# Λειτουργικά Συστήματα 1η Άσκηση


## Χειμερινό Εξάμηνο 2019-2020

**Σταθόπουλος Ιωάννης |** 1043823 (6353) 6ο
**Τσαρκάτογλου Τριαντάφυλλος |** 1043836 (6367) 6ο
**Σπανού Άννα-Μαρία |** 1041884 (236207) 6ο

## Μέρος 1 [30 μονάδες]

### Ερώτημα Α [5]:

Εξηγήστε προσεκτικά τι κάνει το παρακάτω πρόγραμμα. Πόσα μυνήματα 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);
}
```

Το πρόγραμμα ξεκινάει ορίζοντας δυο μεταβλητές ακέραιων αριθμών (integer) με ονομα “I” και “pid”. Η fork() όταν εκτελείται δημιουργεί ένα ακριβές αντίγραφο της τρέχουσας διεργασίας το οποίο εκτελείται παράλληλα με την αρχική διεργασία. Η fork() επιστρέφει την τιμή 0 στην διεργασία παιδί και έναν αριθμό διάφορο του μηδενός στην διεργασία πατέρα.
Στον παραπάνω κώδικα, η fork() εκτελείται μία φορά, και δημιουργεί μία διεργασία «πατέρα» (με pid ίδιο με της αρχικής διεργασίας) και μια διεργασία «παιδί» με pid=0. Στην συνέχεια η for για 200 επαναλήψεις θα εκτελεστεί «παράλληλα» και για το παιδί και για τον πατέρα την εκτύπωση του μηνύματος του εκάστοτε αριθμού. Δηλαδή, απο 1 έως 200 (parent) και απο 1 έως 200 (child).
Επομένως εκτυπώνονται συνολικά 200+200=400 μηνύματα.

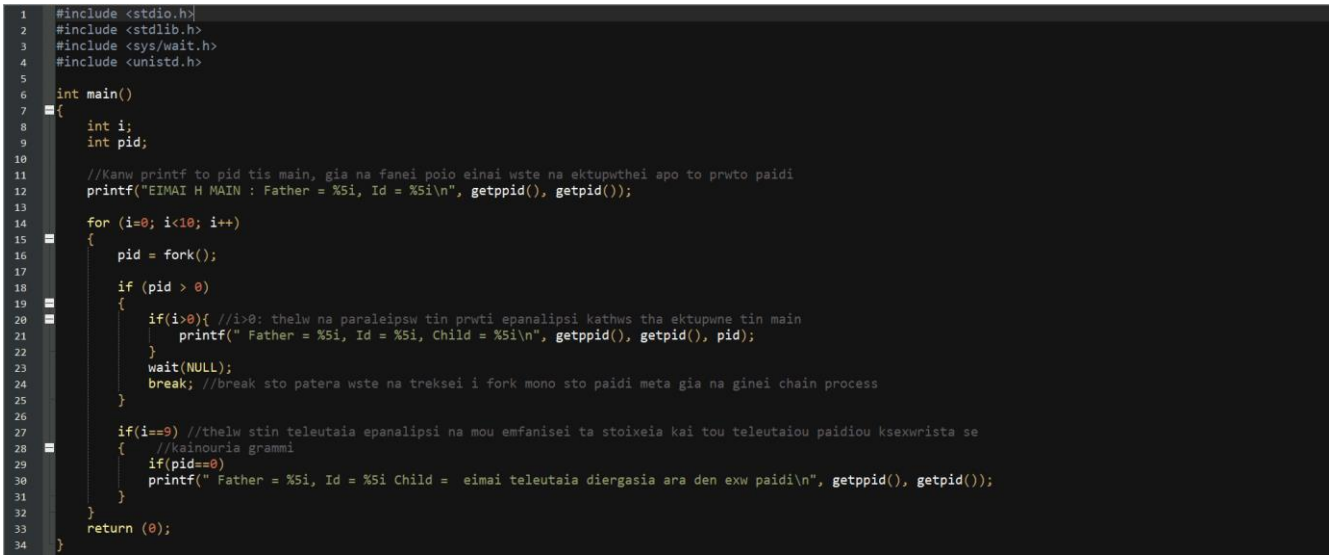
**Ερώτημα Β [5]:** Δημιουργείστε ένα πρόγραμμα στο οποίο μία διεργασία στο Linux/Unix παράγει άλλες 10 θυγατρικές της.

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

int main(void){

    int i, tmp;
    pid_t pid[10];

    for (i=0;i<10;i++) { //ftiaxnw 10 child processes

        pid[i] = fork();

    //kanw break se kathe child wste na min treksei tin fork meta kai oles oi diergasies na exoun ton idio patera
        if (pid[i] == 0) {
            break;
        }
    }

    if (pid[0] !=0 && pid[1] !=0 && pid[2] !=0 && pid[3] !=0 && pid[4] !=0 && pid[5] !=0 && pid[6] !=0 && pid[7] !=0 && pid[8] !=0 && pid[9] !=0 && pid[10] !=0)
    {

        // Autos einai o pateras kai perimenei gia ola ta paidia
        printf("I'm the father [pid: %d, my parent's pid: %d]\n", getpid(), getppid());

        for(i=0;i<10;i++) {
            wait(NULL);
        }

    } else {
        //To paidi ektupwnei to pid tou, kai to pid tou patera tou, to opoio einai koino se ola ta paidia
        printf("I'm one of the children [pid: %d, parent pid: %d]\n",getpid(),getppid());
    }
    return 0;
}
```

