

**INSTYTUT TELEINFORMATYKI I AUTOMATYKI**

**Wydział Cybernetyki WAT**

**Przedmiot: SYSTEMY OPERACYJNE**  
**SPRAWOZDANIE Z ĆWICZENIA LABORATORYJNEGO Nr 10**

**Temat ćwiczenia: SEMAFORY**

**Wykonał:**

**Damian Krata**  
**Grupa: I4X3S1**

**Ćwiczenie wykonane dnia**  
**19.12.2015**

**Prowadzący ćwiczenie**  
**mgr inż. Łukasz Laszko**

**Ocena:**

.....

## Zadanie 1.

### Treść zadania

- 1** Napisać trzy programy tworzące procesy, z których jeden pełni funkcję producenta umieszczającego pobrane z pliku tekstowego dane (np. pojedyncze wyrazy) w obszarze pamięci dzielonej (plik ma mieć co najmniej 100 linii). Dwa pozostałe procesy pełnią funkcje konsumentów – czytają dane przesłane przez producenta. Za pomocą semaforów IPC zaimplementować następujący model synchronizacji dostępu do pamięci dzielonej: jeden konsument odbiera dwa razy częściej niż drugi.

Punkty:  
--/1,00



Przetestować działanie programu i zarejestrować jego wyniki. Następnie usunąć mechanizm synchronizacji (funkcjonalnie wyłączyć semafony), ponownie przetestować działanie programu i zarejestrować wyniki.

W sprawozdaniu przedyskutować wyniki obu testowań programu.

Przy implementacji nie można wykorzystywać funkcji stopujących (`pause()`, ... ) ani usypiających (`sleep()`, `usleep()`, `nanosleep()`, ... ) procesy.

## Opis rozwiązania

Semafony służą do synchronizacji procesów. Pozwalają na czasowe zabezpieczenie jakichś zasobów przed innymi procesami. Ich przykładowe zastosowanie to np. współbieżny serwer zapisujący do pliku wiadomości odebrane od klientów wraz z adresem z którego nadeszły. Jeżeli adres i wiadomość są zapisywane do pliku oddzielnie może się zdarzyć, że pomiędzy zapisaniem adresu a komunikatu do pliku inny proces zapisze tam część swoich danych. Odpowiednie zastosowanie semaforów zabezpiecza nas przed taką sytuacją. Dwoma podstawowymi operacjami na semaforze są jego podniesienie oznaczane V i opuszczenie oznaczane P. Opuszczony semafor oczywiście uniemożliwia dostęp do zasobów.

## Semafor ogólny

Semafor taki może dopuszczać do zasobów określoną ilość procesów na raz. Jeżeli procesów będzie zbyt dużo to nadmiarowe będą wstrzymane do czasu zwolnienia zasobów przez któryś z poprzednich.

- $P(S)$  - jeżeli wartość semafora  $S > 0$  to  $S = S - 1$ . Jeżeli  $S == 0$  to proces zostaje wstrzymany dopóki wartość  $S$  nie będzie większa od zera
- $V(S)$  - jeżeli procesy są wstrzymane to wznów jeden z nich, w przeciwnym przypadku  $S = S + 1$

## Semafor binarny

## Semafor uogólniony

Semafor taki jest albo opuszczony ( $S=0$ ) albo podniesiony ( $S=1$ ). Innych możliwości nie ma - przepuszcza tylko jeden proces.

Semafor ten zmienia wartość licznika  $S$  o dowolną liczbę naturalną:

- $P(S,n)$  - jeżeli  $S > n$  to  $S = S-n$ , w przeciwnym wypadku proces zostaje wstrzymany.
- $V(S,n)$  - jeżeli jakieś procesy są wstrzymane wskutek operacji  $(S,m)$  i istnieje możliwość wznowienia któregoś z nich ( $m < n$ ) to  $S = S+n-m$ . W przeciwnym wypadku  $S = S+n$

## Operacje na semaforach

Oprócz blokujących operacji  $P$  i  $V$  na semaforach możemy wykonywać wiele innych operacji:

-  $Z$  - "przejsie pod semaforem" Jest to odwrotność operacji  $P$ . Jeżeli wartość  $S > 0$  to proces jest wstrzymywany, natomiast jest wznowiany gdy  $S == 0$ . Ta operacja nie zmienia wartości semafora

- nieblokujące operacje  $P$  i  $Z$ . Operacje te nie wstrzymują wykonywania procesu. Jeżeli operacja spowodowałaby wstrzymanie procesu jest sygnalizowany błąd. Operacje te są używane jeżeli chcemy sami w jakiś sposób wykorzystać czas oczekiwania na podniesienie (opuszczenie) semafora.

