Λειτουργικά Συστήματα

2019 - 2020

1η Εργαστηριακή Άσκηση

Άγκο Μπεσιάνα
ΑΜ: 1059658

Ζεκυριά Αθανασία
ΑΜ: 1059660

Περιεχόμενα

Μέρος 1

Ερώτημα Α: Εξηγήστε προσεκτικά τι κάνει το παρακάτω πρόγραμμα. Πόσα μηνύματα εκτυπώνονται συνολικά;

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
      int i;
      int pid;

      pid = fork();

      for (i=1; i<=200; i++)
          if (pid > 0)
              printf("%3i (parent)\n", i);
           else
              printf("%3i (child)\n", i);


      return (0);
}
```

Απάντηση: Το πρόγραμμα εκτελεί την εντολή fork 200 φορές και εμφανίζει 400 μηνύματα: 200 γονείς + 200 παιδία, καθώς η fork δημιουργεί έναν γονέα και ένα παιδί ανά εκτέλεση της.

Ερώτημα Β: Δημιουργήστε ένα πρόγραμμα στο οποίο μία διεργασία στο Linux/Unix παράγει άλλες 10 θυγατρικές της.

Απάντηση:

```
#include<stdio.h>
int main()
{
  for(int i=0;i<10;i++)
   {
      if(fork() == 0)
      {
     printf("[son] pid %d from [parent] pid %d\n",getpid(),getppid());
         exit(0);
      }
   }
   for(int i=0;i<10;i++)
   wait(NULL);
}
```

Ερώτημα Γ: Να δημιουργήσετε ένα πρόγραμμα στο οποίο να δημιουργούνται 10 διεργασίες (αλυσίδα – κάθε μία να είναι παιδί της άλλης). Κάθε διεργασία να τυπώνει, το id του πατέρα της, το id της και το id του μοναδικού παιδιού που δημιουργεί.

Απάντηση:

```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/wait.h>
#include <signal.h>
#include <unistd.h>

int n=10;
int passId[2];
pipe(passId);
pid_t cpid,ppid,mpid;

int print() {


   printf( " My parent id is:%d , My pid is:%d, My child id is:%d \n " ,   ppid ,mpid,cpid);
   return 1;
}

int func(int n)
{

   if (n == 0)
   {
      return 0;
   }
```

```
    int pid = fork();
    if (pid == -1) {
        exit(0);
    }
    if (pid==0) {
        mpid = getppid();
        cpid = getpid();
        print();
        ppid=getppid();

        n = n-1;
        func(n);
        exit(0);
    }
    else {

      wait(NULL);
    }
    return 0;
}

int main(int argc , char *argv[])
{
    func(n);
    return 0;
}
```

Ερώτημα Δ: Να δημιουργήσετε πρόγραμμα στο οποίο:

1. Θα ορίσετε μια συνάρτηση foo() η οποία θα ορίζει μια μεταβλητή int x=0 και θα κάνει την πράξη x=x+10. Αυτή η συνάρτηση δεν κάνει τίποτα χρήσιμο, αλλά απλώς υπάρχει για να μας βοηθήσει στη μέτρηση.

2. Μέσα στη main() θα εκτελεί μια φορά τη συνάρτηση time() με τις κατάλληλες παραμέτρους, θα αποθηκεύεται ο αριθμός των δευτερολέπτων στη μεταβλητή start και αμέσως μετά θα εκτυπώνεται το μήνυμα "Αρχική τιμή δευτερολέπτων" ακολουθούμενο από τη start.

3. Στη συνέχεια θα υπάρχει ένας βρόχος while() ο οποίος θα δημιουργεί 100 διεργασίες (δοκιμάστε και για 5000, 10000) με τη fork() οι οποίες όλες θα εκτελούν τη foo(). Προσοχή, ο πατέρας θα δημιουργήσει όλες τις διεργασίες και όχι η κάθε διεργασία θα δημιουργεί άλλη διεργασία.

4. Μόλις δημιουργηθούν οι 100 διεργασίες και μόνο τότε θα εκτελεί ο πατέρας 100 φορές τη waitpid() ώστε να περιμένει την επιτυχή ολοκλήρωση όλων των θυγατρικών.

5. Στη συνέχεια θα καλείται η time(), θα αποθηκεύεται ο αριθμός των δευτερολέπτων στη μεταβλητή end και αμέσως μετά θα εκτυπώνεται το μήνυμα "Τελική τιμή δευτερολέπτων" ακολουθούμενη από την end. Θα γίνεται η πράξη end-start, θα εκτυπώνεται το αποτέλεσμα ενώ θα εκτυπώνεται και το αποτέλεσμα «(end-start)/100» για να εμφανιστεί ο μέσος χρόνος δημιουργίας, εκτέλεσης και τερματισμού των 100 διεργασιών.

Πόσος είναι ο συνολικός χρόνος και ο μέσος χρόνος δημιουργίας, εκτέλεσης και τερματισμού των 100 διεργασιών στο σύστημα σας (μη ξεχάσετε τη μονάδα μέτρησης);

Απάντηση:

```
#include <stdio.h>
#include <time.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>

void foo();


int main()
```

```
{
   //time_t start; erwthma d me time()
   int child_s;
   clock_t start,end,result;

   //time(&start);
   start = clock();
   printf("Αρχική τιμή δευτερολέπτων = %ld\n", start);

   int i=0;
   while(i<100)
   {
      if(fork() == 0)
      {
         foo();
       //printf("[son] pid %d from [parent] pid %d\n",getpid(),getppid());
         exit(0);
      }
      i++;
   }

  for(int i=0;i<100;i++)
   {
    wait(NULL);
   }


   if(i==99)
   {
    for(int j=0;j<100;j++)
    {
```

```
      pid_t waitp= waitpid(-1,&child_s,0);
     }
   }

  //  time_t end;


   //time(&end);
   end = clock();
   printf("Τελική τιμή δευτερολέπτων = %ld\n", end);
   //time_t result;

   result=end-start;
   printf("%ld\n",result);
   result=result/100;
   printf("%ld ",result);

}
void foo()
 {

 int x=0;
  x=x+10;

}
```

Μέρος 2

Ερώτημα Α: Θεωρήστε τις διεργασίες Process1, Process2 και Process3 που εκτελούνται «παράλληλα» (δηλ. εκτελούνται σύμφωνα με το σχήμα εκτέλεσης “cobegin Process1; Process2; Process3; coend”). Κάθε διεργασία εκτελεί επαναληπτικά (για 10 φορές) δύο εντολές. Συγκεκριμένα:

• Η διεργασία Process1 εκτελεί επαναληπτικά για 10 φορές την εντολή Ε1.1 και την εντολή Ε1.2.

• Η διεργασία Process2 εκτελεί επαναληπτικά για 10 φορές την εντολή Ε2.1 και την εντολή Ε2.2.

• Η διεργασία Process3 εκτελεί επαναληπτικά για 10 φορές την εντολή Ε3.1 και την εντολή Ε3.2.

Ο κώδικας των διεργασιών δίνεται ακολούθως:

```
     Process1              Process2              Process3
for k = 1 to 10 do    for j = 1 to 10 do    for l = 1 to 10 do
  begin                 begin                 begin
     E1.1;                 E2.1;                 E3.1;
     E1.2;                 E2.2;                 E3.2;
  end                   end                   end
```

Καλείστε να συμπληρώσετε τον παραπάνω κώδικα των τριών διεργασιών με εντολές P (ή wait ή down) και V (ή signal ή up) σε σημαφόρους (που πρέπει να αρχικοποιήσετε κατάλληλα), προκειμένου να εξασφαλίζονται οι ακόλουθες συνθήκες συγχρονισμού για τη σειρά εκτέλεσης των εντολών των διεργασιών:

1. Στην i επανάληψη (1<=i<=10) της Process2, η εκτέλεση της Ε2.1 να έπεται της εκτέλεσης της Ε1.1 στην ίδια επανάληψη i της Process1. Π.χ. στην 5η επανάληψη της Process2, η Ε2.1 να εκτελείται μετά από την Ε1.1 στην 5η επανάληψη της Process1.

2. Στην i επανάληψη (1<=i<=10) της Process3, η εκτέλεση της Ε3.1 να έπεται της εκτέλεσης της Ε2.1 στην ίδια επανάληψη i της Process2. Π.χ. στην 5η επανάληψη της Process3, η Ε3.1 να εκτελείται μετά από την Ε2.1 στην 5η επανάληψη της Process2.

3. Στην i επανάληψη (1<=i<=10) της Process2, η εκτέλεση της Ε2.2 να έπεται της εκτέλεσης της Ε1.2 στην ίδια επανάληψη i της Process1. Π.χ. στην 5η επανάληψη της Process2, η Ε2.2 να εκτελείται μετά από την Ε1.2 στην 5 η επανάληψη της Process1. Πολυτεχνική Σχολή Τμήμα Μηχανικών Ηλεκτρονικών Υπολογιστών & Πληροφορικής 3

4. Στην i επανάληψη (1<=i<=10) της Process3, η εκτέλεση της Ε3.2 να έπεται της εκτέλεσης της Ε2.2 στην ίδια επανάληψη i της Process2. Π.χ. στην 5η επανάληψη της Process3, η Ε3.2 να εκτελείται μετά από την Ε2.2 στην 5 η επανάληψη της Process2.

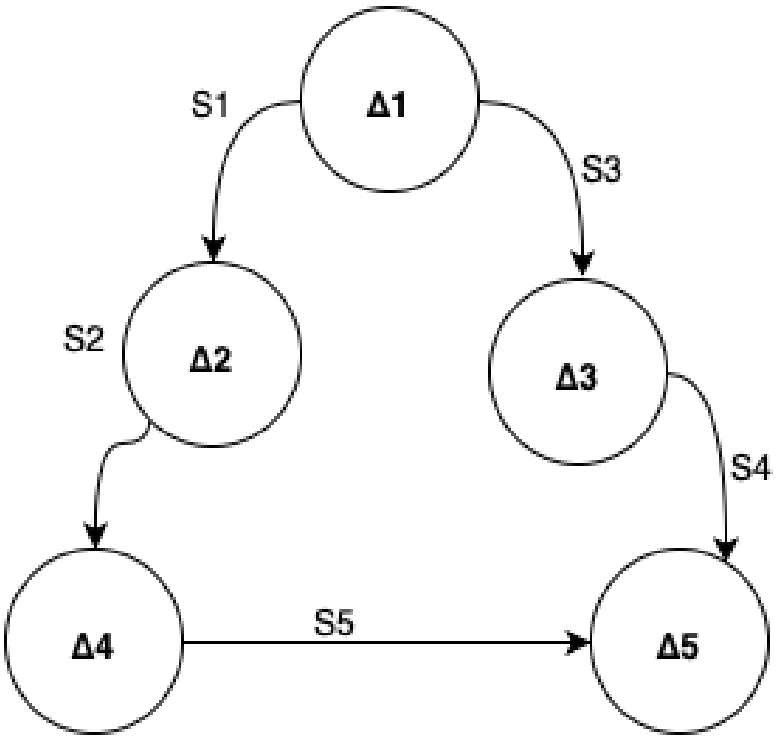
5. Στην i επανάληψη (2<=i<=10) της Process1, η εκτέλεση της Ε1.1 να έπεται της εκτέλεσης της Ε3.2 στην προηγούμενη επανάληψη (i-1) της Process3. Π.χ. στην 5η επανάληψη της Process1, η Ε1.1 να εκτελείται μετά από την Ε3.2 στην 4 η επανάληψη της Process3.

(i) Δώστε αρχικά μία λύση χρησιμοποιώντας πέντε (5) σημαφόρους (έναν για κάθε συνθήκη συγχρονισμού που αναφέρεται παραπάνω).

Απάντηση:

```
Var

S1,s2,s3,s4,s5: semaphores;

s1=s3=s4=s5=0;

s2=1;


P1:

For k=1 to 10 do

Begin

Wait(s5);

E1.1;

Signal(s1);

Wait(s2);

E2.1;

Signal(s3);

End


P2:

For j=1 to 10 do

Begin
```

Wait(s1);



E2.1;

Signal(s2);

Wait(s3);

E2.2;

Signal(s4);

End

P3:

For l=1 to 10 do

Begin

Wait(s2);

E3.1;

Signal(s2);

Wait(s4);

E3.1;

Signal(s5);

End

(ii) Στη συνέχεια δώστε μια λύση προσπαθώντας να χρησιμοποιήσετε τον ελάχιστο (κατά την κρίση σας) δυνατό αριθμό σημαφόρων που αρκεί για να επιτευχθούν οι προαναφερόμενες συνθήκες συγχρονισμού.

Απάντηση:

(ii):

Var

S1,s2,s3: semaphores;

s1=s2=s3=0;

P1:

For k=1 to 10 do

Begin

Wait(s3);

E1.1;

Signal(s1);

Wait(s2);

E2.1;

Signal(s3);

End

P2:

For j=1 to 10 do

Begin

Wait(s1);

E1.1;

Signal(s2);

Wait(s3);

E2.1;

Signal(s1);

End

P3:

For l=1 to 10 do

Begin

Wait(s2);

E1.1;

Signal(s2);

Wait(s1);

E2.1;

Signal(s3);

End

Ερώτημα Β: Ένα λειτουργικό σύστημα εξυπηρετεί 5 παράλληλες διεργασίες Δi (i = 1..5). Οι 5 διεργασίες εκτελούνται συνεχώς (δηλ. ο κώδικας της κάθε μιας βρίσκεται εντός ενός αδιάκοπου βρόγχου). Το λειτουργικό σύστημα προσφέρει τη χρήση σημαφόρων. Στη συνέχεια δίνεται η περιγραφή των 5 διεργασιών.

Διεργασία Δ1: Η διεργασία Δ1 βρίσκει έναν τυχαίο αριθμό από το διάστημα 1..10 και τον γράφει στον απομονωτή (buffer) buf1.

Διεργασία Δ2: Η διεργασία Δ2 διαβάζει από τον απομονωτή buf1 την τιμή που έχει γράψει στον ίδιο κύκλο επανάληψης, η διεργασία Δ1, και γράφει την τιμή που διάβασε στον απομονωτή buf2.

Διεργασία Δ3: Η διεργασία Δ3 διαβάζει από τον απομονωτή buf1 την τιμή που έχει γράψει στον ίδιο κύκλο επανάληψης, η διεργασία Δ1, και γράφει την τιμή που διάβασε στον απομονωτή buf3.

Διεργασία Δ4: Η διεργασία Δ4 διαβάζει από τον απομονωτή buf2 την τιμή που έχει γράψει στον ίδιο κύκλο επανάληψης, η διεργασία Δ2. Στη συνέχεια βρίσκει μια τυχαία τιμή από το διάστημα 1..10 και την προσθέτει στην τιμή που διάβασε από τον απομονωτή buf2. Αν το αποτέλεσμα είναι μεγαλύτερο του αντίστοιχου αποτελέσματος (στον ίδιο κύκλο επανάληψης) της διεργασίας Δ5, τότε εκτυπώνει στη οθόνη το όνομά της.

Διεργασία Δ5: Η διεργασία Δ5 διαβάζει από τον απομονωτή buf3 την τιμή που έχει γράψει στον ίδιο κύκλο επανάληψης, η διεργασία Δ3. Στη συνέχεια βρίσκει μια τυχαία τιμή από το διάστημα 1..10 και την προσθέτει στην τιμή που διάβασε από τον απομονωτή buf3. Αν το αποτέλεσμα είναι μεγαλύτερο ή ίσο του αντίστοιχου αποτελέσματος (στον ίδιο κύκλο επανάληψης) της διεργασίας Δ4, τότε εκτυπώνει στη οθόνη το όνομά της.

Απάντηση:

S1=d12

S2=d24

S3=d13

S4=d35

S5=d45

Ερώτημα Β.1: Τεκμηριώστε τον απαιτούμενο συγχρονισμό και προσδιορίστε τη σειρά εκτέλεσης των διεργασιών. Δώστε το γράφο προτεραιότητας (για έναν κύκλο επανάληψης) για τις διεργασίες Δ1, Δ2, Δ3, Δ4, Δ5.

Cobegin

Δ1;

Begin

Δ2;Δ3;

End

Begin

Δ4;Δ5;

End

Coend

Ερώτημα Β.2: Δώστε τον κώδικα των 5 παράλληλων διεργασιών. Ο κώδικας θα πρέπει να υλοποιεί τις διεργασίες όπως αυτές περιγράφονται ανωτέρω και να συμπεριλαμβάνει τις απαραίτητες εντολές συγχρονισμού με σημαφόρους.

