Tomasz Ligęza

Programowanie równoległe. Przetwarzanie równoległe i rozproszone.

Sprawozdanie z laboratorium 8.

Cel zajęć:

- doskonalenie umiejętności realizacji synchronizacji w języku C za pomocą zmiennych warunku oraz w programach obiektowych w Javie za pomocą narzędzi pakietu java.util.concurrency.

W ramach zajęć zrealizowałem następujące kroki:

1. Utworzyłem katalog roboczy lab_8,
2. Przetestowałem działanie programu bariera, bez żadnych zmian w kodzie.
3. Wprowadziłem zmiany w kodzie pliku bariera.c tak, by program działał poprawnie:
    a. dodałem zmienne globalne typu int: liczba_watkow_aktualna, liczba_watkow_docelowa,
    b. dodałem zmienne globalne muteksu oraz warunku,
    c. aby zapewnić poprawność przy wielokrotnym uruchamianiu funkcji bariera należy zerować zmienną liczba_watkow_aktualna gdy jest ona równa liczba_watkow_docelowa,

```
int liczba_watkow_aktualna;
int liczba_watkow_docelowa;
pthread_mutex_t muteks = PTHREAD_COND_INITIALIZER;
pthread_cond_t warunek;

void bariera_init(int liczba_watkow) {
   liczba_watkow_aktualna = 0;
   liczba_watkow_docelowa = liczba_watkow;
   pthread_mutex_init(&muteks, NULL);
   pthread_cond_init(&warunek, NULL);
}

void bariera() {
   pthread_mutex_lock(&muteks);
   liczba_watkow_aktualna++;
   if(liczba_watkow_aktualna != liczba_watkow_docelowa) {
       pthread_cond_wait(&warunek, &muteks);
   }
   else {
       liczba_watkow_aktualna = 0;
       pthread_cond_broadcast(&warunek);
   }
   pthread_mutex_unlock(&muteks);
}
```

4. Pobrałem paczkę CzytPis_Pthreads.tgz, rozpakowałem ją oraz uruchomiłem program, aby zidentyfikować problem.
5. Uzupełniłem pola struktury czytelnia_t:

```
/*** Definicje typow zmiennych ***/
typedef struct {
   int liczba_czyt;
   int liczba_pis;

   int liczba_czekajacych_pisarzy;
   int liczba_czekajacych_czytelnikow;

   pthread_cond_t pisarze;
   pthread_cond_t czytelnicy;
   pthread_mutex_t muteks;
 // <- zasoby czytelni
} czytelnia_t;
```

6. Dodałem sprawdzanie warunków poprawnych wartości aktualnych liczb pisarzy i czytelników w procedurach pisania i czytania:

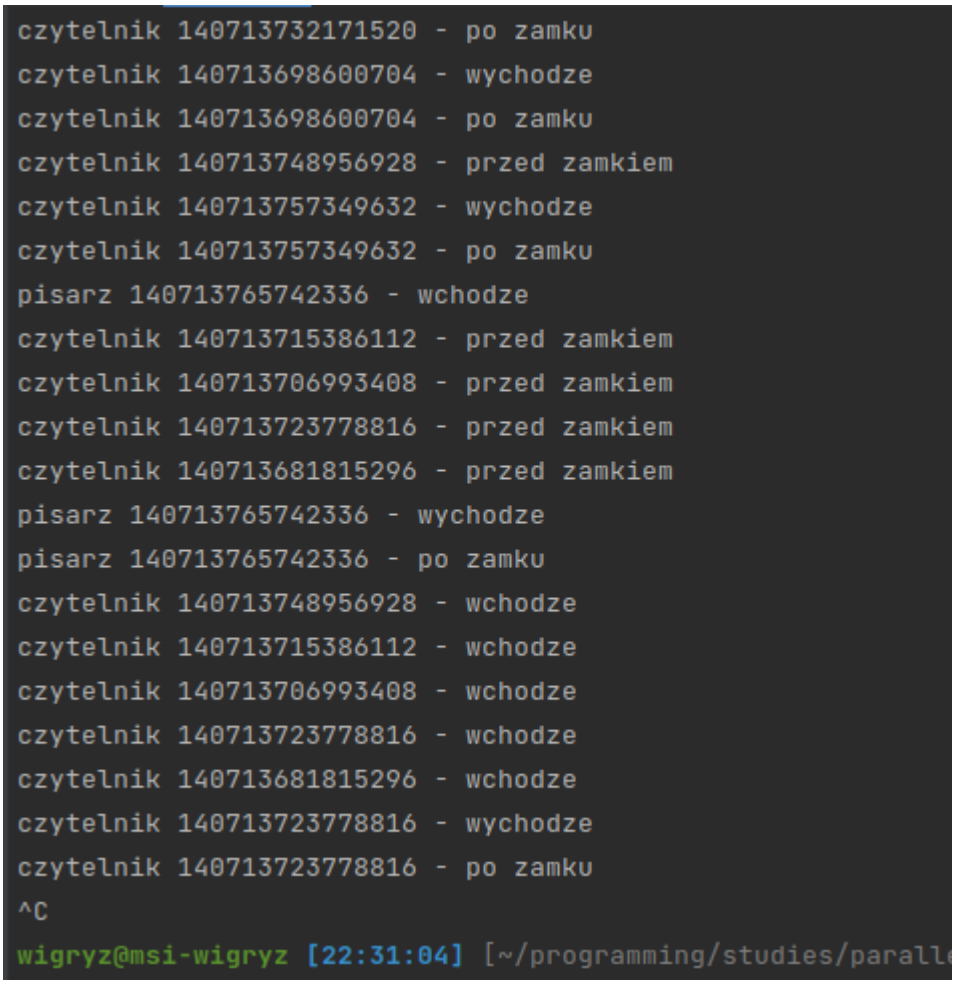
```
void czytam(czytelnia_t *czytelnia_p) {
   usleep(rand() % 300000);
   if(czytelnia_p->liczba_pis > 1 ||
     (czytelnia_p->liczba_pis == 1 && czytelnia_p->liczba_czyt > 0)) {
       printf("Blad! piszacych: %d, czytajacych: %d.",
              czytelnia_p->liczba_pis, czytelnia_p->liczba_czyt);
       exit(1);
   }
}

//w metodzie warunek wygląda identycznie
```

