2ο Project Λειτουργικών Συστημάτων

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

Ερώτημα Α:

Ο κώδικας για το ερώτημα μας θα είναι:

```
#include <stdio.h>
#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          */
#define N 2
typedef sem_t Semaphore;
Semaphore* s; //epeidh to heap einai koino se oles tis diergasies prepei na epitrepoume mono se mia ana pasa stigmh na peirazei thn p kai to heap
int main(){
key_t shmkey;
int shmid;
shmkey = ftok ("/dev/null", 5);
shmid = shmget (shmkey, (N+2)*sizeof(int), 0644 | IPC_CREAT);//koinh mnhmh pou apoteleitai apo ton pointer sto heap array metavliti size diladi prosorino megethos tou pinaka kai p to athrisma pou theloume
int* p;
int* heap;
int* size;
p=(int*)shmat(shmid, NULL, 0);
*p=0; //vazoume thn metavliti p sth koinh mnhmh kai thn arxikopoioume se 0
heap=p+1;
size=p+2;
*size=0;
*(heap+0)=-1;
//arxikopoihsh tou heap
eisagogi();
int   i;
pid_t pid[2];

s = sem_open ("Semaphore", O_CREAT | O_EXCL, 0644, 1);//arxikopoihsh shmaforou se 1
for (i=0; i<2; i++)
{//dhmiourgoume dio paidia meso ths fork
pid[i] = fork ();
if (pid[i] == 0)
{
break;
}
}
if(pid[0]==0){
sem_wait(s);//sygxronizoume katalila th diergasia oste na borei na peiraksei to p mono ayth eno oi ales diegasies de tha dikaountai prosvasi
int i;
```

```
for(i=1;i<=10;i++){
(*p)=(*p)+getMin();
}
sem_post(s);
}
else if(pid[0]!=0&&pid[1]==0){
int j;
sem_wait(s);
for(j=1;j<=10;j++){
(*p)=(*p)+getMin(); //an eixame N diergasies genika tote to orio sto for tha htan pali 10 logv tou oti 10*N/N=10
}
sem_post(s);
}

}
int getMin(){
int min,last,temp,paidi;
min=*(heap+1);
last=*(heap+(*size)--)
for(temp=1;temp*2<=*(size);temp=child){
child=child*2;
if (child != *(size) && *(heap+child + 1)) < *(heap+child) {
child++;
}
if(last>*(heap+child)){
*(heap+temp)=*(heap+child);
}
else{
break;
}
*(heap+temp)=last;
return min;
} //synarthsh pou pernei to elaxisto apo to min heap(ayto to stixio sth riza) kai to diagrafei apo to heap epeita


}
void eisagogi(){//synarthsh pou eisagei ta stixia ths ekfonhshs sto min heap
int i;
for(i=1;i<=10*N;i++){
*(size)= *(size)+1;
*(heap+size)=i;
int temp=*size;
while (*(heap+temp/2) >i) {
*(heap+temp) = *(heap+ temp/ 2);
temp /= 2;
}
*(heap+temp)=i;
}
}
```

B)

Αν το heap δεν ήταν διαμοιραζόμενο τότε η μεταβλητή p δε θα πάρει τελική τιμή το άθροισμα των αριθμών [1,…,20] π.χ για N=2 δηλαδή *p=210 αλλά *p=110. Γενικά για N διεργασίες όπου το άθροισμα κανονικά(shared heap) θα είναι 10*N(10*N+1)/2 θα είναι N*10*11/2=55*N. Από αυτό καταλαβαίνουμε ότι αν δεν είναι κοινό το heap σε κάθε διεργασία παιδί το πρόγραμμα θα προσθέτει στην p μόνο τα 10 πρώτα στοιχεία του πίνακα κάθε φορά.

