ΕΡΓΑΣΤΗΡΙΟ 4

Διαχείριση διεργασιών

1. Να εκτελεστούν οι εντολές (α) ps, (β) ps –x, (γ) ps –elf και να σχολιαστεί το αποτέλεσμα. Για πληροφορίες σχετικά με τις επιλογές που χρησιμοποιούνται κατά την κλήση της εντολής, ανατρέξτε στη σελίδα βοήθειας της εντολής ps που εμφανίζεται εκτελώντας την εντολή man ps.

   Η εντολή ps εκτυπώνει τα στοιχεία των διεργασιών που έχουν το ίδιο ενεργό αναγνωριστικό χρήστη (effective user id, euid) με αυτό του τρέχοντος χρήστη (δηλαδή του χρήστη που εκτελεί την εντολή και το όνομα του οποίου εκτυπώνεται από την εντολή whoami). Εάν η εντολή κληθεί χωρίς ορίσματα εκτυπώνει τις διεργασίες **που έχουν ξεκινήσει από τον τρέχοντα φλοιό**, συμπεριλαμβανομένου και του ίδιου του φλοιού.

```
amarg@amarg-vbox:~$ ps
    PID TTY          TIME CMD
   2546 pts/0    00:00:00 bash
   2637 pts/0    00:00:00 ps
```

Η εντολή ps με το διακόπτη –x εμφανίζει **όλες τις διεργασίες του συστήματος** με κάτοχο τον τρέχοντα χρήστη.

```
amarg@amarg-vbox:~$ ps -x
    PID TTY      STAT   TIME COMMAND
   1830 ?        Ss     0:00 /lib/systemd/systemd --user
   1831 ?        S      0:00 (sd-pam)
   1836 ?        S<sl   0:00 /usr/bin/pulseaudio --daemonize=no --
   1838 ?        SNsl   0:00 /usr/libexec/tracker-miner-fs
   1841 ?        Sl     0:00 /usr/bin/gnome-keyring-daemon --daemo
```

Η εντολή ps με το διακόπτη –e εμφανίζει όλες τις ενεργές διεργασίες χρησιμοποιώντας το προεπιλεγμένο στυλ εμφάνισης του Linux (generic Linux format).

```
amarg@amarg-vbox:~$ ps -e
    PID TTY          TIME CMD
      1 ?        00:00:01 systemd
      2 ?        00:00:00 kthreadd
      3 ?        00:00:00 rcu_gp
      4 ?        00:00:00 rcu_par_gp
      6 ?        00:00:00 kworker/0:0H-kblockd
      7 ?        00:00:00 kworker/u2:0-events_unbound
      8 ?        00:00:00 mm_percpu_wq
      9 ?        00:00:00 ksoftirqd/0
     10 ?        00:00:00 rcu_sched
     11 ?        00:00:00 migration/0
     12 ?        00:00:00 idle_inject/0
```

Προκειμένου να εμφανίσουμε το μέγιστο πλήθος πληροφοριών που εκτυπώνει η εντολή, χρησιμοποιούμε μαζί με το διακόπτη –e (που όπως είδαμε εμφανίζει όλες τις ενεργές διεργασίες), μαζί με τους διακόπτες –f (full listing) και –l (long format). Για να το κάνουμε αυτό γράφουμε ps –e –l –f ή πιο απλά ps –elf. Το αποτέλεσμα είναι

```
amarg@amarg-vbox:~$ ps -elf
F S UID          PID    PPID  C PRI  NI ADDR SZ WCHAN  STIME TTY          TIME CMD
4 S root           1       0  0  80   0 - 25532 -      09:43 ?        00:00:01 /sbin/init splash
1 S root           2       0  0  80   0 -     0 -      09:43 ?        00:00:00 [kthreadd]
1 I root           3       2  0  60 -20 -     0 -      09:43 ?        00:00:00 [rcu_gp]
1 I root           4       2  0  60 -20 -     0 -      09:43 ?        00:00:00 [rcu_par_gp]
1 I root           6       2  0  60 -20 -     0 -      09:43 ?        00:00:00 [kworker/0:0H-kblockd]
1 I root           7       2  0  80   0 -     0 -      09:43 ?        00:00:00 [kworker/u2:0-events_unbound]
1 I root           8       2  0  60 -20 -     0 -      09:43 ?        00:00:00 [mm_percpu_wq]
1 S root           9       2  0  80   0 -     0 -      09:43 ?        00:00:00 [ksoftirqd/0]
1 I root          10       2  0  80   0 -     0 -      09:43 ?        00:00:00 [rcu_sched]
1 S root          11       2  0 -40   - -     0 -      09:43 ?        00:00:00 [migration/0]
5 S root          12       2  0   9   - -     0 -      09:43 ?        00:00:00 [idle_inject/0]
1 S root          14       2  0  80   0 -     0 -      09:43 ?        00:00:00 [cpuhp/0]
5 S root          15       2  0  80   0 -     0 -      09:43 ?        00:00:00 [kdevtmpfs]
1 I root          16       2  0  60 -20 -     0 -      09:43 ?        00:00:00 [netns]
```

