1ο Project ΛΣ 2020-2021

Θεοδώρου Μιχαήλ 1067391
Φωτεινός Εμμανουήλ 1067428
Μαρκοδήμος Παναγιώτης 1067523

Μέρος 1

Ερώτημα Α)

Το συγκεκριμένο πρόγραμμα μέσω της μεθόδου fork() δημιουργεί 30 διεργασίες παιδιά με κοινό γονέα τα οποία αφού εκτελέσουν την sleep, τερματίζουν. Στην συνέχεια εκτυπώνει το process id της κάθε διεργασίας και το πως τερματίστηκε!

```
pid_t pid[N];
int i;
int child_status;
```

Πιο συγκεκριμένα:

Εδώ δημιουργεί έναν μονοδιάστατο πίνακα 30 θέσεων (γιατί N=30) με το όνομα pid για να αποθηκεύει τα ids των διεργασιών και 2 μεταβλητές , η 1η για τον μετρητή και η 2η για τις καταστάσεις των διεργασιών-παιδιών

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

      if (pid[i] == 0)
      {
            sleep(60-2*i);
            exit(100+i);
      }
}
```

Σε αυτή την λούπα η γονική διεργασία δημιουργεί τις 30 διεργασίες-παιδιά μέσω της fork αντιγράφοντας την διεργασία του γονέα δίνοντας όμως στο παιδί διαφορετική διεύθυνση στην μνημη και pid. Αφού η fork επιστρέφει 0 για τις διεργασίες-παιδιά τότε το pid[i]==0 είναι αληθές με αποτέλεσμα όλα τα παιδιά να εκτελέσουν sleep και τερματίσουν με exit status 100+1 (εφόσον ο τερματισμός είναι επιτυχής)

```
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);
}
```

Αφού η fork επιστρέφει το

pid του κάθε παιδιού για τη γονική διεργασία, η γονική διεργασία δεν θα μπει στο προηγούμενο if και θα συνεχίσει σε αυτό το for. Για κάθε παιδί, θα καλέσει την μέθοδο waitpid, η οποία θα ελέγξει την κατάσταση του παιδιού, και θα περιμένει να αλλάξει. Αφού αλλάξει, θα αποθηκεύσει την κατάσταση στην μεταβλητή child_status και θα επιστρέψει το pid του παιδιού (που θα αποθηκευτεί στο wpid).
Στην συνέχεια, στο if, μέσω του WEXITSTATUS(child_status) θα ελέγξει αν η διεργασία παιδί τερματίστηκε κανονικά όπου θα επιστρέψει 1, αλλιώς 0. Αν η διεργασία παιδί τερματίστηκε κανονικά, θα το τυπώσει μαζί με το pid και το exit status μεσω του WEXITSTATUS(child_status). Αλλιώς θα τυπώσει ότι τερματίστηκε αφύσικα μαζί με το wpid.

Ερώτημα Β)

Δημιουργήσαμε ένα πρόγραμμα σύμφωνα με τις απαιτήσεις της εκφώνησης, πραγματοποιώντας τον αλγόριθμο bakery και το shared array μέσω inter process communication με χρήση shared memory. Έχουμε δημιουργήσει ένα function find_maximum γιατί ήταν απαραίτητο για το number του αλγόριθμου bakery. Το κάθε παιδί πριν και αφού αυξήσει το περιεχόμενο του shared array, τυπώνει μια θέση του για έλεγχο. Επίσης, μετά το τερματισμό του κάθε παιδιού, η διεργασία-γονέας τυπώνει αν αυτός ήταν κανονικός ή αφύσικος.Τέλος, υπάρχει έλεγχος για το αν οι εντολές σχετικά με το shared memory εκτελέστηκαν σωστά.

To find_maximum function:

```
int find_maximum(int *p) //Function για την εύρεση του μέγιστου για την επιλογή number/ticket.
{
    int i;
    int max = 0;

    for (i = 0; i < N; i++)
        if (*(p + i) >= max)
            max = *(p + i);

    return max;
}
```

Main:

Στην αρχή της main αρχικοποιούμε της απαραίτητες μεταβλητές, το shared memory array και κάνουμε attach.

```c
int main()
{

int *sa_pointer; //Ο pointer που θα δείχνει στην αρχή του shared memory αλλα και του shared memory array
int j; //Μετρητές
int i;
int pid[N]; //Array για την αποθήκευση των pid των παιδιών
int child_status; //Αποθήκευση κατάστασης ενός παιδιού

//Αρχικοποιούμε τα array στο shared memory
int shmid = shmget (9090, 3 * N, 0644|IPC_CREAT); //Διαλέγουμε ένα κλείδι που πιστεύουμε οτι δεν χρησιμοποιείται

if(shmid < 0) //Έλεγχος σφάλματος
{
    perror("shmget\n");
    exit(1);
}
//Κάνουμε attach του pointer με το shared memory
sa_pointer = (int *)shmat(shmid, (void*)0, 0);
```

Στην συνέχεια προσαρμόζουμε τα pointers ώστε να δείχνουν στις σωστές θέσεις του shared memory και θέτουμε τα arrays στο 0.

