Μέρος Α

Ερώτημα Α

(Α)

Οι βιβλιοθήκες μου οι οποίες χρειάζονται για τις διεργασίες μου, τον κοινά διαμοιραζόμενο πίνακά μου και την μέτρηση των κύκλων της CPU.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/ipc.h>
#include <sys/types.h>
#include <sys/shm.h>
#include <sys/wait.h>
#include <time.h>
```

Main:

Αρχικά ορίζω τις μεταβλητές

```
int main ()
{
int size;
int i;
int anything=123;
int *p=&anything;
clock_t start, end, totaltime;
```

Έπειτα σκανάρω το μέγεθος του πίνακά μου.

```
//scan size of array
printf("Give size of array: ");
scanf ("%d",&size);
```

Δημιουργώ τον πίνακά μου, εκχωρώντας μνήμη για ακέραιους και επιστρέφοντάς τον ως δείκτη, και σκανάρω τις τιμές του πίνακα .

```
int *get_num_arr = malloc (size * sizeof (int));
    for (i = 0; i < size; ++i){
        printf("give array values: ");
        scanf ("%d", &get_num_arr[i]);
        }
```

Ξεκινάω να μετράω τον χρόνο αθροίσματος.

```
start = clock();
```

Αποκτώ ένα τμήμα διαμοιρασμένης μνήμης με κλειδί=14.

```
int shared_memory= shmget (14, size * sizeof (int), IPC_CREAT | 0666);;
```

Ο αριθμός 0666 ορίζει τα δικαιώματα πρόσβασης ενώ το IPC_CREAT ειδοποιεί το σύστημα να δημιουργήσει ένα νέο τμήμα μνήμης για την διαμοιρασμένη μνήμη.

Προσκολλώ ένα νέο πίνακά τον new_array στο τμήμα της διαμοιρασμένης μνήμης.

```
int *new_array = shmat (shared_memory, NULL, 0);
```

Για κάθε θέση του πίνακα δημιουργώ μία νέα διεργασία-παιδί και 'γεμίζω' τον new_array με τις τιμές του αρχικού μου πίνακα.

```
for (i = 0; i < size; ++i){

      if (fork ()==0)
      {
              new_array[i] = get_num_arr[i];
          return 0;
      }
  }
```

Περιμένω να δημιουργηθούν όλες οι διεργασίες μου.

```
 for (i = 0; i < size; ++i)
     wait (&i);
```

Προσθέτω τις τιμές του πίνακά μου στον δείκτη p και σταματάω να μετράω χρόνο.

```
for (i = *p = 0; i < size; ++i){
    *p += new_array[i];
    }
    end = clock();//end of counting
```

Εκτυπώνω το άθροισμα (τον δείκτη p) και τον συνολικό χρόνο σε δευτερόλεπτα.

```
printf ("Total: %d\n", *p);
totaltime = (double)(end - start) / CLOCKS_PER_SEC;
printf("Total time taken by CPU: %f\n", totaltime );
```

Απελευθερώνω το τμήμα διαμοιρασμένης μνήμης και τον αρχικό πινακα μου.

```
    shmdt (new_array);
    shmctl (shared_memory, IPC_RMID, NULL);

    free (get_num_arr);
    return 0;
}
```

Το IPC_RMID καταστρέφει το τμήμα διαμοιρασμένης μνήμης αφού πρώτα απελευθερωθεί.

Ένα παράδειγμα αποτελέσματος είναι το εξής:

Give size of array: 3

give array values: 5

give array values: 5

give array values: 5

Total: 15

Total time taken by CPU: 0.000000

(B)

Το πρόγραμμά μου είναι το ίδιο με το Α ερώτημα με τη διαφορά ότι δεν αποκτώ τμήμα διαμοιρασμένης μνήμης και επομένως δεν προσκολλώ τον πίνακα μου σε αυτό το τμήμα. Εφόσον η μνήμη δεν είναι διαμοιρασμένη δεν μπορώ να έχω πρόσβαση σε όλες τις τιμές αφού βρίσκονται σε διαφορετικές διεργασίες.

Με αυτόν τον τρόπο όμως δεν έχω το επιθυμητό αποτέλεσμα.

Ένα παράδειγμα αποτελέσματος είναι το εξής:

Give size of array: 3

give array values: 5

give array values: 5

give array values: 5

Total: 1

Total time taken by CPU: 0.000000

Ερώτημα Β

Αρχικά δηλώνω τις βιβλιοθήκες οι οποίες χρειάζονται για τα threads , τους semaphores, την εντολή sleep και τη Boolean μεταβλητή μου.

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <unistd.h>
#include <semaphore.h>
#include <stdbool.h>
```

```
void DecideWhichMaterialsToSell(int x){

printf("\nTo sell paper and matches press 1 ,to sell tabacoo and matches press 2 , for anything else press 3\n");
scanf("%d",&x);
count=x-1;// value-1 of choice saved in global count
printf("the item is: %s\n",smoker_types[count]);
}
```