Οι πιο σημαντικές από τις πληροφορίες που εμφανίζονται εδώ, είναι η κατάσταση της διεργασίας, ο κωδικός της διεργασίας και της γονικής διεργασίας, η τιμή της προτεραιότητας, ο χρόνος έναρξης και ο συνολικός χρόνος και το όνομα της εντολής που δημιούργησε τη διεργασία. Για περισσότερες πληροφορίες σχετικά με τις στήλες στην έξοδο της εντολής ανατρέξτε στη διεύθυνση

https://man7.org/linux/man-pages/man1/ps.1.html

Για έναν κατάλογο με παραδείγματα χρήσης της εντολής ps ανατρέξτε στη διεύθυνση

https://www.tecmint.com/ps-command-examples-for-linux-process-monitoring/

2. Να εμφανίσετε το δέντρο των διεργασιών του συστήματος εκτελώντας στη γραμμή εντολών την εντολή pstree.

3. Να δημιουργήσετε το αρχείο sample.txt με την εντολή touch.txt και τον κατάλογο test με την εντολή mkdir test. Στη συνέχεια για το παραπάνω αρχείο και τον παραπάνω κατάλογο, χρησιμοποιώντας την εντολή chmod να δώσετε τα επόμενα δικαιώματα:
    i. r w – r - - r - x ενεργοποιώντας το setuid bit
    ii. r – x r w – r - - ενεργοποιώντας το setuid bit και το setgid bit
    iii. r w x r – x r - - ενεργοποιώντας το setgid bit και το sticky bit
    iv. r w - - w x r w - ενεργοποιώντας όλα τα ειδικά bits
    v. r w – r w x – w x ενεργοποιώντας το sticky bit
    vi. r – x rw - - r w x ενεργοποιώντας το userid bit και το sticky bit

    Κατασκευάζοντας το αρχείο sample.txt και τον κατάλογο test με τις εντολές touch και mkdir, διαπιστώνουμε πως οι μάσκες δικαιωμάτων για αυτά τα δύο αντικείμενα είναι αντίστοιχα r w – r w – r - - (664) και r w x r w x r – x (775) όπως επιβάλλεται από την τιμή της umask η οποία είναι 002. Όσον αφορά τώρα την εκχώρηση των παραπάνω δικαιωμάτων, αυτή πραγματοποιείται με την εντολή

    chmod ABCD name

    όπου η παράμετρος name είναι είτε το sample.txt είτε το test, ενώ η μάσκα ABCD (που τώρα έχει μήκος 4 ψηφία επειδή θα πρέπει να ορίσουμε και τα τρία ειδικά bits) κατασκευάζεται ως εξής:

Για τα εννέα δικαιώματα πρόσβασης χρησιμοποιούνται τα τρία τελευταία bits BCD, με το γνωστό πλέον τρόπο και η σωστή τιμή για τις παραπάνω περιπτώσεις είναι η εξής (τα bits της δεύτερης τριάδας είναι bold για λόγους ευκολίας στην προεπισκόπηση του κειμένου):

i. r w – r - - r – x ➔ 1 1 0 **1 0 0** 1 0 1 ➔ 6 4 5
ii. r – x r w – r - - ➔ 1 0 1 **1 1 0** 1 0 0 ➔ 5 6 4
iii. r w x r – x r - - ➔ 1 1 1 **1 0 1** 1 0 0 ➔ 7 5 4
iv. r w - - w x r w – ➔ 1 1 0 **0 1 0** 1 1 0 ➔ 6 3 6
v. r w – r w x – w x ➔ 1 1 0 **1 1 1** 0 1 0 ➔ 6 7 3
vi. r – x r w – r w x ➔ 1 0 1 **1 0 0** 1 1 1 ➔ 5 6 7

Η τιμή του A που θα τοποθετηθεί μπροστά υπολογίζεται ως εξής: έστω u το setuid bit, g το setgid bit και t το sticky bit. Το A τότε είναι ένας δυαδικός αριθμός της μορφής ugt όπου το u παίρνει την τιμή 1 εάν ενεργοποιείται το setuid bit και την τιμή 0 στην αντίθετη περίπτωση, το g παίρνει την τιμή 1 εάν ενεργοποιείται το setgid bit και την τιμή 0 στην αντίθετη περίπτωση και το το t παίρνει την τιμή 1 εάν ενεργοποιείται το sticky bit και την τιμή 0 στην αντίθετη περίπτωση. Μετά τον ορισμό των τιμών για τα τρία αυτά bits, η τιμή του A μετατρέπεται στο οκταδικό σύστημα. Κατά συνέπεια, η τιμή του A για τις παραπάνω περιπτώσεις υπολογίζεται ως εξής:

1. Ενεργοποίηση setuid bit ➔ A = 1 0 0 = 4
2. Ενεργοποίηση setuid bit και setgid bit ➔ A =1 1 0 = 6
3. Ενεργοποίηση setgid bit και sticky bit ➔ A = 0 1 1 = 3
4. Ενεργοποίηση όλων των ειδικών bits ➔ A = 1 1 1 = 7
5. Ενεργοποίηση του sticky bit ➔ A = 0 0 1 =1
6. Ενεργοποίηση του setuid bit και του sticky bit ➔ A = 1 0 1 = 5