(αρχείο erB.c)

```
I'm the father [pid: 22773, my parent's pid: 22768]
I'm one of the children [pid: 22775, parent pid: 22773]
I'm one of the children [pid: 22781, parent pid: 22773]
I'm one of the children [pid: 22783, parent pid: 22773]
I'm one of the children [pid: 22782, parent pid: 22773]
I'm one of the children [pid: 22780, parent pid: 22773]
I'm one of the children [pid: 22779, parent pid: 22773]
I'm one of the children [pid: 22778, parent pid: 22773]
I'm one of the children [pid: 22777, parent pid: 22773]
I'm one of the children [pid: 22776, parent pid: 22773]
I'm one of the children [pid: 22774, parent pid: 22773]


...Program finished with exit code 0
Press ENTER to exit console.
```

**Ερώτημα Γ [5]:** 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.



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

int main()
{
    int i;
    int pid;

    //Kanw printf to pid tis main, gia na fanei poio einai wste na ektupwthei apo to prwto paidi
    printf("EIMAI H MAIN : Father = %5i, Id = %5i\n", getppid(), getpid());

    for (i=0; i<10; i++)
    {
        pid = fork();

        if (pid > 0)
        {
            if(i>0){ //i>0: thelw na paraleipsw tin prwti epanalipsi kathws tha ektupwne tin main
                printf(" Father = %5i, Id = %5i, Child = %5i\n", getppid(), getpid(), pid);
            }
            wait(NULL);
            break; //break sto patera wste na treksei i fork mono sto paidi meta gia na ginei chain process
        }

        if(i==9) //thelw stin teleutaia epanalipsi na mou emfanisei ta stoixeia kai tou teleutaiou paidiou ksexwrista se
        {    //kainouria grammi
            if(pid==0)
            printf(" Father = %5i, Id = %5i Child =  eimai teleutaia diergasia ara den exw paidi\n", getppid(), getpid());
        }
    }
    return (0);
}
```

(αρχείο erG.c)

```
EIMAI H MAIN : Father =  4657, Id =  4662
 Father =  4662, Id =  4663, Child =  4664
 Father =  4663, Id =  4664, Child =  4665
 Father =  4664, Id =  4665, Child =  4666
 Father =  4665, Id =  4666, Child =  4667
 Father =  4666, Id =  4667, Child =  4668
 Father =  4667, Id =  4668, Child =  4669
 Father =  4668, Id =  4669, Child =  4670
 Father =  4669, Id =  4670, Child =  4671
 Father =  4670, Id =  4671, Child =  4672
 Father =  4671, Id =  4672 Child =  eimai teleutaia diergasia ara den exw paidi


