|  |  |
| --- | --- |
| **SYSTEMY OPERACYJNE – PROJEKT NR 2** Pracownia Specjalistyczna | **DATA:** 24 I 2013 r. |
| Dokumentacja „ projektu”. Temat: Problem czytelników i pisarzy. | **PROWADZACY**: dr inż. Dorota Duda |
| **GRUPA:** 1. Sebastian Burzyński 2. Maciej Januszewski | **OCENA (pkt):** |

**I. CEL PROJEKTU**Problem ten ma trzy typy rozwiązań:

a) rozwiązanie z możliwością zagłodzenia czytelników.

b) rozwiązania z możliwością zagłodzenia pisarzy.

c) rozwiązania nie dopuszczające do zagłodzenia.  
  
Programy powinny wypisywać na standardowe wyjście informacje wskazujące, że procesy są poprawnie

synchronizowane. Wypisywanie komunikatów należy przeprowadzać w odrębnej sekcji krytycznej aby uniknąć

wyścigów.

**II. OPIS DZIAŁANIA  
a) Opis zmiennych wykorzystanych przy realizacji programu:**//Z góry ustalona przez nas ilość pisarzy  
#define WRITERS  
#define WRITERS\_SPOTS  
  
//Z góry ustalona przez nas ilość czytelników  
#define READERS

#define READERS\_SPOTS//Deklarujemy globalnie semafory, żeby wątki miały do nich dostęp

sem\_t mutex;

sem\_t acces;  
  
//Zliczanie ilości czytelników

int readers\_count;

//Ilość czytelników oczekujących na wejście do czytelni

int readers\_waiting\_count;  
  
//Ilość pisarzy oczekujących na wejście do czytelni oraz pisarzy aktualnie piszących

int waiting\_writers = 0;

int working\_writers = 0;  
  
//Ilość czytelników oczekujących na wejście do czytelni oraz czytelników aktualnie czytających  
int waiting\_readers = 0;

int working\_readers = 0;

//Semafory zliczające

sem\_t writers;

sem\_t readers;  
  
//Semafory binarne dające dostęp i odbierające dostep. Przyjmują wartosc 1 albo 0

sem\_t acces\_sem;

sem\_t writers\_block;  
  
//Deklaracje tablic z wątkami pisarzy i czytelników  
pthread\_t WritersThreads[WRITERS];

pthread\_t ReadersThreads[READERS];

**b) sposoby rozwiązania problemu:  
Faworyzacja czytelników**

Czytelnicy nie mają obowiązku czekania na otrzymanie dostępu do zasobu, jeśli w danym momencie nie otrzymał go pisarz. Pisarz może otrzymać tylko dostęp wyłączny, musi czekać na opuszczenie zasobu przez wszystkie inne procesy. Jeżeli czytelnicy przybywają odpowiednio szybko, może dojść do zagłodzenia pisarza: w tej sytuacji będzie on w nieskończoność czekał na zwolnienie zasobu przez wciąż napływających nowych czytelników.