```c
/*Οι παρακάτω προσαρμογές γίνονται επειδή αποφασίσαμε να χωρίσουμε το 3N μέγεθους shared memory σε
τρία μέρη, με τις N πρώτες θέσεις να χρησιμοποιούνται για το shared array, οι επόμενες N για το
array του choosing και οι υπόλοιπες N για το number*/

int *choosing_pointer = sa_pointer + N; //Προσαρμογή του sa_pointer ωστε να δείχνει στις θέσεις που χρησιμοποιούνται για το choosing
int *number_pointer = sa_pointer + 2 * N; //Προσαρμογή του sa_pointer ωστε να δείχνει στις θέσεις που χρησιμοποιούνται για το number


for (i=0; i<(3 * N); i++) //θέτουμε στο 0 όλες τις θέσεις του shared memory στο 0.
{
    *(sa_pointer + i) = 0;
}
```

Η γονική διεργασία εκτελεί την παρακάτω for για να δημιουργήσει N παιδιά, και γίνεται έλεγχος αν έγινε σωστά.

```c
for (i = 0; i < N; i++) //For loop για την διεργασία-γονεάς, ώστε να εκτελέσει N fork.
{
    pid[i] = fork(); //Εκτέλεση fork. Το pid χρησιμοποιείται από την διεργασία-γονέας για να αποθηκέυει το pid των παιδιών της.

    if (pid[i] < 0)
    {
        printf("Fork error.\n");
        return (-1);
    }
```

Μέσα σε αυτήν την for, οι διεργασίες-παιδιά μπαίνουν στην παρακάτω if, όπου εκτελούν τον αλγόριθμο bakery για να μπουν στην κρίσιμη περιοχή, και όταν μπουν, αυξάνουν κατα i το περιεχόμενο του shared array και τερματίζουν.

```
if (pid[i] == 0) //Έλεγχος, ώστε μονο αν η διεργασία είναι διεργασία-παιδί θα εισέλθει στο if, αφου θα έχει pid=0.
{

    *(choosing_pointer + i) = 1; //choosing = true
    *(number_pointer + i) = find_maximum + 1; //Η διεργασία παίρνει number/ticket.
    *(choosing_pointer + i) = 0; //choosing = false

    for (j = 0; j<N; j++)
    {
        while (*(choosing_pointer + j) == 1) //Όσο το choosing του j είναι true, μην κάνεις τίποτα
        {
            sleep(1);
        }

        /*Όσο το number του j δεν είναι μηδέν και είναι μικρότερο απο το number σου (number του i) ή
        όσο το number του j είναι ίσο με το δικό σου άλλα το j είναι μικρότερο απο εσένα (το i), μην
        κάνεις τίποτα. Αλλιώς, η διεργασία μπαίνει στο κρίσιμο τμήμα.*/
        while (*(number_pointer + j) != 0 && *(number_pointer + j) < *(number_pointer + i) || (*(number_pointer + j) == *(number_pointer + i) && j < i) )
        {
           sleep(1);
        }
    }
    /*Τα print έχουν τοποθετηθεί για έλεγχο. Όλες οι θέσεις του shared array πρέπει να έχουν την ίδια τιμή,
    για αυτό ελέγχουμε την πρώτη και την τελευταία ώστε να είμαστε σίγουροι ότι όλα δουλεύουν σωστά.*/
    printf("I am child %d and this is position 0 of the shared array before I added: %d \n", i, *(sa_pointer));
    for (j=0; j<N; j++) //Προσθήκη i σε κάθε θέση του shared array.
    {
       *(sa_pointer + j) += i;
    }
    printf("I am child %d and this is position 29 of the shared array after I added: %d \n", i, *(sa_pointer + 29));

    *(number_pointer + i) = 0; //Θέτει το number στο μηδέν αφού έχει βγει απο το κρίσιμο τμήμα.

    exit(i); //Τερματισμός διεργασίας-παιδού
}
```

Τέλος, η διεργασία γονέας, που δεν έχει μπει μέσα στο προηγούμενο if, συνεχίζει στον παρακάτω κώδικα, όπου περιμένει κάθε διεργασία-παιδί να τερματίσει, και τυπώνει πως τερματίστηκε. Στην συνέχεια κάνει detach απο το shared memory και επιστρέφει 0.

```
for (i=0; i<N; i++) //Η διεργασία-γονέας περιμένει να τερματιστούν όλες οι διεργασίες-παιδιά της και τυπώνει αν τερματίστηκαν σωστά.
{
    pid_t wpid = waitpid(pid[i] ,&child_status, 0);

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

//Shared memory detach
shmdt(sa_pointer);
shmctl(shmid, IPC_RMID, 0);

return (0);

