

Λειτουργικά Συστήματα

2022 - 2023

1η Εργαστηριακή Άσκηση

- Γρηγόρης Δεληπαλταδάκης, AM: 1084647, email: up1084647@upnet.gr
- Δαμιανός Διασάκος, AM: 1084632, email: up1084632@upnet.gr
- Αλκιβιάδης Δασκαλάκης, AM: 1084673, email: up1084673@upnet.gr
- Ιάσων Ράικος, AM: 1084552, email: up1084552@upnet.gr

Περιεχόμενα

Ερώτημα 1: Shell Scripting.....	2
Ερώτημα 2: Διεργασίες.....	11
Ερώτημα 3: Διαδιεργασιακή Επικοινωνία.....	15
a).....	15
b).....	17
c).....	18
Ερώτημα 4: Χρονοπρογραμματισμός Διεργασιών	20
α) Διαγράμματα Gantt.....	20
β) Υπολογισμοί Χρόνων	21
γ) Ποσοστά μεταβολής του μέσου χρόνου διεκπεραίωσης.....	26
δ) LRTFP (Longest Remaining Time First Preemptive) scheduling policy	26

Ερώτημα 1: Shell Scripting

Προσοχή!!!: Ο συγκεκριμένος κώδικας για το shell scripting γράφτηκε και εκτελέστηκε στο Visual Studio Code. Επομένως, αν το τρέξετε στο VS CODE ακολουθείτε τα παρακάτω βήματα.

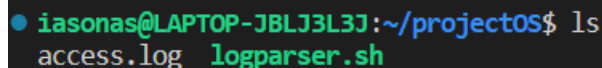
Σε περίπτωση που το τρέξετε σε Ubuntu τότε θα χρειαστεί να εκτελέσετε και τις παρακάτω εντολές πριν εκτελέσετε το logparser.sh:

```
sudo chmod +x logparser.sh
```

```
sudo apt install dos2unix
```

```
dos2unix logparser.sh
```

Πρώτα από όλα για να ξεκινήσουμε φτιάξαμε ένα file με όνομα **access.log** στο οποίο βάλαμε όλα τα δεδομένα τα οποία θα χρησιμοποιήσουμε και ένα file με όνομα **logparser.sh** που θα έχει τον κώδικα **bash shell**.



```
● iasonas@LAPTOP-JBLJ3L3J:~/project05$ ls
access.log  logparser.sh
```

Στην αρχή του **logparser.sh** έχουμε τα ονοματεπώνυμα και τα Α.Μ. σαν σχόλια. Ο πραγματικός κώδικας ξεκινάει με το **shopt -s extglob** του οποίου την χρήση θα εξηγήσουμε παρακάτω και δύο συναρτήσεις(**mining_usernames()** και **count_browsers()**) οι οποίες επίσης χρησιμοποιούνται παρακάτω και θα τις εξηγήσουμε τότε.

Για την ώρα ξεκινάμε από την **case "\$1|\$2|\$3"**. Όπως και σε άλλες προγραμματιστικές γλώσσες η **case** λειτουργεί σαν πολλά δομημένα **else-if** που στην περίπτωσή μας είναι τέλεια λόγω των πολλαπλών **conditions** γιατί θα πρέπει να ελέγχουμε τις παραμέτρους. Το \$1,\$2,\$3 αναφέρονται στις τρεις παραμέτρους που χρησιμοποιούνται για όλα τα υποερωτήματα αυτής της άσκησης. Οι χαρακτήρες “|” ανάμεσά τους είναι για να μπορούμε να αναγνωρίσουμε πιο εύκολα τα patterns πιο κάτω, τα οποία αν εκπληρωθεί κάποιο από αυτά τότε θα εκτελεστεί και το συγκεκριμένο “**else-if**”. Πάντα μέσα στα patterns θα έχουμε τουλάχιστον τα δύο “|” γιατί παραμένουν σταθερά. Άρα στην ουσία το **"\$1|\$2|\$3"** είναι string το οποίο βάζει στην “θέση” που απεικονίζεται την παράμετρο και τις χωρίζει με τον χαρακτήρα “|”. Έπειτα συγκρίνει αυτό το string με τα patterns παρακάτω.

```
case "$1|$2|$3" in
  "|")
    echo "1084647|1084632|1084673|1084552"
  ;;
```

Το πρώτο pattern είναι το "|" το οποίο σημαίνει ότι οι τρεις παράμετροι είναι άδειες.

```
● iasonas@LAPTOP-JBLJ3L3J:~/projectOS$ ./logparser.sh
1084647|1084632|1084673|1084552
```

Αν αυτό ισχύει τότε εκτυπώνει τα Α.Μ. της ομάδας.

```
!(*.log)"|""|")
echo "Wrong File Argument!"
;;
```

Το δεύτερο pattern είναι το `!(*.log)"|""|")` το οποίο σημαίνει ότι η δεύτερη και τρίτη παράμετρος μπορεί να είναι οτιδήποτε (ο `*` σημαίνει οποιοσδήποτε συνδυασμός χαρακτήρα ή χαρακτήρων συμπεριβαλλόμενου και του κενού) αλλά η πρώτη παράμετρος (`$1`) ΔΕΝ είναι της μορφής `.log`. Η έκφραση `!` σημαίνει να μην αντιστοιχεί στο συγκεκριμένο pattern. Είχαμε αναφέρει πιο πάνω την `shopt -s extglob` η οποία επιτρέπει την χρήση έκφρασης `!` γιατί χωρίς αυτήν δεν αναγνωρίζεται.

```
● iasonas@LAPTOP-JBLJ3L3J:~/projectOS$ ./logparser.sh sdadasd sadasfasf asdasda
Wrong File Argument!
```

Επομένως, αν απλά η πρώτη παράμετρος δεν έχει την μορφή `.log` εκτυπώνει `'error'` για την λάθος μορφή. Είναι σημαντικό να σημειώσουμε πως αυτό το pattern έπρεπε να είναι δεύτερο και όχι πρώτο γιατί σε αυτό μπορεί να αντιστοιχηθεί και η περίπτωση που όλοι οι παράμετροι είναι κενοί, αλλά αν το βάλουμε δεύτερο λόγω το ότι η `case` σταματάει με το που βρει ένα pattern που να αντιστοιχεί σταματάει την εκτέλεσή του και δεν ψάχνει αν ταιριάζει στα υπόλοιπα pattern.

```
*.log"||")
cat $1
echo
;;
```

Το τρίτο pattern είναι το `*.log"|"` το οποίο κοιτάει αν η πρώτη παράμετρος είναι της μορφής `.log` και οι άλλες είναι κενές.

```
::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
::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
○ iasonas@LAPTOP-JBLJ3L3J:~/projectOS$
```

Αν αυτό ισχύει τότε εκτυπώνει όλα τα δεδομένα του αρχείου που δόθηκε.

```
*.log"|--usrid|")
mining_usernames $1
;;
```

Το τέταρτο pattern είναι το `*.log"|--usrid|"` το οποίο κοιτάει το ίδιο με το τρίτο pattern με μόνη διαφορά ότι η δεύτερη παράμετρος είναι το `--usrid`. Αν αυτό ισχύει τότε καλεί την συνάρτηση `mining_usernames()` με πρώτη παράμετρο την πρώτη παράμετρο. Η συνάρτηση αυτή καλεί μια `awk` που παίρνει το τρίτο column του αρχείου, το οποίο αντιστοιχεί στον `user` και μετά με την χρήση `pipes` ταξινομεί αλφαριθμητικά τα `users` με την `sort` και μετράει πόσες φορές εμφανίζεται ο κάθε `user` με την `uniq -c`.

```
● iasonas@LAPTOP-JBLJ3L3J:~/projectOS$ ./logparser.sh access.log --usrid
15710 -
  124 admin
   34 president
  181 root
  110 user1
   91 user2
   39 user3
```

Έτσι εκτυπώνει δύο columns. Η πρώτη είναι ο αριθμός που εμφανίζεται ο `user` και η δεύτερη το όνομα του `user`.

```
*.log"|--usrid|"*)
awk -v user_id="$3" 'user_id~$3 {print}' $1
;;
```