Ερώτημα Β:

```
#include <stdio.h>
#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         */
typedef sem_t Semaphore;
Semaphore* cSem;
Semaphore* DataBase;


int main(){
int i;                    /*     loop variables  */
pid_t pid[2];
cSem=sem_open ("cSem", O_CREAT | O_EXCL, 0644, 1);
DataBase=sem_open ("DataBase", O_CREAT | O_EXCL, 0644, 1);
key_t shmkey;
int shmid;
shmkey = ftok ("/dev/null", 5);
shmid = shmget (shmkey, 3*sizeof(int), 0644 | IPC_CREAT);
int* rc;
int* wc;
int* priority;
rc=(int*)shmat(shmid, NULL, 0);
wc=rc+1;
priority=rc+2;
*wc=0;
*rc=0;
*priority=0;

for (i=0; i<2; i++)
{
pid[i]=fork ();

if (pid[i] == 0)
{
break;
}
}
//anagnostis
if(pid[0]==0){
while(1){
sem_wait(cSem);
*rc=*rc+1;
sem_post(cSem);
sem_wait(DataBase);
if(*priority==0){
printf("Reading\n");
sem_wait(cSem);
*rc=*rc-1;
if(*wc!=0){
*priority=1;
}
sem_post(cSem);
}
```

```
else{
sem_wait(cSem);
if(*wc==0){
*priority=0;
}
*rc=*rc-1;
sem_post(cSem);
}
sem_post(DataBase);
}
}
//eggrafeas
else if(pid[0]!=0 && pid[1]==0){
while(1){
sem_wait(cSem);
*wc=*wc+1;
sem_post(cSem);
sem_wait(DataBase);
if(*priority==1){
printf("Writing\n");
sem_wait(cSem);
*wc=*wc-1;
if(*rc!=0){
*priority=0;
}
sem_post(cSem);
}
else{
sem_wait(cSem);
if(*rc==0){
*priority=1;
}
*wc=*wc-1;
sem_post(cSem);
}
sem_post(DataBase);
}
}
else{
sem_unlink ("cSem");
sem_close(cSem);
sem_unlink ("DataBase");
sem_close(DataBase);
}
```

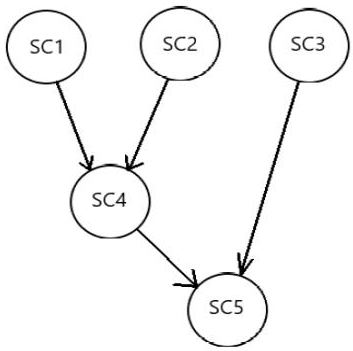
```
}
```

Μεταγλωττίζοντας το πρόγραμμα με gcc -pthread ektel askisi2_b_project.c και εκτελώντας με ./ektel
Επειδή οι δύο διεργασίες θέλουν να προσπελάσουν ταυτόχρονα τη βάση(και ο αναγνώστης και ο εγγραφέας) τυπώνεται στην οθόνη:

Reading
Writing
Reading
Και ούτω καθ’εξής όπως ζητείται.

<u>Ερώτημα Γ:</u>

Γράφος Προτεραιοτήτων:



Var s1,s2,s3,s4 semaphores;

S1=s2=s3=s4=0;
cobegin
begin SC1; signal(S1); **end**
begin SC2; signal(S2); **end**
begin wait(S1); wait(S2) SC4; signal(S3);**end**
begin SC3; signal(S4); **end**
begin wait(S3); wait(s4) SC5; **end**
coend

Κωδικας:

```
#include <stdio.h>
#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          */
typedef sem_t Semaphore;
Semaphore* s1;
Semaphore* s2;
Semaphore* s3;
Semaphore* s4;



int main(){
int   i;
pid_t pid[5];
s1 = sem_open("Sem1", O_CREAT | O_EXCL, 0644, 0);
s2 = sem_open("Sem2", O_CREAT | O_EXCL, 0644, 0);
s3 = sem_open("Sem3", O_CREAT | O_EXCL, 0644, 0);
s4 = sem_open("Sem4", O_CREAT | O_EXCL, 0644, 0);


for (i=0; i<5; i++)
{
pid[i] = fork();

if (pid[i] == 0)
{
break;
}
}
if (pid[0] == 0){
```

```
system("echo system call 1");
sem_post(s1); //signal ton shmaforo
}

else if (pid[0] != 0 && pid[1] == 0)
{
system("echo system call 2");
sem_post (s2);
}

else if (pid[0] != 0 && pid[1] != 0 && pid[2] == 0)
{
system("echo system call 3");
sem_post (s4);
}

else if (pid[0] != 0 && pid[1] != 0 && pid[2] != 0 && pid[3] == 0)
{
sem_wait(s1);
sem_wait(s2);
system("echo System call 4");
sem_post (s3);
}

else if (pid[0] != 0 && pid[1] != 0 && pid[2] != 0 && pid[3] != 0 && pid[4]==0)
{
sem_wait(s3);
sem_wait(s4);
system("echo system call 5");
}
else{
sem_unlink ("Sem1");
sem_close(s1);
sem_unlink ("Sem2");
sem_close(s2);
sem_unlink ("Sem3");
sem_close(s3);
sem_unlink ("Sem4");
sem_close(s4);
}
}
```

Μεταγλωττίζοντας το πρόγραμμα με gcc -pthread ektel2 erotimac_semaphoroi.c και εκτελώντας με ./ektel2
Τυπώνονται τα εξής:
System call 1
System call 2
System call 3
System call 4
System call 5
Σε άλλο σενάριο εκτέλεσης έχουμε:
System call 1
System call 3
System call 2
System call 4
System call 5
Που επιβεβαιώνει τον συγχρονισμό των system call με χρήση σημαφόρων.

