ΕΡΓΑΣΤΗΡΙΟ 7

Σήματα

Να πληκτρολογηθεί και να εκτελεστεί ο κώδικας των επόμενων παραδειγμάτων που επιδεικνύουν τη χρήση των σημάτων ως μέσο επικοινωνία μεταξύ διεργασιών.

Παράδειγμα 1. Σύλληψη του σήματος SIGINT (αρχείο sigExample1.c).

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

Ο χειριστής του σήματος SIGINT το μόνο που κάνει είναι να εκτυπώσει απλά ένα ενημερωτικό μήνυμα ότι το εν λόγω σήμα έχει παραληφθεί. Με άλλα λόγια *τροποποιείται η προεπιλεγμένη συμπεριφορά* σύμφωνα με την οποία η παραλαβή αυτού του σήματος προκαλεί τη διακοπή της διεργασίας.

```
void sig_handler(int signo) {
   if (signo == SIGINT)
      printf("received SIGINT\n"); }

int main(void) {
```

Η συνάρτηση signal συσχετίζει το σήμα SIGINT με το χειριστή handler και σε περίπτωση που αυτό δεν συμβεί επιστρέφει τον κωδικό σφάλματος SIG_ERR.

```
   if (signal(SIGINT, sig_handler) == SIG_ERR)
      printf("\ncan't catch SIGINT\n");
  // A long long wait so that we can easily
  // issue a signal to this process
```

Η ανταλλαγή σημάτων ανάμεσα στις διεργασίες είναι ταχύτατη και πραγματοποιείται σε απειροελάχιστο χρονικό διάστημα για τα δεδομένα του χρήστη. Προκειμένου να είναι δυνατή η αποστολή ενός σήματος από το χρήστη στη διεργασία, εκτελούμε ένα βρόχο με άπειρες επαναλήψεις έτσι ώστε να διασφαλίσουμε ότι η διεργασία θα συλλάβει το σήμα που απέστειλε ο χρήστης. Το σήμα SIGINT δεν προκαλεί τη διακοπή της διεργασίας αφού ο χειριστής handler απλά εκτυπώνει ένα μήνυμα και ο μοναδικός τρόπος για να συμβεί αυτό είναι να ανοίξουμε ένα τερματικό, να προσδιορίσουμε τον κωδικό της διεργασίας με την ps και να της στείλουμε το σήμα SIGKILL (9) (ανοίξτε λοιπόν ένα δεύτερο terminal, εκτελέστε την εντολή ps –x για να βρείτε το PID της διεργασίας και μετά τερματίστε την με την εντολή kill -9 PID – δείτε την έξοδο που ακκολουθεί).

```
  while(1)
     sleep(1);
  return 0; }
```

Αυτή η συμπεριφορά επιδεικνύεται στη συνέχεια.

Παράδειγμα 2. Αδυναμία σύλληψης των σημάτων SIGKILL και SIGSTOP

(αρχείο sigExample2.c).

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

Σε αυτό το παράδειγμα επιδεικνύεται ένα ενδιαφέρον χαρακτηριστικό του λειτουργικού συστήματος, σύμφωνα με το οποίο τα σήματα SIGKILL και SIGSTOP δεν μπορούν να υποστούν χειρισμό από το χρήστη (υπό την έννοια πως δεν μπορεί να μεταβληθεί η προεπιλεγμένη συμπεριφορά τους), παρά μόνο από τον πυρήνα. Αυτό γίνεται διότι αυτά τα σήματα επιτρέπουν τον βίαιο τερματισμό μιας διεργασίας η οποία για κάποιο λόγο έχει σταματήσει να αποκρίνεται. Δεν είναι δύσκολο να γίνει αντιληπτό πως εάν αναθέσουμε στη διεργασία το χειρισμό αυτών των δύο σημάτων και η διεργασία σταματήσει να αποκρίνεται τότε δεν υπάρχει κανένας τρόπος για να τερματιστεί.

Ο χειριστής των σημάτων SIGKILL και SIGSTOP το μόνο που κάνει για το καθένα από αυτά, είναι να εκτυπώσει απλά ένα ενημερωτικό μήνυμα ότι το εν λόγω σήμα έχει παραληφθεί. Με άλλα λόγια *τροποποιείται η προεπιλεγμένη συμπεριφορά* σύμφωνα με την οποία η παραλαβή αυτού του σήματος προκαλεί τον απότομο τερματισμό της διεργασίας.

```
void sig_handler(int signo) {
   if (signo == SIGKILL)
      printf("received SIGKILL\n");
   if (signo == SIGSTOP)
      printf("received SIGSTOP\n"); }

int main(void) {
```

