INSTYTUT TELEINFORMATYKI I AUTOMATYKI

Wydział Cybernetyki WAT

Przedmiot: SYSTEMY OPERACYJNE

SPRAWOZDANIE Z ĆWICZENIA LABORATORYJNEGO Nr 11

Temat ćwiczenia**: SEMAFORY/PAMIĘĆ DZIELONA**

Wykonał:

Piotr Matyjek

Grupa: **I6Y3S1**

Ćwiczenie wykonane dnia
04.01.2018

Prowadzący ćwiczenie
mgr. inż. Stanisław Matusiak

Ocena:
……

Opis rozwiązania

Został mi przydzielony numer zadania 1. Statyczna struktura buf wykorzystywana jest w funkcjach „podnieś” i „opusc”. Zgodnie z nazwami służą one do odpowiednio podnoszenia i opuszczania semafora o określonym semid i semnum. Dalej w programie deklarowane są odpowiednie zmienne. Następnie tworzone są zestawy semaforów. W pierwszym zestawie semaforów, semafor 0 odpowiada za dostęp do pisania we wspólnym obszarze pamięci przez pierwszy proces potomny. Semafor 1 odpowiada za dostęp do pisania w pamięci dzielonej przez drugi proces potomny. A semafor 2 odpowiada za dostęp do czytania pamięci dzielonej przez proces macierzysty. W drugim zestawie semaforów jest tylko jeden semafor, który służy do informowania czy pierwszy proces potomny zakończył czytanie „/etc/passwd”. Następnie tworzona jest 160-bajtowa pamięć dzielona. Następnie tworzone są procesy potomne. W tych procesach przy użyciu sprintf konwertuje pid z typu int na string. Następnie wpisuje w pierwsze miejsca tablicy znaków, a następnie przy użyciu fgets odczytuje z odpowiednich plików tekstowych linijki znaków. Ta tablica następnie jest kopiowana do obszaru pamięci dzielonej, uprzednio opuszczając odpowiednie semafory. Po kopiowaniu podnoszony jest semafor czytania. Pierwszy proces potomny po zakończeniu czytania pliku opuszcza odpowiedni semafor. Po zakończeniu czytania pliku przez drugi proces potomny ustawiany jest odpowiedni znak do pamięci dzielonej, który jest warunkiem przerwania pętli w procesie macierzystym. Proces macierzysty w pętli while czyta najpierw to co zapisał pierwszy proces potomny, o ile warunek (semafor z drugiego zestawu jest podniesiony) jest spełniony. Następnie czyta to co zapisał drugi proces potomny. „usleep” na końcu tej pętli jest ustawiony po to, aby gdy pierwszy proces potomny skończy czytać swój plik mógł zmienić wartość semafora. Po zakończeniu czytania obu plików, pętla w procesie macierzystym zostaje przerwana, zestawy semaforów są usuwane i pamięć dzielona jest zwalniana.