...Program finished with exit code 0
Press ENTER to exit console.
```

**Ερώτημα Δ [15]:** Ερωτήματα 1 έως 5.

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

void foo()  //orizetai i sunartisi foo
{
    int x=0;
    x=x+1;
}

time_t time(time_t *t);

int main(){

    int k, status, count=0, i=0;
    pid_t pid[1000];
    time_t start, end, diff;
    start = time(NULL);  //ekteleitai i time kai apothikeutai sti metabliti start
    float mo;
    printf("Arxiki timi deuteroleptwn %ld\n", start);

    while(i<1000)
    {
        pid[i] = fork();
        if(pid[i]==0){
            foo();
            exit(0);//o pateras dimiourgei ta paidia kai oxi i kathe paidi ena allo
        }
        i++;
    }

    for(k=0; k<1000; k++){
        if(pid[k]>0){
            waitpid (pid[k], &status, 0); //o pateras perimenei kathe paidi ksexwrista
        }
    }

    end = time(NULL); //mesw tis time() pairnei timi end
    printf("Teliki timi deuteroleptwn %ld\n", end);
    diff = end-start;
    printf("diff=%ld\n", diff );
    mo=(diff/1000.0); //upologizw ton meso oro
    printf("mesos oros %lf\n", mo);
    return 0;
}
//Gia 100 diergasies otan trexei to programma den parathreitai kapoia diafora stis times twn start, end opote i diafora tous
//einai 0. Epomenws trexoume to programma gia 1000 diergasies opou paratireitai diafora 1, ara mesos oros mo=0,001
```

(αρχείο erD.c)

```
Arxiki timi deuteroleptwn 1574268635
Teliki timi deuteroleptwn 1574268635
diff=0
mesos oros 0.000000


...Program finished with exit code 0
Press ENTER to exit console.
```

# Μέρος 2 [70 μονάδες]:

## Ερώτημα Ε:

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

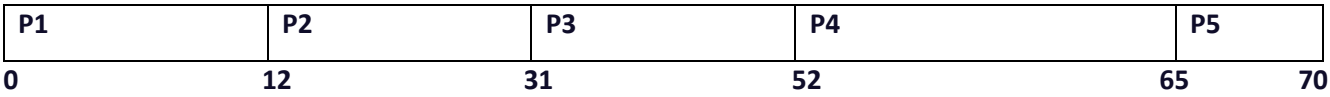
| Διεργασία | Χρόνος Άφιξης | Χρόνος Εκτέλεσης | Προτεραιότητα |
|---|---|---|---|
| P1 | 0 | 12 | 3 |
| P2 | 5 | 19 | 3 |
| P3 | 8 | 21 | 5 |
| P4 | 11 | 13 | 2 |
| P5 | 15 | 5 | 3 |

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

- FCFS (First Come First Served).
- SJF (Shortest Job First).
- SRTF (Shortest Remaining Time First).
- Priority Scheduling – μή προεκχωρητικός (non-preemptive priority).
- ( Round Robin) με κβάντο χρόνου 4 χρονικές μονάδες.

**Απάντηση**:

**FCFS (FIRST COME FIRST SERVED):**




Μ.Χ.Α. :

- ΧΑ(P1)= 0
- ΧΑ(P2)= 12-5 = 7
- ΧΑ(P3)= 31-8 = 23
- ΧΑ(P4)= 52-11 = 41
- ΧΑ(P5)=65-15 = 50

Μ.Χ.Α.=[ΧΑ(P1)+ΧΑ(P2)+ΧΑ(P3)+ΧΑ(P4)+ΧΑ(P5)]/5 = (0 + 7 + 23 + 41 + 50)/5 =>

**Μ.Χ.Α.=24,2**

Μ.Χ.Δ. :

