

1^η Εργαστηριακή Άσκηση στο Μάθημα: “Λειτουργικά Συστήματα”

Ομάδα Εργασίας: Καλημέρη Σταματία
Λαζανάς Αλέξιος

Τμήμα Μηχανικών Η/Υ και Πληροφορικής (Πανεπιστήμιο Πατρών) 3 Μαρτίου 2025



Περιεχόμενα

0	Γενικά Σχόλια και Παραδοχές	3
1	Άσκηση 1 (Shell Scripting)	4
2	Άσκηση 2 (Συγχρονισμός με Σημαφόρους)	12
2.1	Εργαλεία που χρησιμοποιήθηκαν (C++)	12
2.2	Θεωρητική περιγραφή του συγχρονισμού και ψευδοκώδικες	12
2.3	Ο Κώδικας της Άσκησης και σχόλια σε αυτόν	14
3	Άσκηση 3 (Χρονοπρογραμματισμός Διεργασιών)	18
3.1	Εργαλεία που χρησιμοποιήθηκαν (Γλώσσα C++)	18
3.2	Θεωρητική περιγραφή αλγορίθμου χρονοπρογραμματισμού (Round-Robin)	18
3.3	Θεωρητική περιγραφή του αλγορίθμου για διάθεση μνήμης στις διεργασίες	19
3.4	Ο κώδικας της άσκησης και σχόλια σε αυτόν	21
4	Άσκηση 4 (Διαγράμματα Gantt, Μέσοι χρόνοι διεκπεραίωσης, αναμονής, απόδοσης)	24

0 Γενικά Σχόλια και Παραδοχές

1. Η υλοποίηση των προγραμμάτων της εργασίας διεκπεραιώθηκε σε Manjaro Linux amd64
2. Το συμπιεσμένο αρχείο που παραδόθηκε περιέχει 3 φακέλους (με όνομα `exrsX`) ο καθένας με τα προγράμματα της αντίστοιχης άσκησης `X` και τα αντίστοιχα εκτελέσιμα αρχεία.
3. Έγινε χρήση της `C++20` στις ασκήσεις 2 και 3, λόγω της αντικειμενοστραφούς δυνατότητάς της και των δομών στην βιβλιοθήκη της, την προτιμήσαμε. Για τον συγχρονισμό των νημάτων στην άσκηση 2 χρησιμοποιούμε `std::mutex` και `std::counting_semaphore`. Για την δημιουργία των νημάτων χρησιμοποιήσαμε την δομή `std::thread` (περισσότερα στο κεφάλαιο αφιερωμένο στην άσκηση 2).
4. Τα διαγράμματα Gantt της άσκησης 4 είναι σχεδιασμένα με τρόπο τέτοιο ώστε να είναι εμφανής η εναλλαγή χρονικής στιγμής και η εναλλαγή διεργασίας (κάθε διεργασία έχει και το δικό της χαρακτηριστικό χρώμα)

1 Άσκηση 1 (Shell Scripting)

Υλοποίηση Εισαγωγής CSV αρχείου

Η συνάρτηση *insert_data* ζητά από τον χρήστη να εισάγει το όνομα του αρχείου που αυτός επιθυμεί με σχετικό μήνυμα. Στην περίπτωση που δεν εισάγει κάποιο όνομα ή απλά πατήσει *enter* εισάγεται το *passenger_data.csv*

Εικόνες:

Αν βάλω ένα οποιοδήποτε αρχείο

```
Welcome to PassFinder...\nInserting data ..... \ninsert the name of the file\ntext.txt\nThe file of your choice is inserted
```

Αν δεν βάλω τίποτα στο όνομα αρχείου

```
Inserting data ..... \ninsert the name of the file\n\nThe passenger_data.csv is inserted
```

Εμφάνιση CSV αρχείου

Η εμφάνιση αφού γίνεται μέσω της *more* καθώς ζητείται εμφάνιση ανά σελίδες

Εικόνες:

Αν βάλω ένα οποιοδήποτε αρχείο:

```
insert the name of the file\ntest.txt\nThe file of your choice is inserted\ndisplaying Data...\nhi i am the wrong file
```

Αν έχω το *passenger_data.csv*:

```
displaying Data...
code;fullname;age;country;status;rescued
1;Nunez Jorge;20;Russia;Passenger;Yes
2;Hartmann Wolfgang;21;Germany;Passenger;Yes
3;Haarhoff Lily Tembo;22;United Kingdom;Passenger;Yes
4;Zhang Wei;23;France;Passenger;yes
5;Zhang Yang;24;Italy;Passenger;yes
6;Nguyen Cam;25;Spain;Passenger;yes
7;Znaimer Moses;26;Poland;Passenger;yes
8;Phan Don;27;Ukraine;Passenger;no
9;Takahashi Koji;28;Romania;Passenger;yes
10;Chen Ben;29;Netherlands;Passenger;yes
11;Ngoche Alex Obanda;30;Belgium;Crew;no
12;Kobayashi Ken;31;Czech Republic (Czechia);Crew;no
13;Ben Dhifallah Karim;32;Sweden;Passenger;Yes
14;Santos Carlos;33;Portugal;Passenger;yes
15;Korner Karl;34;Greece;Passenger;yes
16;Fayed Paul;35;Hungary;Passenger;yes
17;Charoenpura Somchai;36;Austria;Passenger;yes
18;Abe Kenji;37;Belarus;Passenger;yes
19;Li Lei;38;Switzerland;Passenger;no
20;Khan Babar;39;Bulgaria;Passenger;yes
21;Williams Jack;40;Serbia;Passenger;yes
22;Chen Hsin;41;Denmark;Crew;no
23;Wagner Richard;42;Finland;Passenger;Yes
24;Basov Irina;43;Norway;Passenger;Yes
25;Bhiari Ammar;44;Slovakia;Passenger;yes
26;Wolf Paul;45;Ireland;Passenger;yes
--More-- ( 2%)
```

Υλοποίηση *search_passenger*

Η συνάρτηση αυτή υλοποιήθηκε μέσω της συνάρτησης *grep*. Το αποτέλεσμα της είναι να εμφανίσει όλες γραμμές περιέχουν το όνομα ή το επίθετο που θα εισάγει ο χρήστης. Από την στιγμή που υπάρχουν πολλοί που έχουν είτε το ίδιο όνομα είτε το ίδιο επίθετο θα έχω αρκετά αποτελέσματα.

Επιπλέον πριν από την πρώτη αναζήτηση και μετά από το τέλος της ζητάει από τον χρήστη αν θέλει να γίνει η αναζήτηση. Προσοχή αναγνωρίζει μόνο το Y για *yes* και N *no*. Σε περίπτωση που δεν ακολουθηθεί αυτή η συμφωνία το πρόγραμμα θα ζητάει συνεχώς από τον χρήστη το όνομα για την αναζήτηση.