Kod źródłowy programu

```c
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<unistd.h>
#include<sys/types.h>
#include<sys/ipc.h>
#include<sys/sem.h>
#include<sys/shm.h>
#include <sys/stat.h>
#include<fcntl.h>

static struct sembuf buf;

void podnies (int semid, int semnum)
{
        buf.sem_num = semnum;
        buf.sem_op = 1;
        buf.sem_flg = 0;
        if(semop(semid, &buf, 1) == -1)
        {
        printf("blad przy podnoszeniu semafora\n");
        exit(1);
        }


}

void opusc (int semid, int semnum)
{
        buf.sem_num = semnum;
        buf.sem_op = -1;
        buf.sem_flg = 0;
        if(semop(semid, &buf, 1) == -1)
        {
        printf("blad przy opuszczaniu semafora\n");
        exit(1);
        }



}

int main()
{
int NR_semafora, NR_pamieci, NR_sem2;
int semid, shmid, pid, pid2, semid2;
FILE *fp1, *fp2;
char *buf_shm;
struct semid_ds buf3;
struct shmid_ds buf4;

//========tworzenie semafora========

NR_semafora=ftok(".", 'G');
NR_sem2=ftok(".", 'A');

semid = semget(NR_semafora, 3, IPC_CREAT|0666);
if(semid==-1)
{
printf("err semafor projekt\n");
exit(2);
}
```

```c
semid2 = semget(NR_sem2, 1, IPC_CREAT|0666);
if(semid2==-1)
{
printf("err semafor projekt22\n");
exit(2);
}

if(semctl(semid, 0, SETVAL, 1)== -1)
{
printf("blad nadania wartosci projekt\n");
exit(3);
}

if(semctl(semid, 1, SETVAL, 0)== -1)
{
printf("blad nadania wartosci projekt\n");
exit(3);
}
------------------------------------------------------------
if(semctl(semid, 2, SETVAL, 0)== -1)
{
printf("blad nadania wartosci projekt\n");
exit(3);
}

if(semctl(semid2, 0, SETVAL, 1)== -1)
{
printf("blad nadania wartosci projekt222\n");
exit(3);
}

//========koniec tworzenia semafora

//========tworzenie pamieci wspoldzielonej====
NR_pamieci=ftok(".", 'P');
shmid = shmget(NR_pamieci, 160, IPC_CREAT|0666);
buf_shm=(char*)shmat(shmid, NULL, 0);


//========koniec tworzenia pamieci wspoldzielonej====

if((pid=fork())<0)
{
printf("blad tworzenia potomka\n");
exit(1);
}
else
{
        if(pid==0)
        {
                char tab[160];
                char tab2[4];
                int pid=getpid(),i;

                sprintf(tab2, "%d", pid);
                for(i=0;i<4;i++)
                tab[i]=tab2[i];
                tab[4]=' ';

                if((fp1=fopen("/etc/profile","r"))==NULL)
                {
                printf("cos nie teges\n");
                exit(1);

                }
```

```
        while((fgets(&tab[5], 155, fp1))!=NULL)
        {

        opusc(semid, 0);

        strncpy(buf_shm, (char*)tab, 160);

        podnies(semid,2);
        usleep(50000);

        }

    opusc(semid2,0);
    fclose(fp1);


}
else
{
    if((pid2=fork())==0)
    {
        char tab[160];
        char tab2[4];
    int pid=getpid(),i;

    sprintf(tab2, "%d", pid);
    for(i=0;i<4;i++)
    tab[i]=tab2[i];
    tab[4]=' ';

    if((fp2=fopen("matyjek_shm.c","r"))==NULL)
    {
    printf("cos nie teges\n");
    exit(1);

    }
        while((fgets(&tab[5], 155, fp2))!=NULL)
        {
        opusc(semid, 1);
        strncpy(buf_shm, (char*)tab, 160);

        podnies(semid,2);
        }

    fclose(fp2);
    opusc(semid, 1);
    buf_shm[0]='\0';
    podnies(semid, 2);
    }
    else
    {       char tab[160];
        int valsem;
        usleep(10000);
        while(buf_shm[0]!='\0')
        {
                valsem=semctl(semid2, 0, GETVAL, 0);

                if((valsem))
                {
                opusc(semid, 2);

                strncpy(tab, buf_shm, 160);
                printf("%s",tab);
                }
```

```
                        podnies(semid, 1);
                        opusc(semid, 2);

                        strncpy(tab, buf_shm, 160);
                        printf("%s", tab);
                                if(valsem)
                                {
                                podnies(semid, 0);

                                }
                        usleep(50000);          //po to aby pierwszy potomek mógł zmienić semafor po
                                                //zakończeniu działania


                }
                semctl(semid, 0, IPC_RMID, &buf3);
                semctl(semid2, 0, IPC_RMID, &buf3);
                shmctl(shmid, IPC_RMID, &buf4);
                printf("Koncze prace\n");
        }
    }
```

```
}


return 0;
}
```

