ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Λειτουργία Συστήματα Υπολογιστών

Αναφορά για την 2η Εργαστηριακή Άσκηση: Διαχείριση Διεργασιών και Διαδιεργασιακή Επικοινωνία

Ον/μο :Μεσσής Κωνσταντίνος
Α.Μ.: :03111122

Ον/μο :Μπερέτσος Θεόδωρος
Α.Μ.: :03111612

Ημερομηνία παράδοσης: 18/12/2014

1.1 Δημιουργία δεδομένου δέντρου διεργασιών

-Πηγαίος κώδικας (source code)

ask2-fork.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 b, c, d;
        int status;

        change_pname("A");
        printf("A: Created.\n");


        fprintf(stderr, "Parent, PID = %ld: Creating child...\n", (long)getpid());
        b = fork();
        if (b < 0) {
                /* fork failed */
                perror("fork");
                exit(1);
        }
        if (b == 0) {
                /* In child process */

                change_pname("B");
                printf("B: Created.\n");
                fprintf(stderr, "Parent, PID = %ld: Creating child...\n",
(long)getpid());
                d = fork();
                if (d < 0){
                        perror("fork");
                        exit(1);
                }
                if (d == 0){
                        /*In child D process */
                        change_pname("D");
                        printf("D: Created!\n");
                        printf("D: Sleeping...\n");
                        sleep(SLEEP_PROC_SEC);
                        printf("D: Exiting...\n");
                        exit(13);
```

```
                }
                change_pname("B");
                printf("Parent, PID = %ld: Created child with PID = %ld, waiting for it
to terminate...\n", (long)getpid(), (long)d);
                printf("B: Waiting...\n");
                d = wait(&status);
                explain_wait_status(d, status);
                printf("B: Exiting...\n");
                exit(19);
        }

        fprintf(stderr, "Parent, PID = %ld: Creating child...\n", (long)getpid());
        c = fork();
        if (c < 0) {
                /* fork failed */
                perror("fork");
                exit(1);
        }
        if (c == 0) {
                /* In child process */
                change_pname("C");
                printf("C: Created.\n");
                printf("C: Sleeping...\n");
                sleep(SLEEP_PROC_SEC);
                printf("C: Exiting...\n");
                exit(17);
        }


        change_pname("A");
        /*
         * In parent process. Wait for the child to terminate
         * and report its termination status.
         */
        printf("Parent, PID = %ld: Created child with PID = %ld, waiting for it to
terminate...\n", (long)getpid(), (long)b);
        printf("Parent, PID = %ld: Created child with PID = %ld, waiting for it to
terminate...\n", (long)getpid(), (long)c);
        printf("A: Waiting...\n");
        b = wait(&status);
        explain_wait_status(b, status);
        c = wait(&status);
        explain_wait_status(c, status);

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

/*
 * The initial process forks the root of the process tree,
 * waits for the process tree to be completely created,
 * then takes a photo of it using show_pstree().
 *
 * How to wait for the process tree to be ready?
 * In ask2-{fork, tree}:
 *      wait for a few seconds, hope for the best.
 * In ask2-signals:
 *      use wait_for_ready_children() to wait until
 *      the first process raises SIGSTOP.
 */
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-signals */
        /* wait_for_ready_children(1); */

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


        /* 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 */
        pid = wait(&status);
        explain_wait_status(pid, status);

        return 0;
}
```

-Έξοδο εκτέλεσης του προγράμματος

Η εντολή `./ask2-fork` έχει ως έξοδο στην οθόνη το εξής μήνυμα:

```
oslabb15@ithaki:~/oslab_ex2/forktree$ ./ask2-fork
A: Created.
Parent, PID = 5743: Creating child...
Parent, PID = 5743: Creating child...
Parent, PID = 5743: Created child with PID = 5744, waiting for it to terminate...
Parent, PID = 5743: Created child with PID = 5745, waiting for it to terminate...
A: Waiting...
B: Created.
Parent, PID = 5744: Creating child...
Parent, PID = 5744: Created child with PID = 5746, waiting for it to terminate...
B: Waiting...
C: Created.
C: Sleeping...
D: Created!
D: Sleeping...


