Πολυτεχνική Σχολή

Τμήμα Μηχανικών Ηλεκτρονικών Υπολογιστών & Πληροφορικής

CEID

Λειτουργικά Συστήματα

1ο Project

Λειτουργικών Συστημάτων 2020 - 2021

ΑΒΡΑΜΟΠΟΥΛΟΣ ΜΙΧΑΗΛ | ΚΑΡΑΓΙΑΝΝΗΣ ΑΛΕΞΑΝΔΡΟΣ | ΠΑΝΤΑΖΑΡΑΣ ΕΥΣΤΡΑΤΙΟΣ

Περιεχόμενα

Μέρος 1

Ερώτημα Α

Ο κώδικας αποτελεί ένα πρόγραμμα το οποίο δημιουργεί 30 child processes. Τα processes αυτά τρέχουν <παράλληλα> και μπαίνουν σε sleep για έναν προκαθορισμένο χρόνο (διαφορετικό για το κάθε process π.χ. 60sec, 58sec, 56sec…). Έπειτα, στο τέλος γίνετε έλεγχος για το αν εκτελέστηκαν σωστά όλα τα processes και εκτυπώνετε το αντίστοιχο μήνυμα. Αναλυτικότερα ο κώδικας περιγράφετε με τα αντίστοιχα σχόλια στην παρακάτω εικόνα, δίπλα από κάθε σειρά.

```c
#include <unistd.h> /*Εισαγωγή της βιβλιοθήκης 'unistd.h' για τη χρήση
                      των συναρτήσεων 'fork', 'sleep' και 'exit'*/

#include <stdio.h> /*Εισαγωγή της βιβλιοθήκης 'stdio.h' για τη χρήση
                     της συνάρτησης 'printf'*/

#include <sys/wait.h> /*Εισαγωγή της βιβλιοθήκης 'sys/wait.h' για τη
                        χρήση των συναρτήσεων 'WAITPID','WIFEXITED','WEXITSTATUS'*/

 #define N 30    //Ανάθεση της τιμής 30 στη μεταβλητή 'N'

int main()  //Έναρξη κύριου προγράμματος
 {
    pid_t pid[N];   //Ορισμός πίνακα για τα pid
    int i;
    int child_status;   /*Ορισμός ακέραιας μεταβλητής 'child_status' για την
                          αποθήκευση του exitstatus*/
    for (i = 0; i < N; i++) //30 επαναλήψεις
    {
        pid[i] = fork();    //Δημιουργία child processes
        if (pid[i] == 0)    //έλεγχος για το αν τρέχει στο child process
        {
            sleep(60 - 2 * i);  //'Κοιμάται' το child process για (60-2*i) δευτερόλεπτα
            exit(100 + i);  //Έξοδος διεργασίας 'i' του παιδιού με κωδικό (100+i)
        }
    }
    for (i = 0; i < N; i++) //30 επαναλήψεις
    {
        pid_t wpid = waitpid(pid[i], &child_status, 0); /*περιμένει να τελειώσει ένα
                                                          child process και επιστρέφει
                                                          το status code του*/

        if (WIFEXITED(child_status))    //έλεγχος αν το process τερματίστηκε κανονικά
        {
            printf("Child%d terminated with exit status %d\n", wpid,
                WEXITSTATUS(child_status)); /*εμφάνιση μηνύματος που περιέχει το id
                                              του process και το exitstatus*/
        }
        else
        {
            printf("Child%d terminated abnormally\n", wpid); /*εμφάνιση μηνύματος που
            περιέχει το id του process στη περίπτωση που δεν τερματίσει σωστά το process*/
        }
    }
    return (0);
}
```


