



ΠΑΝΕΠΙΣΤΗΜΙΟ
ΠΑΤΡΩΝ
UNIVERSITY OF PATRAS

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΚΑΙ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΜΑΘΗΜΑ : ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ

ΑΝΤΙΚΕΙΜΕΝΟ: 2η Εργαστηριακή Άσκηση 2018 - 2019



των φοιτητών

Παναγιώτη Καββαδία
ΑΜ: 1054350

pkavadia13@gmail.com
up1054350@upnet.gr

Θωμά Χατζόπουλου
ΑΜ: 1054288

chatzothomas@gmail.com
up1054288@upnet.gr

Διδάσκοντες Καθηγητές: Σπυρίδων Σιούτας
Χρήστος Μακρής
Αριστείδης Ηλίας

Περιεχόμενα:

Μέρος 1^ο:	2
Ερώτημα Α: Εξήγηση προγράμματος	2
Ερώτημα Β: Παραγωγή 4 θυγατρικών Linux/Unix διεργασιών	3
Ερώτημα Γ: Παραγωγή 5 αλυσιδωτών Linux/Unix διεργασιών	3
Ερώτημα Δ: Δημιουργία προγράμματος για χρονική μέτρηση start-end διεργασιών .	4
Ερώτημα Ε: Πρόγραμμα συγχρονισμού διεργασιών με βάση γράφο προτεραιοτήτων .	5
Ερώτημα ΣΤ: Πρόγραμμα n-διεργασιών (θυγατρικών) με κρίσιμο τμήμα	7
Μέρος 2^ο	8
Ερώτημα Α: Πίνακας τμημάτων διεργασίας, φυσικές και λογικές διευθύνσεις	8
Ερώτημα Β: Πίνακες με Φυσική και Λογική Μνήμη και Σελίδες	9
Ερώτημα Γ: Διευθυνσιοδότηση φυσικής και Λογικής Μνήμης	10
Ερώτημα Δ: Ακολουθία αναφοράς διεργασίας, Σελίδες και σφάλματα	10
Παράρτημα με κώδικες 1^ο μέρους	11
Ερώτημα Α: 1A.c	11
Ερώτημα Β: 1B.c	12
Ερώτημα Γ: 1C.c	13
Ερώτημα Δ: 1D.c	14
Ερώτημα Ε: 1E.c	16
Ερώτημα ΣΤ: 1F.c	18

Μέρος 1^ο

Ερώτημα Α

Αρχικά, αφού μεταγλωττίσουμε, εκτελούμε το δοθέν πρόγραμμα και λαμβάνουμε τα παρακάτω αναμενόμενα αποτελέσματα:

```
gcc -pthread -o 1A 1A.c
1A.c: In function 'main':
1A.c:12:11: warning: implicit declaration of function 'fork'
[-Wimplicit-function-declaration]
    pid = fork();
           ^
1A.c:12:11: note: did you mean 'fork' in the namespace 'std'?
1A.c:12:11: note: (You may want to add #include <unistd.h> to the source
file.)
t@t-VirtualBox:~/Documents$ ./1A
x = 11 y = 9
x = 12 y = 8
t@t-VirtualBox:~/Documents$ x = 11 y = 9
x = 10 y = 10
x = 11 y = 9
x = 10 y = 10
t@t-VirtualBox:~/Documents$
```

Αν βάλουμε μια printf, ώστε πριν από κάθε εκτύπωση των μεταβλητών x,y να εκτυπώνονται πληροφορίες σχετικά με την τρέχουσα διεργασία, λαμβάνουμε τα εξής αποτελέσματα:

```
t@t-VirtualBox:~/Documents$ ./1An
getpid() = 16135      , getppid() = 1813, pid=16136, if 1
x = 11 y = 9

getpid() = 16135      , getppid() = 1813, pid=16137, if 2
x = 12 y = 8

t@t-VirtualBox:~/Documents$ getpid() = 16137      , getppid() = 736, pid=0, if 2
x = 11 y = 9

getpid() = 16136      , getppid() = 736, pid=0, if 1
x = 10 y = 10

getpid() = 16136      , getppid() = 736, pid=16138, if 2
x = 11 y = 9

getpid() = 16138      , getppid() = 736, pid=0, if 2
x = 10 y = 10
```