}
```

Μέρος 2

Ερώτημα Α)

Υπάρχουν 3 πιθανές τελικές τιμές που μπορεί να λάβει η μεταβλητή X μετά το πέρας του παράλληλου κώδικα. Αν και υπάρχουν πολλά σενάρια, θα αναφέρουμε ένα για κάθε μία:

- Οι εντολές εκτελούνται με τη σειρά. Εκτελείται πρώτα ολόκληρη η εντολή X:=X+1, οπότε το X γίνεται 1 και στην συνέχεια ακολουθεί ολόκληρη η X:=Y+1, οπότε το X

γίνεται 10. Τελική τιμή X = 11.

- Οι εντολες εκτελούνται ανάποδα. Αυτή τη φορά, εκτελείται πρώτα η εντολή X:= Y+1 οπότε το X γίνεται 11 και μετά η εντολή X:=X+1 μετατρεπει την μεταβλητη X σε 12. Τελική τιμή X = 12.

- Στην τελευταία περίπτωση τα βηματα αναμιγνύονται, καθώς ξεκινάει η μία από τις δύο(2) εντολές αλλά διακόπτεται από την άλλη μετά το πέρας του 1ου βήματος. Έστω ότι 1η ξεκινάει η εντολή X:=X+1, που όπως προαναφέραμε διακόπτεται από ολόκληρη την X:= Y+1 η οποία κάνει την τιμή του X 11, παρ' όλ' αυτά επιστρέφοντας στην 1η εντολή ο καταχωρητής TX έχει ήδη την τιμή 0 απο πριν και στο βημα 2 γίνεται 1. Ετσι η τελικη τιμή του X γίνεται και αυτή με την σειρά της 1.

Ερώτημα Β)

(α):

```
//Αρχικοποίηση
var s1, s2, s3 semaphore;
s1:=1, s2:=0, s3:=0
cobegin
```

Process 1	Process 2	Process 3

while (true) {
  down(s1)
  print("P")
  print("I")
  up(s2)
}
	while (true) {
	  down(s2)
	  print("Z")
	  up(s3)
	}
		while (true) {
		  down(s3)
		  up(s2)
		  down(s3)
		  print("A")
		}

```
coend
```

(β):

```
//Αρχικοποίηση
var s1, s2, s3 semaphore;
s1:=1, s2:=0, s3:=0
cobegin
```

Process 1	Process 2	Process 3

while (true) {
down(s1)
print("P")
print("I")
up(s2)
}
	while (true) {
	down(s2)
	print("Z")
	up(s3)
	}
		while (true) {
		down(s3)
		up(s2)
		down(s3)
		print("A")
		up(s1)
		}

coend

Σημειωσεις:
Για το (α): Αρχικά, μόνο το process 1 μπορεί να εκτελέσει down, γιατί s1 = 1, αρα γίνεται s1 = 0, τυπώνεται PI, και γίνεται s1 = 0, s2 = 1. Στην συνέχεια μόνο το process 2 μπορεί να εκτελέσει down, γιατί s2 = 1, άρα γίνεται s2 = 0, τυπώνεται Z (PIZ μέχρι τώρα) και γίνεται s3 = 0. Το process 3 εκτελεί το πρώτο down(s3), s3 = 0, up(s2), s2 = 1, και μπαίνει σε αναμονή όταν εκτελεί το δεύτερο down(s3). Η s2 ξαναεκτελείται. (PIZZ μέχρι τώρα). s3 = 1. Το process 3 μπορεί να συνεχίσει, s3 = 0, τυπώνει A. Έχει τυπωθεί "PIZZA". Όλοι οι σηματοφόροι είναι στο 0 άρα δεν θα ολοκληρωθεί καμία άλλη διεργασία.

Για το (β): Γίνονται όλα ακριβώς όπως και στο A, μονο που στο τέλος του process 3 το s1 γίνεται 1 οπότε το process 1 μπορεί να ξαναεκτελεστεί ολόκληρο και επαναλαμβάνεται η ίδια διαδικασία, με αποτέλεσμα να τυπώνεται "PIZZAPIZZAPIZZA…"

Ερώτημα Γ)

Εκτελεση διεργασιων με την ζητουμενη σειρα του ερωτηματος (α):

Οι διεργασίες ξεκινάνε ταυτόχρονα.
Έστω ότι η B1 παίρνει την CPU πρώτη, εκτελεί down(s2); ομως s2=0 άρα τίθεται σε αναμονή.
Η A1 παίρνει την CPU, εκτελεί down(s1);, s1=1.
Η A1 εκτελεί up(s2), s2=1
