

ΑΝΑΦΟΡΑ 1^{ΟΥ} PROJECT ΛΕΙΤΟΥΡΓΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΜΕΡΟΣ ΠΡΩΤΟ

Ερώτημα Α

Το πρόγραμμα που μας δίνεται αρχικοποιεί μια ακέραια μεταβλητή `pid`, η οποία στη συνέχεια επιστρέφει τη τιμή της συνάρτησης `fork()`, η οποία διασπάει την διεργασία σε άλλες δύο. Έτσι η μεταβλητή `pid` θα έχει την τιμή 0 αν πρόκειται για διεργασία - παιδί, και μεγαλύτερη του 0 αν πρόκειται για γονική. Έπειτα τρέχει ένα loop 200 φορές, στο οποίο ανάλογα με την τιμή του `pid` διαχωρίζονται οι διεργασίες σε γονικές και διεργασίες - παιδιά.

Μετά την εκτέλεση του προγράμματος στο τερματικό των Linux, παρατηρούμε πως εκτυπώνονται 200 μηνύματα γονέα και 200 μηνύματα παιδιού, άρα συνολικά 400 μηνύματα, το οποίο ήταν αναμενόμενο λόγω της `fork()`. Η σειρά εμφάνισης αυτών των μηνυμάτων διαφέρει από υπολογιστή σε υπολογιστή, αλλά η μέση σειρά εμφάνισης έπειτα από αρκετές δοκιμές ήταν να εμφανίζονται πρώτα τα 200 μηνύματα γονέα και μετά τα 200 μηνύματα παιδιού.

Ερώτημα Β

Αρχικοποιούμε την ακέραια μεταβλητή `pid`, η οποία θα έχει την επιστρεφόμενη τιμή κλήσης της `fork()`, η οποία καλείται 10 φορές μέσα σε ένα loop, όσες είναι και οι θυγατρικές διεργασίες που επιθυμούμε να παράξουμε. Αν το `pid` ισούται με 0, τότε πρόκειται για θυγατρική διεργασία και εκτυπώνεται το `id` της και το `id` του γονέα της, το οποίο παραμένει ίδιο και για τις 10 θυγατρικές, αφού είναι παιδιά του ίδιου πατέρα. Ο πατέρας περιμένει την ολοκλήρωση της εκτέλεσης όλων των παιδιών μέσω της συνάρτησης `wait()` και έπειτα τερματίζει αισιώς και ο ίδιος.