Στο πρόγραμμα αρχικά εκχωρούμε σε δύο μεταβλητές, τις x,y, την τιμή 10 και στην συνέχεια καλούμε την fork() στην μεταβλητή pid, δημιουργώντας ένα πιστό αντίγραφο της διεργασίας που εκτελείται. Η διεργασία παιδί, που προέκυψε, αντικαθιστά το πρόγραμμα που εκτελεί (αρχικά ίδιο με του πατέρα) με το νέο πρόγραμμα. Γνωρίζουμε ότι η fork() επιστρέφει στο γονέα την ταυτότητα της διεργασίας του παιδιού, στο παιδί την τιμή 0, ενώ σε περίπτωση λάθους στο γονέα -1. Επομένως, η συνθήκη επιλογής [if (pid != 0)] ικανοποιείται σε κάθε περίπτωση, εκτός από διεργασία παιδιού. Καλώντας την fork(), ουσιαστικά δημιουργούμε ένα δέντρο από διεργασίες, οι οποίες έχουν σχέση πατέρας-παιδί μεταξύ τους.

Το παιδί από την πρώτη fork() (**Δ1**) είναι αντίγραφο της πρώτης διεργασίας (πατέρα), πριν μεταβληθούν οι τιμές των μεταβλητών x,y· έτσι είναι x = 10 y = 10.

Αφού, αρχικά, εκτελείται η διεργασία πατέρα (**Δ0**), η συνθήκη ικανοποιείται και έτσι οι τιμές των x,y μεταβάλλονται (αυξάνεται και μειώνεται κατά 1 αντίστοιχα) και εκτυπώνεται το μήνυμα « x = 11 y = 9 ».

Στην συνέχεια καλούμε ξανά την fork(). Η συνθήκη ικανοποιείται, αφού εκτελείται η διεργασία πατέρα (**Δ0**) και έτσι οι τιμές των x,y μεταβάλλονται (αυξάνεται και μειώνεται κατά 1 αντίστοιχα) και εκτυπώνεται το μήνυμα « x = 12 y = 8 ».

Το παιδί από την δεύτερη fork() (**Δ2**) είναι αντίγραφο της διεργασίας πατέρα, αφού μεταβληθούν οι τιμές των μεταβλητών x,y από το σώμα της πρώτης συνθήκης if· έτσι είναι x = 11 y = 9.

Εκτυπώθηκε	Διεργασία
X = 11 y = 9	Δ0
X = 12 y = 8	Δ0
X = 11 y = 9	Δ2
X = 10 y = 10	Δ1
X = 11 y = 9	Δ1'
X = 10 y = 10	Δ3

Με την δεύτερη, όμως, fork(), δημιουργείται και ένα ακόμα παιδί (**Δ3**), το οποίο έχει πατέρα την διεργασία της πρώτης fork() και συνεπώς οι τιμές των μεταβλητών x,y είναι x=10 y=10. Η Δ1 τότε λειτουργεί ως διεργασία πατέρας (Δ1'), οπότε θα ισχύει η δεύτερη συνθήκη και θα μεταβληθούν οι τιμές της, οπότε και θα εκτυπωθεί X = 11 y = 9.

Ερώτημα Β

Ο κώδικας του ερωτήματος βρίσκεται στο παράρτημα και στον φάκελο με τα αρχεία με όνομα « 1B.c ».

Προκειμένου να βεβαιωθούμε ότι όντως οι νέες διεργασίες είναι θυγατρικές, εκτυπώνουμε το όνομα του κάθε παιδιού και του πατέρα του. Τα αποτελέσματα που λάβαμε είναι τα εξής:

```
t@t-VirtualBox:~/Documents$ ./1B
pid = 2531
The parent (Id=2530) have a child with id: 2531

pid = 2532
The parent (Id=2530) have a child with id: 2532

pid = 2533
The parent (Id=2530) have a child with id: 2533

pid = 2534
The parent (Id=2530) have a child with id: 2534
```

Ερώτημα Γ

Ο κώδικας του ερωτήματος βρίσκεται στο παράρτημα και στον φάκελο με τα αρχεία με όνομα « 1C.c ».

Όπως ζητείται, τυπώνουμε για κάθε διεργασία το id του πατέρα της, το id της και το id του μοναδικού παιδιού που δημιουργεί.