Η A1 έχει ολοκληρωθεί, άρα φεύγει από την CPU
Η B1 παίρνει την CPU, έχει εκτελέσει το πρώτο down(s2), μπορει να συνεχίσει αφου s2=1, αρα γίνεται s2=0. Συνεχίζει εκτελώντας το δεύτερο down(s2), s2=0, τίθεται πάλι σε αναμονή.
Η A2 παίρνει την CPU, εκτελεί down(s1), s1=0.

Η Α2 εκτελεί up(s2), s2=1
Η Α2 έχει ολοκληρωθεί, άρα φεύγει από την CPU
Η Β1 παίρνει την CPU, που έχει εκτελέσει το δεύτερο down(s2), μπορει να συνεχίσει αφου s2=1, αρα γίνεται s2=0.
Η Β1 εκτελεί up(s1), s1=1.
Η Β1 εκτελεί up(s2), s2=1. Η Β1 έχει ολοκληρωθεί.
Η Β2 παίρνει την CPU, εκτελεί το πρώτο down(s2), s2=0
Η Β2 παίρνει την CPU, εκτελεί το δεύτερο down(s2), ομως s2=0 άρα τίθεται σε αναμονή.
Η Α3 παίρνει την CPU, εκτελεί down(s1), s1=0.
Η Α3 εκτελεί up(s2), s2=1. Η Α3 έχει ολοκληρωθεί.
Η Β2 παίρνει την CPU, έχει εκτελέσει down(s2), μπορει να συνεχίσει αφου s2=1, αρα γίνεται s2=0.
Η Β2 εκτελεί up(s1), s1=1.
Η Β2 εκτελεί up(s2), s2=1. Η Β2 έχει ολοκληρωθεί.

Άρα είναι δυνατή η ολοκλήρωση των διεργασιών με αυτή τη σειρά.

Εκτέλεση διεργασιών με την ζητούμενη σειρά ερωτήματος (β):

Οι διεργασίες ξεκινάνε ταυτόχρονα.
Έστω ότι η Β1 παίρνει την CPU πρώτη, εκτελεί down(s2); ομως s2=0 άρα τίθεται σε αναμονή.
Η Α1 παίρνει την CPU, εκτελεί down(s1);, s1=1.
Η Α1 εκτελεί up(s2), s2=1
Η Α1 έχει ολοκληρωθεί, άρα φεύγει από την CPU
Η Β1 παίρνει την CPU, έχει εκτελέσει το πρώτο down(s2), μπορει να συνεχίσει αφου s2=1, αρα γίνεται s2=0. Συνεχίζει εκτελώντας το δεύτερο down(s2), s2=0, τίθεται πάλι σε αναμονή.
Η Α2 παίρνει την CPU, εκτελεί down(s1), s1=0.
Η Α2 εκτελεί up(s2), s2=1
Η Α2 έχει ολοκληρωθεί, άρα φεύγει από την CPU
Η Β1 παίρνει την CPU, που έχει εκτελέσει το δεύτερο down(s2), μπορει να συνεχίσει αφου s2=1, αρα γίνεται s2=0.
Η Β1 εκτελεί up(s1), s1=1.
Η Β1 εκτελεί up(s2), s2=1. Η Β1 έχει ολοκληρωθεί.
Η Β2 παίρνει την CPU, εκτελεί το πρώτο down(s2), s2=0
Η Β2 παίρνει την CPU, εκτελεί το δεύτερο down(s2), ομως s2=0 άρα τίθεται σε αναμονή.

Εδώ φαίνεται ότι δεν είναι δυνατή η ολοκλήρωση της B2 πριν από την A3, αφού αναμένει να αυξηθεί το s2 κατά 1 για να συνεχίσει, και ο μόνος τρόπος να συμβεί αυτό είναι με την τελευταία εντολή της A3, up(s2), που θα ολοκληρωθεί μόλις την εκτελέσει.

<u>Ερώτημα Δ)</u>

Ερώτημα (α):

Ένα σενάριο εκτέλεσης το οποίο δεν οδηγεί στο επιθυμητό αποτέλεσμα είναι το εξής:

Η διεργασία i λαμβάνει την CPU και μετατρεπει το L σε K (L=1)
και στην συνεχεια μετατρεπει το K σε K+11 (K=12). Έπειτα βγαίνει από την CPU χωρίς να κάνει print_num.
Η διεργασία j λαμβάνει την CPU και μετατρέπει το L σε K (L=12) και στην συνέχεια μετατρέπει το K σε K+11 (K=23)
Η διεργασία j κάνει print_num(L, L+10).

Άρα, με αυτό το σενάριο, ο κώδικας δεν οδήγησε στο επιθυμητό αποτέλεσμα, αφού η πρώτη print_num που θα εκτελεστεί θα τυπώσει τους αριθμούς από το 12 έως το 22

Ερώτημα (β):

<u>Κώδικας:</u>