Συνδυάζοντας τα παραπάνω αποτελέσματα, η εντολή chmod για την καθεμία από τις επόμενες περιπτώσεις θα πρέπει να κληθεί ως εξής (όπου name το sample.txt και το test για το αρχείο και τον κατάλογο αντίστοιχα):

a. r w – r - - r – x ενεργοποιώντας το setuid bit ➔ **chmod 4645 name**
b. r – x r w – r - - ενεργοποιώντας το setuid bit και το setgid bit ➔ **chmod 6564 name**
c. r w x r – x r - - ενεργοποιώντας το setgid bit και το sticky bit ➔ **chmod 3754 name**
d. r w - - w x r w – ενεργοποιώντας όλα τα ειδικά bits ➔ **chmod 7636 name**
e. r w – r w x – w x ενεργοποιώντας το sticky bit ➔ **chmod 1673 name**
f. r – x r w – r w x ενεργοποιώντας το userid bit και το sticky bit ➔ **chmod 5567 name**

Το αποτέλεσμα της διαδικασίας για το αρχείο sample.txt ακολουθεί στη συνέχεια (οι ίδιες εντολές θα εκτελεστούν και για τον κατάλογο test αντικαθιστώντας το όνομα sample.txt με το όνομα test).

```
amarg@amarg-vbox:~$ chmod 4645 sample.txt
amarg@amarg-vbox:~$ ls -l sample.txt
-rwSr--r-x 1 amarg amarg 26 Νοε   2 10:57 sample.txt
amarg@amarg-vbox:~$ chmod 6564 sample.txt
amarg@amarg-vbox:~$ ls -l sample.txt
-r-srwSr-- 1 amarg amarg 26 Νοε   2 10:57 sample.txt
amarg@amarg-vbox:~$ chmod 3754 sample.txt
amarg@amarg-vbox:~$ ls -l sample.txt
-rwxr-sr-T 1 amarg amarg 26 Νοε   2 10:57 sample.txt
amarg@amarg-vbox:~$ chmod 7636 sample.txt
amarg@amarg-vbox:~$ ls -l sample.txt
-rwS-wsrwT 1 amarg amarg 26 Νοε   2 10:57 sample.txt
amarg@amarg-vbox:~$ chmod 1673 sample.txt
amarg@amarg-vbox:~$ ls -l sample.txt
-rw-rwx-wt 1 amarg amarg 26 Νοε   2 10:57 sample.txt
amarg@amarg-vbox:~$ chmod 5567 sample.txt
amarg@amarg-vbox:~$ ls -l sample.txt
-r-srw-rwt 1 amarg amarg 26 Νοε   2 10:57 sample.txt
```

Θυμηθείτε πως ένα μικρό γράμμα (s ή t) για το ειδικό bit σημαίνει πως το bit εκτέλεσης x που βρίσκεται σε εκείνη τη θέση είναι ενεργοποιημένο, ενώ αντίθετα, ένα κεφαλαίο γράμμα (S ή T) για το ειδικό bit σημαίνει πως το bit εκτέλεσης x που βρίσκεται σε εκείνη τη θέση **δεν** είναι ενεργοποιημένο. Με τον τρόπο αυτό είναι δυνατή η κωδικοποίηση δύο πληροφοριών (περί της ενεργοποίησης ή όχι του execute bit και του sticky bit) χρησιμοποιώντας ένα απλό bit.

4. Να εκτελέσετε την εντολή ls –l με αυξημένη τιμή προτεραιότητας κατά 12 και την εντολή pwd με αυξημένη τιμή προτεραιότητας κατά 5.

   Η προεπιλεγμένη τιμή προτεραιότητας στο λειτουργικό σύστημα Linux είναι η τιμή 0 όπως φαίνεται αν εκτελέσουμε την εντολή nice χωρίς ορίσματα.

```
amarg@amarg-vbox:~/Desktop$ nice
0
amarg@amarg-vbox:~/Desktop$
```

   Στη γενική περίπτωση η τιμή της προτεραιότητας που ορίζεται με την nice έχει τιμές από -20 έως 19 με τη μέγιστη προτεραιότητα να είναι η -20 και η ελάχιστη προτεραιότητα να είναι η 19.

   Όσον αφορά στην επίλυση της άσκησης, οι εντολές που δίδονται εάν εκτελεστούν ως έχουν θα εκτελεστούν με την προεπιλεγμένη τιμή προτεραιότητας που είναι η τιμή 0. Για να εκτελέσουμε την εντολή ls –l με αυξημένη τιμή προτεραιότητας κατά 12 θα πρέπει να γράψουμε

```
amarg@amarg-vbox:~$ nice -n 12 ls -l
total 97440
-rwxrwxr-x  1 amarg amarg    16872 Σεπ  29 12:13 acc1
-rw-rw-r--  1 amarg amarg     2264 Σεπ  29 12:13 acc1.o
-rwxrwxr-x  1 amarg amarg    17064 Οκτ  18 11:41 addrInfo
-rwxrwxrwx  1 amarg amarg     1064 Οκτ  18 11:44 addrInfo.c
-rw-rw-r--  1 amarg amarg     2704 Οκτ  18 11:40 addrInfo.o
-rw-rw-r--  1 amarg amarg      386 Οκτ  20 11:53 allusers
-rw-rw-r--  1 amarg amarg      167 Σεπ  22 10:27 awkscript
drwxrwxr-x  3 amarg amarg     4096 Οκτ  15 14:36 bin
-rw-rw-r--  1 amarg amarg      124 Οκτ  28 17:30 book.txt
```

