ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ
UNIVERSITY OF PATRAS

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ ΚΑΙ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ

Ακαδημαϊκό Έτος 2022-2023

ΤΕΧΝΙΚΗ ΑΝΑΦΟΡΑ

1η ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ

ΤΟΠΑΛΗΣ ΛΑΖΑΡΟΣ
upnet email ceid email
A.M.: 1088101
ΣΑΜΑΡΑ ΧΡΙΣΤΙΝΑ – ΕΛΕΑΝΝΑ
upnet email ceid email
A.M.: 1084622

Πάτρα 2022-2023

ΠΕΡΙΕΧΟΜΕΝΑ

1. Πρόλογος

Όλα τα μέρη του project υλοποιήθηκαν πλήρως. Οι αλγόριθμος και τυχόν εικόνες που χρησιμοποιήθηκαν, βρίσκονται και στο github.

2. Shell Scripting

Για το πρώτο ερώτημα, δημιουργήσαμε το πρόγραμμα **logparser.sh** σε BASH shell, το οποίο ανάλογα με τις παραμέτρους που καλείται, εκτελεί και μια διαφορετική λειτουργία. Για αυτό φτιάξαμε ένα case loop που περιγράφει τις διαφορετικές τιμές που παίρνει η μεταβλητή **$#**, η οποία προσδιορίζει τον αριθμό των παραμέτρων. Στην περίπτωση της κλήσης του logparser.sh χωρίς καμία παράμετρο (**$# == 0**) εκτυπώνονται στην οθόνη τα ΑΜ «1084622|1088101» με την χρήση της εντολής **echo**.

Η επόμενη περίπτωση είναι να έχει μια παράμετρο (**$# == 1**). Τότε ελέγχεται εάν αυτή η παράμερος αποτελεί log αρχείο με την συνθήκη **[$1 == *".log"]**, αν δηλαδή η κατάληξή της αποτελεί κατάληξη log αρχείου. Αν η συνθήκη είναι αληθής, τότε καλείται η εντολή **cat $1**, με την οποία εκτυπώνονται τα περιεχόμενα του αρχείου που δόθηκε σαν παράμετρος. Σε περίπτωση που δεν δοθεί log αρχείο σαν όρισμα εκτυπώνεται το μήνυμα "*Wrong File Argument*" με την εντολή echo.

Στην συνέχεια έχουμε τη περίπτωση των δύο παραμέτρων (**$# == 2**). Πάλι έχουμε τον έλεγχο της πρώτης παραμέτρου για το αν αυτή αποτελεί log αρχείο. Αν είναι αληθής, το πρόγραμμα περνάει στο εμφωλευμένο if loop, με το οποίο ελέγχει τις τιμές της δεύτερης παραμέτρου. Στην περίπτωση που είναι ψευδής εκτυπώνεται το μήνυμα "*Wrong File Argument*".

Αν η δεύτερη παράμετρος είναι "**--usrid**", τότε το πρόγραμμα καλεί την **mining_usernames()**. Η συνάρτηση αυτή απομονώνει την τρίτη στήλη του αρχείου **access.log** και τα αποθηκεύει στο αρχείο **temp.txt**, αντικαθιστώντας τα περιεχόμενα του, αν έχει. Στην συνέχεια, το σύστημα διαβάζει κάθε γραμμή του αρχείου και τα αποθηκεύει στην μεταβλητή **line**. Με την εντολή **grep $line** αναζητεί στο αρχείο access.log κάθε εμφάνιση της μεταβλητής line και με την **wc -l** τις μετράει και βγάζει το σύνολο. Στο τέλος της επανάληψης, θα εκτυπωθεί κάθε όνομα χρήστη μαζί με τον συνολικό αριθμό εμφανίσεών του στο access.log. Στην συνέχεια το προσωρινό αρχείο temp.txt διαγράφεται.

Αν η δεύτερη παράμετρος πάρει την τιμή "**--browsers**" τότε εκτελείται η συνάρτηση **count_browsers $1**, η οποία ψάχνει τα προγράμματα περιήγησης {Mozilla, Chrome, Safari, Edg} στο αρχείο access.log. Συγκεκριμένα, δημιουργήσαμε την συνάρτηση **match()**, η οποία παίρνει δύο ορίσματα και αναζητεί το δεύτερο όρισμα στο αρχείο που προσδιορίζει το πρώτο. Έτσι, για κάθε browser στην λίστα μετριούνται οι εμφανίσεις του και εκτυπώνονται.
*match και στην mining_usersname()

Τέλος, αν ως δεύτερη παράμετρος οριστεί η "**--datum**" τότε εκτυπώνεται το μήνυμα "*Wrong Date*".

Σε κάθε άλλη περίπτωση εκτυπώνεται το μήνυμα "*Wrong Option Argument*".

