Λειτουργικά Συστήματα

2022 - 2023

1η Εργαστηριακή Άσκηση

Μέλη Ομάδας:

1) Όνομα : Αγγουρά Ρουμπίνη Μαρία
   ΑΜ: 1084634
   Email: up1084634@upnet.gr

2) Όνομα : Παυλόπουλος Ιάσονας
   ΑΜ: 1084565
   Email: up1084565@upnet.gr

Περιεχόμενα:

Εισαγωγή

Πριν ξεκινήσουμε την παρουσίαση του Project, θα θέλαμε να σας ενημερώσουμε πως όλα τα resource files, όπως το αρχείο με τον κώδικα για την άσκηση 2, το logparsher.sh, συμπεριλαμβανομένης και αυτής της αναφοράς μπορείτε να τα βρείτε στο παρακάτω Github Repository. Επίσης μπορείτε να δείτε τις αλλαγές που έχουν γίνει από την αρχή δημιουργίας του Project (commits).

Github Repository: https://github.com/Roumpini21/Project-1-OperatingSystems

Ερώτημα 1

i. Για την εκτέλεση αυτού του ερωτήματος, δημιουργήσαμε ένα case για το πρώτο argument ($1), που αν αυτό είναι κενό (“”), εκτυπώνει στην οθόνη τα ΑΜ μας.

```
jason@jason-virtualbox:~/Project$ ./logparser.sh
1084634 | 1084565
```

ii. Για αυτό το ερώτημα, προσθέσαμε στο ήδη υπάρχον case, για το πρώτο argument, μία ακόμα περίπτωση, δηλαδή να είναι ίσο με το access.log. Και μετά από αυτό φτιάξαμε ένα case και για το δεύτερο argument ($2), και ορίσαμε ότι αν είναι κενό (“”), δηλαδή αν δεν υπάρχει τίποτα μετά το access.log, να εκτυπώνει στην οθόνη τα περιεχόμενα όλου του αρχείου.
Ακόμα προσθέσαμε και ένα default case για το πρώτο argument, το οποίο ελέγχει αν το αρχείο που θα δοθεί τελειώνει σε .log, αν δεν τελειώνει σε αυτό, εκτυπώνει την φράση Invalid Arguments. Αυτό το πετύχαμε χρησιμοποιώντας wildcard characters (**), για να ταιριάξουμε μια συγκεκριμένη φράση μέσα σε ένα string.

```
jason@jason-virtualbox:~/Project$ ./logparser.sh access.log
::1 - - [13/Oct/2022:03:23:21 +0300] "POST /phpmyadmin/index.php?route=/lint HTTP/1.1" 200 28
::1 - - [13/Oct/2022:03:47:16 +0300] "POST /phpmyadmin/index.php?route=/ HTTP/1.1" 200 1659
::1 - - [13/Oct/2022:03:59:55 +0300] "POST /phpmyadmin/index.php?route=/ HTTP/1.1" 200 1659
::1 - - [13/Oct/2022:03:59:57 +0300] "POST /phpmyadmin/index.php?route=/ HTTP/1.1" 200 1659
::1 - - [13/Oct/2022:04:23:57 +0300] "POST /phpmyadmin/index.php?route=/ HTTP/1.1" 200 1659
::1 - - [13/Oct/2022:04:47:08 +0300] "POST /phpmyadmin/index.php?route=/ HTTP/1.1" 200 1659
::1 - - [13/Oct/2022:11:34:12 +0300] "POST /phpmyadmin/index.php?route=/ HTTP/1.1" 200 1659
::1 - - [13/Oct/2022:11:52:25 +0300] "POST /phpmyadmin/index.php?route=/ HTTP/1.1" 200 1659
::1 - - [13/Oct/2022:12:05:22 +0300] "POST /phpmyadmin/index.php?route=/ HTTP/1.1" 200 1659
::1 - - [13/Oct/2022:12:18:06 +0300] "POST /phpmyadmin/index.php?route=/ HTTP/1.1" 200 1659
::1 - - [13/Oct/2022:12:30:38 +0300] "POST /phpmyadmin/index.php?route=/ HTTP/1.1" 200 1659
::1 - - [13/Oct/2022:12:42:57 +0300] "POST /phpmyadmin/index.php?route=/ HTTP/1.1" 200 1659
::1 - - [13/Oct/2022:12:55:04 +0300] "POST /phpmyadmin/index.php?route=/ HTTP/1.1" 200 1659
::1 - - [13/Oct/2022:13:06:59 +0300] "POST /phpmyadmin/index.php?route=/ HTTP/1.1" 200 1659
::1 - - [13/Oct/2022:13:42:16 +0300] "POST /phpmyadmin/index.php?route=/ HTTP/1.1" 200 1659
::1 - - [13/Oct/2022:13:53:49 +0300] "POST /phpmyadmin/index.php?route=/ HTTP/1.1" 200 1659
::1 - - [13/Oct/2022:14:05:12 +0300] "POST /phpmyadmin/index.php?route=/ HTTP/1.1" 200 1659
::1 - - [13/Oct/2022:14:16:24 +0300] "POST /phpmyadmin/index.php?route=/ HTTP/1.1" 200 1659
::1 - - [13/Oct/2022:14:27:24 +0300] "POST /phpmyadmin/index.php?route=/ HTTP/1.1" 200 1659
::1 - - [15/Oct/2022:19:27:55 +0300] "GET /MyDocs/ HTTP/1.1" 200 3948
```

```
jason@jason-virtualbox:~/Project$ ./logparser.sh access.lo
Invalid Arguments
```

iii. Στο case για το δεύτερο argument που δημιουργήσαμε στο δεύτερο ερώτημα, θα προσθέσουμε μία περίπτωση, η οποία θα αναγνωρίζει αν το δεύτερο argument είναι ίσο με –-usrid. Δημιουργούμε ένα ακόμα case για το τρίτο argument καθώς θα μας χρειαστεί παρακάτω, και ορίζουμε ότι αν αυτό είναι κενό (“”), τότε να καλείται η συνάρτηση mining_usernames για το πρώτο argument, δηλαδή στην περίπτωση μας, για το αρχείο access.log.
Η συνάρτηση mining_usernames, με την βοήθεια του awk, κοιτάει την τρίτη στήλη του αρχείου και δημιουργεί έναν πίνακα. Για κάθε καινούρια λέξη που εντοπίζει στο αρχείο δημιουργεί μια καινούρια γραμμή σε αυτόν τον πίνακα, και με κάθε επανάληψη λέξης που έχει ξανασυναντήσει αυξάνει το count αυτής της λέξης κατά ένα. Έτσι στο τέλος θα έχει μετρήσει τις εμφανίσεις όλων των χρηστών. Έπειτα με ένα for loop και ένα pipe, εμφανίζει πόσες φορές έχει βρει κάθε χρήστη και το όνομα του. Το pipe πιο συγκεκριμένα, με την χρήση του -k 2 ταξινομεί με αλφαβητική σειρά την δεύτερη στήλη της εκτύπωσης μας.

```
mining_usernames(){
awk '{count[$3]++} END {for (word in count) print count[word], word}' $1 | sort -k 2
}
```

```
jason@jason-virtualbox:~/Project$ ./logparser.sh access.log --usrid
15710 -
124 admin
34 president
181 root
110 user1
91 user2
39 user3
```

Έπειτα εκμεταλλευόμενοι το case για το τρίτο argument που δημιουργήσαμε προηγουμένως, του προσθέτουμε και άλλες περιπτώσεις ανάλογα με τον user που θέλουμε να επιλέξουμε. Για κάθε user, εντοπίζει με την χρήση awk, τις γραμμές στις οποίες υπάρχει το username του στο αρχείο στην Τρίτη στήλη(στην οποία γίνεται γενικά η αναγραφή των usernames) και τυπώνει όλες τις γραμμές που εντόπισε.

Παραθέτουμε παράδειγμα για τον κενό χρήστη (-).

```
jason@jason-virtualbox:~/Project$ ./logparser.sh access.log --usrid -
::1 - - [13/Oct/2022:03:23:21 +0300] "POST /phpmyadmin/index.php?route=/lint HTTP/1.1" 200 28
::1 - - [13/Oct/2022:03:47:16 +0300] "POST /phpmyadmin/index.php?route=/ HTTP/1.1" 200 1659
::1 - - [13/Oct/2022:03:59:55 +0300] "POST /phpmyadmin/index.php?route=/ HTTP/1.1" 200 1659
::1 - - [13/Oct/2022:03:59:57 +0300] "POST /phpmyadmin/index.php?route=/ HTTP/1.1" 200 1659
::1 - - [13/Oct/2022:04:23:57 +0300] "POST /phpmyadmin/index.php?route=/ HTTP/1.1" 200 1659
::1 - - [13/Oct/2022:04:47:08 +0300] "POST /phpmyadmin/index.php?route=/ HTTP/1.1" 200 1659
::1 - - [13/Oct/2022:11:34:12 +0300] "POST /phpmyadmin/index.php?route=/ HTTP/1.1" 200 1659
::1 - - [13/Oct/2022:11:52:25 +0300] "POST /phpmyadmin/index.php?route=/ HTTP/1.1" 200 1659
::1 - - [13/Oct/2022:12:05:22 +0300] "POST /phpmyadmin/index.php?route=/ HTTP/1.1" 200 1659
::1 - - [13/Oct/2022:12:18:06 +0300] "POST /phpmyadmin/index.php?route=/ HTTP/1.1" 200 1659
::1 - - [13/Oct/2022:12:30:38 +0300] "POST /phpmyadmin/index.php?route=/ HTTP/1.1" 200 1659
::1 - - [13/Oct/2022:12:42:57 +0300] "POST /phpmyadmin/index.php?route=/ HTTP/1.1" 200 1659
::1 - - [13/Oct/2022:12:55:04 +0300] "POST /phpmyadmin/index.php?route=/ HTTP/1.1" 200 1659
::1 - - [13/Oct/2022:13:06:59 +0300] "POST /phpmyadmin/index.php?route=/ HTTP/1.1" 200 1659
::1 - - [13/Oct/2022:13:42:16 +0300] "POST /phpmyadmin/index.php?route=/ HTTP/1.1" 200 1659
::1 - - [13/Oct/2022:13:53:49 +0300] "POST /phpmyadmin/index.php?route=/ HTTP/1.1" 200 1659
::1 - - [13/Oct/2022:14:05:12 +0300] "POST /phpmyadmin/index.php?route=/ HTTP/1.1" 200 1659
::1 - - [13/Oct/2022:14:16:24 +0300] "POST /phpmyadmin/index.php?route=/ HTTP/1.1" 200 1659
::1 - - [13/Oct/2022:14:27:24 +0300] "POST /phpmyadmin/index.php?route=/ HTTP/1.1" 200 1659
::1 - - [15/Oct/2022:19:27:55 +0300] "GET /MyDocs/ HTTP/1.1" 200 3948
::1 - - [15/Oct/2022:19:27:55 +0300] "GET /icons/blank.gif HTTP/1.1" 200 148
::1 - - [15/Oct/2022:19:27:55 +0300] "GET /icons/back.gif HTTP/1.1" 200 216
::1 - - [15/Oct/2022:19:27:55 +0300] "GET /icons/folder.gif HTTP/1.1" 200 225
::1 - - [15/Oct/2022:19:27:57 +0300] "GET /MyDocs/Vathmoi/ HTTP/1.1" 200 517
::1 - - [15/Oct/2022:19:27:59 +0300] "GET /MyDocs/ HTTP/1.1" 200 3948
```

