Λειτουργικά Συστήματα

Απαντήσεις πρώτου Project

Λουδάρος Ιωάννης (1067400) και Τσικέλης Ιωάννης (1067407)

Μπορείτε να δείτε την τελευταία έκδοση του Project εδώ ή σκανάροντας τον κωδικό QR που βρίσκεται στην επικεφαλίδα.

Μέρος 1ο

Ερώτημα Α: Παρακάτω περιγράφεται η λειτουργία του δοθέντος προγράμματος

```
#include <unistd.h>
#include <stdio.h>
#include <sys/wait.h>
#define  N       30
int main()
{
      pid_t pid[N];
      int i;
      int child_status;

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

            if (pid[i] == 0)
            {
                  sleep(60-2*i);
                  exit(100+i);
            }
      }
for (i = 0; i < N; i++)
      {
      pid_t wpid = waitpid(pid[i], &child_status, 0);

      if (WIFEXITED(child_status))
            {
                  printf("Child%d terminated with exit status %d\n", wpid,
WEXITSTATUS(child_status));
            }
else
            {
                  printf("Child%d terminated abnormally\n", wpid);
            }
      }

      return (0);
}
```

Μια διεργασία η οποία δημιουργεί 30 παιδιά και αποθηκεύει τα PID τους στον πίνακα pid[N]. Το κάθε παιδί περιμένει 60-2*(αριθμός-παιδιού) δευτερόλεπτα προτού τερματιστεί με κωδικό εξόδου 100+i. Αφού δημιουργηθούν τα παιδιά, ο γονέας περιμένει να τερματιστεί το κάθε παιδί και αν αυτό τερματίστηκε φυσιολογικά τότε εκτυπώνει το μήνυμα “Child(PID παιδιού) terminated with exit status (κωδικός εξόδου παιδιού)”. Σε περίπτωση που κάποιο από τα παιδιά δεν τερματίστηκε κανονικά ο χρήστης ενημερώνεται με αντίστοιχο μήνυμα που περιλαμβάνει το PID του αντίστοιχου παιδιού. Αν τρέξουμε τον κώδικα, πράγματι βλέπουμε ότι εκτυπώνεται για κάθε παιδί (και με την σειρά που δημιουργήθηκε)
“**Child<8915-8944> terminated with exit status <100-129>**”

Σημείωση: Στην πραγματικότητα ο γονέας θα χρειαστεί να περιμένει μόνο για την πρώτη διεργασία-παιδί, οι επόμενες θα έχουν τελειώσει νωρίτερα.

Ερώτημα Β: Ο απαιτούμενος κώδικας περιλαμβάνεται στο ίδιο archive με αυτό το pdf στον υποφάκελο Question_B. Παραθέτουμε επεξηγήσεις για τα αρχεία που θα δείτε.

Question_B.c : Η κύρια υλοποίηση. Περιλαμβάνει:
- Υλοποίηση του Bakery Algorithm με σηματοφόρους και χρήση κοινής μνήμης.
- Χρήση processes. Για την ικανοποίηση της συνθήκης “Μόλις δημιουργηθούν οι N διεργασίες και μόνο τότε” χρησιμοποιείται ένας σηματοφόρος αρχικοποιημένος σε 0, τον οποίο περιμένει η διεργασία γονέας για να συνεχίσει την εκτέλεση της και που μπορεί να γίνει post μόνο από την τελευταία διεργασία παιδί (ξέρουμε ποια είναι η τελευταία διεργασία παιδί γιατί θα έχει pid=pid_γονέα+N).
- Υλοποίηση του πίνακα SA με κοινή μνήμη χρησιμοποιώντας shmget(), shmat() και ftok().

headers/max.h : Δήλωση βοηθητικής συνάρτησης για την εύρεση του μεγαλύτερου ακέραιου μέσα σε έναν πίνακα

headers/aprint.h : Δήλωση βοηθητικής συνάρτησης για την εκτύπωση όλων των στοιχείων ενός πίνακα με ακέραιους σε μορφή $\{\sigma_0,\ldots,\sigma_{N-1}\}$ όπου σ_i είναι το i-οστό στοιχείο στον πίνακα.