Τέλος, έχουμε τις περιπτώσεις τριών παραμέτρων. Ο έλεγχος για το αν η πρώτη παράμετρος αποτελεί log αρχείο παραμένει, ενώ αν δεν είναι εκτυπώνεται το μήνυμα λάθος "*Wrong File Argument*". Αν η πρώτη παράμετρος ικανοποιεί την συνθήκη, περνάμε στον έλεγχο της δεύτερης παραμέτρου, η οποία πρέπει να έχει κάποια από τις επόμενες τιμές: { "**--usrid**", "**--method**", "**--servprot**", "**--datum**" }.

Αν το όρισμα **$2** είναι το "**--usrid**", τότε χρησιμοποιείται η εντολή **awk**, με την οποία αναζητείται το τρίτο όρισμα στο τρίτο πεδίο του αρχείου **$1**. Αν ταιριάζει εκτυπώνεται ολόκληρη η γραμμή.

Αν ως όρισμα **$2** δοθεί το "**--method**", και το όρισμα **$3** είναι είτε **POST** είτε **GET**, τότε χρησιμοποιείται η εντολή **cat** για το άνοιγμα του αρχείου **$1** και εκτυπώνονται οι εγγραφές που περιέχουν την αντίστοιχη μέθοδο.

Αν το όρισμα **$2** οριστεί ως "**--servprot**" και **$3 ==** "**IPv4**", τότε με την εντολή **awk** βρίσκονται και εκτυπώνονται όλες οι γραμμές οι οποίες περιέχουν την τελεία "**.**" στο πρώτο πεδίο τους.

Αν το όρισμα **$2** έχει τιμή πάλι "**--servprot**" και **$3 ==** "**IPv6**", τότε με τον ίδιο τρόπο βρίσκονται και εκτυπώνονται όλες οι γραμμές οι οποίες περιέχουν άνω-κάτω τελεία "**:**" στο πρώτο πεδίο τους.

Τέλος, αν το όρισμα **$2** είναι το "**--datum**" και ως όρισμα **$3** έχουμε έναν από τους μήνες {Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec}, τότε το σύστημα με τις εντολές **awk** και **grep** βρίσκει όλες τις εγγραφές του αρχείου **$1** που δημιουργήθηκαν τον μήνα **$3**. Σε περίπτωση λάθους στο τρίτο όρισμα εκτυπώνεται το μήνυμα "*Wrong Date*".

3. Διεργασίες

Για την υλοποίηση των μηνυμάτων κατασκευάστηκε μία δομή mesg_buffer με δύο πεδία, ένα mesg_type που παίρνει έναν αριθμό που αντιπροσωπεύει τον τύπο του μηνύματος και έναν πίνακα mesg_data μεγέθους τριών double αριθμών. Όταν η αρχική διεργασία στέλνει στα παιδιά το μήνυμα, η μεταβλητή mesg_type φέρει την τιμή 1, και τα δύο πρώτα κελιά του πίνακα mesg_data ενημερώνουν το κάθε παιδί για ποιο διάστημα να υπολογίσει το ολοκλήρωμα. Για παράδειγμα, αν το συνολικό ολοκλήρωμα πρέπει να υπολογιστεί για το διάστημα [1, 4] και έχουμε 4 διεργασίες, συμπεριλαμβανομένης και της αρχικής, τότε το πρώτο παιδί θα υπολογίσει το ολοκλήρωμα στο διάστημα [1, 2], το δεύτερο στο [2, 3] και το τελευταίο στο διάστημα [3, 4]. Αντίθετα, όταν υπολογιστεί το αποτέλεσμα του ολοκληρώματος, η κάθε διεργασία στέλνει ένα μήνυμα στον γονέα με mesg_type = 2, στο οποίο το τρίτο πεδίο του πίνακα mesg_data έχει το υπολογισμένο αποτέλεσμα.

Πιο ειδικά, με την έναρξη του προγράμματος ερωτάται ο χρήστης για το πλήθος των διεργασιών που επιθυμεί να δημιουργηθούν και πραγματοποιείται ένας τυπικός έλεγχος για το πλήθος που ζητήθηκε (να είναι υποχρεωτικά θετικός). Έπειτα για τον σκοπό της δημιουργίας ενός αρκετά σπάνιου κλειδιού, το οποίο θα χρησιμοποιηθεί για την δημιουργία της ουράς μηνυμάτων, δημιουργείται ένα προσωρινό κρυφό αρχείο, ονόματι .tempfile και στην συνέχεια, με χρήση αυτού, και το κλειδί της ουράς μηνυμάτων. Ακολουθεί η εκτέλεση της fork() που έχει ως σκοπό την δημιουργία τόσων διεργασιών όσες ζήτησε ο χρήστης στην αρχή. Η γονική διεργασία συνεχίζει την εκτέλεση της με τον υπολογισμό των διαστημάτων στο οποίο οι διεργασίες πρέπει να υπολογίσουν το ολοκλήρωμα. Η διεργασία απλά χωρίζει το διάστημα σε ίσα μέρη, όσα και οι διεργασίες και αναθέτει σε κάθε μία ένα από αυτά, στέλνει με μήνυμα αυτές της πληροφορίες στις διεργασίες-παιδιά και έπειτα περιμένει ώστε να ολοκληρωθούν όλες και να λάβει ένα μήνυμα τύπου mesg_buffer, το οποίο, όπως αναφέρθηκε θα περιέχει το τελικό αποτέλεσμα κάθε διεργασίας.