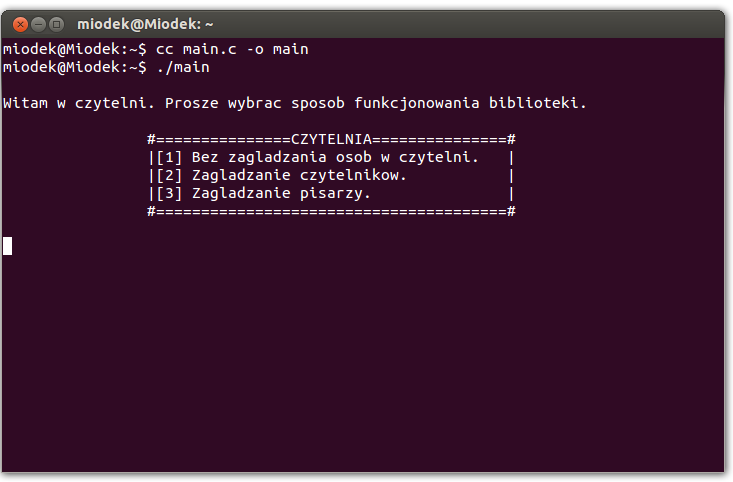
**Faworyzacja pisarzy**

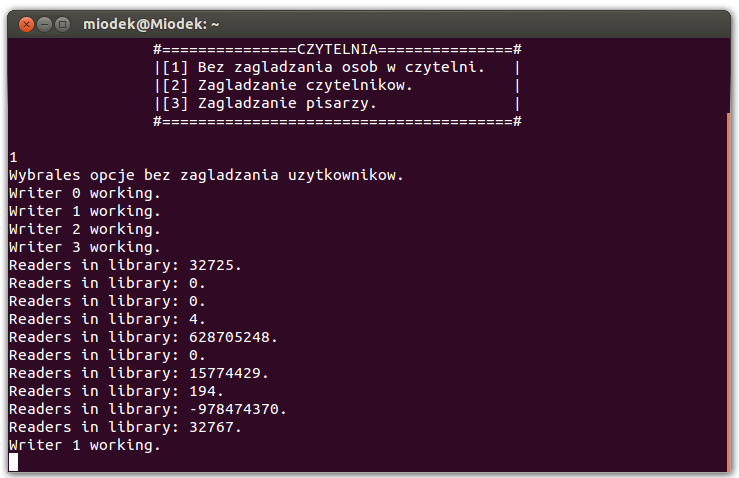
Czytelnicy nie mogą otrzymać dostępu do zasobu, jeżeli oczekuje na niego pisarz. Oczekujący pisarz otrzymuje dostęp najwcześniej, jak to jest możliwe, czyli zaraz po opuszczeniu zasobu przez ostatni proces, który przybył przed nim. W tym wariancie może dojść do zagłodzenia oczekujących czytelników.

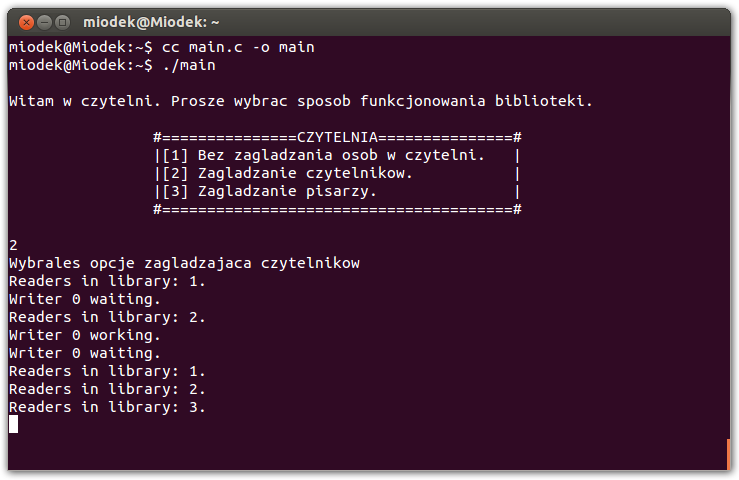
**Inne**

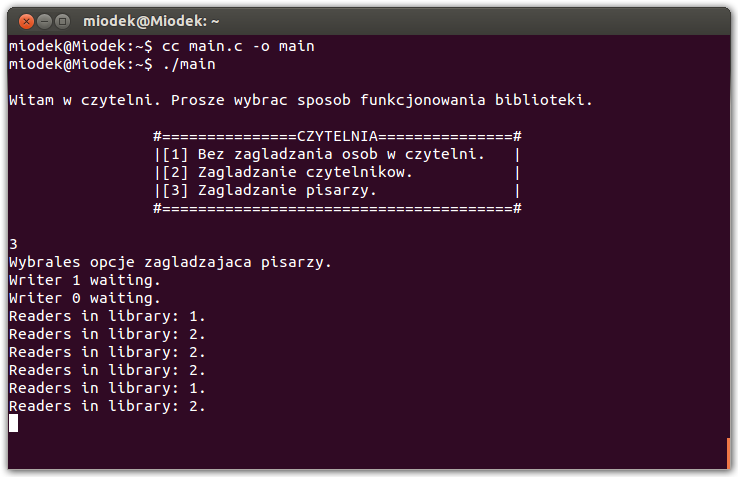
Inne rozwiązania problemu zakładają między innymi równoczesne wyeliminowanie możliwości zagłodzenia obu typów procesów, np. poprzez zastosowanie kolejki FIFO.

