Politechnika Białostocka

Wydział Informatyki Politechniki Białostockiej

Katedra Oprogramowania -

Pracownia Specjalistyczna Systemów Operacyjnych

Dokumentacja techniczna projektu z przedmiotu – INF1SOP

Systemy Operacyjne

Dokumentacja techniczna projektu -

Czytelnicy i pisarze

Projekt 2, nr zadania projektowego - 2

Grupa : PS 2 **Skład grupy:** 1. **Artur Leszczak** 2. **Dawid Ugniewski**	**Data złożenia projektu:** **16.06.2023 r.** **Prowadzący:** **mgr inż. Daniel Reska**
Studia: dzienne **Semestr: IV** **Kierunek studiów:** **Informatyka**	**Ocena:**

Spis Treści

1. Założenia oraz wymagania projektowe:

Z czytelni korzysta na okrągło pewna ilość czytelników i pisarzy (dwa typy wątków), przy czym jednocześnie może w niej znajdować się albo dowolna ilość czytelników, albo jeden pisarz, albo nikt - nigdy inaczej. Problem ten ma trzy rozwiązania - z możliwością zagłodzenia pisarzy, z możliwością zagłodzenia czytelników oraz wykluczające zagłodzenie.

Napisać:

- dwa programy symulujące dwa różne rozwiązania tego problemu, bez korzystania ze zmiennych warunkowych **[17 p]**, albo (!)
- dwa programy symulujące dwa różne rozwiązania tego problemu, przy czym jeden z nich musi korzystać ze zmiennych warunkowych (condition variable). **[27 p],** albo
- trzy programy symulujące trzy różne rozwiązania tego problemu, przy czym przynajmniej jeden z nich musi korzystać ze zmiennych warunkowych **[34 p]**.

Ilość wątków pisarzy P i czytelników C można przekazać jako argumenty linii poleceń. Zarówno czytelnicy jak i pisarze wkrótce po opuszczeniu czytelni, po losowym czasie (maks. kilka sekund - nie może być stały z każdym wywołaniem), próbują znów się do niej dostać (wątki działają dalej). Użytkownicy czytelni również przebywają w środku przez losowy czas. Program powinien wypisywać komunikaty według poniższego przykładu:

Kczytelnikow: 11 Kpisarzy: 10 [C:0 P:1]

Oznacza to, że w kolejce przed czytelnią czeka 10 pisarzy i 11 czytelników a sama czytelnia zajęta jest przez jednego pisarza. Po uruchomieniu programu z parametrem -info należy wypisywać całą zawartość kolejek czytelników i pisarzy, a także listę osób przebywających w czytelni. Komunikat należy wypisywać w momencie zmiany którejkolwiek z tych wartości.

2. Implementacja

a) Rozwiązanie wykluczające zagłodzenie

Rozwiązanie używa kolejki FIFO implementując mechanizmy ticket lock i spin lock.

W rozwiązaniu do synchronizacji użyto następujących zmiennych warunkowych i mutexów:

```
/*
*  Zmienne symbolizujace pobyt w czytelni
*/

pthread_mutex_t mutex_dane = PTHREAD_MUTEX_INITIALIZER;
volatile int ilosc_pisarzy_w_czytelni = 0;
volatile int ilosc_czytelnikow_w_czytelni = 0;

/**********/

pthread_mutex_t mutex_nextTicket = PTHREAD_MUTEX_INITIALIZER;
volatile int nextTicket = 0;

/**********/
```

```
pthread_mutex_t mutex_servingTicket = PTHREAD_MUTEX_INITIALIZER;
volatile int servingTicket = 1;
pthread_cond_t czekanieWKolejce = PTHREAD_COND_INITIALIZER;

/*
*  Zmienna symbolizujaca ile czytelnikow z rzedu wzielo bilet
*/

pthread_mutex_t mutex_wstrzymanieCzytelnikow =
PTHREAD_MUTEX_INITIALIZER;
volatile int ticketyCzytelnikow = 0;
pthread_cond_t wstrzymanieCzytelnikow = PTHREAD_COND_INITIALIZER;

/*
*  Zmienne warunkowe sygnalizujace spelnienie warunkow pobytu w czytelni
*/

pthread_cond_t czekanieAzWyjdziePisarz = PTHREAD_COND_INITIALIZER;
pthread_cond_t czekanieAzWyjdaWszyscy = PTHREAD_COND_INITIALIZER;
```

Mutexy mają za zadanie chronić zmienne przed równoczesnym dostępem do nich kilku wątków.

