ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ Η/Υ

ΑΣΚΗΣΗ 2

ΟΜΑΔΑ OSLABC34:

Χριστοδουλόπουλος Δανιήλ 03114797

Θηβαίος Αναστάσιος 03114785

Άσκηση 1:

ex_1.1.c

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

#include "proc-common.h"

#define SLEEP_PROC_SEC  10
#define SLEEP_TREE_SEC  3

/*
 * Create this process tree:
 * A-+-B---D
 *   `-C
 */
void fork_procs(void)
{
        /*
         * initial process is A.
         */
    pid_t pB, pC, pD;
    int status;

        change_pname("A");
    printf("A: Starting...\n");
    fprintf(stderr, "Parent, PID = %ld: Creating child...\n", (long)getpid());
    pC = fork();

    if ( pC < 0) {
        perror("fork");
        exit(1);
    }

    if ( pC == 0 ) {
        change_pname("C");
        printf("C: Starting...\n");
        printf("C: Sleeping...\n");
        sleep(SLEEP_PROC_SEC);
        printf("C: Exiting...\n");
        exit(17);
    }
```

Συνέχεια….

```
    fprintf(stderr, "Parent, PID = %ld: Creating child...\n", (long)getpid());
    pB = fork();

    if ( pB < 0 ) {
        perror("fork");
        exit(1);
    }

    if ( pB == 0) {
        change_pname("B");
        printf("B: Starting...\n");
        fprintf(stderr, "Parent, PID = %ld: Creating child...\n", (long)getpid());
        pD = fork();

        if ( pD < 0) {
            perror("fork");
            exit(1);
        }

        if ( pD == 0) {
            change_pname("D");
            printf("D: Starting...\n");
            printf("D: Sleeping...\n");
            sleep(SLEEP_PROC_SEC);
            printf("D: Exiting...\n");
            exit(13);
        }
        printf("B: Sleeping...\n");
        sleep(SLEEP_PROC_SEC);
        pD  = wait(&status);
        explain_wait_status(pD , status);

        printf("B: Exiting...\n");
        exit(19);

    }

            printf("A: Sleeping...\n");
            sleep(SLEEP_PROC_SEC);
    pB  = wait(&status);
            explain_wait_status(pB , status);

    pC  = wait(&status);
            explain_wait_status(pC, status);

    printf("A: Exiting...\n");
            exit(16);
}
```

Συνέχεια…

```
int main(void) {

        pid_t pid;
        int status;

        /* Fork root of process tree */
        pid = fork();
        if (pid < 0) {
        perror("main: fork");
        exit(1);
        }
        if (pid == 0) {
        /* Child */
        fork_procs();
        exit(1);
        }

        /*
         * Father
         */

        /* for ask2-{fork, tree} */
        sleep(SLEEP_TREE_SEC);

        /* Print the process tree root at pid */
        show_pstree(pid);


        /* Wait for the root of the process tree to terminate */
        pid = wait(&status);
        explain_wait_status(pid, status);

        return 0;
}
```

Η έξοδος της εκτέλεσης της 1ης άσκησης:

```
.../Ergastirio/Askisi 2/ex 1.1 ./ex_1.1
A: Starting...
Parent, PID = 10607: Creating child...
Parent, PID = 10607: Creating child...
C: Starting...
C: Sleeping...
A: Sleeping...
B: Starting...
Parent, PID = 10609: Creating child...
B: Sleeping...
D: Starting...
D: Sleeping...