**c) realizacja i wykonanie:**

Dla wygody użytkownika stworzyliśmy dodatkowy program obsługujący wszystkie typy rozwiązań.  
Użytkownik wprowadza komendę, aby go skompilować :  
**cc main.c -o main**  
  
Po uruchomieniu za pomocą komendy **./main** program czeka na wprowadzenie polecenia.  
Użytkownikomi pokazuje się bardzo wygodne w użyciu menu z możliwością wyboru danego typu rozwiązania.  
Do wyboru ma jedną z trzech dostępnych opcji:  
  


**Jeżeli wybierze opcję 1:**  
  


**Jeżeli wybierze opcję 2:**   
  


**Jeżeli wybierze opcję 3:**  
****

**KODY PROGRAMÓW:  
a) bez zagładzania:**

#include <stdio.h>

#include <string.h>

#include <unistd.h>

#include <stdlib.h>

#include <sys/types.h>

#include <pthread.h>

#include <semaphore.h>

#define WRITERS\_SPOTS 4 //ilość pisarzy

#define READERS\_SPOTS 10 //ilość czytelników

int waiting\_writers = 0;

int working\_writers = 0;

int waiting\_readers = 0;

int working\_readers = 0;

sem\_t writers;

sem\_t readers;

sem\_t acces\_sem;

sem\_t writers\_block;

void \*Writer\_TH(void \*arg){

int i;  
 //Rzutowanie zmiennej arg typu void na zmienną tmp typu int  
 int \*tmp = (int\*)arg;

while(1){   
 //Blokujemy dostęp

sem\_wait(&acces\_sem);

waiting\_writers++;

//Jeżeli nie ma czekających czytelników wpuszczamy pisarzy

if(waiting\_readers == 0){

working\_writers++;

sem\_post(&acces\_sem); //Zwalniamy dostęp

}

else{

//Jeżeli są czytelnicy czekamy na zwolnienie semafora acces

sem\_post(&acces\_sem);

sem\_wait(&writers); //Pisarz czeka na dostęp

}

//Pisarz wchodzi więc blokujemy dostęp dla innych pisarzy

sem\_wait(&writers\_block);   
 printf("Writer %d working.\n", \*tmp); sleep(1);

sem\_post(&writers\_block); //Koniec pracy, odblokowujemy

sem\_wait(&acces\_sem);

working\_writers--;

waiting\_writers--;

//Jeżeli wszyscy pisarze wyszli wpusczczamy czytelników  
 if(working\_writers == 0){

do{

working\_readers++;

sem\_post(&readers); //Wpuszczamy czytelników

}

while(waiting\_readers > working\_readers);

}

sem\_post(&acces\_sem);

}

}

void\* Reader\_TH(void\* arg){

int i;

int \*tmp = (int\*)arg; //Rzutowanie, jak wyżej

while(1){

sem\_wait(&acces\_sem); //Blokujemy dostęp

waiting\_readers++;

//Jeżeli nie ma czekających pisarzy wpuszczamy czytelników

if(waiting\_writers == 0){

working\_readers++;

sem\_post(&acces\_sem); //Zwalniamy dostęp

}

else{

sem\_post(&acces\_sem); //Zwalniamy dostęp

sem\_wait(&readers); //Czytelnik czeka na dostęp

}

printf("Readers in library: %d.\n", \*tmp); sleep(1);

sem\_wait(&acces\_sem);

working\_readers--;

waiting\_readers--;