Στη συνέχεια δηλώνω τις μεταβλητές :

```
int count;
sem_t sell_sem;
sem_t smoker_sem[3];
char* smoker_types[3] = { "paper and matches", "tabacoo and matches ", "anything else" };
bool table[3]={true,true,true};
```

Έπειτα έχω τη συνάρτηση DecideWhichMaterialToSell που σκανάρει την επιλογή του χρήστη και αποθηκεύσει την τιμή που δόθηκε μείον 1 στην global μεταβλητή count.

Ακολουθεί η συνάρτηση TakesMaterialsFromtable που εκτυπώνει ποιος καπνιστής και ποιο είδος προϊόντος αγόρασε.

```
void TakeMaterialsFromtable(){
printf("=====================================================\n");
printf("smoker %d takes %s from table\n",count+1,smoker_types[count]);
printf("=====================================================\n");

}
```

Οι 3ις threads των καπνιστών είναι ως εξής:

```
void *mycustomerThread1(void *passedthreadID) //smoker thread 1
{

    while(1){

    sem_wait(&smoker_sem[0]); //down semaphore of 1st smoker
        sleep(1);

        TakeMaterialsFromtable();

        sem_post(&sell_sem); //up seller's semaphore

    }

}
void *mycustomerThread2(void *passedthreadID) //smoker thread 2
{

    while(1){
    sem_wait(&smoker_sem[1]); //down semaphore of 2nd smoker
        sleep(1);

        TakeMaterialsFromtable();

        sem_post(&sell_sem);//up seller's semaphore

    }

}
```

```
}
void *mycustomerThread3(void *passedthreadID) //smoker thread 3
{

    while(1){
    sem_wait(&smoker_sem[2]); //down semaphore of 3rd smoker
        sleep(1);

        TakeMaterialsFromtable();

        sem_post(&sell_sem);//up seller's semaphore

    }

}
```

Όπου η τιμή του σημαφόρου του αντίστοιχου καπνιστή μειώνεται κατά 1 ,καλείται η συνάρτηση TakesMaterialsFromtable και αυξάνεται η τιμή του σημαφόρου του καπνιστή.

Η thread του πωλητή έχει ως εξής:

```
void *mysellerThread(void *passedthreadID) //seller thread
{

    while(1){

    sem_wait(&sell_sem);//down seller's semaphore
    sleep(1);
    int pr;
    DecideWhichMaterialsToSell(pr);
    sleep(1);

    if(count==0 && table[0]==true){
    table[0]=false; //value in position 0 turns false because 1st product in no longer available
    sem_post(&smoker_sem[0]);// up semaphore of the thread of 1st smoker
    }
    else if(count==1 && table[1]==true){
    table[1]=false;//value in position 1 turns false because 2nd product in no longer available
    sem_post(&smoker_sem[1]);// up semaphore of the thread of 2nd smoker
    }
    else if(count==2 && table[2]==true){
    table[2]=false;//value in position 2 turns false because 3rd product in no longer available
    sem_post(&smoker_sem[2]);// up semaphore of the thread of 3rd smoker
    }
    else{
    printf("The %s is no longer available\n",smoker_types[count]);//requested product is no longer available
    exit(0);//end running of program
    }


    }

}
```

Όπου καλείται η DecideWhichMaterialToSell και έπειτα μέσω μιας if και ανάλογα με την επιλογή του χρήστη και την τιμή της μεταβλητής table είτε αυξάνεται η τιμή του αντίστοιχου σημαφόρου και ενημερώνεται η τιμή του table είτε ο χρήστης ενημερώνεται ότι το προϊόν δεν είναι διαθέσιμο και το πρόγραμμα τερματίζεται.