Τα αποτελέσματα εκτέλεσης του κώδικα φαίνονται παρακάτω:

```
t@t-VirtualBox:~/Documents$ gcc -pthread -o newC newC.c
t@t-VirtualBox:~/Documents$ ./newC
The parent (Id=30480) has a child with id: 385
The parent (Id=385) has a child with id: 386
The child (Id=386) has parent with id: 385
The parent (Id=386) has a child with id: 387
The child (Id=387) has parent with id: 386
The parent (Id=387) has a child with id: 388
The child (Id=388) has parent with id: 387
The parent (Id=388) has a child with id: 389
The child (Id=389) has parent with id: 388
t@t-VirtualBox:~/Documents$
```

Ερώτημα Δ

Ο κώδικας του ερωτήματος βρίσκεται στο παράρτημα και στον φάκελο με τα αρχεία με όνομα « 1D.c ».

Στο πρόγραμμα αυτό ορίζουμε τη συνάρτηση `nothing()`, η οποία ορίζει μια μεταβλητή `int x=0` και εκτελεί την πράξη `x=x+1`. Αυτή η συνάρτηση μας βοηθάει στη μέτρηση.

Μέσα στη `main()` εκτελούμε τη συνάρτηση `time()` προκειμένου να καταμετρήσουμε και να αποθηκεύσουμε τον αριθμός των δευτερολέπτων με την βοήθεια των μεταβλητών `start` και `end`, με την παράλληλη εκτύπωση μηνυμάτων “Αρχική τιμή δευτερολέπτων” στην αρχή και “Τελική τιμή δευτερολέπτων” στο τέλος της καταμέτρησης.

Στη συνέχεια σε βρόχο `while()` με την εντολή `fork()` δημιουργούμε 100 διεργασίες (ο πατέρας θα δημιουργήσει όλες τις διεργασίες και όχι η κάθε διεργασία θα δημιουργεί άλλη διεργασία), με την βοήθεια πίνακα (`pid[]`), οι οποίες, όλες, εκτελούν τη `nothing()`. Όταν δημιουργηθούν οι 1000 διεργασίες, η διεργασία πατέρας (την έχουμε αποθηκεύσει σε ξεχωριστή μεταβλητή) εκτελεί 1000 φορές τη `waitpid()`, ώστε να περιμένει την επιτυχή ολοκλήρωση όλων των θυγατρικών.

Για να επιβεβαιώσουμε την ορθή λειτουργία του προγράμματος εκτυπώνουμε τον πίνακα `pid[]` για όλες τις διεργασίες και το πλήθος φορών που εκτελέστηκε η `waitpid()`.

Τέλος, με την βοήθεια των μεταβλητών `start` και `end` εκτελούμε τις πράξεις “`end-start`” και “`(end-start)/100`” και εκτυπώνουμε τα αποτελέσματα, για να εμφανιστεί ο μέσος χρόνος δημιουργίας, εκτέλεσης και τερματισμού των 100 διεργασιών.

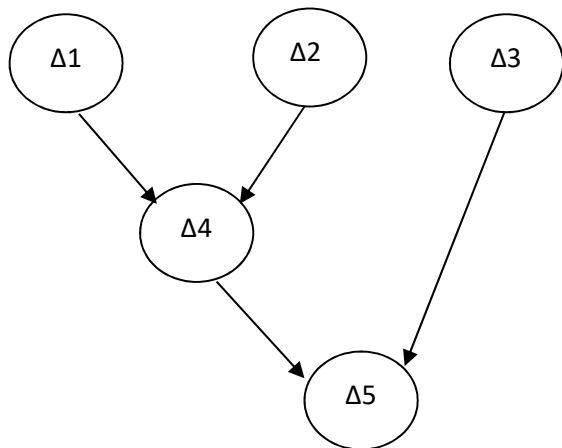
Τα αποτελέσματα της ορθής εκτέλεσης του κώδικα φαίνονται παρακάτω:

```
t@t-VirtualBox:~/Documents$ ./1D
Arxikh timh deuteroleptwn: 1546946388 sec
pid[5] = 11219
pid[4] = 11218
pid[6] = 11220
pid[7] = 11221
pid[3] = 11217
pid[8] = 11222
pid[2] = 11216
pid[9] = 11223
pid[15] = 11229

in 996
in 997
in 998
in 999
telikh timh deuteroleptwn: 1546946389 sec
Sunolikos xronos ekteleshhs: 0 sec
Mesos xronos ekteleshhs: 0 sec
t@t-VirtualBox:~/Documents$
```

Παρατηρούμε ότι ο συνολικός χρόνος εκτέλεσης, συνεπώς και ο μέσος χρόνος εκτέλεσης του προγράμματος είναι 0. Αυτό δικαιολογείται, διότι η συνάρτηση `time()` επιστρέφει χρόνο σε seconds, ενώ το υπολογιστικό σύστημα εκτελεί πράξεις σε msec σε nsecs. Άμα ο αριθμός των δημιουργούμενων διεργασιών αυξηθεί (>>100), τότε ο Συνολικός χρόνος εκτέλεσης θα είναι μεγαλύτερος του 0.