//Jeżeli nie ma już czytelników możemy wpuszczać pisarzy

if(working\_readers == 0){

do{

working\_writers++;

sem\_post(&writers);

}

while(waiting\_writers > working\_writers);

}

sem\_post(&acces\_sem);

}

}

int main(){

int i,j;

sem\_init(&writers, 0, 0); //Semafor zliczający

sem\_init(&readers,0,0); //Semafor zliczający

sem\_init(&acces\_sem,0, 1); //Semafor binarny

sem\_init(&writers\_block,0,1); //Semafor binarny

//Tablica z wątkami pisarzy, wielkość narzucona z góry

pthread\_t WritersThreads[WRITERS\_SPOTS];  
  
 //Tabilca z wątkami czytelnikow, wielkość narzucona z góry

pthread\_t ReadersThreads[READERS\_SPOTS];

int tab1[WRITERS\_SPOTS];

int tab2[READERS\_SPOTS];

for(i = 0; i < WRITERS\_SPOTS; i++){

tab1[i] = i;

pthread\_create(&WritersThreads[i], NULL, Writer\_TH, &tab1[i]);

}

for(j = 0; j < READERS\_SPOTS; j++){

tab2[i] = i;

pthread\_create(&ReadersThreads[j], NULL, Reader\_TH, &tab2[j]);

}

for(i = 0; i < WRITERS\_SPOTS; i++){

pthread\_join(WritersThreads[i], NULL);

}

for(j = 0; j < READERS\_SPOTS; j++){

pthread\_join(ReadersThreads[j], NULL);

}

}

**b) zagłodzenie pisarzy:**

#include <stdio.h>

#include <string.h>

#include <unistd.h>

#include <stdlib.h>

#include <sys/types.h>

#include <pthread.h>

#include<semaphore.h>

#define WRITERS 2 //Ilość pisarzy

#define READERS 2 //Ilość czytelników

//Deklarujemy globalnie semafory, żeby wątki miały do nich dostęp

sem\_t mutex;

sem\_t acces;

int readers\_count;

//Funkcja dla pisarza

void\* Writer\_TH(void\* arg){

while(1){

int \*tmp = (int\*)arg;

printf("Writer %d waiting.\n", \*tmp);

//sleep(1);

sem\_wait(&acces); //Blokujemy dostęp do czytelni waitem

printf("Writer %d working.\n",\*tmp);

sleep(2); //Tak jakby jego praca - sekunda

sem\_post(&acces); //Zwalniamy semafor

}

}

//Funkcja dla czytelnika

void\* Reader\_TH(void\* arg){

while(1){

sem\_wait(&mutex);

readers\_count++;

if(readers\_count == 1){

//Blokujemy semafor dla pisarzy jeżeli readers\_count > 0

sem\_wait(&acces);

}

//Odblokowujemy mutex czyli jakby dostęp dla czytelników

sem\_post(&mutex);

printf("Readers in library: %d.\n", readers\_count);

sleep(1); //Czyta, tak jak pisarz pracuje - sekunda

sem\_wait(&mutex); //Blokujemy wejście dla czytelników

readers\_count--; //Zmniejszamy aż wszyscy wyjdą

//Jeżeli wszystkich usunęliśmy to możemy odblokować dostęp dla pisarzy

if(readers\_count == 0){

sem\_post(&acces);

}

sem\_post(&mutex); //Na sam koniec znów odblokowujemy czytelników

}

}

int main(){

int i;

sem\_init(&mutex, 0, 1);

sem\_init(&acces,0, 1);

int tab[WRITERS];

//Tablica z wątkami pisarzy

pthread\_t WritersThreads[WRITERS];

//Tablica z wątkami czytelników

pthread\_t ReadersThreads[READERS];

for(i = 0; i < WRITERS; i++){ //Dla każdego pisarza tworzymy wątek

tab[i] = i;

pthread\_create(&WritersThreads[i], NULL, Writer\_TH, &tab[i]);