Ορίζω τις μεταβλητές:

```
int p;
int k=9;
```

Τις οποίες χρησιμοποιώ στην main.

Main:

Ορίζω τις 3ις threads των καπνιστών και του πωλητή ,τον πίνακα a που χρησιμοποιώ στη αρχικοποίηση των threads των καπνιστών και τέσσερις int μεταβλητές για την αρχικοποίηση των threads.

```
int main()
{

    int a[3] = {0,1,2};
    pthread_t threads1;
    pthread_t threads2;
    pthread_t threads3;
    pthread_t sellerthread;

    int customer1;
    int customer2;
    int customer3;
    int seller;
```

Αρχικοποιώ τον σημαφόρο του πωλητή και δημιουργώ τη thread του.

```
   sem_init(&sell_sem,1,1);



   seller=pthread_create(&sellerthread,NULL,mysellerThread,(void*)&k);
```

Αρχικοποιώ τους σημαφόρους και δημιουργώ τις threads των καπνιστών.

```
  for(p=0;p<NUM_THREADS;p++){
   //smoker's semaphores initialaization
      sem_init(&smoker_sem[p],1,0);
  }
  //creating 3 threads for 3 customers
  customer1=pthread_create(&threads1, NULL, mycustomerThread1, (void *)&a[1]);
  customer2=pthread_create(&threads2, NULL, mycustomerThread2, (void *)&a[2]);
  customer3=pthread_create(&threads3, NULL, mycustomerThread3, (void *)&a[3]);
```

Τερματίζω την λειτουργεία των threads μου.

```
  pthread_join(threads1,NULL);
  pthread_join(threads2,NULL);
  pthread_join(threads3,NULL);
  pthread_join(sellerthread,NULL);
```

Καταστρέφω τους σημαφόρους μου επειδή πλέον δε τους χρειάζομαι.

```
    sem_destroy(&sell_sem);
    for(p=0;p<3;p++){
    sem_destroy(&smoker_sem[p]);
    }


    return 0;
}
```

Ερώτημα Γ

Στην αρχή, δηλώνω τις βιβλιοθήκες μου, οι οποίες χρειάζονται για τα threads, τους semaphores και την εντολή sleep.

```
1 #include<string.h>
2 #include<stdio.h>
3 #include<stdlib.h>
4 #include<unistd.h>
5 #include<pthread.h>
6 #include<semaphore.h>
```

Στην συνέχεια, δηλώνω τους semaphores (2).

```
8 sem_t semaphore1;
9 sem_t semaphore2;
```

Έπειτα, είναι η δήλωση των δύο διεργασιών μου, P και Q.

```
44 void* Q(void* args) {

11 void* P(void* args) {
```

Διεργασίες :

Αρχικά, έχω βάλει την εντολή printf με σύμβολα ανάμεσα από κάθε εντολή της διεργασίας, έτσι ώστε να είναι εμφανής η εναλλαγή μεταξύ εντολών στις δύο διεργασίες. Αυτό ισχύει και για τον κώδικα στην διεργασία Q.

```
printf("================================================================================\n");
```

E1 : Μου εμφανίζει τους φακέλους στον τρέχων φάκελο.

```
14          printf("Files in directory are :\n"); //E1
15          system("ls -l");
```

Sleep(2) : Παύση δύο δευτερολέπτων μέχρι την επόμενη εντολή.

```
17          sleep(2);
```

Στην συνέχεια, βλέπουμε για πρώτη φορά εντολή για semaphore. Η συγκεκριμένη αυξάνει την τιμή του semaphore 1 σε 1 από 0 κι έτσι η διεργασία P έχει αποκλειστικότητα στις εντολές, μέχρι να κατέβει η τιμή του semaphore 1 ξανά στο 0 ή ο semaphore 2 να λάβει αρνητική τιμή.

```
18          sem_post(&semaphore1);
```

E5 : Εμφανίζει τις τρέχουσες διεργασίες του συστήματος.

```
20          printf("The processes running in the system are :\n"); //E5
21          system("ps -l");
```

```
34          sleep(2); 23          sleep(2);
```

Αύξηση της τιμής του semaphore 1 κατά 1 (πλέον 2).

```
18          sem_post(&semaphore1);
```

E8 : Δείχνει τον τρέχοντα φάκελο.

```
26          printf("The current working directory is :\n"); //E8
27          system("pwd");
```

