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.