Ερώτημα Γ

Ακολουθώντας την ίδια διαδικασία με προηγουμένως, τρέχουμε την `fork()` 10 φορές όσες είναι και οι διεργασίες που θέλουμε να παράξουμε. Αν η τιμή του `pid` που επιστρέφεται από τη `fork()` είναι μεγαλύτερη του 0, πρόκειται για γονική διεργασία που έχει ένα μοναδικό παιδί και τυπώνει το `id` του πατέρα της, το δικό της `id` και το `id` του παιδιού που δημιουργεί. Κάθε γονική διεργασία κάποιας άλλης διεργασίας περιμένει πρώτα να ολοκληρωθεί η εκτέλεση του παιδιού της, μέσω της `wait()`, ούτως ώστε να τερματίσει και εκείνη. Έτσι δημιουργούμε μια αλυσίδα 10 διεργασιών, όπου το `Id` της κάθε μίας είναι το `Id` του γονέα της άλλης.

```
P1 codes — -bash — 60x25
1 (parent)
2 (parent)
3 (parent)
4 (parent)
5 (parent)
6 (parent)
7 (parent)
8 (parent)
9 (parent)
10 (parent)
11 (parent)
12 (parent)
13 (parent)
14 (parent)
15 (parent)
16 (parent)
17 (parent)
18 (parent)
19 (parent)
20 (parent)
21 (parent)
22 (parent)
23 (parent)
24 (parent)
25 (parent)

P1 codes — -bash — 60x25
193 (parent)
194 (parent)
195 (parent)
196 (parent)
197 (parent)
198 (parent)
199 (parent)
200 (parent)
1 (child)
2 (child)
3 (child)
4 (child)
5 (child)
6 (child)
7 (child)
8 (child)
9 (child)
10 (child)
11 (child)
12 (child)
13 (child)
14 (child)
15 (child)
16 (child)
17 (child)

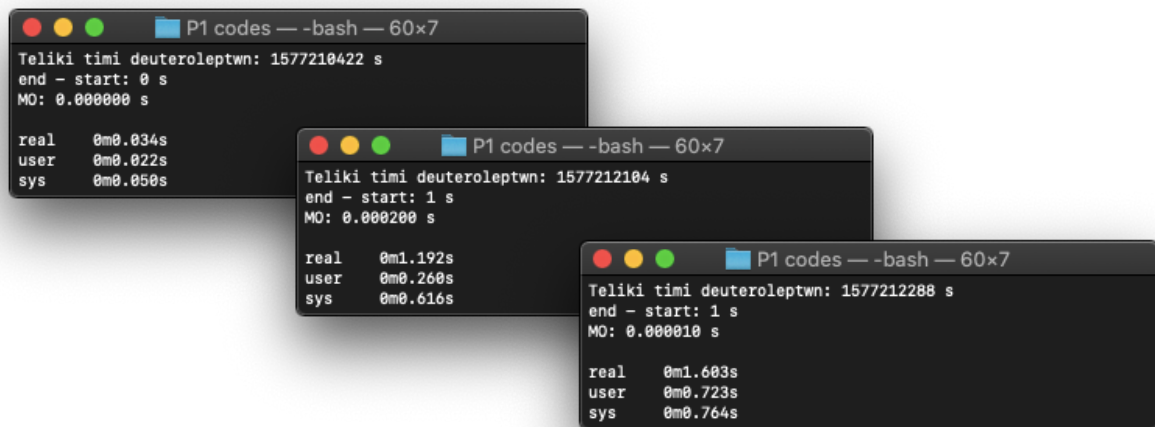
P1 codes — -bash — 60x25
176 (child)
177 (child)
178 (child)
179 (child)
180 (child)
181 (child)
182 (child)
183 (child)
184 (child)
185 (child)
186 (child)
187 (child)
188 (child)
189 (child)
190 (child)
191 (child)
192 (child)
193 (child)
194 (child)
195 (child)
196 (child)
197 (child)
198 (child)
199 (child)
200 (child)
```

```
P1 codes — -bash — 60x10
My pid is 6586 and my parent's id is 6585
My pid is 6587 and my parent's id is 6585
My pid is 6588 and my parent's id is 6585
My pid is 6589 and my parent's id is 6585
My pid is 6590 and my parent's id is 6585
My pid is 6591 and my parent's id is 6585
My pid is 6592 and my parent's id is 6585
My pid is 6593 and my parent's id is 6585
My pid is 6594 and my parent's id is 6585
My pid is 6595 and my parent's id is 6585
```

```
P1 codes — -bash — 60x10
Father = 762, Id = 7225, Child = 7239
Father = 7225, Id = 7239, Child = 7240
Father = 7239, Id = 7240, Child = 7241
Father = 7240, Id = 7241, Child = 7242
Father = 7241, Id = 7242, Child = 7243
Father = 7242, Id = 7243, Child = 7244
Father = 7243, Id = 7244, Child = 7245
Father = 7244, Id = 7245, Child = 7246
Father = 7245, Id = 7246, Child = 7247
Father = 7246, Id = 7247, Child = 7248
```