Παράλληλα με αυτούς τους υπολογισμούς της γονικής διεργασίας, εκτελούνται και οι διεργασίες-παιδιά. Με την δημιουργία τους, χρησιμοποιούν και αυτές το αρχείο .tempfile που δημιουργήθηκε, ώστε να παράγουν τον ίδιο μοναδικό κωδικό με αυτόν του γονέα, ώστε να επιτευχθεί η αμφίδρομη επικοινωνία μεταξύ γονέα-παιδιού και περιμένουν έως ότου λάβουν κάποιο μήνυμα, και συγκεκριμένα ένα μήνυμα με mesg_type = 2, εφόσον έτσι το ορίσαμε στην γονική διεργασία. Η γονική τοποθετεί όλα τα μηνύματα σε μία ροή και η κάθε διεργασία-παιδί λαμβάνει το πρώτο που θα βρει. Στην συνέχεια εκτελεί τον υπολογισμό που απαιτείται, με την μόνη διαφορά ότι το διάστημα που χωρίζεται το υπολογιζόμενο από κάθε διεργασία-παιδί διάστημα είναι διαφορετικό από το δοθέν. Αυτό συνέβη καθώς, σε αυτήν την περίπτωση, το διάστημα που λαμβάνει η κάθε διεργασία είναι μικρότερο σε σχέση με το συνολικό και η διατήρηση του ίδιου πλήθους υποδιαστημάτων δημιουργεί ένα υπερβολικά πολύ πυκνό χώρο που οδηγούσε, όπως παρατηρήθηκε σε μεγάλες απαιτήσεις χρόνου, μεγαλύτερες και από ότι όταν τον υπολογισμό τον εκτελούσε μόνο μία διεργασία. Για τον λόγο αυτόν, λήφθηκε η απόφαση να μειωθεί και το πλήθος των υποδιαστημάτων. Εφόσον ολοκληρωθεί ο υπολογισμός, η κάθε διεργασία τοποθετεί το αποτέλεσμα που βρήκε στο 3 κελί του πίνακα mesg_data, θέτει τον τύπο του μηνύματος σε 2, το στέλνει στον γονέα και τερματίζει.

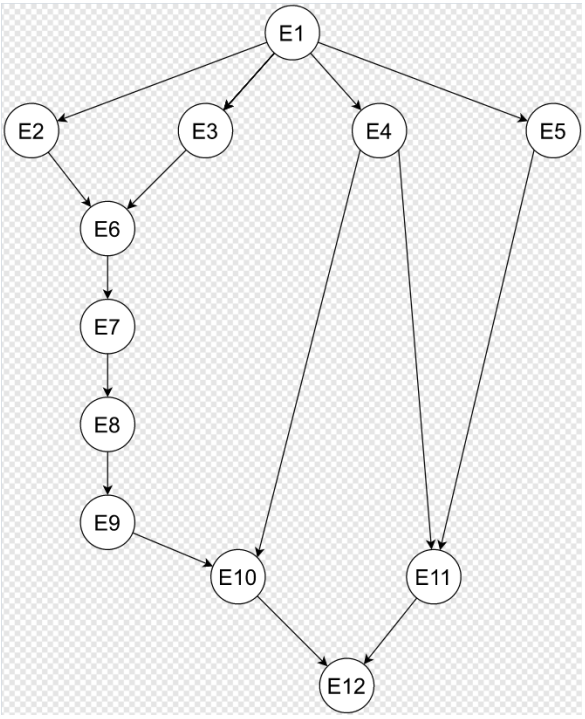
Εφόσον έχουν τερματίσει όλες οι διεργασίες, ο γονέας έχει λάβει τα μηνύματα με τα αποτελέσματα. Έπειτα προσθέτει όλα τα ληφθέντα από τα παιδιά αποτελέσματα, βρίσκει το συνολικό άθροισμα και το τυπώνει στην οθόνη μαζί με τον χρόνο που χρειάστηκε ώστε να πραγματοποιηθεί όλη η διαδικασία. Τέλος αποδεσμεύει την μνήμη

που χρειάστηκε, διαγράφει το αρχείο .tempfile, κλείνει την ροή μηνυμάτων που είχε δημιουργήσει στην αρχή και τερματίζει.

```
number of processes: 3
Time=5.577230, Result=4.282459
```