```
Var

b: integer;

a,d,c: shared integer;

S1,s2,s3,s4,s5,s6:semaphores;

Begin

a:=0;b:=0;c:=0;d:=0;

s1=s2=s3=s4=0;


Cobegin
```

Process Δ1:

```
repeat

a=random(1,10);

write(buf1,a);

signal(s1);

signal(s3);

Forever;
```

Process Δ2:

```
Repeat

wait(s1);

read(buf1,a);

write(buf3,a);
```

signal(s2);

Forever;

Process Δ3:

Repeat

wait(s3);

read(buf2,a);

write(buf3,a);

signal(s4);

Forever;

Process Δ4:

Repeat

wait(s2);

read(buf2,a);

b=random(1,10);

c=a+b;

Wait(s5);

If (c>d)

write(process Δ5);

Else

signal(s6);

Forever;

Process Δ5:

Repeat

read(buf3,a);

d=random(1,10);

d=d+a;

wait(s6);

If (d>=c)

write(process Δ4);

Else

signal(s5);

Forever;

Coend

End

Ερώτημα Γ: Θεωρείστε τον ακόλουθο κώδικα για δύο διεργασίες Δ0 και Δ1 που εκτελούνται παράλληλα. Στον κώδικα των διεργασιών χρησιμοποιούνται οι εξής κοινά διαμοιραζόμενες μεταβλητές:οι λογικές μεταβλητές flag0 και flag1, που η καθεμία αρχικοποιείται στην τιμή FALSE (ψευδής), και η ακέραια μεταβλητή turn, με αρχική τιμή 0.

```
shared boolean flag0 = flag1 = FALSE;
shared integer turn = 0;
```

Διεργασία Δ0

```
repeat
   flag0 =  TRUE;
   while (turn != 0) {
       while (flag1 == TRUE) do noop;
       turn = 0;
   }
   <ΚΡΙΣΙΜΟ ΤΜΗΜΑ>
   flag0 = FALSE;
forever
```

Διεργασία Δ1

```
repeat
   flag1 = TRUE
   while (turn != 1) {
       while (flag0 == TRUE) do noop;
       turn = 1;
   }
   <ΚΡΙΣΙΜΟ ΤΜΗΜΑ>
   flag1 = FALSE;
forever
```

Ο παραπάνω κώδικας δεν εξασφαλίζει τον αμοιβαίο αποκλεισμό των διεργασιών Δ0 και Δ1, αφού υπάρχει περίπτωση οι δύο διεργασίες να εκτελούν ταυτόχρονα το τμήμα εντολών . Ζητείται να προσδιορίσετε ένα σενάριο εκτέλεσης των εντολών των δύο διεργασιών που οδηγεί σε αυτή την περίπτωση. Για την περιγραφή του σεναρίου, καλείστε να συμπληρώσετε πιο συγκεκριμένα τον ακόλουθο ημιτελή πίνακα, παρουσιάζοντας την εκτέλεση των εντολών των δύο διεργασιών που οδηγεί σε παραβίαση του αμοιβαίου αποκλεισμού.

Απάντηση:

Διεργασία 0	**Διεργασία 1**	**Flag 0**	**Flag 1**	**Turn**
		False	False	0
Flag0=TRUE		True	False	0
Εκτελεί το εξωτερικό while		True	False	0
Εισέρχεται στο κρίσιμο τμήμα		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 σε κρίσιμο τμήμα		True	True	0
	Εκτελεί το εξωτερικό while	True	True	1
	Εισέρχεται σε κρίσιμο τμήμα	True	True	1

Το ζητούμενο είναι ο χρονοδρομολογητής να διακόψει την διεργασία 1 προτού μεταβληθεί η τιμή της turn σε 1. Αν ο χρονοδρομολογητής σταματήσει τη διεργασία 0 κατά την εκτέλεση του κρίσιμου τμήματός της και επιτρέψει στην διεργασία 1 να εκτελεστεί, αυτή θα μεταβάλλει την turn σε 1, το while σε false και θα εισαχθεί στο κρίσιμο τμήμα της.

Ερώτημα Δ: Στο χώρο ενός εργοταξίου ισχύει ένας συγκεκριμένος κανονισμός λειτουργίας που επιβάλλει να είναι παρών ένας επιβλέπων μηχανικός για κάθε τρεις εργάτες. Κάθε επιβλέπων μηχανικός που προσέρχεται στο εργοτάξιο προσομοιώνεται από τη διεργασία Supervisor, ενώ κάθε εργάτης που προσέρχεται στο εργοτάξιο προσομοιώνεται από τη διεργασία Worker.

Ο κώδικας των διεργασιών δίνεται ακολούθως και για να επιτευχθεί ο απαραίτητος συγχρονισμός (δηλ. για να ισχύει πάντοτε ο κανονισμός λειτουργίας που προαναφέρθηκε) χρησιμοποιούνται σημαφόροι (που αρχικοποιούνται σε συγκεκριμένες τιμές) και αντίστοιχες εντολές signal / wait επί αυτών των σημαφόρων.