   ενώ για να εκτελέσουμε την εντολή pwd με αυξημένη τιμή προτεραιότητας κατά 5 θα πρέπει να γράψουμε

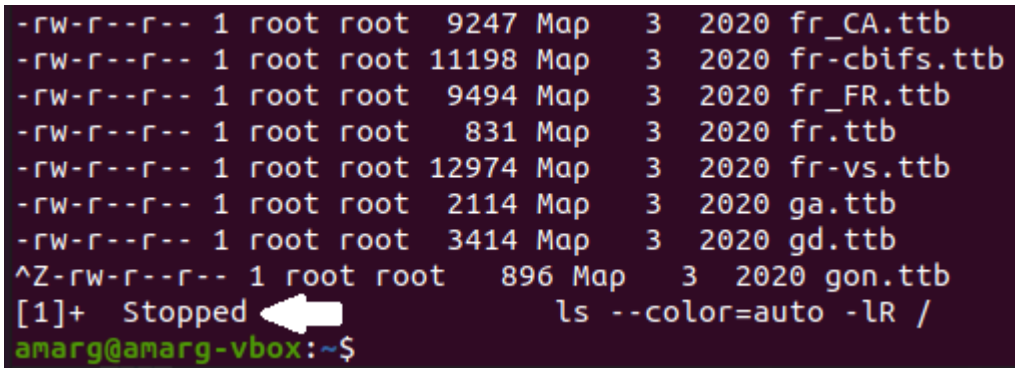
```
amarg@amarg-vbox:~$ nice -n 5 pwd
/home/amarg
amarg@amarg-vbox:~$
```

   Σημειώστε πως η προτεραιότητα **NI** που ορίζεται με τη nice σχετίζεται με το χώρο του χρήστη και **ΔΕΝ** είναι η πραγματική προτεραιότητα **PR** της διεργασίας η οποία σχετίζεται με το χώρο του πυρήνα. Οι δύο αυτές τιμές προτεραιότητα σχετίζονται με την έκφραση **PR = 20 + NI** και για τιμή PR = 0 η τιμή του NI είναι ίση με -20. Δυστυχώς είναι αδύνατο να δοθούν εδώ περισσότερες πληροφορίες και ευελπιστώ όλα αυτά να τα δείτε αναλυτικά στο μάθημα των λειτουργικών συστημάτων.

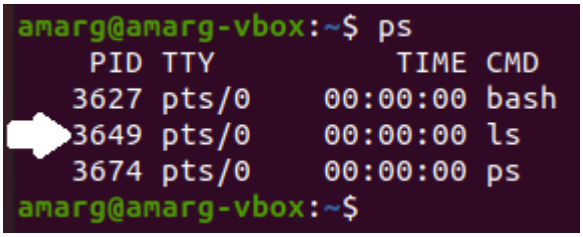
5. (α) Να εκτελέσετε την εντολή ls –lR / (που εμφανίζει την απόλυτη διαδρομή όλων των αρχείων του συστήματος) από 2-3 δευτερόλεπτα λειτουργίας σταματήστε τη πατώντας το συνδυασμό πλήκτρων Ctrl-Z. (β) Εκτελέστε την εντολή ps και αναζητήστε την εντολή στη λίστα των ενεργών διεργασιών που εκτυπώνεται. (γ) Καταγράψτε το pid της. (δ) Τερματίστε την εντολή γράφοντας kill -9 pid (για να δείτε το ρόλο της επιλογής -9 εμφανίστε τη σελίδα βοηθείας της εντολής γράφοντας man kill). (ε) επαληθεύστε τον τερματισμό της εντολής εκτελώντας ξανά την εντολή ps. (στ) Να επαναλάβετε το βήμα (α) αλλά αντί για το βήμα (β) να εκτελέσετε την εντολή fg για να επαναφέρετε τη διεργασία στο προσκήνιο. Στη συνέχεια σταματήστε τη εκ νέου με Ctrl-Z και αυτή τη φορά

χρησιμοποιήστε την εντολή bg για να εκτελέσετε τη διεργασία στο παρασκήνιο. Είναι δυνατή τώρα η αναστολή της εκτέλεση της εντολής και αν όχι, γιατί?

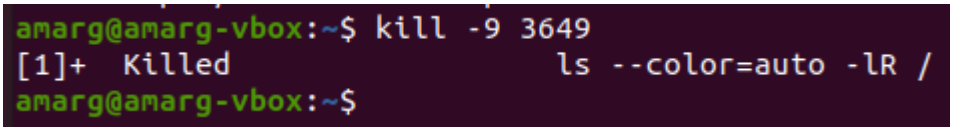
Αρχικά εκτελούμε την εντολή ls –l R και μετά από λίγο πατάμε Ctrl-Z αναστέλλοντας τη λειτουργία της.