A(10607)─┬─B(10609)──D(10610)
         └─C(10608)


C: Exiting...
My PID = 10607: Child PID = 10608 terminated normally, exit status = 17
D: Exiting...
My PID = 10609: Child PID = 10610 terminated normally, exit status = 13
B: Exiting...
My PID = 10607: Child PID = 10609 terminated normally, exit status = 19
A: Exiting...
My PID = 10606: Child PID = 10607 terminated normally, exit status = 16
.../Ergastirio/Askisi 2/ex 1.1
```

Απαντήσεις στις ερωτήσεις για την 1η άσκηση:

1. Αυτό που θα γίνει αν τερματίσουμε πρόωρα την διεργασία Α είναι ότι η διεργασία Α θα τερματίσει, και ο πατέρας του Α (ex_1.1) θα λάβει το σήμα ότι τερματίστηκε το παιδί του (Α) με σήμα signo=9. Επίσης τα παιδία του Α θα μείνουν "ορφανά" οπότε θα γίνουν παιδία της διεργασίας init (systemd).

2. Εάν κάνουμε στην main show_pstree(getpid()) αντί για show_pstree(pid) θα δούμε κάποιες επιπλέον διεργασίες. Η πρώτη είναι το πρόγραμμα που τρέχουμε, δηλαδή ex_1.1, που εκτός απο το Α σαν παιδί θα έχει το sh. Το sh είναι το dash (shell) και δημιουργείται διότι χρησιμοποιούμε την εντολή pstree μέσω της συνάρτησης show_pstree όπου είναι το παιδί της. Τέλος αυτά εμφανίζονται διότι το getpid() επιστρέφει το process ID της καλούσας διεργασίας δηλαδή της ex_1.1.

3. Ο διαχειριστής το κάνει αυτό διότι ένας χρήστης αν μπορέι να δημιούργει όσες διεργασίες θέλει ενδεχομένως το Λειτουργικό Σύστημα να μην μπορεί να διαχειριστεί

τον τεράστιο αυτό όγκο. Οπότε θέτει όρια έτσι ώστε να είναι μπορεί πάντα να διαχειρίζεται όλες τις διεργασίες απ' όλους τους χρήστες.

Άσκηση 2:

ex_1.2.c

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

#include "tree.h"
#include "proc-common.h"

#define SLEEP_PROC_SEC 10
#define SLEEP_TREE_SEC 3

void fork_procs(struct tree_node *root)
{
    /*
     * Start
     */
    pid_t pid[root->nr_children];
    int i, status;

    printf("PID = %ld, name %s, starting...\n", (long)getpid(), root->name);
    change_pname(root->name);

    for (i=0; i < (root->nr_children); i++) {
        pid[i] = fork();

        if (pid[i] < 0){
            perror("fork_procs: fork");
            exit(1);
        }
        if (pid[i] == 0) {
            /* Child */
            fork_procs((root->children)+i);
            exit(1);
        }
    }
```

Συνέχεια...

```
    if (root->nr_children != 0){
            sleep(SLEEP_TREE_SEC);
        for(i=0; i < root->nr_children; i++){
            int tmpPID =  wait(&status);
            explain_wait_status(tmpPID,status);
        }
    }
    else sleep(SLEEP_PROC_SEC);

    printf("PID = %ld, name %s, exiting...\n", (long)getpid(), root->name);
    /*
     * Exit
     */
    exit(0);
}

int main(int argc, char *argv[]) {
    pid_t pid;
    int status;
    struct tree_node *root;

    if (argc < 2){
        fprintf(stderr, "Usage: %s <tree_file>\n", argv[0]);
        exit(1);
    }

    /* Read tree into memory */
    root = get_tree_from_file(argv[1]);
    // print_tree(root);

    /* Fork root of process tree */
    pid = fork();
    if (pid < 0) {
        perror("main: fork");
        exit(1);
    }
    if (pid == 0) {
        /* Child */
        fork_procs(root);
        exit(1);
    }
```

Συνέχεια...

```
    /*
     * Father
     */

    /* for ask2-{fork, tree} */
    sleep(SLEEP_TREE_SEC);

    /* Print the process tree root at pid */
    show_pstree(pid);


    /* Wait for the root of the process tree to terminate */
    wait(&status);
    explain_wait_status(pid, status);

    return 0;
}
```

