Λειτουργικά Συστήματα

2019 - 2020

2η Εργαστηριακή Άσκηση

Άγκο Μπεσιάνα
ΑΜ: 1059658

Ζεκυριά Αθανασία
ΑΜ: 1059660

Ριτσίκαλης Γεώργιος
ΑΜ: 1057770

Περιεχόμενα

Μέρος 1

Ερώτημα Α: Εξηγήστε προσεκτικά τι κάνει το παρακάτω πρόγραμμα.

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
        int pid1;
        int pid2;
        pid1 = fork();
        if (pid1 < 0)
          printf("Could not create any child\n");
        else
        {
          pid2 = fork();
          if (pid2 < 0)
            printf("Could not create any child\n");
          else
           if ( (pid1 < 0) && (pid2 < 0) kill(pid1,9);
        }
        sleep(20);

        return (0);
}
```

Απάντηση: Το πρόγραμμα δημιουργεί 2 μεταβλητές pid1 και pid2, και στη συνέχεια, καλεί την συνάρτηση fork αρχικά για το pid1 ελέγχοντας αν είναι μικρότερο του 0, σε περίπτωση που είναι, εκτυπώνει «Could not create any child», αλλιώς αφού έχει δημιουργήσει γονέα καλεί την fork για το pid2 κάνοντας τον ίδιο έλεγχο και εκτυπώνει το ίδιο. Αλλιώς ελέγχει εάν και οι 2 μεταβλητές είναι, ταυτόχρονα, μικρότερες του 0, αν είναι και οι 2 τότε «σκοτώνει» την pid1. Όταν τελειώσει η αρχική συνθήκη κάνει sleep 20 δευτερόλεπτα και μετά τερματίζει.

i.Πόσες διεργασίες υπάρχουν σε κατάσταση sleeping 10 δευτερόλεπτα μετά την έναρξή του;

Απάντηση: 6

ii. Τροποποιήστε κατάλληλα το κώδικα έτσι ώστε κάθε διεργασία να τυπώνει το id της και το id του γονέα της πριν βρεθεί σε κατάσταση sleeping.

Απάντηση:

```
#include <stdio.h>

#include <stdlib.h>

int main()
```

```
{
int pid1;
int pid2;
pid1 = fork();
if (pid1 < 0)
printf("Could not create any child\n");
else
{

printd("[son]pid %d from [parent] pid %d\n", getpid(), getppid());

pid2 = fork();

if(pid2 < 0)
printf("Could not create any child\n");
else
printd("[son]pid %d from [parent] pid %d\n", getpid(), getppid());

if ( (pid1 < 0) && (pid2 < 0) )
kill (pid1,9);
 }
sleep(20);
return(0);
```

Ερώτημα Β: Δημιουργήστε ένα πρόγραμμα στο οποίο μία διεργασία στο Linux/Unix παράγει άλλες 100 θυγατρικές της. Κάθε μία διαβάζει μία rand( ) ακέραια τιμή που δείχνει την προτεραιότητά της και τη γράφει ατομικά σε μία heap shared δομή. Συγχρονίστε τις παραπάνω διεργασίες, ώστε να εξασφαλίσετε αμοιβαίο αποκλεισμό στην προσπέλαση της heap δομής με απότερο στόχο την αποθήκευση της χαμηλότερης προτεραιότητας πάντα στη ρίζα της min heap δομής.

Απάντηση:

```
#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <pthread.h>

#include <semaphore.h>

#include <time.h>

#include <string.h>



FILE *file;

sem_t writef;

sem_t readf;


void* FileReadOrWrite(void *ptr)

{

  int rand;

    rand=randoms();

    char c[20];

    char buf[50];



  if(rand>25)
```

```
{

printf("User %d entered the queue for file writing \n",rand);
sem_wait(&writef);
sem_wait(&readf);
printf(" User %d entering the critical region\n",rand);

file= fopen("fileforproject.txt","w");
sleep(5);
fprintf(file, "File modified by User %d \n",rand);
fclose(file);
sem_post(&readf);
sem_post(&writef);

printf("User %d is out of the critical region \n",rand );
}
else
{

 //sem_wait(&writef);


 printf("User %d is reading now the file\n",rand );

 file=fopen("fileforproject.txt","r");
 //sem_wait(&readf);
   sleep(3);
```

```
    // fclose(file);
    //sem_post(&readf);
    printf("User  %d is not reading the file  \n",rand)
    //sem_post(&writef);
    //sem_post(&readf);
  }

  return NULL;
}

int randoms()
{
  int random= (rand() %(50 - 1)) + 1;

  return random;
}

 int main()
 {
  sem_init(&writef, 0, 1);
  sem_init(&readf,0,1);
  pthread_t fileWrite[9];
  pthread_t fileRead;
  int rand ;
  srand(time(0));
```