ΕΡΩΤΗΜΑ Δ:

Το λάθος οφείλεται στο τρόπο που χρησιμοποιούνται οι δύο διαμοιραζόμενες μεταβλητές στη συνάρτηση Leave_p(free_p). Έστω ότι έχουμε το ακόλουθο σενάριο:
Έστω κάποια στιγμή το parking είναι γεμάτο και ότι ένας πελάτης θέλει να μπει στο parking. Δηλαδή θα έχουμε free_s=0 και free_a[0,n-1]=false. Η enter_p() θα σταματήσει την εκτέλεση στη γραμμή await(free_s>0). Αν κάποια μετέπειτα στιγμή κάποιος άλλος πελάτης φύγει από το parking εκτελώντας την Leave_p(free_p) θα έχουμε free_s=1. Έστω όμως ότι η εκτέλεση της Leave_p(free_p) διακοπεί πριν προλάβει να εκτελέσει την free_a[free_p]=true. Έτσι όταν θα πάει να εκτελεστεί από τον άλλο πελάτη ο υπόλοιπος κώδικας της Enter_p() . Αφού θα μειώσει την free_s=0 στη συνέχεια θα πάει να εκτελέσει την Επιλογή_θέσης() η οποία δε θα μπορεί να επιστρέψει κάποια έγκυρη θέση αφού όλες οι θέσεις του free_a είναι στο false μιας και δε πρόλαβε να εκτελεστεί η δεύτερη κρίσιμη περιοχή της Leave_p(free_p). Έτσι ο πελάτης ενώ υπάρχει ελεύθερη θέση δε του δίνεται ποτέ και παραμένει συνέχεια στο χώρο αναμονής.
Ένας τρόπος να το λύσουμε αυτό είναι να αλλάξουμε τη σειρά με την οποία εκτελούνται οι δύο κρίσιμες περιοχές στην leave_p(free_p) δηλαδή πρώτα να εκτελεστεί η :

```
region free_a do
{
free_a[free_p]:= TRUE;
}
```

Και μετά η :

```
region free_s do
{
free_s := free_s + 1;
}
```

Άρα ο κώδικας μας θα είναι:

```
Enter_p(){
Integer free_p;
Region free_s do{
Await(free_s>0);
Free_s=free_s-1;
}
Region free_a do{
Free_p=Επιλογή_θέσης(free_a);
If έγκυρη free_p{
Free_a[free_p]=false;
Τύπωσε εισιτήριο στάθμευσης με θέση free_p;
}

}

}
```

```
Leave_p(free_p){
Region free_a do{
Free_a[free_p]=true;
}
Region free_s do{
Free_s=free_s+1;
}

}
```

ΜΕΡΟΣ Β

ΕΡΩΤΗΜΑ Α

Χρονική Στιγμή	Άφιξη	Εικόνα Μνήμης	ΚΜΕ	Ουρά Μνήμης	Ουρά ΚΜΕ	Τέλος
0	P1	O:620K	-	P_1	-	-
1	P2	P1:180K O:440K	P1	P_2	-	-
2	P3	P1:180K P2:100K O:340K	P2	P_3	P1	-
3	P4,P5	P1:180K P2:100K O:340K	P2	P_3,P_5,P4	P1	P2
4	P6	P1:180K P5:90K O:350K	P5	P_3,P_4,P6	P1	-
5	-	P1:180K P5:90K P6:80K O:270K	P6	P_3,P_4	P1,P5	-
6	-	P1:180K P5:90K P6:80K O:270K	P6	P_3,P_4	P1,P5	P6
7	-	P1:180K P5:90K P3:350K	P5	P4	P1,P3	-
8	-	P1:180K P5:90K P3:350K	P5	P4	P1,P3	-
9	-	P1:180K P5:90K P3:350K	P5	P4	P1,P3	P5
10	-	P1:180K O:90K P3:350K	P3	P4	P1	-
11	-	P1:180K O:90K P3:350K	P3	P4	P1	-
12	-	P1:180K O:90K P3:350K	P3	P4	P1	-
13	-	P1:180K O:90K P3:350K	P3	P4	P1	-
14	-	P1:180K O:90K P3:350K	P3	P4	P1	P3
15	-	P1:180K P4:100K O:340K	P1	-	P4	-
16	-	P1:180K P4:100K O:340K	P1	-	P4	-
17	-	P1:180K P4:100K O:340K	P1	-	P4	
18	-	P1:180K P4:100K O:340K	P1	-	P4	-
19	-	P1:180K P4:100K O:340K	P1	-	P4	-
20	-	P1:180K P4:100K O:340K	P1	-	P4	P1
21	-	O:180K P4:100K O:340K	P4	-	-	-
22	-	O:180K P4:100K O:340K	P4	-	-	-
23	-	O:180K P4:100K O:340K	P4	-	-	-
24	-	O:180K P4:100K O:340K	P4	-	-	-
25	-	O:180K P4:100K O:340K	P4	-	-	-
26	-	O:180K P4:100K O:340K	P4	-	-	P4