Οι έξοδοι της εκτέλεσης της 2ης άσκησης:

```
.../Ergastirio/Askisi 2/ex_1.2 ./ex_1.2 proc.tree
PID = 10724, name A, starting...
PID = 10725, name B, starting...
PID = 10726, name C, starting...
PID = 10727, name D, starting...
PID = 10728, name E, starting...
PID = 10729, name F, starting...


A(10724)─┬─B(10725)─┬─E(10728)
         │          └─F(10729)
         ├─C(10726)
         └─D(10727)


PID = 10726, name C, exiting...
PID = 10727, name D, exiting...
PID = 10729, name F, exiting...
PID = 10728, name E, exiting...
My PID = 10724: Child PID = 10726 terminated normally, exit status = 0
My PID = 10724: Child PID = 10727 terminated normally, exit status = 0
My PID = 10725: Child PID = 10728 terminated normally, exit status = 0
My PID = 10725: Child PID = 10729 terminated normally, exit status = 0
PID = 10725, name B, exiting...
My PID = 10724: Child PID = 10725 terminated normally, exit status = 0
PID = 10724, name A, exiting...
My PID = 10723: Child PID = 10724 terminated normally, exit status = 0
.../Ergastirio/Askisi 2/ex_1.2
```

```
.../Ergastirio/Askisi 2/ex 1.2  ./ex_1.2 in1.tree
PID = 10769, name *, starting...
PID = 10770, name +, starting...
PID = 10771, name *, starting...
PID = 10772, name *, starting...
PID = 10773, name 3, starting...
PID = 10774, name 4, starting...
PID = 10775, name 2, starting...
PID = 10776, name 2, starting...
PID = 10777, name 6, starting...


*(10769)─┬─*(10771)─┬─2(10775)
         │          └─3(10773)
         └─+(10770)─┬─*(10772)─┬─2(10776)
                    │          └─6(10777)
                    └─4(10774)


PID = 10773, name 3, exiting...
PID = 10774, name 4, exiting...
PID = 10775, name 2, exiting...
PID = 10777, name 6, exiting...
My PID = 10771: Child PID = 10773 terminated normally, exit status = 0
My PID = 10770: Child PID = 10774 terminated normally, exit status = 0
PID = 10776, name 2, exiting...
My PID = 10772: Child PID = 10777 terminated normally, exit status = 0
My PID = 10771: Child PID = 10775 terminated normally, exit status = 0
PID = 10771, name *, exiting...
My PID = 10772: Child PID = 10776 terminated normally, exit status = 0
PID = 10772, name *, exiting...
My PID = 10769: Child PID = 10771 terminated normally, exit status = 0
My PID = 10770: Child PID = 10772 terminated normally, exit status = 0
PID = 10770, name +, exiting...
My PID = 10769: Child PID = 10770 terminated normally, exit status = 0
PID = 10769, name *, exiting...
My PID = 10768: Child PID = 10769 terminated normally, exit status = 0
.../Ergastirio/Askisi 2/ex 1.2
```

Απαντήσεις στις ερωτήσεις για την 2η άσκηση:

1. Τα μηνύματα έναρξης των διεργασιών εμφανίζονται αν επίπεδο το δέντρου, δηλαδή δημιούργειται πρώτα το 1ο επίπεδο (A), μετά τα παιδία του A που βρίσκονται στο ίδιο επίπεδο και έπειτα το τελευταίο επίπεδο. Τα μηνύματα τερματισμού είναι ανάλογα με το ποια διεργασία κοιμήθηκε πρώτη και ξύπνησε επομένως πρώτη, εκτός αν έχει παιδία. Αν έχει παιδία θα πρέπει να τα δημιουργήσει και έπειτα να περιμένει το τερματισμό τους. Έτσι όταν τερματίσουν ολα τα παιδία της, θα μπορεί να τερματίσει και αυτή. Γι' αυτό βλέπουμε ότι δημιουργείται πρώτη η διεργασία A και τερματίζει τελευταία.