Ερώτημα Ε



Ο κώδικας του ερωτήματος βρίσκεται στο παράρτημα και στον φάκελο με τα αρχεία με όνομα « 1Ε.ε ».

Για τις διεργασίες Δ1, Δ2, Δ3, Δ4 και Δ5, οι οποίες είναι θυγατρικές μιας μόνο διεργασίας, προκειμένου να εκτελεστεί η Δ4 απαιτείται να έχουν ολοκληρωθεί οι Δ1 και Δ2 (χρήση του 1^{ου} σημαφόρου), ενώ για την εκτέλεση της Δ5 απαιτείται να έχουν ολοκληρωθεί οι Δ3 και Δ4 (χρήση του 2^{ου} σημαφόρου). Προκειμένου αυτό να επιτευχθεί, όπως ήδη φαίνεται, αρκούν 2 σημαφόροι, flagD4 και flagD5 $\in [-2,2]$.

Δημιουργήσαμε τις 5 διεργασίες, οι οποίες ως εντολές συστήματος εκτελούν, η:

```
Δ1: system("ls -l");  
Δ2: system("pwd");  
Δ3: system("ps -l");  
Δ4: system("pwd");  
Δ5: system("tree");
```

Από την εκτέλεση του προγράμματος λάβαμε τα εξής αποτελέσματα:

```

t@t-VirtualBox:~/Documents$ ./1E

Third Progress:
Second Progress:
First Progress:
/home/t/Documents

F S  UID    PID    PPID    C  PRI   NI ADDR SZ WCHAN  TTY          TIME CMD
0 S   1000   1442   1432    0   80    0  -  7391 wait  pts/0      00:00:00 bash
0 S   1000  12668   1442    0   80    0  -  1675 wait  pts/0      00:00:00 1E
1 S   1000  12671  12668    0   80    0  -  1675 wait  pts/0      00:00:00 1E
1 Z   1000  12672  12668    0   80    0  -    0 -      pts/0      00:00:00 1E <defunct>
1 S   1000  12673  12668    0   80    0  -  1675 wait  pts/0      00:00:00 1E
1 S   1000  12674  12668    0   80    0  -  1675 futex_ pts/0      00:00:00 1E
1 S   1000  12675  12668    0   80    0  -  1675 futex_ pts/0      00:00:00 1E
0 S   1000  12676  12673    0   80    0  -  1157 wait  pts/0      00:00:00 sh
4 R   1000  12678  12676    0   80    0  -  9005 -      pts/0      00:00:00 ps
0 S   1000  12679  12671    0   80    0  -  1157 wait  pts/0      00:00:00 sh

total 132
-rwxr-xr-x 1 t t 8472 Iav 9 16:08 1A
-rw-rw-r-- 1 t t 1408 Iav 9 16:08 1A.c
-rwxr-xr-x 1 t t 8432 Iav 2 14:42 1B
-rw-r--r-- 1 t t 395 Iav 8 19:06 1B.c
-rwxr-xr-x 1 t t 8344 Δεκ 30 23:58 1B_old
-rw-r--r-- 1 t t 785 Δεκ 31 00:26 1B_old.c
-rwxr-xr-x 1 t t 8432 Iav 2 17:59 1C
-rw-r--r-- 1 t t 1050 Iav 4 12:33 1C.c
-rwxr-xr-x 1 t t 8480 Iav 7 17:28 1C_new
-rw-r--r-- 1 t t 621 Iav 8 10:40 1C_new.c
-rwxr-xr-x 1 t t 8560 Iav 8 13:19 1D
-rw-r--r-- 1 t t 1497 Iav 8 18:49 1D.c
-rwxr-xr-x 1 t t 12872 Iav 9 18:13 1E
-rw-r--r-- 1 t t 1908 Iav 9 18:13 1E.c
-rwxr-xr-x 1 t t 8528 Iav 8 11:19 newC
-rw-r--r-- 1 t t 527 Iav 8 10:57 newC.c

Fourth Progress:
/home/t/Documents

Fifth Progress:
t@t-VirtualBox:~/Documents$ .
|-- 1A
|-- 1A.c
|-- 1B
|-- 1B_old
|-- 1B_old.c
|-- 1C
|-- 1C.c
|-- 1C_new
|-- 1C_new.c
|-- 1D
|-- 1D.c
|-- 1E
|-- 1E.c
|-- newC
|-- newC.c

0 directories, 16 files

^C
t@t-VirtualBox:~/Documents$