```
//for(int i =0 ; i<5 ; i++)
   // {

        pthread_create(&fileWrite[0], NULL, FileReadOrWrite,NULL);
        sleep(1);
        pthread_create(&fileWrite[1], NULL, FileReadOrWrite,NULL);
        sleep(1);
        pthread_create(&fileWrite[2],NULL,FileReadOrWrite,NULL);
        sleep(1);
        pthread_create(&fileWrite[3], NULL, FileReadOrWrite, NULL);
        sleep(1);
        pthread_create(&fileWrite[4], NULL, FileReadOrWrite, NULL);
        sleep(1);



        pthread_join(fileWrite[0],NULL);
        pthread_join(fileWrite[1],NULL);
        pthread_join(fileWrite[2],NULL);
        pthread_join(fileWrite[3],NULL);
        pthread_join(fileWrite[4],NULL);

   //}

sem_destroy(&readf);
sem_destroy(&writef);
exit(0);
```

Ερώτημα Γ: Γράψτε πρόγραμμα που να υλοποιεί με χρήση σημαφόρων το πρόβλημα συγχρονισμού των διαδικασιών ΑΝΑΓΝΩΣΗΣ (READ) και ΕΓΓΡΑΦΗΣ (WRITE) σε ένα κοινό διαμοιραζόμενο αρχείο (SHARED FILE). Θεωρείστε ότι πρέπει να επιτρέπουμε πολλές αναγνώσεις ταυτόχρονα, αλλά μόνο μία εγγραφή.

Απάντηση:

```
#include <stdio.h>

#include<stdlib.h>

#include <time.h>

#include<limits.h>

#include <sys/types.h>

#include <unistd.h>

#include <semaphore.h>

#include <signal.h>


int heap[1000000], heapSize;

time_t t;

sem_t semaphore;

int pid ;

int randNum;

int heapSave[8];

int heapElements=100;


void EnterHeap(int count );

void Init();

void Insert(int element);

int DeleteMin();

void PrintHeap();


int main()
```

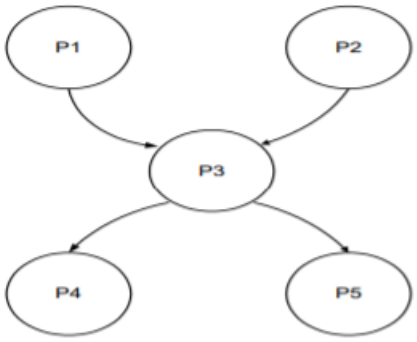
```
{
   sem_init(&semaphore, 0, 1);

     printf("Before heap insertion\n");
     for(int i=0 ; i<heapElements ; i++)
   {
     pid=fork();

   if(pid==0)
     {
       sem_wait(&semaphore);
       EnterHeap(i);
       sem_post(&semaphore);

     }
     else
     {
       wait(NULL);
     }
   }
   printf("\nAfter heap insertion\n");
   PrintHeap();
   printf("\n");

       kill(pid,SIGKILL);
   return 0 ;
}
```

```
void EnterHeap(int count )
{

    sleep(1);
    srand((unsigned) time(&t));
    randNum=rand()%1000;
    printf("%d\n",randNum);
    Insert(randNum);
 }
 void PrintHeap()
 {

      for(int i=0 ; i<heapElements;i++)
      {
        printf("%d\n",DeleteMin());
      }
   }

void Init() {

   heapSize = 0;

   heap[0] = -INT_MAX;

}
void Insert(int element) {
```

```
    heapSize++;

    heap[heapSize] = element;

    int now = heapSize;

    while (heap[now / 2] > element) {

        heap[now] = heap[now / 2];

        now /= 2;

    }
    heap[now] = element
}
int DeleteMin() {
    int minElement, lastElement, child, now;

    minElement = heap[1];

    lastElement = heap[heapSize--];

    for (now = 1; now * 2 <= heapSize; now = child) {

        child = now * 2;
```

```
        if (child != heapSize && heap[child + 1] < heap[child]) {


            child++;