```
semaphore S = W = 0;
binary semaphore mutex = 1;
```

Διεργασία Supervisor	**Διεργασία Worker**
<Προσέλευση Επιβλέποντα στο Εργοτάξιο>	<Προσέλευση Εργάτη στο Εργοτάξιο>
signal(S);	wait(S);
signal(S);	<Εργασία στο Εργοτάξιο>
signal(S);	signal(W);
<Επίβλεψη Εργατών>	<Αποχώρηση Εργάτη από το Εργοτάξιο>
wait(mutex);	
wait(W);	
wait(W);	
wait(W);	
signal(mutex);	
<Αποχώρηση Επιβλέποντα από το Εργοτάξιο>	

Αφού μελετήσετε τον κώδικα και κατανοήσετε το ρόλο των σημαφόρων και των εντολών signal/wait στην επίτευξη του απαραίτητου συγχρονισμού, καλείστε να εντοπίσετε/προσδιορίσετε ποιο πρόβλημα μπορεί να συμβεί εάν δεν χρησιμοποιηθεί ο δυαδικός σημαφόρος mutex και διαγραφούν οι αντίστοιχες εντολές (wait(mutex) και signal(mutex)) από τον κώδικα της διεργασίας Supervisor. Για περαιτέρω τεκμηρίωση, να δώσετε ένα σενάριο εκτέλεσης των εντολών των διεργασιών που να καταλήγει στο πρόβλημα που εντοπίσατε.

Απάντηση:

wait: αποκλείει τη διαδικασία κλήσης έως ότου τερματιστεί μία από τις θυγατρικές διεργασίες ή ληφθεί σήμα. Μετά την ολοκλήρωση της διαδικασίας του παιδιού, ο γονέας συνεχίζει την εκτέλεση μετά την εντολή κλήσης συστήματος αναμονής.

signal: επιστρέφει την προηγούμενη τιμή του χειριστή σήματος ή SIG_ERR αν υπάρξει σφάλμα. Σε περίπτωση σφάλματος, το errno είναι ρυθμισμένο να υποδεικνύει την αιτία.