Άσκηση 3:

ex_1.3.c

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

#include "tree.h"
#include "proc-common.h"

#define SLEEP_PROC_TIME 10
#define SLEEP_TREE_TIME 3

void fork_procs(struct tree_node *root){
    /*
     * Start
     */
    pid_t pid[root->nr_children];
    int i;
    int status;
    printf("PID = %ld, name %s, starting...\n", (long)getpid(), root->name);
    change_pname(root->name);

    for (i=0; i < (root->nr_children); i++) {
        pid[i] = fork();

        if (pid[i] < 0){
            perror("fork_procs: fork");
            exit(1);
        }
        if (pid[i] == 0) {
            /* Child */
            fork_procs((root->children)+i);
            exit(1);
        }
    }
```

Συνέχεια....

```
    if (root->nr_children !=0) {
        wait_for_ready_children(root->nr_children);
        raise(SIGSTOP);
        printf("PID = %ld, name = %s is awake\n", (long)getpid(), root->name);
        for(i=0; i<root->nr_children; i++){
            kill(pid[i],SIGCONT);
            wait(&status);
            explain_wait_status(pid[i],status);
        }
    }
    else {
        sleep(SLEEP_TREE_TIME);
        raise(SIGSTOP);
        printf("PID = %ld, name = %s is awake\n", (long)getpid(), root->name);
    }

    exit(0);
}

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

    pid_t pid;
    int status;
    struct tree_node *root;

    if (argc < 2){
        fprintf(stderr, "Usage: %s <tree_file>\n", argv[0]);
        exit(1);
    }

    /* Read tree into memory */
    root = get_tree_from_file(argv[1]);

    /* Fork root of process tree */
    pid = fork();
    if (pid < 0) {
        perror("main: fork");
        exit(1);
    }
    if (pid == 0) {
        /* Child */
        fork_procs(root);
        exit(1);
    }
```

Συνέχεια...

```
  /*
   * Father
   */

  /* for ask2-signals */
  wait_for_ready_children(1);


  /* Print the process tree root at pid */
  show_pstree(pid);

  /* for ask2-signals */
  kill(pid, SIGCONT);

  /* Wait for the root of the process tree to terminate */
  wait(&status);
  explain_wait_status(pid, status);

  return 0;
}
```

Οι έξοδοι της εκτέλεσης της 3ης άσκησης:

```
.../Ergastirio/Askisi 2/ex_1.3 ./ex_1.3 proc.tree
PID = 10831, name A, starting...
PID = 10832, name B, starting...
PID = 10833, name C, starting...
PID = 10834, name D, starting...
PID = 10835, name E, starting...
PID = 10836, name F, starting...
My PID = 10831: Child PID = 10833 has been stopped by a signal, signo = 19
My PID = 10831: Child PID = 10834 has been stopped by a signal, signo = 19
My PID = 10832: Child PID = 10835 has been stopped by a signal, signo = 19
My PID = 10832: Child PID = 10836 has been stopped by a signal, signo = 19
My PID = 10831: Child PID = 10832 has been stopped by a signal, signo = 19
My PID = 10830: Child PID = 10831 has been stopped by a signal, signo = 19


A(10831)─┬─B(10832)─┬─E(10835)
         │          └─F(10836)
         ├─C(10833)
         └─D(10834)


PID = 10831, name = A is awake
PID = 10832, name = B is awake
PID = 10835, name = E is awake
My PID = 10832: Child PID = 10835 terminated normally, exit status = 0
PID = 10836, name = F is awake
My PID = 10832: Child PID = 10836 terminated normally, exit status = 0
My PID = 10831: Child PID = 10832 terminated normally, exit status = 0
PID = 10833, name = C is awake
My PID = 10831: Child PID = 10833 terminated normally, exit status = 0
PID = 10834, name = D is awake
My PID = 10831: Child PID = 10834 terminated normally, exit status = 0
My PID = 10830: Child PID = 10831 terminated normally, exit status = 0
.../Ergastirio/Askisi 2/ex_1.3
```