- zwracanie wartości semafora

- zwracanie ilości procesów oczekujących na wykonanie operacji  $P$  lub  $Z$

Ponieważ unix operuje na całych zbiorach semaforów na semaforach należących do takiego zbioru można wykonywać jednoczesne operacje. Trzeba przy tym pamiętać, że jeżeli któraś z tych operacji jest nieblokująca to jeżeli któraś z operacji na zbiorze nie może być wykonana od razu cała "duża operacja" zwróci błąd. Jeżeli wszystkie operacje są blokujące to całość zosatnie wykonana dopiero gdy można będzie wykonać wszystkie operacje elementarne.

## Obsługa semaforów z konsoli

### semget()

```
int semget(key_t key, int ns, int flags);
```

Ta funkcja na podstawie klucza tworzy lub umożliwia nam dostęp do zbioru semaforów. Parametr `key` jest kluczem do zbioru semaforów. Jeżeli różne procesy chcą uzyskać dostęp do tego samego zbioru semaforów muszą użyć tego samego klucza. Parametr `ns` to liczba semaforów która ma znajdować się w tworzonym zbiorze. Parametr `flags` określa prawa dostępu do semaforów oraz sposób wykonania funkcji.

### semctl()

```
int semctl(int semid, int semnum, int cmd, union semun arg);
```

Funkcja służy do sterowania semaforami. Jej parametry to kolejno:

`semid`    Numer zbioru semaforów

semnum Numer semafora w zbiorze (począwszy od 0)  
cmd Polecenie jakie ma być wykonane na zbiorze semaforów  
arg Parametry polecenia (definicja semnum pod tabelką)

arg - parametry polecenia, przy czym semun jest zdefiniowana następująco:

//DELETEME!

Unia *semnum* zdefiniowana jest następująco:

```
union semun
{
    int val;
    struct semid_ds *buf;
    ushort *array;
};
```

Najważniejsze polecenia do wykonania na semaforach

SETVAL nadanie semaforowi o numerze semnum wartości podanej w `arg.val`  
GETVAL odczytanie wartości semafora o numerze `semnum`  
SETALL nadanie wartości wszystkim semaforom w zbiorze. Wartości do nadania podaje się przy pomocy tablicy `arg.array`  
GETALL pobranie wartości wszystkich semaforów do tablicy wskazywanej przez `arg.array`  
GETNCNT odczytanie liczby procesów czekających wskutek wywołania operacji  $P(S)$   
GETZCNT odczytanie liczby procesów wstrzymanych wskutek wywołania operacji  $Z(S)$   
IPC\_RMID usunięcie podanego zbioru semaforów  
GETPID pobranie PID'u procesu który wykonywał operację na semaforze jako ostatni.

Pozostałe operacje korzystając z `arg.buf` umożliwiają pobranie i ustawienie niektórych ogólnych informacji o danym zbiorze semaforów.

W przypadku błędu funkcja zwraca -1 i ustawia zmienną `errno`, w przypadku sukcesu funkcja zwraca 0 lub - w zależności od polecenia `cmd` - żadaną wartość. Poleceniami dla których funkcja zwraca wartość są GETNCNT, GETZCNT, GETPID oraz GETVAL.