4. Διαδιεργασιακή Επικοινωνία

α. Γράφημα Προτεραιότητας

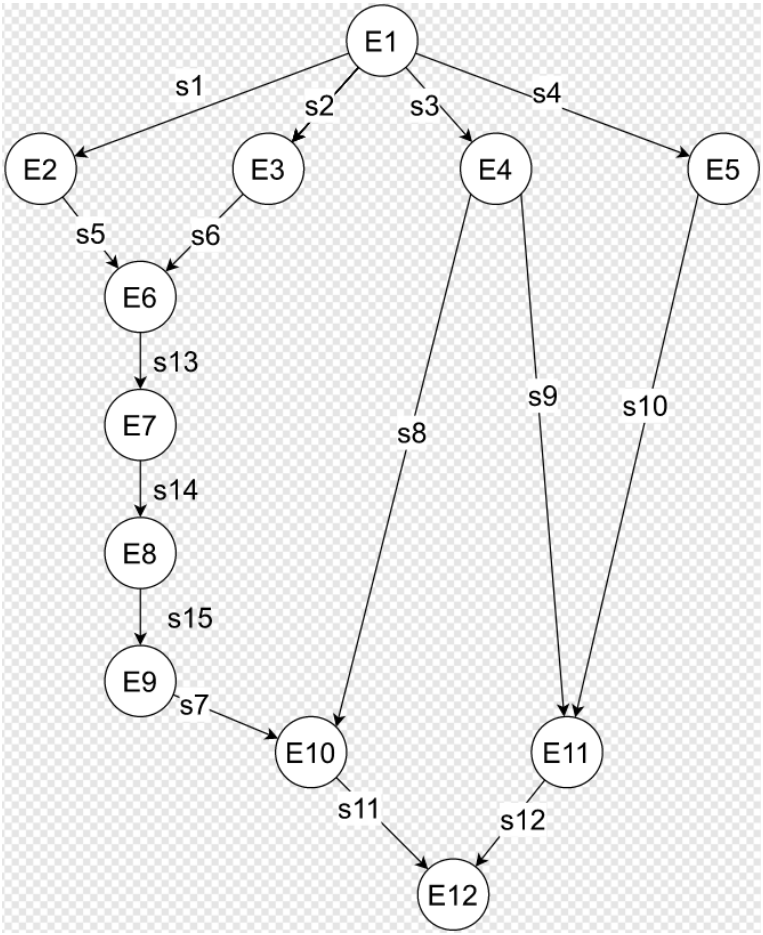


Στο διπλανό σχήμα απεικονίζεται ο γράφος προτεραιότητας της γραμμής παραγωγής μιας φαρμακοβιομηχανίας. Οι κόμβοι αντιπροσωπεύουν τις εργασίες που απαιτούνται να εκτελεστούν με την κατάλληλη σειρά. Όπως βλέπουμε, η διεργασία **E1** είναι η πρώτη που εκτελείται. Στην συνέχεια εκτελούνται παράλληλα οι **E2**, **E3**, **E4** και **E5**. Οι **E6, E7, E8** και **E9** εκτελούνται σειριακά, αφού τερματίσουν οι E2 και η E3. Στην συνέχεια, έχουμε την διεργασία **E10**, η οποία μπορεί να εκτελεστεί αφού τερματίσει τόσο η E9 όσο και η E4. Μετά έχουμε την διεργασία **E11** που εκτελείται μετά τις E4 και E5, και τέλος η διεργασία **E12**, που εκτελείται μετά τον τερματισμό των E10 και E11.

β. Ψευδοκώδικας συγχρονισμού χωρίς σημαφόρους

```
E1;
        COBEGIN
                BEGIN
                        COBEGIN
                                E2;
                                E3;
                        COEND;
                        E6;
                        E7;
                        E8;
                        E9;
                END;
                E4;
                E5;
        COEND;
        COBEGIN
                E10;
                E11;
        COEND;
E12;
```