Εικόνα:

```
Do you want to search someone [Y/N]?
Y
name or surname of passenger
Zhu
134;Zhu Lin;67;Italy;Passenger;no
179;Zhu Zhong;36;Poland;Passenger;yes
288;Zhu Jun;69;Lithuania;Passenger;Yes
406;Zhu Lin;35;Switzerland;Passenger;yes
830;Zhu Ge;79;Sweden;Passenger;yes
838;Zhu Zheng;11;Serbia;Passenger;Yes
943;Zhu Baby;40;Andorra;Passenger;no
1042;Zhu Clara;63;Netherlands;Passenger;no
1079;Zhu Lei;24;France;Passenger;Yes
1200;Zhu Zheng;69;Iceland;Passenger;Yes
1280;Zhu Hao;73;Latvia;Passenger;yes
Do you want to search someone else [Y/N]?
```

```
Do you want to search someone else [Y/N]?
Y
name or surname of passenger
Baby
254;Li Baby;35;Iceland;Passenger;Yes
943;Zhu Baby;40;Andorra;Passenger;no
984;Zhang Baby;81;Malta;Passenger;no
Do you want to search someone else [Y/N]?
█
```

Υλοποίηση *change_data*

Σε αυτή την συνάρτηση αξίζει να αναφερθούν τα εξής:

1. Αρχικά ο κωδικός είναι είτε ο αριθμός εγγραφής είτε το όνομα του επιβαίνοντα είτε το επώνυμο. Για αυτόν τον λόγο υπάρχει ένας έλεγχος που κοιτά αν αυτό που έδωσα είναι πρώτα αριθμός. Σε αντίθετη περίπτωση ζητά από τον χρήστη το ονοματεπώνυμο του επιβαίνοντα και με αυτό γίνεται η αναζήτηση. Με αυτόν τον τρόπο ξεχωρίζω με βάση ποιά στήλη θα ψάξει.
2. Μέσω της *awk* γίνεται και η αναζήτηση αλλά και η αλλαγή. Τα αποτελέσματα αυτής της αλλαγής γίνονται στο αρχείο *tmp.csv*, το οποίο μετά το πέρας της συνάρτησης διαγράφεται, ώστε να γίνει δυνατή η ενημέρωση του κυρίως αρχείου.
3. Με βάση το δεύτερο πεδίο που δίνει ο χρήστης από το *commandline* καθορίζεται αν θα αλλάξει όλη η εγγραφή ή ένα μέρος της.
4. Στην περίπτωση που επιλεγεί να αλλαχθεί το όνομα ο χρήστης δίνει το όνομα που αλλάζει με το επώνυμο και το όνομα να διαχωρίζονται με κενό. Αυτό ισχύει φυσικά και για όταν αλλάζω ολόκληρη την εγγραφή, το όνομα αποτελείται από 2 πεδία. Προσοχή δεν ισχύει το ίδιο για την χώρα προέλευσης του επιβαίνοντα αν αποτελείται από 2 λέξεις. Για παράδειγμα η χώρα *DominicanRepublic* θα πρέπει να γράφει *Dominican_Republic* ή ως μία λέξη.

Παραδείγματα εκτέλεσης της εφαρμογής:
Αλλαγή ενός μέρους μιας εγγραφής:

```
[stamyk@stamyk-inspiron153520 PROJECT1]$ ./processes_ipc.sh 1278 fullname: Ice Baby
```

```
Change part of record
Showing change....
1280;Ice Baby;73;Latvia;Passenger;yes
```

Για αλλαγή όλης της εγγραφής:

```
[stamyk@stamyk-inspiron153520 PROJECT1]$ ./processes_ipc.sh 1278 record: 1278 Sotiria Bera 50 Greece Passenger Yes
```

```
Information inserted
Showing change....
1278;Sotiria Bera;50;Greece;Passenger;Yes
```

Υλοποίηση *generate_reports*

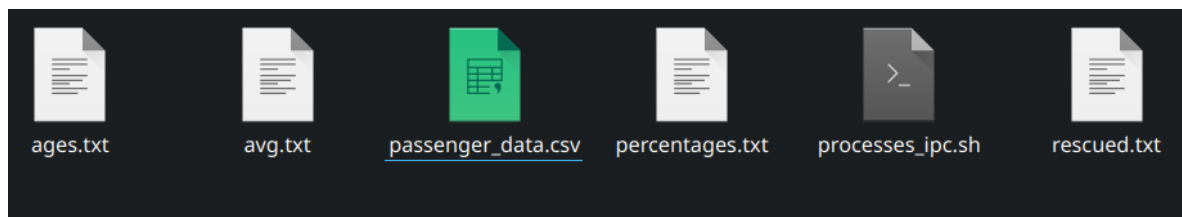
Στην ουσία αυτή η συνάρτηση βγάζει τις καταστάσεις για κάθε περίπτωση

1. Για τον διαχωρισμό των διασωθέντων γίνεται πολύ εύκολα με την *grep*
2. Για τις υπόλοιπες καταστάσεις έχει γίνει η σύμβαση ότι θα υπάρχουν 4 ηλικιακές ομάδες (0-18),(19-35),(36-50) και 51+
3. Η ανάλυση αλλά και η εξαγωγή των αποτελεσμάτων για κάθε μία περίπτωση που ζητείται στο *project* γίνεται με την χρήση *awk*.
4. Μετά το πέρας εξαγωγής αποτελεσμάτων βγάζει ένα σχετικό μήνυμα που ενημερώνει τον χρήστη για τη εξαγωγή των αποτελεσμάτων στα ζητούμενα αρχεία

Παραδείγματα:

```
[stamyk@stamyk-inspiron153520 test]$ ./processes_ipc.sh reports
displaying reports\n
Rescued are stored in file rescued.txt
The age groups insering in a file called ages.txt:
the average per group insering in a file called avg.txt:
the precentage per group for helping the rescue insering in a file called percentages.txt:
```

Μετά το πέρας της συνάρτησης τα αρχεία που δημιουργούνται



Σχόλιο: Τα ποσοστά και οι μέσοι όροι σε κάθε περίπτωση ΔΕΝ έχουν στρογγυλοποιηθεί

Σχόλια για την μορφή του προγράμματος

1. Με το που τρέχει ζητά όπως είναι λογικό το όνομα του αρχείου και μετέπειτα το εκτυπώνει με την ζητούμενη μορφή.
2. Έπειτα ζητά από τον χρήστη αν θέλει να γίνει η αναζήτηση. Η αναζήτηση γίνεται όσες φορές θέλει.
3. Τέλος με βάση το περιεχόμενο που έχει δοθεί στο *commandline* γίνεται είτε η *change_data* είτε η *generate_reports*. Αν δεν δοθεί τίποτα από το τερματικό δεν εκτελείται καμία από της δύο και το πρόγραμμα τερματίζει.

```
echo "Welcome to PassFinder...\n"
echo "Inserting data ..... \n"
insert_data
echo "displaying Data..."
display_file

#choice=$1
#choice can be code for update or reports
# $2 field or record
echo "Do you want to search someone [Y/N]?"
read search
until [ "$search" == "N" ]
do
search_passenger
echo "Do you want to search someone else [Y/N]?"
read search
done

if [ "$1" == "reports" ]
then
echo "displaying reports\n"
generate_reports
elif [ "$2" == "record:" ] || [ "$2" == "fullname:" ] || [ "$2" == "age:" ] || [ "$2" == "country:" ] || [ "$2" == "status:" ] || [ "$2" == "rescued:" ]
then
echo "update passenger"
change_data $1 $2 $3 $4 $5 $6 $7 $8 $9
else
echo "Bye!"
fi
```



```
#!/bin/bash

#####
#####
# Alexios Lazanas, 1100605
# Stamatia Kalimeri, 1103080
#####
#####

#SOS αλλαγή Change_data πιο απλη η 2η awk με mapping
#Το προγραμμα ολο λειτουργει

#read csv data
insert_data(){
    echo "insert the name of the file"
    read filename
    if [ -z "$filename" ]
    then
        echo "The passenger_data.csv is inserted"
        filename=passenger_data.csv
    else
        echo "The file of your choice is inserted"

    fi
}

search_passenger(){
    echo "name or surname of passenger"
    read name
    grep $name -w $filename
}