Ερώτημα Δ

Δημιουργούμε το ζητούμενο πρόγραμμα όπως αυτό περιγράφεται στην εκφώνηση του ερωτήματος. Για τη λήψη των πειραματικών δεδομένων χρησιμοποιήσαμε τη συνάρτηση `time()`, η κλήση της οποίας επιστρέφει σε δευτερόλεπτα το χρονικό διάστημα από την 1^η Ιανουαρίου του 1970 (Unix timestamp) μέχρι τη στιγμή που γίνεται η κλήση κάθε φορά της συνάρτησης. Αρχικά, τρέχουμε το πρόγραμμα για 100 διεργασίες και παρατηρούμε πως δεν υπάρχει καμία χρονική διαφορά ανάμεσα στην αρχή και το τέλος εκτέλεσης του προγράμματος. Γράφοντας στο τερματικό την εντολή `time ./a.out` (με `a.out` να είναι το όνομα του εκτελέσιμου), μπορούμε να δούμε την `real` τιμή από αυτές που εμφανίζει, όπου πρόκειται για τον πραγματικό χρόνο εκτέλεσης του προγράμματος, ο οποίος είναι πράγματι αμελητέος. Επειδή τα αποτελέσματα για 5000 και για 10000 είναι παρεμφερή, δοκιμάζουμε να τρέξουμε το πρόγραμμα και για 100000 διεργασίες. Παρατηρούμε πως αυξάνοντας τον αριθμό των διεργασιών, η διαφορά μεγαλώνει αλλά όχι σε πολύ μεγάλο βαθμό. Παραθέτουμε τα αποτελέσματα με την εντολή `time` για 100, 5000 και 100000 διεργασίες:



```
P1 codes — -bash — 60x7
Teliki timi deuteroleptwn: 1577210422 s
end - start: 0 s
MO: 0.000000 s

real    0m0.034s
user    0m0.022s
sys     0m0.050s

P1 codes — -bash — 60x7
Teliki timi deuteroleptwn: 1577212104 s
end - start: 1 s
MO: 0.000200 s

real    0m1.192s
user    0m0.260s
sys     0m0.616s

P1 codes — -bash — 60x7
Teliki timi deuteroleptwn: 1577212288 s
end - start: 1 s
MO: 0.000010 s

real    0m1.603s
user    0m0.723s
sys     0m0.764s
```

ΜΕΡΟΣ ΔΕΥΤΕΡΟ

Ερώτημα Α

α) Λύση με 5 σημαφόρους:

```
var S1, S2, S3, S4, S5: semaphores;
S1 = S2 = S3 = S4 = 0;
S5 = 1;

cobegin

  Process1
  for k=1 to 10 do
    begin
      wait(S5);
      E1.1;
      signal(S1);
      E1.2;
      signal(S3);
    end

  Process2
  for j=1 to 10 do
    begin
      wait(S1);
      E2.1;
      signal(S2);
      wait(S3);
      E2.2;
      signal(S4);
    end

  Process3
  for l=1 to 10 do
    begin
      wait(S2);
      E3.1;
      wait(S4);
      E3.2;
      signal(S5);
    end

coend
```

β) Μπορούμε να ελαττώσουμε τον αριθμό των σημοφόρων σε 3, καθώς η Process2 πρέπει να περιμένει την Process1, και αντίστοιχα η Process3 την Process2 και η Process1 την Process3. Λύση με χρήση λιγότερων σημαφόρων πιθανόν να οδηγούσε σε περιπτώσεις αδιεξόδου ή καταστρατήγησης της ορθής σειράς εκτέλεσης εντολών.

```
var S1, S2, S3: semaphores;
S1 = S2 = 0;
S3 = 1;

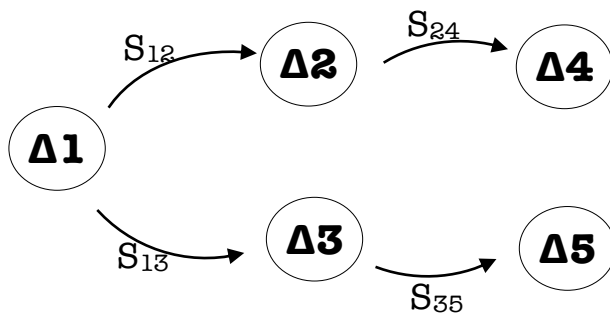
cobegin

  Process1
  for k=1 to 10 do
    begin
      wait(S3);
      E1.1;
      signal(S1);
      E1.2;
      signal(S1);
    end

  Process2
  for j=1 to 10 do
    begin
      wait(S1);
      E2.1;
      signal(S2);
      wait(S1);
      E2.2;
      signal(S2);
    end

  Process3
  for l=1 to 10 do
    begin
      wait(S2);
      E3.1;
      wait(S2);
      E3.2;
      signal(S3);
    end

coend
```

Ερώτημα Β1

Προκειμένου να πετύχουμε τον ζητούμενο συγχρονισμό, χρησιμοποιούμε έναν σημαφόρο S_{ij} με αρχική τιμή 0, για έλεγχο κάθε σχέσης προτεραιότητας $\Delta_i \rightarrow \Delta_j$.