γ. Ψευδοκώδικας συγχρονισμού με σημαφόρους



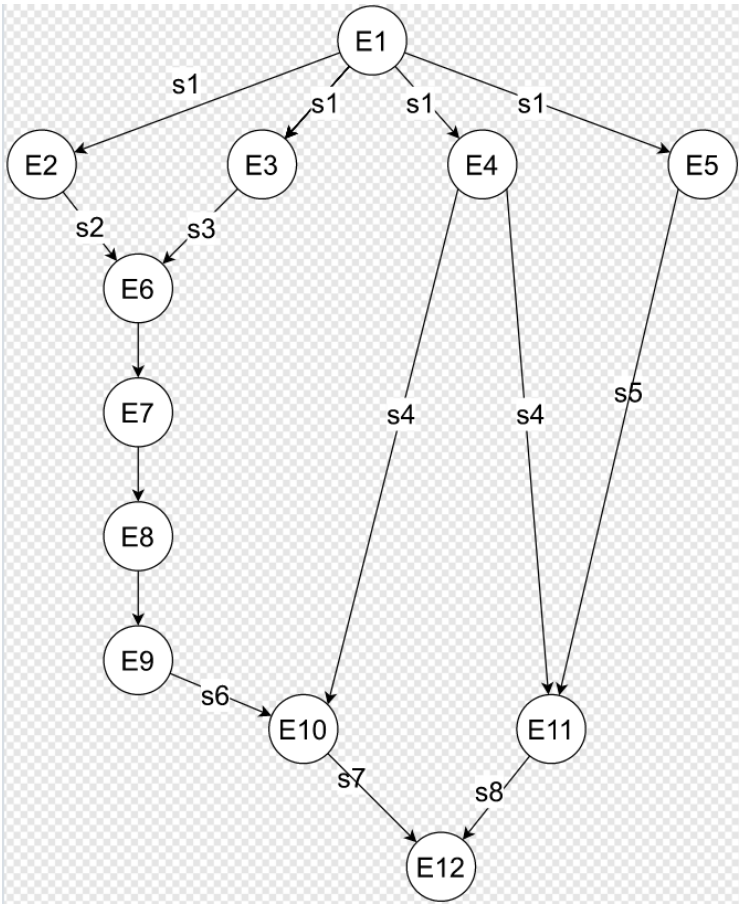
Για την απλή εκδοχή του συγχρονισμού με σημαφόρους, ορίζουμε μια μεταβλητή για κάθε ακμή του γραφήματός μας. Όλοι οι σημαφόροι αρχικοποιούνται με 0. Κάθε φορά που μια διεργασία φτάνει στο τέλος της ανεβάζει κατά 1 την τιμή των σημαφόρων των εξερχόμενων ακμών.

ΚΩΔΙΚΑΣ

```
type semaphore = record
   count: integer;
   queue: ουρά διεργασιών;
end;

var s1, s2, s3, s4, s5, s6, s7, s8, s9, s10, s11, s12, s13, s14, s15: semaphore;
s1 = s2 = s3 = s4 = s5 = s6 = s7 = s8 = s9 = s10 = s11 = s12 = s13 = s14 = s15 = 0;

cobegin
   repeat
      begin E1; up(s1); up(s2); up(s3); up(s4); end;
      begin down(s1); E2; up(s5); end;
      begin down(s2); E3; up(s6); end;
      begin down(s3); E4; up(s8); up(s9); end;
      begin down(s4); E5; up(s10); end;
      begin down(s5); down(s6); E6; up(s13); end;
```

```
      begin down(s13); E7; up(s14); end;
      begin down(s14); E8; up(s15); end;
      begin down(s15); E9; up(s7); end;
      begin down(s7); down(s8); E10; up(s11); end;
      begin down(s9); down(s10); E11; up(s12); end;
      begin down(s11); down(s12); E12; end;
   forever;
coend;
```

δ. <u>Βέλτιστος ψευδοκώδικας συγχρονισμού με σημαφόρους</u>



Για τον βέλτιστο ψευδοκώδικα συγχρονισμού περιορίσαμε τον αριθμό των σημαφόρων στους 8. Συγκεκριμένα, στις ακμές που εξέρχονται από τον ίδιο κόμβο αντιστοιχεί ο ίδιος σημαφόρος. Επίσης, για τις ακμές που ενώνουν τις διεργασίες **E6**, **E7**, **E8** και **E9**, επειδή αυτές εκτελούνται σειριακά χωρίς εξαρτήσεις άλλων διεργασιών, δεν τους ανατέθηκε σημαφόρος.

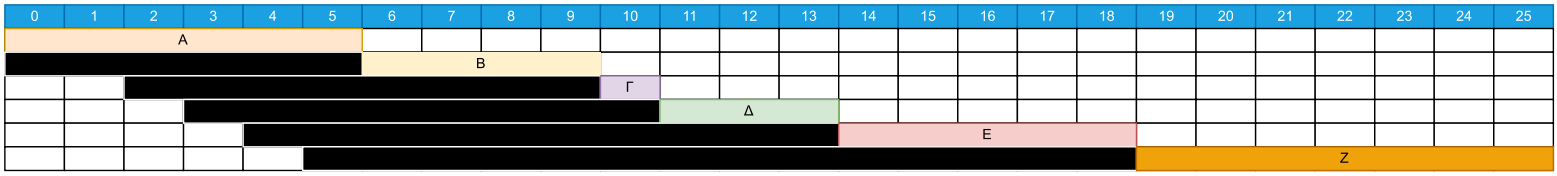
<u>ΚΩΔΙΚΑΣ</u>

```
type semaphore = record

  count: integer;
  queue: ουρά διεργασιών;
end;
```

```
var s1, s2, s3, s4, s5, s6, s7, s8: semaphore;
s1 = s2 = s3 = s4 = s5 = s6 = s7 = s8 = 0;

cobegin
   repeat
      begin E1; up(s1); end;
      begin down(s1); E2; up(s2); end;
      begin down(s1); E3; up(s3); end;
      begin down(s1); E4; up(s4); end;
      begin down(s1); E5; up(s5); end;
      begin down(s2); down(s3); E6; E7; E8; E9; up(s6); end;
      begin down(s4); down(s6); E10; up(s7); end;
      begin down(s4); down(s15); E11; up(s8); end;
      begin down(s7); down(s8); E12; end;
   forever;
coend;
```