}

for(i = 0; i < READERS; i++){ //To co wyżej, tylko dla czytelników

pthread\_create(&ReadersThreads[i], NULL, Reader\_TH, NULL);

}

for(i = 0; i < WRITERS; i++){ //Czekamy na zakończenie watkow

pthread\_join(WritersThreads[i], NULL);

}

for(i = 0; i < READERS; i++){ //To co wyżej

pthread\_join(ReadersThreads[i], NULL);

}

}

**c) zagłodzenie czytelników:**

#include <stdio.h>

#include <string.h>

#include <unistd.h>

#include <stdlib.h>

#include <sys/types.h>

#include <pthread.h>

#include<semaphore.h>

#define WRITERS 1 //ilość pisarzy

#define READERS 10 //ilość czytelników

int readers\_waiting\_count;

int readers\_count;

sem\_t mutex;

sem\_t acces\_sem;

//Nie wpuszczamy czytelników tak długo, jak czeka jakiś pisarz

void\* Writer\_TH(void \*arg){

while(1){

int \*tmp = (int\*)arg;

readers\_waiting\_count++;

sem\_wait(&mutex);

printf("Writer %d waiting.\n", \*tmp);

sem\_post(&mutex);

sem\_wait(&acces\_sem);

printf("Writer %d working.\n", \*tmp);

sleep(2);

readers\_waiting\_count--;

sem\_post(&acces\_sem);

}

}

void \*Reader\_TH(void \*arg){

while(1){

if(readers\_waiting\_count == 0){

sem\_wait(&mutex);

readers\_count++;

sem\_post(&mutex);

if(readers\_count == 1)

sem\_wait(&acces\_sem);

printf("Readers in library: %d.\n", readers\_count);

sleep(1);

sem\_wait(&mutex);

readers\_count--;

if(readers\_count == 0)

sem\_post(&acces\_sem);

sem\_post(&mutex);

}

}

}

int main(){

int i;

sem\_init(&mutex, 0, 1);

sem\_init(&acces\_sem,0, 1);

int tab[WRITERS];

pthread\_t WritersThreads[WRITERS];

pthread\_t ReadersThreads[READERS];

for(i = 0; i < WRITERS; i++){

tab[i] = i;

pthread\_create(&WritersThreads[i], NULL, Writer\_TH, &tab[i]);

}

for(i = 0; i < READERS; i++){

pthread\_create(&ReadersThreads[i], NULL, Reader\_TH, NULL);

}

for(i = 0; i < WRITERS; i++){

pthread\_join(WritersThreads[i], NULL);

}

for(i = 0; i < READERS; i++){

pthread\_join(ReadersThreads[i], NULL);

}

}

**d) dodatkowy program (main):**

#include <stdio.h>

#include <string.h>

#include <unistd.h>

#include <stdlib.h>

#include <sys/types.h>

#include <pthread.h>

#include<semaphore.h>

int main(){

int choice;

printf("\nWitam w czytelni.");  
 printf("\nProsze wybrac sposob funkcjonowania biblioteki.\n\n");

printf("\t\t#===============CZYTELNIA===============#\n"

"\t\t|[1] Bez zagladzania osob w czytelni. |\n"

"\t\t|[2] Zagladzanie czytelnikow. |\n"

"\t\t|[3] Zagladzanie pisarzy. |\n"

"\t\t#=======================================#\n\n");

scanf("%d", &choice);

switch(choice){

case 1:

printf("Wybrales opcje bez zagladzania uzytkownikow.\n");

system("cc bez\_zagladzania.c -o bez -lpthread");

system("./bez\_zagladzania");

break;

case 2:

printf("Wybrales opcje zagladzajaca czytelnikow\n");

system("cc zagladzanie\_czytelnikow.c -o czytelnikow -lpthread");

system("./zagladzanie\_czytelnikow");

break;

case 3:

printf("Wybrales opcje zagladzajaca pisarzy.\n");

system("cc zaglodzenie\_pisarzy.c -o pisarzy -lpthread");

system("./zaglodzenie\_pisarzy");

break;

}

return 0;

}