Εκτελώντας την εντολή ps διαπιστώνουμε πως ο κωδικός της διεργασίας (process id, PID) που δημιουργήθηκε από την εκτέλεση της εντολής είναι 3649.



Στη συνέχεια τερματίζουμε τη διεργασία καλώντας την εντολή kill -9 PID όπου PID = 3649 ο κωδικός διεργασίας.



και επαληθεύουμε πως η διεργασία έχει τερματιστεί εκτελώντας ξανά την ps, η οποία πλέον ΔΕΝ την εμφανίζει.

Ας εκτελέσουμε ξανά την εντολή και ας τη σταματήσουμε και πάλι με το συνδυασμό Ctrl-Z. Εάν τώρα στη γραμμή εντολών πληκτρολογήσουμε fg (foreground) και πατήσουμε το Enter προκαλούμε τη συνέχιση της λειτουργίας της σταματημένης εντολής στο προσκήνιο, ενώ εάν αντίθετα πληκτρολογήσουμε στη γραμμή εντολών bg (background) και πατήσουμε το Enter, προκαλούμε τη συνέχιση της λειτουργίας της σταματημένης εντολής στο παρασκήνιο. Η διαφορά ανάμεσα στο προσκήνιο και στο παρασκήνιο είναι η εξής: Όταν μία διεργασία εκτελείται στο προσκήνιο δεσμεύει το τερματικό μας αφού η έξοδός της εκτυπώνεται στην οθόνη και κατά συνέπεια για να μπορέσουμε να χρησιμοποιήσουμε ξανά το τερματικό θα πρέπει η διεργασία να ολοκληρώσει τη λειτουργία της. Ωστόσο, δεν χάνουμε την πρόσβαση στο πληκτρολόγιο και κατά συνέπεια μπορούμε να αλληλοεπιδράσουμε με μία διεργασία η οποία εκτελείται στο προσκήνιο (για παράδειγμα, να αναστείλουμε ή να τερματίσουμε τη λειτουργία της με τους συνδυασμούς πλήκτρων Ctrl-Z και Ctrl-C). Από την άλλη πλευρά, όταν μία διεργασία εκτελείται στο παρασκήνιο δεν συνδέεται με το πληκτρολόγιο και ως εκ τούτου δεν μπορούμε να αλληλοεπιδράσουμε μαζί της. Για παράδειγμα, εάν επαναφέρουμε τη σταματημένη ls –lR / στο παρασκήνιο εκτελώντας την εντολή bg θα διαπιστώσουμε πως τώρα οι συνδυασμοί πλήκτρων Ctrl-C και Ctrl-Z δεν λειτουργούν διότι έχει χαθεί η σύνδεση με το πληκτρολόγιο και κατά συνέπεια θα πρέπει αναγκαστικά να περιμένουμε να ολοκληρωθεί. Εναλλακτικά μπορούμε να ανοίξουμε ένα άλλο τερματικό, να ανακτήσουμε το PID της διεργασίας με την εντολή ps και να τερματίσουμε τη λειτουργία της με την εντολή kill όπως κάναμε προηγουμένως.

Να πληκτρολογηθεί και να εκτελεστεί ο κώδικας των επόμενων παραδειγμάτων που επιδεικνύουν τη συνδυασμένη χρήση των συναρτήσεων fork, wait και exec.

Παράδειγμα 1. Απλό πρόγραμμα με τη γονική και τη θυγατρική διεργασία να εκτυπώνουν η καθεμία το δικό της μήνυμα (αρχείο forkex1.c).

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

void  main(void) {
    pid_t  pid;
```

Η συνάρτηση fork προκαλεί την κλωνοποίηση της γονικής διεργασίας και για το λόγο αυτό παρά το γεγονός πως υπάρχει μία μόνο printf, θα εκτυπωθούν δύο μηνύματα, αφού η printf θα κληθεί και από τις δύο διεργασίες.

```
    pid = fork();
    printf ("Hello !! my pid is %d\n", getpid()); }
```

Η έξοδος της εφαρμογής ακολουθεί στη συνέχεια.

```
amarg@amarg-vbox:~/shared/Lab4$ ./forkex1
Hello !! my pid is 4005
amarg@amarg-vbox:~/shared/Lab4$ Hello !! my pid is 4006
```

Παρατηρήστε πως το δεύτερο μήνυμα εκτυπώθηκε μετά την εμφάνιση του command prompt και αυτό αποτελεί ένδειξη πως η γονική διεργασία ολοκληρώθηκε πρώτη. Παρατηρήστε επίσης πως οι κωδικοί των διεργασιών έχουν συνεχόμενες τιμές (4005 και 4006) κάτι που είναι εύλογο και συμβαίνει σχεδόν πάντοτε.

Παράδειγμα 2. Σε αυτό το παράδειγμα, η χρήση της συνάρτησης wait διασφαλίζει τον τερματισμό της γονικής διεργασίας μετά τον τερματισμό της θυγατρικής διεργασίες (αρχείο forkex2.c).

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

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

Σε αυτό το σημείο καλείται η fork για τη δημιουργία της θυγατρικής διεργασίας. Η fork επιστρέφει την τιμή 0 στη θυγατρική διεργασία και τιμή διάφορη του μηδενός (και ίση με το pid της θυγατρικής διεργασίας) στη γονική διεργασία, ενώ σε περίπτωση αποτυχίας επιστρέφει την τιμή -1.