iv. Επιστρέφοντας στις περιπτώσεις του case για το δεύτερο argument, προσθέτουμε μία ακόμα που εντοπίζει το -method. Προσθέτουμε έπειτα άλλο ένα case για το τρίτο argument που εντοπίζει τι θα γραφτεί μετά από το -method. Ανάλογα με το αν θα εντοπίσει το GET ή το POST, εκτυπώνει με όλες τις γραμμές στις οποίες υπάρχει εμφάνισή τους, με την χρήση του awk. Ακόμα προσθέτουμε ένα default case που αν δοθεί λάθος όνομα method ή καθόλου method, θα τυπώσει την φράση Wrong Method Name.

```
jason@jason-virtualbox:~/Project$ ./logparser.sh access.log -method GET
::1 - - [15/Oct/2022:19:27:55 +0300] "GET /MyDocs/ HTTP/1.1" 200 3948
::1 - - [15/Oct/2022:19:27:55 +0300] "GET /icons/blank.gif HTTP/1.1" 200 148
::1 - - [15/Oct/2022:19:27:55 +0300] "GET /icons/back.gif HTTP/1.1" 200 216
::1 - - [15/Oct/2022:19:27:55 +0300] "GET /icons/folder.gif HTTP/1.1" 200 225
::1 - - [15/Oct/2022:19:27:57 +0300] "GET /MyDocs/Vathmoi/ HTTP/1.1" 200 517
::1 - - [15/Oct/2022:19:27:59 +0300] "GET /MyDocs/ HTTP/1.1" 200 3948
::1 - - [15/Oct/2022:19:28:04 +0300] "GET /MyDocs/Software HTTP/1.1" 301 350
::1 - - [15/Oct/2022:19:28:04 +0300] "GET /MyDocs/Software/ HTTP/1.1" 200 519
::1 - - [15/Oct/2022:19:28:08 +0300] "GET /MyDocs/Software/after.php HTTP/1.1" 200 393
::1 - - [15/Oct/2022:19:28:10 +0300] "GET /MyDocs/Software/ HTTP/1.1" 200 519
```

```
jason@jason-virtualbox:~/Project$ ./logparser.sh access.log -method
Wrong Method Name
jason@jason-virtualbox:~/Project$ ./logparser.sh access.log -method Get
Wrong Method Name
```

v. Επιστρέφοντας στις περιπτώσεις του case για το δεύτερο argument, προσθέτουμε ακόμα μία περίπτωση στην οποία θα εντοπίζει το –servprot. Ακριβώς όπως και πριν με την χρήση case τρίτου argument και awk, εντοπίζουμε σε ποιες γραμμές υπάρχει το πρωτόκολλο που δόθηκε και τυπώνουμε αυτές τις γραμμές στις οθόνη. Σε περίπτωση λάθους, τυπώνεται στην οθόνη λόγω default case η φράση Wrong Network Protocol.

```
jason@jason-virtualbox:~/Project$ ./logparser.sh access.log --servprot IPv4
127.0.0.1 - - [20/Jul/2022:19:06:42 +0300] "GET /MyDocs/AEKX/en/searchsinglecrcindexes.php HTTP/1.1" 200 14220 "http://localhost/MyDocs/AEKX/
en/searchqualindexes.php" "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:102.0) Gecko/20100101 Firefox/102.0"
127.0.0.1 - - [20/Jul/2022:19:06:43 +0300] "POST /MyDocs/AEKX/en/resusearccrc.php HTTP/1.1" 200 182 "http://localhost/MyDocs/AEKX/en/searchs
nglecrcindexes.php" "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:102.0) Gecko/20100101 Firefox/102.0"
127.0.0.1 - - [20/Jul/2022:19:07:28 +0300] "POST /MyDocs/AEKX/en/resusearccrc.php HTTP/1.1" 200 18862 "http://localhost/MyDocs/AEKX/en/searc
singlecrcindexes.php" "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:102.0) Gecko/20100101 Firefox/102.0"
127.0.0.1 - - [02/Oct/2022:12:28:52 +0300] "GET /MyDocs/IPEM/ HTTP/1.1" 200 3587 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:105.0) Ge
ko/20100101 Firefox/105.0"
127.0.0.1 - - [02/Oct/2022:12:28:52 +0300] "GET /MyDocs/IPEM/images/favicon-32x32.png HTTP/1.1" 404 296 "http://localhost/MyDocs/IPEM/" "Moz
lla/5.0 (Windows NT 10.0; Win64; x64; rv:105.0) Gecko/20100101 Firefox/105.0"
127.0.0.1 - - [02/Oct/2022:12:28:52 +0300] "GET /MyDocs/IPEM/images/favicon-16x16.png HTTP/1.1" 404 296 "http://localhost/MyDocs/IPEM/" "Moz
lla/5.0 (Windows NT 10.0; Win64; x64; rv:105.0) Gecko/20100101 Firefox/105.0"
jason@jason-virtualbox:~/Project$ ./logparser.sh access.log --servprot
Wrong Network Protocol
jason@jason-virtualbox:~/Project$ ./logparser.sh access.log --servprot ipv7
Wrong Network Protocol
```

vi. Επιστρέφοντας στο case για το δεύτερο argument, προσθέτουμε μια περίπτωση που μετράει πόσες φορές χρησιμοποιήθηκε ο κάθε browser. Για την υλοποίηση αυτού, δημιουργήσαμε την συνάρτηση count_browsers. Αυτή η συνάρτηση με την χρήση awk και της συνάρτησης match(), ελέγχει για κάθε γραμμή του κώδικα αν εμφανίζεται κάποιος browser στις στήλες 12-25. Στο τέλος αποθηκεύει τις εμφανίσεις τους σε μία μεταβλητή για τον καθένα, και εκτυπώνει στην οθόνη τα αποτελέσματα.

```
jason@jason-virtualbox:~/Project/Project$ ./logparser.sh access.log --browsers
Mozilla: 1057
Chrome: 124
Safari: 124
Edge: 63
```

vii. Επιστρέφοντας στο case για το δεύτερο argument, προσθέτουμε μια περίπτωση που αναγνωρίζει το –datum. Έπειτα προσθέτουμε και case για το τρίτο argument του οποίου του δίνουμε την δυνατότητα να αναγνωρίσει πολλά διαφορετικά strings στην ίδια περίπτωση και έπειτα με την χρήση grep να εκτυπώσει τις γραμμές στις οποίες εμφανίστηκε το string που αναγνώρισε. Τέλος βάζουμε για ακόμα μια φορά default case, στο οποίο αν δεν αναγνωριστεί κάποιο από τα strings που δηλώσαμε, εμφανίζει την φράση Wrong Date.

```
jason@jason-virtualbox:~/Project$ ./logparser.sh access.log --datum Jan
::1 - - [29/Jan/2022:15:52:30 +0300] "GET /phpmyadmin/js/vendor/codemirror/addon/runmode/runmode.js?v=5.1.1 HTTP/1.1" 200 2773
::1 - president - [29/Jan/2022:15:52:30 +0300] "GET /phpmyadmin/js/vendor/codemirror/mode/sql/sql.js?v=5.1.1 HTTP/1.1" 200 47998
::1 - user3 - [29/Jan/2022:15:52:30 +0300] "GET /phpmyadmin/js/vendor/codemirror/addon/hint/show-hint.js?v=5.1.1 HTTP/1.1" 200 19089
::1 - user1 - [29/Jan/2022:15:52:30 +0300] "GET /phpmyadmin/js/vendor/codemirror/addon/hint/sql-hint.js?v=5.1.1 HTTP/1.1" 200 9604
::1 - - [29/Jan/2022:15:52:30 +0300] "GET /phpmyadmin/js/vendor/codemirror/addon/lint/lint.js?v=5.1.1 HTTP/1.1" 200 9125
::1 - root - [29/Jan/2022:15:52:30 +0300] "GET /phpmyadmin/js/dist/codemirror/addon/lint/sql-lint.js?v=5.1.1 HTTP/1.1" 200 959
::1 - admin - [29/Jan/2022:15:52:30 +0300] "GET /phpmyadmin/js/dist/console.js?v=5.1.1 HTTP/1.1" 200 49656
::1 - - [29/Jan/2022:15:52:30 +0300] "GET /phpmyadmin/js/messages.php?l=en&v=5.1.1&lang=en HTTP/1.1" 200 7921
::1 - - [29/Jan/2022:15:52:30 +0300] "GET /phpmyadmin/js/vendor/codemirror/lib/codemirror.js?v=5.1.1 HTTP/1.1" 200 398364
::1 - - [29/Jan/2022:15:52:30 +0300] "GET /phpmyadmin/themes/pmahomme/img/logo_left.png HTTP/1.1" 200 2713
jason@jason-virtualbox:~/Project$ ./logparser.sh access.log --datum Ja
Wrong Date
jason@jason-virtualbox:~/Project$ ./logparser.sh access.log --datum
Wrong Date
```