```

```
change_data(){
    echo "Information inserted"
    code=$1

    #και αλλες μεταβλητες για ολο το record?
    #search and test if code is a number
    if [[ $1 =~ ^[0-9]+$ ]]
    then
        col=1
    else
        echo "please give full_name"
        read code
        col=2
    fi

    if [ $2 == "record:" ]
    then
        awk -F";" -v Code="$code" -v Id=$3 -v Name="$4" -v Surname="$5" -v Age=$6 -v Country="$7" -v Status="$8" -v Rescued="$9" -v Col=$col 'BEGIN{FS=OFS=";"}'
        {
            if($Col==Code){
                Fullname=Name " " Surname;
                $1=Id
                $2=Fullname
                $3=Age
                $4=Country
                $5=Status
                $6=Rescued
            }
        }
    }1' passenger_data.csv > tmp.csv
}

```

```
#!/bin/bash
# passenger_data.csv > tmp.csv
echo "Change part of record"
#actual change data
awk -F";" -v Field=$2 -v Changes=$3 -v ChangeId=$4 -v Code=$code -v Col=$col 'BEGIN{FS=OFS=";"}'
{
    if(Field == "Fullname:")
    {
        if($Col==Code)
        {
            Fullname=Change " " ChangeId;
            $2=Fullname
        }
    }
    else if(Field=="age:")
    {
        if($Col==Code)
        {
            ($3=Change)
        }
    }
    else if(Field=="country:")
    {
        if($Col==Code)
        {
            ($4=Change)
        }
    }
    else if(Field=="status:")
    {
        if($Col==Code)
        {
            ($5=Change)
        }
    }
    else if(Field=="rescued:")
    {
        if($Col==Code)
        {
            ($6=Change)
        }
    }
    else
    {
        print "Not given"
    }
}1' passenger_data.csv > tmp.csv
#το ου αλλαζει υποκει via test
#
echo "Showing change..."
cat tmp.csv > passenger_data.csv

```

```

# αυτο θα αλλαξει υπαρχει για test
fi
echo "Showing change...."
cat tmp.csv > passenger_data.csv
rm tmp.csv
grep "$code" passenger_data.csv

}

#show data
display_file(){
    more $filename #press space to view the file page by page
}

#create reports
generate_reports(){
    #φιλτράρισμα για διασωθέντες
    grep "yes" $filename > rescued.txt

    #ευρεση ηλικιακων ομαδων
    eof=$(wc -l < passenger_data.csv )
    people=1280
    awk -F";" -v EOF=$eof -v People=$people '{
        i=0
        grp1
        grp2
        grp3
        grp4
        mo1
        mo2
        mo3
        mo4
        p1
        p2
        p3
        p4
        while (i <= EOF){
            if($1==i){
                if($3 <= 18)
                    {

```

```

eof=$(wc -l < passenger_data.csv )
people=1280
awk -F";" -v EOF=$eof -v People=$people '{
    i=0
    grp1
    grp2
    grp3
    grp4
    mo1
    mo2
    mo3
    mo4
    p1
    p2
    p3
    p4
    while (i <= EOF){
        if($1==i){
            if($3 <= 18)
            {
                mo1+=$3
                grp1++
                if($5 == "Crew")
                {
                    p1++
                }
            }
            else if($3 >= 19 && $3 <= 35)
            {
                mo2+=$3
                grp2++
                if($5 == "Crew")
                {
                    p2++
                }
            }
            else if($3 >= 36 && $3 <= 50)
            {
                mo3+=$3
                grp3++
                if($5 == "Crew")
                {

```


2 Άσκηση 2 (Συγχρονισμός με Σημαφόρους)

2.1 Εργαλεία που χρησιμοποιήθηκαν (C++)

Η άσκηση έχει υλοποιηθεί με νήματα (threads) της C++ ο κάθε επιβάτης έχει ένα δικό του νήμα ροής εκτέλεσης όπως και κάθε βάρκα. Επειδή προφανώς η άσκηση απαιτεί συγχρονισμό των νημάτων (thread scheduling) χρησιμοποιήθηκαν οι σηματοφόροι που παρέχονται από το API της C++20 `std::counting_semaphore` και για τον αμοιβαίο αποκλεισμό χρησιμοποιήθηκαν `std::mutex` που χειροκίνητα κλειδώνουν και ξεκλειδώνουν όποτε έχουμε πρόσβαση στην κοινή μεταβλητή `psaved`, που μετρά το πλήθος των ανθρώπων που έχουν σωθεί.

Επιλέξαμε την χρήση νημάτων καθώς επιτρέπουν ευκολότερη υλοποίηση για τους σκοπούς της άσκησης που έχουμε κοινή μνήμη και τα νήματα έχουν αυτόματα κοινή μνήμη (αφού ουσιαστικά ανήκουν σε μια διεργασία). Η λογική συγχρονισμού σε μια υλοποίηση με διεργασίες θα ήταν αντίστοιχη απλά θα ορίζαμε ρητά τις κοινές περιοχές μνήμης.

2.2 Θεωρητική περιγραφή του συγχρονισμού και ψευδοκώδικες

- Ένας επιβάτης θα επιβιβαστεί μόνο αν υπάρχει καθίσμα ελεύθερο για αυτόν σε κάποια βάρκα.
- Μια βάρκα ξεκινά άδεια και φεύγει μόνο όταν γεμίσει ή όταν έχουν επιβιβαστεί όλοι οι διαθέσιμοι επιβάτες (σε περίπτωση που περισφύουν καθίσματα σε κάποιο iteration)
- Κάθε επιβάτης επιβιβάζεται μόνο μια φορά στη βάρκα και μόνο όταν έρθει η σειρά του.
- Όλοι οι επιβάτες επιθυμούν να σωθούν.
- Όλες οι βάρκες έχουν την ίδια χωρητικότητα
- Κάθε βάρκα χρειάζεται ένα χρονικό διάστημα για να φτάσει στην ακτή (ΔΕΝ φτάνει αμέσως).