Αρχικά εκτελείται η Δ_1 . Έπονται οι Δ_2 και Δ_3 , οι οποίες εκτελούνται αφού ολοκληρωθεί η Δ_1 . Τέλος η Δ_4 εκτελείται μετά το πέρας της Δ_2 και η Δ_5 μετά την Δ_3 .

Ερώτημα Β2

```

var S12, S13, S24, S35: semaphores;
S12 = S13 = S24 = S35 = 0;

cobegin

    Process1
begin
    y = random(1...10);
    write(buf1, y);
    up(S12);
    up(S13);
end

    Process2
begin
    down(S12);
    read(buf1, y);
    write(buf2, y);
    up(S24);
end

    Process3
begin
    down(S13);
    read(buf1, y);
    write(buf3, y);
    up(S35);
end

    Process4
begin
    down(S24);
    read(buf2, y);
    sum1 = y + random(1...10);
    if (sum1 > sum2)
        then write(sum1);
    end

    Process5
begin
    down(S35);
    read(buf3, y);
    sum2 = y + random(1...10);
    if (sum2 > sum1)
        then write(sum2);
    end

coend
  
```

Ερώτημα Γ

<u>Διεργασία Δ0</u>	<u>Διεργασία Δ1</u>	<u>flag0</u>	<u>flag1</u>	<u>turn</u>
		FALSE	FALSE	0
flag0 = TRUE		TRUE	FALSE	0
Εκτελεί το εξωτερικό while		TRUE	FALSE	0
Εισέρχεται στο ΚΡΙΣ. ΤΜΗΜΑ		TRUE	FALSE	0
	flag1 = TRUE	TRUE	TRUE	0
	Εκτελεί το εξωτερικό while	TRUE	TRUE	0
	Εκτελεί το εσωτερικό while	TRUE	TRUE	0
flag0 = FALSE		FALSE	TRUE	0
	Εκτελεί το εσωτερικό while	FALSE	TRUE	0
flag0 = TRUE		TRUE	TRUE	0
Εκτελεί το εξωτερικό while		TRUE	TRUE	0
Εισέρχεται στο ΚΡΙΣ. ΤΜΗΜΑ		TRUE	TRUE	0
	turn = 1	TRUE	TRUE	1
	Εκτελεί το εξωτερικό while	TRUE	TRUE	1
	Εισέρχεται στο ΚΡΙΣ. ΤΜΗΜΑ	TRUE	TRUE	1

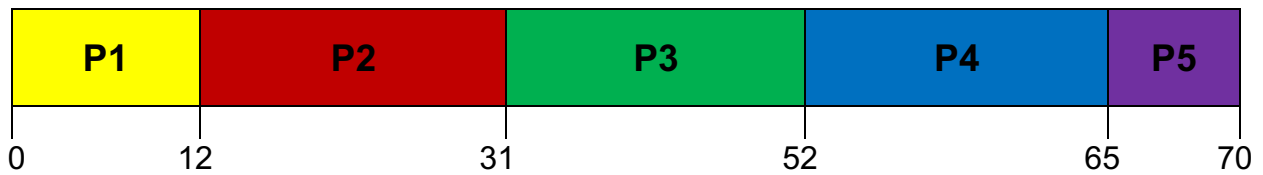
Ερώτημα Δ

Ο σημαφόρος `mutex` χρησιμοποιείται στο παραπάνω πρόβλημα συγχρονισμού έτσι ώστε να εξασφαλίζει πως ένας μόνο επιβλέπων μηχανικός, κάθε φορά, θα είναι υπεύθυνος για κάθε τρεις εργάτες. Ουσιαστικά, εξασφαλίζει αμοιβαίο αποκλεισμό μεταξύ διαφορετικών επιβλεπόντων. Αν δεν χρησιμοποιηθεί ο δυαδικός σημαφόρος `mutex`, τότε καταλήγουμε σε αδιέξοδο (deadlock).

Στη συνέχεια, παρουσιάζουμε ένα τέτοιο σενάριο κατά το οποίο δύο επιβλέποντες είναι υπεύθυνοι για τρεις εργάτες.

