Λειτουργικά Συστήματα
2019 - 2020
2η Εργαστηριακή Άσκηση

Αλέξανδρος Νατσολλάρη **Α.Μ.: 1057769**
Αργύρης Μάλλιος **Α.Μ.: 1051937**
Παναγιώτης Μπαρμπούνης **Α.Μ.: 1054382**

Μέρος 1^{ο}

Ερώτημα Α:

i) Στο πρόγραμμα του ερωτήματος 10 δευτερόλεπτα μετά την έναρξη του βρίσκονται 5(δηλαδή όλες) διεργασίες σε sleep.

ii)

```
child pid= 26024 , parent pid= 26023
child pid= 26025 , parent pid= 26024
child pid= 26027 , parent pid= 26025
child pid= 26026 , parent pid= 26024

Process returned 0 (0x0)   execution time : 20.009 s
Press ENTER to continue.
```

```
#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);
            }
      }
         printf( "child pid is: %d parent pid is: %d\n", getpid(), getppid() );
         sleep(20);
         return (0);
```

Ερώτημα Β:

```
#include<stdio.h>

void create(int []);
void down_adjust(int [],int);
int heap2[30];

int main()
{
	int heap[30],n,i,last,temp;
	int temp2,last2;

	printf("Enter no. of elements:");
	scanf("%d",&n);

	printf("\nEnter elements:");
	for(i=1;i<=n;i++){
		scanf("%d",&heap[i]);
		printf("deuteros pinakas");
		scanf("%d",&heap2[i]);
		}
	//create a heap
	heap[0]=n;
	heap2[0]=n;
	create(heap);
	//sorting
	while(heap[0] > 1)
	{
		//swap heap[1] and heap[last]
		last=heap[0];
		last2=heap2[0];

		temp=heap[1];
		temp2=heap2[1];

		heap[1]=heap[last];
		heap2[1]=heap2[last2];

		heap[last]=temp;
		heap2[last2]=temp2;

		heap[0]--;
		heap2[0]--;
		down_adjust(heap,1);


	}
	//print sorted data
	printf("\nArray after sorting:\n");
	for(i=1;i<=n;i++)
		printf("%d	%d \n",heap[i],heap2[i]);
```

```
        return 0;
}
void create(int heap[])
{
        int i,n;
        n=heap[0]; //no. of elements
        for(i=n/2;i>=1;i--)
                down_adjust(heap,i);
}
void down_adjust(int heap[],int i)
{
        int j,temp,n,flag=1;
        int temp2;
        n=heap[0];
        while(2*i<=n && flag==1)
        {
                j=2*i;    //j points to left child
                if(j+1<=n && heap[j+1] > heap[j])
                        j=j+1;
                if(heap[i] > heap[j])
                        flag=0;
                else
                {
                        temp=heap[i];
                        temp2=heap2[i];

                        heap[i]=heap[j];
                        heap2[i]=heap2[j];

                        heap[j]=temp;
                        heap2[j]=temp2;
                        i=j;
                }
        }
}
```

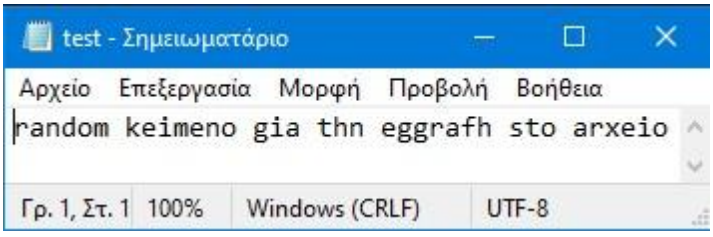
Ερώτημα Γ:

Το πρόγραμμα για να τρέξει σωστά χρειάζεται ένα αρχείο test.txt στον φάκελο που περιέχει τον κώδικα main.c.

