1ο PROJECT ΛΕΙΤΟΥΡΓΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Συνεργάτες:
Βέργος Γεώργιος , ΑΜ:1072604
Τσούλος Βασίλειος, ΑΜ:1072605
Γκίκας Πέτρος , ΑΜ:1072512
Βλάχος Σταύρος ,ΑΜ:1072489
Εξάμηνο: 5ο
Ημερομηνία:23/1/2022

1ο ΜΕΡΟΣ
ΕΡΩΤΗΜΑ Α

1.

```
#include <unistd.h>
#include <stdio.h>
int main()
{
int i, pid;
pid = fork();

if (pid > 0)
{
sleep(2);
return(0);
}
for (i=0; i<3; i++)
{
printf("My parent is %d\n", getppid());
sleep(1);
}

return (0);
}
```

Εξηγηση:

Στο παραπάνω πρόγραμμα η γονική διεργασία τερματίζει πριν τη διεργασία παιδί. Η. Η πατρική διεργασία καλεί την fork(), περιμένει 2 δευτερόλεπτα (καλεί την sleep(2)) και στην συνέχεια τερματίζει. Η διεργασία παιδί συνεχίζει εμφανίζοντας το process id της γονικής διεργασίας για 3 δευτερόλεπτα. Στη πρώτη επανάληψη θα τυπώσει το id του πατέρα θα περιμένει 1 δευτερόλεπτο έπειτα θα τυπώσει πάλι το id του πατέρα ωστόσο έχουν περάσει 2 δευτερόλεπτα που αυτό σημαίνει ότι η γονική διεργασία τερμάτισε. Έτσι αλλάζει και το ppid που είναι 1 μιας και η αρχική διεργασία στο linux(init) έχει pid=1.

Θα έχουμε ως αποτέλεσμα στην οθόνη:

```
[st1072605@diogenis LS]$ ./a.out
My parent is 15777
My parent is 15777
```

Και My parent is 1.

2)

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
       int i;
       int pid;
     pid = fork();
       for (i=1; i<=500; i++)
           if (pid > 0)
                 printf("%3i (parent)\n", i);
              else
                 printf("%3i (child)\n", i);
              return (0);
}
```

Εξηγηση:

Στο παραπάνω πρόγραμμα έχουμε το εξής σενάριο: διεργασίες που εκτελούνται μαζί πατέρας και παιδί(λόγω της fork()). Εάν βρισκόμαστε στη διεργασία πατέρα αυτή θα τυπώσει 500 γραμμές το μήνυμα 001(parent) ,002(parent),...,500(parent) . Όταν βρισκόμαστε στη διεργασία παιδί θα τυπώσει 500 γραμμές το μήνυμα : 000(child),001(child),…,500(child). Επειδή όμως αυτές οι δύο διεργασίες τρέχουν ‘παράλληλα’ οι τυπώσεις αυτές επικαλύπτονται μεταξύ τους.

Θα έχουμε ως αποτελέσματα:

Κι ούτω καθεξής δηλαδή i parent

I child

Μέχρι i=500.

Ερώτημα Β

Αρχικά παραθέτουμε τον κώδικα όπου μία διεργασία δημιουργεί δύο θυγατρικές διεργασίες όπου κάθε θυγατρική διεργασία γράφει σε μία κοινή διαμοιραζόμενη μεταβλητή με αρχική τιμή 0.:

```
#include <stdio.h>          /* printf()                */
#include <stdlib.h>         /* exit(), malloc(), free() */
#include <sys/types.h>      /* key_t, sem_t, pid_t      */
#include <sys/shm.h>        /* shmat(), IPC_RMID        */
#include <errno.h>          /* errno, ECHILD            */
#include <semaphore.h>      /* sem_open(), sem_destroy(), sem_wait().. */
#include <fcntl.h>          /* O_CREAT, O_EXEC          */
int main(){
key_t shmkey;
int shmid;
int* X;


shmkey=ftok("/dev/null",5);
shmid = shmget (shmkey, sizeof(int), 0644 | IPC_CREAT);//orizeis to id ths koinhs mnhmhs
if (shmid < 0)
{
perror ("shmget\n"); // elegxos an egine lathos kata th dhmiourgia ths koinhs mnimis.
exit (1);
```