Οι παραπάνω παραδοχές μας οδηγούν στους εξής ψευδοκώδικες.

```
.global_variables
int psaved=0, max_cap=M, boats_num=B;
counting_semaphore_t sem_seat(max_cap*boats_num), sem_pass(0);
mutex_t mtx;
```

Algorithm 1 passenger

```
sem_wait(sem_seat);
mtx_lock(mtx);
psaved = psaved+1;
sem_signal(sem_pass);
mtx_unlock(mtx);
reach_coast();
```

Algorithm 2 boat

```
trip_done = false;
while true do
  mtx_lock(mtx);
  if  $psaved \geq pass\_num$  then
    mtx_unlock(mtx);
    break;
  end if
  mtx_unlock(mtx);
  j=0;
  while true do
    mtx_lock(mtx);
    if  $j \geq max\_cap \vee psaved \geq pass\_num$  then
      mtx_unlock(mtx);
      break;
    end if
    mtx_unlock(mtx);
    sem_wait(sem_pass);
    j=j+1;
  end while
  mtx_lock(mtx);
  if  $j \geq \min(max\_cap, pass\_num - psaved)$  then
    mtx_unlock(mtx);
    trip_done=true;
    reach_coast();
  else
    mtx_unlock(mtx);
  end if
  mtx_lock(mtx);
  remaining = min(max_cap, pass_num-psaved);
  mtx_unlock(mtx);
  sem_signal(sem_seat, remaining);
end while
if  $\neg trip\_done$  then
  reach_coast();
end if
```

Οι αλγόριθμοι περιγράφουν την λογική της άσκησης. Αν ένας επιβάτης επιβιβαστεί σε κάποια βάρκα θεωρείται πως έχει σωθεί και η μεταβλητή *psaved* αυξάνεται για αυτό τον λόγο.

Στην υλοποίηση το να φτάσει μια βάρκα στην ακτή (συνάρτηση *reach_coast()*) παίρνει κάποιο χρονικό διάστημα σε milliseconds, αυτό προσομοιώνεται με την *this_thread::sleep_for()*.

2.3 Ο Κώδικας της Άσκησης και σχόλια σε αυτόν

Αρχείο "ipc_utils.h"

```
.....  
//Alexios Lazanas, 1100605  
//Stamatia Kolimeri, 1103080  
.....  
#ifndef IPC_UTILS_H  
#define IPC_UTILS_H  
#include <iostream>  
#include <list>  
#include <thread>  
#include <semaphore>  
#include <mutex>  
#include <sstream>  
using namespace std;  
  
class passenger{  
public:  
    static list<passenger> pass_queue;  
    int id;  
    bool wants;  
    thread* th;  
  
    passenger(int id);  
    void get_boat();  
    void reach_coast();  
    void join() const;  
    ~passenger();  
};  
  
class boat{  
public:  
    int id;  
    thread* th;  
    static list<boat> boats;  
  
    boat(int id);  
    void get_passenger();  
    void reach_coast();  
    void join() const;  
    ~boat();  
};  
  
#endif // IPC_UTILS_H
```

Στο αρχείο αυτό βρίσκονται οι δηλώσεις των κλάσεων “ επιβάτης” (passenger) και “βάρκα” (boat) καθώς και όλων των συναρτήσεων-μελών αυτών των κλάσεων. Οι συναρτήσεις passenger::get_boat() και boat::get_passenger() είναι εκείνες που καλούνται κατά την εκτέλεση του προγράμματος και τα σώματα των οποίων υλοποιούν τους αλγορίθμους που παρουσιάστηκαν στην προηγούμενη υποενότητα.

Αρχείο "passenger.cpp"

```
.....  
//Alexios Lazanas, 1100605  
//Stamatia Kolimeri, 1103080  
.....  
#include "ipc_utils.h"  
#include <limits>  
#include <chrono>  
using namespace std::chrono;  
  
int psaved=0, boats_num, max_cap, pass_num;  
counting_semaphore<INT_MAX> sem_pass(0), sem_seat(0);  
mutex mtx1, mtx_print1, mtx_print2;  
list<passenger> passenger::pass_queue;  
list<boat> boat::boats;  
  
//functions for passenger  
passenger::passenger(int id){  
    this->id=id;  
    wants=true;  
    th=new thread(&passenger::get_boat, this);  
}  
  
void passenger::get_boat(){  
    sem_seat.acquire();  
    mtx1.lock();  
    psaved++;  
    sem_pass.release();  
    mtx1.unlock();  
    reach_coast();  
}  
  
void passenger::reach_coast(){  
    ostringstream oss;  
    oss<<"Passenger: "<<id<<" reached the coast\n";  
    mtx_print2.lock();  
    cout<<oss.str();  
    mtx_print2.unlock();  
}  
  
void passenger::join() const(  
    if(th->joinable())  
        th->join();  
}  
  
passenger::~passenger(){  
    delete th;  
}
```

Στην εικόνα φαίνονται οι καθολικές μεταβλητές από το αρχείο αλλά και τα σώματα των συναρτήσεων της κλάσης passenger, η συνάρτηση get_boat() υλοποιεί τον αλγόριθμο που παρουσιάστηκε για το νήμα του επιβάτη στην προηγούμενη υποενότητα.

Δύο παραπάνω mutex είναι εμφανή και χρησιμοποιούνται για να είναι ατομική και ολοκληρωμένη η έξοδος στην reach_coast(), δηλαδή να μην υπάρχει interleaving.

Η κλάση `passenger` έχει ως μέλος ένα νήμα (thread) το οποίο εκτελεί την συνάρτηση `get_boat()`. Η συνάρτηση `join()` καλώντας `join()` στο νήμα αυτό το εκτελεί.

```
//Functions for boat
boat::boat(int id){
    this->id=id;
    th=new thread(&boat::get_passenger, this);
}

void boat::get_passenger(){
    bool trip_done=false;
    while(true){
        mtx1.lock();
        if(psaved>pass_num){
            mtx1.unlock();
            break;
        }
        else
            mtx1.unlock();
        int j=0;
        while(true){
            mtx1.lock();
            if((j>max_cap) || (psaved>pass_num)){
                mtx1.unlock();
                break;
            }
            else
                mtx1.unlock();
            sem_pass.acquire();
            j++;
        }
        mtx1.lock();
        if(j>min(max_cap, pass_num-psaved)){
            mtx1.unlock();
            trip_done=true;
            reach_coast();
        }
        else
            mtx1.unlock();
        mtx1.lock();
        int remaining=min(max_cap, pass_num-psaved);
        mtx1.unlock();
        sem_sea.release(remaining);
    }
    if(!trip_done)
        reach_coast();
}

void boat::reach_coast(){
    ostringstream oss;
    oss<<"Boat: "<<id<<" reached coast\n";
    this_thread::sleep_for(milliseconds(14));
    mtx_print1.lock();
    cout<<oss.str();
    mtx_print1.unlock();
}

void boat::join() const{
    if(th->joinable())
        th->join();
}
```

Στην παραπάνω εικόνα βρίσκεται ο κώδικας των συναρτήσεων της κλάσης `boat` όσα γράφτηκαν για το νήμα ροής εκτέλεσης στην κλάση `passenger` ισχύουν και για την κλάση `boat`.

Η συνάρτηση `reach_coast()` αναγκάζει το νήμα που την εκτελεί να περιμένει 14 ms πρώτου εκτυπώσει το μήνυμα έτσι προσομοιώνουμε τον χρόνο που απαιτείται για να φτάσει η βάρκα στην ακτή και να γυρίσει για να είναι και πάλι διαθέσιμη να υποδεχτεί νέους επιβάτες προς διάσωση.