## semop()

```
int semop(int semid, struct sembuf *sops, size_t nsops);
```

Funkcja służy do wykonywania operacji na semaforach. Jej parametry to:

semid Identyfikator zbioru semaforów

nsops Liczba semaforów (w tym wypadku elementów tablicy `sops`) na których ma być wykonana operacja

sops Wskaźnik do tablicy struktur określających operacje na semaforach

Każa ze struktur zadeklarowana jest następująco:

```
struct sembuf
{
    ushort semnum;
    short sem_op;
    ushort sem_flg;
```

```
};
```

gdzie:

semnum Numer semafora w zbiorze

Operacja na semaforze. Możliwe operacje:

```
sem_op > 0   operacja V - powoduje zwiększenie wartości semafora o sem_op
sem_op < 0   operacja P - wstrzymuje proces lub powoduje zmniejszenie wartości
              semafora o sem_op
sem_op = 0   operacja Z
```

Możliwe flagi:

```
sem_flg 0      Operacja blokująca
IPC_NOWAIT Operacja nieblokująca
```

Ponadto wykorzystałem:

WYWOŁANIE SYSTEMOWE: shmget();

```
PROTOTYP: int shmget ( key_t key, int size, int shmflg );
ZWRACA: identyfikator pamięci dzielonej, jeżeli sukces
        -1 - błąd: errno = EINVAL ( nieprawidłowy rozmiar segmentu )
        EEXIST ( segment istnieje, nie można utworzyć
)
        EIDRM ( segment czeka na usunięcie, lub
usunięty )
        ENOENT ( segment nie istnieje )
        EACCES ( brak prawa dostępu )
        ENOMEM ( brak pamięci do utworzenia
segmentu )
```

**Pliki, które powstały przy rozwiązywaniu :**

**Lab10.c**

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <unistd.h>
#include "semun.h"
#include "naglowek.h"
```

```
static int sem_odczytu1,sem_odczytu2,sem_zapisu;
```

```
int main(void)
```

```

{
static union semun arg;
/* utworzenie zbiorow semaforow */
if ((sem_odczytu1=semget(ID_SEM_ODCZYTU1,1,IPC_CREAT | 0666)) < 0) {
printf("blad");
return 1; }
if ((sem_odczytu2=semget(ID_SEM_ODCZYTU2,1,IPC_CREAT | 0666)) < 0) {
printf("blad");
return 1; }
if ((sem_zapisu=semget(ID_SEM_ZAPISU,1,IPC_CREAT | 0666)) < 0) {
printf("blad");
return 1; }
/* ustawienie wartosci poczatkowych semaforow */
arg.val = SIZE;
if (semctl(sem_zapisu,0,SETVAL,arg) <0) {
printf("blad");
return 1; }
arg.val = 0;
if (semctl(sem_odczytu1,0,SETVAL,arg) <0) {
printf("blad");
return 1; }
arg.val = 0;
if (semctl(sem_odczytu2,0,SETVAL,arg) <0) {
printf("blad");
return 1; }

//// tworzenie pamieci dzielonej

int id = shmget (klucz, SIZE, 0666|IPC_CREAT);
id = shmget (klucz, SIZE, 0666|IPC_EXCL);
struct PamiecDzielona* buff = shmat (id, 0, 0);
inicjuj(buff);
wypisz(buff);

int potpid = fork();

if (potpid==0)
{
printf("jestesmy w procesie potomnym (producent): PID:%d,PPID:%d\n", getpid(),
getppid());
execl("producent","producent",NULL);
}

int potpid2 = fork();
if (potpid2==0)
{

```

```

        printf("jestesmy w procesie potomnym (konsument1): PID:%d,PPID:%d\n", getpid(),
getppid());
        execl("konsument1","konsument1",NULL);
    }

    int potpid3 = fork();
    if (potpid3==0)
    {
        printf("jestesmy w procesie potomnym (konsument2): PID:%d,PPID:%d\n", getpid(),
getppid());
        execl("konsument2","konsument2",NULL);
    }

    printf("proces glowny czeka na zakonczenie potomkow\n");

    wait(0);

    wait(0);

    wait(0);

}

```

## Plik SEMUN.h

```

#ifndef SEMUN_H
#define SEMUN_H                /* Prevent accidental double inclusion */

#include <sys/types.h>          /* For portability */
#include <sys/sem.h>

#if ! defined(__FreeBSD__) && ! defined(__OpenBSD__) && \
    ! defined(__sgi) && ! defined(__APPLE__)
    /* Some implementations already declare this union */
union semun {                  /* Used in calls to semctl() */
    int          val;
    struct semid_ds *   buf;
    unsigned short *   array;
#if defined(__linux__)
    struct seminfo *    __buf;
#endif
};

```