Ερώτημα 2

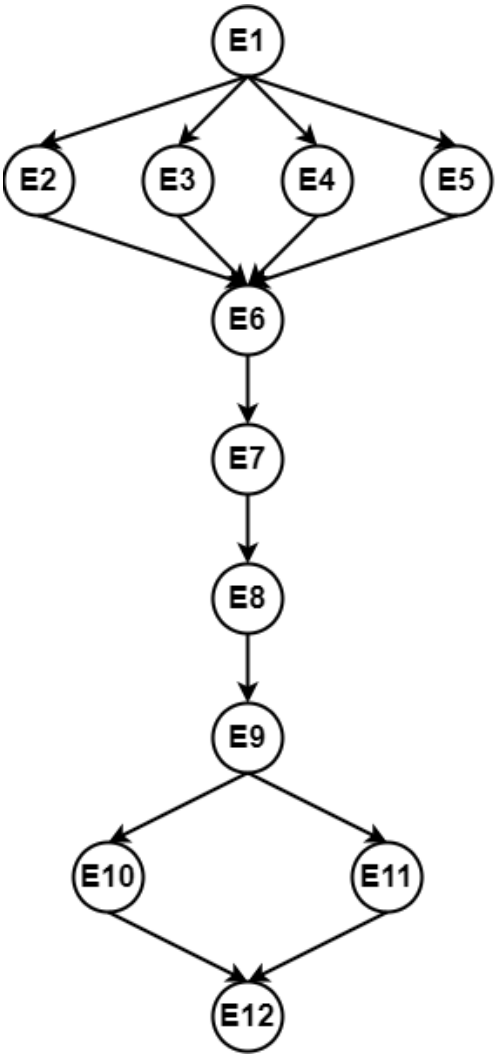
Στο συγκεκριμένο ερώτημα, για την υλοποίηση των ζητουμένων ορίστηκε η struct mess για να αποθηκευτεί το μήνυμα, που περιέχει μία μεταβλητή long "mestype" και μία μεταβλητή double "res". Έπειτα δημιουργήσαμε την συνάρτηση child, στην οποία χρησιμοποιήθηκε η msgsnd για να στείλουμε το μήνυμα. Στην main, χρησιμοποιήθηκε η msgrcv για να λάβουμε το μήνυμα, με ορίσματα την msqid (που χρησιμοποιείται για να αρχικοποιήσουμε το message queue και να πάρουμε το id της ουράς), το struct “message” και ότι ο τύπος μηνυμάτων θα είναι 1 όπως θέσαμε στην main. Επίσης, για να υλοποιήσουμε το δυναμικό όρισμα του αριθμού διεργασιών, δημιουργήσαμε την μεταβλητή proc_num. Αρχικά λάβαμε υπόψη τις περιπτώσεις ο χρήστης να μην εισάγει αριθμό διεργασιών, ή να εισάγει αριθμό μικρότερο ή ίσο του 0. Αν συμβούν οποιεσδήποτε από τις παραπάνω περιπτώσεις εμφανίζεται ανάλογο μήνυμα. Τέλος, με ένα for loop που τρέχει όσες φορές όσες η μεταβλητή proc_num δηλαδή ο αριθμός διεργασιών, παίρνουμε το pid των διεργασιών κάνοντας fork() και καλούμε την child με ορίσματα τις μεταβλητές που είχαμε από το αρχείο που μας δινόταν για τον υπολογισμό του ολοκληρώματος και το msqid.

Παράδειγμα εκτέλεσης για 4 διεργασίες:

```
jason@jason-virtualbox:~/Project$ ./secondtask 4
Time=43.427401 seconds, Result=29.01414811
```

Ερώτημα 3

α)



Για την δημιουργία του σχήματος ακολουθούμε αυστηρά τις οδηγίες της εκφώνησης και όχι την λογική συνέχεια των διεργασιών.

Με βάση την εκφώνηση παρατηρούμε ότι πρώτη πρέπει απαραίτητα να υλοποιηθεί η Διεργασία Ε1, καθώς δεν γίνεται να περάσουμε στην υλοποίηση κάποιας άλλης αν δεν έχει γίνει πρώτα αυτή.

Στην συνέχεια παρατηρούμε πως οι Διεργασίες Ε2, Ε3, Ε4 και Ε5 δεν εξαρτώνται η μία από την άλλη και μπορούν να γίνουν με όποια σειρά επιθυμούμε. Έτσι θα μπουν στο διάγραμμα κάτω από την πρώτη και παράλληλα μεταξύ τους.

Για να υλοποιηθεί η Διεργασία Ε6, πρέπει να έχουν υλοποιηθεί οι Διεργασίες Ε2, Ε3,Ε4 και Ε5, οπότε και την τοποθετούμε κάτω από αυτές.

Οι Διεργασίες Ε7, Ε8, Ε9 θα τοποθετηθούν η μία κάτω από την άλλη αφού χρειάζεται να υλοποιηθούν με συγκεκριμένη σειρά.

Η Διεργασίες Ε10 και Ε11, μπορούν να υλοποιηθούν μετά από την Ε9 αλλά και παράλληλα μεταξύ τους, καθώς στην εκφώνηση δεν υπάρχει κάποιος όρος που να δείχνει ότι υλοποιούνται η μία μετά από την άλλη. (πχ να γράφει: στην συνέχεια ή ακολουθεί).

Και τέλος η Ε12 υλοποιείται τελευταία από όλες, αφού προϋποθέτει την προετοιμασία όλων των φιαλιδίων και χαρτόκουτων.

β)

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

γ) var s1,s2,s3,s4,s5,s6,s7,s8,s9,s10,s11,s12,s13,s14,s15 : semaphores;

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

cobegin

begin E1; up(s1); up(s2); up(s3); up(s4); end;

begin down(s1); E2; up(s5); end;

begin down(s1); E3; up(s6); end;

begin down(s1); E4; up(s7); end;

begin down(s1); E4; up(s8); end;

begin down(s5); down(s6); down(s7); down(s8); E6; up(s9); end;

begin down(s9); E7; up(s10); end;

begin down(s10); E8; up(s11); end;

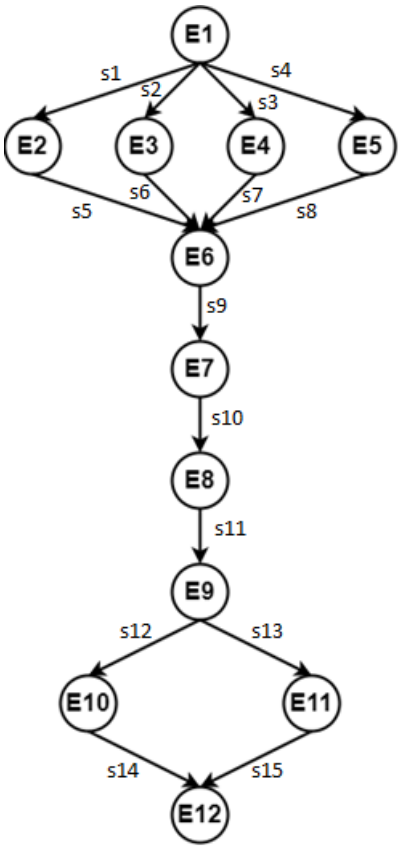
begin down(s11); E9; up(s12); up(s13); end;

begin down(s12); E10; up(s14); end;

begin down(s13); E11; up(s15); end;

begin down(s14); down(s15); E12; end;

coend



δ) var s1, s2 : semaphores;

s1 = s2 = 0;

cobegin

begin E1; up(s1); end;

begin down(s1); E2; up(s2); end;

begin down(s1); E3; up(s2); end;

begin down(s1); E4; up(s2); end;

begin down(s1); E5; up(s2); end;

begin down(s2); E6; up(s1); end;

begin down(s1); E7; up(s2); end;

begin down(s2); E8; up(s1); end;

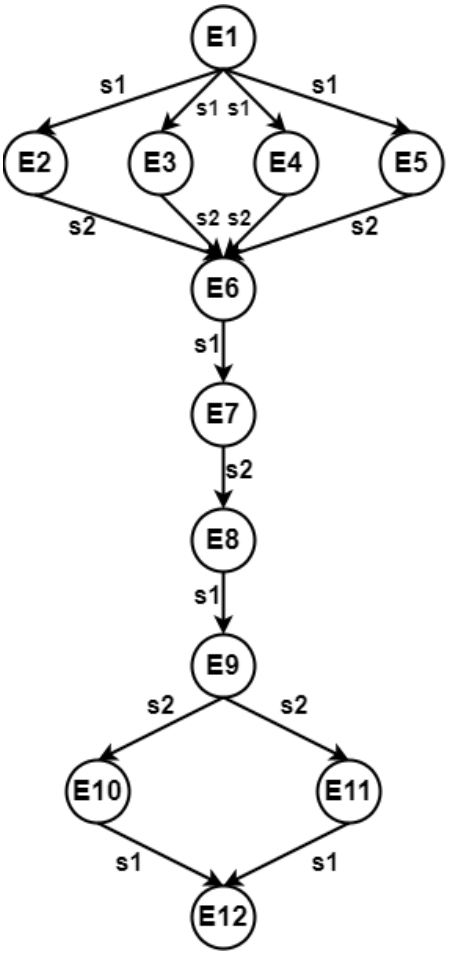
begin down(s1); E9; up(s2); end;

begin down(s2); E10; up(s1); end;

begin down(s2); E11; up(s1); end;

begin down(s1); E12; end;

coend

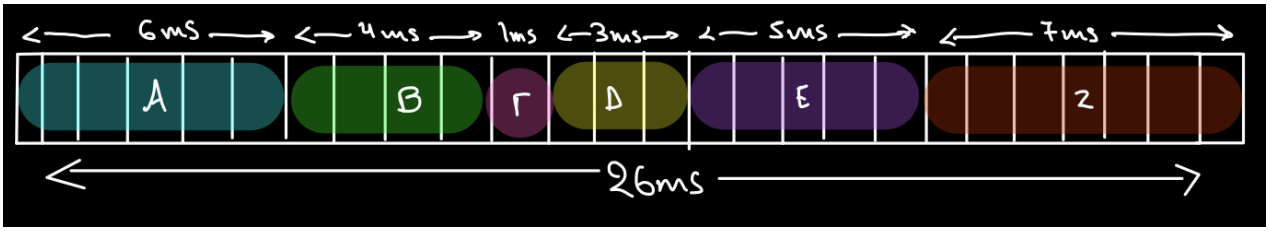


Ερώτημα 4

α)

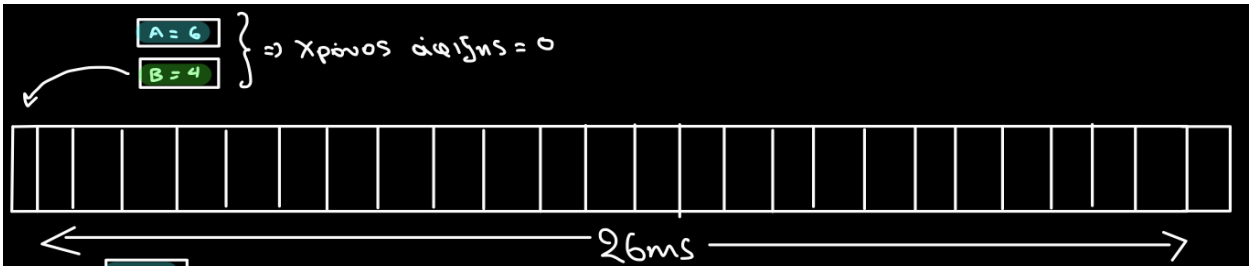
- <u>FCFS (First Come First Served)</u>

Στον αλγόριθμο FCFS υλοποιείται κάθε φορά η διεργασία που βρίσκεται πρώτη στην σειρά. Για αυτόν τον λόγο το διάγραμμα Gantt είναι αρκετά απλό καθώς οι διεργασίες υλοποιούνται απλά με την σειρά.