Το Αρχείο "launch.cpp"

```
#include "ipc_utils.h"
using namespace std;
#define MAX_PASS 10
#define MAX_BOAT 10
void initialize();
void join();
void working_pass();
void working_boat();

extern int psaved, boats_num, max_cap, pass_num;
extern counting_semaphore<INT_MAX> sem_sea;

int main(){
    do{
        cout<<"Insert passengerNumber\n";
        cin>>pass_num;
        cout<<"Insert boatsNumber\n";
        cin>>boats_num;
        cout<<"Insert max capacity\n";
        cin>>max_cap;
    }while((pass_num>MAX_PASS || boats_num>MAX_BOAT));
    sem_sea.release(boats_num*max_cap);
    initialize();
    join();
    cout<<"All passengers are now safe\n";
}

void initialize(){
    passenger* ps;
    boat* bt;
    for(int j=1; j<pass_num; j++){
        passenger::pass_queue.emplace_back(j);
    }
    for(int j=1; j<boats_num; j++){
        boat::boats.emplace_back(j);
    }
}

void join(){
    working_boat();
    working_pass();
}

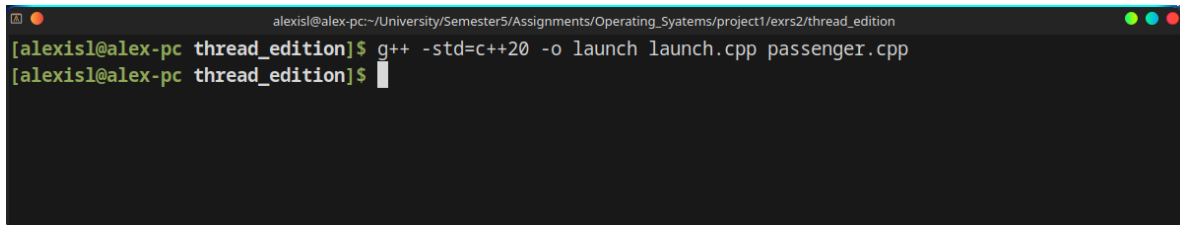
void working_pass(){
    for(auto ps : passenger::pass_queue){
        ps.join();
    }
}

void working_boat(){
    for(auto bt : boat::boats){
        bt.join();
    }
}
```

Για λόγους ασφαλείας (επειδή το πρόγραμμα έρχεται σε ατέρμων βρόχο και δεν τερματίζει αν το πλήθος των νημάτων που δημιουργούνται είναι υψηλό) έχει χρησιμοποιηθεί αμυντικός προγραμματισμός που αναγκάζει τον χρήστη να μην δώσει υψηλό πλήθος επιβατών ή βαρκών.

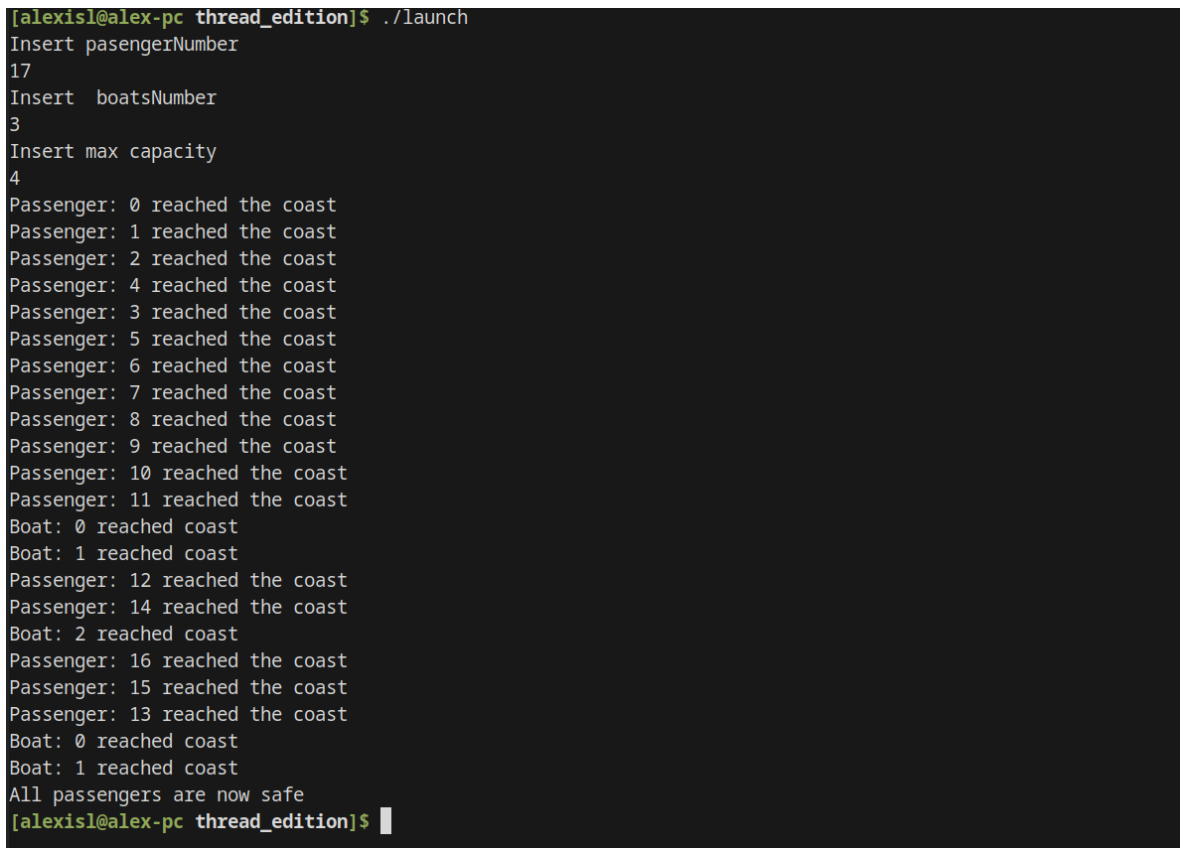
Η συνάρτηση `initialize()` αρχικοποιεί τις λίστες πλοίων και επιβατών ενώ οι συναρτήσεις `working_pass()` και `working_boat()` εκκινούν τα νήματα ροής για κάθε επιβάτη ή βάρκα αντίστοιχα, η συνάρτηση `join()` είναι εκείνη που καλεί αυτές τις δυο συναρτήσεις και τελικά καλείται από την `main` και έτσι εκκινούνται τα νήματα ροής εκτέλεσης.

Η χρήση των σημαφόρων ως εργαλεία μας παρέχεται από την `C++20` και έπειτα συνεπώς στην εντολή για `compilation` πρέπει να τοποθετηθεί ανάλογο `flag`



```
alexisl@alex-pc:~/University/Semester5/Assignments/Operating_Systems/project1/exrs2/thread_edition
[alexisl@alex-pc thread_edition]$ g++ -std=c++20 -o launch launch.cpp passenger.cpp
[alexisl@alex-pc thread_edition]$
```

Η εκτέλεση του προγράμματος μας δίνει ένα αποτέλεσμα της παρακάτω μορφής.



```
[alexisl@alex-pc thread_edition]$ ./launch
Insert passengerNumber
17
Insert boatsNumber
3
Insert max capacity
4
Passenger: 0 reached the coast
Passenger: 1 reached the coast
Passenger: 2 reached the coast
Passenger: 4 reached the coast
Passenger: 3 reached the coast
Passenger: 5 reached the coast
Passenger: 6 reached the coast
Passenger: 7 reached the coast
Passenger: 8 reached the coast
Passenger: 9 reached the coast
Passenger: 10 reached the coast
Passenger: 11 reached the coast
Boat: 0 reached coast
Boat: 1 reached coast
Passenger: 12 reached the coast
Passenger: 14 reached the coast
Boat: 2 reached coast
Passenger: 16 reached the coast
Passenger: 15 reached the coast
Passenger: 13 reached the coast
Boat: 0 reached coast
Boat: 1 reached coast
All passengers are now safe
[alexisl@alex-pc thread_edition]$
```

Παρατηρούμε ότι όλοι οι επιβάτες σώζονται και ότι οι βάρκες συνολικά φτάνουν στην ακτή 5 φορές όπου $(4*5)=20$ δηλαδή τις ελάχιστες.