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <semaphore.h>
#include <unistd.h>
#define MAX_NUM 50
sem_t sem1, sem2;
FILE *fp;
void *f1(void* p)
{
 int i;
 sem_wait(&sem1);
 for(i=1; i<=MAX_NUM; i++)
 {
  int a[51];
      a[i]=i;
      fp = fopen("test.txt", "r+");
      printf("To arxeio diavastike:%d fores\n",a[i]);
 }
  sem_post(&sem2);
}
void *f2(void* p)
{
 sem_wait(&sem2);
      fprintf(fp,"random keimeno gia thn eggrafh sto arxeio\n");
      fclose(fp);
   sem_post(&sem1);
}
int main()
{
  pthread_t p1, p2;
  sem_init(&sem1, 0, 1);
  sem_init(&sem2, 0, 0);
 pthread_create(&p1, NULL, f1,  (void*)NULL);
 pthread_create(&p2, NULL, f2,  (void*)NULL);
 pthread_join(p1, NULL);
 pthread_join(p2, NULL);
 return 0;
}
```

```
To arxeio diavastike:1 fores
To arxeio diavastike:2 fores
To arxeio diavastike:3 fores
To arxeio diavastike:4 fores
To arxeio diavastike:5 fores
To arxeio diavastike:6 fores
To arxeio diavastike:7 fores
To arxeio diavastike:8 fores
To arxeio diavastike:9 fores
To arxeio diavastike:10 fores
To arxeio diavastike:11 fores
To arxeio diavastike:12 fores
To arxeio diavastike:13 fores
To arxeio diavastike:14 fores
To arxeio diavastike:15 fores
To arxeio diavastike:16 fores
To arxeio diavastike:17 fores
To arxeio diavastike:18 fores
To arxeio diavastike:19 fores
To arxeio diavastike:20 fores
To arxeio diavastike:21 fores
To arxeio diavastike:22 fores
To arxeio diavastike:23 fores
To arxeio diavastike:24 fores
To arxeio diavastike:25 fores
To arxeio diavastike:26 fores
To arxeio diavastike:27 fores
To arxeio diavastike:28 fores
To arxeio diavastike:29 fores
To arxeio diavastike:30 fores
To arxeio diavastike:31 fores
To arxeio diavastike:32 fores
To arxeio diavastike:33 fores
To arxeio diavastike:34 fores
To arxeio diavastike:35 fores
To arxeio diavastike:36 fores
To arxeio diavastike:37 fores
To arxeio diavastike:38 fores
To arxeio diavastike:39 fores
To arxeio diavastike:40 fores
To arxeio diavastike:41 fores
To arxeio diavastike:42 fores
To arxeio diavastike:43 fores
To arxeio diavastike:44 fores
To arxeio diavastike:45 fores
To arxeio diavastike:46 fores
To arxeio diavastike:47 fores
To arxeio diavastike:48 fores
To arxeio diavastike:49 fores
To arxeio diavastike:50 fores

Process returned 0 (0x0)   exe
Press any key to continue.
```



Ερώτημα Δ:

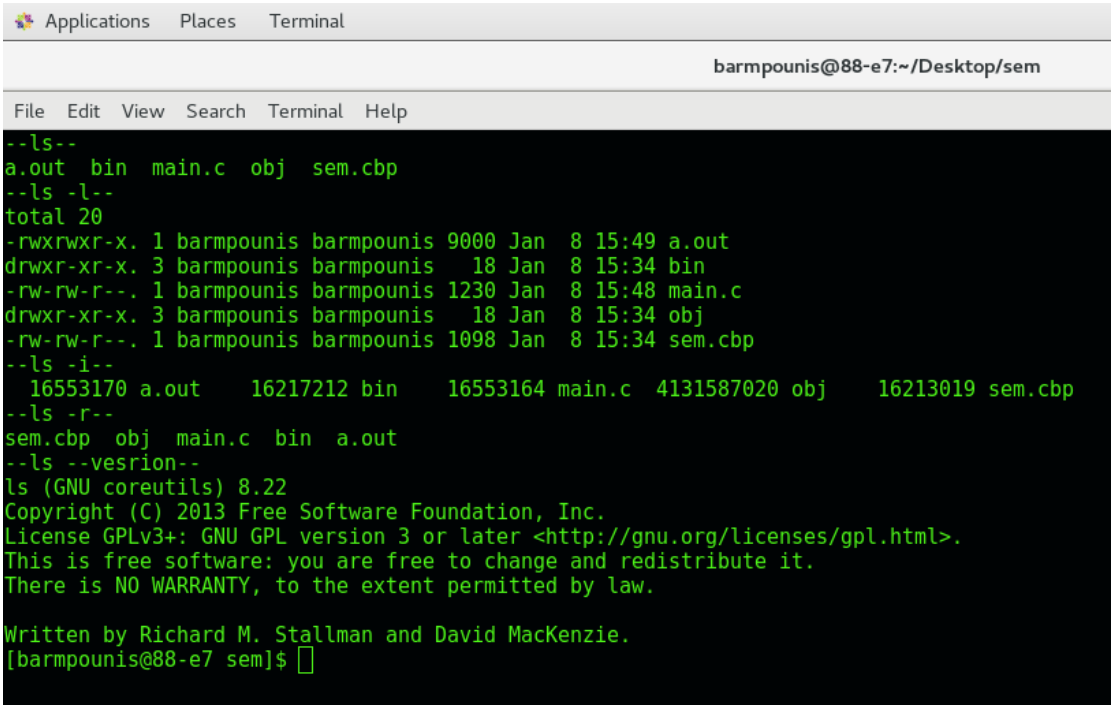
1)

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/shm.h>
#include <errno.h>
#include <semaphore.h>
#include <fcntl.h>

typedef sem_t Semaphore;
Semaphore *synch1;
Semaphore *synch2;
int main (int argc, char **argv)
{
   int i;
   pid_t pid;
          system("clear");
   synch1 = sem_open ("Sem1", O_CREAT | O_EXCL, 0644, 0);
   synch2 = sem_open ("Sem2", O_CREAT | O_EXCL, 0644, 0);

   pid = fork ();
if (pid > 0)
          {
             printf("--ls--\n");
             system("ls");            //P1
      printf("--ls -l--\n");
             system("ls -l");         //P2
      sem_post (synch1);
      sem_unlink ("Sem1");
      sem_close(synch1);
      sem_unlink ("Sem2");
      sem_close(synch2);
   }
else if (pid == 0)
          {
          sem_wait (synch1);
          printf("--ls -i--\n");
      system("ls -i");                //P3
          sem_post (synch2);
          sem_post (synch1);
          sem_wait (synch1);
          sem_wait (synch2);
          printf("--ls -r--\n");
          system("ls -r");                    //P4
```