Το γεγονός πως το λειτουργικό σύστημα δεν επιτρέπει το χειρισμό των δύο αυτών σημάτων από τη διεργασία, παρουσιάζεται στην πράξη ελέγχοντας τον κωδικό επιστροφή της συνάρτησης signal. Εάν αυτός ο κωδικός είναι ίσος με SIG_ERR, αυτό σημαίνει πως δεν κατέστη δυνατή η συσχέτιση του καθενός από αυτά τα σήματα με την αντίστοιχη συνάρτηση χειρισμού, επειδή πολύ απλά το λειτουργικό σύστημα απαγόρευσε την πραγματοποίηση αυτής της συσχέτισης. Η εκτέλεση του κώδικα αποκαλύπτει πώς τα πράγματα είναι όντως έτσι. Τα δύο λοιπόν αυτά σήματα, δεν μπορούν να εκχωρηθούν για χειρισμό στις διεργασίες του χρήστη, αφού αυτό αποτελεί καθήκον του πυρήνα και μόνο του πυρήνα.

```
   if (signal(SIGKILL, sig_handler) == SIG_ERR)
      printf("\ncan't catch SIGKILL\n");
   if (signal(SIGSTOP, sig_handler) == SIG_ERR)
      printf("\ncan't catch SIGSTOP\n");
  // A long long wait so that we can easily
  // issue a signal to this process
```

Όπως έχει ήδη αναφερθεί, η ανταλλαγή σημάτων ανάμεσα στις διεργασίες είναι ταχύτατη και πραγματοποιείται σε απειροελάχιστο χρονικό διάστημα και για το λόγο αυτό, όπως και πριν προκειμένου να είναι δυνατή η αποστολή ενός σήματος από το χρήστη στη διεργασία, εκτελούμε ένα βρόχο με άπειρες επαναλήψεις έτσι ώστε να διασφαλίσουμε ότι η διεργασία θα συλλάβει το σήμα που απέστειλε ο χρήστης.

```
  while(1)
    sleep(1);
  return 0; }
```

Παράδειγμα εξόδου της εφαρμογής ακολουθεί στη συνέχεια. Η εφαρμογή όπως και πριν τερματίζεται με την εντολή kill -9 PID όπου PID ο κωδικός της διεργασίας που μπορεί να βρεθεί με την εντολή ps –x. Ωστόσο, τώρα μπορούμε να χρησιμοποιήσουμε και το συνδυασμό Ctrl-C αφού δεν τροποποιήσαμε τη συμπεριφορά της διεργασίας όταν δέχεται το σήμα SIGINT.

```
  59216 pts/0    Ss+    0:00 bash
  59697 ?        Sl     0:08 /snap/snap-store/481/usr
  60537 pts/1    Ss     0:00 bash
  61270 pts/1    S+     0:00 ./sigExample2
  61278 pts/2    Ss     0:00 bash
  61289 pts/2    R+     0:00 ps -x
amarg@amarg-vbox:~/shared/Lab7$ kill -9 61270
amarg@amarg-vbox:~/shared/Lab7$
```

```
amarg@amarg-vbox:~/shared/Lab7$ ./sigExample2

can't catch SIGKILL

can't catch SIGSTOP
Killed
amarg@amarg-vbox:~/shared/Lab7$
```

Παράδειγμα 3. Τα σήματα γενικής χρήσεως SIGUSR1 και SIGUSR2
(αρχείο sigExample3.c).

Αυτό το παράδειγμα επιδεικνύει τη χρήση των σημάτων SIGUSR1 και SIGUSR2 τα οποία είναι γενικής χρήσεως και μπορούν να χρησιμοποιηθούν για να καλύψουν τις υφιστάμενες σε κάθε περίπτωση ανάγκης του χρήστη. Στο απλό παράδειγμα που ακολουθεί. ο χρήστης εκτελεί την εφαρμογή sigExample3 με το γνωστό τρόπο. Στη συνέχεια ανοίγει ένα δεύτερο τερματικό, βρίσκει το process ID της διεργασίας με την εντολή ps -x και κατόπιν χρησιμοποιώντας την εντολή kill, αποστέλει στη διεργασία τα δύο παραπάνω σήματα. Αυτά τα δύο σήματα έχουν συσχετιστεί με την ίδια συνάρτηση χειρισμού, η οποία όταν καλείται κατά την λήψη του σήματος, απλά εκτυπώνει ένα ενημερωτικό μήνυμα σύμφωνα με το οποίο τα σήματα παρελήφθησαν.

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

Ο χειριστής των σημάτων SIGUSR1 και SIGUSR2

```
void sig_handler(int signo) {
   if (signo == SIGUSR1)
      printf("received USR1\n");
   if (signo == SIGUSR2)
      printf("received USR2\n"); }
```

Η συνάρτηση signal συσχετίζει τα σήματα SIGUSR1 και SIGUSR2 με το χειριστή handler και σε περίπτωση που αυτό δεν συμβεί επιστρέφει τον κωδικό σφάλματος SIG_ERR.