```
}
/*arxikopoihsh ths koinhs metavlitis X se 0*/
X=(int*)shmat(shmid,NULL,0);
*X=0;
int tmp,temp;//h tmp einai topikh gia kathe diergasia opote de th vazoume sth koinh mnhmh


pid_t pid[2];
int i;
for(i=0;i<2;i++){
pid[i]=fork(); // dhmiourgia 2 paidion
if(pid[i]==0){
    break;
}
}
if(pid[0]!=0 && pid[1]!=0){
for(i=0;i<2;i++){
wait(&temp); //o pateras tha perimenei thn oloklirosi ton 2 paidion tou kai meta termatizei.
}
}
/*-----dimiourgoume ta dio paidia------*/
//1o paidi
else if(pid[0]==0){
for(i=1;i<=500;i++){
tmp=(*X);
tmp=tmp+1;
(*X)=tmp;
}
}
//2o paidi
else if(pid[0]!=0 && pid[1]==0){
for(i=1;i<=500;i++){
tmp=(*X);
tmp=tmp+1;
(*X)=tmp;
}
}
```

}

Η μέγιστη τιμή που παίρνει η X(στο σενάριο ότι οι δύο διεργασίες τρέχουν ακολουθιακά) είναι X=1000.

Για να εξασφαλίσουμε ότι η X θα παίρνει τη μέγιστη τιμή X=1000 εφαρμόζουμε τον αλγόριθμο του Peterson προκειμένου ανά πάσα στιγμή μόνο μία διεργασία επιτρέπεται να βρίσκεται/εκτελεί την κρίσιμη περιοχή δηλαδή την επεξεργασία της κοινής μεταβλητής X. Για 2 διεργασίες που έχουμε(οι δυο θυγατρικές) θα χρησιμοποιήσουμε μία μεταβλητή ακέραια turn που δείχνει ποιας διεργασίας είναι η σειρά να εκτελέσει τον κώδικα της και έναν bool πίνακα flag[2] που για true τιμές δείχνει ότι η διεργασία I επιθυμεί να εισέλθει στη κρίσιμη περιοχή. Ακολουθεί ο κώδικας:

```
#include <stdio.h>          /* printf()                 */
#include <stdlib.h>         /* exit(), malloc(), free() */
#include <sys/types.h>      /* key_t, sem_t, pid_t      */
#include <sys/shm.h>        /* shmat(), IPC_RMID        */
#include <errno.h>          /* errno, ECHILD            */
#include <semaphore.h>      /* sem_open(), sem_destroy(), sem_wait().. */
#include <fcntl.h>          /* O_CREAT, O_EXEC          */
#include <stdbool.h>
int main(){
key_t shmkey;
int shmid;//to id ths koinhs mnhmhs pou kaneis allocate
int* X;//koinh metavliti X
int* turn;//koinh metavliti pou deixnei pias diergasias einai h seira
bool* flag;//koinh metavliti flag(einai pinakas 2 theseon)


shmkey=ftok("/dev/null",5);
shmid = shmget (shmkey, 2*sizeof(int)+2*sizeof(bool), 0644 | IPC_CREAT);//kini
mnimi: 4 byte gia thn turn 4 byte gia thn X kai 2 byte gia ton bool pinaka
if (shmid < 0)
{
perror ("shmget\n");
exit (1);
}
/*arxikopoihsh ths koinhs metavlitis X se 0*/
X=(int*)shmat(shmid,NULL,0);
turn=X+1;//o turn deixnei sto deytero int ths koinhs mnhmhs
flag=X+2;
```

```
*X=0;
*turn=0;
*flag=false;
*(flag+1)=false;

int tmp,temp;

pid_t pid[2];
int i;
for(i=0;i<2;i++){
pid[i]=fork();
if(pid[i]==0){
    break;
}
}
if(pid[0]!=0 && pid[1]!=0){
for(i=0;i<2;i++){
wait(&temp);
}
}
/*-----dimiourgoume ta dio paidia------*/
//1o paidi
else if(pid[0]==0){
*flag=true;//deixnoume oti to proto paidi thelei na bei sto critical section
*turn=1;//dilonoume oti h allh diergasia meta tha einai h seira ths na ektelesei critical section
while(turn==1 && *(flag+1)==true);
for(i=1;i<=500;i++){
tmp=(*X);
tmp=tmp+1;
(*X)=tmp;
}
*flag=false;
}
else if(pid[0]!=0 && pid[1]==0){
*(flag+1)=true;//deixnoume oti to deytero paidi thelei na bei sto critical section
*turn=0;
while(turn==0 && *(flag)==true);
```

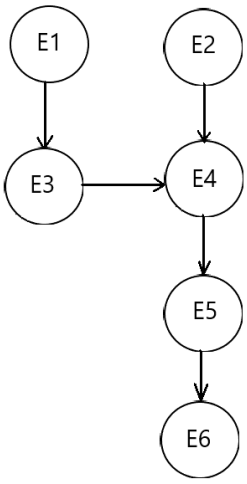
```
for(i=1;i<=500;i++){
tmp=(*X);
tmp=tmp+1;
(*X)=tmp;
}
*(flag+1)=false;
}