Πιο συγκεκριμένα στον κώδικα μας με τον δυαδικό σημαφόρο mutex:

wait(mutex): επιτρέπει μόνο σε έναν εργαζόμενο να περάσει

signal(mutex): δίνει σήμα ένας εργαζόμενος να περάσει

Αν δεν χρησιμοποιηθεί το mutex και διαγραφούν οι παραπάνω εντολές, δεν θα υπάρχει περιορισμός ενός επιβλέποντα στη wait και δεν θα αποχωρήσει ο επιβλέπων. Η λύση για το πρόβλημά μας θα είναι λάθος σε περίπτωση που λείψουν και τα δύο γιατί κατ'αυτόν τον τρόπο, δεν περιορίζεται οι είσοδος 3 εργαζομένων.

Ερώτημα Ε: Δίνεται το παρακάτω σύνολο διεργασιών οι οποίες καταφθάνουν για να εκτελεστούν σε ένα υπολογιστικό σύστημα:

Διεργασία	Χρόνος Άφιξης	Χρόνος Εκτέλεσης	Προτεραιότητα
P1	0	12	3
P2	5	19	3
P3	8	21	5
P4	11	13	2
P5	15	5	3

Σχεδιάζοντας τα κατάλληλα διαγράμματα Gantt δείξτε πώς θα εκτελεστούν οι διεργασίες αυτές στην Κεντρική Μονάδα Επεξεργασίας (ΚΜΕ), και υπολογίστε τους μέσους χρόνους διεκπεραίωσης (ΜΧΔ) και αναμονής (ΜΧΑ), για κάθε έναν από τους παρακάτω αλγόριθμους χρονοδρομολόγησης. Θεωρήστε ότι ο χρόνος εναλλαγής (context switch) είναι αμελητέος και ότι μεγάλες τιμές προτεραιότητας σηματοδοτούν μεγαλύτερες προτεραιότητες.

α) FCFS (First Come First Served)

β) SJF (Shortest Job First)

γ) SRTF (Shortest Remaining Time First)