Παύση του συστήματος για 2 δευτερόλεπτα.

```
29          sleep(2);
```

E9 : Δημιουργία φακέλου με όνομα CEID.

```
31          printf("Creating a directory named CEID. \n"); //E9
32          system("mkdir CEID");
```

Παύση του συστήματος για 2 δευτερόλεπτα.

```
34          sleep(2);
```

Ύστερα, η τιμή του semaphore 2 μειώνεται κατά 1, οπότε γίνεται αρνητική. Άρα, η αποκλειστικότητα της διεργασίας P σταματά και πάμε στην διεργασία Q.

```
35          sem_wait(&semaphore2);
```

Μειώνεται η τιμή του semaphore 1 κατά 1 (1).

```
46          sem_wait(&semaphore1);
```

E2 : Μας εμφανίζει τις πρώτες 10 γραμμές του κώδικα.

```
47          printf("The first 10 lines of askisi3.c file are :\n"); //E2
48          system("head -10 askisi3.c");
```

Παύση του συστήματος για 2 δευτερόλεπτα.

```
50          sleep(2);
```

E3 : Μας εμφανίζει τις τελευταίες 10 γραμμές του κώδικα.

```
52        printf("The last 10 lines of askisi3.c file are :\n"); //E3
53        system("tail -10 askisi3.c");
```

Παύση του συστήματος για 2 δευτερόλεπτα.

```
55        sleep(2);
```

Μειώνεται η τιμή του semaphore 1 κατά 1 (0).

```
56        sem_wait(&semaphore1);
```

E6 : Δημιουργία φακέλου με όνομα Hearts.

```
58        printf("Creating a directory named Hearts.\n"); //E6
59        system("mkdir Hearts");
```

Παύση του συστήματος για 2 δευτερόλεπτα.

```
61        sleep(2);
```

Μετά, γίνεται αύξηση του semaphore 2 κατά 1 (0) και μείωση του semaphore 2 κατά 1, οπότε γίνεται αρνητική. Άρα, η εκτέλεση του προγράμματος μεταφέρεται στην διεργασία P.

```
62        sem_post(&semaphore2);
63        sem_wait(&semaphore1);
```

E7 : Εμφανίζει τα περιεχόμενα του τρέχοντος φακέλου σε μορφή δένδρου.

```
37        printf("The content of this directory in a tree-like format is :\n"); //E7
38        system("tree");
```

Παύση του συστήματος για 2 δευτερόλεπτα.

```
40        sleep(2);
```

Στην συνέχεια, αυξάνεται η τιμή του semaphore 1 κατά 1 (0). Με αυτή την εντολή, τελειώνει η εκτέλεση της διεργασίας P, οπότε το πρόγραμμα συνεχίζει με την διεργασία Q, από εκεί που είχε σταματήσει.

```
41        sem_post(&semaphore1);
```

E4 : Μετακινεί τον φάκελο CEID μέσα στον φάκελο Hearts. Έτσι, γίνεται CEID is in our Hearts (Ha, get it..?).

```
65        printf("Moving directory CEID into Hearts. \n"); //E4
66        system("mv CEID ./Hearts");
```

<u>Συνάρτηση Main :</u>

Πρώτα δηλώνω την συνάρτηση μου.

```
71 int main(int argc, char *argv[]) {
```

Στην συνέχεια, δημιουργώ το thread.

```
73          pthread_t t_id;
```

Ύστερα, αρχικοποιώ τους semaphores μου με τις κατάλληλες τιμές. Η τιμή 1 δείχνει ότι οι semaphores θα μοιραστούν μεταξύ διεργασιών και όχι threads και η τιμή 0 είναι η αρχική τιμή τους.

```
74          sem_init(&semaphore1,1,0);
75          sem_init(&semaphore2,1,0);
```

Μετά, βάζω τον thread μου να τρέχει στις αντίστοιχες διεργασίες. Στα ορίσματα έχω το όνομα του thread που θέλω να τρέχει, τις διεργασίες και NULL τιμές, έτσι ώστε τα threads να έχουν default values. Το sleep(10) ανάμεσα το έβαλα γιατί αλλιώς δεν γίνονταν σωστά οι παύσεις μεταξύ των εντολών στις διεργασίες.

```
77          pthread_create(&t_id, NULL, &P, NULL);
78          sleep(10);
79          pthread_create(&t_id, NULL, &Q, NULL);
```