Το πέμπτο pattern είναι το `*.log"|--usrid|"` το οποίο κοιτάει το ίδιο με το τέταρτο pattern με μόνη διαφορά ότι η τρίτη παράμετρος μπορεί να έχει οτιδήποτε. Μπορεί να είναι και ο κενός χαρακτήρας αλλά αν αυτό ίσχυε τότε θα εκτελούσαν το τέταρτο pattern. Η τρίτη παράμετρος στην συγκεκριμένη περίπτωση εκπροσωπεί το όνομα ενός **user**. Ελέγχει λοιπόν με ένα **awk** αν το τρίτο column του αρχείου που είναι το όνομα του **user** είναι ίδιο με την τρίτη παράμετρο. Χρησιμοποιήσαμε το **-v** για να μεταθέσουμε την τιμή της τρίτης παραμέτρου σε μια **awk variable** με όνομα **user_id** η οποία χρησιμοποιήθηκε για να κάνουμε τον έλεγχο που δεν θα μπορούσε να γίνει χωρίς αυτήν γιατί θα αναγκαζόμασταν να γράψουμε `“$3~$3”` αντί για `“user_id~$3”` και η **bash** δεν θα μπορούσε να καταλάβει ότι το ένα \$3 αναφέρεται στην τρίτη παράμετρο και το άλλο \$3 αναφέρεται στο τρίτο column του αρχείου.

```
iasonas@LAPTOP-JBLJ3L33:~/projectOS$ ./logparser.sh access.log --usrid root
127.0.0.1 - root - [29/Mar/2021:16:47:19 +0300] "GET / HTTP/1.1" 302 - "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:98.0) Gecko/20100101 Firefox/98.0"
127.0.0.1 - root - [29/Mar/2021:16:47:20 +0300] "GET /dashboard/images/xampp-logo.svg HTTP/1.1" 200 5427 "http://localhost/dashboard/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:98.0) Gecko/20100101 Firefox/98.0"
127.0.0.1 - root - [29/Mar/2021:16:47:24 +0300] "GET /phpmyadmin/js/dist/ajax.js?v=5.1.1 HTTP/1.1" 200 31313 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:98.0) Gecko/20100101 Firefox/98.0"
127.0.0.1 - root - [29/Mar/2021:16:47:24 +0300] "GET /phpmyadmin/js/vendor/jquery/jquery.ba-hashchange-2.0.js?v=5.1.1 HTTP/1.1" 200 10505 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:98.0) Gecko/20100101 Firefox/98.0"
127.0.0.1 - root - [29/Mar/2021:16:47:24 +0300] "GET /phpmyadmin/js/dist/config.js?v=5.1.1 HTTP/1.1" 200 24956 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:98.0) Gecko/20100101 Firefox/98.0"
127.0.0.1 - root - [29/Mar/2021:16:47:24 +0300] "GET /phpmyadmin/js/vendor/codemirror/addon/runmode/runmode.js?v=5.1.1 HTTP/1.1" 200 2773 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:98.0) Gecko/20100101 Firefox/98.0"
```

Αν είναι τα ίδια εκτυπώνει αυτή την γραμμή που έχει το συγκεκριμένο όνομα **user**.

```
*.log"| -method|GET" | *.log"| -method|POST")
if [[ "$3" = "GET" ]]
then
sed -n '/GET/p' $1
else
sed -n '/POST/p' $1
fi

echo
;;
```

Το έκτο pattern είναι το `*.log"| -method|GET" | *.log"| -method|POST"` το οποίο ελέγχει δύο περιπτώσεις. Στην πρώτη περίπτωση η πρώτη παράμετρος είναι της μορφής **log**, η δεύτερη παράμετρος είναι το **-method** και η τρίτη παράμετρος είναι το **GET**. Η δεύτερη περίπτωση είναι όπως η πρώτη με μόνη διαφορά ότι η τρίτη παράμετρος είναι το **POST**. Μόλις αναγνωριστεί μία από τις δύο περιπτώσεις τότε ελέγχουμε με ένα **if** αν η τρίτη παράμετρος είναι το **GET**. Αν είναι, τότε με ένα **sed** ψάχνουμε όλες τις γραμμές που να έχουν την συμβολοσειρά **GET**. Αν δεν είναι η τρίτη παράμετρος το **GET** τότε αναγκαστικά πρέπει να είναι το **POST** και επομένως μέσα στο **else** με ένα **sed** ψάχνουμε όλες τις γραμμές που έχουν την λέξη **POST** μέσα.

(Εκτέλεση με τρίτη παράμετρο το **GET**)

```
ken=32567d5d795a3d6d36285a4b68702655 HTTP/1.1" 200 1754
::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
○ iasonas@LAPTOP-JBLJ3L3J:~/project05$
```

(Εκτέλεση με τρίτη παράμετρο το **POST**)

```
::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
○ iasonas@LAPTOP-JBLJ3L3J:~/project05$
```

Αρα, στο τέλος γίνεται η εκτύπωση των γραμμών που έχουν **GET** ή **POST** ανάλογα με το τι έχει δοθεί στην τρίτη παράμετρο.

```
*.log"| -method|"*)
echo "Wrong Method Name!"
;;
```

Το έβδομο pattern είναι το ***.log"| -method|"*** το οποίο λειτουργεί παρόμοια με το έκτο, με την διαφορά πως αν δεν έχει γίνει αναγνώριση του έκτου pattern, αλλά η πρώτη και η δεύτερη παράμετροι καλύπτουν τις προϋποθέσεις για το έκτο pattern τότε η τρίτη παράμετρος δεν είναι ούτε **GET** ή **POST**. Επομένως, αν η τρίτη παράμετρος είναι οτιδήποτε τότε εκτελείται αυτό το “else-if”.

```
○ iasonas@LAPTOP-JBLJ3L3J:~/project05$ ./logparser.sh access.log -method sadasd
Wrong Method Name!
```

Όταν εκτελεστεί εκτυπώνεται ένα “error”.

```

*.log"|--servprot|IPv4" | *.log"|--servprot|IPv6")
if [[ "$3" = "IPv4" ]]
then
sed -n '/127.0.0.1/p' $1
else
sed -n '::-1/p' $1
fi

echo
;;

```

Το όγδοο pattern είναι το `*.log"|--servprot|IPv4" | *.log"|--servprot|IPv6"` το οποίο λειτουργεί σχεδόν ακριβώς σαν το έκτο pattern με μόνη διαφορά πως αντί για **GET** και **POST** έχουμε **IPv4** και **IPv6** και η δεύτερη παράμετρος είναι `--servprot` αντί για `-method`. Το **IPv4** αντιστοιχεί στο **127.0.0.1** και το **IPv6** αντιστοιχεί στο `::1`.

(Εκτέλεση με τρίτη παράμετρο το **IPv4**)

```

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/" "Mozilla/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/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:105.0) Gecko/20100101 Firefox/105.0"
127.0.0.1 - - [02/Oct/2022:12:37:21 +0300] "GET /MyDocs/IPEM/ HTTP/1.1" 200 4073 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:105.0) Gecko/20100101 Firefox/105.0"
127.0.0.1 - - [02/Oct/2022:12:37:21 +0300] "GET /MyDocs/IPEM/images/logo-header.png HTTP/1.1" 200 22434 "http://localhost/MyDocs/IPEM/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:105.0) Gecko/20100101 Firefox/105.0"
127.0.0.1 - - [02/Oct/2022:12:37:21 +0300] "GET /MyDocs/IPEM/images/favicon-16x16.png HTTP/1.1" 200 958 "http://localhost/MyDocs/IPEM/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:105.0) Gecko/20100101 Firefox/105.0"
o iasonas@LAPTOP-JBLJ3L3J:~/project05$

```

(Εκτέλεση με τρίτη παράμετρο το **IPv6**)

```

::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
::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
o iasonas@LAPTOP-JBLJ3L3J:~/project05$

```

Άρα, στο τέλος γίνεται η εκτύπωση των γραμμών που έχουν **IPv4** ή **IPv6** ανάλογα με το τι έχει δοθεί στην τρίτη παράμετρο.

```

*.log"|--servprot|"*)
echo "Wrong Network Protocol!"
;;

```


Το ένατο pattern είναι το `*.log"--servprot"` το οποίο λειτουργεί σχεδόν ακριβώς σαν το έβδομο pattern με μόνη διαφορά πως η δεύτερη παράμετρος είναι το `--servprot` αντί το `method`.

```
● iasonas@LAPTOP-JBLJ3L3J:~/project05$ ./logparser.sh access.log --servprot asdasdasd
Wrong Network Protocol!
```

