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.
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:

   r w – r - - r - x ενεργοποιώντας το setuid bit
   r – x r w – r - - ενεργοποιώντας το setuid bit
   r w x r – x r - - ενεργοποιώντας το setgid bit
   r w - - w – r w - ενεργοποιώντας το setuid bit
   r w – r w x – w - ενεργοποιώντας το sticky bit
   r – x r - - r w x ενεργοποιώντας το sticky bit

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

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

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[]) {

   /* Create child process.
    * On success, fork() returns the process ID of the child (> 0) to the
    * parent and 0 to the child. On error, -1 is returned. */

   pid_t child_pid = fork();

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

      /* Print message from child process.
       * getpid() returns the PID (process identifier) of current process,
       * which is typically int but doesn't have to be, so we cast it.
       * getppid() returns the PID of the parent process. */

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

      /* We can do something here, e.g. load another program using exec(). */
      exit(33);

   } else if (child_pid > 0) {

      /* Print message from parent process.  */
      printf("from parent: pid=%d child_pid=%d\n", (int) getpid(), (int) child_pid);
```

```
    /* Wait until child process exits or terminates.
     * The return value of waitpid() is PID of the child process, while
     * its argument is filled with exit code and termination reason. */

    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) {

       if (WIFEXITED(status)) {
          /* WIFEXITED(status) returns true if the child has terminated
           * normally. In this case WEXITSTATUS(status) returns child's
           * exit code. */
          printf("from parent: child exited with code %d\n", WEXITSTATUS(status)); }}}
```

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;

   /* now create new process */
   child_pid = fork();

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

         /* Increment the local and global variables */
         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};
        return execv("/usr/bin/whoami",cmd);  }

      else /* parent process */ {
        printf("parent process!\n");
        printf("parent PID =  %d, child pid = %d\n", getpid(), child_pid);
        wait(&status); /* wait for child to exit, and store child's exit status */
        printf("Child exit code: %d\n", WEXITSTATUS(status));
        /* The change in local and global variable in child process should not reflect
           here in parent process. */
        printf("\nParent'z local = %d, parent's  global = %d\n",local,global);
        printf("Parent says bye!\n");
        exit(0);  /* parent exits */ }}
  else /* failure */ {
     perror("fork");
     exit(0);  }}
```

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>

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);
      pid2 = fork();
      if (pid2>0)
         {
          printf ("Parent process \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); }}
    else {
         printf ("Child1 with pid %d kai ppid %d \n", getpid(), getppid());
         printf ("Creating a new directory...\n");
         execlp ("mkdir", "mkdir", "OSLab", NULL); }}
```

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>

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 {
        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); }}
```

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.