Η επόμενη εντολή, περιμένει το thread να τερματίσει την λειτουργία του.

```
80          pthread_join(t_id,NULL);
```

Έπειτα, καταστρέφω τους semaphores μου, εφόσον έχω τελειώσει την δουλειά μου και δεν τους χρειάζομαι πλέον.

```
82          sem_destroy(&semaphore1);
83          sem_destroy(&semaphore2);
```

Τέλος, επιστρέφω 0, διότι η συνάρτηση μου δεν είναι void.

```
85          return 0;
86 }
```

Μέρος Β

Ερώτημα Α

Έχουμε αλγόριθμο Round Robin, ο οποίος αποδίδει περιοδικά 4 ms του χρόνου της ΚΜΕ σε κάθε μια από τις διεργασίες της ουράς εκτέλεσης, ξεκινώντας από εκείνη που είναι πρώτη στην ουρά και συνεχίζοντας προς το τέλος της.

Με βάση την εκφώνηση, όλες οι διεργασίες προς εκτέλεση έχουν διάρκεια 8 ms. Υπάρχει όμως η εξής διαφοροποίηση ως προς τα χαρακτηριστικά των δύο τύπων διεργασιών: Οι διεργασίες τύπου P έχουν απαίτηση μόνο για ΚΜΕ (τέτοιες διεργασίες καλούνται «περιορισμένες από την ΚΜΕ»),ενώ οι διεργασίες τύπου Q έχουν ελάχιστη απαίτηση για ΚΜΕ (25% της συνολικής τους διάρκειας) και κυρίως εκτελούν λειτουργίες Ε/Ε (τέτοιες διεργασίες καλούνται «περιορισμένες από Ε/Ε»).

Ο πίνακας υλοποιείται ως εξής :

Χρονική Στιγμή	Άφιξη	Εικόνα Μνήμης	Ουρά Μνήμης	ΚΜΕ	Ε/Ε	Ουρά ΚΜΕ	Ουρά Ε/Ε	Τέλος
0	P1	<Οπή 2048 K>	P1	-	-	-	-	-
1	Q1	<P1 300K> <Οπή 1748K>	Q1	P1	-	-	-	-
2	P2	<P1 300K> <Q1 1200K> <Οπή 548K>	P2	P1	-	Q1	-	-
3	Q2	<P1 300K> <Q1 1200K> <P2 300K> <Οπή 248K>	Q2	P1	-	Q1,P2	-	-
4	P3	<P1 300K> <Q1 1200K> <P2 300K> <Οπή 248K>	Q2,P3	P1	-	Q1,P2	-	-
5	-	<P1 300K> <Q1 1200K> <P2 300K> <Οπή 248K>	Q2,P3	Q1	-	P2,P1	-	-
6	-	<P1 300K> <Q1 1200K> <P2 300K> <Οπή 248K>	Q2,P3	P2	Q1	P1	-	-
7	-	<P1 300K> <Q1 1200K> <P2 300K> <Οπή 248K>	Q2,P3	P2	Q1	P1	-	-
8	-	<P1 300K> <Q1 1200K> <P2 300K> <Οπή 248K>	Q2,P3	P2	Q1	P1	-	-
9	-	<P1 300K> <Q1 1200K> <P2 300K> <Οπή 248K>	Q2,P3	P2	-	P1,Q1	-	-
10	-	<P1 300K> <Q1 1200K> <P2 300K> <Οπή 248K>	Q2,P3	P1	-	Q1,P2	-	-