```
.../Ergastirio/Askisi_2/ex_1.3  ./ex_1.3 expr.tree
PID = 10961, name +, starting...
PID = 10962, name 10, starting...
PID = 10963, name *, starting...
PID = 10964, name +, starting...
PID = 10965, name 4, starting...
PID = 10966, name 5, starting...
PID = 10967, name 7, starting...
My PID = 10961: Child PID = 10962 has been stopped by a signal, signo = 19
My PID = 10963: Child PID = 10965 has been stopped by a signal, signo = 19
My PID = 10964: Child PID = 10966 has been stopped by a signal, signo = 19
My PID = 10964: Child PID = 10967 has been stopped by a signal, signo = 19
My PID = 10963: Child PID = 10964 has been stopped by a signal, signo = 19
My PID = 10961: Child PID = 10963 has been stopped by a signal, signo = 19
My PID = 10960: Child PID = 10961 has been stopped by a signal, signo = 19


+(10961)─┬─*(10963)─┬─+(10964)─┬─5(10966)
         │          │          └─7(10967)
         │          └─4(10965)
         └─10(10962)


PID = 10961, name = + is awake
PID = 10962, name = 10 is awake
My PID = 10961: Child PID = 10962 terminated normally, exit status = 0
PID = 10963, name = * is awake
PID = 10964, name = + is awake
PID = 10966, name = 5 is awake
My PID = 10964: Child PID = 10966 terminated normally, exit status = 0
PID = 10967, name = 7 is awake
My PID = 10964: Child PID = 10967 terminated normally, exit status = 0
My PID = 10963: Child PID = 10964 terminated normally, exit status = 0
PID = 10965, name = 4 is awake
My PID = 10963: Child PID = 10965 terminated normally, exit status = 0
My PID = 10961: Child PID = 10963 terminated normally, exit status = 0
My PID = 10960: Child PID = 10961 terminated normally, exit status = 0
.../Ergastirio/Askisi_2/ex_1.3
```

<u>Απαντήσεις στις ερωτήσεις για την 3η άσκηση:</u>

1. Το πλεονέκτημα των σημάτων για τον συγχρονισμό των διεργασιών είναι ότι μπορούμε να έχουμε τον έλεγχο των διεργασιών εμείς και δεν βασιζόμαστε την "τύχη". Μπορούμε λοιπόν να ρυθμίζουμε πότε θα ξεκινάει μια διεργασία ή πότε θα τερματίζει ανεξάρτητα από χρόνους.

2. Ο ρόλος της wait_for_ready_children() είναι να περιμένει για όσες διεργασίες-παιδία του πούμε να κάνουν raise(SIGSTOP) (δεν μας ενδιαφέρει πιο παιδί θα σταματήσει πρώτο). Η χρήση της μας εξασφαλίζει ότι ο πατέρας δεν θα σταματήσει την λειτουργία του πριν τα παιδία σταματήσουν πρώτα. Το πρόβλημα που θα δημιουργούσε εάν την παραλείπαμε θα ήταν ότι δεν θα ήμασταν σίγουροι ότι όλα τα παιδία μας σταμάτησαν οπότε δεν θα μπορούσαμε να εμφανίσουμε τα μηνύματα awake κατα βάθος.

Άσκηση 4:

ex_1.4.c

```c
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <string.h>
#include "tree.h"
#include "proc-common.h"

#define SLEEP_PROC_SEC 10
#define SLEEP_TREE_SEC 3

void fork_procs(struct tree_node *root, int fd){

    pid_t pid[root->nr_children];
    int i;
    int pfd[2];
    char name[NODE_NAME_SIZE];
    int res = 0;

    printf("PID = %ld, name %s, starting...\n", (long)getpid(), root->name);
    change_pname(root->name);

    printf("Parent: Creating pipe...\n");
    if (pipe(pfd) < 0) {
        perror("pipe");
        exit(1);
    }
    for (i=0; i < (root->nr_children); i++) {


        pid[i] = fork();

        printf("Parent: Creating child...\n");
        if (pid[i] < 0){
            perror("fork_procs: fork");
            exit(1);
        }
        if (pid[i] == 0) {
            /* Child */
            fork_procs(((root->children)+i),pfd[1]);
            assert(0);
        }
    }
```

Συνέχεια...