```
   pid_t child_pid = fork();

   if (child_pid < 0) {
      perror("fork() failed");
      exit(EXIT_FAILURE);
   } else if (child_pid == 0) {
```

Η θυγατρική διεργασία εκτυπώνει τις τιμές των pid (process id) και ppid (parent process id) καλώντας τις συναρτήσεις getpid () και getppid (που επιστρέφουν αυτές τις δύο τιμές).

```
    printf("from child: pid=%d, parent_pid=%d\n", (int) getpid(), (int) getppid());
```

Σε αυτό το σημείο και πριν την κλήση της exit για τον τερματισμό της διεργασίας μπορούμε να γράψουμε τον κώδικα που υλοποιεί τη διαδικασία που θα πραγματοποιεί η διεργασία.

```
    exit(33);

  } else if (child_pid > 0) {
```

Η γονική διεργασία εκτυπώνει τις τιμές των pid (process id) και ppid (parent process id) καλώντας τις συναρτήσεις getpid () και getppid (που επιστρέφουν αυτές τις δύο τιμές).

```
    printf("from parent: pid=%d child_pid=%d\n", (int) getpid(), (int) child_pid);
```

Στο σημείο αυτό καλούμε τη συνάρτηση waitpid προκειμένου η γονική διεργασία να **περιμένει** τον ομαλό ή απότομο τερματισμό της θυγατρικής διεργασίας προκειμένου να τερματιστεί και η ίδια. Η waitpid επιστρέφει το process id τη θυγατρικής διεργασίας ενώ επιστρέφει πληροφορίες σχετικά με τον κώδικα επιστροφής και το λόγο του τερματισμού.

```
    int status;
    pid_t waited_pid = waitpid (child_pid, &status, 0);

    if (waited_pid < 0) {
       perror("waitpid() failed");
       exit(EXIT_FAILURE);
    } else if (waited_pid == child_pid) {
```

Η μακροεντολή WIFEXITED επιστρέφει true εάν η θυγατρική διεργασία ολοκληρώθηκε κανονικά και false στην αντίθετη περίπτωση. Στην περίπτωση του κανονικού τερματισμού, η WIFEXITED επιστρέφει τον κωδικό επιστροφής της θυγατρικής διεργασίας.

```
       if (WIFEXITED(status)) {
          printf("from parent: child exited with code %d\n", WEXITSTATUS(status)); }}}
```

Η έξοδος της εφαρμογής ακολουθεί στη συνέχεια.

```
amarg@amarg-vbox:~/shared/Lab4$ ./forkex2
from parent: pid=4057 child_pid=4058
from child: pid=4058, parent_pid=4057
from parent: child exited with code 33
amarg@amarg-vbox:~/shared/Lab4$
```

Παράδειγμα 3. Σε αυτό το παράδειγμα, η γονική και η θυγατρική διεργασία αρχικοποιούν η καθεμία το δικό της αντίγραφο των μεταβλητών local και global σε διαφορετικές τιμές (αρχείο forkex3c).

```
#include <unistd.h>
#include <sys/types.h>
#include <errno.h>
#include <stdio.h>
```

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

int global = 0;

int main() {
   pid_t child_pid;
   int status;
   int local = 0;
```

Σε αυτό το σημείο καλείται η fork για τη δημιουργία της θυγατρικής διεργασίας. Η fork επιστρέφει την τιμή 0 στη θυγατρική διεργασία και τιμή διάφορη του μηδενός (και ίση με το pid της θυγατρικής διεργασίας) στη γονική διεργασία, ενώ σε περίπτωση αποτυχίας επιστρέφει την τιμή -1.

```
   child_pid = fork();

   if (child_pid >= 0) /* fork succeeded */ {
```

Κώδικας για τη θυγατρική διεργασία (child_pid = 0)

```
      if (child_pid == 0) /* fork() returns 0 for the child process */ {
         printf("child process!\n");
```

Η αύξηση των τιμών των μεταβλητών κατά μία μονάδα, αφορά **μόνο** τα αντίγραφα των μεταβλητών που ανήκουν στη θυγατρική διεργασία. Αντίθετα οι τιμές των μεταβλητών που ανήκουν στη γονική διεργασία **δεν** επηρεάζονται.

```
         local++;
         global++;
         printf("child PID =  %d, parent pid = %d\n", getpid(), getppid());
         printf("\nchild's local = %d, child's global = %d\n",local,global);

         char * cmd[] = {"whoami",(char*)0};
```

Η θυγατρική διεργασία μέσα από την execv καλεί την εντολή whoami για την εκτύπωση του ονόματος του χρήστη.

```
         return execv("/usr/bin/whoami",cmd);  }
```

Κώδικας για τη γονική διεργασία (child_pid > 0)

```
      else /* parent process */ {

         printf("parent process!\n");
         printf("parent PID =  %d, child pid = %d\n", getpid(), child_pid);
```