Μέρος 2ο

Ερώτημα Α: Αφού ο κώδικας είναι παράλληλος, υπάρχουν πολλοί τρόποι να εκτελεστούν τα αδιαίρετα κομμάτια του οδηγώντας σε διαφορετικά αποτελέσματα.

```
                X=0;
                Y=10;
              cobegin
TX:=X; (1)                  TY:=Y; (4)
TX:=TX+1; (2)               TY:=TY+1; (5)
X:=TX; (3)                  X:=TY; (6)
               coend
```

Η σειρά που επιθυμούμε είναι προφανώς αυτή που δεν έχει interleavings, συνεπώς μια από τις παρακάτω 2:

A. (1)→(2)→(3)→(4)→(5)→(6), με την X να παίρνει την τιμή **11**.

B.(4)→(5)→(6)→(1)→(2)→(3), με την X να παίρνει την τιμή **12**.

Αλλά υπάρχουν συνολικά 6! - 2 περιπτώσεις οι οποίες παρουσιάζουν interleavings. Παραθέτω κάποιες χαρακτηριστικές περιπτώσεις που οδηγούν στα δυνατά διαφορετικά αποτελέσματα:

C.(1)→(4)→(2)→(5)→(3)→(6), με την X να παίρνει την τιμή **1**.
D.(1)→(2)→(4)→(5)→(3)→(6), με την X να παίρνει την τιμή 11.
E.(4)→(1)→(5)→(2)→(6)→(3), με την X να παίρνει την τιμή 1

Ερώτημα Β: Οι ψευδοκώδικες για τις δυο περιπτώσεις παρατίθενται παρακάτω:

(α)

```
var s1,s2,s3 : semaphore;
var i : int;
begin
s1:=1; s2=0; s3=0; i=0;

cobegin
//Process 1
while (TRUE) {
    wait(s1);
    print("P");
    print("I");
    signal(s2);
    signal(s2);
}
//Process 2
while (TRUE){
    wait(s2);
    print("Z");
    i++;
if(i==2)signal(s3);
}
//Process 3
while (TRUE){
    wait(s3);
    print("A");
}coend;
end
```

(β)

```
var s1,s2,s3 : semaphore;
var i : int;
begin
s1:=1; s2=0; s3=0; i=0;

cobegin
//Process 1
while (TRUE) {
    wait(s1);
    print("P");
    print("I");
    signal(s2);
    signal(s2);
}
//Process 2
while (TRUE){
    if(i==2)
      {
       signal(s3);
       i=0;
      }
    wait(s2);
    print("Z");
    i++;
}
//Process 3
while (TRUE){
    wait(s3);
    print("A");
    signal(s1);
}coend;
```

Και στις δυο περιπτώσεις χρησιμοποιούμε σημαφόρους για να διασφαλίσουμε την σειρά εκτέλεσης των διεργασιών. Ιδιαίτερο ενδιαφέρον έχει να παρατηρήσουμε το Process_2, το οποίο πρώτα σιγουρεύεται ότι έχει εκτελεστεί δύο φορές (μέσω του i) και μετά “δίνει σήμα” στο Process_3. Οι υλοποιήσεις διαφέρουν μεταξύ τους μόνο στο ότι το Process_3 κάνει signal() το Porcess_1 να ξαναξεκινήσει τον κύκλο και ότι υπάρχει η ανάγκη μηδενισμού του i ώστε να μετριούνται σωστά κάθε φορά τα “Z”.

Ερώτημα Γ: Οι δύο περιπτώσεις αναλύονται παρακάτω:

Γνωρίζουμε τα εξής:

- Αρχικά ο s1 έχει τιμή 2 και ο s2 έχει τιμή 0.
- Για να εκτελεστεί μια διεργασία τύπου Α χρειάζεται να καταναλώσει “1” από τον s1 και προσθέτει “1” στον s2.
- Για να εκτελεστεί μια τύπου Β χρειάζεται να καταναλώσει “2” από τον s2 και προσθέτει “1” στους s1 και s2.

(α) Η σειρά A1;A2;B1;A3;B2; δεν παραβιάζει τις παραπάνω προϋποθέσεις σε περίπτωση που οι εντολές εκτελεστούν ακολουθιακά. **Συνεπώς θα εκτελεστούν κανονικά**, αφού όπως βλέπουμε παρακάτω, κάθε φορά ο αντίστοιχος σημαφόρος είναι αρκετά up για να αντέξει τα εκάστοτε down:

```
A1: down(s1); up(s2);                          (1) [s1=1, s2=1]
A2: down(s1); up(s2);                          (2) [s1=0, s2=2]
B1: down(s2); down(s2); up(s1); up(s2);)       (3) [s1=1, s2=1]
A3: down(s1); up(s2);                          (4) [s1=0, s2=2]
B2: down(s2); down(s2); up(s1); up(s2);)       (5) [s1=1, s2=1]
```