```
if (root->nr_children != 0){

    /* Code of Parent*/

    /* Need to close the write pipe Parent to his Child */

    close(pfd[1]);

    sleep(SLEEP_TREE_SEC);
    int flag=0;

    /* Check if compute addition(0) or
     * multiplication(1)
     */

    if(!strcmp(root->name,"*")) {
        flag=1;
        res=1;
    }

    /* Parent starts to read from pipe*/

    for (i=0; i<root->nr_children; i++){

        if (read(pfd[0], &name, sizeof(name)) < 0) {
            perror("parent: read from pipe");
            exit(1);
        }
        if (flag) res = res*atoi(name);
        else res = res + atoi(name);

    }

    char tmp[NODE_NAME_SIZE];
    sprintf(tmp,"%d",res);

    /* Parent writes to his parent */

    if (write(fd, &tmp, sizeof(tmp)) != sizeof(tmp)) {
        perror("parent: write to pipe fork_procs");
        exit(1);
    }

    /*Finish read. Close read pipe*/
    close(pfd[0]);
}
```

Συνέχεια...

```
    else {

        /* Code of node without childrens
         * Need to close the pipe of read
         */

        close(pfd[0]);

        sleep(SLEEP_PROC_SEC);

        /*The child writes to his Parent */

        if (write(fd, &root->name, sizeof(root->name)) != sizeof(root->name)) {
            perror("parent: write to pipe fork_procs");
            exit(1);
        }
    }

    /*Finish write. Close write pipe*/
    close(fd);

    /*
     * Exit
     */
    exit(0);
}


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

    pid_t p;
    int pfd[2], status, final;
    char result[16];
    struct tree_node *root;


    if (argc < 2){
        fprintf(stderr, "Usage: %s <tree_file>\n", argv[0]);
        exit(1);
    }

    /* Read tree into memory */
    root = get_tree_from_file(argv[1]);

    printf("Parent: Creating pipe...\n");
    if (pipe(pfd) < 0) {
        perror("pipe");
        exit(1);
    }
```

Συνέχεια...

```
    printf("Parent: Creating child...\n");
    p = fork();
    if (p < 0) {
        /* fork failed */
        perror("fork");
        exit(1);
    }
    if (p == 0) {
        /* In child process */
        fork_procs(root,pfd[1]);
        /*
         * Should never reach this point,
         * child() does not return
         */
        assert(0);
    }

    sleep(SLEEP_TREE_SEC);

    /*Close write pipe. Don't need.*/
    close(pfd[1]);

    printf("Parent: Created child with PID = %ld, waiting for it to terminate...\n", (long)p);

    show_pstree(p);

    if (read(pfd[0], &result, sizeof(result)) < 0) {
        perror("parent: write to pipe");
        exit(1);
    }

    p = wait(&status);
    explain_wait_status(p, status);

    final = atoi(result);

    printf("Parent: All done, exiting...\n");
    printf("Result = %d\n", final);

    /* Close read pipe */
    close(pfd[0]);

    return 0;
}
```

Οι έξοδοι της εκτέλεσης της 4ης άσκησης:

```
.../Ergastirio/Askisi 2/ex_1.4  ./ex_1.4 expr.tree
Parent: Creating pipe...
Parent: Creating child...
PID = 11000, name +, starting...
Parent: Creating pipe...
Parent: Creating child...
Parent: Creating child...
Parent: Creating child...
PID = 11001, name 10, starting...
Parent: Creating pipe...
Parent: Creating child...
PID = 11002, name *, starting...
Parent: Creating pipe...
Parent: Creating child...
Parent: Creating child...
Parent: Creating child...
PID = 11003, name +, starting...
Parent: Creating child...
Parent: Creating pipe...
PID = 11004, name 4, starting...
Parent: Creating pipe...
Parent: Creating child...
Parent: Creating child...
PID = 11005, name 5, starting...
Parent: Creating child...
Parent: Creating pipe...
Parent: Creating child...
PID = 11006, name 7, starting...
Parent: Creating pipe...
Parent: Created child with PID = 11000, waiting for it to terminate...