Η γονική διεργασία καλεί τη wait για να περιμένει την ολοκλήρωση της εκτέλεσης της θυγατρικής διεργασίας πριν τερματιστεί και η ίδια και στη συνέχεια εκτυπώνει τον κωδικό εξόδου της θυγατρικής διεργασίας που επιστρέφεται από τη μακροεντολή WEXITSTATUS

```
         wait(&status); /* wait for child to exit, and store child's exit status */
         printf("Child exit code: %d\n", WEXITSTATUS(status));
```

Η εκτύπωση των τιμών των μεταβλητών της γονικής διεργασίας αποκαλύπτει ότι πράγματι η αλλαγή που έγινε προηγουμένως αφορά μόνο στα αντίγραφα των μεταβλητών της θυγατρικής διεργασίας. Αντίθετα, οι μεταβλητές της γονικής διεργασίας έχουν τις αρχικές τους μηδενικές τιμές.

```
    printf("\nParent'z local = %d, parent's  global = %d\n",local,global);
    printf("Parent says bye!\n");
    exit(0);  /* parent exits */ }}
```

Εάν η fork επιστρέψει αρνητικό αριθμό, έχει λάβει χώρα κάποιο σφάλμα το οποίο εκτυπώνεται από τη συνάρτηση perror η οποία εκτυπώνει τη λεκτική περιγραφή του σφάλματος errno.

```
else /* failure */ {
   perror("fork");
   exit(0);  }}
```

Η έξοδος της εφαρμογής ακολουθεί στη συνέχεια.

```
amarg@amarg-vbox:~/shared/Lab4$ ./forkex3
parent process!
parent PID =  4191, child pid = 4192
child process!
child PID =  4192, parent pid = 4191

child's local = 1, child's global = 1
amarg
Child exit code: 0

Parent'z local = 0, parent's  global = 0
Parent says bye!
```

Παράδειγμα 4. Δημιουργία τοπολογίας διεργασιών δύο επιπέδων με τη γονική διεργασία να δημιουργεί δύο θυγατρικές διεργασίες (αρχείο forkex4.c).

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

Το γεγονός πως στην προκειμένη περίπτωση θα δημιουργηθούν δύο θυγατρικές διεργασίες, σημαίνει πως η συνάρτηση fork θα πρέπει να κληθεί δύο φορές.

```
int main () {
   int pid1, pid2, status1, status2, child1, child2;
```

Η πρώτη κλήση της fork

```
   pid1 = fork();
   if (pid1<0) {
      printf ("Fork operation was uncussesfull\n");
      return -1 ; }
   else if (pid1>0) {
```

```
printf ("Parent process with pid %d and ppid %d \n",getpid(), getppid());
child1 = wait(&status1);
printf ("Child with code %d has beem terminated\n", child1);
```

Η δεύτερη κλήση της fork – προκειμένου οι δύο θυγατρικές διεργασίες να ανήκουν στο ίδιο επίπεδο, δηλαδή αμφότερες να αποτελούν άμεσα παιδιά της γονικής διεργασίας, θα πρέπει η δεύτερη fork να τοποθετηθεί στον κώδικα της γονικής διεργασίας.

```
pid2 = fork();
```

Αυτός ο κώδικας όπως και πριν εκτελείται από τη γονική διεργασία

```
if (pid2>0)
   {
    printf ("Parent process \n");
    child2 = wait (&status2);
    printf ("Child with code %d has been terminated\n", child2); }
else {
```

Αυτός ο κώδικας εκτελείται από την πρώτη θυγατρική διεργασία. Αυτή η διεργασία καλεί τη συνάρτηση execl μέσα από την οποία εκτελεί την εντολή ***pwd*** του λειτουργικού συστήματος.

```
    printf ("Child2 with pid %d and ppid %d \n", getpid(), getppid());
    printf ("Calling pwd command...");
    execl ("/usr/bin/pwd","pwd",NULL); }}
else {
```

Αυτός ο κώδικας εκτελείται από τη δεύτερη θυγατρική διεργασία. Αυτή η διεργασία καλεί τη συνάρτηση execl μέσα από την οποία εκτελεί την εντολή ***mkdir*** του λειτουργικού συστήματος για την κατασκευή του καταλόγου OSLab.

```
printf ("Child1 with pid %d kai ppid %d \n", getpid(), getppid());
printf ("Creating a new directory...\n");
execlp ("mkdir", "mkdir", "OSLab", NULL); }}
```

Η έξοδος της εφαρμογής ακολουθεί στη συνέχεια.

```
Parent process with pid 4269 and ppid 3627
Child1 with pid 4270 kai ppid 4269
Creating a new directory...
Child with code 4270 has beem terminated
Parent process
Child2 with pid 4271 and ppid 4269
/home/amarg/shared/Lab4
Child with code 4271 has been terminated
```

Παρατηρήστε πως οι κωδικοί της γονικής (4269) και των δύο θυγατρικών διεργασιών (4270 και 4271) είναι συνεχόμενοι, κάτι που είναι αναμενόμενο. Από την άλλη πλευρά ο κωδικός του γονέα της γονικής διεργασίας είναι ο 3627. Ποιος είναι ο γονέας της γονικής διεργασίας? Μελετώντας την έξοδο της εντολής ps

```
amarg@amarg-vbox:~/shared/Lab4$ ps
    PID TTY          TIME CMD
   3627 pts/0    00:00:00 bash
```