Όταν εκτελεστεί εκτυπώνεται ένα `'error'`.

```
*.log"--browsers" )
count_browsers $1
;;
```

Το δέκατο pattern είναι το `*.log"--browsers"` το οποίο ελέγχει αν η πρώτη παράμετρος είναι της μορφής `.log`, η δεύτερη παράμετρος είναι το `--browsers` και η τρίτη παράμετρος είναι κενή. Μόλις αναγνωριστεί το pattern αυτό καλείται η συνάρτηση `count_browsers()` με πρώτη παράμετρο την πρώτη παράμετρο. Η `count_browsers()` χρησιμοποιεί την `match()` για να βρεί σε κάθε γραμμή τον κάθε browser και μετά κάνει το ίδιο pipe με την συνάρτηση `mining_usernames()` με μόνη διαφορά πως στο τέλος κάνουμε ένα τελευταίο pipe το `perl -ane 'print "$F[1] $F[0]\n"'` το οποίο απλά εναλλάσσει στην εκτύπωση τις δύο κολώνες του `output`.

```
● iasonas@LAPTOP-JBLJ3L3J:~/project05$ ./logparser.sh access.log --browsers
Chrome 124
Edg 63
Mozilla 1057
Safari 124
```

Όπως βλέπουμε αυτό που εκτυπώνεται είναι δύο στήλες. Η μια στήλη είναι οι `browsers` και η άλλη είναι το πλήθος που χρησιμοποιήθηκαν.

```
*.log"--datum|Jan" | *.log"--datum|Feb" | *.log"--datum|Mar" | *.log"--datum|Apr" | *.log"--datum|May" | *.log"--datum|Jun" |
awk -v month="$3" '{print}' $1
;;
```

Το ενδέκατο pattern είναι το `*.log"--datum|Jan" | *.log"--datum|Feb" | *.log"--datum|Mar" | *.log"--datum|Apr" | *.log"--datum|May" | *.log"--datum|Jun" | *.log"--datum|Jul" | *.log"--datum|Aug" | *.log"--datum|Sep" | *.log"--datum|Oct" | *.log"--datum|Nov" | *.log"--datum|Dec"` το οποίο ελέγχει αν η πρώτη παράμετρος είναι της μορφής `.log`, η δεύτερη παράμετρος είναι το `--datum` και η τρίτη παράμετρος είναι ένας από τους δώδεκα μήνες περιορισμένους σε τρεις χαρακτήρες. Αν το pattern αυτό αναγνωριστεί τότε ελέγχουμε ποιες γραμμές στο αρχείο `.log` έχουν αυτό τον μήνα που δόθηκε στην τρίτη

παράμετρο. Για τον έλεγχο και την εκτύπωση χρησιμοποιούμε μια εντολή **awk -v**(ο λόγος που χρησιμοποιούμε την **-v** είναι ο ίδιος που είχαμε εξηγήσει πιο πάνω).

(Εκτέλεση με τρίτη παράμετρο το **Mar**)

```
::1 - - [29/Mar/2022:17:43:30 +0300] "GET /phpmyadmin/index.php?route=/recent-table&ajax_request=1&recent_table=1&no_debug=true&_nocache=1654699410903446583&token=32567d5d795a3d6d36285a4b68702655 HTTP/1.1" 200 1734
::1 - - [29/Mar/2022:17:43:35 +0300] "GET /phpmyadmin/index.php?route=/table/sql&db=aekx&table=complaints&ajax_request=true&ajax_page_request=true&_nocache=1654699415132588064&token=32567d5d795a3d6d36285a4b68702655 HTTP/1.1" 200 5258
::1 - - [29/Mar/2022:17:43:35 +0300] "GET /phpmyadmin/index.php?route=/recent-table&ajax_request=1&recent_table=1&no_debug=true&_nocache=1654699415527541261&token=32567d5d795a3d6d36285a4b68702655 HTTP/1.1" 200 1734
::1 - - [29/Mar/2022:17:43:35 +0300] "POST /phpmyadmin/index.php?route=/lint HTTP/1.1" 200 28
::1 - - [29/Mar/2022:17:43:38 +0300] "POST /phpmyadmin/index.php?route=/database/sql/autocomplete HTTP/1.1" 200 3312
::1 - - [29/Mar/2022:17:43:38 +0300] "POST /phpmyadmin/index.php?route=/lint HTTP/1.1" 200 28
::1 - - [29/Mar/2022:17:43:42 +0300] "POST /phpmyadmin/index.php?route=/import HTTP/1.1" 200 2325
```

Με την εκτέλεση του **awk -v** εκτυπώνονται οι γραμμές στο αρχείο **.log** που έχουν τον μήνα που δόθηκε στην τρίτη παράμετρο.

```
*.log"|--datum|"* )
echo "Wrong Date!"
;;
```

Το δωδέκατο pattern είναι το ***.log"|--datum|"*** το οποίο λειτουργεί παρόμοια με το εντέκατο με την διαφορά πως αν δεν έχει γίνει αναγνώριση του εντέκατου pattern αλλά η πρώτη και η δεύτερη παράμετροι καλύπτουν τις προϋποθέσεις για το εντέκατο pattern τότε η τρίτη παράμετρος δεν είναι κανένας από τους μήνες. Επομένως, αν η τρίτη παράμετρος είναι οτιδήποτε τότε εκτελείται αυτό το “**else-if**”.

```
● iasonas@LAPTOP-JBLJ3L3J:~/projectOS$ ./logparser.sh access.log --datum Jasdfas
Wrong Date!
```

Όταν εκτελεστεί εκτυπώνεται ένα “**error**”.

```
*)
echo -e "Wrong input!\n\nCheck if the arguments are correct."
;;
```

Το τελευταίο pattern είναι το ***** το οποίο είναι το οτιδήποτε. Αυτό το έχουμε γιατί αν δεν γίνει αναγνώριση κανενός από τα παραπάνω pattern τότε σημαίνει πως δόθηκαν λάθος δεδομένα εισόδου και επομένως οι παράμετροι πρέπει να αλλάξουν.

```
● iasonas@LAPTOP-JBLJ3L3J:~/project05$ ./logparser.sh access.log --datuasdscdam Jan  
Wrong input!  
  
Check if the arguments are correct.
```

Όταν εκτελεστεί εκτυπώνεται ένα ‘error’.

Σημείωση!!!:

Τα ερωτήματα έγιναν με τις εντολές που ζητήθηκαν που είναι οι awk, sed και case. Προτού γίνουν αυτά όμως είχαμε κάνει άλλα πράγματα για το τρίτο υποερώτημα γιατί δεν είχαμε δει ότι εξαρχής χρειαζόταν αυτές τις εντολές.

Η λογική που ακολουθήσαμε είναι η δημιουργία δύο πινάκων ένας usernames() που θα έχει τους users και numbers() που θα κρατά πόσες εμφανίσεις είχαν. Η θέση του πίνακα που έχει το πλήθος των εμφανίσεων στο numbers ενός user είναι ίση με την θέση του πίνακα του συγκεκριμένου user στο usernames().

Με μια λούπα και διαβάζοντας γραμμή γραμμή, κόβαμε τους χαρακτήρες και παίρναμε τους users. Ελέγχουμε αν ήδη υπάρχουν στο πίνακα και αν δεν υπάρχουν τότε τους αποθηκεύαμε στο usernames και αυξάναμε την αντίστοιχη θέση του numbers κατά ένα. Αφού τελειώσει η λούπα μπαίναμε σε μια άλλη που έτρεχε στο usernames() και αντικαθιστούσε τον κενό χαρακτήρα με την παύλα.

Έπειτα, κάναμε αλφαβητική ταξινόμηση στο usernames() χρησιμοποιώντας διπλή λούπα και παίρνοντας το max αλφαβητικά και το βάζαμε πρώτο. Με το που γινόταν αλλαγή κάναμε και την αντίστοιχη αλλαγή και στο numbers(). Τέλος, εκτυπώναμε τους δύο πίνακες.

Προσοχή!!!: Τον κώδικα για το παραπάνω δεν τον κατέχουμε γιατί τον σβήσαμε και πολύ αργότερα σκεφτήκαμε να δώσουμε ένα άλλο σκεπτικό αλλά δεν θέλαμε να πάει εντελώς χαμένη η δουλειά. Επομένως, ελπίζουμε να σας καλύπτει αυτό προς την προγραμματιστική λογική.