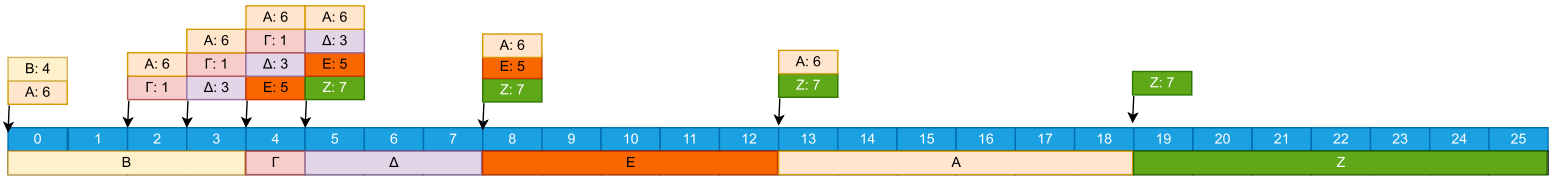
5. Χρονοπρογραμματισμός Διεργασιών

α. Ο αλγόριθμος FCFS λαμβάνει υπόψιν του μόνο το χρόνο άφιξης της κάθε διεργασίας και τις εκτελεί με βάση αυτό. Ο αλγόριθμος περιμένει έως ότου να ολοκληρωθεί η προηγούμενη και έπειτα συνεχίζει στην εκτέλεση της επόμενης, χωρίς να διακόπτεται η εκτέλεση της προηγούμενης.



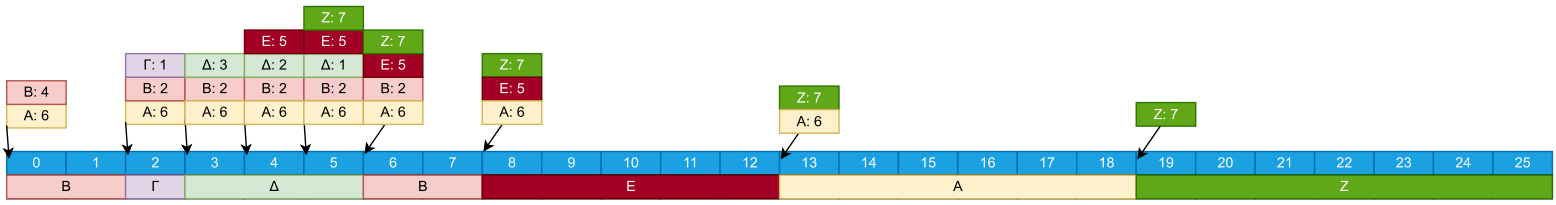
Εικόνα 1 Εκτέλεση FCFS

Για τον αλγόριθμο SJF παρατηρούμε ότι οι χρόνοι άφιξης των διεργασιών είναι διαφορετικοί και συνεπώς εκτελείται ο αλγόριθμος SJFP. Σύμφωνα με αυτόν, εκτελούνται πρώτα οι διεργασίες που έχουν μικρότερη διάρκεια εκτέλεσης κάθε φορά που ολοκληρώνεται μία διεργασία. Στην αρχή, οι διεργασίες Α και Β, για παράδειγμα, τοποθετούνται στην ούρα και πρώτα εκτελείται η Β αφού έχει χρόνο εκτέλεσης μικρότερο σε σχέση με την Α. Καθόλη την διάρκεια εκτέλεσης της διεργασίας Β, αφικνούνται κι άλλες διεργασίες. Μετά το πέρας των $3^{ων}$ χρονικών στιγμών που απαιτούνται για την ολοκλήρωση της Β, ελέγχουμε ποια από τις διεργασίες που έχουν φτάσει έχει το μικρότερο χρόνο εκτέλεσης και έτσι επιλέγεται η Γ, κ.ο.κ.



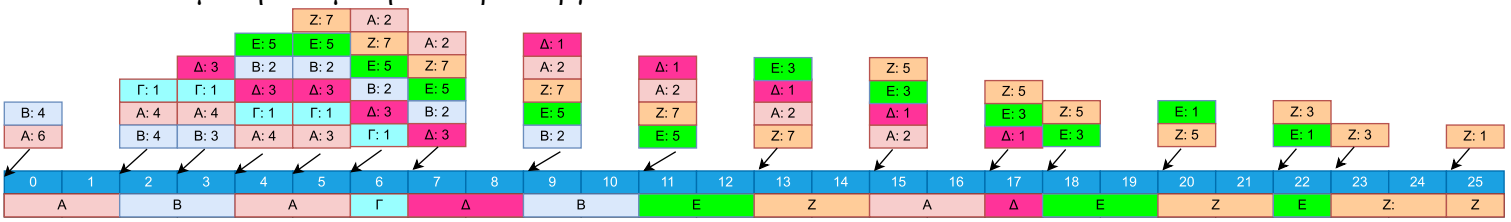
Εικόνα 2 Εκτέλεση SJF

Ο αλγόριθμος SRTF ελέγχει, κάθε φορά που μια διεργασία αφικνείται ή κάθε φορά που ολοκληρώνεται κάποια άλλη, τον εναπομείναντα χρόνο για την διεκπεραίωση κάθε διεργασίας και επιλέγεται η μικρότερης διάρκειας.



