Łukasz Oprych Gr.5 lab	Programowanie Równoległe T: Monitory i zmienne warunku	02.12.2023.r.

Cel ćwiczenia:

Zapoznanie się z synchronizacją przy pomocy zmiennych warunku oraz asynchronicznym dostępem do zasobu.

Przebieg ćwiczenia:

Po pobraniu plików oraz przygotowaniu środowiska zgodnie z poleceniami prowadzącego w katalogu lab_8_bariera zaimplementowano algorytm realizujący funkcję bariery w utworzonym pliku bariera.c. Dodano między innymi zmienne statyczne ilosc_watkow i ilosc_watkow_w_funkcji, które związane są z ilością wątków biorących udział w przechodzeniu przez barierę oraz zmienną odpowiadającą za warunek i muteksy. W funkcji bariera() zastosowano pthread_cond_signal czyli zmienną rozgłaszającą sygnał, w tym przypadku, budzono wątki, po tym jak już ostatni wątek ze wszystkich dotarł do bariery. Wątki, które nie dotarły do bariery jako ostatnie były usypiane za pomocą pthread_cond_wait i muteksu.

bariera.c

```c
static int ilosc_watkow_w_funkcji = 0;
static int ilosc_watkow = 0;
pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t condition = PTHREAD_COND_INITIALIZER;
pthread_mutex_t increment_lock = PTHREAD_MUTEX_INITIALIZER;

void bariera_init(int il_watkow) {
        ilosc_watkow = il_watkow;
}

void bariera (){
        pthread_mutex_lock(&increment_lock);
        ilosc_watkow_w_funkcji++;
        pthread_mutex_unlock(&increment_lock);
        bool ostatni_watek = false;
        if (ilosc_watkow == ilosc_watkow_w_funkcji) {
                ostatni_watek = true;
                ilosc_watkow_w_funkcji = 0;
                for(int i =0; i < ilosc_watkow - 1; i++){
                        pthread_cond_signal(&condition);
                }
        }
        pthread_mutex_lock(&lock);
        if (!ostatni_watek){
        pthread_cond_wait(&condition, &lock);
        }
        pthread_mutex_unlock(&lock);
}
```

main.c

```c
extern void bariera_init(int);
extern void bariera(void);

#define LICZBA_W 4
```

```c
void *cokolwiek( void *arg){

  int i, moj_id;

  moj_id = *( (int *) arg );

  printf("przed bariera 1 - watek %d\n",moj_id);

  bariera();

    printf("przed bariera 2 - watek %d\n",moj_id);

  bariera();

  printf("przed bariera 3 - watek %d\n",moj_id);

  bariera();

  printf("przed bariera 4 - watek %d\n",moj_id);

  bariera();

  printf("po ostatniej barierze - watek %d\n",moj_id);

  pthread_exit( (void *)0);
}
```

Wynik programu:

```
loprych@loprych-VirtualBox:~/Desktop/PR_lab/lab_8/lab_8_bariera$ ./ma
przed bariera 1 - watek 2
przed bariera 1 - watek 0
przed bariera 1 - watek 1
przed bariera 1 - watek 3
przed bariera 2 - watek 2
przed bariera 2 - watek 0
przed bariera 2 - watek 3
przed bariera 2 - watek 1
przed bariera 3 - watek 2
przed bariera 3 - watek 1
przed bariera 3 - watek 0
przed bariera 3 - watek 3
przed bariera 4 - watek 2
przed bariera 4 - watek 3
przed bariera 4 - watek 1
przed bariera 4 - watek 0
po ostatniej barierze - watek 2
po ostatniej barierze - watek 0
po ostatniej barierze - watek 3
po ostatniej barierze - watek 1
loprych@loprych-VirtualBox:~/Desktop/PR_lab/lab_8/lab_8_bariera$
```

Jak widać za każdym razem, gdy wszystkie zadane 4 wątki docierały do bariery, bariera została zwalniana, po czym wątki docierały do kolejnej bariery, do czasu aż pokonały wszystkie 4 bariery, które zadano.