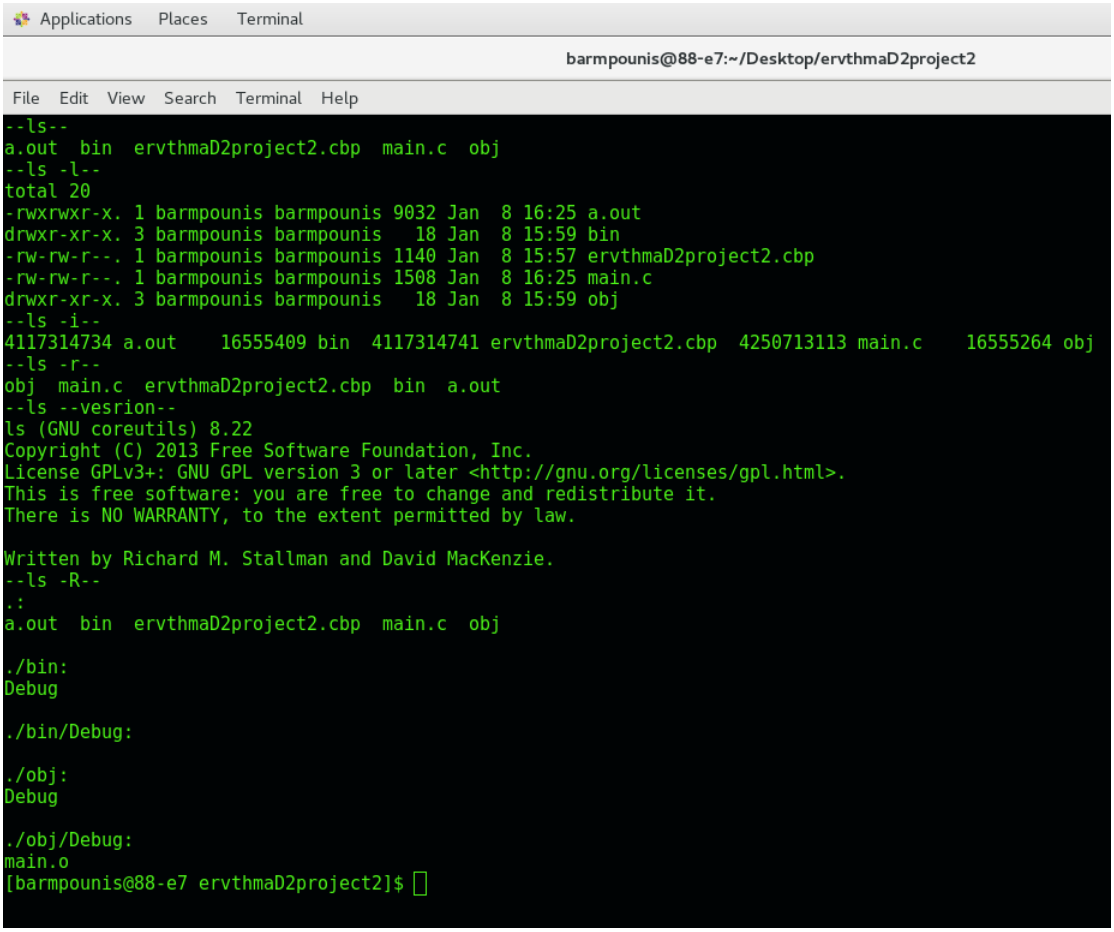
```
            printf("--ls --vesrion--\n");
            system("ls --version");                 //P5
        wait(NULL);
    }
else
            {
        sem_unlink ("Sem1");
        sem_close(synch1);
        sem_unlink ("Sem2");
        sem_close(synch2);
        printf ("Fork error.\n");
    }
    sleep(3);
    return (0);
}
```