Na zmiennej warunkowej czekanieWKolejce czekają wątki o bilecie mniejszym niż obsługiwany.

Na zmiennej warunkowej wstrzymanieCzytelnikow czekają wątki czytelników, które już odebrały bilet, a dawno biletu nie odebrał pisarz. Ten spinlock przepuszcza 10 czytelników do wzięcia biletu i blokuje się aż do odebrania biletu przez pisarza. Kolejne porcje czekających na bilet czytelników mogą być większe w zależności jak pisarze biorą bilety. Zapewnia to, że mimo wpuszczania wielu czytelników na raz do czytelni pisarze mogą brać bilety dołączając do kolejki FIFO.

Na zmiennych warunkowych czekanieAzWyjdziePisarz i czekanieAzWyjdaWszyscy czekają odpowiednio wątki czytelników i wątki pisarzy, których bilet kolejka FIFO już przepuściła i teraz czekają na spełnienie warunków czytelni (opuszczenie poprzednich wątków bez wywłaszczania ich wcześniej).

Wątki pracują na funkcjach : void *Funkcja_czytelnika() i void *Funkcja_pisarza() oraz wypisywarkach void Wypisz_komunikat() i void Wypisz_stan wypisujących stan czytelni i kolejki.

Funkcje czytelnika i pisarza są podobne.

Fukncja czytelnika:

```
int ticket = -1;

int numer = *(int *) arg;

pthread_mutex_lock(&mutex_dane);
kolejka_czytelnikow[numer][0] = numer;
kolejka_czytelnikow[numer][1] = 0;
pthread_mutex_unlock(&mutex_dane);


      while(1)
```

```
      {
            //sprawdzenie czy za duzo czytelnikow z rzedu nie pobralo

            pthread_mutex_lock( &mutex_wstrzymanieCzytelnikow );
            ++ticketyCzytelnikow; //zaktualizuj liczbe czytelnikow z
rzedu
            while ( ticketyCzytelnikow >= 10)
            {
                  pthread_cond_wait( &wstrzymanieCzytelnikow,
&mutex_wstrzymanieCzytelnikow );
            }
            pthread_mutex_unlock( &mutex_wstrzymanieCzytelnikow );

            // pobieranie biletu
            pthread_mutex_lock(&mutex_nextTicket);
            ticket = ++nextTicket;
            pthread_mutex_unlock(&mutex_nextTicket);

            // weryfikacja biletu

            pthread_mutex_lock(&mutex_servingTicket);
            while( ticket != servingTicket)
            {
                  pthread_cond_wait(&czekanieWKolejce,
&mutex_servingTicket);
            }
            //tu wpuszcza po kolei (FIFO)
            ++servingTicket;
            pthread_mutex_unlock(&mutex_servingTicket);


            // wpuszczono watek do drzwi
            // sprawdz czy nie ma pisarza, jesli jest => czekaj

            pthread_mutex_lock(&mutex_dane);

            while( ilosc_pisarzy_w_czytelni > 0 )
            {
                  pthread_cond_wait( &czekanieAzWyjdziePisarz,
&mutex_dane );
            }

            // jesli nie ma, wpusc czytelnika
            // i wolaj nastepnego do bramki

            ++ilosc_czytelnikow_w_czytelni;
           kolejka_czytelnikow[numer][1] = 1;

            if(infoflag == 0)
                  Wypisz_komunikat();
            if(infoflag == 1)
                  Wypisz_stan();

            pthread_mutex_unlock( &mutex_dane);
```

```
            pthread_cond_broadcast(&czekanieWKolejce); //wolaj natepnego
do bramki

            sleep(LosujKilkaSekund()); //czytelnik przebywa w czytelni

            pthread_mutex_lock( &mutex_dane);
            --ilosc_czytelnikow_w_czytelni;
            kolejka_czytelnikow[numer][1] = 0;
            if(infoflag == 0)
                  Wypisz_komunikat();
            if(infoflag == 1)
                  Wypisz_stan();

            pthread_mutex_unlock(&mutex_dane); //czytelnik wyszedl

            //sprawdz, czy piarz moze juz wejsc
            pthread_cond_broadcast( &czekanieAzWyjdaWszyscy );

            sleep(LosujKilkaSekund());
      }
```

Funkcja wątku czytelnika zaczyna od spin locka który blokując wątki zależnie od zmiennej ticketyCzytelników zapewnia co jakiś czas gwarancję wzięcia blietu pisarzowi, ten z kolei wysyła sygnał broadcast czytelnikom czekającym na zmiennej warunkowej wstrzymanieCzytelnikow.