Εικόνα 3 Εκτέλεση SRTF

Ο αλγόριθμος RR εκτελεί τις διεργασίες με την σειρά που γίνεται η άφιξη τους αλλά μόνο για κάθε μία συγκεκριμένη χρονική διάρκεια και έπειτα γίνεται εναλλαγή με την επόμενη σε σειρά διεργασία.



Εικόνα 4 Εκτέλεση RR

β.

FCFS				
ΔΙΕΡΓΑΣΙΕΣ	ΧΡΟΝΟΣ ΑΝΑΜΟΝΗΣ	ΧΡΟΝΟΣ ΑΠΟΚΡΙΣΗΣ	ΧΡΟΝΟΣ ΟΛΟΚΛΗΡΩΣΗΣ	ΘΕΜΑΤΙΚΕΣ ΕΝΑΛΛΑΓΕΣ
Α	6 - 6 = 0	0 – 0 = 0	6 - 0 = 6	1
Β	10 - 4 = 6	6 – 0 = 6	10 - 0 = 10	1
Γ	9 - 1 = 8	10 – 2 = 8	11 - 2 = 9	1
Δ	11 - 3 = 8	11 – 3 = 8	14 - 3 = 11	1
Ε	15 - 5 = 10	14 - 4 = 10	19 - 4 = 15	1
Ζ	21 - 7 = 14	19 – 5 = 14	26 - 5 = 21	1
	46 / 6 = **7.67**	49 / 6 = **7.67**	72 / 6 = **12**	6 / 6 = **1**

SJF				
ΔΙΕΡΓΑΣΙΕΣ	ΧΡΟΝΟΣ ΑΝΑΜΟΝΗΣ	ΧΡΟΝΟΣ ΑΠΟΚΡΙΣΗΣ	ΧΡΟΝΟΣ ΟΛΟΚΛΗΡΩΣΗΣ	ΘΕΜΑΤΙΚΕΣ ΕΝΑΛΛΑΓΕΣ
Α	19 – 6 = 13	13 – 0 = 13	19 – 0 = 19	1
Β	4 – 4 = 0	0 – 0 = 0	4 – 0 = 4	1
Γ	3 – 1 = 2	4 – 2 = 2	5 – 2 = 3	1
Δ	5 – 3 = 2	5 – 3 = 2	8 – 3 = 5	1
Ε	9 – 5 = 4	8 – 4 = 4	13 – 4 = 9	1
Ζ	21 – 7 = 14	19 – 5 = 14	26 – 5 = 21	1
	35 / 6 = **5.83**	35 / 6 = **5.83**	61 / 6 = **10.16**	6 / 6 = **1**

SRTF				
ΔΙΕΡΓΑΣΙΕΣ	ΧΡΟΝΟΣ ΑΝΑΜΟΝΗΣ	ΧΡΟΝΟΣ ΑΠΟΚΡΙΣΗΣ	ΧΡΟΝΟΣ ΟΛΟΚΛΗΡΩΣΗΣ	ΘΕΜΑΤΙΚΕΣ ΕΝΑΛΛΑΓΕΣ
Α	19 – 6 = 13	13 – 0 = 13	19 – 0 = 19	1
Β	8 – 4 = 4	0 – 0 = 0	8 – 0 = 8	2
Γ	1 – 1 = 0	2 – 2 = 0	3 – 2 = 1	1
Δ	3 – 3 = 0	3 – 3 = 0	6 – 3 = 3	1
Ε	9 – 5 = 4	8 – 4 = 4	13 – 4 = 9	1
Ζ	21 – 7 = 14	19 – 5 = 14	26 – 5 = 21	1
	35 / 6 = **5.83**	31 / 6 = **5.16**	61 / 6 = **10.16**	7 / 6 = **1.16**