}
```

2^{ο} ΜΕΡΟΣ

<u>Ερώτημα Α</u>

1.

Ο γράφος προτεραιοτήτων είναι ο ακόλουθος για τις εντολές $E_1,\ldots,E_6$



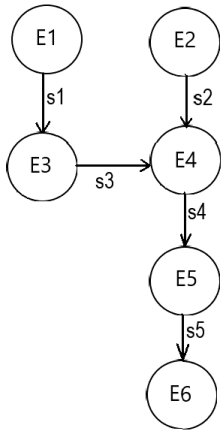
2.

Ο ‘’παράλληλος κώδικας’’ για τον παραπάνω γράφο

```
cobegin
begin
E1;
E3;
end;
E2;
coend;
begin
E4;
E5;
```

E6;
end;

3.



Συγχρονισμός των εντολών με χρήση σημαφόρων:
Το ‘’παράλληλο’’ πρόγραμμα μας θα είναι:
Var s1,s2,s3,s4,s5:semaphores;
s1=s2=s3=s4=s5=0;
cobegin
begin E1; up(s1);end;
begin down(s1); E3; up(s3); end;
begin E2; up(s2); end;
begin down(s3); down(s2); E4; up(s4); end;
begin down(s4); E5; up(s5); end;
begin down(s5); E6; end;
coend

Ερώτημα Β

Θα έχουμε:

Var s1,s2,s3,s4,s5:semaphores;

s1=s2=s3=s4=s5=0;

cobegin

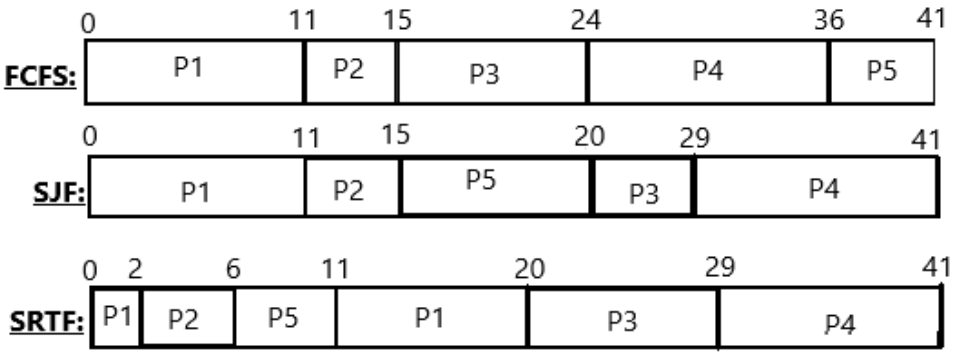
P1	P2	P3
E1.1;	wait(s1);	E3.1;
Signal(s1);	wait(s2);	signal(s2);
Wait(s3);	E2.1;	wait(s5);
E1.2;	signal(s3);	E3.2;
Signal(s4);	wait(s4);	
	E2.2;	
	Signal(s5);	

Coend;

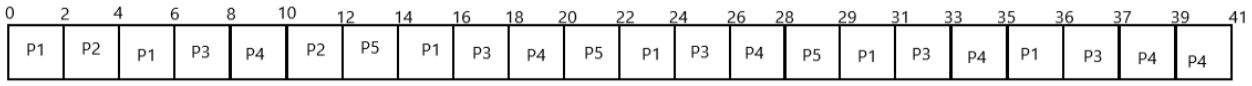
Ερώτημα Γ

Producer	Consumer	empty	Full
		2	0
	Wait(full)	2	0
Wait(empty)		1	0
Signal(full)		1	1
Wait(empty)		0	1
	Signal(empty)	1	0
Signal(full)		0	1
Wait(empty)		-1	1
Signal(full)		-1	2
Wait(empty)		-2	2
	Wait(full)	-2	1
	Signal(empty)	-1	1

Ερώτημα Ε



PR(2):



Υπολογισμός Μέσων Χρόνων Διεκπεραίωσης (ΜΧΔ):

Διεργασία	FCFS	SJF	SRTF	RR(2)
P1	11-0=11	11-0=11	20-0=2	36-0=36
P2	15-2=13	15-2=13	6-2=4	12-2=10
P3	24-3=21	29-3=26	29-3=26	37-3=34
P4	36-3=33	41-3=38	41-3=38	41-3=38
P5	41-5=36	20-5=15	11-5=6	29-5=24
ΜΧΔ	22.8	20.6	18.8	28.4

Υπολογισμός Μέσων Χρόνων Αναμονής (ΜΧΑ):

Διεργασία	FCFS	SJF	SRTF	RR(2)

P1	11-11=0	11-11=0	2-11=9	36-11=25
P2	13-4=9	13-4=9	4-4=0	10-4=6
P3	21-9=12	26-9=17	26-9=17	34-9=25
P4	33-12=21	38-12=26	38-12=26	38-12=26
P5	36-5=31	15-5=10	6-5=1	24-5=19
MXA	14.6	12.4	10.6	20.2