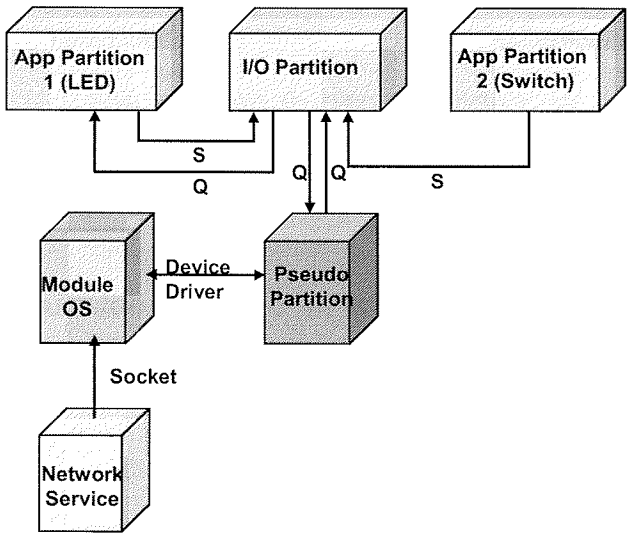
© 2008 Wind River

WIND RIVER

Student Notes

Pseudo Partition

- pseudoPortCreate
  - Port creation routines
- 2 options
  - Local Loopback
  - I/O Simulator Tool
- Processing of data occurs in time frame assigned to I/O partition



Slide 24

© 2008 Wind River

WIND RIVER

Student Notes

Questions

1. For Module OS device driver models, the driver is split between a partition and the Module OS. True or False?
2. For Partition Level device drivers:
   a. The partition is typically an I/O partition
   b. Any partition code can access device directly
   c. Other partitions access devices indirectly
   d. Only polling mode is supported
   e. All of the above
3. Local ports allow communication between _____
4. Communication that uses direct access ports must be 1-to-1. True or False?
5. With a Module OS device driver, partitions can access the device using the following:
   a. Open
   b. Read
   c. Write
   d. Close
   e. All of the above

Slide 25

© 2008 Wind River

WIND RIVER

Student Notes

1. False
2. A, C and D
3. Partitions
4. True
5. E

Review

In this chapter you learned to:

- Configure the Device Drivers into the system using three different models
- Use the ARINC ports and Device Drivers

Slide 26

© 2008 Wind River

WIND RIVER

Student Notes

Printed in the UK