```
#endif
```

```
#endif
```

## **Plik naglowek.h**

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <unistd.h>
#include <string.h>
#ifndef _NAGLOWEK
#define _NAGLOWEK
```

```
#define ID_SEM_ODCZYTU1 876543219
#define ID_SEM_ODCZYTU2 875555555
#define ID_SEM_ZAPISU 111111111
#define SIZE 8
#define klucz 123456789
#define WORDSIZE 70
```

```
struct PamiecDzielona
{
    char buff[SIZE][WORDSIZE];
    int indeks_odczytu;
    int indeks_zapisu;
    int ile_zajetych;
    int czyja_kolej;
};
```

```
void P_zapisu(int sem_zapisu)
{
    struct sembuf sb;
    sb.sem_num=0;
    sb.sem_op=-1;
    sb.sem_flg=SEM_UNDO;
    semop(sem_zapisu,&sb,1);
}
```

```
void P_odczytu(int sem_odczytu1,int sem_odczytu2,int czyja_kolej)
{
    if(czyja_kolej==2)
    {
```



```

        struct sembuf sb;
        sb.sem_num=0;
        sb.sem_op=-1;
        sb.sem_flg=SEM_UNDO;
        semop(sem_odczytu2,&sb,1);
    }
    else
    {
        struct sembuf sb;
        sb.sem_num=0;
        sb.sem_op=-1;
        sb.sem_flg=SEM_UNDO;
        semop(sem_odczytu1,&sb,1);
    }
}

```

```

void V_zapisu(int sem_zapisu)
{
    struct sembuf sb;
    sb.sem_num=0;
    sb.sem_op=1;
    sb.sem_flg=SEM_UNDO;
    semop(sem_zapisu,&sb,1);
}

```

```

void V_odczytu(int sem_odczytu1,int sem_odczytu2,int czyja_kolej)
{
    if(czyja_kolej==2)
    {
        struct sembuf sb;
        sb.sem_num=0;
        sb.sem_op=1;
        sb.sem_flg=SEM_UNDO;
        semop(sem_odczytu2,&sb,1);
    }
    else
    {
        struct sembuf sb;
        sb.sem_num=0;
        sb.sem_op=1;
        sb.sem_flg=SEM_UNDO;
        semop(sem_odczytu1,&sb,1);
    }
}

```

```

void inicjuj(struct PamiecDzielona *buff)
{
    int i=0;
    for (i=0;i<SIZE;i++)
    {

```

```

        strcpy(buff->buff[i], "-");
    }

    buff->indeks_odczytu=0;
    buff->indeks_zapisu=0;
    buff->ile_zajetych=0;
    buff->czyja_kolej=0;
}

void wypisz(struct PamiecDzielona *buff)
{
    int i=0;
    for (i=0;i<SIZE;i++)
    {
        printf("%d %s\n",i,buff->buff[i]);
    }
}

#endif

```

## Plik konsument1.c

```

#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <unistd.h>
#include "semun.h"
#include "naglowek.h"

static int sem_odczytu1,sem_odczytu2,sem_zapisu;

void konsument1

(struct PamiecDzielona* buff)
{
    char slowo[WORDSIZE];
    while(1)
    {
        P_odczytu (sem_odczytu1,sem_odczytu2,0);
        strcpy(slowo,buff->buff[buff->indeks_odczytu]);
    }
}

```

```

        strcpy(buff->buff[buff->indeks_odczytu], "-");
        printf("konsument1: zjadlem slowo: %s\n", slowo);
        wypisz(buff);
        buff->indeks_odczytu=(buff->indeks_odczytu+1)%SIZE;
        buff->ile_zajetych=buff->ile_zajetych-1;
        buff->czyja_kolej=(buff->czyja_kolej+1)%3;
        if(buff->ile_zajetych==0)
        {
        }
        else if(buff->czyja_kolej==2)
        {
            V_odczytu(sem_odczytu1, sem_odczytu2, 2);
        }
        else
        {
            V_odczytu(sem_odczytu1, sem_odczytu2, 1);
        }

        V_zapisu(sem_zapisu);
    }
}

