Λειτουργικά Project Β

Μέλη ομάδας: Μπουλαφέντης Χρήστος Ντάκος Γιώργος Παπαβασιλείου Μάριος

ΑΜ μελών: 1059612 1059569 1059649

**Μέρος Α**

---------------------------------------------------------------------------------------------------------------------------------------

**Ερώτημα Α:**

i)Με την ολοκλήρωση των 10 δευτερολέπτων, όλες οι διεργασίες που έχουν δημιουργηθεί, εκτέλεσαν την εντολή “sleep(20)” και επομένως βρίσκονται σε κατάσταση αναμονής. Άρα την χρονική στιγμή των 10 δευτερολέπτων από την εκτέλεση του προγράμματος υπάρχουν 4 διεργασίες που βρίσκονται σε κατάσταση sleeping.

ii)

#include <stdio.h>

#include <stdlib.h>

int main()

{

int pid1;

int pid2;

pid1 = fork();

if (pid1 < 0)

printf("Could not create any child\n");

else

{

pid2 = fork();

if (pid2 < 0)

printf("Could not create any child\n");

else if ( (pid1 < 0) && (pid2 < 0))

kill(pid1,9);

}

printf("I am process %d, my father is %d. Im going to sleep.\n",getpid(),getppid());

sleep(20);

return (0);

}

**Ερώτημα B:**

#include <stdio.h>

#include <stdlib.h>

#include <semaphore.h>

#include <fcntl.h>

#include <sys/mman.h>

#define max 100

#define PAGESIZE 4096

sem\_t \*synch;

int \*heap;

int \*size;

void heapifyUp(){ // Υπορουτίνα που εκτελεί την διαδικασία Heapify Up

int i = \*size;

while(1){

int parent = i/2;

if (parent > 0 && heap[parent] > heap[i]){

int t = heap[parent];

heap[parent] = heap[i];

heap[i]=t;

i = parent;

} else {

break;

}

}

}

void insert(int key){ //Υπορουτίνα που εκτελεί την διαδικασία insert

\*size = \*size + 1;

heap[\*size] = key;

heapifyUp();

}

void printHeap(){ //Υπορουτίνα που τυπώνει το δέντρο

int i = 1;

while(i <= \*size){

printf("%d ",heap[i]);

i++;

}

printf("\n");

}

int main()

{

int i=0,pid,value;

synch = sem\_open ("Sem1", O\_CREAT | O\_EXCL, 0644, 1);

size = mmap(0,sizeof(int), PROT\_READ|PROT\_WRITE, // Δημιουργεία κοινής μνήμης για την μεταβλητή size

MAP\_SHARED | MAP\_ANONYMOUS, -1, 0);

heap = mmap(0, max\*sizeof(int), PROT\_READ|PROT\_WRITE, //Δημιουργεία κοινής μνήμης για τον πίνακα heap

MAP\_SHARED | MAP\_ANONYMOUS, -1, 0);

for(i=0;i<100;i++) //Δημιουργεία υποδιεργασιών

{

pid=fork();

if(pid<0)

{printf("Fork Error!\n");}

else if(pid==0)

{break;}

}

sem\_wait(synch); //Αρχή Κρίσιμης Περιοχής

int upper=1000,lower=0; // Τροποποίηση ώστε να περιοριστούν οι τυχαίοι αριθμοί

srand(time(NULL) ^ (getpid()<<16)); // σε έυρος του 0 και του 1000

value=(rand() % (upper - lower + 1)) + lower;

insert(value);

sem\_post(synch); //Τέλος Κρίσιμης Περιοχής

sem\_unlink ("Sem1");

sem\_close(synch);

if(pid>0)

{printHeap();} //Εκτύπωση του δέντρου από τον πατέρα

return(0);}

**Ερώτημα Γ:**

Σε αυτό το ερώτημα,δεν προλάβαμε να γράψουμε το κώδικα της άσκησης,αλλά ακολουθεί ψευδοκώδικας που θα αποτελούσε πρότυπο του προγράμματος που είχαμε να σκοπό να γράψουμε: (

binary semaphore mutex = 1; (Λαβαμε υπ’οψη μας οτι για να κανει καποια εργασια εγγραφη

semaphore s = 0; πρεπει να περιμενει ολες τις ενεργες reading να τελειωσουν)

shared integer write=0;

Reading

{ while (write=1) do noop; //Περιμενει μεχρι write=0

up(s); //Δηλωνει πως αρχισε αναγνωση

<Reading function>

down(s); //Δηλωνει πως τελειωσε αναγνωση

}

Writing

{ down(mutex); //Απαγορευει αλλες διεργασιες απο το να γραψουν.

write=1; //Οταν write=1 αποτρεπει αλλες διεργασιες απο το να ξεκινησουν αναγνωση.

while (s/=0) do noop; //Οταν s/=0 σημαινει πως υπαρχουν ακομα ενεργες διεργασιες αναγωνσης.

