SYSTEMY OPERACYJNE – PROJEKT NR 2 Pracownia Specjalistyczna	**DATA:** 05.06.2017 r.
Dokumentacja Temat: Problem czytelników i pisarzy.	**PROWADZACY**: dr inż. Wojciech Kwedlo
1. Karolina Grodzka 2. Ewa Kulesza 3. Damian Kiołbasa	**OCENA (pkt):**

I. CEL PROJEKTU

Czytelnicy i pisarze.

Z czytelni korzysta na okrągło pewna ilość czytelników i pisarzy, 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:
c) trzy programy symulujące trzy różne rozwiązania tego problemu, przy czym przynajmniej jeden z nich musi korzystać ze zmiennych warunkowych [34 p].

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];
```

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);
      }
}
```

```c
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 1 //Ilość pisarzy
#define READERS 3 //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 4 //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):

```c
#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;
}
```