```

```

int main()
{
    if ((sem_odczytu1=semget(ID_SEM_ODCZYTU1, 1, IPC_EXCL)) < 0) {
        printf("blad");
        return 1; }
    if ((sem_odczytu2=semget(ID_SEM_ODCZYTU2, 1, IPC_EXCL)) < 0) {
        printf("blad");
        return 1; }
    if ((sem_zapisu=semget(ID_SEM_ZAPISU, 1, IPC_EXCL)) < 0) {
        printf("blad");
        return 1; }
    printf("jestesmy w procesie potomnym (konsument1) po funkcji execl: PID:%d,
PPID:%d\n", getpid(), getppid());
    int id = shmget (klucz, SIZE, 0666|IPC_EXCL);
    struct PamiecDzielona* buff = shmat (id, 0, 0);
    wypisz(buff);
    konsument1(buff);
}

```

## Plik konsument2.c

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <unistd.h>
#include "semun.h"
#include "naglowek.h"
```

```
static int sem_odczytu1,sem_odczytu2,sem_zapisu;
```

```
void konsument1
```

```
(struct PamiecDzielona* buff)
{
    char slowo[WORDSIZE];
    while(1)
    {
        P_odczytu(sem_odczytu1,sem_odczytu2,2);
        strcpy(slowo,buff->buff[buff->indeks_odczytu]);
        strcpy(buff->buff[buff->indeks_odczytu],"-");
        printf("konsument2: zjadlem slowo: %s\n",slowo);
        wypisz(buff);
        buff->indeks_odczytu=(buff->indeks_odczytu+1)%SIZE;
        buff->ile_zajetych=buff->ile_zajetych-1;
        buff->czyja_kolej=(buff->czyja_kolej+1)%3;
        if(buff->ile_zajetych==0)
        {
        }
        else if(buff->czyja_kolej==2)
        {
            V_odczytu(sem_odczytu1, sem_odczytu2,2);
        }
        else
        {
            V_odczytu(sem_odczytu1, sem_odczytu2,1);
        }

        V_zapisu(sem_zapisu);
    }
}
```

```
}
```

```
int main()
{
if ((sem_odczytu1=semget(ID_SEM_ODCZYTU1,1,IPC_EXCL)) < 0) {
printf("blad");
return 1; }
if ((sem_odczytu2=semget(ID_SEM_ODCZYTU2,1,IPC_EXCL)) < 0) {
printf("blad");
return 1; }
if ((sem_zapisu=semget(ID_SEM_ZAPISU,1,IPC_EXCL)) < 0) {
printf("blad");
return 1; }
    printf("jestesmy w procesie potomnym (konsument1) po funkcji execl: PID:%d,
PPID:%d\n", getpid(), getppid());
    int id = shmget (klucz, SIZE, 0666|IPC_EXCL);
    struct PamiecDzielona* buff = shmat (id, 0, 0);
    wypisz(buff);
    konsument1(buff);
}
```

## **Plik producent.c**

```
#include <stdio.h>
#include <ctype.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <unistd.h>
#include "semun.h"
#include "naglowek.h"
```

```
static int sem_odczytu1,sem_odczytu2,sem_zapisu;
```

```
void producent
```

```
(struct PamiecDzielona* buff)
{
    char slowo[WORDSIZE];
```

```

char znak;
int indeks=0;
FILE *f;
f=fopen("/home/damian/Pulpit/lab10/text.txt","r");
while( EOF != znak )
{
    indeks=0;
    int i;

    for(i=0; i<WORDSIZE; i++) slowo[i]='\0';
    while((znak=fgetc(f))!=32 && znak!=EOF && indeks<WORDSIZE)
    {
        slowo[indeks] = znak;
        indeks++;
    }
    slowo[indeks]='\0';

    strcpy(buff->buff[buff->indeks_zapisu],slowo);
    printf("wyprodukowalem slowo: %s\n",slowo);
    wypisz(buff);
    buff->indeks_zapisu=(buff->indeks_zapisu+1)%SIZE;
    buff->ile_zajetych=buff->ile_zajetych+1;
    if(buff->ile_zajetych==1)
    {
        V_odczytu(sem_odczytu1, sem_odczytu2,buff->czyja_kolej);
    }
    sleep(3);
}
fclose(f);
}