        }
        if (lastElement > heap[child]) {


            heap[now] = heap[child];


        } else
        {
            break;
        }
    }
    heap[now] = lastElement;
    return minElement;
}
```

Ερώτημα Δ: : (1) Να γράψετε πρόγραμμα συγχρονισμού για τον παρακάτω γράφο προτεραιότητας κάνοντας χρήση σημαφόρων. Θεωρείστε ότι κάθε διεργασία εκτελεί μία εντολή του συστήματος της αρεσκείας σας . π.χ. system("ls -l")ή system("ps -l") κ.τ.λ... Επίσης θεωρείστε ότι οι P1, P2, P3, P4 και P5 είναι θυγατρικές μιας μόνο διεργασίας.



(2) Θεωρήστε ότι πρέπει να συγχρονίσετε την εκτέλεση των διαδικασιών Δ1, Δ2, Δ3, Δ4, Δ5 και Δ6 σύμφωνα με τους παρακάτω περιορισμούς:

- Η Δ1 εκτελείται πριν από τις Δ2 και Δ3.
- Η Δ2 εκτελείται πριν από τις Δ4 και Δ5.
- Η Δ3 εκτελείται πριν από την Δ5.
- Η Δ6 εκτελείται μετά από τις Δ3 και Δ4.

Η έκφραση «η διαδικασία Δi εκτελείται πριν από τη διαδικασία Δj» σημαίνει ότι η εκτέλεση της Δi πρέπει να ολοκληρωθεί πριν αρχίσει η εκτέλεση της Δj. Αναλόγως η έκφραση «η διαδικασία Δi εκτελείται μετά από τη διαδικασία Δj» σημαίνει ότι η εκτέλεση της Δi μπορεί να αρχίσει αφού ολοκληρωθεί η εκτέλεση της Δj.

Να γράψετε πρόγραμμα συγχρονισμού που θα ικανοποιεί τους προηγούμενους περιορισμούς κάνοντας χρήση σημαφόρων. Θεωρείστε ότι κάθε διεργασία εκτελεί μία εντολή του συστήματος της αρεσκείας σας . π.χ. system("ls -l")ή system("ps -l") κ.τ.λ... Επίσης θεωρείστε ότι οι Δ1, Δ2, Δ3, Δ4 και Δ5 είναι θυγατρικές μιας μόνο διεργασίας.

Απάντηση:

(1)

P1	P2	P3	P4	P5
up(s1)	up(s2)	wait(s1)	wait(s3)	wait(s4)
		wait(s2)		
		up(s3)		
		up(s4)		

Var

S1,s2,s3,s4: semaphores;

s1=s2=s3=s4=0;

Cobegin

Begin P1; up(s1); end;

Begin P2; up(s2); end;

Begin wait(s1); wait(s2); S3; up(s3); up(s4); end;

Begin wait(s3); P4; end;

Begin wait(s4); P5; end;

(2):

P1->P2->P3->P4->P5

S1=Σ12

S2=Σ13

S3=Σ24

S4=Σ25

S5=d35

S6=Σ36

S7=d46

Var

S1,s2,s3,s4,s5,s6,s7: semaphores;

s1:=s2:=s3:=s4:=s5:=s6:=s7:=0;

Cobegin

Begin p1; up (s1); up(s2); end;

Begin down(s1); P2; up (s3); up(s4); end;

Begin down(s2); P3; up (s5); up(s6); end;

Begin down(s3); P4; up(s7); end;

Begin down(s4); down(s5); P5; end;

Begin down(s7); down(s6); P6; end;

Coend

ΜΕΡΟΣ 2

Ερώτημα Α: Σε ένα σύστημα με σελιδοποίηση (paging) θεωρήστε ότι υποστηρίζονται λογικές διευθύνσεις των 32 bits όπου τα πρώτα (πιο σημαντικά) 18 bits αναπαριστούν τον αριθμό σελίδας κάθε διεύθυνσης.