Ερώτημα 2: Διεργασίες

Για την υλοποίηση του κώδικα χρησιμοποιώντας πολλαπλές διεργασίες και ουρές μηνυμάτων για την απαραίτητη επικοινωνία μεταξύ τους χρησιμοποιήσαμε την εκδοχή όπου ο χρήστης εισάγει τον αριθμό των διεργασιών:

Πέρα από τις βιβλιοθήκες που χρειαζόταν το πρόγραμμα αρχικά:

```
#include <stdio.h>
#include <math.h>
#include <sys/time.h>
```

προσθέσαμε επιπλέον τις εξής:

```
#include <sys/ipc.h>
#include <sys/msg.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/wait.h>
```

Οι οποίες χρειάζονται για την διαδιεργασιακή επικοινωνία μέσω ουρών μηνυμάτων.
Συγκεκριμένα:

- Οι <sys/ipc.h> και <sys/msg.h> παρέχουν τις συναρτήσεις για τις ουρές μηνυμάτων.
- Η <unistd.h> μας παρέχει την συνάρτηση fork() για τη δημιουργία των διεργασιών-παιδιών(child processes).
- Η <stdlib.h> παρέχει την exit() για τον τερματισμό των child processes.

Και

- Η <sys/wait.h> παρέχει την wait() την οποία χρησιμοποιεί η διεργασία-πατέρας ώστε να περιμένει να τελειώσουν οι διεργασίες-παιδιά.

Η παρακάτω συνάρτηση χρησιμοποιείται για να μετρήσει το χρόνο εκτέλεσης του προγράμματος μας:

```
double get_wtime(void)
{
    struct timeval t;
    gettimeofday(&t, NULL);
    return (double)t.tv_sec + (double)t.tv_usec*1.0e-6;
}
```

Η συνάρτηση $f(x)$ χρησιμοποιείται για τον υπολογισμό του ολοκληρώματος της $f(x)=\log(x)*\sqrt{x}$.

```
double f(double x)
{
    return log(x)*sqrt(x);
}
```

Στην main ορίζουμε τα όρια (a και b) του ολοκληρώματος καθώς και τον αριθμό των διαστημάτων για το ολοκλήρωμα (n) και το πλάτος κάθε διαστήματος (dx).

```
double a = 1.0;
double b = 4.0;
unsigned long const n = 1e9;
const double dx = (b-a)/n;
```

Η μεταβλητή S χρησιμοποιείται για την αποθήκευση του αποτελέσματος και η t0 για την αρχή του προγράμματος μας.

Η msgid αποθηκεύει το αναγνωριστικό(id) της ουράς μηνυμάτων και η key το μοναδικό κλειδί της.

```
int msgid; //message queue identifier

key_t key;
```

Η δομή msg_buf καθορίζει τον buffer μηνυμάτων(message buffer) που θα σταλεί και αποκτηθεί μέσω της ουράς. Η mtype καθορίζει το είδος του μηνύματος και η mtext τα δεδομένα(message data).

```
struct msg_buf {
    long mtype;
    double mtext; //message data
} buf;
```

Η ftok() δημιουργεί ένα μοναδικό κλειδί για την ουρά μας και η msgget() δημιουργεί την ουρά με το εκείνο το μοναδικό κλειδί.

```
key = ftok(".", 'S'); //unique key for the queue

msgid = msgget(key, 0666 | IPC_CREAT);
```

Η pid αποθηκεύει το αναγνωριστικό(process id) της διεργασίας-παιδί. Εάν είναι 0 τότε η τρέχουσα διεργασία είναι διεργασία-παιδί.

```
pid_t pid;
```

Στον βρόγχο for η fork() καλείται για να δημιουργήσει N child processes.

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

Η start αποθηκεύει το αρχικό σημείο ενός child process, πολλαπλασιάζοντας τον μετρητή(i) με τον συνολικό αριθμό επαναλήψεων δια τον συνολικό αριθμό των διεργασιών. Ομοίως στην end αποθηκεύεται το τελικό σημείο με τον ίδιο τρόπο απλώς τώρα στον μετρητή i προστίθεται 1.

```
unsigned long start = i*n/N;

unsigned long end = (i+1)*n/N;
```

Η S_local θα χρησιμοποιηθεί για την αποθήκευση του τοπικού αθροίσματος για την τρέχουσα child process.

```
double S_local = 0;
```

Αυτός ο βρόχος επαναλαμβάνεται στο εύρος των επαναλήψεων που έχει εκχωρηθεί στην τρέχουσα child process και υπολογίζει το τοπικό άθροισμα αξιολογώντας τη συνάρτηση f() σε κάθε σημείο και προσθέτοντάς την στη μεταβλητή S_local.

```
for (unsigned long j = start; j < end; j++) {
    double xi = a + (j + 0.5)*dx;
    S_local += f(xi);
}
```

Αυτό στέλνει το buffer μηνυμάτων(message buffer) στην ουρά μηνυμάτων χρησιμοποιώντας τη συνάρτηση msgsnd():

```
msgsnd(msgid, &buf, sizeof(buf), 0);
```

Τέλος η child process τερματίζει με την exit(0);

Αυτός ο βρόχος εκτελείται στη γονική διαδικασία και λαμβάνει τα message buffer από την ουρά μηνυμάτων χρησιμοποιώντας τη συνάρτηση msgrcv(). Προσθέτει το κείμενο του μηνύματος από κάθε buffer στη μεταβλητή καθολικού αθροίσματος S:

```
for (i = 0; i < N; i++) {
    msgrcv(msgid, &buf, sizeof(buf), 1, 0);
    S += buf.mtext;
}
```

Η t1 χρησιμοποιείται για τον υπολογισμό της τελικής χρονικής στιγμής του προγράμματος μας.

Καταργεί την ουρά μηνυμάτων χρησιμοποιώντας τη συνάρτηση `msgctl()` με τη σημαία `IPC_RMID`:

```
//close the queue  
msgctl(msgid, IPC_RMID, NULL);
```

Τώρα αν κάνουμε `compile` το πρόγραμμα μας ως:

```
iam_alkis@iamalkis-HP-255-G7-Notebook-PC:~/Desktop$ gcc integral_seq_msgq.c -o integral_seq_msgq.exe -lm
```

Τότε εάν εισάγουμε 10, δηλαδή να εκτελεστεί ο κώδικας χρησιμοποιώντας 10 διεργασίες θα εκτυπωθεί:

```
iam_alkis@iamalkis-HP-255-G7-Notebook-PC:~/Desktop$ ./integral_seq_msgq.exe  
Please enter the number of processes: 10  
Time=8.119118 seconds, Result=4.28245881  
Number of processes: 10
```

Όπου παρατηρούμε ότι χρειάστηκαν 8.12 δευτερόλεπτα μέχρι να υπολογιστεί το ολοκλήρωμα,

Ενώ εάν εισάγουμε $10000 = 10^4$ θα εκτυπωθεί:

```
iam_alkis@iamalkis-HP-255-G7-Notebook-PC:~/Desktop$ ./integral_seq_msgq.exe  
Please enter the number of processes: 10000  
Time=136.325054 seconds, Result=4.28245881  
Number of processes: 10000
```

παρατηρούμε ότι χρειάστηκαν 136.33 δευτερόλεπτα για τον αντίστοιχο υπολογισμό, δηλαδή η αλλαγή στον χρόνο που χρειάζεται το πρόγραμμα να εκτελέσει με πολλές περισσότερες διεργασίες είναι εμφανής.