ΑΒΡΑΜΟΠΟΥΛΟΣ ΜΙΧΑΗΛ | ΚΑΡΑΓΙΑΝΝΗΣ ΑΛΕΞΑΝΔΡΟΣ | ΠΑΝΤΑΖΑΡΑΣ ΕΥΣΤΡΑΤΙΟΣ

Ερώτημα Β

Ο κώδικας του προγράμματος μαζί με αναλυτικά σχόλια είναι ο παρακάτω:

```
#include <unistd.h>
#include "string.h"
#include <sys/wait.h>
#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>


/*-------------------------------------------------
           ΥΛΟΠΟΙΗΣΗ BAKERY ALGORITHM
-------------------------------------------------*/

/*συνάρτηση που επιστρέφει το μέγιστο στοιχείο ενός πίνακα*/
int largest(int arr[], int n)
{
    int i;
    int max = arr[0];
    for (i = 1; i < n; i++)
        if (arr[i] > max)
            max = arr[i];

    return max;
}

int main()
{
     int n = 1; /*αρχική τιμή του n σε περίπτωση που γίνει λάθος  είσοδος*/
    /*απλό user input για τον αριθμό n*/
    printf("Enter a number of processes: ");
    scanf("%d", &n);

    key_t shmkey; //κλειδί κοινής μνήμης
    int shmid;    //identifier κοινής μνήμης
    pid_t pid[n]; //πίνακας με όλα τα process id
    int i;        //loop index
    int child_status;

    struct shared /*όλα τα shared variables ενωμένα σε ένα struct
                   (για ευκολότερη δημιουργία του shared memory space)*/
```

```
    {   bool choosing[n]; /*δείχνει αν μια διεργασία επιλέγει αυτή τη στιγμή
                            νούμερο true ή false*/

        int number[n];    /*τρέχον νούμερο μιας διεργασίας*/
        int SA[n];        /*ο κοινός πόρος, ένα απλό array*/
    };

    shmkey = ftok("/dev/null", 6); /*δημιουργία ενός κλειδιού κοινής μνήμης*/
    shmid = shmget(shmkey, sizeof(struct shared), 0644 | IPC_CREAT); /*ανάθεση
                                                κοινής μνήμης ίσης με το μέγεθος
                                                ένος "struct shared"*/

    /*attach της κοινής μνήμης σε ένα struct shared και αρχικοποίηση των πεδίων
      του σε 0*/
    struct shared *shr = (struct shared *)shmat(shmid, (void *)0, 0);
    memset((void *)shr->choosing, false, sizeof(shr->choosing));
    memset((void *)shr->number, 0, sizeof(shr->number));
    memset((void *)shr->SA, 0, sizeof(shr->SA));

    /*εκτελείται όσες φορές πει ο χρήστης*/
    for (i = 0; i < n; i++)
    {
        pid[i] = fork(); //δημιουργία διεργασίας

        if (pid[i] == 0) //τρέχει μόνο στο child process
        {

/*σε αυτό το σημείο είναι η ουσιαστική υλοποίηση του bakery algorithm*/
            int j = 0;                      //loop index
            shr->choosing[i] = true;        //αλλαγή του chosing σε true
            shr->number[i] = largest(shr->number, n) + 1; /*επιλογή αριθμού
                                                    προτεραιότητας ίσου
                                                    με το μέγιστο των
                                                    αριθμών των άλλων
                                                    διεργασιών*/

            shr->choosing[i] = false;    //αλλαγή του chosing σε false
            for (j = 0; j < n; j++)     //loop αναμονής προτεραιότητας,
                                          διατρέχει κάθε διεργασία
            {
                //όσο η διεργασία j επιλέγει
                while (shr->choosing[j] == true)
                    ; //no-op

                do
                    ; //no-op

                /*όσο η διεργασία j δεν έχει νούμερο μηδέν και (number[i],
                  i)>(number[j], j) δηλαδή όσο η j έχει προτεραιότητα*/
               while ((shr->number[j] != 0) && (shr->number[i] >
               shr->number[j] || (shr->number[i] == shr->number[j] && i > j)));
            }
```

```
            //κρίσιμο τμήμα
            for (j = 0; j < n; j++)
            {
                shr->SA[j] = shr->SA[j] + i; //απλώς αύξηση κάθε θέσης του
                                             πίνακα κατά i
            }
            shr->number[i] = 0; //θέτουμε το νούμερο σε 0, μή κρίσιμο τμήμα
            shmdt(shr);         //detach κοινής μνήμης
            exit(0);            //exit της διεργασίας-παιδί
        }
        else if (pid[i] < 0) //σε περίπτωση fork error
        {
            printf("egine lathos fork\n");
        }
    }

    /*αυτό το loop τρέχει μόνο στο parent process αφού όλα τα child έχουν κάνει
      exit*/
    for (i = 0; i < n; i++) //για κάθε παιδί
    {
        pid_t wpid = waitpid(pid[i], &child_status, 0); //αναμονή μέχρι να
                                                          τελειώσει και
                                                          αποθήκευση του status
        if (WIFEXITED(child_status))                    //αν έκανε exit χωρίς
                                                          error
        {
            //exit message με status code
            printf("Child %d terminated with exit status %d\n", wpid,
                   WEXITSTATUS(child_status));
        }
        else //αν έκανε exit με error
        {
            //exit message με status code
            printf("Child %d terminated abnormally\n", wpid);
        }
    }
    //αφού τελείωσαν τα process, κοιτάμε τι έχει ο πίνακας
    for (i = 0; i < n; i++) //για κάθε θέση του πίνακα
    {
        //εκτύπωση του περιεχομένου
        printf("Array index %d has value %d\n", i, shr->SA[i]);
    }
    shmdt(shr);                 //detach κοινής μνήμης
    shmctl(shmid, IPC_RMID, 0); //mark shared memory segment to be destroyed

    return 0;
}
```