Δίνεται άλλο ένα *demo* της εξόδου του προγράμματος για μια άλλη είσοδο.


```

[alexisl@alex-pc thread_edition]$ ./launch
Insert passengerNumber
23
Insert boatsNumber
5
Insert max capacity
3
Passenger: 1 reached the coast
Passenger: 2 reached the coast
Passenger: 0 reached the coast
Passenger: 3 reached the coast
Passenger: 4 reached the coast
Passenger: 5 reached the coast
Passenger: 6 reached the coast
Passenger: 7 reached the coast
Passenger: 8 reached the coast
Passenger: 9 reached the coast
Passenger: 10 reached the coast
Passenger: 11 reached the coast
Passenger: 12 reached the coast
Passenger: 13 reached the coast
Passenger: 14 reached the coast
Boat: 1 reached coast
Boat: 0 reached coast
Boat: 2 reached coast
Passenger: 15 reached the coast
Passenger: 22 reached the coast
Passenger: 16 reached the coast
Passenger: 20 reached the coast
Passenger: 18 reached the coast
Boat: 3 reached coast
Boat: 4 reached coast
Passenger: 21 reached the coast
Passenger: 19 reached the coast
Passenger: 17 reached the coast
Boat: 1 reached coast
Boat: 0 reached coast
Boat: 2 reached coast
All passengers are now safe
[alexisl@alex-pc thread_edition]$

```

Και εδώ παρατηρούμε ότι σώζονται όλοι οι επιβάτες και οι βάρκες που χρησιμοποιούνται είναι ξανά οι ελάχιστες δυνατές $(8 \cdot 3) = 24$.

Υποσημείωση: Σε όλες τις εισόδους που ελέγξαμε το πρόγραμμα (που ήταν αρκετές) παρατηρήσαμε ότι χρησιμοποιούνταν πάντοτε ο ελάχιστος δυνατός αριθμός καραβιών που έφταναν στην ακτή (δεδομένου ότι τα καθίσματα των βαρκών συνολικά είναι λιγότερα από το συνολικό πλήθος των προς διάσωση επιβατών).

3 Άσκηση 3 (Χρονοπρογραμματισμός Διεργασιών)

3.1 Εργαλεία που χρησιμοποιήθηκαν (Γλώσσα C++)

Η άσκηση έχει ως σκοπό να προσομοιάσει την χρονοδρομολόγηση διεργασιών σε ένα σύστημα με έναν επεξεργαστή και 512 *KiB* μνήμης. Ένας μετρητής κρατά τον χρόνο που περνά και έτσι εξομοιώνουμε την πάροδο του χρόνου στο σύστημα. Όταν μια διεργασία εκτελείται καλείται η συνάρτηση `std::this_thread::sleep_for()` η συνάρτηση αυτή αντικαθιστά την εκτέλεση της διεργασίας που μόλις επιλέχθηκε στην ουσία το πρόγραμμα κοιμάται για το απαραίτητο χρονικό διάστημα ίσο με αυτό που θα γινόταν η εκτέλεση τη εργασίας που επιλέχθηκε από τον δρομολογητή. Συνεπώς χρησιμοποιήσαμε την βιβλιοθήκη `std::chrono` για να μετρά την διάρκεια σε *s* όταν το πρόγραμμα κοιμάται (παρόλο που η άσκηση δηλώνει *ms* εμείς βάλουμε δευτερόλεπτα για να είναι η καθυστέρηση εμφανής).

3.2 Θεωρητική περιγραφή αλγορίθμου χρονοπρογραμματισμού (Round-Robin)

Ο αλγόριθμος Round-Robin εκτελεί τις διεργασίες με τρόπο κυκλικό. Δηλαδή βγάζει την πρώτη διεργασία από την ουρά την εκτελεί για `min(remaining_time_of_current_process, time_slice)` και έπειτα αν η διεργασία ΔΕΝ έχει ολοκληρωθεί την επανατοποθετεί στην ουρά των διεργασιών. Κάθε πρόγραμμα άπαξ και λάβει *CPU* για πρώτη φορά εισέρχεται στην μνήμη και φεύγει μόνο όταν ολοκληρωθεί. Αν μια διεργασία διακοπεί λόγω λήξης του χρονομεριδίου την ίδια χρονική στιγμή με αυτή όπου μια διεργασία εισέρχεται στο σύστημα τότε έχει προτεραιότητα η νέα διεργασία (εκείνη που μόλις εισέρχεται στο σύστημα) Για αυτό μετά σε κάθε επανάληψη στο τέλος τοποθετούνται στην ουρά όλες οι διεργασίες όσες είχαν χρόνο άφιξης \leq από την τρέχουσα χρονική στιγμή.

Παρακάτω βρίσκεται ο αλγόριθμος ενδεικτικά σε ψευδοκώδικα:

```
.global_variables
list_t all_process;
circ_queue_t pqueue, memory;
int num_process, curr_time=0;

function RROBIN
    exec=all_process.top();
    if exec.arrival_time≤curr_time then
        all_process.pop();
        pqueue.enqueue(exec);
    else if pqueue is Empty then
        wait(exec.arrival_time);
        all_process.pop();
        pqueue.enqueue(exec);
    end if
label:
while num_processes>0 do
    to_exec=pqueue.dequeue();
    exec.time= min(TIME_SLICE, to_exec.remaining_time);
    if to_exec.remaining_time==to_exec.duration then
        if enough_memory then
            to_exec.reserve_memory()
        else
            pqueue.enqueue(to_exec);
            goto label
        end if
    end if
    to_exec.remaining_time=to_exec.remaining_time-exec_time;
```

```

execute(to_exec);
Insert all processes that arrival_time ≤ curr_time from all_process to pqueue;
if to_exec finished then
    remove to_exec from memory;
else
    pqueue.enqueue(to_exec);
end if
end while
end function

```

Είναι προφανές ότι αν μια διεργασία δεν χωρά στην μνήμη μια δεδομένη χρονική στιγμή που προσπαθεί να δεσμεύσει, τότε μεταφέρεται στο τέλος της ουράς και μεταβαίνουμε στην επόμενη. Αυτή η τακτική είναι προφανές ότι μπορεί να οδηγήσει σε ατέρμων βρόχο σε μία μόνο περίπτωση. Αν σε όλες τις χρονικές στιγμές δεν υπάρχει στο σύστημα επαρκής μνήμη για να εξυπηρετήσει μια σειρά διεργασιών και έτσι αυτές δεν μπορούν να εκτελεστούν. Αυτό μπορεί να συμβεί μόνο αν ΚΑΘΕΜΙΑ από αυτές τις διεργασίες ζητά μνήμη μεγαλύτερη από εκείνη του συστήματος.