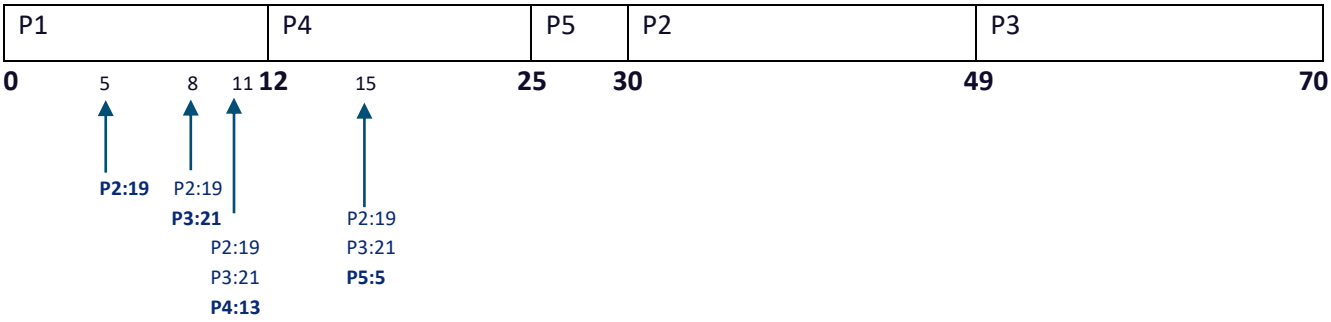
- ΧΔ(P1)= 12-0 = 12
- ΧΔ(P2)= 31-5 = 26
- ΧΔ(P3)=52-8 = 46

- ΧΔ(P4)=65-11 = 54
- ΧΔ(P5)=70-15 = 55

M.Χ.Δ. = [ΧΔ(P1)+ΧΔ(P2)+ΧΔ(P3)+ΧΔ(P4)+ΧΔ(P5)]/5 = (12+26+46+54+55)/5 =>

**M.Χ.Δ.=38.6**

**SJF (Shortest Job First)**




M.Χ.Α. :

- ΧΑ(P1)= 0
- ΧΑ(P4)= 12-11 = 1
- ΧΑ(P5)= 25-15 = 10
- ΧΑ(P2)= 30-5 = 25
- ΧΑ(P3)= 49-8 = 41

M.Χ.Α.=[ΧΑ(P1)+ΧΑ(P2)+ΧΑ(P3)+ΧΑ(P4)+ΧΑ(P5)]/5 = ( 0+1+10+25+41)/5 =>

**M.Χ.Α.= 15,4**

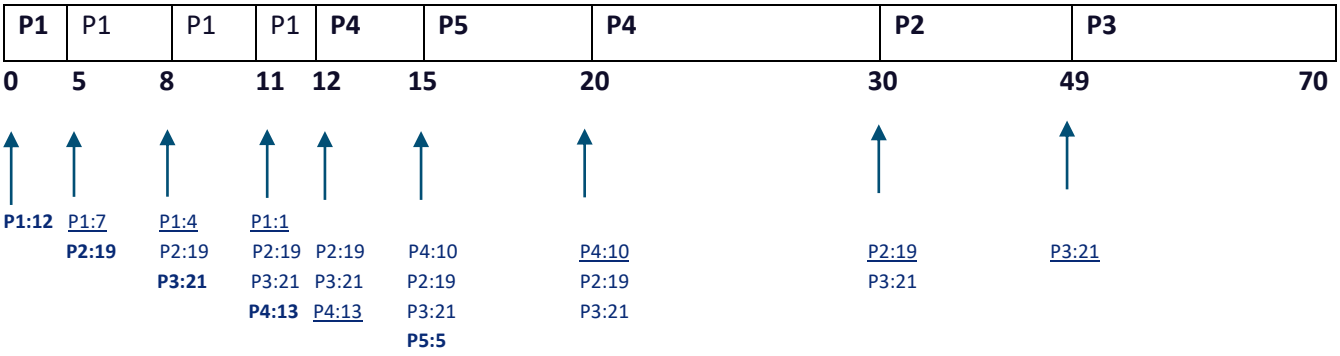
M.Χ.Δ. :