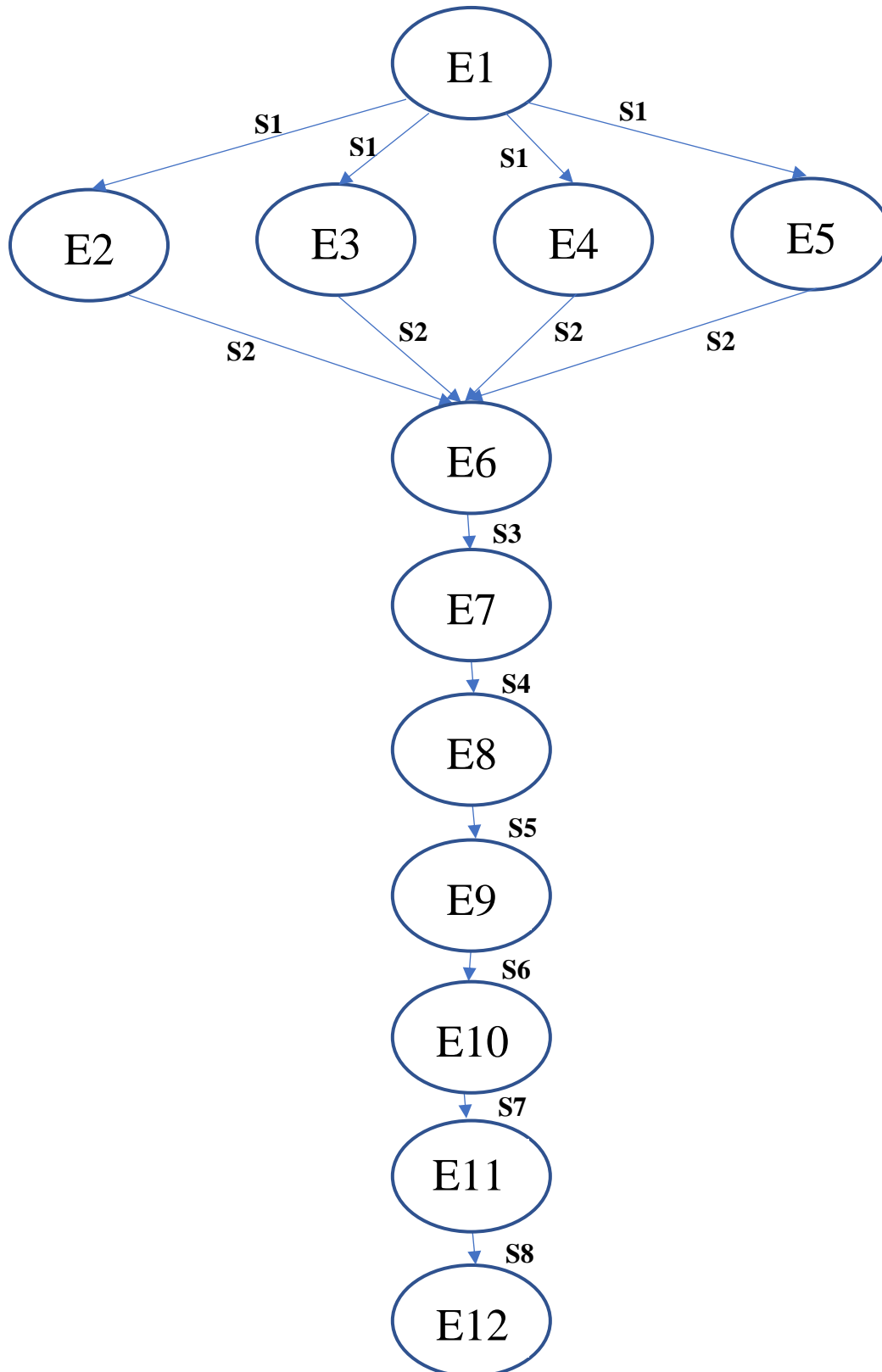
Στο πρόγραμμα έχει προβλεφθεί και η περίπτωση ο χρήστης να δώσει αρνητικό αριθμό ή 0 από διεργασίες. Σε αυτήν την περίπτωση εμφανίζεται το αντίστοιχο μήνυμα λάθους. Π.χ. η ακόλουθη περίπτωση:

```
iam_alkis@iamalkis-HP-255-G7-Notebook-PC:~/Desktop$ ./integral_seq_msgq.exe  
Please enter the number of processes: -1  
Error: You cannot compute the integral using 0 or less processes!iam_alkis@i
```

Ερώτημα 3: Διαδικεργασιακή Επικοινωνία

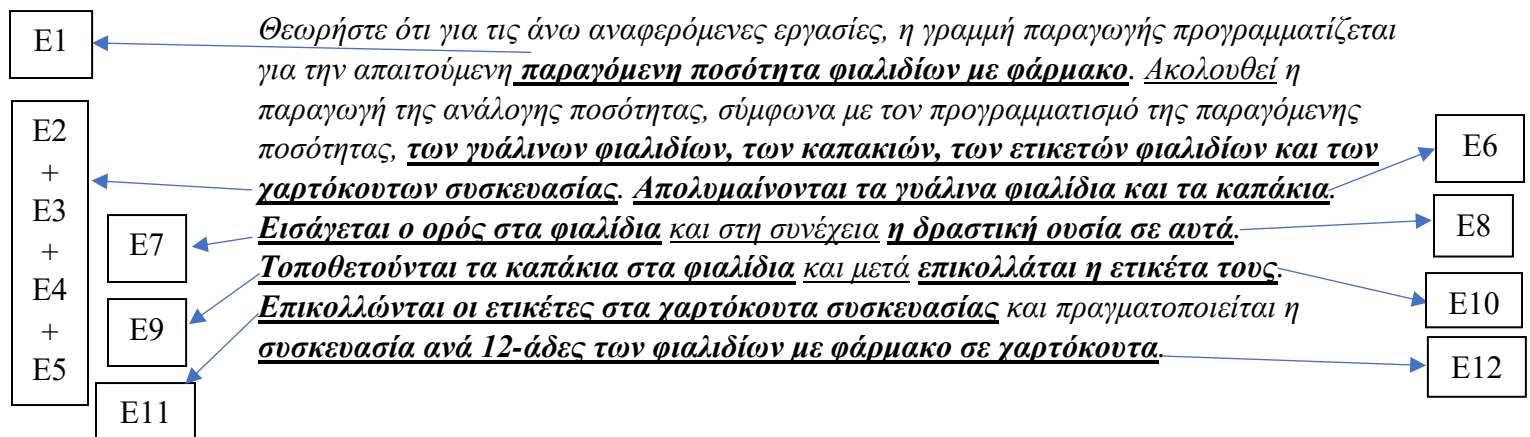
a)

Σύμφωνα με την περιγραφή για το πως συσκευάζονται τα φάρμακα σε φιαλίδια, το γράφημα προτεραιότητας των εργασιών είναι το εξής:



Για να καταλήξουμε στο παραπάνω διάγραμμα ακολουθήσαμε την περιγραφή της εκφώνησης:

- Ανάγνωση προγραμματισμένης παραγόμενης ποσότητας φιαλιδίων με φάρμακο (E1)
- Παραγωγή γυάλινων φιαλιδίων (E2)
- Παραγωγή καπακιών για τα φιαλίδια (E3)
- Παραγωγή ετικετών για τα φιαλίδια και χαρτόκουτα (E4)
- Παραγωγή χαρτόκουτων συσκευασίας (E5)
- Απολύμανση γυάλινων φιαλιδίων και καπακιών (E6)
- Εισαγωγή ορού στα φιαλίδια (E7)
- Εισαγωγή δραστικής ουσίας στα φιαλίδια (E8)
- Τοποθέτηση καπακιών στα φιαλίδια (E9)
- Κόλληση ετικετών στα φιαλίδια (E10)
- Κόλληση ετικετών στις συσκευασίες χαρτόκουτου (E11)
- Συσκευασία των φιαλιδίων σε χαρτόκουτα (E12)



Οπότε παρακάτω θα περιγράψουμε σε κώδικα πως θα πρέπει να γίνει η διάταξη των εργασιών.

b)

```
For(from i=1 to i=12 with step i=i+1)
{
  E1;
  begin
  cobegin
  E2;
  E3;
  E4;
  E5;
  coend;
  E6;
  E7;
  E8;
  E9;
  E10;
}
E11;
end;
E12;
```

Ο παραπάνω κώδικας δείχνει το πώς πρέπει να γίνει η συσκευή των φιαλιδίων. Αρχικά, ξεκινάει η διεργασία E1 και μόλις τελειώσει εκτελούνται παράλληλα οι E2,E3,E4,E5. Έπειτα ,εκτελούνται σειριακά οι E6,E7,E8,E8 και E10. Όμως, επειδή κάθε κουτί πρέπει να έχει 12 φιαλίδια θα χρειαστεί να γίνει αυτή η διαδικασία 12 φορές με έναν βρόγχο και έπειτα θα εκτελεστούν σειριακά η E11 και E12 που θα κολλήσουν την ετικέτα στο κουτί και θα συσκευάσουν τα φιαλίδια σε αυτό.