```

Φαίνεται η παράλληλη εκτέλεση των Δ1, Δ2, Δ3 και η σειρά εκτέλεσης όλων των διεργασιών, όπως ορίζει ο γράφος προτεραιοτήτων.

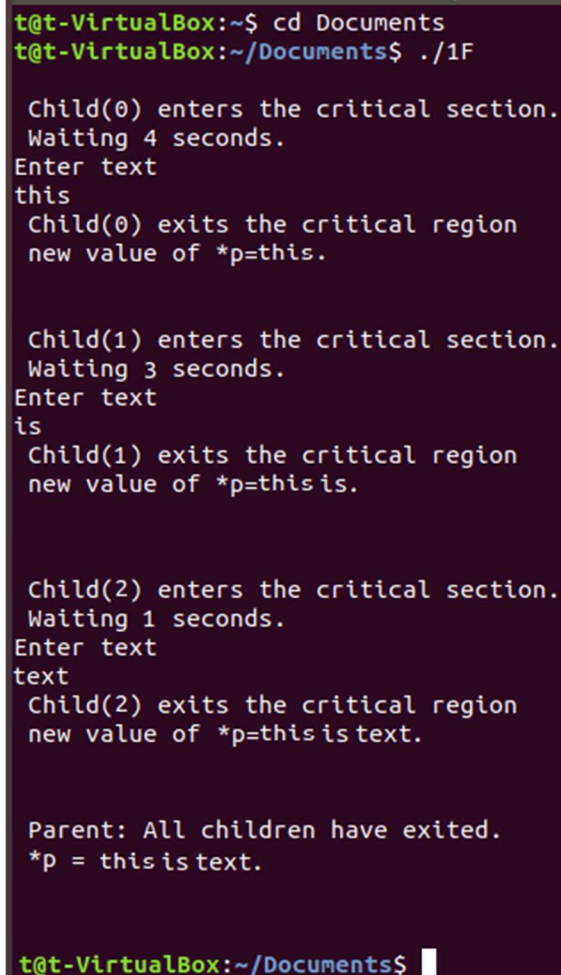
Ερώτημα ΣΤ

Ο κώδικας του ερωτήματος βρίσκεται στο παράρτημα και στον φάκελο με τα αρχεία με όνομα « 1F.c ».

Αρχικά δημιουργούμε N-διεργασίες (θυγατρικές) κάθε μία από τις οποίες να εκτελεί το παρακάτω κομμάτι ψευδο-κώδικα:

```
Char *p;  
t = *p;  
    CONCATENATE TO p A CHILD[i]--text;  
    sleep (nSeconds[i]);  
*p = t;
```

όπου στο text εισάγει ο χρήστης κείμενο στην κάθε διεργασία. Για την δημιουργία τυχαίων αριθμών χρησιμοποιούμε κατάλληλα την συνάρτηση rand(), ενώ να ενώσουμε τις επιμέρους συμβολοσειρές χρησιμοποιούμε την strcat().



```
t@t-VirtualBox:~$ cd Documents  
t@t-VirtualBox:~/Documents$ ./1F  
  
Child(0) enters the critical section.  
Waiting 4 seconds.  
Enter text  
this  
Child(0) exits the critical region  
new value of *p=this.  
  
Child(1) enters the critical section.  
Waiting 3 seconds.  
Enter text  
is  
Child(1) exits the critical region  
new value of *p=this is.  
  
Child(2) enters the critical section.  
Waiting 1 seconds.  
Enter text  
text  
Child(2) exits the critical region  
new value of *p=this is text.  
  
Parent: All children have exited.  
*p = this is text.  
  
t@t-VirtualBox:~/Documents$
```