Wyniki uruchomienia

```
piotr@Piotr-Debian:~/Dokumenty/shm$ gcc matyjek_shm.c -o matyjek.o
piotr@Piotr-Debian:~/Dokumenty/shm$ ./matyjek.o

2648 # /etc/profile: system-wide .profile file for the Bourne shell (sh(1))
2649 #include<stdio.h>
2648 # and Bourne compatible shells (bash(1), ksh(1), ash(1), ...).
2649 #include<string.h>
2648
2649 #include<stdlib.h>
2648 if [ "`id -u`" -eq 0 ]; then
2649 #include<unistd.h>
2648   PATH="/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
2649 #include<sys/types.h>
2648 else
2649 #include<sys/ipc.h>
2648   PATH="/usr/local/bin:/usr/bin:/bin:/usr/local/games:/usr/games"
2649 #include<sys/sem.h>
2648 fi
2649 #include<sys/shm.h>
2648 export PATH
2649 #include <sys/stat.h>
2648
2649 #include<fcntl.h>
2648 if [ "${PS1-}" ]; then
2649
2648   if [ "${BASH-}" ] && [ "$BASH" != "/bin/sh" ]; then
2649 static struct sembuf buf;
2648     # The file bash.bashrc already sets the default PS1.
2649
2648     # PS1='\h:\w\$ '
2649 void podnies (int semid, int semnum)
2648     if [ -f /etc/bash.bashrc ]; then
2649 {
2648       . /etc/bash.bashrc
2649    buf.sem_num = semnum;
2648     fi
2649    buf.sem_op = 1;
2648   else
2649    buf.sem_flg = 0;
2648     if [ "`id -u`" -eq 0 ]; then
2649    if(semop(semid, &buf, 1) == -1)
2648       PS1='# '
2649    {
2648     else
2649    printf("blad przy podnoszeniu semafora\n");
2648       PS1='$ '
2649    exit(1);
2648     fi
2649    }
2648   fi
2649
2648 fi
2649
```

```
2649
2648
2649 }
2648 if [ -d /etc/profile.d ]; then
2649
2648   for i in /etc/profile.d/*.sh; do
2649 void opusc (int semid, int semnum)
2648     if [ -r $i ]; then
2649 {
2648       . $i
2649    buf.sem_num = semnum;
2648     fi
2649    buf.sem_op = -1;
2648   done
2649    buf.sem_flg = 0;
2648   unset i
2649    if(semop(semid, &buf, 1) == -1)
2648 fi
2649    {
2649    printf("blad przy opuszczaniu semafora\n");
2649    exit(1);
2649    }
2649
2649
2649
2649 }
2649
2649 int main()
2649 {
2649 int NR_semafora, NR_pamieci, NR_sem2;
2649 int semid, shmid, pid, pid2, semid2;
2649 FILE *fp1, *fp2;
2649 char *buf_shm;
2649 struct semid_ds buf3;
2649 struct shmid_ds buf4;
2649
2649 //========tworzenie semafora========
2649
2649 NR_semafora=ftok(".", 'G');
2649 NR_sem2=ftok(".", 'A');
2649
2649 semid = semget(NR_semafora, 3, IPC_CREAT|0666);
2649 if(semid==-1)
2649 {
2649 printf("err semafor projekt\n");
2649 exit(2);
2649 }
2649
2649 semid2 = semget(NR_sem2, 1, IPC_CREAT|0666);
2649 if(semid2==-1)
2649 {
2649 printf("err semafor projekt22\n");
```

```
2649 exit(2);
2649 }
2649
2649 if(semctl(semid, 0, SETVAL, 1)== -1)
2649 {
2649 printf("blad nadania wartosci projekt\n");
2649 exit(3);
2649 }
2649
2649 if(semctl(semid, 1, SETVAL, 0)== -1)
2649 {
2649 printf("blad nadania wartosci projekt\n");
2649 exit(3);
2649 }
2649
2649 if(semctl(semid, 2, SETVAL, 0)== -1)
2649 {
2649 printf("blad nadania wartosci projekt\n");
2649 exit(3);
2649 }
2649
2649 if(semctl(semid2, 0, SETVAL, 1)== -1)
2649 {
2649 printf("blad nadania wartosci projekt222\n");
2649 exit(3);
2649 }
2649
2649 //========koniec tworzenia semafora
2649
2649 //========tworzenie pamieci wspoldzielonej====
2649 NR_pamieci=ftok(".", 'P');
2649 shmid = shmget(NR_pamieci, 160, IPC_CREAT|0666);
2649 buf_shm=(char*)shmat(shmid, NULL, 0);
2649
2649
2649 //========koniec tworzenia pamieci wspoldzielonej====
2649
2649 if((pid=fork())<0)
2649 {
2649 printf("blad tworzenia potomka\n");
2649 exit(1);
2649 }
2649 else
2649 {
2649    if(pid==0)
2649    {
2649             char tab[160];
2649             char tab2[4];
2649             int pid=getpid(),i;
2649
2649             sprintf(tab2, "%d", pid);
2649             for(i=0;i<4;i++)
```

```
2649            tab[i]=tab2[i];