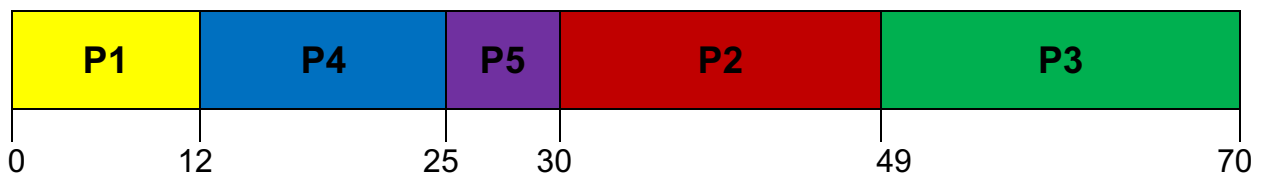
<u>Worker 1</u>	<u>Worker 2</u>	<u>Worker 3</u>	<u>Supervisor 1</u>	<u>Supervisor 2</u>
			signal(S)	signal(S)
				S = 2, W = 0
wait(S)				
wait(S)				S = 0, W = 0
			signal(S)	signal(S)
				S = 2, W = 0
	wait(S)			
	wait(S)			S = 0, W = 0
			signal(S)	signal(S)
				S = 2, W = 0
		wait(S)		
		wait(S)		S = 0, W = 0
signal(W)	signal(W)	signal(W)		S = 0, W = 3
			wait(W)	wait(W)
				S = 0, W = 1
			wait(W)	wait(W)
				S = 0, W = -1 Αδιέξοδο!
			wait(W)	wait(W)
				S = 0, W = -3 Αδιέξοδο!

Οι supervisors είναι μπλοκαρισμένοι επειδή δεν μπορούν να “τραβήξουν” κι άλλους εργάτες, συνεπώς οδηγούμαστε σε αδιέξοδο.

Ερώτημα Ε**α) FCFS (First Come First Serve)**

Διεργασία	Χρόνος Διεκπεραίωσης	Χρόνος Αναμονής
P1	12-0=12	12-12=0
P2	31-5=26	26-19=7
P3	52-8=44	44-21=23
P4	65-11=54	54-13=41
P5	70-15=55	55-5=50

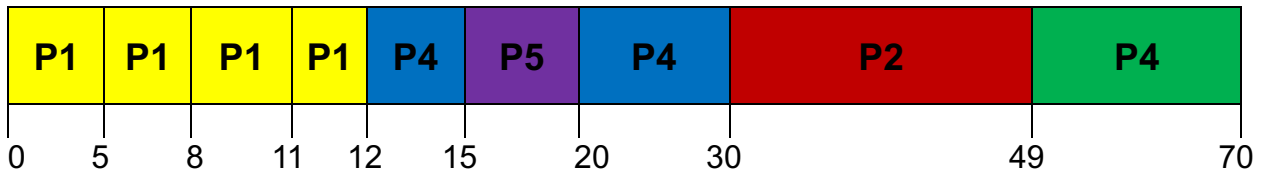
- $MX\Delta = (12 + 26 + 44 + 54 + 55) / 5 = 38,2 \text{ ms}$
- $MXA = (0 + 7 + 23 + 41 + 50) / 5 = 24,2 \text{ ms}$

β) SJF (Shortest Job First)

Διεργασία	Χρόνος Διεκπεραίωσης	Χρόνος Αναμονής
P1	12-0=12	12-12=0
P2	49-5=44	44-19=25
P3	70-8=62	62-21=41
P4	25-11=14	14-13=1
P5	30-15=15	15-5=10

- $MX\Delta = (12 + 44 + 62 + 14 + 15) / 5 = 29,4 \text{ ms}$
- $MXA = (0 + 25 + 41 + 1 + 10) / 5 = 15,4 \text{ ms}$