διαπιστώνουμε πως το 3627 είναι το PID του φλοιού bash ο οποίος επομένως είναι ο γονέας της γονικής διεργασίας. Το γεγονός αυτό είναι αναμενόμενο, αφού για να δημιουργήσουμε τη γονική διεργασία εκτελέσαμε την εφαρμογή από τη γραμμή εντολών του λειτουργικού, δηλαδή ουσιαστικά από το φλοιό bash. Ας αναφέρουμε παρεμπιπτόντως πως η γονική διεργασία του φλοιού bash είναι ο terminal server που χρησιμοποιείται σε κάθε περίπτωση όπως φαίνεται από την εντολή ps –elf

```
0 S amarg     2493    1830  0  80   0 - 40566 poll_s 09:45 ?        00:00:00 /usr/libexec/gvfsd-metadata
0 S amarg     2496    2086  0  80   0 - 105697 poll_s 09:45 ?       00:00:00 update-notifier
0 S amarg     2538    1830  0  80   0 - 204970 poll_s 09:48 ?       00:00:29 /usr/libexec/gnome-terminal-server
0 S amarg     3627    2538  0  80   0 -  2753 do_wai 12:10 pts/0    00:00:00 bash
0 T amarg     3707    3627  0  80   0 -  2343 do_sig 12:30 pts/0    00:00:00 ls --color=auto -lR /
```

γεγονός που και αυτό είναι αναμενόμενο, αφού για να χρησιμοποιήσουμε το φλοιό του λειτουργικού συστήματος θα πρέπει να χρησιμοποιήσουμε ένα τερματικό.

Παράδειγμα 5. Δημιουργία τοπολογίας διεργασιών τριών επιπέδων με τη γονική και τη θυγατρική διεργασία να δημιουργούν η καθεμία θυγατρική διεργασία (αρχείο forkex5.c).

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

Όπως και στην προηγούμενη άσκηση, το γεγονός πως στην προκειμένη περίπτωση θα δημιουργηθούν δύο θυγατρικές διεργασίες, σημαίνει πως η συνάρτηση fork θα πρέπει να κληθεί δύο φορές.

```
int main () {
   int pid1, pid2, status1, status2, child1, child2;
   pid1 = fork();
   if (pid1<0) {
      printf ("Fork operation was uncussesfull\n");
      return -1 ; }
   else if (pid1>0) {
       printf ("Parent process with pid %d and ppid %d \n",getpid(), getppid());
       child1 = wait(&status1);
       printf ("Child with code %d has beem terminated\n", child1);
       }
   else {
```

Ωστόσο, στην προκειμένη περίπτωση, η δεύτερη κλήση της fork – προκειμένου οι δύο θυγατρικές διεργασίες να σχετίζονται μέσω μίας σχέσης πατέρα / παιδιού και κατά συνέπεια η μία διεργασία να αποτελεί παιδί της μίας και γονέας της άλλης - θα πρέπει η δεύτερη fork να τοποθετηθεί στον κώδικα της γονικής διεργασίας.

```
       printf ("Child1 with pid %d kai ppid %d \n", getpid(), getppid());
       pid2 = fork();
       if (pid2>0)
        {
          printf ("Parent process (child1) for child2\n");
          child2 = wait(&status2);
          printf ("Child with code %d has been terminated\n", child2); }
      else {
          printf ("Child2 with pid %d and ppid %d \n", getpid(), getppid());
          printf ("Calling pwd command...");
          execl ("/usr/bin/pwd","pwd",NULL); }
```

```
printf ("Creating a new directory...\n");
execlp ("mkdir", "mkdir", "OSLab", NULL); }}
```

Η έξοδος της εφαρμογής ακολουθεί στη συνέχεια. Οι δύο νέες διεργασίες λειτουργούν όπως και πριν και κατά συνέπεια, η δημιουργία του καταλόγου OSLab δεν είναι δυνατή, αφού αυτός ο κατάλογος είχε δημιουργηθεί προηγουμένως, με αποτέλεσμα την εμφάνιση του κατάλληλου μηνύματος σφάλματος.

```
amarg@amarg-vbox:~/shared/Lab4$ ./forkex5
Parent process with pid 4892 and ppid 3627
Child1 with pid 4893 kai ppid 4892
Parent process (child1) for child2
Child2 with pid 4894 and ppid 4893
/home/amarg/shared/Lab4
Child with code 4894 has been terminated
Creating a new directory...
mkdir: cannot create directory ‘OSLab’: File exists
Child with code 4893 has beem terminated
amarg@amarg-vbox:~/shared/Lab4$
```

Άσκηση. Συνδυάζοντας τα Παραδείγματα 4 και 5 να κατασκευάσετε την τοπολογία διεργασιών του επόμενου σχήματος. Η κάθε διεργασία θα εκτυπώνει τις τιμές των παραμέτρων pid και ppid και στη συνέχεια θα καλεί την exec για να εκτελέσει την εντολή που τη συνοδεύει στο επόμενο σχήμα (για όσες έχουν συνοδευτική εντολή). Η κάθε γονική διεργασία θα αναμένει την ολοκλήρωση των θυγατρικών διεργασιών της και μετά θα τερματίζει και η ίδια.