δ) Προεκχωρητικός Προτεραιότητας (Preemptive Priority Scheduling) [με χρήση αλγορίθμου FCFS (First Come First Served) στις περιπτώσεις που έχουμε ίσες προτεραιότητες]

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

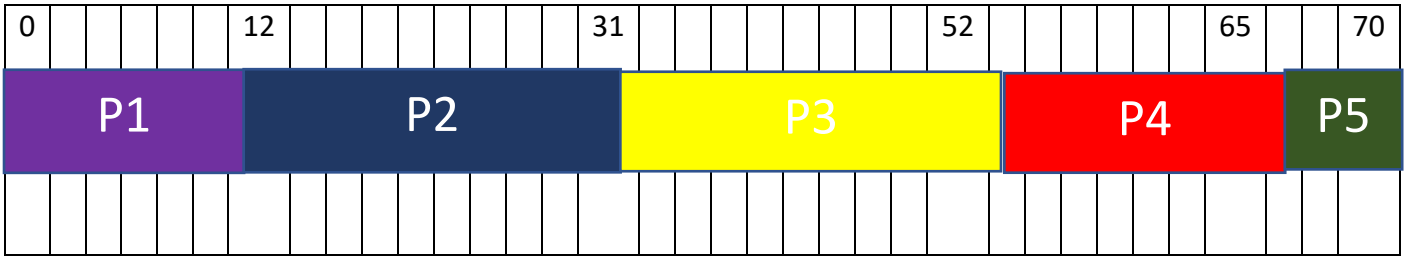
Απάντηση:

Ας δώσουμε συγκεκριμένα χρώματα σε κάθε διεργασία:
P1 P2 P3 P4 P5

α) FCFS (First Come First Served): είναι ένας αλγόριθμος χρονοπρογραμματισμού ο οποίος έχει ως καθήκον να επιλέγει με ποια σειρά θα εξυπηρετηθούν οι αιτήσεις προς ένα σκληρό δίσκο που αφορούν

εγγραφή και ανάγνωση σε αυτόν. Ο αλγόριθμος αυτός είναι ο πιο δίκαιος αλγόριθμος αλλά γενικά δεν είναι και ο γρηγορότερος καθώς η κεφαλή του δίσκου μπορεί να κινηθεί από την μια του άκρη στην άλλη για να εξυπηρετήσει μια αίτηση αγνοώντας αιτήσεις κοντινότερες προς αυτή.

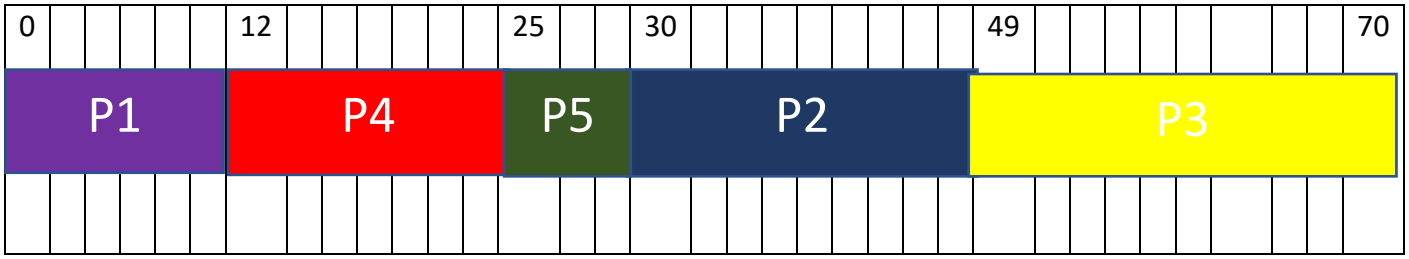
Στη συγκεκριμένη περίπτωση ισχύει:



ΜΧΑ: (0+7+23+41+50)/5=24,2 msec

ΜΧΔ: (12+26+44+54+55)/5 =38,2 msec

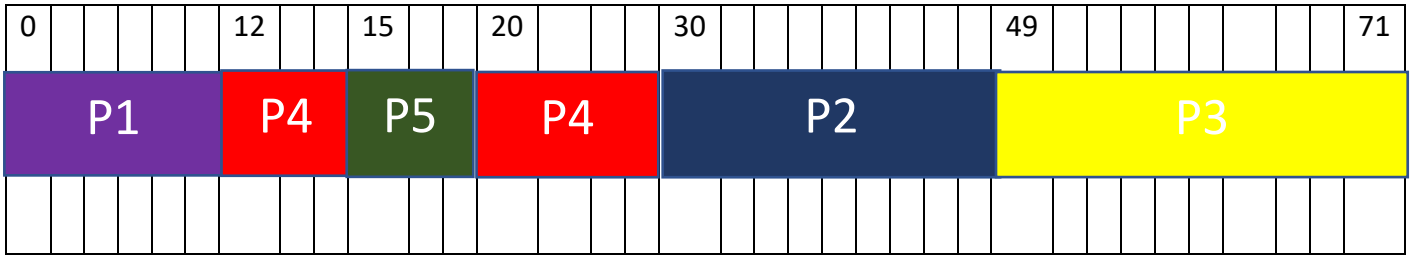
β)



ΜΧΑ: (25+41+1+10+0)/5=15,4 msec

ΜΧΔ: (25+14+62+44+12)/5=31,4msec

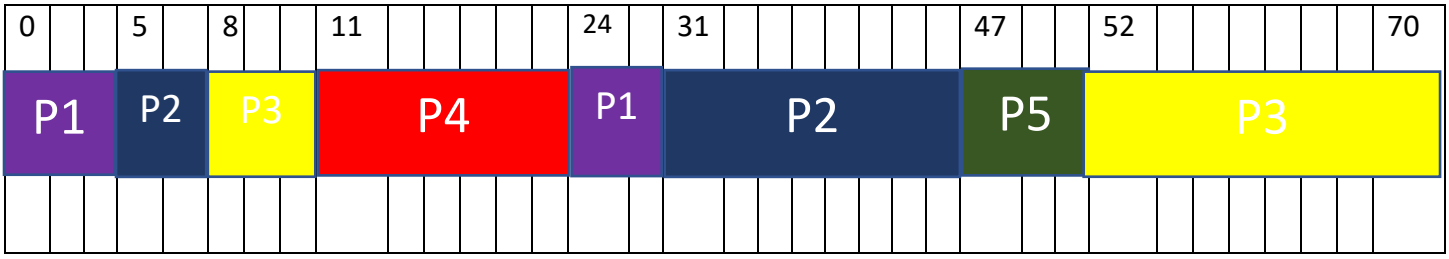
γ)



ΜΧΑ: (25+0+41+6+0)/5=14,4 msec

ΜΧΔ: (12+44+63+19+9)/5=29,4 msec

δ)



ΜΧΑ: (41+32+23+19+0)/5=23 msec

ΜΧΔ: (70+52+47+31+24)/5=44,8 msec