<Writing function>

write=0; // Επιτρεπει να ξεκινησουν παλι την αναγνωση.

up(mutex); //Επιτρεπει παλι αλλες διεργασιες να γραψουν.

}

**Ερώτημα Δ:**

i)

#include<stdio.h>

#include<stdlib.h>

#include<semaphore.h>

#include <fcntl.h>

typedef sem\_t Semaphore;

Semaphore \*synch13;

Semaphore \*synch23;

Semaphore \*synch34;

Semaphore \*synch35;

int main()

{

int i;

pid\_t pid[4];

synch13 = sem\_open ("Sem1", O\_CREAT | O\_EXCL, 0644, 0);

synch23 = sem\_open ("Sem2", O\_CREAT | O\_EXCL, 0644, 0);

synch34 = sem\_open ("Sem3", O\_CREAT | O\_EXCL, 0644, 0);

synch35 = sem\_open ("Sem4", O\_CREAT | O\_EXCL, 0644, 0);

for(i=0;i<5;i++)

{

pid[i]=fork();

if(pid[i]<0)

{printf("Fork Error!\n");}

else if(pid[i]==0)

{break;}

}

if(pid[0]==0) //P1

{

system("ls -l");

printf("\n");

sem\_post(synch13);

}

if(pid[1]==0) //P2

{

system("ps -l");

printf("\n");

sem\_post(synch23);

}

if(pid[2]==0) //P3

{

sem\_wait(synch13);

sem\_wait(synch23);

system("ls -l");

printf("\n");

sem\_post(synch34);

sem\_post(synch35);

}

if(pid[3]==0) //P4

{

sem\_wait(synch34);

system("ps -l");

printf("\n");

}

if(pid[4]==0) //P5

{

sem\_wait(synch35);

system("ls -l");

printf("\n");

}

sem\_unlink ("Sem1");

sem\_close(synch13);

sem\_unlink ("Sem2");

sem\_close(synch23);

sem\_unlink ("Sem3");

sem\_close(synch34);

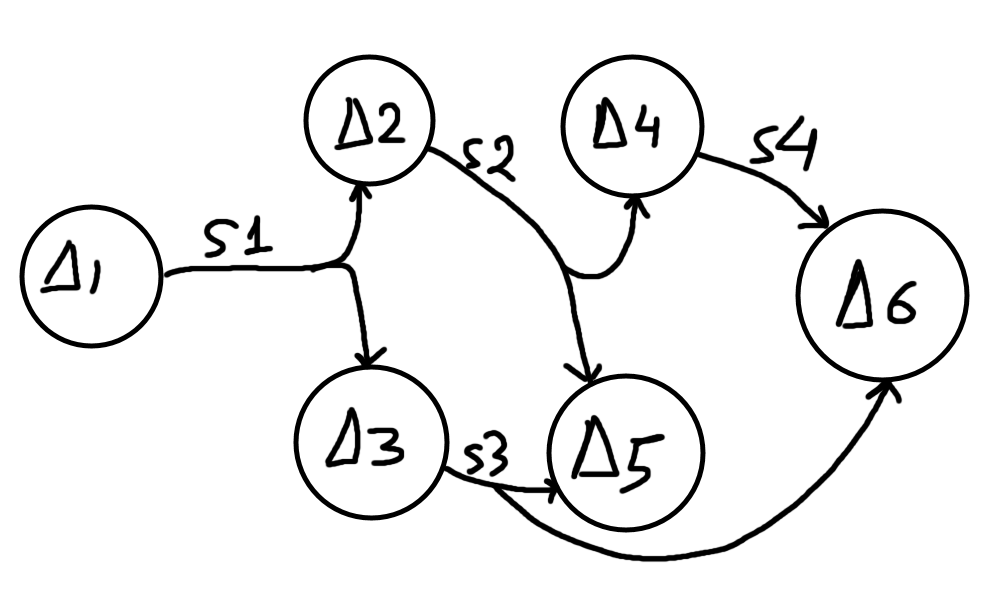
sem\_unlink ("Sem4");

sem\_close(synch35);

return(0);

}

ii)



#include<stdio.h>

#include<stdlib.h>

#include<semaphore.h>

#include <fcntl.h>

typedef sem\_t Semaphore;

Semaphore \*synch1;

Semaphore \*synch2;

Semaphore \*synch3;