2)

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/shm.h>
#include <errno.h>
#include <semaphore.h>
#include <fcntl.h>
typedef sem_t Semaphore;
Semaphore *synch1;
Semaphore *synch2;
Semaphore *synch3;
int main (int argc, char **argv)
{
   int i;
   pid_t pid;
   system("clear");
   synch1 = sem_open ("Sem1", O_CREAT | O_EXCL, 0644, 0);
   synch2 = sem_open ("Sem2", O_CREAT | O_EXCL, 0644, 0);
   synch3 = sem_open ("Sem3", O_CREAT | O_EXCL, 0644, 0);
   pid = fork ();
if (pid > 0)
          {
             printf("--ls--\n");
      system("ls");                    //D1
      sem_post (synch1);
      sem_unlink ("Sem1");
      sem_close(synch1);
      sem_unlink ("Sem2");
      sem_close(synch2);
      sem_unlink ("Sem3");
      sem_close(synch3);
   }
else if (pid == 0)
          {
            sem_wait (synch1);
            printf("--ls -l--\n");
              system("ls -l");          //D2
             printf("--ls -i--\n");
             system("ls -i");           //D3
           sem_post (synch2);
           sem_post (synch1);
           sem_wait (synch1);
           sem_wait (synch2);
           printf("--ls -r--\n");
```