```
shared var K=L=1;
var s semaphore;
s:=1;
cobegin
```

```
Process_i
while (true) {

    down(s);
    L:=K;
    K:=K+11;
    print_num(L, L+10);
    up(s)

}
```
coend

Ερώτημα Ε)

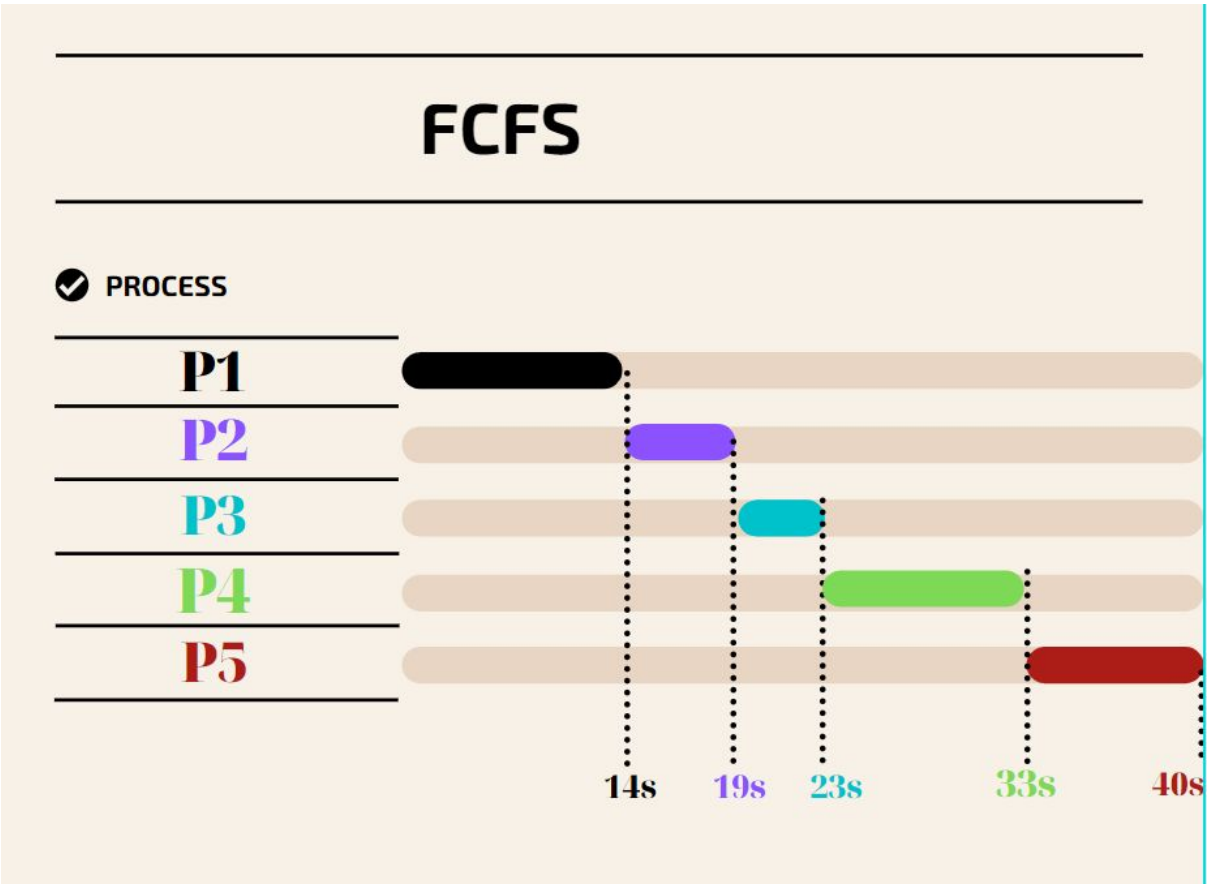
- Χρόνος ολοκλήρωσης ή χρονος διεκπεραίωσης(ΧΔ) είναι ο χρόνος που μια διεργασία καταναλώνει από τη χρονική στιγμή εισόδου της στο σύστημα μέχρι την ολοκλήρωση της εκτέλεσής της από την CPU.

  Επομένως εάν είναι t1 η χρονική στιγμή εισόδου και t2 η χρονική στιγμή εξόδου και tCPU είναι ο χρόνος που χρειάζεται η διεργασία για εξυπηρέτηση από τη CPU τότε:
  Χρόνος διεκπεραίωσης (ΧΔ) = t2 - t1