(α) Έστω μία διεργασία η οποία αποτελείται από 39500(16δικο) bytes. Αν θεωρήσουμε ότι ή εν λόγω διεργασία είναι ολόκληρη φορτωμένη στη μνήμη, πόσα πλαίσια σελίδων καταλαμβάνει και πόση εσωτερική κλασματοποίηση προκαλεί;

Απάντηση:

Εφόσον τα 18 bits από τα 32 που αναπαριστούν τον αριθμό της σελίδας κάθε διεύθυνσης, τα υπόλοιπα 14 αποτελούν το offset.

Συνεπώς, το μέγεθος πλαισίου είναι 2^14 bytes = 16.384.

Η διαδικασία είναι 39.500 bytes στο 16δικο σύστημα, επομένως στο 10δικό είναι 234.752 bytes.

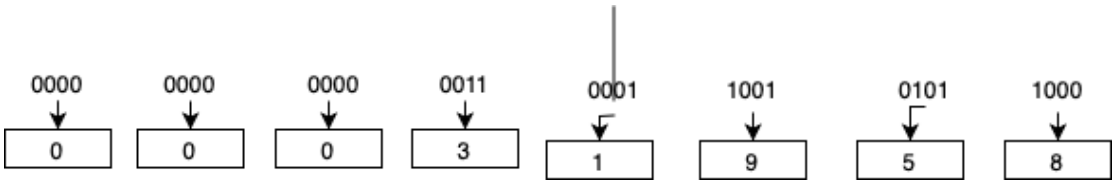
Θα χρειαστώ 15 σελίδες (14+1) για να καλύψω ολόκληρη την διεργασία (15x16.384=245.760 > 234.752 bytes).

Η εσωτερική κλασματοποίηση περιέχει τη μνήμη που δεν χρησιμοποιείται, δηλαδή 245.760-234.752=11.008 bytes.

(β) Υποθέστε στη συνέχεια ότι η εν λόγω διεργασία κατά το τρέχον χρονικό διάστημα έχει φορτωμένες στη μνήμη μόνο τις τελευταίες πέντε σελίδες της, στα ακόλουθα κατά σειρά πλαίσια (οι αριθμοί δίνονται στο δεκαδικό σύστημα): 16, 225, 170, 35, 51 (δηλαδή η τελευταία σελίδα της είναι φορτωμένη στο πλαίσιο 51, η προτελευταία στο πλαίσιο 35, η προ-προτελευταία στο πλαίσιο 170 κοκ). Με βάση τα παραπάνω δεδομένα, υπολογίστε σε ποιες φυσικές διευθύνσεις αντιστοιχούν οι ακόλουθες λογικές διευθύνσεις της διεργασίας (δίνονται στο δεκαεξαδικό σύστημα / στο δεκαεξαδικό σύστημα θα πρέπει να δώσετε και τις απαντήσεις σας):

(i) 0003195816

Απάντηση:

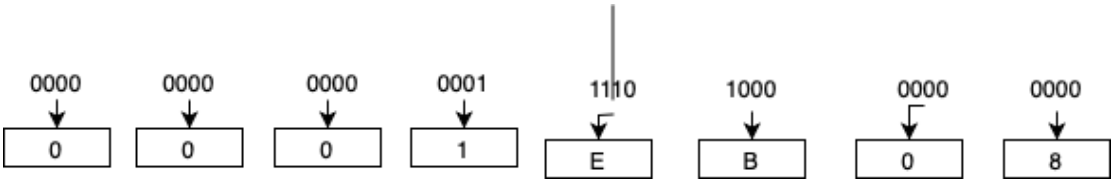


Παράλληλα ισχύει ότι:

- Λογική σελίδα (18bits) = 1100(2) = 12(10)
- Έχουμε 15 σελίδες, η 12 είναι η τέταρτη από το τέλος, επομένως σύμφωνα με την εκφώνηση θα είναι στο πλαίσιο 225(10) = E1(16)
- Η μετατόπιση θα είναι 1958(16) = 6488(10)
- Η φυσική διεύθυνση θα είναι E1(16) – 1958(16) = 385958(16)

(ii) 0001E80016

Απάντηση:



Παράλληλα ισχύει ότι:

- Λογική σελίδα (18bits) = 0110(2) = 7(10). Ωστόσο, αυτή δεν είναι φορτωμένη στη μνήμη επομένως υπάρχει λάθος σελίδας.

Ερώτημα Β: Σε ένα σύστημα τμηματοποιημένης μνήμης (segmentation) θεωρήστε το παρακάτω μέρος του Πίνακα Τμημάτων μίας διεργασίας (όλοι οι αριθμοί του πίνακα δίνονται στο δεκαδικό σύστημα):

Αριθμός Τμήματος	Διεύθυνση Βάσης	Μήκος Τμήματος
0	1650	1110
1	3200	2350
2	10310	1290
.....	.....	.....
10	5950	2255
11	9050	1230
12	12270	5535
.....	.....	.....

Θεωρείστε επίσης ότι η κάθε λογική διεύθυνση αποτελείται από 32 bits και ότι το μέγιστο υποστηριζόμενο μέγεθος ενός τμήματος είναι 16 MBytes.

(α) Ποιος είναι ο μέγιστος υποστηριζόμενος αριθμός τμημάτων για μία διεργασία;