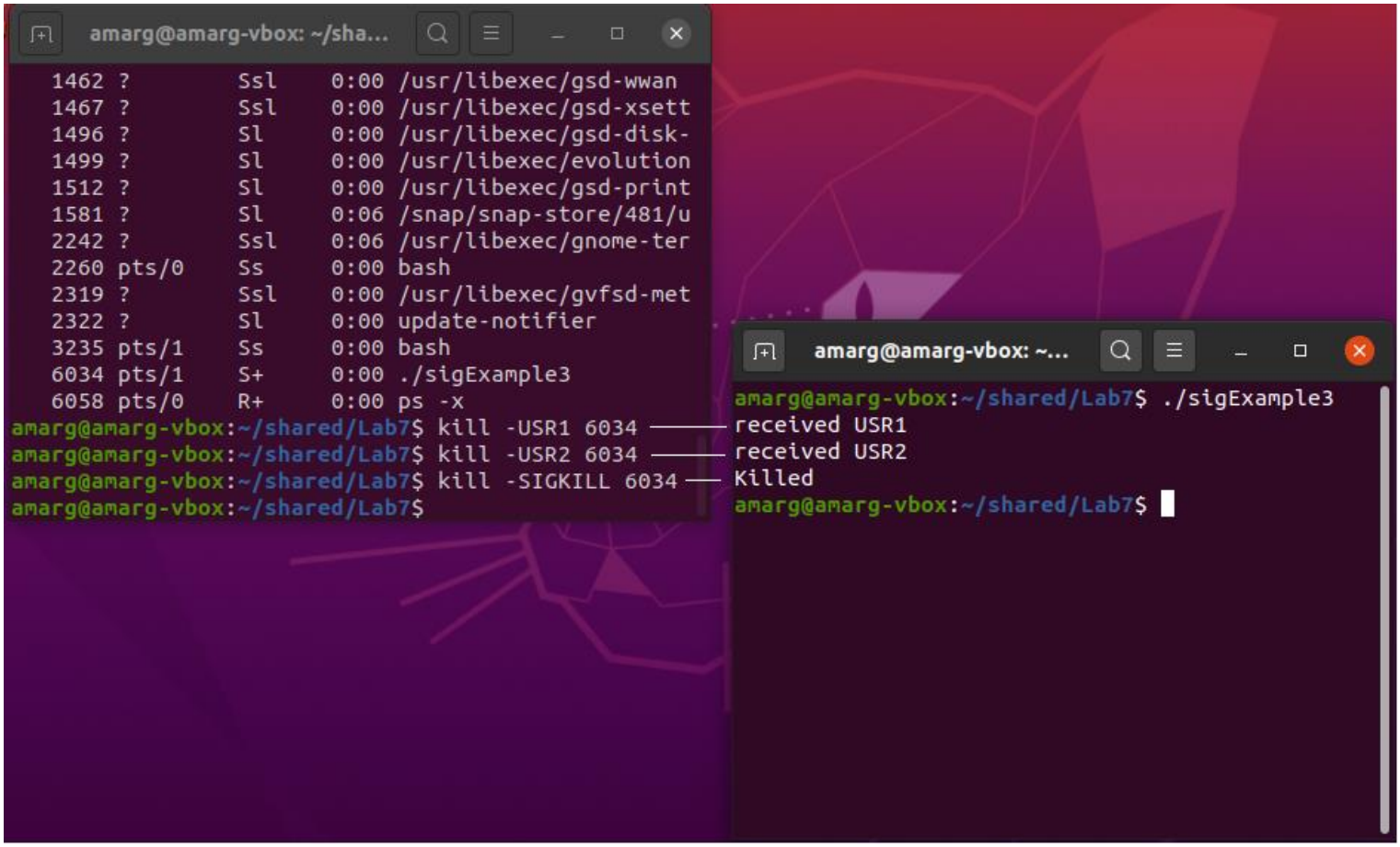
```
int main(void) {
   if (signal(SIGUSR1, sig_handler) == SIG_ERR)
       printf("\ncan't catch USR1\n");
   if (signal(SIGUSR2, sig_handler) == SIG_ERR)
       printf("\ncan't catch USR2\n");

  // A long long wait so that we can easily
  // issue a signal to this process
```

Όπως και προηγουμένως μετά τα παραπάνω εκτελούμε ένα βρόχο με άπειρες επαναλήψεις έτσι ώστε να διασφαλίσουμε ότι η διεργασία θα συλλάβει το σήμα που απέστειλε ο χρήστης

```
  while(1)
     sleep(1);
  return 0; }
```

Παράδειγμα εξόδου της εφαρμογής ακολουθεί στη συνέχεια.



Παράδειγμα 4. Ανταλλαγή σήματος μεταξύ γονικής και θυγατρικής διεργασίας – Παράδειγμα Α (αρχείο sigExample4.c).

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

Αυτό το απλό παράδειγμα περιλαμβάνει τη χρήση μιας γονικής και μία θυγατρικής διεργασίας. Η γονική διεργασία δημιουργεί τη θυγατρική διεργασία καλώντας τη συνάρτηση fork και στη συνέχεια της αποστέλλει διαδοχικά τα σήματα SIGHUP, SIGINT και SIGQUIT. Στην προκειμένη περίπτωση υπάρχουν τρεις διεργασίες χειρισμού, μία διεργασία για κάθε σήμα, οι οποίες απλά εκτυπώνουν ένα ενημερωτικό μήνυμα σύμφωνα με το οποίο παρέλαβαν το σήμα. Η αποστολή των σημάτων γίνεται ξανά με την kill, όχι όμως με την εντολή που καλείται από το τερματικό αλλά δια της κλήσεως της ομώνυμης συνάρτησης της γλώσσας C, η οποία δέχεται ως όρισμα τον κωδικό της διαδικασίας στην οποία θα αποτελεί ένα σήμα και τον κωδικό αυτού του σήματος. Τα υπόλοιπα είναι ακριβώς ίδια με πριν.