7. Na podstawie pseudokodu monitora Czytelnia poprawiłem kod tak by rozwiązać problem czytelników i pisarzy z użyciem zmiennych warunku:

```
int my_read_lock_lock(czytelnia_t *czytelnia_p) {
   pthread_mutex_lock(&czytelnia_p->muteks);
   if(czytelnia_p->liczba_pis > 0 ||
      czytelnia_p->liczba_czekajacych_pisarzy > 0) {
       czytelnia_p->liczba_czekajacych_czytelnikow++;
       pthread_cond_wait(&czytelnia_p->czytelnicy, &czytelnia_p->muteks);
       czytelnia_p->liczba_czekajacych_czytelnikow--;
   }
   czytelnia_p->liczba_czyt++;
   pthread_mutex_unlock(&czytelnia_p->muteks);
   pthread_cond_signal(&czytelnia_p->czytelnicy);
}


int my_read_lock_unlock(czytelnia_t *czytelnia_p) {
   pthread_mutex_lock(&czytelnia_p->muteks);
   czytelnia_p->liczba_czyt--;
   if(czytelnia_p->liczba_czyt == 0)
       pthread_cond_signal(&czytelnia_p->pisarze);
   pthread_mutex_unlock(&czytelnia_p->muteks);
}


int my_write_lock_lock(czytelnia_t *czytelnia_p) {
   pthread_mutex_lock(&czytelnia_p->muteks);
   if(czytelnia_p->liczba_pis + czytelnia_p->liczba_czyt > 0) {
       czytelnia_p->liczba_czekajacych_pisarzy++;
       pthread_cond_wait(&czytelnia_p->pisarze, &czytelnia_p->muteks);
       czytelnia_p->liczba_czekajacych_pisarzy--;
   }
   czytelnia_p->liczba_pis++;
   pthread_mutex_unlock(&czytelnia_p->muteks);
}


int my_write_lock_unlock(czytelnia_t *czytelnia_p) {
   pthread_mutex_lock(&czytelnia_p->muteks);
   czytelnia_p->liczba_pis--;
   if(czytelnia_p->liczba_czekajacych_czytelnikow > 0)
       pthread_cond_signal(&czytelnia_p->czytelnicy);
   else
       pthread_cond_signal(&czytelnia_p->pisarze);
   pthread_mutex_unlock(&czytelnia_p->muteks);
}
```

8. Uruchomiłem program i po kilku minutach poprawnego działania uznałem, że zrealizowałem zadanie prawidłowo.

Zrzut ekranu z wykonania programu:



Wnioski:

Pierwszym problemem był problem bariery - wątek może zakończyć pewną czynność dopiero, gdy reszta wątków zacznie wykonywać tę czynność. Do rozwiązania tego problemu wykorzystaliśmy zmienną warunku. W momencie, gdy wątek wchodził do funkcji i nie był on “ostatnim” wątkiem, to przechodził w stan oczekiwania. Gdy jednak do funkcji wchodził wątek ostatni, to wywoływał on funkcję `pthread_cond_broadcast` ze zmienną warunku jako argumentem. W ten sposób informował on pozostałe oczekujące wątki o możliwości kontynuacji pracy.

Problem pisarzy i czytelników to często występujący schemat - grupa procesów modyfikuje zasób, podczas gdy druga grupa jedynie go odczytuje. W celu rozwiązania tego problemu wykorzystuje się zmienne warunku. Dzięki nim możemy w łatwy sposób (wcale nie taki łatwy ;)) zarządzać współdzielonym zasobem.