- Χρόνος αναμονης (ΧΑ) είναι ο χρόνος που μια διεργασία καταναλώνει αναμένοντας στο σύστημα χωρίς να πραγματοποιείται εξυπηρέτησή της από την CPU.

  Επομένως εάν είναι t1 η χρονική στιγμή εισόδου και t2 η χρονική στιγμή εξόδου και tCPU είναι ο χρόνος που χρειάζεται η διεργασία για εξυπηρέτηση από τη CPU τότε:
  Χρόνος αναμονής (ΧΑ) = t2 - t1 - tCPU
  ή
  Χρόνος αναμονής (ΧΑ) = ΧΔ - tCPU

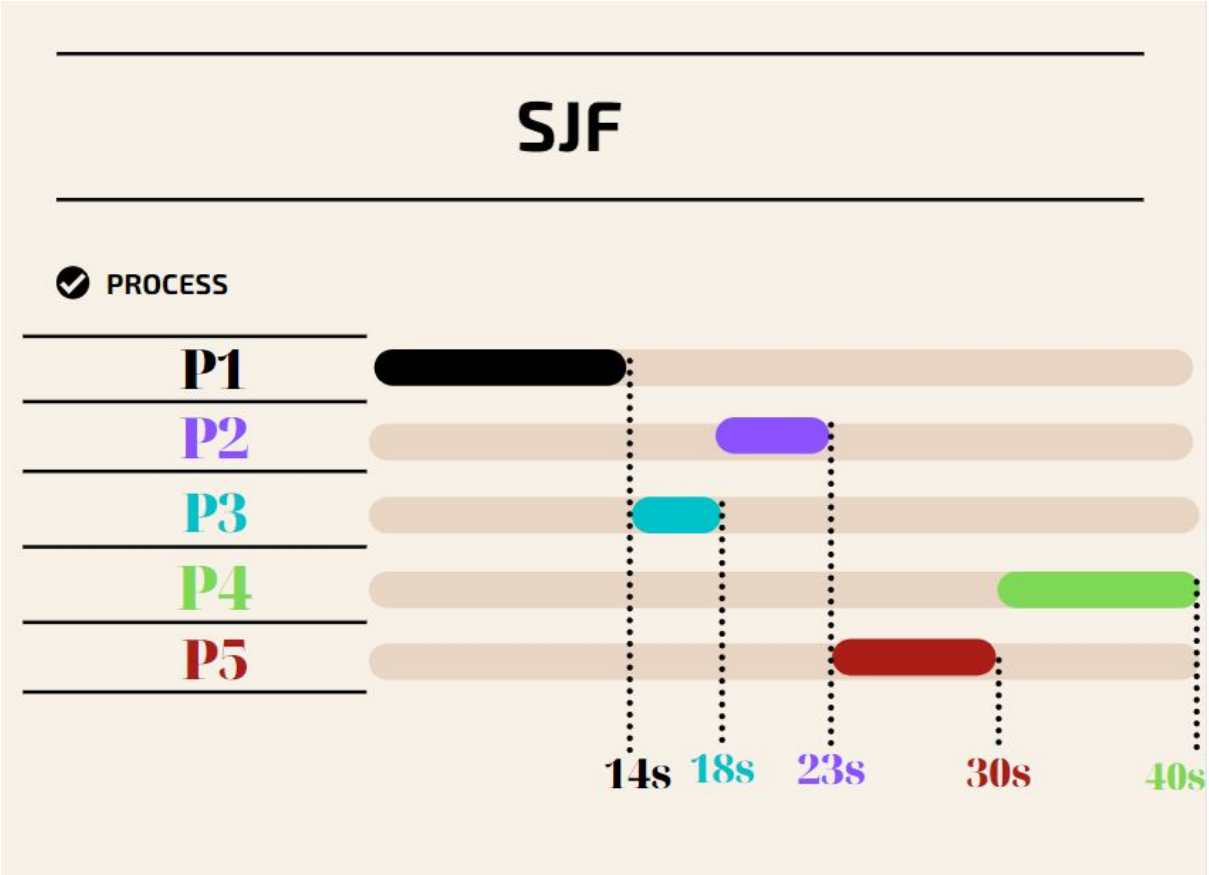
Διαγράμματα Gantt:



ΧΔ1 = 14 - 0 = 14s
ΧΔ2 = 19 - 2 = 17s
ΧΔ3 = 23 - 4 = 19s
ΧΔ4 = 33 - 7 = 26s
ΧΔ5 = 40 - 12 = 28s

ΧΑ1 = 14 - 14 = 0s
ΧΑ2 = 17 - 5 = 12s
ΧΑ3 = 19 - 4 = 15s
ΧΑ4 = 26 - 10 = 16s
ΧΑ5 = 28 - 7 = 21s