Ο χειριστής του σήματος SIGHUP

```
void sighup() { // handler for SIGHUP
    signal(SIGHUP, sighup);
    printf("CHILD: I have received a SIGHUP\n"); }
```

Ο χειριστής του σήματος SIGINT

```
void sigint() { // handler for SIGINT
    signal(SIGINT, sigint); /* reset signal */
    printf("CHILD: I have received a SIGINT\n"); }
```

Ο χειριστής του σήματος SIGQUIT

```
void sigquit() { // handler for SIGQUIT
   printf("My DADDY has Killed me!!!\n");
   exit(0); }

void main() {
    int pid;
```

Η συσχέτιση των συναρτήσεων χειρισμού με τα κατάλληλα σήματα

```
    signal(SIGHUP, sighup);
    signal(SIGINT, sigint);
    signal(SIGQUIT, sigquit);
```

Δημιουργία της θυγατρικής διεργασίας με τη fork

```
    pid = fork ();
    if (pid < 0) {
        perror("fork");
        exit(1); }
```

Η θυγατρική διεργασία εκτελεί απλά έναν ατέρμονα βρόχο χωρίς να κάνει απολύτως τίποτε, απλά και μόνο για να παραλάβει τα σήματα από τη γονική διεργασία. Ο βρόχος for (;;); είναι ατέρμων όπως και ο while (1) { … } χρησιμοποιήσαμε στα προηγούμενα παραδείγματα.

```
    if (pid == 0) { for (;;);  }
```

Η γονική διεργασία χρησιμοποιώντας τη συνάρτηση kill της C, αποστέλλει στη θυγατρικηή διαδικασία τα σήματα SIGHUP, SIGINT και SIGQUIT το ένα μετά το άλλο και με καθυστέρηση τριών δευτερολέπτων ανάμεσα στις διαδοχικές αποστολές - αυτό το κάνει καλώντας τη συνάρτηση sleep(3) - προκειμένου ο χρήστης να προλάβει να αντιληφθεί τη διαδικασία παραλαβής και λήψης των σημάτων . Το τελευταίο σήμα SIGQUIT, προκαλεί και τον τερματισμό της θυγατρικής διεργασίας.

```
    else {
      printf("\nPARENT: sending SIGHUP\n\n");
      kill(pid, SIGHUP);
      sleep(3); // pause for 3 secs
      printf("\nPARENT: sending SIGINT\n\n");
      kill(pid, SIGINT);
      sleep(3); // pause for 3 secs
      printf("\nPARENT: sending SIGQUIT\n\n");
      kill(pid, SIGQUIT);
      sleep(3); }}
```

Παράδειγμα εξόδου της εφαρμογής ακόλουθεί στη συνέχεια.

```
amarg@amarg-vbox:~/shared/Lab7$ ./sigExample4

PARENT: sending SIGHUP

CHILD: I have received a SIGHUP

PARENT: sending SIGINT

CHILD: I have received a SIGINT

PARENT: sending SIGQUIT

My DADDY has Killed me!!!
```

Παράδειγμα 5. Ανταλλαγή σήματος μεταξύ γονικής και θυγατρικής διεργασίας – Παράδειγμα Β (αρχείο sigExample5.c).

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

// SIGALARM = 14
// SIGCHLD =  17
```

Αυτό το παράδειγμα επιδεικνύει το γεγονός πως όταν τερματίζεται η λειτουργία μιας θυγατρικής διεργασίας η γονική διεργασία ενημερώνεται για τον τερματισμό της θυγατρικής διεργασίας λαμβάνοντας το ειδικό σήμα SIGCHILD. Πράγματι, στην εφαρμογή του παραδείγματος η θυγατρική διεργασία αποστέλλει στη γονική διεργασία μόνο το σήμα SIGALARM - ωστόσο η γονική διεργασία δεν λαμβάνει μόνο ένα, αλλά δύο σήματα, με το δεύτερο σήμα SIGCHLID που αποστέλλεται σε αυτή όταν η θυγατρική διεργασία τερματιστεί.

Ο χειριστής των σημάτων SIGALARM, SIGUSR1 και SIGCHLD (το SIGUSR δεν χρησιμοποιείται εδώ αλλά επειδή είναι σήμα γενικής χρήσεως θα μπορούσαμε εάν το επιθυμούσαμε να εκχωρήσουμε σε αυτό κάποια διαδικασία).

```
void signalHandler(int signal) {
    printf("Cought signal %d!\n",signal);
    if (signal==SIGCHLD) {
            printf("Child ended\n");
            wait(NULL); }}

int main() {
```

Η συσχέτιση της συνάρτησης χειρισμού με τα τρία σήματα.

```
        signal(SIGALRM,signalHandler);
        signal(SIGUSR1,signalHandler);
        signal(SIGCHLD,signalHandler);
```

Ο κώδικας της θυγατρικής διεργασίας. Παρατηρήστε πως η θυγατρική διεργασία στέλνει στη γονική διεργασία μόνο το σήμα SIGALARM και τίποτε άλλο.