```
            system("ls -r");              //D4
            sem_post (synch3);
            printf("--ls --vesrion--\n");
            system("ls --version");               //D5
            sem_wait (synch3);
            printf("--ls -R--\n");
            system("ls -R");              //D6
        wait(NULL);
    }
else
            {
        sem_unlink ("Sem1");
        sem_close(synch1);
        sem_unlink ("Sem2");
        sem_close(synch2);
        sem_unlink ("Sem3");
        sem_close(synch3);
        printf ("Fork error.\n");
    }
    sleep(3);
    return (0);
}
```



Μέρος $2^{ο}$

Ερώτημα Α:

Α)

Main Memory Size =2^{32} = 4 GB.

Αφού τα 18 bits αναπαριστούν τον αριθμό σελίδας, τότε τα υπόλοιπα 14 αναπαριστούν την μετατόπιση. Άρα το μέγεθος της κάθε σελίδας είναι 2^{14} = 16384 bytes.

Όποτε η διεργασία καταλαμβάνει 39500_{16} = 234752_{10} / 16384 = 14,232 δηλαδή 14 πλαίσια σελίδων και 5376 bytes.

Άρα η εσωτερική ελασματοποίηση είναι 16384 – 5376 = 11008 bytes.

Β)

Αφού έχουμε 14 πλαίσια σελίδας , ξεκινώντας από το 0_{10} τα 5 τελευταία πλαίσια σελίδων είναι τα 9,10,11,12,13(δεκαδικό). Άρα:

Αριθμός Σελίδας	**Αριθμός Πλαισίου**
9	16
10	225
11	170
12	35
13	51

i)

00031958_{16} = $0000\ 0000\ 0000\ 0011\ 0001\ 1001\ 0101\ 1000_2$

Αριθμός Σελίδας : $0000\ 0000\ 0000\ 0011\ 00 = 000C_{16} = 12_{10}$

Μετατόπιση: $01\ 1001\ 0101\ 1000 = 1958_{16} = 6488_{10}$

Με βάση τον πινάκα σελίδων η λογική σελίδα ($12_{10} = C_{16}$) αντιστοιχεί στην φυσική σελίδα (πλαίσιο) : $35_{10} = 23_{16}$.

Άρα η **τελική φυσική διεύθυνση** είναι η : $231958_{16} = 0010\ 0011\ 0001\ 1001\ 0101\ 1000_2$

ii)

$0001E800_{16}$: $0000\ 0000\ 0000\ 0001\ 1110\ 1000\ 0000\ 0000_2$

Αριθμός Σελίδας :$0000\ 0000\ 0000\ 0001\ 11 = 7_{10}$

Μετατόπιση : $10\ 1000\ 0000\ 0000 = 2800_{16}$

Βλέπουμε ότι η διεύθυνση δεν αντιστοιχεί σε κάποια σελίδα (page error).

Ερώτημα Β:

Α)

Μέγεθος Κύριας Μνήμης $= 2^{32} = 4GB$.

Αφού το μέγιστο υποστηριζόμενο μέγεθος ενός τμήματος είναι 16 Mbytes , τότε ο μέγιστος υποστηριζόμενος αριθμός τμημάτων είναι : 4096 MB/16 MB = 256 τμήματα.

Β)

i)

$0B00042A_{16} = 0000\ 1011\ 0000\ 0000\ 0000\ 0100\ 0010\ 1010_2$

Από τα δεδομένα έχουμε ότι η μετατόπιση είναι 24 bits και ότι τα τμήματα είναι 256. Άρα έχουμε 8 bits για τον αριθμό τμήματος.

Αριθμός Τμήματος : $0000\ 1011 = 0B_{16} = 11_{10}$.

Μετατόπιση: $0000\ 0000\ 0000\ 0100\ 0010\ 1010 = 00042A_{16} = 1066_{10}$.

Όποτε ανήκει στο πλαίσιο 11 του πινάκα, άρα αφού **Μετατόπιση τμήματος < Μήκους τμήματος** (1066<1230-11) τότε η **φυσική διεύθυνση** που αντιστοιχεί στην λογική διεύθυνση(0B00042A) **είναι η:** $9050 + 1066 = 10116_{10} = \mathbf{2784_{16}}$.

ii)

$02000B6D_{16} = 0000\ 0010\ 0000\ 0000\ 1011\ 0110\ 1101_2$

Αριθμός Τμήματος : $0000\ 0010 = 02_{16} = 2_{10}$

Μετατόπιση : $0000\ 0000\ 1011\ 0110\ 1101 = 00B6D_{16} = 2925_{10}$

Όποτε αφού ανήκει στο πλαίσιο 2 του πινάκα τμημάτων και επειδή το πλαίσιο αυτό έχει μήκος 1290(295>1290), συμπεραίνουμε ότι πρόκειται για μη έγκυρη πρόσβαση.

Ερώτημα Γ:

Α)

Μέγεθος Σελίδας = 512 Byte $= 2^9$ byte , άρα μετατόπιση σελίδας = d' = 9 bits.

Αφού λογά εκφώνησης S(αρχικό S) = αριθμός τμήματος = 8 bits και αφού d' = 9 bits , τότε αριθμός σελίδας = p = 32-8-9 = 15 bits.