c)

```
var s1, s2, s3, s4, s5, s6, s7, s8 : semaphores;
s1 = s3 = s4 = s5 = s6 = s7 = s8 = 0;
s2 = -3;
cobegin
E1:  begin printf("Eimai stin E1"); up(s1); up(s1); up(s1); up(s1); end;
E2:  begin down(s1); printf("Eimai stin E2"); up(s2); end;
E3:  begin down(s1); printf("Eimai stin E3"); up(s2); end;
E4:  begin down(s1); printf("Eimai stin E4"); up(s2); end;
E5:  begin down(s1); printf("Eimai stin E5"); up(s2); end;
E6:  begin down(s2); printf("Eimai stin E6"); up(s3); end;
E7:  begin down(s3); printf("Eimai stin E7"); up(s4); end;
E8:  begin down(s4); printf("Eimai stin E8"); up(s5); end;
E9:  begin down(s5); printf("Eimai stin E9"); up(s6); end;
E10: begin down(s6); printf("Eimai stin E10"); up(s7); end;
E11: begin down(s7); printf("Eimai stin E11"); up(s8); end;
E12: begin down(s8); printf("Eimai stin E12"); end;
coend;
down(S){
if (s ==0) then sleep();
else s = s - 1;
}
up(S){
s: = s + 1;}
```

Έχοντας στο νου το γράφημα προτεραιότητας μπορούμε να υλοποιήσουμε το down των διεργασιών E2,E3,E4 και E5 με έναν μόνο σεμαφόρο **(οπότε απαντάμε το γ και δ ερώτημα σε ένα)**. Αυτό μπορεί να συμβεί διότι, όταν τελειώσει η διεργασία E1 δεν χρειάζεται να χρησιμοποιήσουμε ξεχωριστούς σεμαφόρους για τις παραπάνω διεργασίες αφού εκτελούνται παράλληλα ξεκινούν ταυτόχρονα. Όπως αναφέρθηκε και στην εκφώνηση, σαν διεργασία E_n όπου $n=\{1,2,\dots,12\}$ μπορούμε όσον αφορά το περιεχόμενο τους απλά να τυπώνετε ένα μήνυμα κειμένου.

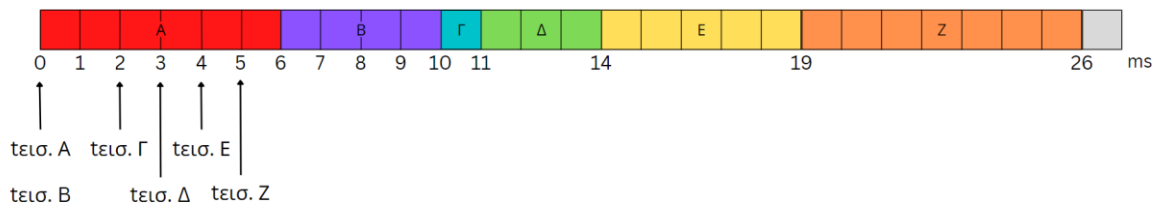
Ερώτημα 4: Χρονοπρογραμματισμός Διεργασιών

Διεργασία	Χρόνος Αφίξης	Διάρκεια Εκτέλεσης	PID
A	0	6	3
B	0	4	1
Γ	2	1	2
Δ	3	3	5
E	4	5	4
Z	5	7	1

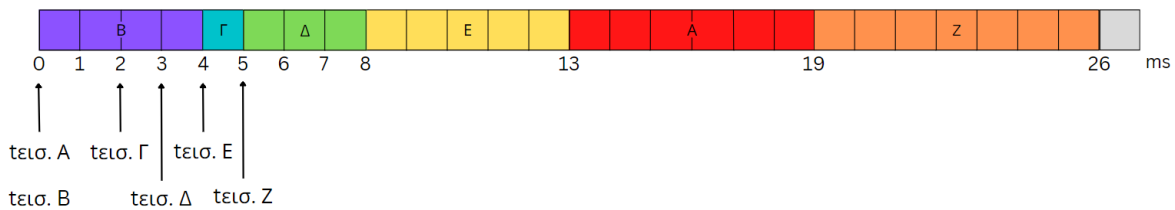
Πίνακας διεργασιών

α) Διαγράμματα Gantt

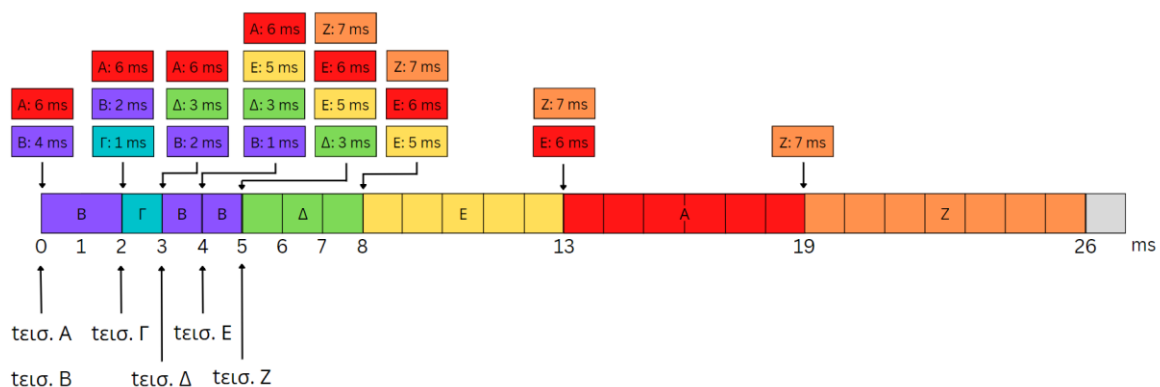
FCFS:



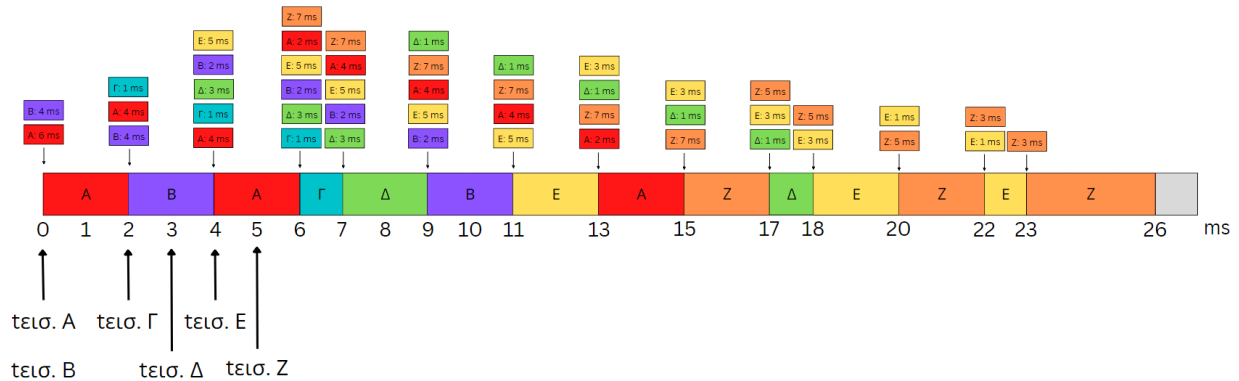
SJF:



SRTF:



RR (κβάντο χρόνου = 2 ms):



β) Υπολογισμοί Χρόνων

Τύποι:

$$\text{Χρόνος Αναμονής (XA)} = t_{\varepsilon\xi} - t_{\varepsilon\sigma} - t_{cpu}$$

$$t_{cpu} = \text{χρόνος εξυπηρέτησης διεργασίας}$$

$$\text{Μέσος Χρόνος Αναμονής (MXA)} = \frac{XA_1 + XA_2 + \dots + XA_n}{n}$$

$$\text{Χρόνος Απόκρισης (XA\pi)} = t_{\alpha\pi} - t_{\varepsilon\sigma}$$

$$t_{\alpha\pi} = \text{χρονική στιγμή πρώτης απόκρισης του συστήματος}$$

$$\text{Μέσος Χρόνος Απόκρισης (MXA\pi)} = \frac{XA\pi_1 + XA\pi_2 + \dots + XA\pi_n}{n}$$

$$\text{Χρόνος Ολοκλήρωσης (XO)} = t_{\varepsilon\xi} - t_{\varepsilon\sigma}$$

$$\text{Μέσος Χρόνος Ολοκλήρωσης (MXO)} = \frac{XO_1 + XO_2 + \dots + XO_n}{n}$$

#θεματικών εναλλαγών

$$= \#θεμ. εν. A + \#θεμ. εν. B + \#θεμ. εν. \Gamma + \#θεμ. εν. \Delta + \#θεμ. εν. E + \#θεμ. εν. Z$$