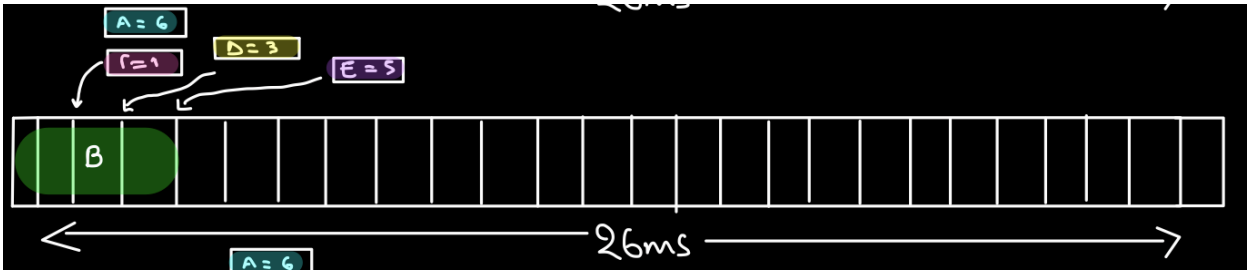


- <u>SJF (Shortest Job First)</u>

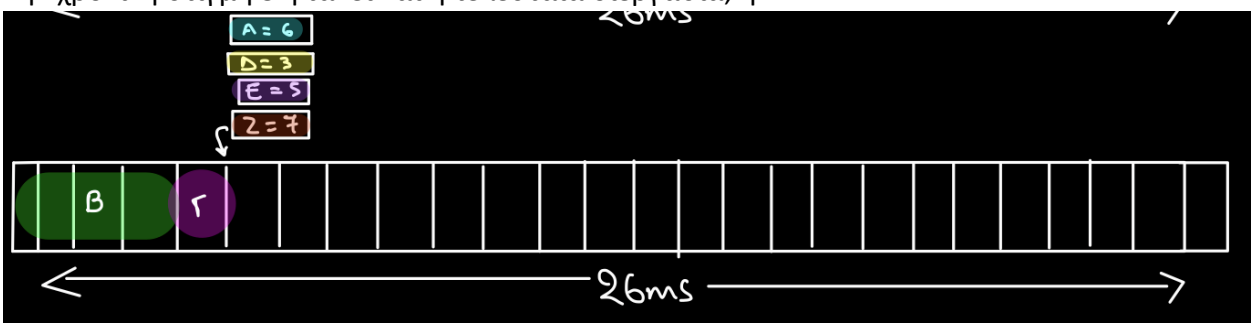
Στον αλγόριθμο SJF υλοποιείται κάθε φορά η διεργασία που έχει τον μικρότερο χρόνο εκτέλεσης. Ξεκινώντας, την χρονική στιγμή 0 φτάνουν οι Διεργασίες Α και Β.



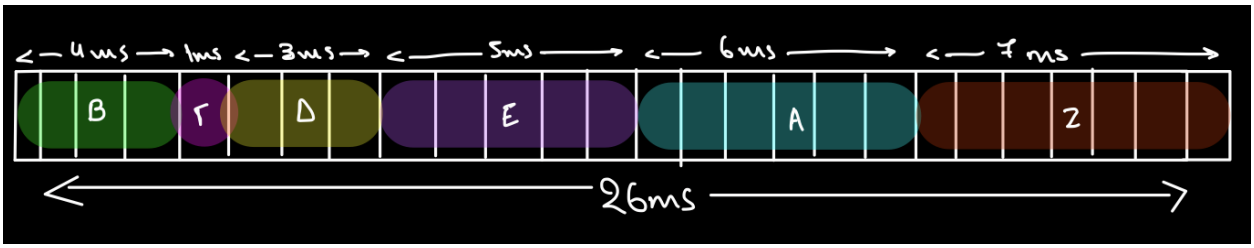
Συγκρίνοντας τες συμπεραίνουμε πως αυτή με τον μικρότερο χρόνο εκτέλεσης είναι η Β. Έτσι την προσθέτουμε πρώτη στο διάγραμμα. Την χρονική στιγμή 2 φτάνει η διεργασία Γ, την χρονική στιγμή φτάνει η διεργασία Δ, ενώ την χρονική στιγμή 4 φτάνει η διεργασία Ε.



Συγκρίνοντας τες συμπεραίνουμε ότι αυτή με τον μικρότερο χρόνο εκτέλεσης είναι η Γ. Έτσι την προσθέτουμε στο διάγραμμα.
Την χρονική στιγμή 5 φτάνει και η τελευταία διεργασία, η Z.

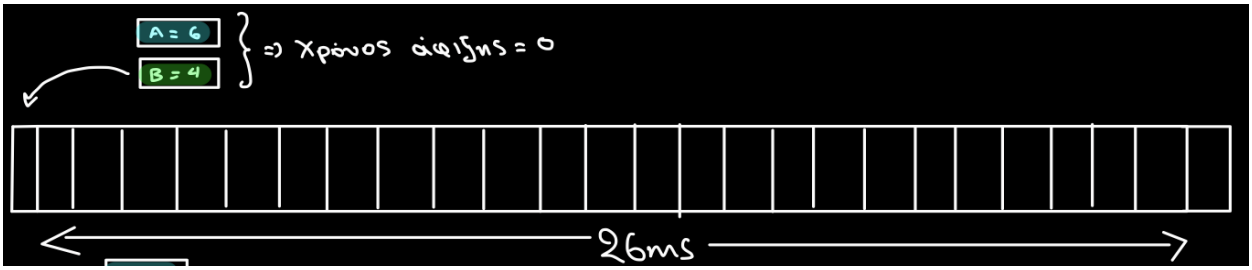


Στην συνέχεια, από την στιγμή που έχουν φτάσει όλες οι διεργασίες, απλά προσθέτουμε τις διεργασίες με την σειρά χρόνου εκτέλεσης. Δηλαδή Ε, Α και έπειτα την Ζ.
Η τελική μορφή του διαγράμματος Gantt είναι η εξής:

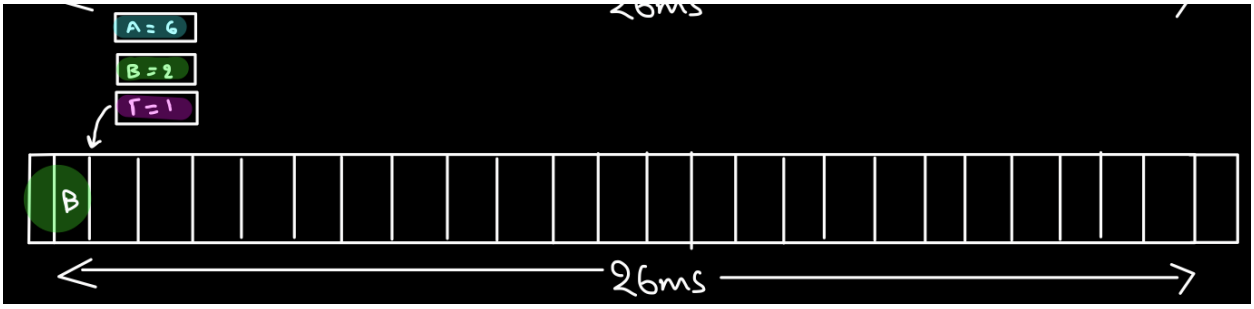


- <u>SRTF (Shortest Remaining Time First)</u>

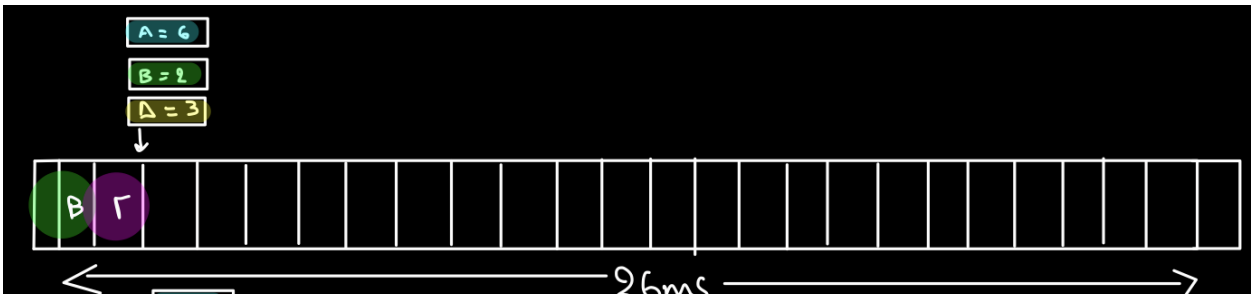
Στον αλγόριθμο SRTF οι διεργασίες υλοποιούνται με βάση τον εναπομείναντα χρόνο εκτέλεσης.
Ξεκινώντας, την χρονική στιγμή 0 φτάνουν οι Διεργασίες Α και Β.



Παρατηρούμε, πως μικρότερο εναπομείναντα χρόνο εκτέλεσης έχει η διεργασία Β.
Άρα ξεκινάμε με την εκτέλεση της.

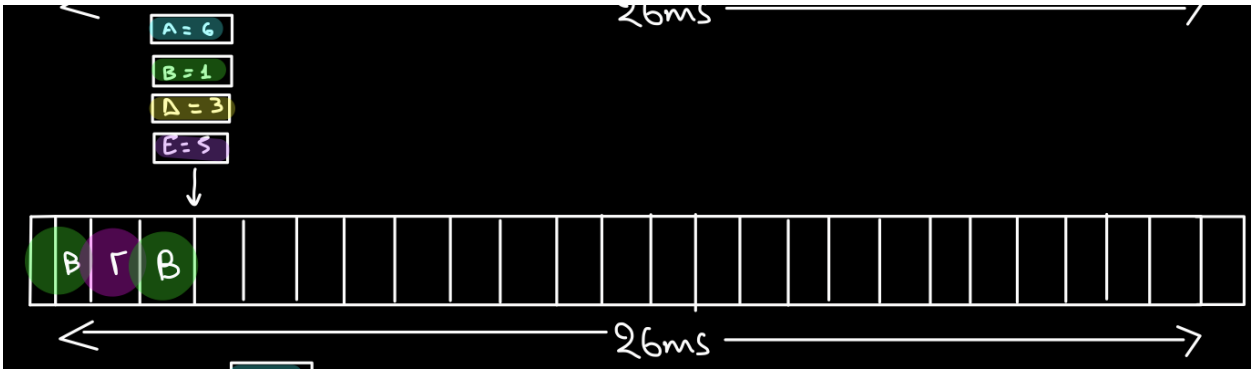


Την χρονική στιγμή 2 σταματάμε την εκτέλεση της Β, καθώς φτάνει η διεργασία Γ. Και πρέπει να συγκρίνουμε με αυτήν τον εναπομείναντα χρόνο της Β, για να δούμε ποια θα συνεχίσει να εκτελείται. Παρατηρούμε πως στην Β απομένουν 2ms, ενώ στην Γ μόνο 1ms. Έτσι ξεκινά την εκτέλεση της η Γ.
Η Γ τελειώνει την εκτέλεση της την χρονική στιγμή 3 και ταυτόχρονα φτάνει η διεργασία Δ.