- ΧΔ(P1)= 12
- ΧΔ(P4)= 25-11 = 14
- ΧΔ(P5)= 30-15 = 15
- ΧΔ(P2)= 49-5= 44
- ΧΔ(P3)= 70-8= 62

M.Χ.Δ. = [ΧΔ(P1)+ΧΔ(P2)+ΧΔ(P3)+ΧΔ(P4)+ΧΔ(P5)]/5 = (12+14+15+44+62)/5 =>

**M.Χ.Δ.=29,4**

**SRTF (Shortest Remaining Time First)**




M.X.A. :

- XA(P1)= 0
- XA(P4)= 1+5 = 6
- XA(P5)= 0
- XA(P2)= 30-5 = 25
- XA(P3)= 49-8 = 41

M.X.A.=[XA(P1)+XA(P2)+XA(P3)+XA(P4)+XA(P5)]/5 = (0+0+6+25+41)/5 =>

**M.X.A.= 14,4**

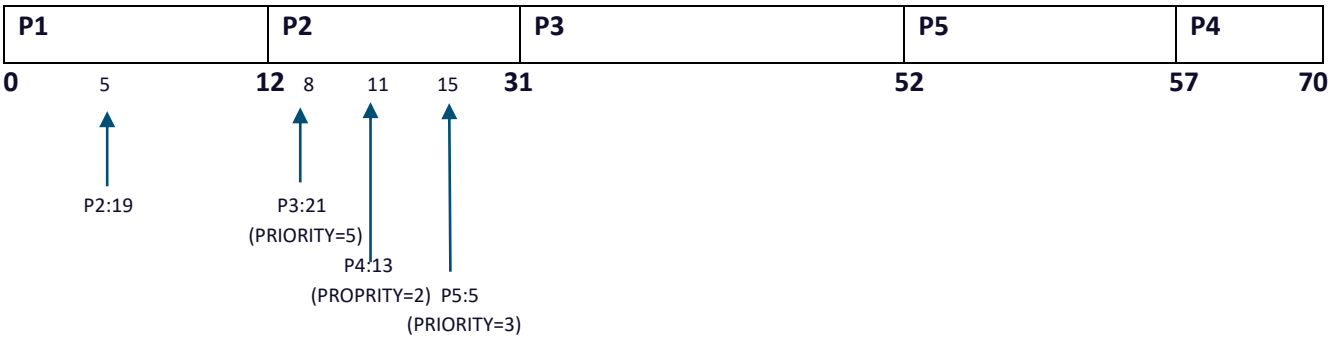
M.X.Δ. :

- XΔ(P1)= 12-0 = 12
- XΔ(P4)= 30-11 = 19
- XΔ(P5)= 20-15 = 5
- XΔ(P2)= 49-5 = 44
- XΔ(P3)= 70-8 = 62

M.X.Δ. = [XΔ(P1)+XΔ(P2)+XΔ(P3)+XΔ(P4)+XΔ(P5)]/5 = (12+19+5+44+62)/5 =>

**M.X.Δ.=28,4**

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




Μ.Χ.Α. :