Μέρος 2

Ερώτημα Α

1η περίπτωση:

Η μεταβλητή X **λαμβάνει** κάθε φορά **την τιμή 11** αν το 3ο βήμα της εντολής X := Y + 1 εκτελεστεί τελευταίο.

Για παράδειγμα έστω ότι η εντολή X := X + 1 είναι η διεργασία Δ1 και η εντολή X := Y + 1 είναι η διεργασία Δ2, τότε:

Διεργασία Δ1	**Διεργασία Δ2**	**X**
TX := X		0
TX := TX +1		0
	TY := Y	0
X := TX		1
	TY := TY +1	1
	X := TY	**11**

2η περίπτωση:

Η μεταβλητή X **λαμβάνει** τελικά **την τιμή 12** αν η εντολή X := Y + 1 εκτελεστεί ολόκληρη πριν από την X := X + 1

Για παράδειγμα:

Διεργασία Δ1	**Διεργασία Δ2**	**X**
	TY := Y	0
	TY := TY +1	0
	X := TY	11
TX := X		11
TX := TX +1		11
X := TX		**12**

3η περίπτωση:

Η μεταβλητή X **λαμβάνει** τελικά **την τιμή 1** αν το 1ο βήμα της εντολής X := X + 1 εκτελεστεί πριν ολοκληρωθεί η εντολή X := Y + 1 και το 3ο βήμα της X := X + 1 εκτελεστεί τελευταίο.

Για παράδειγμα:

Διεργασία Δ1	**Διεργασία Δ2**	**X**
	TY := Y	0
	TY := TY +1	0
TX := X		0
TX := TX +1		0
	X := TY	11
X := TX		**1**

Συνεπώς **οι τελικές τιμές** που είναι δυνατό να λάβει η μεταβλητή X μετά το πέρας της εκτέλεσης του παράλληλου κώδικα, λαμβάνοντας υπόψη όλες τις δυνατές περιπτώσεις εκτέλεσης (αναμείξεις interleavings) των εντολών του κώδικα **είναι 1, 11 και 12**.

Ερώτημα Β

(α)

```
var s1, s2, s3: semaphore;
s1 = s2 = s3 = 0;

cobegin
process1;
process2;
process3;
coend


process1{
    while (TRUE) {
        print("P");
        print("I");
        signal(s2);
        wait(s1);
        signal(s2);
        wait(s1);
        signal(s3);
        wait(s1);
    }
}

process2{
    while (TRUE) {
        wait(s2);
        print("Z");
        signal(s1);
    }
}

process3{
    while (TRUE) {
        wait(s3);
        print("A");
    }
})
```

- Αρχικά εκτυπώνεται το "P" και το "I".
- Στο πρώτο signal(s2) του process 1 στέλνουμε σήμα στο process 2 έτσι ώστε να εκτυπωθεί το πρώτο "Z"
- Με την εντολή wait(s1) παγώνουμε προσωρινά την εκτέλεση του process 1
- Στο signal(s1) του process 2 στέλνουμε σήμα για να συνεχίσει το process 1
- Η εντολή wait(s2) του process 2 χρησιμοποιείται για να σταματήσουν οι επαναλήψεις του process 2.
- Στο δεύτερο signal(s2) του process 1 στέλνουμε πάλι σήμα στο process 2 έτσι ώστε να εκτυπωθεί και το δεύτερο "Z".
- Αφού εκτυπωθεί και το δεύτερο "Z" συνεχίζει να εκτελείται το process 1 που είχαμε παγώσει προσωρινά προκειμένου με την εντολή signal(s3) να εκτελεστεί το process 3.
- Με την τελευταία εντολή wait(s1) του process 1 παγώνει η εκτέλεση του prosses 1.
- Εκτυπώνεται το "A"
- Η εντολή wait(s3) του process 3 χρησιμοποιείται για να σταματήσουν οι επαναλήψεις του process 3.