FCFS:***Μέσος Χρόνος Αναμονής***

$$XA_A = 6 - 0 - 6 = 0 \text{ ms}$$

$$XA_B = 10 - 0 - 4 = 6 \text{ ms}$$

$$XA_\Gamma = 11 - 2 - 1 = 8 \text{ ms}$$

$$XA_\Delta = 14 - 3 - 3 = 8 \text{ ms}$$

$$XA_E = 19 - 4 - 5 = 10 \text{ ms}$$

$$XA_Z = 26 - 5 - 7 = 14 \text{ ms}$$

$$MXA = \frac{XA_A + XA_B + XA_\Gamma + XA_\Delta + XA_E + XA_Z}{6} = \frac{0 + 6 + 8 + 8 + 10 + 14}{6} = 7.66 \text{ ms}$$

Μέσος Χρόνος Απόκρισης

$$XA\pi_A = 0 - 0 = 0 \text{ ms}$$

$$XA\pi_B = 6 - 0 = 6 \text{ ms}$$

$$XA\pi_\Gamma = 10 - 2 = 8 \text{ ms}$$

$$XA\pi_\Delta = 11 - 3 = 8 \text{ ms}$$

$$XA\pi_E = 14 - 4 = 10 \text{ ms}$$

$$XA\pi_Z = 19 - 5 = 14 \text{ ms}$$

$$MXA\pi = \frac{XA\pi_A + XA\pi_B + XA\pi_\Gamma + XA\pi_\Delta + XA\pi_E + XA\pi_Z}{6} = \frac{0 + 6 + 8 + 8 + 10 + 14}{6} = 7.66 \text{ ms}$$

Μέσος Χρόνος Ολοκλήρωσης

$$XO_A = 6 - 0 = 6 \text{ ms}$$

$$XO_B = 10 - 0 = 10 \text{ ms}$$

$$XO_\Gamma = 11 - 2 = 9 \text{ ms}$$

$$XO_\Delta = 14 - 3 = 11 \text{ ms}$$

$$XO_E = 19 - 4 = 15 \text{ ms}$$

$$XO_Z = 26 - 5 = 21 \text{ ms}$$

$$MXO = \frac{XO_A + XO_B + XO_\Gamma + XO_\Delta + XO_E + XO_Z}{6} = \frac{6 + 10 + 9 + 11 + 15 + 21}{6} = 12 \text{ ms}$$

$$\#\theta\epsilon\mu\alpha\tau\iota\kappa\omega\acute{\nu}\nu \epsilon\nu\alpha\lambda\lambda\alpha\gamma\omega\acute{\nu}\nu = 1 + 1 + 1 + 1 + 1 + 1 = 6$$

SJF:

Μέσος Χρόνος Αναμονής

$$XA_A = 19 - 0 - 6 = 13 \text{ ms}$$

$$XA_B = 4 - 0 - 4 = 0 \text{ ms}$$

$$XA_\Gamma = 5 - 2 - 1 = 2 \text{ ms}$$

$$XA_\Delta = 8 - 3 - 3 = 2 \text{ ms}$$

$$XA_E = 13 - 4 - 5 = 4 \text{ ms}$$

$$XA_Z = 26 - 5 - 7 = 14 \text{ ms}$$

$$MXA = \frac{XA_A + XA_B + XA_\Gamma + XA_\Delta + XA_E + XA_Z}{6} = \frac{13 + 0 + 2 + 2 + 4 + 14}{6} = 5.83 \text{ ms}$$

Μέσος Χρόνος Απόκρισης

$$XA\pi_A = 13 - 0 = 13 \text{ ms}$$

$$XA\pi_B = 0 - 0 = 0 \text{ ms}$$

$$XA\pi_\Gamma = 4 - 2 = 2 \text{ ms}$$

$$XA\pi_\Delta = 5 - 3 = 2 \text{ ms}$$

$$XA\pi_E = 8 - 4 = 4 \text{ ms}$$

$$XA\pi_Z = 19 - 5 = 14 \text{ ms}$$

$$MXA\pi = \frac{XA\pi_A + XA\pi_B + XA\pi_\Gamma + XA\pi_\Delta + XA\pi_E + XA\pi_Z}{6} = \frac{13 + 0 + 2 + 2 + 4 + 14}{6} = 5.83 \text{ ms}$$

Μέσος Χρόνος Ολοκλήρωσης

$$XO_A = 19 - 0 = 19 \text{ ms}$$

$$XO_B = 4 - 0 = 4 \text{ ms}$$

$$XO_\Gamma = 5 - 2 = 3 \text{ ms}$$

$$XO_\Delta = 8 - 3 = 5 \text{ ms}$$

$$XO_E = 13 - 4 = 9 \text{ ms}$$

$$XO_Z = 26 - 5 = 21 \text{ ms}$$

$$MXO = \frac{XO_A + XO_B + XO_\Gamma + XO_\Delta + XO_E + XO_Z}{6} = \frac{19 + 4 + 3 + 5 + 9 + 21}{6} = 10.16 \text{ ms}$$

$$\#\theta\epsilon\mu\alpha\tau\iota\kappa\omega\acute{\nu}\nu \epsilon\nu\alpha\lambda\lambda\alpha\gamma\omega\acute{\nu}\nu = 1 + 1 + 1 + 1 + 1 + 1 = 6$$

SRTF:***Μέσος Χρόνος Αναμονής***

$$XA_A = 19 - 0 - 6 = 13 \text{ ms}$$

$$XA_B = 5 - 0 - 4 = 1 \text{ ms}$$

$$XA_\Gamma = 3 - 2 - 1 = 0 \text{ ms}$$

$$XA_\Delta = 8 - 3 - 3 = 2 \text{ ms}$$

$$XA_E = 13 - 4 - 5 = 4 \text{ ms}$$

$$XA_Z = 26 - 5 - 7 = 14 \text{ ms}$$

$$MXA = \frac{XA_A + XA_B + XA_\Gamma + XA_\Delta + XA_E + XA_Z}{6} = \frac{13 + 1 + 0 + 2 + 4 + 14}{6} = 5.66 \text{ ms}$$

Μέσος Χρόνος Απόκρισης

$$XA\pi_A = 13 - 0 = 13 \text{ ms}$$

$$XA\pi_B = 0 - 0 = 0 \text{ ms}$$

$$XA\pi_\Gamma = 2 - 2 = 0 \text{ ms}$$

$$XA\pi_\Delta = 5 - 3 = 2 \text{ ms}$$

$$XA\pi_E = 8 - 4 = 4 \text{ ms}$$

$$XA\pi_Z = 19 - 5 = 14 \text{ ms}$$

$$MXA\pi = \frac{XA\pi_A + XA\pi_B + XA\pi_\Gamma + XA\pi_\Delta + XA\pi_E + XA\pi_Z}{6} = \frac{13 + 0 + 0 + 2 + 4 + 14}{6} = 5.5 \text{ ms}$$

Μέσος Χρόνος Ολοκλήρωσης

$$XO_A = 19 - 0 = 19 \text{ ms}$$

$$XO_B = 5 - 0 = 5 \text{ ms}$$

$$XO_\Gamma = 3 - 2 = 1 \text{ ms}$$

$$XO_\Delta = 8 - 3 = 5 \text{ ms}$$

$$XO_E = 13 - 4 = 9 \text{ ms}$$

$$XO_Z = 26 - 5 = 21 \text{ ms}$$

$$MXO = \frac{XO_A + XO_B + XO_\Gamma + XO_\Delta + XO_E + XO_Z}{6} = \frac{19 + 5 + 1 + 5 + 9 + 21}{6} = 10 \text{ ms}$$

$$\#\theta\epsilon\mu\alpha\tau\iota\kappa\omega\acute{\nu}\nu \epsilon\nu\alpha\lambda\lambda\alpha\gamma\omega\acute{\nu}\nu = 1 + 2 + 1 + 1 + 1 + 1 = 7$$

RR:

Μέσος Χρόνος Αναμονής

$$XA_A = 15 - 0 - 6 = 9 \text{ ms}$$

$$XA_B = 11 - 0 - 4 = 7 \text{ ms}$$

$$XA_\Gamma = 7 - 2 - 1 = 4 \text{ ms}$$

$$XA_\Delta = 18 - 3 - 3 = 12 \text{ ms}$$

$$XA_E = 23 - 4 - 5 = 14 \text{ ms}$$

$$XA_Z = 26 - 5 - 7 = 14 \text{ ms}$$

$$MXA = \frac{XA_A + XA_B + XA_\Gamma + XA_\Delta + XA_E + XA_Z}{6} = \frac{9 + 7 + 4 + 12 + 14 + 14}{6} = 10 \text{ ms}$$

Μέσος Χρόνος Απόκρισης

$$XA\pi_A = 0 - 0 = 0 \text{ ms}$$

$$XA\pi_B = 2 - 0 = 2 \text{ ms}$$

$$XA\pi_\Gamma = 6 - 2 = 4 \text{ ms}$$

$$XA\pi_\Delta = 7 - 3 = 4 \text{ ms}$$

$$XA\pi_E = 11 - 4 = 7 \text{ ms}$$

$$XA\pi_Z = 15 - 5 = 10 \text{ ms}$$

$$MXA\pi = \frac{XA\pi_A + XA\pi_B + XA\pi_\Gamma + XA\pi_\Delta + XA\pi_E + XA\pi_Z}{6} = \frac{0 + 2 + 4 + 4 + 7 + 10}{6} = 4.5 \text{ ms}$$

Μέσος Χρόνος Ολοκλήρωσης

$$XO_A = 15 - 0 = 15 \text{ ms}$$

$$XO_B = 11 - 0 = 11 \text{ ms}$$

$$XO_\Gamma = 7 - 2 = 5 \text{ ms}$$

$$XO_\Delta = 18 - 3 = 15 \text{ ms}$$

$$XO_E = 23 - 4 = 19 \text{ ms}$$

$$XO_Z = 26 - 5 = 21 \text{ ms}$$

$$MXO = \frac{XO_A + XO_B + XO_\Gamma + XO_\Delta + XO_E + XO_Z}{6} = \frac{15 + 11 + 5 + 15 + 19 + 21}{6} = 14.33 \text{ ms}$$

$$\#\theta\epsilon\mu\alpha\tau\iota\kappa\acute{\omega}\nu \epsilon\nu\alpha\lambda\lambda\alpha\gamma\acute{\omega}\nu = 3 + 2 + 1 + 2 + 3 + 3 = 14$$

γ) Ποσοστά μεταβολής του μέσου χρόνου διεκπεραίωσης

$$MXO_{FCFS} = 12 \text{ ms}$$

$$MXO_{SJF} = 10.16 \text{ ms}$$

$$MXO_{SRTF} = 10 \text{ ms}$$

$$MXO_{RR} = 14.33 \text{ ms}$$

$$\text{Ποσοστό μεταβολής}_{SJF} = \frac{MXO_{SJF} - MXO_{FCFS}}{MXO_{FCFS}} * 100 = \frac{10.16 - 12}{12} * 100 = -15.33\%$$

Άρα μείωση του μέσου χρόνου ολοκλήρωσης κατά 15.33%.

$$\text{Ποσοστό μεταβολής}_{SRTF} = \frac{MXO_{SRTF} - MXO_{FCFS}}{MXO_{FCFS}} * 100 = \frac{10 - 12}{12} * 100 = -16.66\%$$

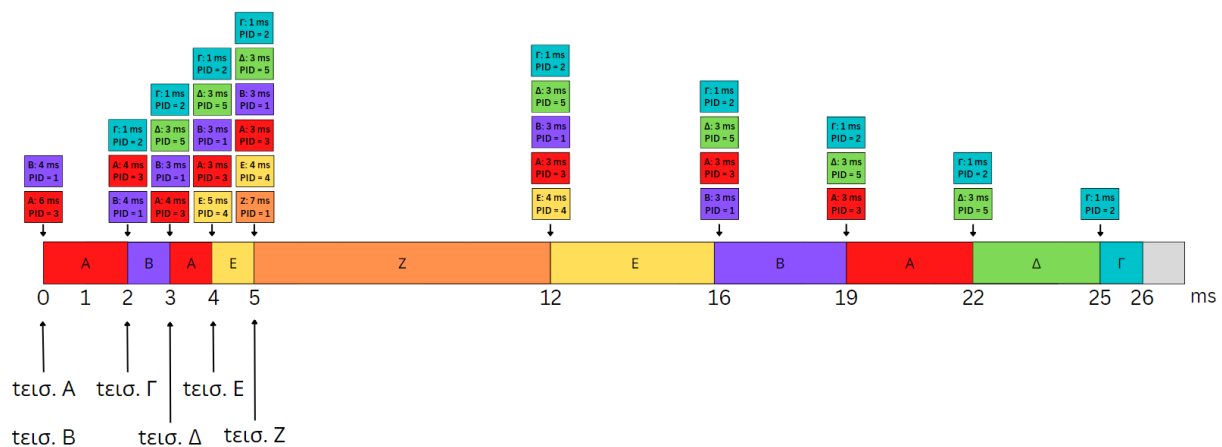
Άρα μείωση του μέσου χρόνου ολοκλήρωσης κατά 16.66%.

$$\text{Ποσοστό μεταβολής}_{RR} = \frac{MXO_{RR} - MXO_{FCFS}}{MXO_{FCFS}} * 100 = \frac{14.33 - 12}{12} * 100 = 19.41\%$$

Άρα αύξηση του μέσου χρόνου ολοκλήρωσης κατά 19.41%.

δ) LRTFP (Longest Remaining Time First Preemptive) scheduling policy

Διάγραμμα Gantt:



Υπολογισμός Χρόνων:

Μέσος Χρόνος Αναμονής

$$XA_A = 22 - 0 - 6 = 16 \text{ ms}$$

$$XA_B = 19 - 0 - 4 = 15 \text{ ms}$$

$$XA_\Gamma = 26 - 2 - 1 = 23 \text{ ms}$$

$$XA_\Delta = 25 - 3 - 3 = 19 \text{ ms}$$

$$XA_E = 16 - 4 - 5 = 7 \text{ ms}$$

$$XA_Z = 12 - 5 - 7 = 0 \text{ ms}$$

$$MXA = \frac{XA_A + XA_B + XA_\Gamma + XA_\Delta + XA_E + XA_Z}{6} = \frac{16 + 15 + 23 + 19 + 7 + 0}{6} = 13.33 \text{ ms}$$

Μέσος Χρόνος Απόκρισης

$$XA\pi_A = 0 - 0 = 0 \text{ ms}$$

$$XA\pi_B = 2 - 0 = 2 \text{ ms}$$

$$XA\pi_\Gamma = 25 - 2 = 23 \text{ ms}$$

$$XA\pi_\Delta = 22 - 3 = 19 \text{ ms}$$

$$XA\pi_E = 12 - 4 = 8 \text{ ms}$$

$$XA\pi_Z = 5 - 5 = 0 \text{ ms}$$

$$MXA\pi = \frac{XA\pi_A + XA\pi_B + XA\pi_\Gamma + XA\pi_\Delta + XA\pi_E + XA\pi_Z}{6} = \frac{0 + 2 + 23 + 19 + 8 + 0}{6} = 8.66 \text{ ms}$$

Μέσος Χρόνος Ολοκλήρωσης

$$XO_A = 22 - 0 = 22 \text{ ms}$$

$$XO_B = 19 - 0 = 19 \text{ ms}$$

$$XO_\Gamma = 26 - 2 = 24 \text{ ms}$$

$$XO_\Delta = 25 - 3 = 22 \text{ ms}$$

$$XO_E = 16 - 4 = 12 \text{ ms}$$

$$XO_Z = 12 - 5 = 7 \text{ ms}$$

$$MXO = \frac{XO_A + XO_B + XO_\Gamma + XO_\Delta + XO_E + XO_Z}{6} = \frac{22 + 19 + 24 + 22 + 12 + 7}{6} = 17.66 \text{ ms}$$

$$\#\theta\epsilon\mu\alpha\tau\iota\kappa\omega\acute{\nu}\ \epsilon\nu\alpha\lambda\lambda\alpha\gamma\omega\acute{\nu}\ = 3 + 2 + 1 + 1 + 2 + 1 = 10$$