Każdy wątek ma swoją zmienna ticket, do której pobiera następny wolny ticket nextTicket, który jest chroniony mutexem, aby tylko jeden wątek jednocześnie mógł odczytać bilet a następnie powiększyć o 1.

Po pobraniu biletu czytelnik trafia do spin locka, gdzie czeka, aż obsługiwany bilet będzie jego biletem. Jest tu mechanizm FIFO ticket lock. Pierwszy ticket nie trafia do pętli i kontynuuje działanie, następne oczekują na zmiennej servingTicket na sygnał o konieczności porównania biletów z servingTicket.

Po zaakceptowaniu biletu czytelnik trafia do spinlocka, gdzie czeka, aż wejście do czytelni nie złamie warunków czytelni. (Blokuje mutex mutex_dane, jeśli złamałby warunki, to wchodzi do pętli, gdzie czeka na zmiennej warunkowej czekanieAzWyjdziePisarz na sygnał od wychodzącego pisarza, po otrzymaniu którego znowu blokuje uprzednio odblokowany mutex mutex_dane, i jeśli tym razem nie złamie warunków czytelni (w czytelni nie ma pisarzy) to wchodzi do niej, wejście do czytelni to zinkrementowanie zmiennej ilosc_czytelnikow_w_czytelni).

Na czas pracy czytelnika w czytelni mutex_dane jest odblokowywany, by kolejno wysłać sygnał broadcast na zmienną warunkową czekanieWKolejce, żeby zweryfikowali swoje bilety, a następnie po skończonej pracy znowu go zablokować, zdekrementować zmienną ilosc_czytelnikow_w_czytelni i znowu odblokować mutex.

Po zdekrementowaniu i zinkrementowaniu ilości czytelników w czytelni wywoływana jest wypisywarka wypisująca stan czytelni i kolejki na stderr.

Na koniec wysyłany jest sygnał broadcast na zmienną czekanieAzWyjdaWszyscy (który odbiera pisarz czekający na pustą czytelnię) i czytelnik usypia na moment przed kolejną próbą otrzymania biletu.

Funkcja pisarza:

```
int ticket = -1;

int numer = *(int *) arg;

pthread_mutex_lock(&mutex_dane);
kolejka_pisarzy[numer][0] = numer;
kolejka_pisarzy[numer][1] = 0;
pthread_mutex_unlock(&mutex_dane);


      while(1)
      {
            // pobieranie biletu

            pthread_mutex_lock(&mutex_nextTicket);
            ticket = ++nextTicket;
            pthread_mutex_unlock(&mutex_nextTicket);

            //zresetuj liczbe czytelnikow z rzedu

            pthread_mutex_lock( &mutex_wstrzymanieCzytelnikow );
            ticketyCzytelnikow = 0;
            pthread_mutex_unlock( &mutex_wstrzymanieCzytelnikow );

            pthread_cond_broadcast( &wstrzymanieCzytelnikow );

            // weryfikacja biletu

            pthread_mutex_lock(&mutex_servingTicket);
            while( ticket != servingTicket)
            {
                  pthread_cond_wait(&czekanieWKolejce,
&mutex_servingTicket);
            }
            //tu wpuszcza po kolei (FIFO)
            ++servingTicket;
            pthread_mutex_unlock(&mutex_servingTicket);

            // wpuszczono watek do drzwi
            // zaktualizuj numer kolejno obslugiwanego

            // sprawdz czy pusta czytelnia, jesli nie => czekaj

            pthread_mutex_lock( &mutex_dane );

            while( ilosc_czytelnikow_w_czytelni > 0 ||
ilosc_pisarzy_w_czytelni > 0)
            {
                  pthread_cond_wait( &czekanieAzWyjdaWszyscy, &mutex_dane
);
            }
```

```
            //jesli pusto => wpusc pisarza
            // i wolaj nastepnego do bramki

            ++ilosc_pisarzy_w_czytelni;
            if(infoflag == 0)
                  Wypisz_komunikat();
            if(infoflag == 1)
                  Wypisz_dokladny_komunikat(pthread_self(), ticket);

            pthread_mutex_unlock( &mutex_dane );

            pthread_cond_broadcast(&czekanieWKolejce); //wolaj natepnego
do bramki

            sleep(LosujKilkaSekund()); // Pisarz przebywa w czytelni

            pthread_mutex_lock( &mutex_dane );
            --ilosc_pisarzy_w_czytelni;
            if(infoflag == 0)
                  Wypisz_komunikat();
            if(infoflag == 1)
                  Wypisz_dokladny_komunikat(pthread_self(), ticket);
            pthread_mutex_unlock( &mutex_dane ); //Pisarz wyszedl

            //sprawdz, czy ktos moze juz wejsc
            pthread_cond_broadcast( &czekanieAzWyjdaWszyscy );
            pthread_cond_broadcast( &czekanieAzWyjdziePisarz );

            sleep(LosujKilkaSekund());
```

Z racji sporego podobieństwa do Funkcji_czytelnika zostaną opisane tylko różnice.