11	-	<P1 300K> <Q1 1200K> <P2 300K> <Οπή 248K>	Q2,P3	P1	-	Q1,P2	-	-
12	-	<P1 300K> <Q1 1200K> <P2 300K> <Οπή 248K>	Q2,P3	P1	-	Q1,P2	-	-
13	-	<P1 300K> <Q1 1200K> <P2 300K> <Οπή 248K>	Q2,P3	P1	-	Q1,P2	-	-
14	-	<P1 300K> <Q1 1200K> <P2 300K> <Οπή 248K>	Q2,P3	P1	-	Q1,P2	-	P1
15	-	<Οπή 300K> <Q1 1200K> <P2 300K> <Οπή 248K>	Q2,P3	Q1	-	P2	-	-
16	-	<Οπή 300K> <Q1 1200K> <P2 300K> <Οπή 248K>	Q2,P3	P2	Q1	-	-	-
17	-	<Οπή 300K> <Q1 1200K> <P2 300K> <Οπή 248K>	Q2,P3	P2	Q1	-	-	-
18	-	<Οπή 300K> <Q1 1200K> <P2 300K> <Οπή 248K>	Q2,P3	P2	Q1	-	-	Q1
19	-	<Οπή 1500K> <P2 300K> <Οπή 248K>	Q2,P3	P2	-	-	-	P2
20	-	<Q2 500K> <P3 700K> <Οπή 848K>	-	Q2	-	P3	-	-
21	-	<Q2 500K> <P3 700K> <Οπή 848K>	-	P3	Q2	-	-	-
22	-	<Q2 500K> <P3 700K> <Οπή 848K>	-	P3	Q2	-	-	-
23	-	<Q2 500K> <P3 700K> <Οπή 848K>	-	P3	Q2	-	-	-
24	-	<Q2 500K> <P3 700K> <Οπή 848K>	-	P3	-	Q2	-	-

25	-	<Q2 500K> <P3 700K> <Οπή 848K>	-	Q2	-	P3	-	-
26	-	<Q2 500K> <P3 700K> <Οπή 848K>	-	P3	Q2	-	-	-
27	-	<Q2 500K> <P3 700K> <Οπή 848K>	-	P3	Q2	-	-	-
28	-	<Q2 500K> <P3 700K> <Οπή 848K>	-	P3	Q2	-	-	Q2
30	-	<Οπή 500K> <P3 700K> <Οπή 848K>	-	P3	-	-	-	P3
31	-	<Οπή 2048K>	-	-	-	-	-	-

Ερώτημα Β

(α) Η λογική διεύθυνση αποτελείται από τον αριθμό σελίδας και μία μετατόπιση (offset) εντός της σελίδας.

Γνωρίζω από τον πίνακα σελίδων ότι ο αριθμός σελίδας είναι 2 δεκαεξαδικά ψηφία επομένως 8 bits. Επίσης γνωρίζω ότι το μέγεθος της σελίδας είναι ίσο με 2^{10} bytes άρα η μετατόπιση στην λογική διεύθυνση ισούται με 10 bits.

Άρα για την αναπαράσταση κάθε λογικής διεύθυνσης του συστήματος απαιτούνται 18 bits.

Ο αριθμός πλαισίου είναι 3 δεκαεξαδικά ψηφία επομένως 12 bits. Γνωρίζω ότι το μέγεθος της σελίδας ισούται με το μέγεθος του πλαισίου άρα η μετατόπιση στη φυσική διεύθυνση ισούται με 10 bits.

Άρα για την αναπαράσταση κάθε φυσικής διεύθυνσης του συστήματος απαιτούνται 22 bits.

(β) Μετατρέπω τη λογική διεύθυνση από το δεκαεξαδικό στο δυαδικό.

$0A0A_{(16)}=\underline{00000010}1000001010_{(2)}$

Τα πρώτα 8 bits αποτελούν τον αριθμό σελίδας ενώ τα υπόλοιπα 10 τη μετατόπιση.

$00000010_{(2)}= 02_{(16)}$

Ο αριθμός πλαισίου που αντιστοιχεί στον αριθμό σελίδας 02 είναι ο 20C.

$20C_{16}=001000001100_{(2)}$

Άρα η φυσική διεύθυνση είναι:

Αριθμός πλαισίου

$001000001100\,1000001010_{(2)}=8320A_{(16)}$

Μετατόπιση

Ερώτημα Γ

Γνωρίζουμε πως διαθέτουμε 4 πλαίσια σελίδων τα οποία είναι αρχικά κενά και πως η ακολουθία διεργασιών είναι :

2 5 8 1 8 7 5 1 8 2 4 2 1 3 6 4 7 5 3 7

Εφαρμόζοντας LRU (Least Recent Used) έχουμε την εξής ακολουθία αναφοράς:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	2	2	2	2	2	7	7	7	7	2	2	2	2	2	2	4	4	4	4	4
1		5	5	5	5	5	5	5	5	5	4	4	4	4	6	6	6	6	3	3
2			8	8	8	8	8	8	8	8	8	8	8	3	3	3	3	5	5	5
3				1	1	1	1	1	1	1	1	1	1	1	1	1	7	7	7	7
	F	F	F	F		F				F	F			F	F	F	F	F	F	