Programowanie współbieżne i rozproszone – laboratorium

Instrukcja do ćwiczenia laboratoryjnego IV

Wątki

Autor: S. Samolej

Rzeszów, 2013

Podziękowania:
Składam podziękowania panu dr inż. Jędrzejowi Ułasiewiczowi za udostepnienie materiałów wykładowych i laboratoryjnych, które były inspiracją do opracowania tych materiałów.

1. Proszę skompilować i uruchomić przykładowy program „thread1.c”. Aby kompilacja tego i pozostałych programów w ćwiczeniu przebiegła pomyślnie trzeba dołączyć w opcjach konsolidatora bibliotekę **pthread** (Build Options->Linker settings->Link Libraries: pthread). W programie pracują 2 wątki (jeden –główny wątek programu, a drugi – nowy jawnie powołany). Proszę zwrócić uwagę technikę powoływania nowych wątków, oczekiwania na zakończenie potomnych wątków oraz na sposób przekazywania parametrów do powoływanych wątków. Wątki współdzielą pamięć procesu je powołującego, więc możliwa jest modyfikacja zmiennych globalnych z poziomu wątków.
2. Proszę skompilować i uruchomić przykładowy program „thread2.c”. Program „udowadnia”, że następuje automatyczne, niewymuszone przełączanie pomiędzy wątkami w czasie ich pracy. Modyfikowana jest współdzielona zmienna i w zależności od jej stanu wątki wypisują pewną informację („1” lub „2”) i zmieniają stan zmiennej. Proszę zwrócić uwagę na fakt ile „cykli” aplikacji mija pomiędzy akcją modyfikacji zmiennej a przełączeniem pomiędzy wątkami.
3. Proszę skompilować i uruchomić przykładowy program „thread3.c”. W programie zastosowano tzw. semafor nienazwany do synchronizacji komunikacji pomiędzy nadawcą a odbiorcą danych. Komunikacja odbywa się przez zmienną dzieloną. Odbiorca oczekuje na zwolnienie semafora, zanim przeprowadzi odczyt ze zmiennej. Proszę zwrócić uwagę na sposób organizacji cyklicznej komunikacji z potwierdzeniem (synchronizacją) operacji modyfikacji zmiennej.
4. Proszę skompilować i uruchomić przykładowy program „thread3a.c”. W programie zastosowano tzw. semafor nienazwany do synchronizacji komunikacji pomiędzy nadawcą a odbiorcą danych. Komunikacja odbywa się przez zmienną dzieloną. Jak się okazuje taka konstrukcja programu nie gwarantuje zawsze efektywną detekcję zmian współdzielonej zmiennej.
5. Proszę skompilować i uruchomić przykładowy program „thread4.c”. Należy zwrócić uwagę na technikę zapewnienia wzajemnego wykluczania do operacji zapisu/odczytu na współdzielonej zmiennej, które odbywają się cyklicznie. Wadą zastosowanego rozwiązania jest cykliczne „odpytywanie” stanu zmiennej.
6. Proszę skompilować i uruchomić przykładowy program „thread5.c”. Program ilustruje metodę modyfikacji atrybutów wątków. W tym wypadku proces macierzysty nie oczekuje na zakończenie wątku potomnego. Atrybuty wątku potomnego powodują, że nie będzie się on starał po zakończeniu obliczeń oddawać sterowania do procesu rodzicielskiego.
7. Proszę skompilować i uruchomić przykładowy program „thread8.c”. Program ilustruje metodę „masowego” powoływania wątków. Należy zweryfikować działanie programu, kiedy z kodu źródłowego zostaną wyeliminowane instrukcje sleep. Proszę skompilować i uruchomić

przykładowy program „thread8a.c” i przeanalizować, dlaczego nie występują w nim błędy występujące podczas uruchamiania zmodyfikowanego (bez sleep) programu „thread8.c”.

8. Proszę zaproponować aplikację składającą się z 4 wątków. 3 wątki cyklicznie „produkują” liczby i zapisują je do jednej współdzielonej zmiennej. Czwarty wątek sprawdza cyklicznie stan zmiennej i wypisuje ją na ekranie. Dostęp do współdzielonej zmiennej proszę zrealizować z zastosowaniem wzajemnego wykluczania. Proszę przetestować działanie programu, gdy wątek odbierający będzie szybki (co 1 sek następuje odbiór) a wątki nadające – wolne (co kilka sekund). Proszę przeanalizować działanie aplikacji w sytuacji odwrotnej, gdy wątek czytający jest wolniejszy od nadawców. Proszę zwrócić uwagę, że w tak zorganizowanej aplikacji nie ma żadnej gwarancji na „przechwycenie” wszystkich modyfikacji zmiennych.
9. Proszę zaproponować aplikację złożoną z 3 wątków. 2 z nich wysyłają dane do trzeciego przez kolejkę. Trzeci odbiera i identyfikuje (rozpoznaje nadawcę) i wyświetla odebrane komunikaty. Aplikacja odbierająca powinna odbierać wszystkie komunikaty znajdujące się w kolejce (za każdym razem powinno się odbyć „czyszczenie” kolejki. W aplikacji nie trzeba stosować mechanizmów synchronizacji i ochrony dostępu do współdzielonego zasobu, ponieważ sa one zastosowane na poziomie systemu operacyjnego.