Na początku pisarz nie czeka w spin locku, za to po wzięciu biletu wysyła sygnał broadcast na zmienną warunkową wstrzymanieCzytelników, że mogą znowu brać bilety.

Po zweryfikowaniu biletu z servingTicket pisarz wchodzi do spin locka o innym warunku, w czytelni nie może być nikogo – czeka na zmiennej warunkowej czekanieAzWyjdaWszyscy.

Po wyjściu z czytelni pisarz wysyłą nie jeden a dwa sygnały broadcast na z mienne warunkowe czekanieAzWyjdaWszyscy i czekanieAzWyjdziePisarz, gdyż zarówno czytelnicy jak i pisarze musieli czekać na wyjście tego pisarza.

b) Rozwiązanie z możliwością zagłodzenia pisarzy

W rozwiązaniu do synchronizacji użyto następujących zmiennych warunkowych i mutexów:

```
pthread_mutex_t mutex_dane = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t czekanieAzWyjdziePisarz = PTHREAD_COND_INITIALIZER;
pthread_cond_t czekanieAzWyjdaWszyscy = PTHREAD_COND_INITIALIZER;
```

Mutex_dane zapewnia watkowi wyłączny dostęp do zmiennych globalnych oraz do zmiennych warunkowych.

Na zmiennych warunkowych czekanieAzWyjdziePisarz i czekanieAzWyjdaWszyscy czekają odpowiednio wątki czytelników i wątki pisarzy, na moment, w którym ich wejście do czytelni nie złamie wymagań czytelni (dowolna ilość czytelników lub jeden pisarz jednocześnie).

Wątki pracują na funkcjach : void *Funkcja_czytelnika() i void *Funkcja_pisarza() oraz wypisywarkach void Wypisz_komunikat() i void Wypisz_stan wypisujących stan czytelni i kolejki.

Funkcje czytelnika i pisarza są podobne.

Fukncja czytelnika:

```
void *Funkcja_czytelnika( void *arg)
{
      int numer = *(int *) arg;

      pthread_mutex_lock(&mutex_dane);
      kolejka_czytelnikow[numer][0] = numer;
      kolejka_czytelnikow[numer][1] = 0;
      pthread_mutex_unlock(&mutex_dane);

      while(1)
      {

            //czekanie na zmiennej czekanieAzWyjdziePisarz

            pthread_mutex_lock( &mutex_dane);
            while( ilosc_pisarzy_w_czytelni > 0)
            {
                  pthread_cond_wait( &czekanieAzWyjdziePisarz,
&mutex_dane);
            }
            ++ilosc_czytelnikow_w_czytelni;

            kolejka_czytelnikow[numer][1] = 1;
            /*Wypisuje komunikat o aktualnym stanie czytelni*/
            if (infoflag == 0)
                  Wypisz_komunikat();
            if (infoflag == 1)
                  Wypisz_stan();

            pthread_mutex_unlock( &mutex_dane);

            sleep(LosujKilkaSekund()*0.5); //czytelnik przebywa w
czytelni

            pthread_mutex_lock( &mutex_dane);
            --ilosc_czytelnikow_w_czytelni;
            kolejka_czytelnikow[numer][1] = 0;

            if(infoflag == 0)
                  Wypisz_komunikat();
            if(infoflag == 1)
                  Wypisz_stan();
            pthread_mutex_unlock(&mutex_dane); //czytelnik wyszedl

            //sprawdz, czy piarz moze juz wejsc
```

```
            pthread_cond_signal( &czekanieAzWyjdaWszyscy );

            sleep(LosujKilkaSekund());

      }
}
```

Czytelnik po przypisaniu sobie odpowiedniego id czeka na sygnal wyjscia pisarza z czytelni na zmiennej czekanieAzWyjdziePisarz – tak samo jak w rozwiązaniu A, następnie tak samo jak w rozwiązaniu A wchodzi do czytelni (blokując mutex), wypisuje informacje, po jakimś czasie (również na zablokowanym mutexie) wychodzi i również wypisuje informacje.