int main()
{
    if ((sem_odczytu1=semget(ID_SEM_ODCZYTU1,1,IPC_EXCL)) < 0) {
        printf("blad");
        return 1; }
    if ((sem_odczytu2=semget(ID_SEM_ODCZYTU2,1,IPC_EXCL)) < 0) {
        printf("blad");
        return 1; }
    if ((sem_zapisu=semget(ID_SEM_ZAPISU,1,IPC_EXCL)) < 0) {
        printf("blad");
        return 1; }
    printf("jestesmy w procesie potomnym (producent) po funkcji execl: PID:%d,
PPID:%d\n", getpid(), getppid());
    int id = shmget (klucz, SIZE, 0666|IPC_EXCL);
    struct PamiecDzielona* buff = shmat (id, 0, 0);
    wypisz(buff);
    producent(buff);
}

```

}

## Wyniki uruchomienia

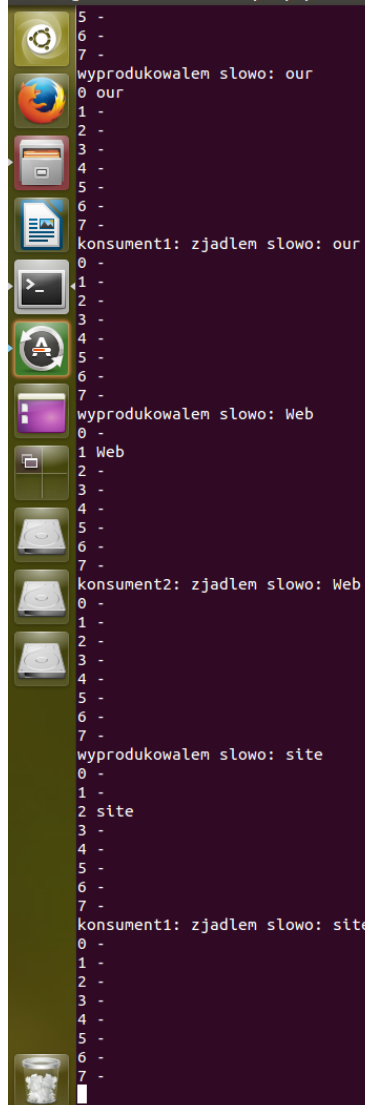
```
damian@damian-Lenovo-G500: ~/Pulpit/lab10
damian@damian-Lenovo-G500:~/Pulpit/lab10$ gcc -o producent producent.c naglowek.h
h semun.h
damian@damian-Lenovo-G500:~/Pulpit/lab10$ gcc -o konsument1 konsument1.c naglowek.h semun.h
damian@damian-Lenovo-G500:~/Pulpit/lab10$ gcc -o konsument2 konsument2.c naglowek.h semun.h
gcc: error: konsument2.h: Nie ma takiego pliku ani katalogu
damian@damian-Lenovo-G500:~/Pulpit/lab10$ gcc -o konsument2 konsument2.c naglowek.h semun.h
damian@damian-Lenovo-G500:~/Pulpit/lab10$ gcc -o program lab10.c naglowek.h semun.h
damian@damian-Lenovo-G500:~/Pulpit/lab10$ ./program
0 -
1 -
2 -
3 -
4 -
5 -
6 -
7 -
proces glowny czeka na zakonczenie potomkow
jestesmy w procesie potomnym (konsument1): PID:2751,PPID:2749
jestesmy w procesie potomnym (producent): PID:2750,PPID:2749
jestesmy w procesie potomnym (konsument2): PID:2752,PPID:2749
jestesmy w procesie potomnym (producent) po funkcji execl: PID:2750, PPID:2749
0 -
1 -
2 -
3 -
4 -
5 -
6 -
7 -
jestesmy w procesie potomnym (konsument1) po funkcji execl: PID:2752, PPID:2749
jestesmy w procesie potomnym (konsument1) po funkcji execl: PID:2751, PPID:2749
0 -
1 -
2 -
3 -
4 -
5 -
6 -
7 -
```

damian@damian-Lenovo-G500: ~/Pulpit/lab10