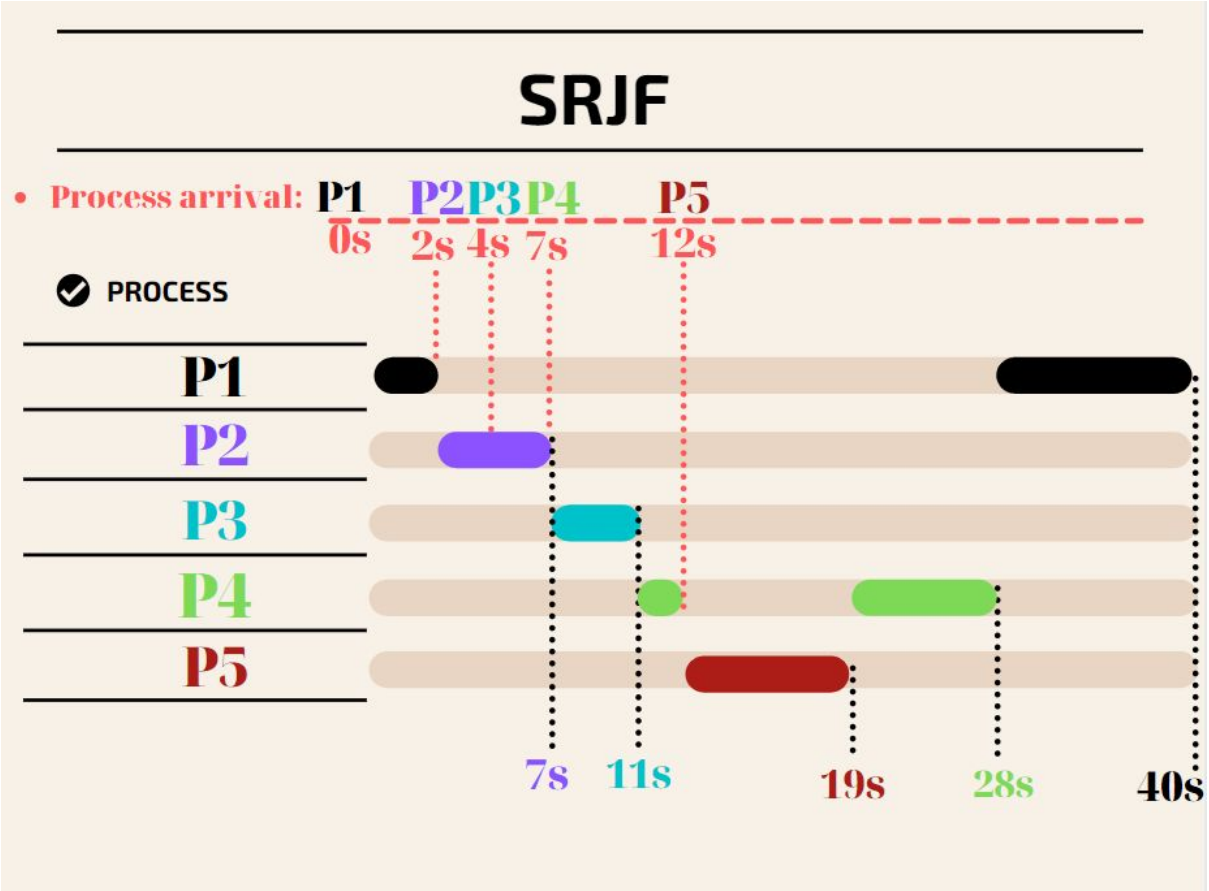
ΜΧΔ = (14 + 17 + 19 + 26 + 28) / 5 = 28,8s
ΜΧΑ = (0 + 12 + 15 + 16 + 21) / 5 = 12,8s



ΧΔ1 = 14 - 0 = 14s
ΧΔ2 = 23 - 2 = 21s
ΧΔ3 = 18 - 4 = 14s
ΧΔ4 = 40 - 7 = 33s
ΧΔ5 = 30 - 12 = 18s

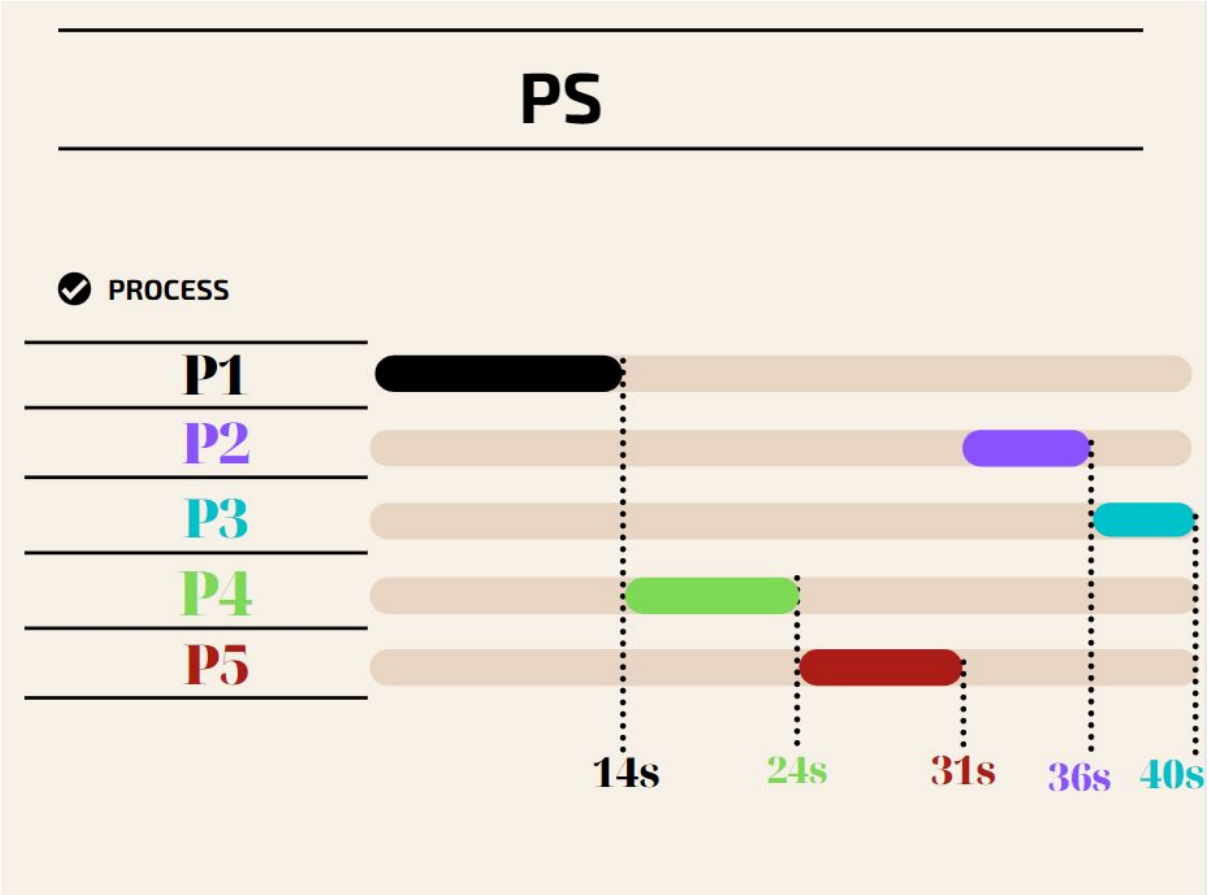
ΧΑ1 = 14 - 14 = 0s
ΧΑ2 = 21 - 5 = 16s
ΧΑ3 = 14 - 4 = 10s
ΧΑ4 = 33 - 10 = 23s
ΧΑ5 = 18 - 7 = 11s

ΜΚΔ = (14 + 21 + 14 + 33 + 18) / 5 = 20s
ΜΧΑ = (0 + 16 + 10 + 23 + 11) / 5 = 12s



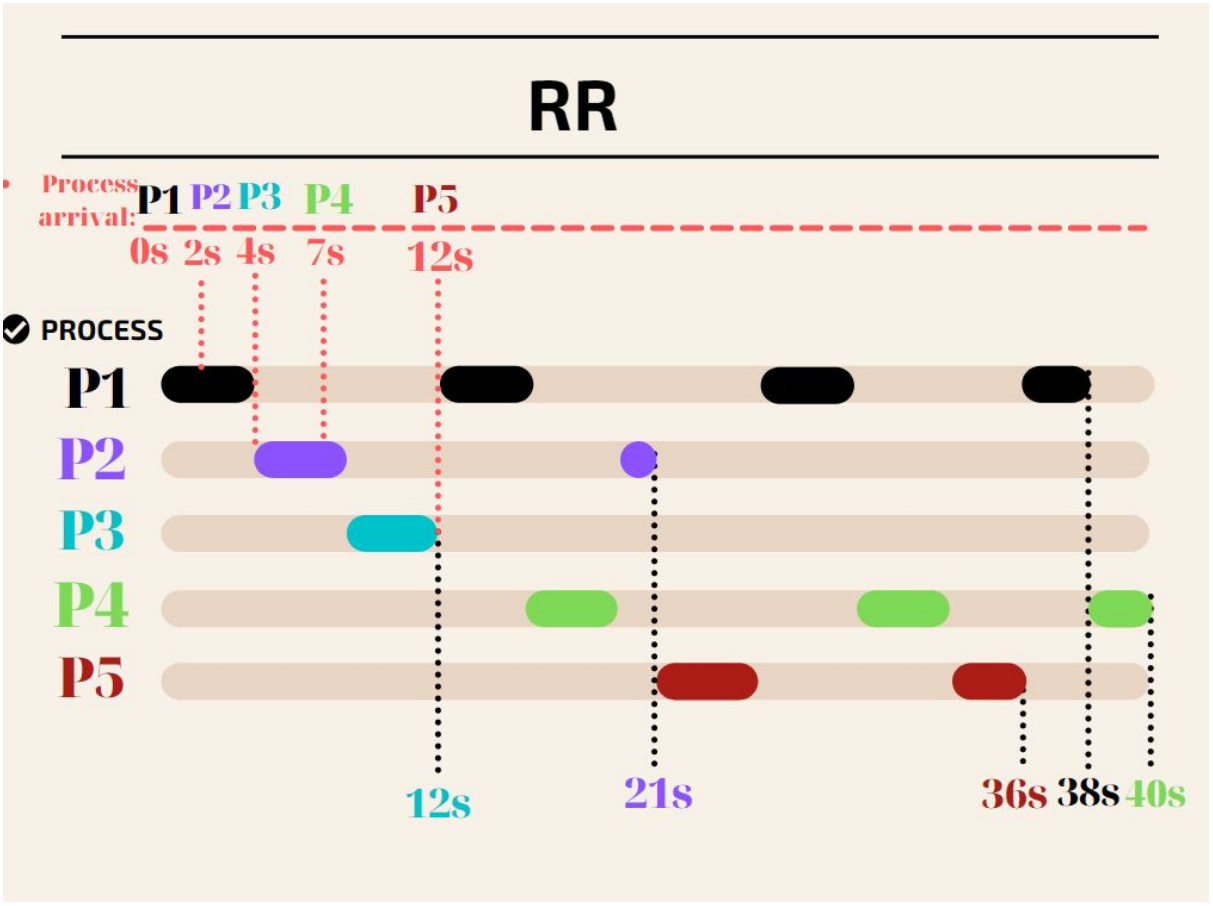
XΔ1 = 40 - 0 = 40s XA1 = 40 - 14 = 26s
XΔ2 = 7 - 2 = 5s XA2 = 5 - 5 = 0s
XΔ3 = 11 - 4 = 7s XA3 = 7 - 4 = 3s
XΔ4 = 28 - 7 = 21s XA4 = 21 - 10 = 11s
XΔ5 = 19 - 12 = 7s XA5 = 7 - 7 = 0s