Οπότε S= 8 bits, p = 15 bits και d' = 9 bits(p+d'= αρχικό d).

Β)

Μια διεργασία μπορεί να αποτελείται κατά μέγιστο από: $2^8 \times 2^{15} = 2^{23}$ σελίδες. Αφού έχουμε 2^8 τμήματα με 2^{15} σελίδες το καθένα, τότε έχουμε σύνολο 2^{23} σελίδες .

Γ)

Βάση του υπό ερωτήματος Β και των δεδομένων του Γ έχουμε :

Πίνακας Τμημάτων		
Αριθμός Τμήματος	**Διεύθυνση Βάσης**	**Μήκος Τμήματος**
0	1650	1110
1	3200	2350

Πίνακας Σελίδων Τμήματος 0	
Αριθμός Σελίδας	**Αριθμός Τμήματος**
0	151F
1	?
2	34FE
3	7E11
4	2345
5	-
….	….

Πίνακας Σελίδων Τμήματος 1	
Αριθμός Σελίδας	**Αριθμός Τμήματος**
0	-
1	2EE1
2	0B0B
3	0C11
4	?
5	1BA2
….	….

i)

$010004CF_{16}$ = 0000 0001 0000 0000 0000 0100 1100 1111_2

S=8->**Αριθμός Τμήματος** = 0000 0001 = $1_{10} = 1_{16}$

Άρα προσπελαύνουμε την **θέση 1** του πινάκα τμημάτων. Σε αυτή τη θέση του πινάκα τμημάτων βρίσκουμε την βάση του πινάκα σελίδων για αυτό το τμήμα. Που είναι η διεύθυνση 3200_{10} καθώς και το μήκος του που είναι 2350_{10} . Αφού d=0000 0000 0000 0100 1100 1111 = 1231_{10} = $0004CF_{16}$ => **d<μήκους τμήματος** , τότε η διεύθυνση είναι έγκυρη . Έπειτα αφού p = 15 bits και λόγο του 0000 0000 0000 010 = $2_{10} = 2_{16}$ μεταβαίνουμε στην **θέση 2** του **πινάκα σελίδων 1**. Όπου στην θέση 2 του πινάκα σελίδων 1 περιέχεται η τιμή **$0B0B_{16}$** . Η τιμή αυτή είναι ο **αριθμός πλαισίου στην φυσική μνήμη** και αντιστοιχεί στο δυαδικό αριθμό 0000 1011 0000 1011. Στον αριθμό αυτό ενώνουμε την μετατόπιση μέσα στην σελίδα που είναι τα υπόλοιπα 9 bits , δηλαδή 0000 1011 0000 1011 0 1100 1111_2 = $1616CF_{16}$. Οπότε **η φυσική διεύθυνση** που αντιστοιχεί **στην λογική διεύθυνση $010004CF_{16}$** είναι η **$1616CF_{16}$.**

ii)

$010009FF_{16}$ = 0000 0001 0000 0000 0000 1001 1111 1111_2

S=8-> 0000 0001 => Προσπελαύνουμε την **θέση 1** του πινάκα ταμάτων.

d=24-> 0000 0000 0000 1001 1111 1111 = 2559_{10} > 2350 = ERROR(page fault).

p=15-> 0000 0000 0000 100 = 4_{10} = 0004_{16} => Προσπελαύνουμε την θέση 4 του πινάκα σελίδων 1 που περιέχει <<?>> , δηλαδή δεν ξέρουμε σε ποιο πλαίσιο έχει φορτωθεί η σελίδα .

$000003F0_{16}$ = 0000 0000 0000 0000 0000 0011 1111 0000_2

S=8->0000 0000 => Προσπελαύνουμε την **θέση 0 του πινάκα τμημάτων**, όπου Βάση = 1650 και Μήκος = 1110.

d=24->0000 0000 0000 0011 1111 0000 = 1008_{10} < 1110_{10} = $0003F0_{16}$ (έγκυρο).

P=15-> 0000 0000 0000 001 = 1_{10} = 1_{16} => **Προσπελαύνουμε την θέση 1 του πινάκα σελίδων 0** που περιέχει <<? >>. Ωστόσο ξέρουμε ότι η λογική διεύθυνση $000003F0_{16}$ αντιστοιχεί στην φυσική διεύθυνση $E0E1F0_{16}$ = 1110 0000 1110 0001 1111 0000_2 . Επίσης αφού d' = 9 bits , τα τελευταία 9 bits της $E0E1F0_{16}$ είναι η μετατόπιση μέσα στην σελίδα και τα προηγούμενα 15 bits είναι το περιεχόμενο της θέσης 1 του πινάκα σελίδων 0**. Όποτε συμπεραίνουμε ότι το << ?>> του πινάκα σελίδων 0 είναι 1110 0000 1110 000 =7070_{16} .**

Άρα οι τελικοί πινάκες σελίδων είναι :

Πίνακας Σελίδων Τμήματος 0	
Αριθμός Σελίδας	**Αριθμός Τμήματος**
0	151F
1	7070
2	34FE
3	7E11
4	2345
5	-
….	….

Πίνακας Σελίδων Τμήματος 1	
Αριθμός Σελίδας	**Αριθμός Τμήματος**
0	-
1	2EE1
2	0B0B
3	0C11
4	-
5	1BA2
….	….

Ερώτημα Δ:

Ακολουθία αναφοράς της διεργασίας:

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

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