```
5 -  
6 -  
7 -  
wyprodukowalem slowo: was  
0 -  
1 -  
2 -  
3 -  
4 -  
5 was  
6 -  
7 -  
konsument2: zjadlem slowo: was  
0 -  
1 -  
2 -  
3 -  
4 -  
5 -  
6 -  
7 -  
wyprodukowalem slowo: designed  
0 -  
1 -  
2 -  
3 -  
4 -  
5 -  
6 designed  
7 -  
konsument1: zjadlem slowo: designed  
0 -  
1 -  
2 -  
3 -  
4 -  
5 -  
6 -  
7 -  
wyprodukowalem slowo: and  
0 -  
1 -  
2 -  
3 -  
4 -  
5 -  
6 -  
7 and  
konsument1: zjadlem slowo: and  
0 -  
1 -  
2 -  
3 -  
4 -  
5 -  
6 -  
7 -  
█
```



damian@damian-Lenovo-G500: ~/Pulpit/lab10



5 -  
6 -  
7 -  
wyprodukowalem slowo: our  
0 our  
1 -  
2 -  
3 -  
4 -  
5 -  
6 -  
7 -  
konsument1: zjadlem slowo: our  
0 -  
1 -  
2 -  
3 -  
4 -  
5 -  
6 -  
7 -  
wyprodukowalem slowo: Web  
0 -  
1 Web  
2 -  
3 -  
4 -  
5 -  
6 -  
7 -  
konsument2: zjadlem slowo: Web  
0 -  
1 -  
2 -  
3 -  
4 -  
5 -  
6 -  
7 -  
wyprodukowalem slowo: site  
0 -  
1 -  
2 site  
3 -  
4 -  
5 -  
6 -  
7 -  
konsument1: zjadlem slowo: site  
0 -  
1 -  
2 -  
3 -  
4 -  
5 -  
6 -  
7 -  
█

damian@damian-Lenovo-G500: ~/Pulpit/lab10

```
5 -  
6 -  
7 -  
wyprodukowalem slowo: eBook  
0 -  
1 -  
2 -  
3 -  
4 eBook  
5 -  
6 -  
7 -  
konsument1: zjadlem slowo: eBook  
0 -  
1 -  
2 -  
3 -  
4 -  
5 -  
6 -  
7 -  
wyprodukowalem slowo: was  
0 -  
1 -  
2 -  
3 -  
4 -  
5 was  
6 -  
7 -  
konsument2: zjadlem slowo: was  
0 -  
1 -  
2 -  
3 -  
4 -  
5 -  
6 -  
7 -  
wyprodukowalem slowo: designed  
0 -  
1 -  
2 -  
3 -  
4 -  
5 -  
6 designed  
7 -  
konsument1: zjadlem slowo: designed  
0 -  
1 -  
2 -  
3 -  
4 -  
5 -  
6 -  
7 -
```

damian@damian-Lenovo-G500: ~/Pulpit/lab10

```
wyprodukowalem slowo: For
0 -
1 -
2 -
3 -
4 For
5 -
6 -
7 -
konsument1: zjadlem slowo: For
0 -
1 -
2 -
3 -
4 -
5 -
6 -
7 -
wyprodukowalem slowo: more
0 -
1 -
2 -
3 -
4 -
5 more
6 -
7 -
konsument1: zjadlem slowo: more
0 -
1 -
2 -
3 -
4 -
5 -
6 -
7 -
wyprodukowalem slowo: free
eBooks
0 -
1 -
2 -
3 -
4 -
5 -
6 free
eBooks
7 -
konsument2: zjadlem slowo: free
eBooks
0 -
1 -
2 -
3 -
4 -
5 -
6 -
7 -
█
```