Απάντηση:

Σύμφωνα με την εκφώνηση το μέγιστο υποστηριζόμενο μέγεθος τμήματος είναι 16Mbytes δηλαδή ίσο με 2^24 bytes. Εφόσον, η μνήμη είναι οργανωμένη σε 1 byte/θέση μνήμης, το offset είναι 24 bits και ο αριθμός τμήματος 8 (32=24+8).

Επομένως, ο μέγιστος υποστηριζόμενος αριθμός τμημάτων είναι 2^8=256.

(β) Υπολογίστε σε ποιες φυσικές διευθύνσεις αντιστοιχούν οι λογικές διευθύνσεις που ακολουθούν (δίνονται στο δεκαεξαδικό σύστημα / στο δεκαεξαδικό σύστημα θα πρέπει να δώσετε και τις απαντήσεις σας):

(i) 0B00042A16

Απάντηση:

8 bits = 2 hex ψηφία, επομένως ο αριθμός s = 0B(16) = 11(10).

Το d είναι το υπόλοιπο 00042A(16) = 1066(10).

Έχουμε ότι: διεύθυνση βάσης + μετατόπιση = 9050 + 1066 = 10116(10) = 2784(16).

(ii) 02000B6D16

Απάντηση:

bits = 2 hex ψηφία, επομένως ο αριθμός s = 02(16) = 2(10), βάσει του πίνακα ανήκει στο τμήμα 2.

Το d είναι το υπόλοιπο 000B6D(16) = 2925 (10).

Ωστόσο το offset είναι μεγαλύτερο από το μήκος τμήματος (2925>1290), άρα η διεύθυνση είναι εκτός ορίων.

Ερώτημα Γ: Έστω ότι στο παραπάνω σύστημα, με στόχο την αποδοτικότερη διαχείριση του συνολικού χώρου μνήμης, αποφασίστηκε για την εκχώρηση μνήμης στα τμήματα κάθε διεργασίας (λόγω του μεγάλου εν δυνάμει μεγέθους τους) να εφαρμοστεί η μέθοδος της σελιδοποίησης με μέγεθος σελίδας 512 bytes.

(α) Υπολογίστε από πόσα bits αποτελείται κάθε ένα από τα τρία μέρη στα οποία χωρίζεται πλέον κάθε λογική διεύθυνση (αριθμός τμήματος, αριθμός σελίδας και μετατόπιση) στο νέο σχήμα μνήμης.

Απάντηση:

Γνωρίζω από την εκφώνηση ότι το μέγεθος σελίδας είναι 512 bytes με οργάνωση 1 byte/θέση μνήμης (512 = 2^9 διευθύνσεις σε κάθε σελίδα). Οπότε χρειαζόμαστε 9 offset bits.

Σύμφωνα με το προηγούμενο ερώτημα, υπάρχουν 8bits που καθορίζουν τον αριθμό τμήματος. Επομένως, απομένουν 15 bits για τον αριθμό της σελίδας (32-8-9 = 15).

(β) Υπολογίστε από πόσες σελίδες μπορεί να αποτελείται κατά μέγιστο μία διεργασία στο νέο σχήμα μνήμης.

Απάντηση:

8 bits αριθμός τμήματος -> 2^8 το πολύ τμήματα

15 bits αριθμός σελίδας -> 2^15 σελίδες/τμήμα

Ο μέγιστος αριθμός σελίδων μιας διεργασίας στο νέο σχήμα μνήμης είναι 2^8 x 2^15 = 2^23.

(γ) Έστω μία διεργασία η οποία αποτελείται από δύο τμήματα (Τμήμα 0 και Τμήμα 1), των οποίων οι Πίνακες Σελίδων (Π.Σ.) δίνονται παρακάτω (με '-' υπονοείται ότι η εν λόγω σελίδα δεν είναι φορτωμένη στη μνήμη, ενώ με '?' υπονοείται ότι είναι μεν φορτωμένη στη μνήμη αλλά δεν ξέρουμε σε ποιο πλαίσιο έχει φορτωθεί).