+(11000)─┬─*(11002)─┬─+(11003)─┬─5(11005)
         │          │          └─7(11006)
         │          └─4(11004)
         └─10(11001)


My PID = 10999: Child PID = 11000 terminated normally, exit status = 0
Parent: All done, exiting...
Result = 58
.../Ergastirio/Askisi 2/ex_1.4
```

```
.../Ergastirio/Askisi 2/ex_1.4  ./ex_1.4 in1.tree
Parent: Creating pipe...
Parent: Creating child...
PID = 11038, name *, starting...
Parent: Creating pipe...
Parent: Creating child...
Parent: Creating child...
Parent: Creating child...
PID = 11039, name +, starting...
Parent: Creating pipe...
Parent: Creating child...
PID = 11040, name *, starting...
Parent: Creating pipe...
Parent: Creating child...
Parent: Creating child...
PID = 11041, name *, starting...
Parent: Creating child...
Parent: Creating pipe...
Parent: Creating child...
PID = 11042, name 4, starting...
Parent: Creating pipe...
Parent: Creating child...
Parent: Creating child...
Parent: Creating child...
Parent: Creating child...
Parent: Creating child...
PID = 11043, name 2, starting...
Parent: Creating pipe...
Parent: Creating child...
PID = 11046, name 2, starting...
Parent: Creating child...
PID = 11045, name 6, starting...
Parent: Creating pipe...
Parent: Creating pipe...
Parent: Creating child...
PID = 11044, name 3, starting...
Parent: Creating pipe...
Parent: Created child with PID = 11038, waiting for it to terminate...


*(11038)─┬─*(11040)─┬─2(11046)
         │          └─3(11044)
         └─+(11039)─┬─*(11041)─┬─2(11043)
                    │          └─6(11045)
                    └─4(11042)


My PID = 11037: Child PID = 11038 terminated normally, exit status = 0
Parent: All done, exiting...
Result = 96
.../Ergastirio/Askisi 2/ex_1.4
```

Απαντήσεις στις ερωτήσεις για την 4η άσκηση:

1. Θα μπορούσαμε να χρησιμοποιήσουμε μια σωλήνωση για κάθε παιδί, ωστόσο εμείς χρησιμοποιούμε μια σωλήνωση για κάθε γονική διεργασία για όλα τα παιδία. Στην προκειμένη άσκηση μπορούμε να χρησιμοποιήσουμε μια σωλήνωση για όλα τα παιδία διότι έχουμε μόνο πρόσθεση και πολλαπλασιασμό που δεν μας ενδιαφέρει η σειρά υπολογισμού. Αν είχαμε αφαίρεση και διαίρεση θα έπαιζε ρόλο η σειρά της πράξης οπότε θα χρειαζόμασταν μια σωλήνωση για κάθε παιδί.

2. Το πλεονέκτημα που θα έχει η αποτίμηση από δέντρο διεργασιών είναι ότι μπορούμε να μοιράσουμε σε μικρότερες διεργασίες τις λειτουργίες που θέλουμε να κάνουμε και να επιστρέφουμε στον πατέρα αυτό που μας ενδιαφέρει κάθε φορά, το αποτέλεσμα. Έτσι θα μπορούμε να τρέχουμε το πρόγραμμα να σπάει σε επιμέρους προγράμματα που θα τρέχουν παράλληλα και θα έχουμε πιο γρήγορα αποτελέσματα, αντί να το τρέχαμα σειριακά.