```
damian@damian-Lenovo-G500: ~/Pulpit/lab10
7 -
^C
damian@damian-Lenovo-G500:~/Pulpit/lab10$ ./program
0 -
1 -
2 -
3 -
4 -
5 -
6 -
7 -
proces glowny czeka na zakonczenie potonkow
jestesmy w procesie potomnym (producent): PID:2421,PPID:2420
jestesmy w procesie potomnym (konsument1): PID:2422,PPID:2420
jestesmy w procesie potomnym (konsument2): PID:2423,PPID:2420
jestesmy w procesie potomnym (konsument1) po funkcji execl: PID:2422, PPID:2420
jestesmy w procesie potomnym (konsument1) po funkcji execl: PID:2423, PPID:2420
0 -
jestesmy w procesie potomnym (producent) po funkcji execl: PID:2421, PPID:2420
1 -
2 -
3 -
4 -
5 -
6 -
7 -
0 -
0 -
1 -
1 -
2 -
2 -
3 -
3 -
4 -
4 -
5 -
5 -
6 -
6 -
7 -
7 -
wyprodukowalem slowo: Pride
0 Pride
1 -
2 -
3 -
4 -
5 -
6 -
7 -
konsument1: zjadlem slowo: Pride
0 -
1 -
2 -
3 -
4 -
5 -
```

W drugiej części zadania miałem do przeanalizowania to samo działanie programu tylko po wyłączeniu przeze mnie wszelkich mechanizmów kontroli dostępu. Zachowałem tylko pamięć współdzieloną a resztę kodu związaną z semaforami za komentowałem. Nie będę wstawiał tu tego kodu, ponieważ nie ma sensu ponownie tego kopiować tylko z kilkoma dodanymi /\* i \*/.

Poniżej przedstawiam wyniki uruchomienia dla programu bez mechanizmów synchronizacji:

```
damian@damian-Lenovo-G500: ~/Pulpit/lab10b/lab10
6 -
6 -
3 -
7 -
4 -
konsument2: zjadlem slowo: -
5 -
0 -
6 -
1 -
7 -
2 -
konsument1: zjadlem slowo: -
3 -
0 -
4 -
1 -
5 -
2 -
6 -
3 -
7 -
4 -
konsument2: zjadlem slowo: -
5 -
0 -
6 -
1 -
7 -
2 -
konsument1: zjadlem slowo: -
3 -
0 -
4 -
1 -
5 -
2 -
6 -
3 -
7 -
4 -
konsument2: zjadlem slowo: -
0 -
5 -
1 -
6 -
2 -
7 -
3 -
konsument1: zjadlem slowo: -
4 -
0 -
5 -
1 -
6 -
2 -
7 -
3 -
```

Jak widzimy, nie jest zachowany układ, że konsument1 dwa razy pobiera dane a następnie dopiero konsument2. Wszystko dzieje się na przemian. Mamy tutaj przykłady sytuacji kiedy to że nawet bez użycia mechanizmów synchronizacji wiele procesów nie może w jednym momencie korzystać z pamięci dzielonej ale procesy niestety korzystają z tej pamięci w sposób niekontrolowany przez programistę.

## **Wnioski:**

Wykonanie powyższego ćwiczenia pozwoliło mi zapoznać się z poleceniami do obsługi semaforów. Przygotowując się do wykonania ćwiczenia nauczyłem się używania gotowych implementacji znalezionych w internecie odnośnie semaforów na stronach takich jak

[http://4programmers.net/C/Artyku%C5%82y/Synchronizacja\\_proces%C3%B3w\\_w\\_linuxie\\_-\\_semafory](http://4programmers.net/C/Artyku%C5%82y/Synchronizacja_proces%C3%B3w_w_linuxie_-_semafory)

<http://www.cs.put.poznan.pl/akobusinska/downloads/semafory.pdf>

<http://students.mimuw.edu.pl/SO/Linux/Temat03/semafory.html>

[http://wazniak.mimuw.edu.pl/images/e/e1/Sop\\_12\\_lab.pdf](http://wazniak.mimuw.edu.pl/images/e/e1/Sop_12_lab.pdf).

Ćwiczenie uważam za zrealizowane w stopniu dobrym mimo, że większość kodu odnoszącego się do tworzenia semaforów i ich obsługi zaczerpnąłem z gotowych rozwiązań. Bardzo dużo czasu zajęło mi zrozumienie mechanizmów rządzących semaforami a i tak pisząc kod miałem problemy z ustawianiem funkcji V i P i robiłem to metodą prób i błędów.

Ponadto zadanie było bardzo podobne do poprzedniego przez co podszkoliłem swoją wiedzę niemniej jednak bardzo czasochłonne w związku z ilością kodu, którą musiałem napisać.