Μέρος 2^ο

Ερώτημα Α

i) Αν πάμε να βρούμε τις φυσικές διευθύνσεις (δηλαδή να προσθέσουμε στην βάση το όριο) που καταλαμβάνει κάθε τμήμα παρατηρούμε τα εξής:

ΤΜΗΜΑ	ΠΡΩΤΗ ΔΙΕΥΘΥΝΣΗ	ΤΕΛΕΥΤΑΙΑ ΔΙΕΥΘΥΝΣΗ
0	1024	2048
1	9896	10024
2	512	640
3	3912	5312
4	1536	2560
5	5688	7688

Παρατηρούμε ότι οι θέσεις μνήμης που καταλαμβάνει το τμήμα 0 επικαλύπτονται με τις θέσεις μνήμης που καταλαμβάνει το τμήμα 4 από την θέση 1536 έως και την θέση 2048. Αν και επιτρέπεται η επικάλυψη θέσεων μνήμης μεταξύ διεργασιών δεν επιτρέπεται η επικάλυψη θέσεων μνήμης της ίδιας διεργασίας (πλην της περίπτωσης ταύτισης των επικαλυπτόμενων τμημάτων) άρα ο πίνακας δεν μπορεί να είναι πίνακας τμημάτων μιας διεργασίας.

ii)

(0, 256)	1280
(1, 40)	9936
(2, 512)	SEGMENTATION FAULT*
(3, 1000)	4912
(5, 1536)	7224

*Το (2, 512) είναι εκτός του ορίου του τμήματος καθώς έχει όριο 128 επομένως θα προκληθεί διακοπή του προγράμματος με σχετικό μήνυμα σφάλματος.

Ερώτημα Β

Λογική μνήμη Διεργασίας 1	Πίνακας σελίδων Διεργασίας 1	Φυσική μνήμη	Πλαίσιο Σελίδας
Σελίδα 0	4		0
Σελίδα 1	7	Δ1,Σ1	1
Σελίδα 2	15	Δ1,Σ4	2
Σελίδα 3	12		3
Σελίδα 4	2	Δ1,Σ0	4
Σελίδα 5	10		5
Σελίδα 6	13		6
Σελίδα 7	18		7
		Δ2, Σ0	8
			9
Λογική μνήμη Διεργασίας 2		Δ1,Σ5	10
Σελίδα 0	8	Δ2, Σ3	11
Σελίδα 1	20	Δ1,Σ3	12
Σελίδα 2	14	Δ1,Σ6	13
Σελίδα 3	11	Δ2, Σ2	14
Σελίδα 4	23	Δ1,Σ2	15
Σελίδα 5	22	Δ3,Σ0	16
Σελίδα 6	19	Δ2, Σ7	17
Σελίδα 7	17	Δ1,Σ7	18
		Δ2, Σ6	19
		Δ2, Σ1	20
		Δ3, Σ1	21
Λογική μνήμη Διεργασίας 3		Δ2, Σ5	22
Σελίδα 0	16	Δ2, Σ4	23
Σελίδα 1	21	Δ3, Σ4	24
Σελίδα 2	14	Δ3,Σ7	25
Σελίδα 3	27	Δ3,Σ5	26
Σελίδα 4	24	Δ3,Σ3	27
Σελίδα 5	26		28
Σελίδα 6	29	Δ3, Σ6	29
Σελίδα 7	25	Δ3, Σ2	30
			31

Ερώτημα Γ

Λογική Διεύθυνση Μνήμης Διεργασίας 1

0	1	1	1	1	0	0	0	0	1	0	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Φυσική Διεύθυνση Μνήμης

0	1	0	1	1	1	1	0	0	0	0	1	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Ερώτημα Δ

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

Στον πίνακα με κόκκινο χρώμα φαίνεται η νέα σελίδα που φορτώνεται όταν προκύπτει σφάλμα σελίδας