- ΧΑ(P1)= 0
- ΧΑ(P4)= 57-11 = 46
- ΧΑ(P5)= 52-15= 37
- ΧΑ(P2)= 12-5 = 7
- ΧΑ(P3)= 31-8 = 23

Μ.Χ.Α.=[ΧΑ(P1)+ΧΑ(P2)+ΧΑ(P3)+ΧΑ(P4)+ΧΑ(P5)]/5 = (0+46+37+7+23)/5 =>

**Μ.Χ.Α.= 22,6**

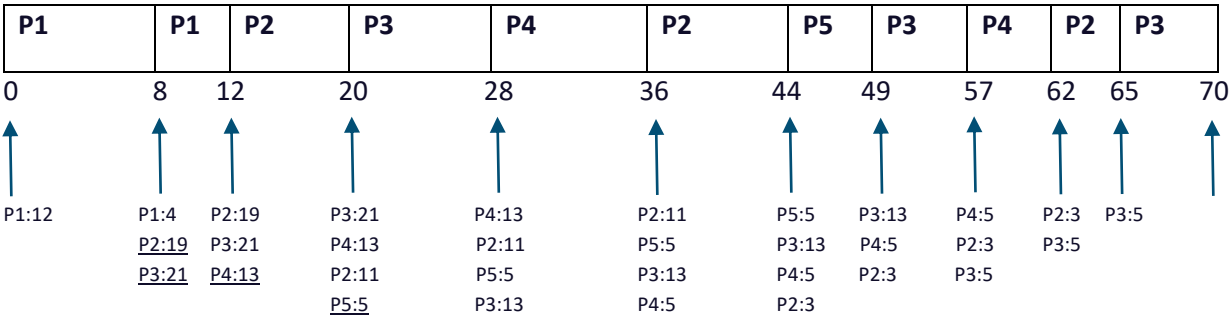
Μ.Χ.Δ. :

- ΧΔ(P1)= 12-0 = 12
- ΧΔ(P4)= 70-11 = 59
- ΧΔ(P5)= 57-15 =42
- ΧΔ(P2)= 31-5 = 26
- ΧΔ(P3)= 52-8 = 44

Μ.Χ.Δ. = [ΧΔ(P1)+ΧΔ(P2)+ΧΔ(P3)+ΧΔ(P4)+ΧΔ(P5)]/5 =(12+59+42+26+44)/5 =>

**Μ.Χ.Δ.=127,8**

**RR (Round Robin) με κβάντο χρόνου 8 msec:**




Μ.Χ.Α. :

- ΧΑ(P1)= 0
- ΧΑ(P4)= (28-11)+8+5+8 = 38
- ΧΑ(P5)= (44-15) = 19
- ΧΑ(P2)= (12-5)+8+8+5+8+5 = 41
- ΧΑ(P3)= (20-8)+8+8+5+5+3 = 41

Μ.Χ.Α.=[ΧΑ(P1)+ΧΑ(P2)+ΧΑ(P3)+ΧΑ(P4)+ΧΑ(P5)]/5 = (0+38+19+41+41)/5 =>

**Μ.Χ.Α.= 27,8**

Μ.Χ.Δ. :

- ΧΔ(P1)= 12-0 = 12
- ΧΔ(P4)= 62-11 = 51
- ΧΔ(P5)= 49-15 = 34
- ΧΔ(P2)= 65-5 = 60
- ΧΔ(P3)= 57-8 = 49

Μ.Χ.Δ. = [ΧΔ(P1)+ΧΔ(P2)+ΧΔ(P3)+ΧΔ(P4)+ΧΔ(P5)]/5 = (12+51+34+60+49)/5 =>

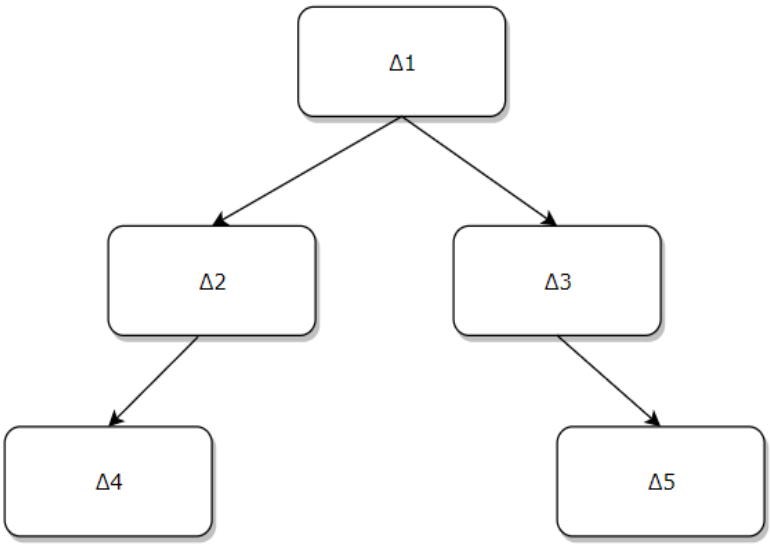
**Μ.Χ.Δ.=41,2**

**Ερώτημα Β [10]:**

**Β.1 [4]:**

Ο συχγρονισμός διεργασιών έχει ως εξής:

Γράφος Προτεραιότητας:




**Β.2 [6]:**

Αρχικοποίηση:

var a,b,c,d,e : semaphores;
a=0;
b=0;
c=0;
d=0;
e=0;

```
cobegin
        begin D 1; signal(a); signal(e); end;
        begin wait(a); D2; signal(b); end;
        begin wait(e); D3; signal(c); end;
        begin wait(b); D4; end;
        begin wait(c); D5; end;
coend;
```

**Ερώτημα Γ [8]**

| **Διεργασία Δ0** | **Διεργασία Δ1** | **Flag0** | **Flag1** | **turn** |
|---|---|---|---|---|
| | | FALSE | FALSE | 0 |
| Flag0 = TRUE; | | TRUE | FALSE | 0 |
| Εκτελεί εξωτερικό WHILE | | TRUE | FALSE | 0 |
| ΚΡΙΣΙΜΟ ΤΜΗΜΑ | | TRUE | FALSE | 0 |
| | Flag1 = TRUE; | TRUE | TRUE | 0 |
| | Εκτελεί εξωτερικό WHILE | TRUE | TRUE | 0 |
| | Εκτελεί εσωτερικό WHILE (noop) | TRUE | TRUE | 0 |
| Flag0 = FALSE ;; | | FALSE | TRUE | 0 |
| (forever) Flag0 = TRUE | | TRUE | TRUE | 0 |
| ΚΡΙΣΙΜΟ ΤΜΗΜΑ | | TRUE | TRUE | 0 |
| | Turn = 1; | TRUE | TRUE | 1 |
| ΚΡΙΣΙΜΟ ΤΜΗΜΑ | ΚΡΙΣΙΜΟ ΤΜΗΜΑ | TRUE | TRUE | 1 |
| | | | | |
| | | | | |
| | | | | |

**Ερώτημα Δ[12]**

| Supervisor | Worker1 | Worker2 | Worker3 | Σχόλια: |
|---|---|---|---|---|
| Signal(s0) | | | | Προσέλευση Επιβλέποντα στο Εργοτάξιο |
| | Wait(s0) | Wait(s0) | Wait(s0) | Προσέρχονται για συνεδρίαση οι εργαζόμενοι 3 |
| Wait(w0,w1,w2) | | | | Εργασία στο εργοτάξιο από εργάτες Επίβλεψη από επιβλέποντες |
| | Signal(s0) | Signal(s0) | Signal(s0) | Αποχώρηση εργατών |
| Wait(w0,w1,w2) | | | | Οι επιβλέποντες είναι μπλοκαρισμένοι ενώ οι εργάτες έχουν τελειώσει τη δουλειά τους |

Ο σημαφόρος signal(mutex) αναγκάζει την αποχώριση του supervisor μετά το πέρας της εργασίας και η wait(mutex) εξασφαλίζει αμοιβαίο αποκλεισμό μεταξύ διαφορετικών εργαζομένων της ομάδας supervisor. Εάν παραλειφθεί ο σημαφόρος signal(mutex) τότε υπάρχει περίπτωση **αδιέξοδου**.

**Ερώτημα Α [20]**

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

Αρχικοποίηση:

```
var a,b,c,d,e : semaphores;
a=0;b=0;c=0;d=0;e=1;

                /*Process1*/
for k=1 to 10 do
        begin wait(e); E1.1; signal(a);
        E1.2; signal(b);
        end
                /*Process2*/
for j=1 to 10 do
        begin wait(a); E2.1; signal(c);
        wait(b); E2.2; signal(d)
        end
                /*Process3*/
for l=1 to 10 do
        begin wait(c); E3.1;
        wait(a); E3.2; signal(e);
        end
```