(β) Η σειρά A1;A2;B1;B2;A3; **δεν είναι δυνατόν να εκτελεστεί**. Το πρόβλημα εντοπίζεται στην γραμμή (4). Ενώ η B2 απαιτεί ο s2 να έχει τουλάχιστον τιμή 2 πριν την εκτέλεση της, αυτός έχει μόνο 1

```
A1: down(s1); up(s2);                          (1) [s1=1, s2=1]
A2: down(s1); up(s2);                          (2) [s1=0, s2=2]
B1: down(s2); down(s2); up(s1); up(s2);)       (3) [s1=1, s2=1]
B2: down(s2); down(s2); up(s1); up(s2);)       (4) [s1=2, s2=0]
(!!!): Για να εκτελεστεί η πάνω σειρά θα πρέπει ο s2 να περάσει από την τιμή
-1 το οποίο είναι αδύνατον.
```

Ερώτημα Δ: Παρατηρούμε απευθείας ότι ο κώδικας που δίνεται για τις διεργασίες δεν λαμβάνει κανένα μέτρο συγχρονισμού, με αποτέλεσμα να καταλήγει σε interleavings.

(α) Μπορούμε να βρούμε πάρα πολλά σενάρια στα οποία μπορούν να προκληθούν λάθη αλλά παραθέτουμε το παρακάτω:

Process 1	Process 2
`L:=K;`	
`K:=K+11;`	
	`L:=K;`
	`K:=K+11;`
`print_num(L, L+10);`	
	`print_num(L, L+10);`

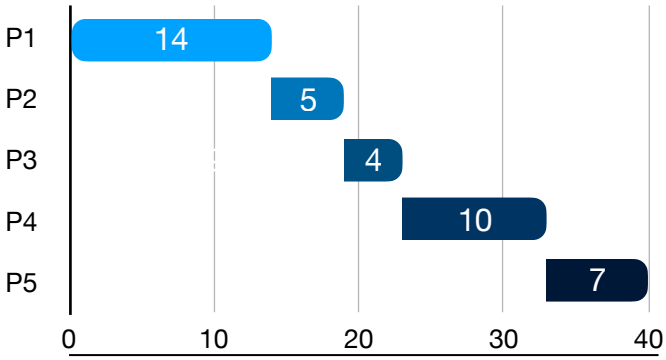
Όπως είναι προφανές, ακόμη και η πρώτη φορά εκτέλεσης της print_num() το αποτέλεσμα θα είναι μη αναμενόμενο. Συγκεκριμένα θα μας τυπωθούν διαδοχικά οι αριθμοί από το 12 έως το 22 (αντί 1 έως 11).

```
shared var K = L = 1;
var mutex : semaphore;


//Process_i

while (TRUE) {
    wait(mutex);
    count = count+1;
    L:=K;
    K:=K+11;
    print_num(L, L+10);
    signal(mutex);
}
```

(β) Στην πραγματικότητα το μόνο που χρειαζόμαστε είναι ένας και μοναδικός σημαφόρος (mutex) που θα εμποδίζει τις διεργασίες από το να εκτελούνται πριν τελειώσει η προηγούμενη. Δεν χρειάζεται καν να ανησυχήσουμε για την σειρά με την οποία θα εκτελεστούν ή αν θα εκτελεστούν όλες αφού όλες κάνουν ακριβώς το ίδιο πράγμα με τις ίδιες μεταβλητές. Έτσι απλά τροποποιώντας τον κώδικα σύμφωνα με αυτά που φαίνονται αριστερά, θα έχουμε το επιθυμητό αποτέλεσμα.

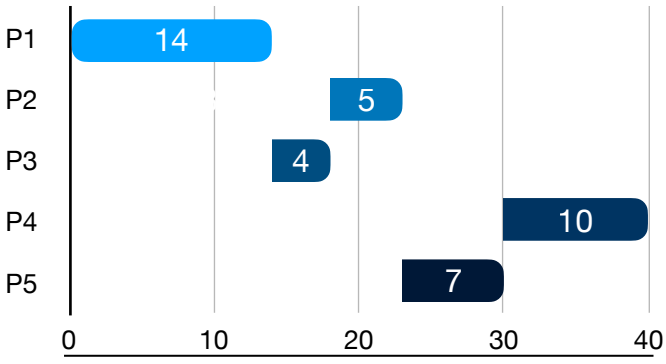
Ερώτημα Ε: Τα Gantt και οι απαραίτητοι υπολογισμοί παρατίθενται παρακάτω. Μέσα σε κάθε χρωματιστό πλαίσιο αναγράφεται το πόσο απασχόλησε η διεργασία την ΚΜΕ. Όταν οι γωνίες του πλαισίου είναι στρογγυλεμένες σημαίνει ότι η διεργασία τερματίστηκε.
Με bold σημειώνονται στους πίνακες τα ζητούμενα αποτελέματα.

