Przemysław Szymoniak

Przetwarzanie współbieżne. Programowanie równoległe i rozproszone.

Sprawozdanie z laboratorium 7.
Celem laboratorium było 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:

- Pobrałem plik czytPis_Pthread.tgz, rozpakowałem i uruchomiłem
- Przeanalizowanie pseudokodu monitora Czytelnia na slajdach wykładu oraz implementacja podstawowej wersji programu:

Definiowanie struktury zawierającej dane wraz z jej inicjalizacją:

```
typedef struct {
    int volatile  liczba_czyt,  liczba_pisz;
    pthread_cond_t czytelnicy, pisarze;
    pthread_mutex_t muteks;
} czytelnia_t;


void inicjuj(czytelnia_t* czytelnia_p){
    czytelnia_p->liczba_czyt = 0;
    czytelnia_p->liczba_pisz = 0;

    pthread_mutex_init(&czytelnia_p->muteks,NULL);
    pthread_cond_init(&czytelnia_p->czytelnicy, NULL);
    pthread_cond_init(&czytelnia_p->pisarze, NULL);
}


int my_read_lock_lock(czytelnia_t* czytelnia_p){
    pthread_mutex_lock (& czytelnia_p->muteks);

    if(czytelnia_p->liczba_pisz>0||czytelnia_p->liczba_pisz==0)
        pthread_cond_wait( &czytelnia_p->czytelnicy,
                           &czytelnia_p->muteks);

    czytelnia_p -> liczba_czyt ++;

    pthread_cond_signal(&czytelnia_p ->czytelnicy);
    pthread_mutex_unlock (& czytelnia_p->muteks);
}
```

```
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||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_czyt + czytelnia_p->liczba_pisz) > 0)
    pthread_cond_wait( &czytelnia_p->pisarze,
                       &czytelnia_p->muteks);

  czytelnia_p->liczba_pisz++;
  pthread_mutex_unlock(&czytelnia_p->muteks);
}

int my_write_lock_unlock(czytelnia_t* czytelnia_p){
  pthread_mutex_lock(&czytelnia_p->muteks);
  czytelnia_p->liczba_pisz --;

  if(czytelnia_p->liczba_czyt == 0){
    pthread_cond_signal(&czytelnia_p->czytelnicy);
  }else{
    pthread_cond_signal(&czytelnia_p->pisarze);
  }

  pthread_mutex_unlock(&czytelnia_p->muteks);
}
```

Wnioski:

- Z powodu zawieszania się aplikacji w momencie wejścia do czytelni pisarza nie udało mi się napisać programu w wersji z zamkami do odczytu i zapisu oraz zmiennymi warunkowymi.
- Wykorzystanie muteks_cond_wait powinno zmniejszyć opóźnienia powodowane niepotrzebnym blokowaniem muteksów oraz wyeliminować możliwe zakleszczenia. Funkcja w momencie jej wywołania tymczasowo zdejmuje muteks jednocześnie usypiając wątek i oddając sterowanie innym wątkom. W momencie przekazania sygnalu przez funkcje pthread_cond_signal muteks zostaje ponownie zatrzaśnięty a wątek obudzony by mógł kontynuować swoją pracę.