(i) Μετατρέψτε τη λογική διεύθυνση 010004CF16 της διεργασίας αυτής στην αντίστοιχη φυσική διεύθυνση (δίνεται στο δεκαεξαδικό σύστημα / στο ίδιο σύστημα θα πρέπει να δώσετε και την απάντησή σας).

Απάντηση:

010004CF(16) = 0000 0001 0000 0000 0000 0100 1100 1111(2)

s=0000 0001 (1 τμήμα)

p=0000 0000 0000 010 ($2^{η}$ σελίδα)

d=0 1100 1111 (offset)

Στο τμήμα 1, στην $2^{η}$ σελίδα αντιστοιχίζεται ο 0B0B = 0000 1011 0000 1011

Φυσική διεύθυνση: 0000 1011 0000 1011 0 1100 1111 = 1616F(16)

(ii) Συμπληρώστε τα ερωτηματικά '?' των δύο Π.Σ. που σας δίνονται, θεωρώντας ως δεδομένα ότι:

- η λογική διεύθυνση 010009FF16 προκαλεί σφάλμα σελίδας (page fault), και
- η λογική διεύθυνση 000003F016 αντιστοιχεί στη φυσική διεύθυνση E0E1F016

Πίν. Σελίδων Τμήματος 0	
Αρ. Σελ.	Αρ. Πλαισίου
0	151F
1	?
2	34FE
3	7E11
4	2345
5	-
...	...

Πίν. Σελίδων Τμήματος 1	
Αρ. Σελ.	Αρ. Πλαισίου
0	-
1	2EE1
2	0B0B
3	0C11
4	?
5	1BA2
...	...

Απάντηση:

❖

010009F(16) = 0000 0001 0000 0000 0000 1001 1111 1111(2)

s=0000 0001(1 τμήμα)

p=0000 0000 0000 100($4^{η}$ σελίδα)

d=1 1111 1111(offset)

Στον πίνακα σελίδων τμήματος 1, στην $4^{η}$ σελίδα έχω σφάλμα σελίδας (άρα -).

❖

000003F0(16) = 0000 0000 0000 0000 0000 0011 1111 0000

s=0000 0000(0 τμήμα)

p=0000 0000 0000 001($1^{η}$ σελίδα)

d=1 1111 0000(offset)

Η φυσική διεύθυνση: E0E1F0=1110 0000 1110 0001 1111 0000 και

d= 1 1111 0000

άρα στον πίνακα σελίδων τμήματος 0, στην $1^{η}$ σελίδα: E0E1F0 – d = 1110 0000 1110 000 = 7070(16)

Ερώτημα Δ: Έστω η παρακάτω ακολουθία αναφοράς μίας διεργασίας:

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

Η διεργασία εκτελείται σε σύστημα που η μνήμη του διαθέτει τέσσερα (4) πλαίσια σελίδων, τα οποία αρχικά είναι κενά. Στον πίνακα που ακολουθεί δώστε την ακολουθία αναφοράς, σημειώνοντας με μαύρο χρώμα ανά χρονική στιγμή τις σελίδες που υπάρχουν στον πίνακα σελίδων και σημειώνοντας με κόκκινο χρώμα μόνο τους αριθμούς σελίδων στα σημεία στα οποία συμβαίνουν σφάλματα σελίδας για την πολιτική αντικατάστασης σελίδων LRU (Least Recently Used).

Απάντηση:

LRU: Αλγόριθμος αντικατάστασης που αντικαθιστά κάθε φορά το μπλοκ που έχει το μεγαλύτερο χρόνο παραμονής στη μνήμη, χωρίς να γίνει αίτηση του επεξεργαστή σε αυτό, για ανάγνωση ή εγγραφή. Σφάλματα προκύπτουν όταν τα πλαίσια είναι άδεια ή όταν δεν έχουν την κατάλληλη πληροφορία που ψάχνουμε.

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

Συνολικά, έχουμε 13 λάθη.