2649            tab[4]=' ';
2649
2649            if((fp1=fopen("/etc/profile","r"))==NULL)
2649            {
2649            printf("cos nie teges\n");
2649            exit(1);
2649
2649            }
2649
2649                    while((fgets(&tab[5], 155, fp1))!=NULL)
2649                    {
2649
2649                    opusc(semid, 0);
2649
2649                    strncpy(buf_shm, (char*)tab, 160);
2649
2649                    podnies(semid,2);
2649                    usleep(50000);
2649
2649                    }
2649
2649            opusc(semid2,0);
2649            fclose(fp1);
2649
2649
2649    }
2649    else
2649    {
2649            if((pid2=fork())==0)
2649            {
2649                    char tab[160];
2649                    char tab2[4];
2649            int pid=getpid(),i;
2649
2649            sprintf(tab2, "%d", pid);
2649            for(i=0;i<4;i++)
2649            tab[i]=tab2[i];
2649            tab[4]=' ';
2649
2649            if((fp2=fopen("matyjek_shm.c","r"))==NULL)
2649            {
2649            printf("cos nie teges\n");
2649            exit(1);
2649
2649            }
2649                    while((fgets(&tab[5], 155, fp2))!=NULL)
2649                    {
2649                    opusc(semid, 1);
2649                    strncpy(buf_shm, (char*)tab, 160);
2649
```

```
2649
2649                    podnies(semid,2);
2649                    }
2649
2649            fclose(fp2);
2649            opusc(semid, 1);
2649            buf_shm[0]='\0';
2649            podnies(semid, 2);
2649            }
2649            else
2649            {       char tab[160];
2649                    int valsem;
2649                    usleep(10000);
2649                    printf("\n");
2649                    while(buf_shm[0]!='\0')
2649                    {
2649                            valsem=semctl(semid2, 0, GETVAL, 0);
2649
2649                            if((valsem))
2649                            {
2649                            opusc(semid, 2);
2649
2649                            strncpy(tab, buf_shm, 160);
2649                            printf("%s",tab);
2649                            }
2649
2649
2649                            podnies(semid, 1);
2649                            opusc(semid, 2);
2649
2649                            strncpy(tab, buf_shm, 160);
2649                            printf("%s", tab);
2649                                    if(valsem)
2649                                    {
2649                                    podnies(semid, 0);
2649
2649                                    }
2649                            usleep(50000);          //po to aby pierwszy potomek mógł zmienić semafor po
2649                                                    //zakończeniu działania
2649
2649
2649                    }
2649                    semctl(semid, 0, IPC_RMID, &buf3);
2649                    semctl(semid2, 0, IPC_RMID, &buf3);
2649                    shmctl(shmid, IPC_RMID, &buf4);
2649                    printf("Koncze prace\n");
2649            }
2649    }
2649
2649 }
2649
2649
2649
2649
2649 return 0;
2649 }
Koncze prace
piotr@Piotr-Debian:~/Dokumenty/shm$
```

PODSUMOWANIE

Program został napisany w języku C i uruchomiony w środowisku systemu Linux Debian ver. 9. Program wykonuje się poprawnie, jak widać na zamieszczonych powyżej przykładach wywołania programów. Program został oddany do sprawdzenia na zajęciach laboratoryjnych i wtedy też został przesłany kod źródłowy programu na wskazany e-mail. Program nie wykonywał się całkowicie poprawnie ze względu na brak synchronizacji między procesem macierzystym i pierwszym procesem potomnym. Problem został rozwiązany i przesyłam wraz z tym sprawozdaniem kod programu.