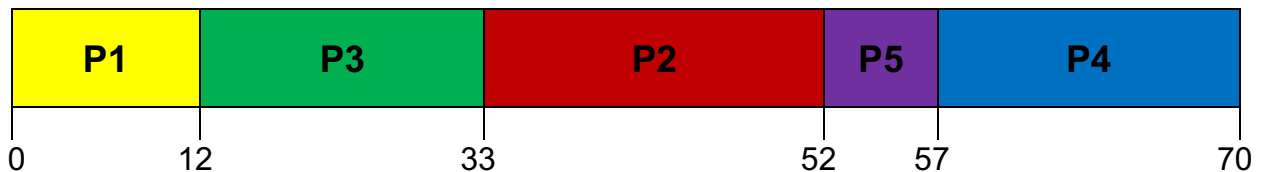
γ) SRTF (Shortest Remaining Time First)



Διεργασία	Χρόνος Διεκπεραίωσης	Χρόνος Αναμονής
P1	12-0=12	12-12=0
P2	49-5=44	44-19=25
P3	70-8=62	62-21=41
P4	30-11=19	19-13=6
P5	20-15=5	5-5=0

- $MX\Delta = (12 + 44 + 62 + 19 + 5) / 5 = 28,4 \text{ ms}$
- $MXA = (0 + 25 + 41 + 6 + 0) / 5 = 14,4 \text{ ms}$

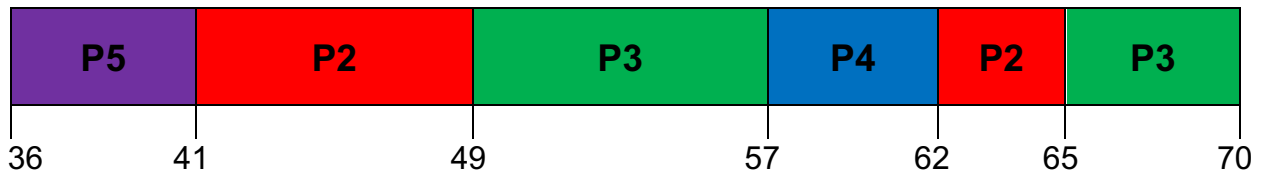
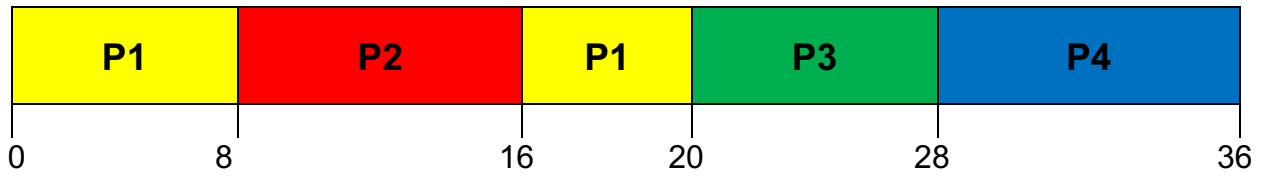
δ) Preemptive Priority Scheduling



Διεργασία	Χρόνος Διεκπεραίωσης	Χρόνος Αναμονής
P1	12-0=12	12-12=0
P2	52-5=47	47-19=28
P3	33-8=25	25-21=4
P4	70-11=59	59-13=46
P5	57-15=42	42-5=37

- $MX\Delta = (12 + 47 + 25 + 59 + 42) / 5 = 37 \text{ ms}$
- $MXA = (0 + 28 + 4 + 46 + 37) / 5 = 23 \text{ ms}$

ε) RR (Round Robin) με κβάντο χρόνου 8 ms



Διεργασία	Χρόνος Διεκπεραίωσης	Χρόνος Αναμονής
P1	20-0=20	20-12=8
P2	65-5=60	60-19=41
P3	70-8=62	62-21=41
P4	62-11=51	51-13=38
P5	41-15=26	26-5=21

- $MX\Delta = (20 + 60 + 62 + 51 + 26) / 5 = 43,8 \text{ ms}$
- $MXA = (8 + 41 + 41 + 38 + 21) / 5 = 29,8 \text{ ms}$

Στοιχεία ομάδας:

Αλέξανδρος Ξιάρχος
Νικηφόρος – Γεώργιος Παπαγεωργίου
Παναγιώτης Συριόπουλος

1059619
1059633
1059664

st1059619@ceid.upatras.gr
st1059633@ceid.upatras.gr
st1059664@ceid.upatras.gr