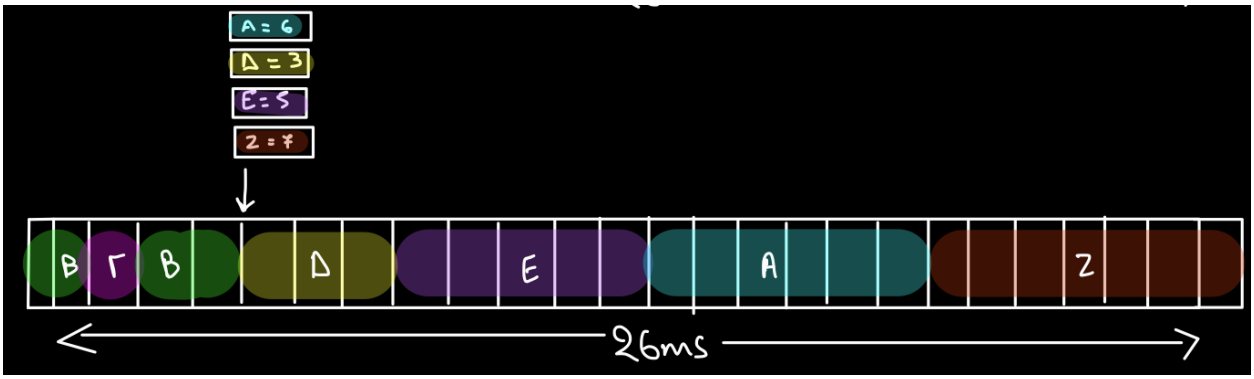
Συνεχίζοντας, θα ξεκινήσει να εκτελείται η διεργασία Β, καθώς είναι αυτή με τον μικρότερο εναπομείναντα χρόνο εκτέλεσης.

Η εκτέλεση της θα διακοπεί μετά από 1ms καθώς θα φτάσει η διεργασία Ε και πρέπει να ελέγξουμε ποια έχει τον μικρότερο εναπομείναντα χρόνο εκτέλεσης. Στην Β τώρα απομένει μόνο 1ms.



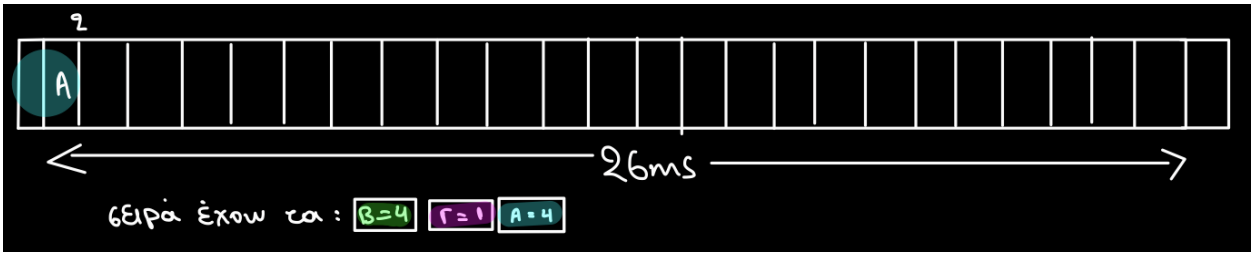
Συγκρίνοντας τους χρόνους που απομένουν για κάθε διεργασία, παρατηρούμε πως αυτή με τον μικρότερο χρόνο είναι και πάλι η Β, οπότε συνεχίζουμε την εκτέλεση της.
Την χρονική στιγμή 5 τελειώνει η Β την εκτέλεση της, και ταυτόχρονα φτάνει και η τελευταία διεργασία, η Ζ. Τις συγκρίνουμε όλες ξανά και παρατηρούμε πως η επόμενη που θα εκτελεστεί είναι η διεργασία Δ, καθώς έχει τον μικρότερο υπολειπόμενο χρόνο. Στην συνέχεια από την στιγμή που δεν περιμένουμε να φτάσει κάποια άλλη διεργασία, οι υπόλοιπες υλοποιούνται με την παρακάτω σειρά.
Δ -> Ε -> Α -> Ζ

Το τελικό διάγραμμα Gantt είναι το εξής:

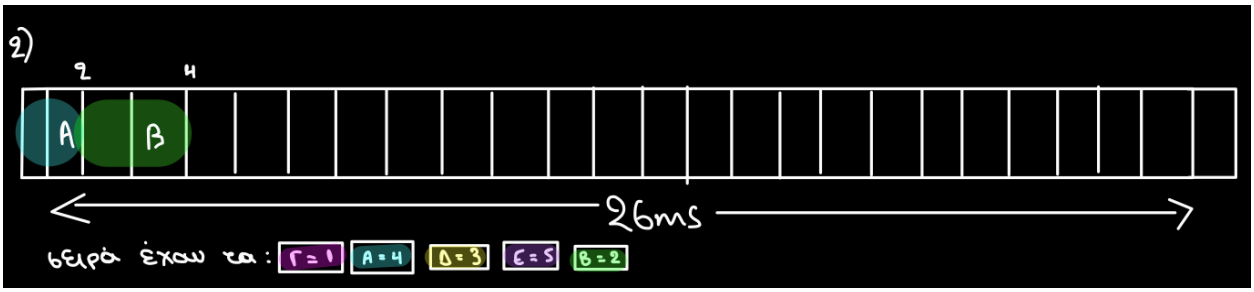


- <u>RR (Round Robin) με κβάντο χρόνου 2 χρονικές μονάδες</u>

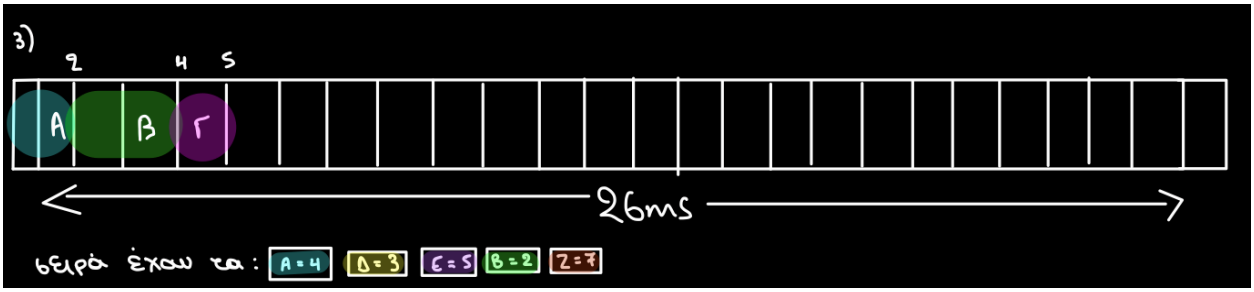
Στον αλγόριθμο Round Robin η κάθε διεργασία που επιλέγεται για εκτέλεση μπορεί να εκτελεστεί μόνο για 2 χρονικές μονάδες. Έπειτα πηγαίνει τελευταία στην ουρά προτεραιότητας και ξαναυλοποιείται για 2 χρονικές μονάδες όταν ξαναέρθει η σειρά της.
Όταν 2 διεργασίες φτάνουν ταυτόχρονα χρησιμοποιούμε τον FCFS για να επιλέξουμε ποια θα εκτελεστεί πρώτη.
Ξεκινώντας, την χρονική στιγμή 0 φτάνουν οι Διεργασίες Α και Β.
Με βάση τον FCFS, θα αρχίσει πρώτα την εκτέλεση της η Α. Θα εκτελεστεί για 2ms και έπειτα θα πάει να περιμένει στο τέλος της σειράς. Μπροστά της θα υπάρχει η Β, και η Γ που έφτασε την χρονική στιγμή 2.



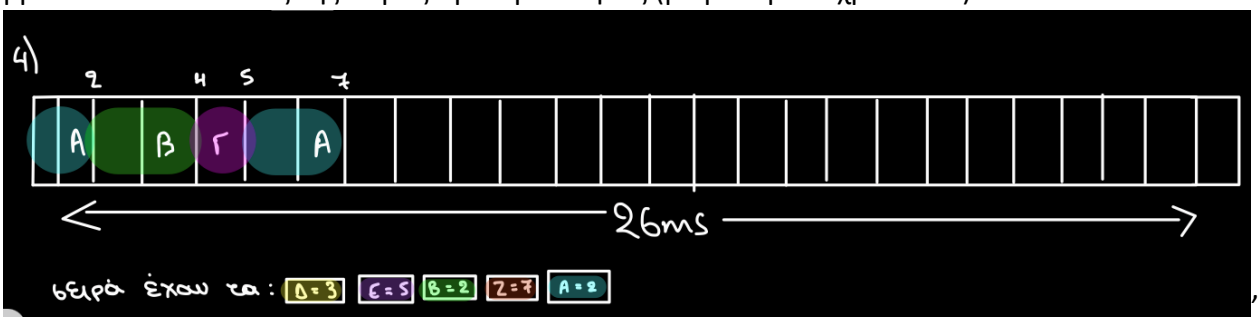
Έπειτα θα ξεκινήσει για 2ms την εκτέλεση της η διεργασία Β.
Όταν αυτά περάσουν θα πάει και θα περιμένει στο τέλος της ουράς προτεραιότητας.
Μπροστά της θα είναι οι Γ, η Α με μειωμένο χρόνο = 4, η Δ που έφτασε την χρονική στιγμή 3 και τέλος η Ε που έφτασε την χρονική στιγμή 4.