Następnie po wypakowaniu kolejnych plików od prowadzącego utworzono katalog lab_8_pthreads. W pliku czytelnia.h zdefiniowano zmienne odpowiadające za czytelników i pisarza.

```
typedef struct {
 int l_c;
 int l_p;
} czytelnia_t;
```

Następnie zaimplementowano śledzenie liczby czytelników i pisarzy w czytelni:

```
int my_read_lock_lock(czytelnia_t* czytelnia_p){
        pthread_mutex_lock (&lock);
        czytelnia_p->l_c++;
        pthread_mutex_unlock (&lock);
}

int my_read_lock_unlock(czytelnia_t* czytelnia_p){
        pthread_mutex_lock (&lock);
        czytelnia_p->l_c--;
        pthread_mutex_unlock (&lock);
}

int my_write_lock_lock(czytelnia_t* czytelnia_p){
        pthread_mutex_lock (&lock);
        czytelnia_p->l_p++;
        pthread_mutex_unlock (&lock);
}

int my_write_lock_unlock(czytelnia_t* czytelnia_p){
        pthread_mutex_lock (&lock);
        czytelnia_p->l_p--;
        pthread_mutex_unlock (&lock);
}
```

Następnie umieszczono w procedurach pisania i czytania sprawdzenia warunków poprawnych wartości aktualnych liczb pisarzy i czytelników, sprawdzamy tu również czy liczba czytelników bądź pisarzy nie jest przypadkiem ujemna. Zgodnie z problemem czytelników i pisarzy, w czytelni w jednym momencie może się znajdować 1 pisarz lub wielu czytelników, obie instrukcje umieszczono wewnątrz kompilacji warunkowej #ifdef MY_DEBUG.

```
void czytam(czytelnia_t* czytelnia_p){
#ifdef MY_DEBUG
        pthread_mutex_lock (&lock);
        printf("liczba czytelnikow = %d, liczba pisarzy = %d\n", czytelnia_p->l_c, czytelnia_p->l_p);
        if(czytelnia_p->l_p>1 || (czytelnia_p->l_p==1 && czytelnia_p->l_c>0) || czytelnia_p->l_p<0 ||
czytelnia_p->l_c<0 ) {
        printf("Nie powinienem czytac\n");
        pthread_mutex_unlock (&lock);
        exit(0);
        }
        pthread_mutex_unlock (&lock);
#endif
    usleep(rand()%300000);
}

void pisze(czytelnia_t* czytelnia_p){
#ifdef MY_DEBUG
        pthread_mutex_lock (&lock);
        printf("liczba czytelnikow = %d, liczba pisarzy = %d\n", czytelnia_p->l_c, czytelnia_p->l_p);
        if( czytelnia_p->l_p>1 || (czytelnia_p->l_p==1 && czytelnia_p->l_c>0) || czytelnia_p->l_p<0 ||
czytelnia_p->l_c<0 ) {
        printf("Nie powinienem pisac\n");
        pthread_mutex_unlock (&lock);
        exit(0);
        }
        pthread_mutex_unlock (&lock);
#endif
    usleep(rand()%300000);
}
```

Fragment Makefile, przekazanie opcji -DMY_DEBUG do kompilatora:

```
czyt_pis: czyt_pis.o czytelnia.o
        $(LD) $(LDFL) -o $(NAZWA) czyt_pis.o czytelnia.o $(LIB)

czyt_pis.o: czyt_pis.c czytelnia.h
        $(CC) $(CFL) -c czyt_pis.c $(INC)

czytelnia.o: czytelnia.c czytelnia.h
        $(CC) $(CFL) -c czytelnia.c $(INC) -DMY_DEBUG
```

Przetestowanie programu:

```
liczba czytelnikow = 3, liczba pisarzy = 0
czytelnik 139757736658496 - wychodze
czytelnik 139757736658496 - po zamku
czytelnik 139757677909568 - przed zamkiem
czytelnik 139757677909568 - wchodze
liczba czytelnikow = 3, liczba pisarzy = 0
czytelnik 139757686302272 - przed zamkiem
czytelnik 139757686302272 - wchodze
liczba czytelnikow = 4, liczba pisarzy = 0
pisarz 139757787014720 - przed zamkiem
pisarz 139757787014720 - wchodze
liczba czytelnikow = 4, liczba pisarzy = 1
Nie powinienem pisac
loprych@loprych-VirtualBox:~/Desktop/PR_lab/lab_8/lab_8_pthreads
lajacaczytelnia$
```

Jak widać doszło do przerwania programu, warunek nie został spełniony, z powodu wystąpienia jednocześnie pisarzy oraz czytelników w jednym momencie w czytelni.

Modyfikacja kodu między innymi na podstawie pseudokodu monitora Czytelnia z wykładu:

Reprezentacja czytelników oraz pisarzy czekających na wejście do czytelni:

```
pthread_cond_t cond_pisarze = P
int l_c_czekajacych = 0;
int l_p_czekajacych = 0;
int my_read_lock_lock(czytelnia
```