```
        if (!fork()) {
            printf("Child running...\n");
            sleep(2);
            printf("Child sending SIGALRM...\n");
            kill (getppid(),SIGALRM);
            sleep(5);
            printf("Child exitting...\n");
            return 0; }
```

Ωστόσο, όταν η θυγατρική διεργασία ολοκληρωθεί, ο πυρήνας του λειτουργικού συστήματος στέλνει στη γονική διεργασία το σήμα SIGCHLD, προκειμένου να της γνωστοποιήσει τον τερματισμό της θυγατρικής διεργασίας. Αυτό δεν φαίνεται πουθενά στον κώδικα, διότι αποτελεί λειτουργία του πυρήνα και θα πραγματοποιηθεί σε κάθε περίπτωση, είτε το ζητήσει ο χρήστης είτε όχι. Το μόνο που έχουμε να κάνουμε (και μπορούμε να κάνουμε) είναι ένα προσδιορίσουμε τη συμπεριφορά της διεργασίας όταν λάβει αυτό το σήμα. Το μόνο που κάνουμε είναι απλά να εκτυπώσουμε στην οθόνη μας το μήνυμα Child ended.

```
        printf("Parent running, PID=%d. Press ENTER to exit.\n",getpid());
        getchar();
```

```
    printf("Parent exitting...\n");
    return 0; }
```

Παράδειγμα εξόδου της εφαρμογής ακολουθεί στη συνέχεια.

```
amarg@amarg-vbox:~$ ./sigExample5
Parent running, PID=1962. Press ENTER to exit.
Child running...
Child sending SIGALRM...
Cought signal 14!
Child exitting...
Cought signal 17!
Child ended

Parent exitting...
```

Παράδειγμα 6. Ανταλλαγή σήματος μεταξύ γονικής και πολλών θυγατρικών διεργασιών (αρχείο sigExample6.c).

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

Σε αυτό το παράδειγμα η γονική διεργασία δημιουργεί τέσσερις θυγατρικές διεργασίες και ανιχνεύει τον τερματισμό τους δια της χρήσεως του σήματος SIGCHILD.

```
#define NUMPROCS 4   /* number of processes to fork */
int nprocs;          /* number of child processes */
```

Η κάθε θυγατρική διεργασία που δημιουργείται, εκτελεί τη συνάρτηση chlild (n) όπου n ένας ακέραιος αριθμός που παίρνει μικρές θετικές τιμές και κάνει κάτι πάρα πολύ απλό (για λόγους επίδειξης και μόνο): εκτυπώνει το PID της καθώ και το PID του γονέα της, αναστέλλει τη λειτουργία της για n δευτερόλεπτα (καλώντας τη sleep) και στη συνέχεια καλεί τη συνάρτηση exit για να τερματιστεί (εκτυπώνοντας το κατάλληλο μήνυμα), με κωδικό επιστροφής 100+n (μπορείτε αν θέλετε να αλλάξετε αυτή τη συμπεριφορά και να καθορίσετε να κάνει κάτι διαφορετικό).

```
void child (int n) {
    printf("\tChild[%d]: child pid=%d, sleeping for %d seconds\n", n, getpid(), n);
    sleep(n);
    printf("\tchild[%d]: I'm exiting\n", n);
    exit(100+n); }
```

Η συνάρτηση catch αποτελεί τη συνάρτηση χειρισμού του σήματος SIGCHLD – αυτό δηλώνεται μέρα στη main. Αν και δέχεται ένα όρισμα snum, ωστόσο δεν το χρησιμοποιεί πουθενά (!). Η συνάρτηση catch καλεί τη wait προκειμένου μέσω της δομής status και δια της χρήσεως της μακροεντολής WEXITSTATUS να ανακτήσει τον κωδικό επιστροφής της θυγατρικής διεργασίας που ολοκληρώθηκε τελευταία, ενώ μειώνει το πλήθος των διεργασιών κατά μία μονάδα.

```
void catch (int snum) {
    int pid, status;
    pid = wait(&status);
    printf("Parent process: child process pid=%d exited with value %d\n",
            pid, WEXITSTATUS(status));
    nprocs--; }
```

```
int main (int argc, char **argv) {
    int pid, i;
```

Το πρώτο πράγμα που κάνουμε στη συνάρτηση main είναι να συσχετίσουμε την συνάρτηση catch με το σήμα SIGCHLD έτσι ώστε η εν λόγω συνάρτηση να χρησιμοποιηθεί ως η συνάρτηση χειρισμού αυτού του σήματος.

```
    signal(SIGCHLD, catch);
```

Η γονική διεργασία μέσα από ένα βρόχο που εκτελείται NUMPROCS φορές καλεί τη fork για να δημιουργήσει ισάριθμες θυγατρικές διεργασίες.

```
    for (i=0;i<NUMPROCS;i++) {
        pid=fork();
        if (pid<0) { perror("fork"); exit(1); }
```

Η κάθε θυγατρική διεργασία καλεί τη συνάρτηση child (i) όπου i η τρέχουσα τιμή του μετρητή του βρόχου.

```
        if (pid==0) child(i);
```

Η γονική διεργασία σε κάθε κύκλο επανάληψης αυξάνει την τιμή του nprocs κατά μία μονάδα (αφού σε κάθε κύκλο επανάληψης δημιουργείται και μία νέα διεργασία).