3.3 Θεωρητική περιγραφή του αλγορίθμου για διάθεση μνήμης στις διεργασίες

Στο σύστημα που προσομοιώνουμε η μνήμη θεωρείται ως ένα διάνυσμα από *memory_block*. Αρχικά έχουμε ένα μπλοκ μνήμης μεγέθους 512KiB ελεύθερο. Ο αλγόριθμος πρώτου ταιριάσματος πρακτικά διασπά το πρώτο επαρκούς μεγέθους ελεύθερο μπλοκ που θα εντοπίσει σε 2 νέα μπλοκ, το πρώτο από αυτά είναι δεσμευμένο από την διεργασία που μόλις εισήλθε στην μνήμη και έχει μέγεθος όσο αυτή χρειάζεται, το δεύτερο είναι ελεύθερο και έχει μέγεθος όσο απέμεινε.

Συνεπώς ορίσαμε τους τύπους δεδομένων *memory_block*, *process*, *b_vector* για το μπλοκ μνήμης, την διεργασία και το διάνυσμα που είναι η μνήμη του συστήματος. Στο πρόγραμμά μας το η κλάση για διάνυσμα της μνήμης είναι εκείνη που διαθέτει συναρτήσεις για την δέσμευση σε μια διεργασία και την αποδέσμευση, όχι διεργασία η ίδια όπως θα έβγαζε κανείς συμπέρασμα από τον ψευδοκώδικα της προηγούμενης υποενότητας.

Αυτοί οι τύποι δεδομένων είναι δηλωμένοι και υλοποιημένοι στο αρχείο *memory_utils.cpp* και είναι όπως παρακάτω:

```

class memory_block{
public:
    size_t start;
    size_t length;
    bool free;
    size_t* owner_pid;
    memory_block(int length, int j){
        free=true;
        this->length=length;
        owner_pid=nullptr;
        start=j;
    }
    memory_block(size_t owner_pid, int length, int j){
        free=false;
        [this->owner_pid=new size_t;
        *(this->owner_pid)=owner_pid;
        this->length=length;
        start=j;
    }
    bool operator == (memory_block& other){
        return(*(this->owner_pid)==*(other.owner_pid));
    }
};

class process{
public:
    size_t pid;
    size_t arrival_time;
    size_t duration;
    size_t remaining_time;
    size_t memory_needed;
    bool in_memory;
    process(){
        process(size_t pid, size_t arrival_time, size_t duration, size_t memory_needed){
            this->pid=pid;
            [this->arrival_time=arrival_time;
            [this->duration=duration;
            [this->memory_needed=memory_needed;
            this->in_memory=false;
            this->remaining_time=duration;
        }
        ~process(){}
    };
};

```

```

class b_vector{
    vector<memory_block> memory;
public:
    b_vector(){
        memory_block mb(MAX_SIZE, 0);
        memory.push_back(mb);
    }

    bool get_proc(process& proc){
        if(memory.size()>=MAX_SIZE){
            return false;
        }

        for(size_t j=0; j<memory.size(); ++j){
            if(memory[j].free && memory[j].length==proc.memory_needed){
                memory_block mb_proc(proc.pid, proc.memory_needed, j);
                memory[j]=mb_proc;
                return true;
            }
            else if(memory[j].free && memory[j].length>proc.memory_needed){
                memory_block mb_proc(proc.pid, proc.memory_needed, j);
                memory_block mb_free(memory[j].length-proc.memory_needed, j+1);
                memory.erase(memory.begin()+j);
                memory.insert(memory.begin()+j, mb_proc);
                memory.insert(memory.begin()+j+1, mb_free);
                return true;
            }
        }
        return false;
    }

    void remove_proc(process& proc){
        memory_block to_remove(proc.pid, 0, 0);

        for(size_t j=0; j<memory.size(); ++j){
            if((memory[j].owner_pid && to_remove.owner_pid) && (memory[j]==to_remove)){
                memory[j].free=true;
                memory[j].owner_pid=nullptr;
            }
        }
        auto it=memory.begin();
        int j=0;
        while(it!=memory.end()-1){
            if((*it).free && (*(it+1)).free){
                memory_block mb_new_b((*it).length+(*(it+1)).length, j);
                it=memory.erase(it);
                it=memory.erase(it);
                it=memory.insert(it, mb_new_b);
            }
            else{
                ++it;
            }
            ++j;
        }
    }

    void print_vector(){
        cout<<"[";
        for(memory_block& mb : memory){
            if(mb.owner_pid){
                cout<<"{ "<<*mb.owner_pid<<"} ";
            }
            cout<<(" "<<mb.length<<"),";
        }
        cout<<"]\n";
    }
};

```

Είναι λογικό ότι επειδή γίνεται η χρήση συναρτήσεων όπως `vector::insert()`, `vector::erase()`. Η δέσμευση μνήμης για μια διεργασία απαιτεί $O(n)$ χρόνο (όπου n το πλήθος των μπλοκ μνήμης), επειδή η διαδικασία σταματά και επιστρέφει όταν γίνεται δέσμευση μνήμης. Αντίθετα η αποδέσμευση απαιτεί συγχώνευση όλων των διαδοχικών ελεύθερων μπλοκ που με απλοϊκή ανάλυση πολυπλοκότητας ανεβάζει την πολυπλοκότητα της διαδικασίας αυτής σε $O(n^2)$.