First Come First Serve



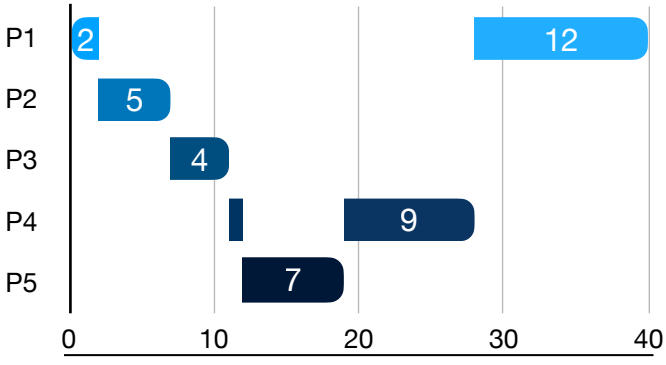
P	t_1	t_2	d	ΧΔ	ΧΑ
P1	0	14	14	14	0
P2	2	19	5	17	12
P3	4	23	4	19	15
P4	7	33	10	26	16
P5	12	40	7	28	21
MXA				-	**12,8**
MXΔ				**20,8**	-

Shortest Job First



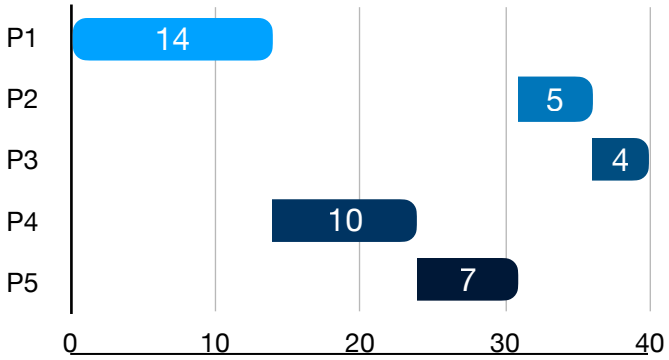
P	t_1	t_2	d	ΧΔ	ΧΑ
P1	0	14	14	14	0
P2	2	23	5	21	16
P3	4	18	4	14	10
P4	7	40	10	33	23
P5	12	30	7	18	11
MXA				-	**12**
MXΔ				**20**	-

Shortest Remaining Time First



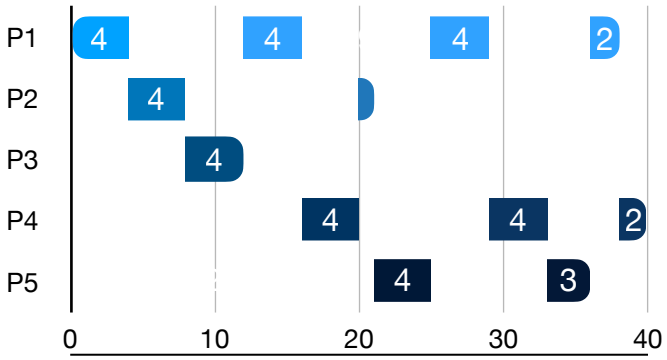
P	t_1	t_2	d	ΧΔ	ΧΑ
P1	0	40	14	40	26
P2	2	7	5	5	0
P3	4	11	4	7	3
P4	7	28	10	21	11
P5	12	19	7	7	0
MXA				-	**8**
MXΔ				**16**	-

Priority Scheduling



P	t_1	t_2	d	XΔ	XA
P1	0	14	14	14	0
P2	2	36	5	34	29
P3	4	40	4	36	32
P4	7	24	10	17	7
P5	12	31	7	19	12
MXA				-	**16**
MXΔ				**24**	-

Round Robing



P	t_1	t_2	d	XΔ	XA
P1	0	38	14	38	24
P2	2	21	5	19	14
P3	4	12	4	8	4
P4	7	40	10	33	23
P5	12	36	7	24	17
MXA				-	**16,4**
MXΔ				**24,4**	-