```
        else nprocs++; }
```

Για όσο χρονικό διάστημα υπάρχουν διεργασίες, η γονική διεργασία αναστέλλει τη λειτουργία της καλώντας τη συνάρτηση sleep. Καθώς οι θυγατρικές διεργασίες ολοκληρώνονται η μία μετά την άλλη και στέλνουν το σήμα SIGCHLD στη γονική διεργασία, αυτό διαβάζεται από τη συνάρτηση catch η οποία εκτυπώνει το μήνυμα τερματισμού της κάθε διεργασίας και μειώνει το πλήθος των διεργασιών κατά μία μονάδα. Όταν τερματιστούν όλες οι θυγατρικές διεργασίες (οπότε είναι nprocs = 0) ενεργοποιείται η γονική διεργασία η οποία τερματίζεται και αυτή.

```
    printf("Parent process: going to sleep\n");
    while (nprocs != 0) {
            printf("parent: sleeping\n");
            sleep(60); }
    printf("Parent process: exiting\n");
   exit(0); }
```

Παράδειγμα εξόδου της εφαρμογής ακολουθεί στη συνέχεια.

```
amarg@amarg-vbox:~$ ./sigExample6
Parent process: going to sleep
parent: sleeping
        Child[3]: child pid=1702, sleeping for 3 seconds
        Child[2]: child pid=1701, sleeping for 2 seconds
        Child[1]: child pid=1700, sleeping for 1 seconds
        Child[0]: child pid=1699, sleeping for 0 seconds
        child[0]: I'm exiting
Parent process: child process pid=1699 exited with value 100
parent: sleeping
        child[1]: I'm exiting
Parent process: child process pid=1700 exited with value 101
parent: sleeping
        child[2]: I'm exiting
Parent process: child process pid=1701 exited with value 102
parent: sleeping
        child[3]: I'm exiting
Parent process: child process pid=1702 exited with value 103
Parent process: exiting
```

Παράδειγμα 7. Παράδειγμα χρήσης των συναρτήσεων διαχείρισης σήματος – Παράδειγμα Α (αρχείο sigExample7.c).

Σε αυτό το παράδειγμα, η διεργασία μπλοκάρει το σήμα SIGTERM για δύο δευτερόλεπτα χρησιμοποιώντας τη συνάρτηση sigprocmask. Μετά την παρέλευση των δύο δευτερολέπτων το σήμα ξεμπλοκάρεται.

```
#include <signal.h>
#include <stdio.h>
#include <string.h>
#include <unistd.h>
```

Η μεταβλητή got_signal έχει τιμή 0 όταν δεν έχει παραληφθεί ένα σήμα και τιμή 1 στην αντίθετη περίπτωση.

```
static int got_signal = 0;
```

Η συνάρτηση χειρισμού σήματος απλά θέτει την καθολική μεταβλητή got_signal στην τιμή 1.

```
static void hdl (int sig) {
    got_signal = 1; }

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

Οι επόμενες δηλώσεις μεταβλητών και αρχικοποιήσεις σχετίζονται με την δομή act που περνάμε ως όρισμα στην sigaction.

```
    sigset_t mask;
    sigset_t orig_mask;
    struct sigaction act;
```

Η δομή act αρχικοποιείται στη μηδενική τιμή και στη συνέχεια το πεδίο handler τίθεται στη συνάρτηση hdl που απλά αρχικοποιεί η μεταβλητή got_signal στην τιμή 1 για να υποδηλώσει πως παρελήφθει ένα σήμα.

```
    memset (&act, 0, sizeof(act));
    act.sa_handler = hdl;
```

Η συνάρτηση sigaction που έχει αντικαταστήσει την παρωχημένη συνάρτηση signal συσχετίζει το σήμα SIGTERM με τη δομή act και έμμεσα με τη συνάρτηση χειρισμού hdl.

```
    if (sigaction(SIGTERM, &act, 0)) {
       perror ("sigaction");
      return 1; }
```

Προκειμένου να αρχικοποιήσουμε το σύνολο σημάτων mask έτσι ώστε να περιέχει μόνο το σήμα SIGTERM που θεωρούμε σε αυτό το παράδειγμα, χρησιμοποιούμε τον πρώτο τρόπο αρχικοποίησης, δηλαδή, εκκενώνουμε τη μάσκα από όλα τα σήματα καλώντας τη συνάρτηση sigemptyset και μετά προσθέτουμε σε αυτή το σήμα SIGTERM με τη συνάρτηση sigaddset.

```
    printf ("Emptying signal set...\n");
    sigemptyset (&mask);
    printf ("Adding SIGTERM to signal set...\n");
    sigaddset (&mask, SIGTERM);

   // SIGTERM signal is added to orig_mask
   printf ("Adding signal set to the original mask...\n");
```

Έχοντας αρχικοποιήσει το σύνολο σημάτων mask έτσι ώτε να περιέχει μόνο το σήμα SIGTERM, καλούμε τη συνάρτηση sigpropmask έτσι ώστε να προσθέσουμε το σύνολο σημάτων της mask (δηλαδή ουσιαστικά το σήμα SIGTERM) στην αρχική μάσκα orig_mask.