Na koniec wysyła jeden sygnał do czekającego pisarza na zmiennej czekanieAzWyjdaWszyscy, by po pewnym czasie znów próbować wejść do czytelni.

Funkcja pisarza:

```
void *Funkcja_pisarza(void *arg)
{
      int numer = *(int *) arg;

      pthread_mutex_lock(&mutex_dane);
      kolejka_pisarzy[numer][0] = numer;
      kolejka_pisarzy[numer][1] = 0;
      pthread_mutex_unlock(&mutex_dane);

      while(1)
      {

            // sprawdz czy pusta czytelnia, jesli nie =>
            // czekaj na sygnal od wychodzacego pisarza/czytelnika

            pthread_mutex_lock( &mutex_dane );
            while( ilosc_czytelnikow_w_czytelni > 0 ||
ilosc_pisarzy_w_czytelni > 0)
            {
                  pthread_cond_wait( &czekanieAzWyjdaWszyscy, &mutex_dane
);
            }

            //jesli pusto => wpusc pisarza
            // i wolaj nastepnego do bramki
            ++ilosc_pisarzy_w_czytelni;
            kolejka_pisarzy[numer][1] = 1;


            if(infoflag == 0)
                  Wypisz_komunikat();
            if(infoflag == 1)
                  Wypisz_stan();

            pthread_mutex_unlock( &mutex_dane );

            sleep(LosujKilkaSekund()); // Pisarz przebywa w czytelni
```

```
            pthread_mutex_lock( &mutex_dane );


            --ilosc_pisarzy_w_czytelni;
            kolejka_pisarzy[numer][1] = 0;

            if(infoflag == 0)
                  Wypisz_komunikat();
            if(infoflag == 1)
                  Wypisz_stan();
            pthread_mutex_unlock( &mutex_dane ); //Pisarz wyszedl

            //sprawdz, czy ktos moze juz wejsc
            pthread_cond_signal( &czekanieAzWyjdaWszyscy );
            pthread_cond_broadcast( &czekanieAzWyjdziePisarz );

            sleep(LosujKilkaSekund());
      }
}
```

Pisarz również po ustaleniu swojego id czeka, aż jego wejście do czytelni nie złamie warunków czytelni (na zmiennej czekanieAzWyjdaWszyscy), po wejściu robi to samo co czytelnik, z tym że na końcu wysyła jeden sygnał do czekającego pisarza na zmiennej czekanieAzWyjdaWszyscy, a potem dodatkowo sygnał broadcast do czekających na zmiennej czekanieAzWyjdziePisarz czytelników.

Możliwość zagłodzenia pisarzy wynika z faktu, że pisarz musi spełnić dużo trudniejszy warunek przed wejściem do czytelni niż czytelnik, gdzie przy dużej ilości wątków może się to nigdy nie stać.

c) Rozwiązanie z możliwością zagłodzenia czytelników

W rozwiązaniu do synchronizacji użyto następujących zmiennych warunkowych i mutexów:

```
pthread_mutex_t mutex_dane = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t czekanieAzWyjdziePisarz = PTHREAD_COND_INITIALIZER;
pthread_cond_t czekanieAzWyjdaWszyscy = PTHREAD_COND_INITIALIZER;
```

Przeznaczenie tych zmiennych i mutexu jest takie samo jak w rozwiązaniu B – ochrona współdzielonych zasobów i czekanie na spełnienie warunku czytelni.

Wątki pracują na funkcjach : void *Funkcja_czytelnika() i void *Funkcja_pisarza() oraz wypisywarkach void Wypisz_komunikat() i void Wypisz_stan wypisujących stan czytelni i kolejki.

Funkcja czytelnika:

Wykorzystująca mechanizmy pozwalające na “zagłodzenie czytelnikow”.

```
void *Funkcja_czytelnika( void *arg)
{

int numer = *(int *) arg; //otrzymuje identyfikator czytelnika

pthread_mutex_lock(&mutex_dane);

kolejka_czytelnikow[numer][0] = numer; //przydziela otrzymany identyfikator do tablicy zawierajacej
informacje o stanie kolejki czytelnikow
```

```
pthread_mutex_unlock(&mutex_dane);

while(1)
{
kolejka_czytelnikow[numer][1] = -1;


pthread_mutex_lock( &mutex_dane);

//aktualizacja stanu czytelni i kolejki

kolejka_czytelnikow[numer][1] = 0;
```

```
/* czeka na dostępność czytelni*/
while((zlicz_pisarzy_w_kolejce() != 0) || (zlicz_pisarzy_w_czytelni() != 0)){
pthread_cond_wait(&czekanieAzWyjdziePisarz, &mutex_dane);
}

kolejka_czytelnikow[numer][1] = 1;

/*Wypisuje komunikat o aktualnym stanie czytelni*/
if (infoflag == 0)
Wypisz_komunikat();

                        if (infoflag == 1)
Wypisz_stan();

pthread_mutex_unlock( &mutex_dane);

pthread_cond_broadcast(&czekanieWKolejce); //wolaj natepnego do bramki

sleep(LosujKilkaSekund()); //czytelnik przebywa w czytelni

pthread_mutex_lock( &mutex_dane);

/*Wypisuje komunikat o aktualnym stanie czytelni*/
if(infoflag == 0)
Wypisz_komunikat();
if(infoflag == 1)
Wypisz_stan();

pthread_mutex_unlock(&mutex_dane); //czytelnik wyszedl
/*Aktualizuje kolejke i stan czytelni*/

kolejka_czytelnikow[numer][1] = -1;

/*Wypisuje komunikat o aktualnym stanie czytelni i kolejkach*/
if(infoflag == 0)
Wypisz_komunikat();
if(infoflag == 1)
Wypisz_stan();

//sprawdz, czy piarz moze juz wejsc
pthread_cond_broadcast( &czekanieAzWyjdaWszyscy );
```

```
/*odczekuje przed ponowieniem próby wejscia do kolejki*/
sleep(LosujKilkaSekund());

}
}
```

Funkcja Pisarza: Wykorzystująca mechanizmy pozwalające na “zagłodzenie czytelnikow”.

```
void *Funkcja_pisarza(void *arg)
{

int numer = *(int *) arg; //stały identyfikator pisarza

pthread_mutex_lock(&mutex_dane);

kolejka_pisarzy[numer][0] = numer; //przydziela identyfikator do tablicy kolejki pisarzy

pthread_mutex_unlock(&mutex_dane);

while(1)
{
//ustawia stan przebywania w kolejce (-1 - nie dołączył do kolejki)
kolejka_pisarzy[numer][1] = -1;

kolejka_pisarzy[numer][1] = 0;


// sprawdz czy pusta czytelnia, jesli nie => czekaj

pthread_mutex_lock( &mutex_dane );

while ((zlicz_czytelnikow_w_czytelni() > 0) || (zlicz_pisarzy_w_czytelni() > 0){
                        pthread_cond_wait(&czekanieAzWyjdaWszyscy, &mutex_dane);
}

//jesli pusto => wpusc pisarza
//aktualizuj info o kolejce iczytelni

kolejka_pisarzy[numer][1] = 1;


//wyswietl zmiany

if(infoflag == 0)
Wypisz_komunikat();
if(infoflag == 1)
Wypisz_stan();


pthread_mutex_unlock( &mutex_dane );


pthread_cond_broadcast(&czekanieWKolejce); //wolaj natepnego do bramki
```

```
sleep(LosujKilkaSekund()); // Pisarz przebywa w czytelni


pthread_mutex_lock( &mutex_dane );


pthread_mutex_unlock( &mutex_dane ); //Pisarz wyszedl
//aktualizuje stan czytelni i kolejki

kolejka_pisarzy[numer][1] = -1;

//wyswietla info
if(infoflag == 0)
Wypisz_komunikat();
if(infoflag == 1)
Wypisz_stan();

//sprawdz, czy ktos moze juz wejsc
pthread_cond_broadcast( &czekanieAzWyjdaWszyscy );
pthread_cond_broadcast( &czekanieAzWyjdziePisarz );

//usypia na moment

sleep(LosujKilkaSekund());
}
}
```

Możliwości zagłodzenia wynikają z faktu, iż czytelnik nie może wejść do biblioteki jeżeli jakikolwiek pisarz czeka w kolejce, ponadto nie może wejść do czytelni także, jeżeli pisarz w niej jest. Przebywający w czytelni czytelnicy opuszczają ją kolejno, a jeżeli w kolejce przebywa pisarz i czytelni okaże się pusta, pisarz zajmie w niej miejsce i jednocześnie blokując dostęp do czytelni czytelnikom. Po pewnym czasie przebywający w kolejce pisarze zaczynają “zagładzać czytelników”, którzy nie mogą wejść do czytelni.