3.4 Ο κώδικας της άσκησης και σχόλια σε αυτόν

Το αρχείο `main.cpp` περιέχει την συνάρτηση για τον αλγόριθμο εκ περιτροπής δρομολόγησης και το κυρίως πρόγραμμα. Από ένα `txt` αρχείο που περιέχει διεργασίες ανά γραμμή ως εξής: `pid`, `arrival_time`, `duration`, `memory_needed` (οι διεργασίες είναι στο αρχείο οπωσδήποτε ταξινομημένες κατά αύξουσα χρονική στιγμή άφιξης) διαβάζονται τα στοιχεία των διεργασιών προς δρομολόγηση. Να τονίσουμε ότι αθροιστικά οι διεργασίες στο `txt` μπορούν να απαιτούν περισσότερη μνήμη από αυτή του συστήματος χωρίς πρόβλημα αρκεί καθεμιά από αυτές να μην απαιτεί από μόνη της περισσότερη μνήμη από εκείνη που το σύστημα διαθέτει.

```
#include "memory_utils.cpp"
#include <string>
#include <sstream>
#include <fstream>
#include <algorithm>
#include <thread>
#include <list>
#include <chrono>
#define TIME_SLICE 3
using namespace std;

list<process> all_process;
list<process> pqueue;
b_vector memory;
size_t num_process;
void initialize();
void Rrobin();

size_t curr_time;

int main(){
    initialize();
    Rrobin();
}

void initialize(){
    string fname = "processes.txt";
    ifstream process_file(fname);

    if(process_file.is_open()){
        string line;
        process* proc;
        while(getline(process_file, line)){
            vector<string> row;
            stringstream ss(line);
            string cell;
            while(getline(ss, cell, ',')) {
                row.push_back(cell);
            }
            proc = new process(stoi(row[0]), stoi(row[1]), stoi(row[2]), stoi(row[3]));
            all_process.push_back(*proc);
        }
        num_process=all_process.size();
    }
}
```

```

void Rrobin(){
    auto executable = all_process.front();
    if(executable.arival_time<=curr_time){
        all_process.pop_front();
        pqueue.push_back(executable);
    }
    else if(pqueue.empty()){
        this_thread::sleep_for(chrono::milliseconds(executable.arival_time));
        curr_time+=executable.arival_time;
        all_process.pop_front();
        pqueue.push_back(executable);
    }
    label:
    while(num_process>0){
        auto to_exec = pqueue.front();
        pqueue.pop_front();
        size_t exec_time = min((size_t)TIME_SLICE, to_exec.remaining_time);
        if(to_exec.remaining_time==to_exec.duration){
            if(memory.get_proc(to_exec)){
                to_exec.remaining_time-=exec_time;
                memory.print_vector();
            }
            else{
                pqueue.push_back(to_exec);
                cerr<<"memory not enough for process: "<<to_exec.pid<<" proceeding to the next\n";
                goto label;
            }
        }
        else{
            to_exec.remaining_time-=exec_time;
        }
        cout<<"executing "<<exec_time<<" of process: "<<to_exec.pid<<endl;
        this_thread::sleep_for(chrono::seconds(exec_time));
        curr_time+=exec_time;
        while(!all_process.empty() && all_process.front().arival_time<=curr_time){
            auto ready = all_process.front();
            all_process.pop_front();
            pqueue.push_back(ready);
        }
        if(to_exec.remaining_time<=0){
            memory.remove_proc(to_exec);
            num_process--;
            memory.print_vector();
        }
        else{
            pqueue.push_back(to_exec);
        }
    }
}

```

Ο αλγόριθμος εκ περιτροπής δρομολόγησης ακολουθεί την λογική του ψευδοκώδικα που περιγράφηκε σε προηγούμενη υποενότητα. Για κάθε εκτέλεση νέας διεργασίας εμφανίζεται το *pid* της και ο χρόνος που χρησιμοποιήθηκε στη τρέχουσα εκτέλεση για αυτήν, επιπλέον εμφανίζεται η κατάσταση της μνήμης μετά από κάθε πράξη δέσμευσης ή αποδέσμευσης.

Το περιεχόμενο του αρχείου που παραδίδεται στο παραδοτέο μας είναι το εξής:

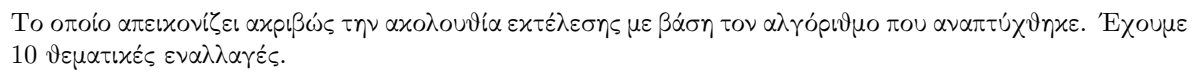
```

3, 0, 6, 85
1, 2, 4, 84
2, 3, 1, 82
5, 4, 3, 87
4, 5, 5, 81
6, 6, 7, 95
|

```

Η εκτέλεση του προγράμματος για τα δεδομένα του αρχείου *txt* που παραδόθηκε στο *zip*, είναι:

Η οποία αν απεικονισθεί σε διάγραμμα *Gantt* μας δίνει το παρακάτω:



4 Άσκηση 4 (Διαγράμματα *Gantt*, Μέσοι χρόνοι διεκπεραίωσης, αναμονής, απόδοσης)

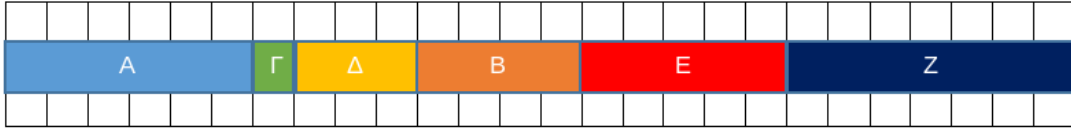
Παρακάτω καταγράφονται τα ζητούμενα της άσκησης 4. Με MXA (Μέσος χρόνος αναμονής), $MXA\pi$ (Μέσος Χρόνος Απόδοσης), MXO (Μέσος χρόνος ολοκλήρωσης) και $\Pi\Theta E$ (Πλήθος Θεματικών Εναλλαγών)

1. Για τον Αλγόριθμο *FCFS*:



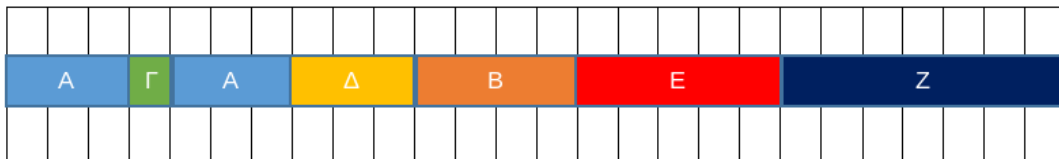
- $MXA = \frac{XA(A)+XA(B)+XA(\Gamma)+XA(\Delta)+XA(E)+XA(Z)}{6} = \frac{0+4+7+7+9+13}{6} \approx 6.66$
- $MXA\pi = \frac{XA\pi(A)+XA\pi(B)+XA\pi(\Gamma)+XA\pi(\Delta)+XA\pi(E)+XA\pi(Z)}{6} = \frac{0+4+7+7+9+13}{6} \approx 6.66$
- $MXO = \frac{XO(A)+XO(B)+XO(\Gamma)+XO(\Delta)+XO(E)+XO(Z)}{6} = \frac{6+8+8+10+14+20}{6} = 11$
- $\Pi\Theta E = 5$

2. Για τον Αλγόριθμο *SJF*



- $MXA = \frac{XA(A)+XA(B)+XA(\Gamma)+XA(\Delta)+XA(E)+XA(Z)}{6} = \frac{0+8+3+3+9+13}{6} = 6$
- $MXA\pi = \frac{XA\pi(A)+XA\pi(B)+XA\pi(\Gamma)+XA\pi(\Delta)+XA\pi(E)+XA\pi(Z)}{6} = \frac{0+8+3+3+9+13}{6} = 6$
- $MXO = \frac{XO(A)+XO(B)+XO(\Gamma)+XO(\Delta)+XO(E)+XO(Z)}{6} = \frac{6+11+4+6+14+20}{6} \approx 10.16$
- $\Pi\Theta E = 5$

3. Για τον Αλγόριθμο *SRTF*



- $MXA = \frac{XA(A)+XA(B)+XA(\Gamma)+XA(\Delta)+XA(E)+XA(Z)}{6} = \frac{1+8+0+3+9+13}{6} \approx 5.66$
- $MXA\pi = \frac{XA\pi(A)+XA\pi(B)+XA\pi(\Gamma)+XA\pi(\Delta)+XA\pi(E)+XA\pi(Z)}{6} = \frac{0+0+3+8+9+13}{6} = 5.5$
- $MXO = \frac{XO(A)+XO(B)+XO(\Gamma)+XO(\Delta)+XO(E)+XO(Z)}{6} = \frac{7+1+6+12+14+20}{6} = 10$
- $\Pi\Theta E = 10$

