Antoni Pawlak

401480

Laboratorium 8: Zmienne warunku

Cel ćwiczenia

- Rozwijanie umiejętności synchronizacji przy pomocy zmiennych warunku
- Rozwiązywanie typowych problemów równoległych
- Zarządzanie asynchronicznym dostępem do zasobu

Wykonanie

Przygotowania

1. Utworzenie katalogu i pobranie plików ze strony

Bariera

2. Utworzenie tablicy struktur przechowującej dane barier

```
bariera_t* bariery[LICZBA_B];

// każda bariera ma swoje dane
// ilość struktur zależna od ilości barier
```

3. Związanie zmiennych warunku i muteksa ze strukturą

```
typedef struct

    {
    int num_threads; // ilość wątków
    int queue_threads; // ilość wątków które "stoją" na barierze
    pthread_cond_t monitor; //zmienna warunku
    pthread_mutex_t muteks; //muteks
    } bariera_t;

bariera_t* bariera_init(int liczba_watkow) {
    bariera_t* bariera = malloc(sizeof(bariera_t));
    bariera->num_threads = liczba_watkow;
    bariera->queue_threads = 0;
    //inicjalizacja muteksa
    pthread_mutex_init(&bariera->muteks, NULL);
    //inicjalizacja zmiennej warunku
    pthread_cond_init(&bariera->monitor, NULL);
    return bariera;
    }
```

4. Zaprojketowanie mechanizmu funkcjonowania bariery

```
void bariera(bariera_t* bariera) {
//zgodnie ze specyfikacją zamykamy muteks
    pthread_mutex_lock(&bariera→muteks);
    //dodajemy wątek do kolejki
    bariera→queue_threads++;
    //oczekujemy jeżeli wszystkie jeszcze nie dotarły
    if(bariera->queue_threads != bariera->num_threads) {
    pthread_cond_wait(&bariera->monitor, &bariera->muteks);
    }
    // odblokowujemy muteks
    pthread_mutex_unlock(&bariera→muteks);
    // dajemy sygnał wszystkim wątkom oczekującym
    pthread_cond_broadcast(&bariera->monitor);
```

Czytelnicy i Pisarze

5. Utworzenie zasobów czytelni

```
typedef struct {

    int ilosc_pisarzy;// licznik pisarzy
    int ilosc_czytelnikow;// licznik czytelników
    int oczekujacy_pisarze;// kolejka pisarzy
    int oczekujacy_czytelnicy;// kolejka czytelników

    pthread_cond_t monitor_czytelnicy;// zmienna warunku czyt.
    pthread_cond_t monitor_pisarze;// zmienna warunku pisarzy
    pthread_mutex_t muteks;// muteks
    } czytelnia_t;

void inicjuj(czytelnia_t* czytelnia_p){
    czytelnia_p->ilosc_czytelnikow = 0;
    czytelnia_p->ilosc_pisarzy = 0;
    czytelnia_p->oczekujacy_czytelnicy = 0;
    czytelnia_p->oczekujacy_pisarze = 0;
    // inicjalizacja zmiennej warunku czytelników
    pthread_cond_init(&(czytelnia_p->monitor_czytelnicy), NULL);
    // inicjalizacja zmiennej warunku pisarzy
    pthread_cond_init(&(czytelnia_p->monitor_pisarze), NULL);
    // inicjalizacja muteksa
    pthread_mutex_init(&(czytelnia_p->muteks), NULL);}
```

6. Chcę pisać

```
int my_write_lock_lock(czytelnia_t* czytelnia_p){
    // blokujemy muteks zgodnie ze specyfikacją
    pthread_mutex_lock(&(czytelnia_p->muteks));
    czytelnia_p->oczekujacy_pisarze++;
    // dodajemy do kolejki i sprawdzamy stan czytelni
    if(czytelnia_p->ilosc_czytelnikow + czytelnia_p->ilosc_pisarzy > 0)
    {
    // jeżeli nie można pisać to czekaj
    pthread_cond_wait(&(czytelnia_p->monitor_pisarze), &(czytelnia_p-
    >muteks));
    }
    // przenosimy pisarza z kolejki do czytelni
    czytelnia_p->oczekujacy_pisarze--;
    czytelnia_p->ilosc_pisarzy++;
    }
```

7. Kończę pisać

```
int my_write_lock_unlock(czytelnia_t* czytelnia_p){
    czytelnia_p->ilosc_pisarzy--;
    // pisarz wychodzi z czytelni, jeżeli są czytelnicy to..
    if(czytelnia_p->oczekujacy_czytelnicy > 0) {
    // daje sygnał czytelnikom
    pthread_cond_signal(&(czytelnia_p->monitor_czytelnicy));
    } else {
    // w przeciwnym wypadku daje sygnał pisarzom
    pthread_cond_signal(&(czytelnia_p->monitor_pisarze));
    }
    // odblokowanie muteksa
    pthread_mutex_unlock(&(czytelnia_p->muteks));
    }
```