Άρα έχουμε εκτυπώσει ολόκληρη τη λέξη **"PIZZA"** και **έχουμε σταματήσει την εκτέλεση του προγράμματος**.

(β)

```
var s1, s2, s3: semaphore;
s1 = s2 = s3 = 0;

cobegin
process1;
process2;
process3;
coend


process1{
    while (TRUE) {
        print("P");
        print("I");
        signal(s2);
        wait(s1);
        signal(s2);
        wait(s1);
        signal(s3);
        wait(s1);
    }
}

process2{
    while (TRUE) {
        wait(s2);
        print("Z");
        signal(s1);
    }
}

process3{
    while (TRUE) {
        wait(s3);
        print("A");
        signal(s1);
    }
}
```

Παρόμοια με το παραπάνω εκτελείται και αυτό το πρόγραμμα, με τη διαφορά ότι **στο τέλος του process3 εκτελείται η εντολή signal(s1)** προκειμένου να εκτελεστεί ξανά από την αρχή ολόκληρο το πρόγραμμα.

Με αυτό τον τρόπο το πρόγραμμά μας **εκτυπώνει συνέχεια** την λέξη "PIZZA" (**PIZZAPIZZAPIZZA…**).

Ερώτημα Γ

Αν οι διεργασίες τύπου Α και οι διεργασίες τύπου Β είναι της μορφής:

Διεργασία_Α	**Διεργασία_Β**
down(s1);	down(s2);
up(s2);	down(s2);
	up(s1);
	up(s2);

Τότε για το (α) ισχύει:

s1= 2;
s2= 0;

A1
s1=2-1= 1;
s2=0+1= 1;

A2
s1=1-1= 0;
s2=1+1= 2;

B1
s2=2-1= 1;
s2=1-1= 0;
s1=0+1= 1;
s2=0+1= 1;

A3
s1=1-1= 0;
s2=1+1= 2;

B2
s2=2-1= 1;
s2=1-1= 0;
s1=0+1= 1;
s2=0+1= 1;

Άρα **μπορεί να ολοκληρωθεί** η εκτέλεση των διεργασιών με την παραπάνω σειρά χωρίς πρόβλημα.

Για το (β) ισχύει:

A1
s1=2-1= 1;
s2=0+1= 1;

A2
s1=1-1= 0;
s2=1+1= 2;

B1
s2=2-1= 1;
s2=1-1= 0;
s1=0+1= 1;
s2=0+1= 1;

B2
s2=1-1= 0;
s2=0-1= -1;

Άρα **δεν μπορεί να ολοκληρωθεί** η εκτέλεση των διεργασιών με την παραπάνω σειρά καθώς ο s2 έχει καταλήξει να έχει τη τιμή 0 και αμέσως μετά πρέπει να αφαιρεθεί ακόμη μία μονάδα. Επομένως η διεργασία B2 δεν μπορεί να συνεχίσει.

Ερώτημα Δ

(α)

```
1  while (TRUE) {
2    L: = K;
3    K: = K + 11;
7    print_num(L, L + 10);
9   }

4  while (TRUE) {
5    L: = K;
6    K: = K + 11;
8    print_num(L, L + 10);
10 }
```

Κατά την παράλληλη εκτέλεση των διεργασιών (αριστερά από κάθε γραμμή φαίνεται η σειρά εκτέλεσης της κάθε μίας) ο παραπάνω κώδικας δεν οδηγεί στο ζητούμενο αποτέλεσμα καθώς **οι 2 child processes κάνουν print το ίδιο αποτέλεσμα** (2 φορές).