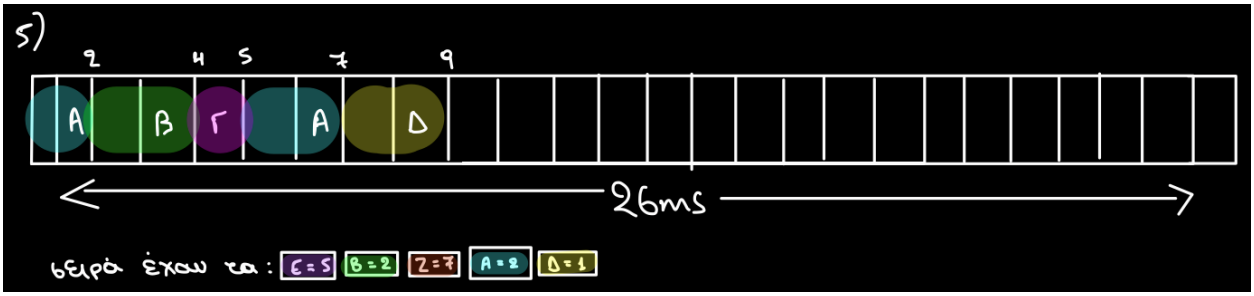
Επόμενη στην σειρά για εκτέλεση λοιπόν είναι η διεργασία Γ. Η διεργασία Γ θα εκτελεστεί μόνο για 1ms , αφού μόνο τόσος χρόνος απαιτείται για την εκτέλεση της.
Άρα την χρονική στιγμή 5ms θα σταματήσει την εκτέλεση της.
Στο τέλος της σειράς αυτήν την φορά θα μπει μόνο η διεργασία Z που έφτασε την χρονική στιγμή 5.



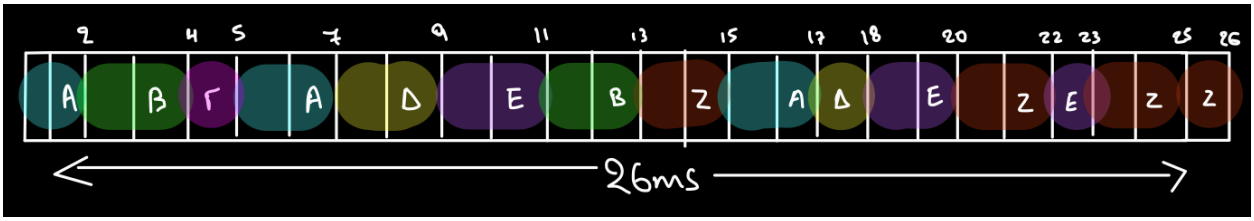
Έπειτα, είναι η σειρά της διεργασίας Α να εκτελεστεί. Θα εκτελεστεί για 2ms και έπειτα θα σταματήσει και θα βρεθεί πάλι στο τέλος της ουράς προτεραιότητας (με μειωμένο χρόνο = 2).



Έπειτα όπως φαίνεται και στο σχήμα είναι η σειρά της διεργασίας Δ να εκτελεστεί.



Αυτή η διαδικασία θα επαναληφθεί αρκετές φορές ακόμα μέχρι να εκτελεστούν όλες οι διεργασίες.
Το τελικό διάγραμμα Gantt είναι το εξής:



β) Για τα παρακάτω έστω ότι:

t1 = χρονική στιγμή εισόδου,

t2 = χρονική στιγμή εξόδου,

tcpu = χρόνος εκτέλεσης,

ΧΑ = Χρόνος Αναμονής,

ΜΧΑ = Μέσος Χρόνος Αναμονής,

tap = χρόνος απόκρισης,

ΜΧΑπ = Μέσος Χρόνος Απόκρισης

- <u>FCFS (First Come First Served)</u>

Μέσος Χρόνος Αναμονής:

ΧΑ(Α) = t2 – t1 – tcpu = 6 – 0 – 6 = 0ms,

ΧΑ(Β) = t2 – t1 – tcpu = 10 – 0 – 4 = 6ms,

ΧΑ(Γ) = t2 – t1 – tcpu = 11 – 2 – 1 = 8ms,

ΧΑ(Δ) = t2 – t1 – tcpu = 14 – 3 – 3 = 8ms,

ΧΑ(Ε) = t2 – t1 – tcpu = 19 – 4 – 5 = 10ms,

ΧΑ(Ζ) = t2 – t1 – tcpu = 26 – 5 – 7 = 14ms

ΜΧΑ = (0 + 6 + 8 + 8 + 10 + 14) / 6 = 7,66ms

Μέσος Χρόνος Απόκρισης:

ΧΑπ(Α) = tap – t1 = 0 – 0 = 0ms,

ΧΑπ(Β) = tap – t1 = 6 – 0 = 6ms,

ΧΑπ(Γ) = tap – t1 = 10 – 2 = 8ms,

ΧΑπ(Δ) = tap – t1 = 11 – 3 = 8ms,

ΧΑπ(Ε) = tap – t1 = 14 – 4 = 10ms,

ΧΑπ(Ζ) = tap – t1 = 19 – 5 = 14ms

ΜΧΑπ = (0 + 6 + 8 + 8 + 10 + 14)/6 = 7,66ms

Μέσος Χρόνος Ολοκλήρωσης:

ΧΟ(Α) = t2 – t1 = 6 – 0 = 6ms,

ΧΟ(Β) = t2 – t1 = 10 – 0 = 10ms,

ΧΟ(Γ) = t2 – t1 = 11 – 2 = 9ms,

ΧΟ(Δ) = t2 – t1 = 14 – 3 = 11ms,

ΧΟ(Ε) = t2 – t1 = 19 – 4 = 15ms,

ΧΟ(Ζ) = t2 – t1 = 26 – 5 = 21ms

ΜΧΟ = (6 + 10 + 9 + 11 + 15 + 21)/6 = 12ms

Πλήθος Θεματικών Εναλλαγών:

#Α = 1, #Β = 1, #Γ = 1, #Δ = 1, #Ε = 1, #Ζ = 1

- SJF (Shortest Job First)

Μέσος Χρόνος Αναμονής:

ΧΑ(Β) = t2 – t1 – tcpu = 4 – 0 – 4 = 0ms,
ΧΑ(Γ) = t2 – t1 – tcpu = 5 – 2 – 1 = 2ms,
ΧΑ(Δ) = t2 – t1 – tcpu = 8 – 3 – 3 = 2ms,
ΧΑ(Ε) = t2 – t1 – tcpu = 13 – 4 – 5 = 4ms,
ΧΑ(Α) = t2 – t1 – tcpu = 19 – 0 – 6 = 13ms,
ΧΑ(Ζ) = t2 – t1 – tcpu = 26 – 5 – 7 = 14ms
ΜΧΑ = (0 + 2 + 2 + 4 + 13 + 14) / 6 = 5,83ms

Μέσος Χρόνος Απόκρισης:

ΧΑπ(Β) = tap – t1 = 0 – 0 = 0ms,
ΧΑπ(Γ) = tap – t1 = 4 – 2 = 2ms,
ΧΑπ(Δ) = tap – t1 = 5 – 3 = 2ms,
ΧΑπ(Ε) = tap – t1 = 8 – 4 = 4ms,
ΧΑπ(Α) = tap – t1 = 13 – 0 = 13ms,
ΧΑπ(Ζ) = tap – t1 = 19 – 5 = 14ms
ΜΧΑπ = (0 + 2 + 2 + 4 + 13 + 14) / 6 = 5,83ms

Μέσος Χρόνος Ολοκλήρωσης:

ΧΟ(Α) = t2 – t1 = 19 – 0 = 19ms,
ΧΟ(Β) = t2 – t1 = 4 – 0 = 4ms,
ΧΟ(Γ) = t2 – t1 = 5 – 2 = 3ms,
ΧΟ(Δ) = t2 – t1 = 8 – 3 = 5ms,
ΧΟ(Ε) = t2 – t1 = 13 – 4 = 9ms,
ΧΟ(Ζ) = t2 – t1 = 26 – 5 = 21ms
ΜΧΟ = (19 + 4 + 3 + 5 + 9 + 21) / 6 = 10,1ms

Πλήθος Θεματικών Εναλλαγών:

#Α = 1, #Β = 1, #Γ = 1, #Δ = 1, #Ε = 1, #Ζ = 1

- SRTF (Shortest Remaining Time First)

Μέσος Χρόνος Αναμονής:

ΧΑ(Α) = t2 – t1 – tcpu = 19 – 0 – 6 = 13ms,
ΧΑ(Β) = t2 – t1 – tcpu = 5 – 0 – 4 = 1ms,
ΧΑ(Γ) = t2 – t1 – tcpu = 3 – 2 – 1 = 0ms,
ΧΑ(Δ) = t2 – t1 – tcpu = 8 – 3 – 3 = 2ms,
ΧΑ(Ε) = t2 – t1 – tcpu = 13 – 4 – 5 = 4ms,
ΧΑ(Ζ) = t2 – t1 – tcpu = 26 – 5 – 7 = 14ms
ΜΧΑ = (13 + 1 + 0 + 2 + 4 + 14) / 6 = 5,66ms

Μέσος Χρόνος Απόκρισης:

ΧΑπ(Α) = tap – t1 = 13 – 0 = 13ms,

ΧΑπ(Β) = tap – t1 = 0 – 0 = 0ms,

ΧΑπ(Γ) = tap – t1 = 2 – 2 = 0ms,

ΧΑπ(Δ) = tap – t1 = 5 – 3 = 2ms,

ΧΑπ(Ε) = tap – t1 = 8 – 4 = 4ms,

ΧΑπ(Ζ) = tap – t1 = 19 – 5 = 14ms

ΜΧΑπ = (13 + 0 + 0 + 2 + 4 + 14) / 6 = 5,5ms

Μέσος Χρόνος Ολοκλήρωσης:

ΧΟ(Α) = t2 – t1 = 19 – 0 = 19ms,

ΧΟ(Β) = t2 – t1 = 5 – 0 = 5ms,

ΧΟ(Γ) = t2 – t1 = 3 – 2 = 1ms,

ΧΟ(Δ) = t2 – t1 = 8 – 3 = 5ms,

ΧΟ(Ε) = t2 – t1 = 13 – 4 = 9ms,

ΧΟ(Ζ) = t2 – t1 = 26 – 5 = 21ms

ΜΧΟ = (19 + 5 + 1 + 5 + 9 + 21) / 6 = 10ms

Πλήθος Θεματικών Εναλλαγών:

#Α = 1, #Β = 2 (3), #Γ = 1, #Δ = 1, #Ε = 1, #Ζ = 1

- <u>RR (Round Robin) με κβάντο χρόνου 2 χρονικές μονάδες</u>

Μέσος Χρόνος Αναμονής:

ΧΑ(Α) = t2 – t1 – tcpu = 17 – 0 – 6 = 11ms,

ΧΑ(Β) = t2 – t1 – tcpu = 13 – 0 – 4 = 9ms,

ΧΑ(Γ) = t2 – t1 – tcpu = 5 – 2 –1 = 2ms,

ΧΑ(Δ) = t2 – t1 – tcpu = 18 – 3 – 3 = 12ms,

ΧΑ(Ε) = t2 – t1 – tcpu = 23 – 4 – 5 = 14ms,

ΧΑ(Ζ) = t2 – t1 – tcpu = 26 – 5 – 7 = 14ms

ΜΧΑ = (11 + 9 + 2 + 12 + 14 + 14) / 6 = 10,33ms

Μέσος Χρόνος Απόκρισης:

ΧΑπ(Α) = tap – t1 = 0 – 0 = 0ms,

ΧΑπ(Β) = tap – t1 = 2 – 0 = 2ms,

ΧΑπ(Γ) = tap – t1 = 4 – 2 = 2ms,

ΧΑπ(Δ) = tap – t1 = 7 – 3 = 4ms,

ΧΑπ(Ε) = tap – t1 = 9 – 4 = 5ms,

ΧΑπ(Ζ) = tap – t1 = 13 – 5 = 8ms

ΜΧΑπ = (0 + 2 + 2 + 4 + 5 + 8) / 6 = 3,5ms

Μέσος Χρόνος Ολοκλήρωσης:

ΧΟ(Α) = t2 – t1 = 17 – 0 = 17ms,

ΧΟ(Β) = t2 – t1 = 13 – 0 = 13ms,

ΧΟ(Γ) = t2 – t1 = 5 – 2 = 3ms,

ΧΟ(Δ) = t2 – t1 = 18 – 3 = 15ms,

ΧΟ(Ε) = t2 – t1 = 23 – 4 = 19ms,

ΧΟ(Ζ) = t2 – t1 = 26 – 5 = 21ms

ΜΧΟ = (17 + 13 + 3 + 15 + 19 + 21) / 6 = 14,66ms

Πλήθος Θεματικών Εναλλαγών:

#A = 3, #B = 2, #Γ = 1, #Δ = 2, #E = 3, #Z = 4

γ) Χρόνος Διεκπεραίωσης = Χρόνος Ολοκλήρωσης

Ο τύπος που θα χρησιμοποιήσουμε είναι {(ταλγ – tfcfs)/tfcfs} * 100%

Όπου ταλγ = ο χρόνος διεκπεραίωσης του SJF, SRTF και RR αντίστοιχα.

- FCFS – SJF
  {(tsjf – tfcfs)/tfcfs} * 100% = {(9,33 – 12)/12} * 100% = (-2,67/12)*100% = -22,25%

- FCFS – SRTF
  {(tsrtf – tfcfs)/tfcfs} * 100% = {(10 – 12)/12} * 100% = (-2/12)*100% = -16,66%

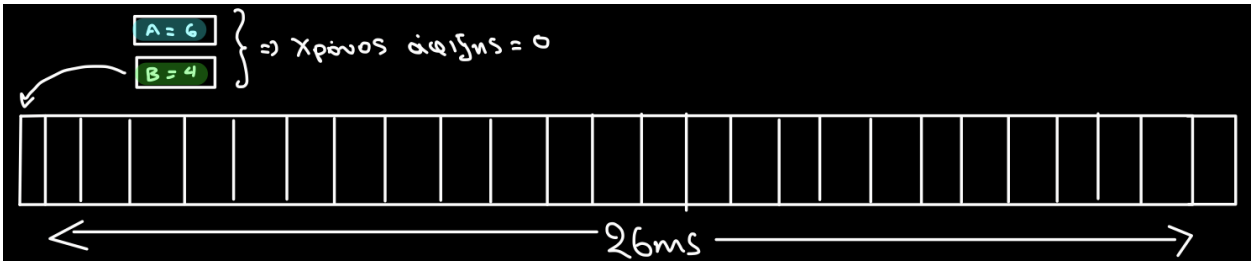
- FCFS – RR
  {(trr – tfcfs)/tfcfs} * 100% = {(14,66 – 12)/12} * 100% = (2,66/12)*100% = 22,16%

δ)

- Διάγραμμα Gantt

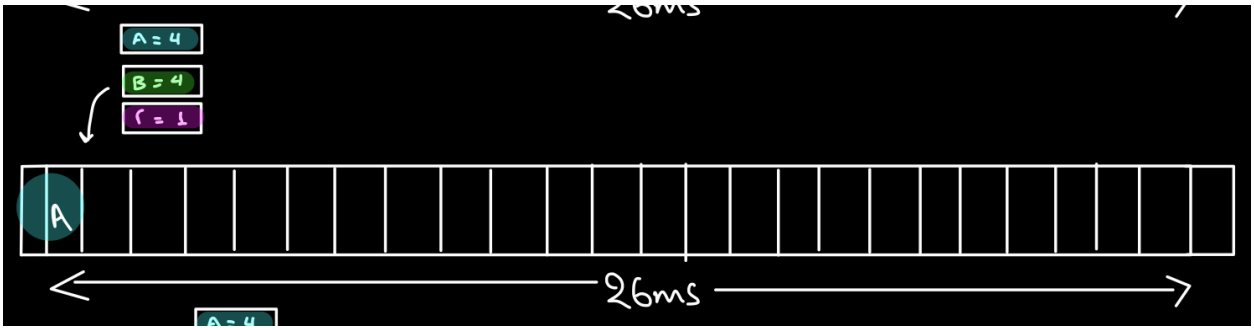
Στον αλγόριθμο LRTFP υλοποιείται κάθε φορά η διεργασία που έχει τον περισσότερο εναπομείναντα χρόνο. Για αυτόν τον λόγο, θα πρέπει μετά από κάθε χρονική μονάδα, δηλαδή μετά από κάθε ms, να σταματάμε για ν ελέγχουμε ποιο είναι αυτό που πληρεί το κριτήριο στην συγκεκριμένη περίπτωση.

Την χρονική στιγμή 0, φτάνουν στον επεξεργαστή οι διεργασίες Α και Β.

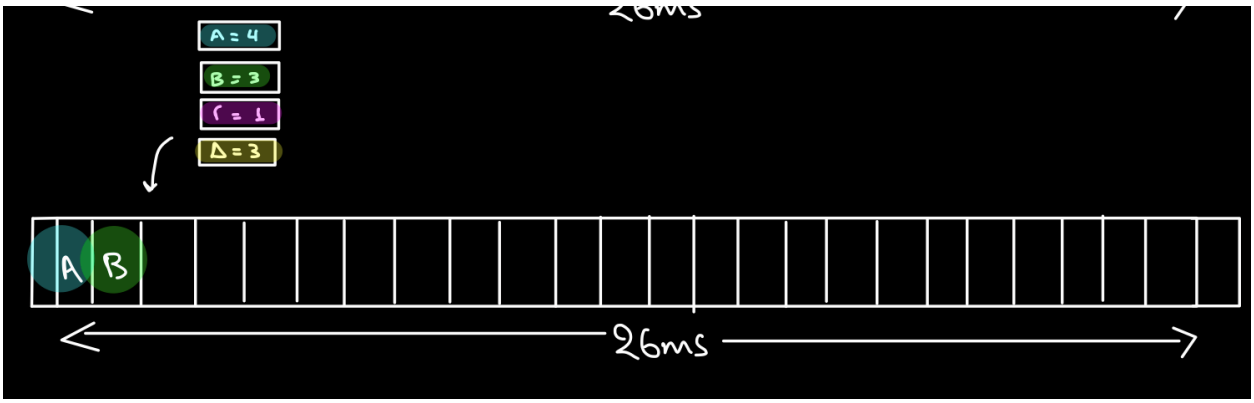


Αυτή με τον περισσότερο εναπομείναντα χρόνο είναι η Α. Την αφήνουμε να τρέξει για 1ms και έπειτα ξαναελέγχουμε. Παραμένει αυτή με τον περισσότερο χρόνο οπότε συνεχίζουμε την επεξεργασία της.

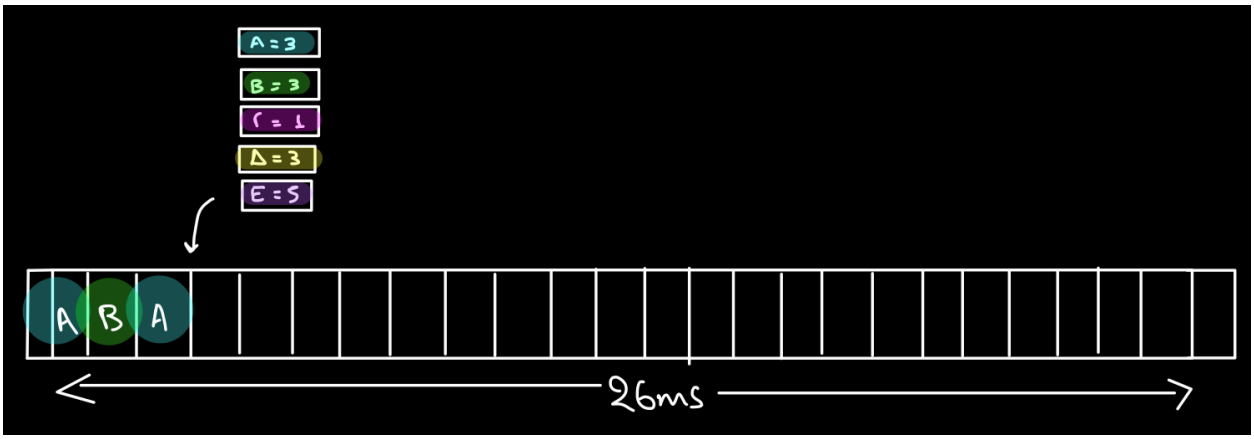
Την χρονική στιγμή 2, φτάνει στον επεξεργαστή η διεργασία Γ, η οποία έχει εναπομείναντα χρόνο = 1ms.