ΠΑΡΑΡΤΗΜΑ

Ερώτημα Α: 1Α.c

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int pid;
    int x,y;

    x = 10;
    y = 10;

    pid = fork();

    if (pid != 0)
    {
        x++;
        y--;
    }
    //print getpid(),getppid and pid for first print(x,y)
    printf("getpid() = %i \t , getppid() = %i, pid=%i, if 1\n",getpid(), getppid(), pid);   printf("x =
%i y = %i\n",x,y);
    printf("\n");

    pid = fork();

    if (pid != 0)
    {
        x++;
        y--;
    }
    //print getpid(),getppid and pid for second print(x,y)
    printf("getpid() = %i \t , getppid() = %i, pid=%i, if 2\n",getpid(), getppid(), pid);
    printf("x = %i y = %i\n",x,y);
    printf("\n");

    return (0);
}
```

Ερώτημα Β: 1B.c

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int pid;
    for(int i=0; i<3; i++)
        pid = fork();          //δημιουργία διεργασιών
        printf("pid = %i\n",pid); //print pid
        if (pid == 0)           //if pid=child
        {
            printf("The child (Id=%i) have ", getpid() );
            printf("parent with id: %i\n\n", getppid() );
        }
        else                   //if pid=parent
        {
            printf("The parent (Id=%i) have ", getpid() );
            printf("a child with id: %i\n\n", pid );
        }
    }

    return (0);
}
```

Ερώτημα Γ: 1C.c

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

int main()
{
    int pid;
    int status;
    //for parent procces
    printf("The parent (Id=%i) has a child with id: %i\n", getppid(), getpid() );
    for (int i=0; i<4; i++)
    {
        pid = fork();    //dhmiourgia diergasiwn
        if (pid == 0)    //if child
        {
            printf("The child (Id=%i) has ", getpid() );
            printf("parent with id: %i\n", getppid() );
        }
        else
        {
            printf("The parent (Id=%i) has ", getpid() );
            printf("a child with id: %i\n", pid );
            wait(&status);
            break;    //if father break
        }
    }

    return (0);
}
```

Ερώτημα Δ: 1D.c

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

#define N 100

int nothing() //nothing function
{
    int x=0;
    x=x+1;
    //printf("x = %i\n", x);
    return(0);
}

int main()
{
    time_t start; //start sec
    time_t end; //end sec
    double timeT; //sunolikos xronos
    pid_t parent=getpid(); //father pid

    int k=0; //counters
    int q=0;

    pid_t pid[N]={0}; //array for progresses
    int child_status; //for father

    time(&start); //start time
    printf("Arxikh timh deuteroleptwn: %i sec\n", start);

    while(k<N)
    {
        k++;
        if(fork()==0) //dhmiourgia diergasiwn: if child
        {
            nothing();
            pid[k]=getpid();
            printf("pid[%i] = %i\n", k, pid[k]);
            q=0;
            break;
        }
        else //parent
        {
            q++;
        }
    }
}
```



```

if(getpid()==parent){ //if father
    for(k=0; k<N; k++)
    {
        printf("in %i\n", k);
        pid_t wpid = waitpid(pid[k], &child_status,0); //wait
    }
}

if(getpid()==parent){ //if father

    time(&end);    //stop time
    printf("telikh timh deuteroleptwn: %i sec\n", end);

    timeT = end-start; //run time
    printf("Sunolikos xronos ekteleshhs: %i sec\n", timeT);

    timeT = timeT/N; //Avg time
    printf("Mesos xronos ekteleshhs: %i sec\n", timeT);
}
return(0);
}

```

Ερώτημα Ε: 1Ε.ε

```
#include <stdio.h>
```

```

#include <stdlib.h>
#include <time.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <sys/shm.h>
#include <unistd.h>
#include <semaphore.h>
#include <errno.h>
#include <fcntl.h>

typedef sem_t Semaphore;
Semaphore *sem_1; //first semaphore for 1/2-->4 progresses
Semaphore *sem_2; //second semaphore for 4/3-->5 progresses

int main()
{
    pid_t pid[5]={0}; //array for progresses
    int i=0; //counter
    int child_status; //for parent
    system("clear"); //clear

    sem_1=sem_open("Sem1",O_CREAT | O_EXCL, 0644,0); //open sem1
    sem_2=sem_open("Sem2",O_CREAT | O_EXCL, 0644,0); //open sem2

    for (int i=0; i<5; i++) //dhmiourgia diergasiwn
    {
        pid[i]=fork();
        if(pid[i]==0)
        {
            break;
        }
        //printf("pid[%i] = %i\n", i, pid[i]);
    }

    if(pid[0]==0){ //1h diergasia
        printf("First Progress:\n");
        system("ls -l"); //emfanish arxeiwn
        printf("\n\n");
        sem_post(sem_1); //up(sem1)
    }

    if(pid[0]!=0 && pid[1]==0) //2h diergasia
    {
        printf("Second Progress:\n");
        system("pwd"); //trexwn fakelos ergasias
        printf("\n\n");
        sem_post(sem_1); //up(sem1)
    }

    if(pid[0]!=0 && pid[1]!=0 && pid[2]==0) //3h diergasia
    {