ΕΡΩΤΗΜΑ Β

Α)
Μέγεθος Σελίδας : 1KB=1024 bytes
Μέγεθος Διεργασίας : 6100 bytes
Ισχύει ότι 6100=5*1024+980. Αυτό σημαίνει ότι η διεργασία θα γεμίσει πλήρως στη φυσική μνήμη 5 πλαίσια σελίδας και στο $6^{ο}$ πλαίσιο θα δεσμεύσει τα 980 bytes από τα 1024.
Άρα η **Εσωτερική Κλασματοποίηση θα είναι:** 1024-980= **44 bytes.**

Β)
Πίνακας Σελίδων

ΑΛΣ	**ΑΠΜ(dem)**	**ΑΠΜ(binary)**
0	11	1011
1	12	1100
2	1	1
3	15	1111
4	-	-
5	8	1000

1024 bytes = 2^{10} bytes αρα 10 bits → **Μετατόπιση.**
και 20 - 10 = 10 bits →**Αριθμός Σελίδας.**
Επειδή το μέγεθος πλαισίου σελίδας είναι 1Kbytes και η φυσική μνήμη έχει χωρητικότητα 4MB αυτό σημαίνει ότι αποτελείται από 2^{12} πλαίσια σελίδας. Η φυσική διεύθυνση θα αποτελείται από 22 bits τα 10 θα είναι η μετατόπιση ενώ τα πιο σημαντικά 12 bits θα είναι ο αριθμός πλαισίου σελίδας.

i)$00388_{(16)}$= 0000000001110001000_2

Αριθμός σελίδας : 0000000000_2
Μετατόπιση :1110001000_2

Από πίνακα σελίδων έχουμε 0→11→1011
Αρα **ΤΕΛΙΚΗ ΦΥΣΙΚΗ ΔΙΕΥΘΥΝΣΗ: 00**00000010111110001000$_2$ = 002F88$_{16}$

i)

0125F $_{(16)}$= 00000001001001011111

Αριθμός σελίδας : 0000000100
Μετατόπιση : 1001011111

Από πίνακα σελίδων έχουμε 4→ - δηλαδή το τμήμα αυτό της διεργασίας δε βρίσκεται στη φυσική μνήμη.

i) 015A4 $_{(16)}$= 00000001010110100100

Αριθμός σελίδας : 0000000101$_2$
Μετατόπιση : 0110100100$_2$

Από πίνακα σελίδων έχουμε 5→8→1000
Αρα **ΤΕΛΙΚΗ ΦΥΣΙΚΗ ΔΙΕΥΘΥΝΣΗ:** = 00000000100001101001002 = 0021A4$_{16}$

<u>Ερώτημα Γ</u>
Αλγόριθμος LRU:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Α.Α.Δ.	**2**	**5**	**8**	**1**	**8**	**7**	**5**	**1**	**8**	**2**	**4**	**2**	**1**	**3**	**6**	**4**	**7**	**5**	**3**	**7**
0	2	2	2	2	2	7	7	7	7	2	2	2	2	2	2	4	4	4	4	4
1		5	5	5	5	5	5	5	5	5	4	4	4	4	6	6	6	6	3	3
2			8	8	8	8	8	8	8	8	8	8	8	3	3	3	3	5	5	5
3				1	1	1	1	1	1	1	1	1	1	1	1	1	7	7	7	7
Μ/Η	**Μ**	**Μ**	**Μ**	**Μ**	**Η**	**Μ**	**Η**	**Η**	**Η**	**Μ**	**Μ**	**Η**	**Η**	**Μ**	**Μ**	**Μ**	**Μ**	**Μ**	**Μ**	**Η**

Αποτυχίες στον LRU = 13.