Pilnowanie aby wątki między sobą prawidłowo modyfikowały dane w czytelni oraz zmienne globalne:

```
pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t lock_czytelnicy = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t lock_pisarze = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t cond_czytelnicy = PTHREAD_COND_INITIALIZER;
pthread_cond_t cond_pisarze = PTHREAD_COND_INITIALIZER;
```

```
int my_read_lock_lock(czytelnia_t* czytelnia_p){
        pthread_mutex_lock (&lock);
        if(czytelnia_p->l_p > 0 || l_p_czekajacych > 0){
                l_c_czekajacych++;
                pthread_mutex_unlock(&lock);
                pthread_mutex_lock(&lock_czytelnicy);
                pthread_cond_wait(&cond_czytelnicy, &lock_czytelnicy);
                pthread_mutex_unlock(&lock_czytelnicy);
                pthread_mutex_lock(&lock);
                l_c_czekajacych--;
        }
        czytelnia_p->l_c++;
        pthread_cond_signal (&cond_czytelnicy);
        pthread_mutex_unlock (&lock);
}

int my_read_lock_unlock(czytelnia_t* czytelnia_p){
        pthread_mutex_lock (&lock);
        czytelnia_p->l_c--;
        if(czytelnia_p->l_c ==0){
                pthread_cond_signal(&cond_pisarze);
        }
        pthread_mutex_unlock (&lock);
}

int my_write_lock_lock(czytelnia_t* czytelnia_p){
        pthread_mutex_lock (&lock);
        if(czytelnia_p->l_c + czytelnia_p->l_p > 0) {
                l_p_czekajacych++;
                pthread_mutex_unlock (&lock);
                pthread_mutex_lock(&lock_pisarze);
                pthread_cond_wait(&cond_pisarze, &lock_pisarze);
                pthread_mutex_unlock(&lock_pisarze);
                pthread_mutex_lock(&lock);
                l_p_czekajacych--;
        }
        czytelnia_p->l_p++;
        pthread_mutex_unlock (&lock);
}

int my_write_lock_unlock(czytelnia_t* czytelnia_p){
        pthread_mutex_lock (&lock);
        czytelnia_p->l_p--;
        if(l_c_czekajacych > 0){
                pthread_cond_signal(&cond_czytelnicy);
        }
        else {
                pthread_cond_signal(&cond_pisarze);
        }
        pthread_mutex_unlock (&lock);
}
```

Przetestowanie programu po modyfikacji:

```
czytelnik 140719748515392 - wchodze
liczba czytelnikow = 1, liczba pisarzy = 0
czytelnik 140719689766464 - wchodze
liczba czytelnikow = 2, liczba pisarzy = 0
czytelnik 140719714944576 - wchodze
liczba czytelnikow = 3, liczba pisarzy = 0
czytelnik 140719706551872 - przed zamkiem
czytelnik 140719723337280 - przed zamkiem
czytelnik 140719740122688 - przed zamkiem
czytelnik 140719765300800 - przed zamkiem
czytelnik 140719748515392 - wychodze
czytelnik 140719748515392 - po zamku
czytelnik 140719714944576 - wychodze
czytelnik 140719714944576 - po zamku
czytelnik 140719689766464 - wychodze
czytelnik 140719689766464 - po zamku
pisarz 140719807264320 - wchodze
liczba czytelnikow = 0, liczba pisarzy = 1
czytelnik 140719756908096 - przed zamkiem
```

```
czytelnik 140719698159168 - wchodze
liczba czytelnikow = 1, liczba pisarzy = 0
czytelnik 140719706551872 - wchodze
liczba czytelnikow = 2, liczba pisarzy = 0
czytelnik 140719731729984 - wchodze
liczba czytelnikow = 3, liczba pisarzy = 0
czytelnik 140719714944576 - wchodze
liczba czytelnikow = 4, liczba pisarzy = 0
czytelnik 140719714944576 - wychodze
czytelnik 140719714944576 - po zamku
czytelnik 140719706551872 - wychodze
czytelnik 140719706551872 - po zamku
czytelnik 140719731729984 - wychodze
czytelnik 140719731729984 - po zamku
czytelnik 140719756908096 - przed zamkiem
czytelnik 140719756908096 - wchodze
liczba czytelnikow = 2, liczba pisarzy = 0
czytelnik 140719756908096 - wychodze
czytelnik 140719756908096 - po zamku
^C
loprych@loprych-VirtualBox:~/Desktop/PR_lab/l
```

Program działa w nieskończonej pętli, nie dochodzi do konfliktów w dostępie do czytelni.

Wnioski:

Aby prawidłowo wykonać oba programy użyliśmy zmiennych warunku (pthread_cond_wait, pthread cond_signal) oraz muteksy (pthread_mutex). Dzięki użyciu polecenia wait mogliśmy, zapewnić, że wątki zostaną zawieszane i będą oczekiwać na wykonanie programu przez pozostałe wątki, czy też używając signal takowe wątki wybudzać. Zmienna warunku może być dobrym zastosowaniem w kwestii ustalenia miejsca synchronizacji współpracujących wątków.