8. Chcę czytać

```
int my_read_lock_lock(czytelnia_t* czytelnia_p){
    // blokujemy muteks zgodnie ze specyfikacją
    pthread_mutex_lock(&(czytelnia_p->muteks));
    czytelnia_p->oczekujacy_czytelnicy++;
    // dodajemy czytelnika do kolejki i sprawdzamy stan czytelni
    if(czytelnia_p->ilosc_pisarzy > 0 || czytelnia_p->oczekujacy_pisarze
    > 0) {
    // jeżeli nie można czytać czekaj
    pthread_cond_wait(&(czytelnia_p-
    >monitor_czytelnicy),&(czytelnia_p->muteks));
    }
    //przenosimy czytelnika z kolejki do czytelni
    czytelnia_p->oczekujacy_czytelnicy--;
    czytelnia_p->ilosc_czytelnikow++;
    // odblokowujemy muteks
    pthread_mutex_unlock(&(czytelnia_p->muteks));
    // dajemy sygnał czytelnikom
    pthread_cond_signal(&(czytelnia_p->monitor_czytelnicy));
    }
```

9. Kończę czytać

```
int my_read_lock_unlock(czytelnia_t* czytelnia_p){
czytelnia_p->ilosc_czytelnikow--;
// jeżeli są oczekujący pisare to ich wpuść
if(czytelnia_p->oczekujacy_pisarze > 0)
//  pthread_cond_signal(&(czytelnia_p->monitor_pisarze));}
```

10. Sprawdzanie błędów

    1. Czytelnik

```
//podczas czytania nie może być pisarzy
    if(czytelnia_p->ilosc_pisarzy > 0)
//liczba pisarzy nie powinna być ujemna
    if(czytelnia_p->ilosc_pisarzy < 0)
// kolejki nie powinny być ujemne
    if(czytelnia_p->oczekujacy_pisarze < 0)
    if(czytelnia_p->oczekujacy_czytelnicy < 0)
```

    2. Pisarz

```
// podczas pisania nie może być czytelników
    if(czytelnia_p->ilosc_czytelnikow > 0)
//może być tylko 1 pisarz
    if(czytelnia_p->ilosc_pisarzy > 1)
//kolejki nie powinny być ujemne
    if(czytelnia_p->oczekujacy_pisarze < 0)
    if(czytelnia_p->oczekujacy_czytelnicy < 0)
```

Wnioski

Rozwiązywanie problemu bariery

- Do rozwiązania problemu zastosowaliśmy
    - strukturę danych
    - zmienną warunku
    - muteks
- Poprzez odpowiednie zarządzanie stanem programu, utworzeniem odpowiednich warunków i zastosowanie mechanizmu zmiennych warunku mogliśmy sprawić, że wątki będą czekać na pozostałe.
- Zmienna warunku pozwala utworzyć punkt komunikacji/synchronizacji między wątkami
- Potencjalne zastosowania bariery:
    - Asynchroniczne obliczenia iteracyjne
    - Odczyt z bazy danych
    - Synchronizacja międzywątkowa
    - Redukcja pętli i tablic
- Rozwiązanie problemu bariery pozwoliło na praktycznym przykładzie obserwować działanie tego mechanizmu, dzięki temu będzie można optymalniej używać gotowych rozwiązań takich jak OMP czy MPI

Rozwiązywanie problemy czytelnicy-pisarze

- Do rozwiązania problemu zastosowaliśmy
  - strukturę danych
  - dwie zmienne warunku
  - muteks
- Dzięki zastosowaniu globalnej struktury danych mogliśmy sterować dostępem do zasobu dla kilku oddzielnych aktorów. Poprzez odpowiednie użycie zmiennych warunku mogliśmy sygnalizować wątkom działania na które mogą sobie pozwolić
- Należało unikać zagłodzenia
  - Czytelnik sprawdzał czy nie czeka jakiś pisarz
  - Pisarz sprawdzał czy nie czeka jakiś czytelnik
- Dostęp do zasobu odbywał się poprzez mechanizm zamków, który został zaimplementoway przy użyciu zmiennych warunku.
- Sprawdzanie błedów pozwoliło zweryfikować implementację
- Potencjalne zastosowania
  - Implementacja silnika bazy danych
  - Programy o pamięci współdzielonej
- Rozwiazanie problemu czytelników i pisarzy pozwoliło na praktyczne analizowanie i zrozumienie problemu. Dzięki takiemu podejściu programista wydajniej będzie używał takich elementów jak zamki ze standardu POSIX.