(β)

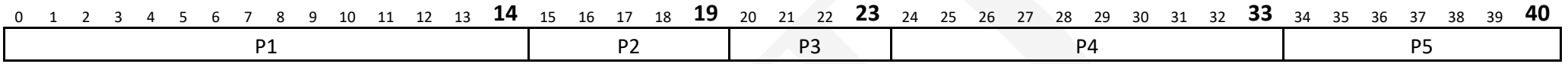
```
var S[N]:semaphores;
S = [1, 0, 0, 0, 0..., 0];
shared var K = L = 1;

//Process_i
while (TRUE) {
    wait(S[i]);
L: = K;
K: = K + 11;
    print_num(L, L + 10);
    signal(S[i + 1]);
}
```

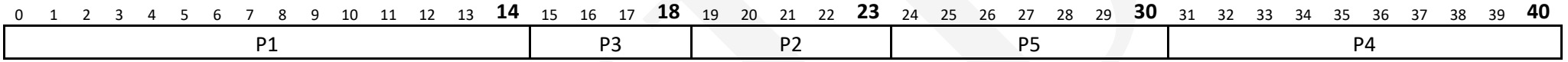
Με την προσθήκη αυτών τον εντολών στον αρχικό κώδικα, **εκτυπώνεται** κανονικά **το ζητούμενο αποτέλεσμα**.

Ερώτημα Ε

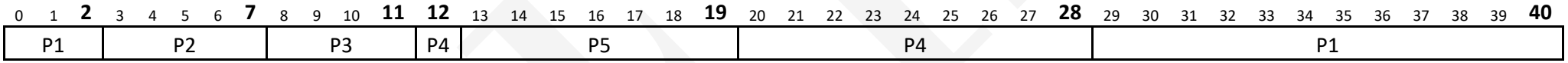
- **FCFS** (First Come First Served):



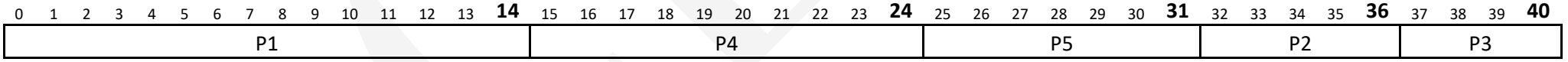
- **SJF** (Shortest Job First):



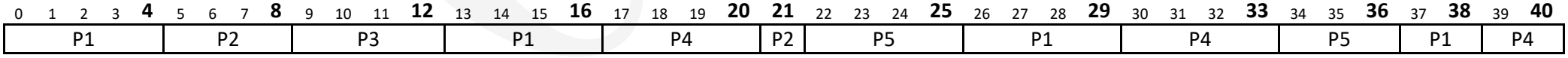
- **SRTF** (Shortest Remaining Time First):



- **PS** (Priority Scheduling) – μη προεκχωρητικός (non-preemptive priority):



- **RR** (Round Robin) με κβάντο χρόνου 4 χρονικές μονάδες:



- **Μέσος Χρόνος Διεκπεραίωσης**
  (ΜΧΔ = χρόνος ολοκλήρωσης της Pi – χρόνος άφιξης της Pi):

Διεργασία	FCFS	SJF	SRTF	PS	RR (4)
P1	14-0= 14	14-0= 14	40-0= 40	14-0= 14	38-0= 38
P2	19-2= 17	23-2= 21	7-2= 5	36-2= 34	21-2= 19
P3	23-4= 19	18-4= 14	11-4= 7	40-4= 36	12-4= 8
P4	33-7= 26	40-7= 33	28-7= 21	24-7= 17	40-7= 33
P5	40-12= 28	30-12= 18	19-12= 7	31-12= 19	36-12= 24
ΜΧΔ	104/5= **20,8**	100/5= **20**	80/5= **16**	120/5= **24**	122/5= **24,4**

- **Μέσος Χρόνος Αναμονής**
  (ΜΧΑ = χρόνος διεκπεραίωσης της Pi – χρόνος εκτέλεσης της Pi στη ΚΜΕ):

Διεργασία	FCFS	SJF	SRTF	PS	RR (4)
P1	14-14= 0	14-14= 0	40-14= 26	14-14= 0	38-14= 24
P2	17-5= 12	21-5= 16	5-5= 0	34-5= 29	19-5= 14
P3	19-4= 15	14-4= 10	7-4= 3	36-4= 32	8-4= 4
P4	26-10= 16	33-10= 23	21-10= 11	17-10= 7	33-10= 23
P5	28-7= 21	18-7= 11	7-7= 0	19-7= 12	24-7= 17
ΜΧΑ	64/5= **12,8**	60/5= **12**	40/5= **8**	80/5= **16**	82/5= **16,4**