Semaphore \*synch4;

int main()

{

int i,x=0;

pid\_t pid[4];

synch1 = sem\_open ("Sem1", O\_CREAT | O\_EXCL, 0644, 0);

synch2 = sem\_open ("Sem2", O\_CREAT | O\_EXCL, 0644, 0);

synch3 = sem\_open ("Sem3", O\_CREAT | O\_EXCL, 0644, 0);

synch4 = sem\_open ("Sem4", O\_CREAT | O\_EXCL, 0644, 0);

for(i=0;i<=5;i++)

{

pid[i]=fork();

if(pid[i]<0)

{printf("Fork Error!\n");}

else if(pid[i]==0)

{break;}

}

if(pid[0]==0) //P1

{

system("ls -l");

printf("1\n");

sem\_post(synch1);

sem\_post(synch1);

}

if(pid[1]==0) //P2

{

sem\_wait(synch1);

system("ps -l");

printf("2\n");

sem\_post(synch2);

sem\_post(synch2);

}

if(pid[2]==0) //P3

{

sem\_wait(synch1);

system("ls -l");

printf("3\n");

sem\_post(synch3);

sem\_post(synch3);

}

if(pid[3]==0) //P4

{

sem\_wait(synch2);

system("ps -l");

printf("4\n");

sem\_post(synch4);

}

if(pid[4]==0) //P5

{

sem\_wait(synch2);

sem\_wait(synch3);

system("ps -l");

printf("5\n");

}

if(pid[5]==0) //P6

{

sem\_wait(synch3);

sem\_wait(synch4);

system("ls -l");

printf("6\n");

}

sem\_unlink ("Sem1");

sem\_close(synch1);

sem\_unlink ("Sem2");

sem\_close(synch2);

sem\_unlink ("Sem3");

sem\_close(synch3);

sem\_unlink ("Sem4");

sem\_close(synch4);

return(0);

}

**Μέρος Β**

**Ερώτημα Α:**

**α) 39500(hex) bytes = 234752 (decimal) bytes = 1878016 (decimal) bits**

**Μεγεθος πλαισιου = 2^(32-18) bits = 16384 bits**

**Ετσι συνολικα πλαισια = 1878016/16384=114.625 ==> 115 πλαισια σελίδων.**

**Για εσωτερικη κλασματοποιηση: 115-114.625=0.375 πλαισια**

**Δηλαδη 0.375\*16384=6144 bits = 768 bytes κλασματοποιηση.**

β) Δεν αντιστοιχουν σε φυσικες διευθυνσεις, καθως:

(αριθμος πλαισιου)

31958(hex)= 1100 01100101011000 (binary)

1E800(hex)= 1111 01000 00000 000(binary)

Οπως βλεπουμε οι αριθμοι πλαισιου ειναι πολυ μικροτεροι απο 110 (οπου ειναι πιο κοντινος απο τους τελευταιους 5) ετσι λοιπον δεν αντιστοιχουν.

Ερώτημα Β)

α) 16Mbytes = 2^4 \* 2^20 \* 2^3 = 2^27 ==> k=27 bits μετατοπιση

Αρα 32-27 = 5 ==> Συνολικος αριθμος τμηματων = 2^5 = 32 τμηματα.

β) ι) Για τον αριθμο τμηματος παιρνουμε τα αρχικα 5 bits:

0Β00042Α(hex)= 00001 011000000000000010000101010 (binary)

Αριθμος τμηματος: 1 Διευθυνση βασης: 3200

Μετατοπιση= 50332714(decimal)

3200+Mετατοπιση = 50335914(decimal) = 310ΑΑ(ΗΕΧ)

ii) 02000B6D(hex) = 00000 010000000000000101101101101 (binary)

Αριθμος τμηματος: 0 Διευθυνση βασης: 1650

Μετατοπιση = 33557357(decimal)

1650+Mετατοπιση=33,559,007(decimal)=20011DF(hex)

Ερώτημα Γ)

α)

Απο ερώτημα Β ξερουμε οτι συνολικα η διευθυνση αποτελειται απο 32 bits. Τα 27 bits ειναι μετατοπιση. Αγου εφαρμοστει η μεθοδος της σελιδοποιησης με μεγεθος σελιδας 512 bytes = 4096 bits συμπαιρενουμε πως η μετατοπιση σελιδας ειναι 12 bits. Αρα τα υπολοιπα 15 bits ειναι αριθμος σελιδας.

β)

Οι σελιδες που μπορει να αποτελειται κατα μεγιστο μια διεργασια στο νεο σχημα μνημης ειναι 2^5 \* 2^15 = 2^20.

Ερώτημα Δ)