Θυμηθείτε πως ο σκοπός της άσκησης είναι να μπλοκάρουμε (δείτε το όρισμα SIG_BLOCK) το σήμα SIGTERM. Η νέα μάσκα που προκύπτει από την κλήση της sigprocmask προκύπτει από την ένωση της αρχικής μάσκας και του συνόλου mask που περέχει τα σήματα (στην προκειμένη περίπτωση το σήμα SIGTERM) που επιθυμούμε να μπλοκάρουμε.

```
if  (sigprocmask(SIG_BLOCK, &mask, &orig_mask) < 0) {
     perror ("sigprocmask");
     return 1; }
```

Στη συνέχεια καλούμε ξανά την sigpropmask αλλά αυτή τη φορά με όρισμα SIG_SETMASK έτσι ώστε να ορίσουμε τη νέα μάσκα ως την τρέχουσα μάσκα, διαδικασία που θα οδηγήσει και στο μπλοκάρισμα του σήματος.

```
// SIGTERM signal is blocked
printf ("Blocking SIGTERM signal...\n");
if  (sigprocmask(SIG_SETMASK, &orig_mask, NULL) < 0) {
     perror ("sigprocmask");
     return 1; }
```

Μετά την παρέλευση δύο δευτερολέπτων (η αναστολή γίνεται με τη sleep) το σήμα ξεμπλοκάρεται.

```
printf ("Sleeping for 2 seconds\n");
sleep (2);
if (got_signal) puts ("Got signal");
return 0; }
```

Παράδειγμα εξόδου της εφαρμογής ακολουθεί στη συνέχεια.

```
amarg@amarg-vbox:~$ ./sigExample7
Emptying signal set...
Adding SIGTERM to signal set...
Adding signal set to the original mask...
Blocking SIGTERM signal...
Sleeping for 2 seconds
amarg@amarg-vbox:~$ gedit sigExample7.c
```

Παράδειγμα 8. Παράδειγμα χρήσης των συναρτήσεων διαχείρισης σήματος – Παράδειγμα Β (αρχείο sigExample8.c).

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

Εδώ χρησιμοποιούνται τα σήματα SIGUSR1 (σήμα γενικής χρήσεως), SIGHUP (στη γενική περίπτωση ενημερώνει τη διεργασία πως το τερματικό που την παρακολουθεί έχει αποσυνδεθεί) και SIGINT (που προκαλεί τη διακοπή της)

void handle_signal(int signal); **Η συνάρτηση χειρισμού των τριών σημάτων**

void handle_sigalrm (int signal); **Ενημερώνει το χρήστη ότι παρέλαβε το σήμα SIGALARM**

void do_sleep(int seconds); **Η βασική συνάρτηση της εφαρμογής**

Στο κυρίως πρόγραμμα αρχικοποιούμε τη δομή sa την οποία σχετίζουμε με τα τρία σήματα ενδιαφέροντος και στη συνέχεια μέσα από έναν ατερόνα βρόχο καλούμε τη συνάρτηση do_sleep.

```
int main() {
   struct sigaction sa;
   printf("My pid is: %d\n", getpid());  // we need the pid to use kill from another terminal
   sa.sa_handler = &handle_signal;   // here we assign the signal handler
```

Η σημαία SA_RESTART ελέγχει το τι ακριβώς συμβαίνει όταν ένα σήμα αποσταλεί και η συνάρτηση χειρισμού του ολοκληρωθεί χωρίς πρόβλημα. Υπάρχουν δύο εναλλακτικές που μπορούν να εμφανιστούν: είτε η βιβλιοθήκη που κάλεσε τη συνάρτηση να συνεχίσει να λειτουργεί, είτε να επιστρέψει κωδικό σφάλματος. Εάν χρησιμοποιηθεί αυτή η σημαία τότε η επιστροφή από το χειριστή του σήματος επιτρέπει τη συνέχιση της διαδικασίας.

```
sa.sa_flags = SA_RESTART;
```

Κατά τη διάρκεια της εκτέλεσης της συνάρτησης χειρισμού μπλοκάρουμε όλα τα σήματα,

```
// Block every signal during the handler
sigfillset(&sa.sa_mask);
```

Εδώ συσχετίζουμε τα τρία σήματα με τη δομή sa που αρχικοποιήθηκε προηγουμένως.

```
// Signals SIGHUP, SIGUSR1 and SIGINT are associated with struct sa

if (sigaction(SIGHUP, &sa, NULL) == -1) { perror("Error: cannot handle SIGHUP"); }
if (sigaction(SIGUSR1, &sa, NULL) == -1) { perror("Error: cannot handle SIGUSR1"); }
if (sigaction(SIGINT, &sa, NULL) == -1) { perror("Error: cannot handle SIGINT"); }
```

Το σήμα SIGKILL δεν μπορεί να δηλωθεί εδώ αφού τερματίζει τη διεργασία.

```
// Will always fail, SIGKILL is intended to force kill your process
if (sigaction(SIGKILL, &sa, NULL) == -1) { perror("Cannot handle SIGKILL"); }
```

Το πρόγραμμα μπαίνει σε ένα infinite loop καλώντας επαναληπτικά τη συνάρτηση do_sleep.

```
for (;;) {
    printf("\nSleeping for ~3 seconds\n");
    do_sleep(3); }}
```