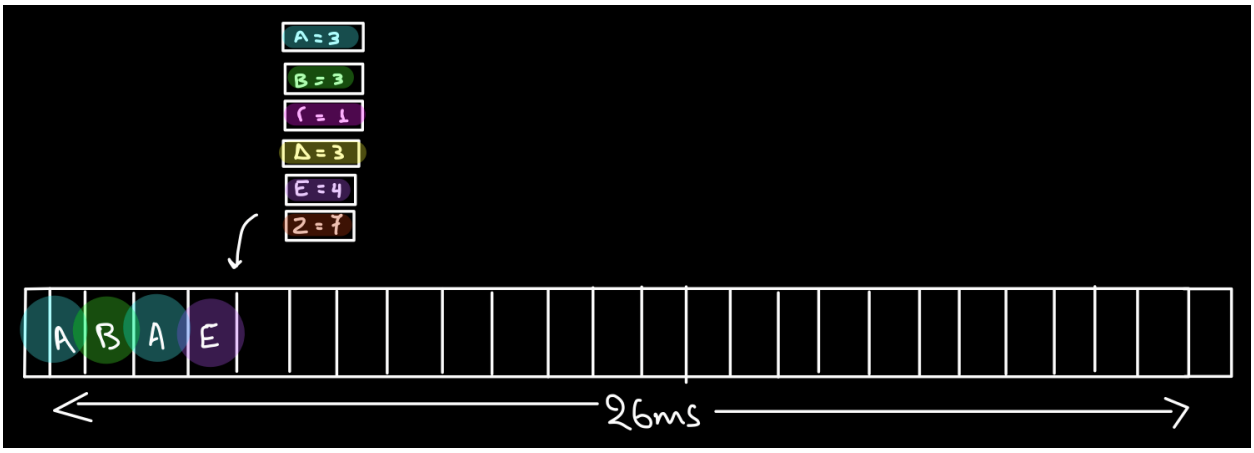
Ελέγχουμε ποια διεργασία θα επεξεργαστεί μετά. Παρατηρούμε πως η Α και η Β έχουν τον ίδιο εναπομείναντα χρόνο = 4. Για αυτόν τον λόγο θα κοιτάξουμε τα PID. Αυτή που έχει το μικρότερο θα εισέλθει πρώτη. Δηλαδή στην συγκεκριμένη περίπτωση η διεργασία Β. Όπως είπαμε και προηγουμένως, θα τρέξει μόνο για 1ms και μετά θα σταματήσει για να επαναληφθεί ο έλεγχος. Η Β λοιπόν σταματάει την χρονική στιγμή 3, και ταυτόχρονα φτάνει στον επεξεργαστή η διεργασία Δ, η οποία έχει εναπομείναντα χρόνο εκτέλεσης =3.



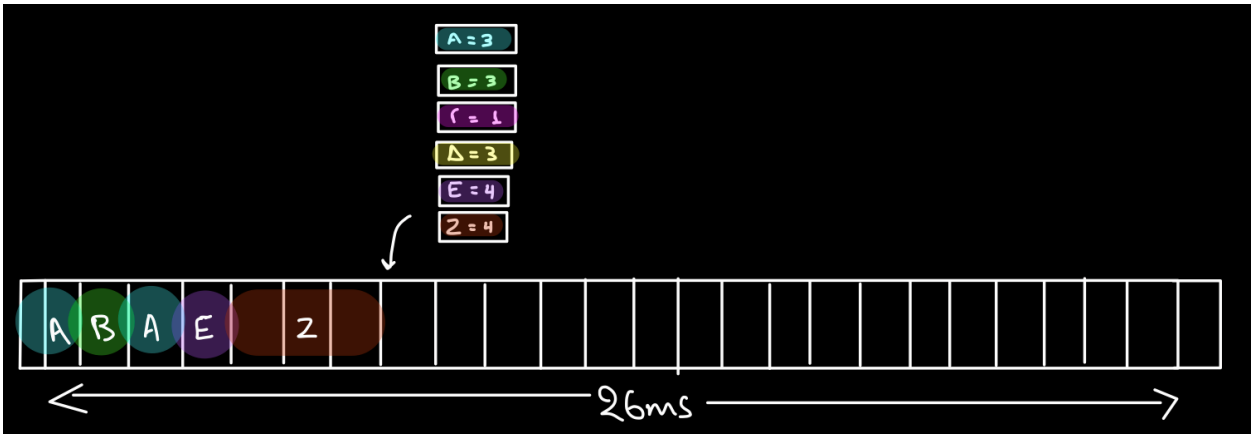
Συγκρίνουμε τον χρόνο που απομένει στην κάθε διεργασία που έχουμε. Αυτή με τον μεγαλύτερο είναι η Α, άρα αυτή θα είναι αυτή που θα εκτελεστεί. Εκτελείται για 1ms και έπειτα σταματάει. Φτάνει τότε και η Ε με χρόνο εκτέλεσης = 5.



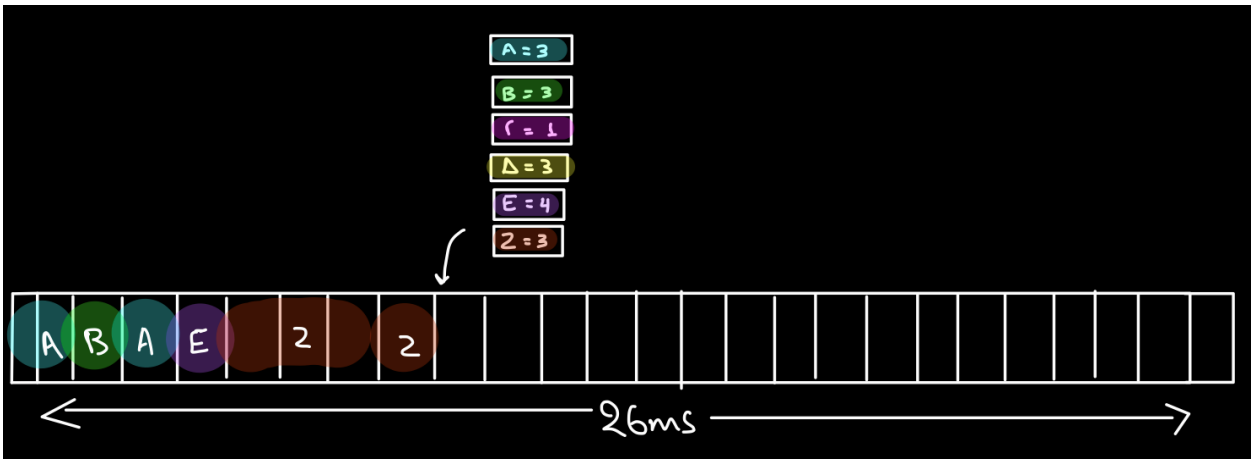
Συγκρίνουμε τον χρόνο που απομένει στην κάθε διεργασία που έχουμε. Αυτή με τον μεγαλύτερο είναι η Ε, άρα αυτή θα είναι αυτή που θα εκτελεστεί. Εκτελείται για 1ms και έπειτα σταματάει. Φτάνει τότε και η Z με χρόνο εκτέλεσης =7.



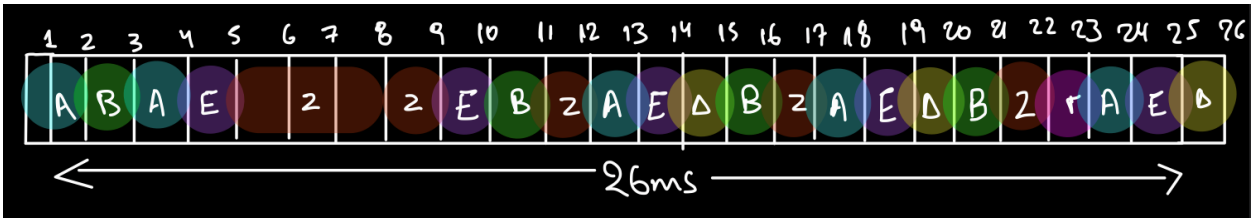
Συγκρίνουμε τον χρόνο που απομένει στην κάθε διεργασία που έχουμε. Αυτή με τον μεγαλύτερο είναι η Z, άρα αυτή θα είναι αυτή που θα εκτελεστεί. Εκτελείται για 1ms και έπειτα σταματάει. Ξαναγίνεται έλεγχος και παρατηρούμε πως η Z εξακολουθεί να είναι αυτή με τον μεγαλύτερο εναπομείναντα χρόνο. Αυτό επαναλαμβάνεται άλλη μια φορά. Μέχρι που καταλήγουμε να έχουμε 2 διεργασίες που έχουν ισοπαλία στον μεγαλύτερο εναπομείναντα χρόνο



Αυτές οι διεργασίες είναι οι Ε και Z. Οπότε θα πρέπει να συγκρίνουμε τα PIDs τους. Μικρότερο έχει η Z οπότε είναι και αυτή που θα συνεχίσει να εκτελείται.



Στην συνέχεια επαναλαμβάνουμε την ίδια διαδικασία και συγκρίσεις μέχρι να εκτελεστούν όλες οι διεργασίες πλήρως. Το τελικό διάγραμμα Gantt είναι το εξής:



- **Μέσος Χρόνος Αναμονής:**

ΧΑ(Α) = t2 – t1 – tcpu = 24 – 0 – 6 = 18ms,
ΧΑ(Β) = t2 – t1 – tcpu = 21 – 0 – 4 = 17ms,
ΧΑ(Γ) = t2 – t1 – tcpu = 23 – 2 –1 = 20ms,
ΧΑ(Δ) = t2 – t1 – tcpu = 26 – 3 – 3 = 20ms,
ΧΑ(Ε) = t2 – t1 – tcpu = 25 – 4 – 5 = 16ms,
ΧΑ(Ζ) = t2 – t1 – tcpu = 22 – 5 – 7 = 10ms,
ΜΧΑ = (18 + 17 + 20 + 20 + 16 + 10)/6= 16,83ms

- **Μέσος Χρόνος Απόκρισης:**

ΧΑπ(Α) = tap – t1 = 0 – 0 = 0ms,
ΧΑπ(Β) = tap – t1 = 2 – 0 = 2ms,
ΧΑπ(Γ) = tap – t1 = 22 – 2 = 20ms,
ΧΑπ(Δ) = tap – t1 = 14– 3 = 11ms,
ΧΑπ(Ε) = tap – t1 = 4 – 4 = 0ms,
ΧΑπ(Ζ) = tap – t1 = 5 – 5 = 0ms,
ΜΧΑπ = (0 + 2 + 20 + 11 + 0 + 0) / 6 = 5,5ms

- **Μέσος Χρόνος Ολοκλήρωσης:**

ΧΟ(Α) = t2 – t1 = 24 – 0 = 24ms,
ΧΟ(Β) = t2 – t1 = 21 – 0 = 21ms,
ΧΟ(Γ) = t2 – t1 =23 – 2 = 21ms,
ΧΟ(Δ) = t2 – t1 = 26 – 3 = 23ms,
ΧΟ(Ε) = t2 – t1 = 25 – 4 = 21ms,
ΧΟ(Ζ) = t2 – t1 = 22 – 5 = 17ms
ΜΧΟ = (24 + 21 + 21 + 23 + 21 + 217) / 6 = 21,16ms

- **Πλήθος Θεματικών Εναλλαγών:**

#Α = 5, #Β = 4, #Γ = 1, #Δ = 3, #Ε = 5, #Ζ = 7