```

```

        printf("Third Progress:\n");
        system("ps -l"); //trexouses diergasies
        printf("\n\n");
        sem_post(sem_2); //up(sem2)
    }

    if(pid[0]!=0 && pid[1]!=0 && pid[2]!=0 && pid[3]==0) //4h diergasia
    {
        sem_wait(sem_1); //down(sem1)
        sem_wait(sem_1); //down(sem1)
        printf("Fourth Progress:\n");
        system("pwd"); //trexwn fakelos ergasias
        printf("\n\n");
        sem_post(sem_2); //up(sem2)
    }

    if(pid[0]!=0 && pid[1]!=0 && pid[2]!=0 && pid[3]!=0 && pid[4]==0) //5h diergasia
    {
        sem_wait(sem_2);
        sem_wait(sem_2);
        printf("Fifth Progress:\n");
        system("tree");
        printf("\n\n");
        //printf("End of programm");
    }

    if(pid[0]!=0 && pid[1]!=0 && pid[2]!=0 && pid[3]!=0 && pid[4]!=0) //parent
    {
        for (i = 0; i < 4; i++)
        {
            pid_t wpid = waitpid(pid[i], &child_status, 0);
        }

        sem_unlink ("Sem1");
        sem_close(sem_1);
        sem_unlink ("Sem2");
        sem_close(sem_2);
    }

    return(0);
}

```

Ερώτημα ΣΤ: 1F.c

```
#include <stdio.h>
```

```

#include <stdlib.h>
#include <time.h>
#include <sys/wait.h>
#include <sys/types.h>
#include <sys/shm.h>
#include <unistd.h>
#include <semaphore.h>
#include <errno.h>
#include <fcntl.h>
#include <string.h>

#define N 3 //plh8os diergasiwn

typedef sem_t Semaphore;
Semaphore *mutex;

int randomInt(int lower,int upper) {
    int num = (rand() %(upper - lower + 1)) + lower;
    return(num);
}

int main(){
    pid_t pid; //pinakas paidiwn
    int i=0;    //counter
    char *t;    //temp pointer
    char temp[255]=""; //temp string for strcat()
    char *p;    //shared memory
    key_t shmkey; // shared memory key
    int shmid;    // shared memory id
    int child_status; //for father
    int random;
    int nSeconds[N];

    system("clear");

    t= temp;

    for(int l=0; l<N; l++){ // random number array
        nSeconds[l] = randomInt(1,5);
    }

    mutex = sem_open ("pSem", O_CREAT | O_EXCL, 0644, 1); //open mutex

    if (shmid < 0) { //if error exit
        perror ("shmget\n");
        exit (1);
    }

    for (int i=0; i<N; i++){ // dhmiourgia diergasiwn
        pid=fork();
        if (pid < 0) { //if error

```

```

        sem_unlink ("pSem");
        sem_close(mutex);
        printf ("Fork error.\n");
    }
    else if(pid==0){// if child
        break;
    }
    //printf("pid[%i] = %i\n", i, pid[i]);
}

if (pid > 0) //if parent
{
    // wait for all children to exit
    while (pid == waitpid (-1, NULL, 0))
    {
        if (errno == ECHILD)
            break;
    }

    printf (" Parent: All children have exited.\n");
    printf (" *p = %d\n\n", *p);

    shmdt (p);
    shmctl (shmid, IPC_RMID, 0);

    sem_unlink ("pSem");
    sem_close(mutex); //close mutex
}
else{
    sem_wait (mutex);        //down(mutex)

    printf (" Child(%d) enters the critical section.\n", i);
    printf (" Waiting %d seconds.\n", nSeconds[i]);

    sleep(nSeconds[i]);    //sleep
    printf("Enter text\n");

    char c=getchar();        //clean buffer
    scanf("%[^\\n]s",temp); //scan text
    t=p;                    //temp pointer with new text

    strcat(t,temp);          //concatenating new text to t
    p=t;                    //set *p to point to the full text
    *p=*t;

    printf (" Child(%d) exits the critical region\n", i);
    printf (" new value of *p=%s.\n\n\n", p);

    sem_post(mutex);        //up(mutex)
}
return(0);
}

```