Η συνάρτηση χειρισμού των τριών σημάτων αρχικοποιεί το όνομα του σήματος που παραλαμβάνεται κάθε φορά. Εάν το σήμα αυτό είναι το SIGINT εκτυπώνει ένα ενημερωτικό μήνυμα πως η διεργασία θα τερματιστεί ενώ αν παραλάβει κάτι μη αναμενόμενο εκτυπώνει μήνυμα σφάλματος στο αρχείο stderr.

```
void handle_signal (int signal) {
  const char *signal_name;
  sigset_t pending;
  // Find out which signal we're handling
  switch (signal) {
    case SIGHUP:
      signal_name = "SIGHUP";
      break;
    case SIGUSR1:
      signal_name = "SIGUSR1";
      break;
    case SIGINT:
      printf("Caught SIGINT, exiting now\n");
      exit(0);
    default:
      fprintf(stderr, "Caught wrong signal: %d\n", signal);
      return; }
```

Μετά την παραλαβή του σήματος η συνάρτηση ζητά από το χρήστη να στείλει ένα άλλο σήμα και τον ενημερώνει πως θα αναστείλει τη λειτουργία της για τρία δευτερόλεπτα.

```
// However, printf in signal handlers IS NOT reccommented !
printf("Caught %s, sleeping for ~3 seconds\n"
    "Try sending another SIGHUP / SIGINT / SIGALRM "
    "(or more) meanwhile\n", signal_name);
do_sleep(3);
printf("Done sleeping for %s\n", signal_name);
```

Εάν σε αυτό το χρονικό διάστημα των τριών δευτερολέπτων ο χρήστης στείλει νέα σήματα στην εργασία αυτά θα μπούνε στη λίστα των εκκρεμών σημάτων προκειμένου η εργασία να τα παραλάβει και να ανταποκριθεί ανάλογα όταν επανέλθει σε λειτουργία. Η ανάκτηση της λίστας αυτών των εκκρεμών σημάτων γίνεται με τη συνάρτηση sigpending η οποία επιστρέφει την εν λόγω πληροφορία στη μεταβλητή pending που είναι τύπου sigset_t.

```
// So what did you send me while I was asleep?
sigpending(&pending);
```

Εάν σε αυτή τη λίστα υπάρχουν τα σήματα SIGHUP και SIGUSR1 η εφαρμογή εκτυπώνει το κατάλληλο ενημερωτικό μήνυμα.

```
if (sigismember(&pending, SIGHUP)) { printf("A SIGHUP is waiting\n"); }
if (sigismember(&pending, SIGUSR1)) { printf("A SIGUSR1 is waiting\n"); }
printf("Done handling %s\n\n", signal_name); }
```

Η συνάρτηση handle_sigalarm εκτυπώνει απλά ένα ενημερωτικό μήνυμα σχετικά με το σήμα που παρέλαβε. Το σήμα που αναμένει να λάβει είναι το SIGALARM που δημιουργείται από την alarm, διαφορετικά εκτυπώνει μήνυμα σφάλματος.

```
void handle_sigalrm(int signal) {
    if (signal != SIGALRM) { fprintf(stderr, "Caught wrong signal: %d\n", signal); }
    printf("Got sigalrm, do_sleep() will end\n"); }
```

```
void do_sleep(int seconds) {
```

Η συνάρτηση do_sleep συσχετίζει το SIGALARM με τη δομή sa που αρχικοποιείται κατάλληλα.

```
    struct sigaction sa;
    sigset_t mask;

    sa.sa_handler = &handle_sigalrm;
```

***SA_RESETHAND* == > Restore the signal action to the default upon entry to the signal handler. This flag is meaningful only when establishing a signal handler.**

```
    sa.sa_flags = SA_RESETHAND;
    sigfillset(&sa.sa_mask);

    sigaction(SIGALRM, &sa, NULL);
```

Ανακτάται η τρέχουσα μάσκα,

```
    // Get the current signal mask
    sigprocmask(0, NULL, &mask);
```

αφαιρείται από αυτή το SIGALARM

```
    // Unblock SIGALRM
    sigdelset(&mask, SIGALRM);
```

και στη συνέχεια στέλνεται αυτό σήμα ύστερα από seconds δευτερόλεπτα. Αυτό γίνεται με τη συνάρτηση alarm.

Η συνάρτηση alarm() δέχεται ως όρισμα έναν ακέραιο αριθμό που εκφράζει ένα χρονικό διάστημα εκφρασμένο σε δευτερόλεπτα και μετά την παρέλευση αυτού του χρονικού διαστήματος δημιουργεί ένα σήμα SIGALARM η προεπιλεγμένη συμπεριφορά του οποίου είναι ο τερματισμός της διεργασίας (ωστόσο, μπορούμε χρησιμοποιώντας την κατάλληλη συνάρτηση χειρισμού, να τροποποιήσουμε αυτή τη συμπεριφορά ανάλογα με τις απαιτήσεις μας).

```
alarm(seconds);
sigsuspend(&mask);
printf("sigsuspend() returned\n"); }
```

Παράδειγμα εξόδου της εφαρμογής ακολουθεί στη συνέχεια.