RR				
ΔΙΕΡΓΑΣΙΕΣ	ΧΡΟΝΟΣ ΑΝΑΜΟΝΗΣ	ΧΡΟΝΟΣ ΑΠΟΚΡΙΣΗΣ	ΧΡΟΝΟΣ ΟΛΟΚΛΗΡΩΣΗΣ	ΘΕΜΑΤΙΚΕΣ ΕΝΑΛΛΑΓΕΣ
Α	17 – 6 = 11	0 – 0 = 0	17 – 0 = 17	3
Β	13 – 4 = 9	2 – 0 = 2	13 – 0 = 13	2
Γ	5 – 1 = 4	6 – 2 = 4	7 – 2 = 5	1
Δ	15 – 3 = 12	7 – 3 = 5	18 – 3 = 15	2
Ε	19 – 5 = 14	11 – 4 = 7	23 – 4 = 19	3
Ζ	21 – 7 = 14	13 – 5 = 8	26 – 5 = 21	4
	64 / 6 = **10.6**	26 / 6 = **4.3**	90 / 6 = **15**	15 / 6 = **2.5**

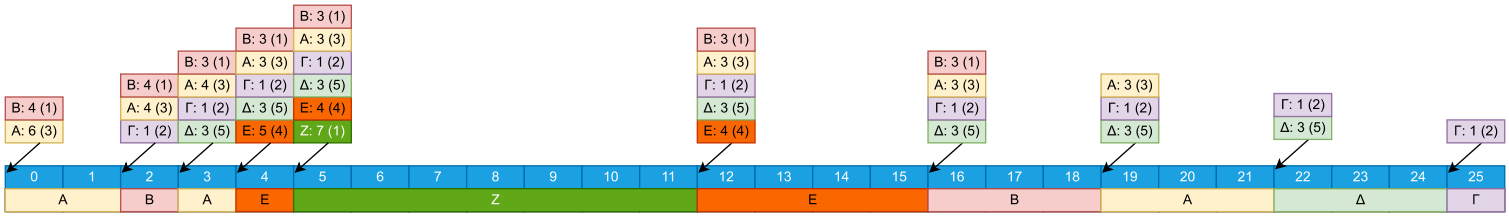
γ. Συνοψίζοντας έχουμε:

ΑΛΓΟΡΙΘΜΟΣ	**ΜΕΣΟΣ ΧΡΟΝΟΣ ΔΙΕΚΠΕΡΑΙΩΣΗΣ**
FCFS	12
SJF	10.16
SRTF	11.5
RR	15

Και συνεπώς το ποσοστό μεταβολής διαμορφώνεται ως εξής:

ΑΛΓΟΡΙΘΜΟΣ	**ΠΟΣΟΣΤΟ ΜΕΤΑΒΟΛΗΣ**		
SRTF	(11.5 – 12) / 12 = -0.04	->	**- 4%**
SJF	(10.16 – 12) / 12 = -0.15	->	**-15%**
RR	(15 – 12) / 12 = 0.25	->	**25%**

δ. Κάθε φορά που μια νέα διεργασία μπαίνει στην ουρά γίνεται έλεγχος για την επιλογή της επόμενης προς εκτέλεση διεργασίας. Στην ουρά υπάρχουν οι διεργασίες, δίπλα σε αυτές αναγράφεται ο χρόνος εκτέλεσης και σε παρένθεση το PID. Την χρονική στιγμή 0, υπάρχουν οι διεργασίες Α και Β, με διαφορετικό χρόνο εκτέλεσης και γι' αυτό επιλέγεται προς εκτέλεση η διεργασία Α καθώς αυτή απαιτεί 6 εναπομείναντα χρονικές μονάδες έναντι της Β που απαιτεί μόνο 4. Έπειτα από 2 χρονικές μονάδες μπαίνει στην ουρά η διεργασία Γ και ελέγχεται εκ νέου ποια απ' όλες θα εκτελεστεί. Οι διεργασίες Α και Β έχουν τον μέγιστο (ίσο) εναπομείναντα χρόνο και συνεπώς η επιλογή θα πρέπει να γίνει με βάση το PID και γι' αυτό επιλέγεται η διεργασία Β προς εκτέλεση. Και έτσι συνεχίζει ο αλγόριθμος.



Εικόνα 5 Εκτέλεση LRTFP

LRTFP				
ΔΙΕΡΓΑΣΙΕΣ	ΧΡΟΝΟΣ ΑΝΑΜΟΝΗΣ	ΧΡΟΝΟΣ ΑΠΟΚΡΙΣΗΣ	ΧΡΟΝΟΣ ΟΛΟΚΛΗΡΩΣΗΣ	ΘΕΜΑΤΙΚΕΣ ΕΝΑΛΛΑΓΕΣ
Α	22 – 6 = 16	0 – 0 = 0	22 – 0 = 22	3
Β	19 – 4 = 15	2 – 0 = 2	19 – 0 = 19	2
Γ	24 – 1 = 23	25 – 2 = 23	26 – 2 = 24	1
Δ	22 – 3 = 19	22 – 3 = 19	25 – 3 = 22	1
Ε	12 – 5 = 7	4 – 4 = 0	16 – 4 = 12	2
Ζ	7 – 7 = 0	5 – 5 = 0	12 – 5 = 7	1
	80 / 6 = **13.33**	44 / 6 = **7.33**	106 / 6 = **17.66**	10 / 6 = **1.66**