MXΔ = (40 + 5 + 7 + 21 + 7) / 5 = 16s
MXA = (26 + 0 + 3 + 11 + 0) / 5 = 8s



XΔ1 = 14 - 0 = 14s XA1 = 14 - 14 = 0s
XΔ2 = 36 - 2 = 34s XA2 = 34 - 5 = 29s
XΔ3 = 40 - 4 = 36s XA3 = 36 - 4 = 32s
XΔ4 = 24 - 7 = 17s XA4 = 17 - 10 = 7s
XΔ5 = 31 - 12 = 19s XA5 = 19 - 7 = 12s

MXΔ = (14 + 34 + 36 + 17 + 19) / 5 = 24s
MXA = (0 + 29 + 32 + 7 + 12) / 5 = 16s



XΔ1 = 38 - 0 = 38s XA1 = 38 - 14 = 24s
XΔ2 = 21 - 2 = 19s XA2 = 19 - 5 = 14s
XΔ3 = 12 - 4 = 8s XA3 = 8 - 4 = 4s
XΔ4 = 40 - 7 = 33s XA4 = 33 - 10 = 23s
XΔ5 = 36 - 12 = 24s XA5 = 24 - 7 = 17s

MXΔ = (38 + 19 + 8 + 33 + 24) / 5 = 24,4s
MXA = (24 + 14 + 4 + 23 + 17) / 5 = 16,4s