A(5743)─┬─B(5744)──D(5746)
        └─C(5745)


C: Exiting...
D: Exiting...
My PID = 5743: Child PID = 5745 terminated normally, exit status = 17
My PID = 5744: Child PID = 5746 terminated normally, exit status = 13
B: Exiting...
My PID = 5743: Child PID = 5744 terminated normally, exit status = 19
A: Exiting...
My PID = 5742: Child PID = 5743 terminated normally, exit status = 16
```

-Σύντομες απαντήσεις στις ερωτήσεις

Ερώτηση1: Τι θα γίνει αν τερματίσετε πρόωρα τη διεργασία Α, δίνοντας `kill -KILL <pid>`, όπου `<pid>` το Process ID της;

Απάντηση:

Με τη χρήση της kill θα σταματήσουμε τη διεργασία Α. Αυτό σημαίνει πως τα παιδιά της Α θα κληρονομηθούν από την διεργασία πατέρα της Α. Έτσι οι εργασίες που γίνονται για τα πρωτεύοντα παιδιά της Α δηλαδή τα Β C όπως εμφανίσεις για εκκίνηση, έξοδο, κλπ. δεν θα εκτελεστούν αφού η Α έχει πεθάνει. Παρ' όλα αυτά οι διαδικασίες αυτές θα εκτελεστούν κανονικά για το παιδί της Β ,D αφού η Β υπάρχει και εξελίσσεται κανονικά.

Ερώτηση 2: Τι θα γίνει αν κάνετε `show_pstree(getpid())` αντί για `show_pstree(pid)` στη `main()`; Ποιες επιπλέον διεργασίες φαίνονται στο δέντρο και γιατί;

Απάντηση:

Η έξοδος του προγράμματος θα είναι η εξής:

```
oslabb15@lesvos:~/oslab_ex2/forktree$ ./ask2-fork
A: Created.
Parent, PID = 24795: Creating child...
Parent, PID = 24795: Creating child...
B: Created.
Parent, PID = 24796: Creating child...
Parent, PID = 24795: Created child with PID = 24796, waiting for it to terminate...
Parent, PID = 24795: Created child with PID = 24797, waiting for it to terminate...
A: Waiting...
C: Created.
C: Sleeping...
Parent, PID = 24796: Created child with PID = 24798, waiting for it to terminate...
B: Waiting...
D: Created!
D: Sleeping...


ask2-fork(24794)─┬─A(24795)─┬─B(24796)──D(24798)
                 │          └─C(24797)
                 └─sh(24799)──pstree(24800)


C: Exiting...
D: Exiting...
My PID = 24795: Child PID = 24797 terminated normally, exit status = 17
My PID = 24796: Child PID = 24798 terminated normally, exit status = 13
B: Exiting...
My PID = 24795: Child PID = 24796 terminated normally, exit status = 19
A: Exiting...
My PID = 24794: Child PID = 24795 terminated normally, exit status = 16
```

Παρατηρούμε ότι εμφανίζεσαι η διεργασία που έφτιαξε τον γονέα Α (`ask2-frok(24794)`). Ακόμη αυτή η διεργασία καλεί την sh (εντολή του φλοιού (shell)), η οποία εκτελεί εντολές για διάβασμα από εντολή, string, standard input συγκεκριμένο αρχείο και εν προκειμένω την τύπωση του δέντρου.

Ερώτηση 3: Σε υπολογιστικά συστήματα πολλαπλών χρηστών, πολλές φορές ο διαχειριστής θέτει όρια στον αριθμό των διεργασιών που μπορεί να δημιουργήσει ένας χρήστης. Γιατί;

Απάντηση:

Η δημιουργία και λειτουργία μιας διεργασίας απαιτεί πόρους συστήματος. Σε περίπτωση που από κάποιο λάθος ο χρήστης δημιουργήσει αλόγιστα πολλές διεργασίες το σύστημα είναι πιθανόν να καταρρεύσει. Για αυτό είναι καλό κανείς να μεριμνά ώστε να δίνει συγκεκριμένο όριο διεργασιών σε κάθε χρήστη ώστε να μην αφήσει ανοικτό το ενδεχόμενο να γίνει κάποιο σφάλμα σαν το παραπάνω.

1.2 Δημιουργία αυθαίρετου δέντρου διεργασιών

-Πηγαίος κώδικας (source code)

ask2-tree.c

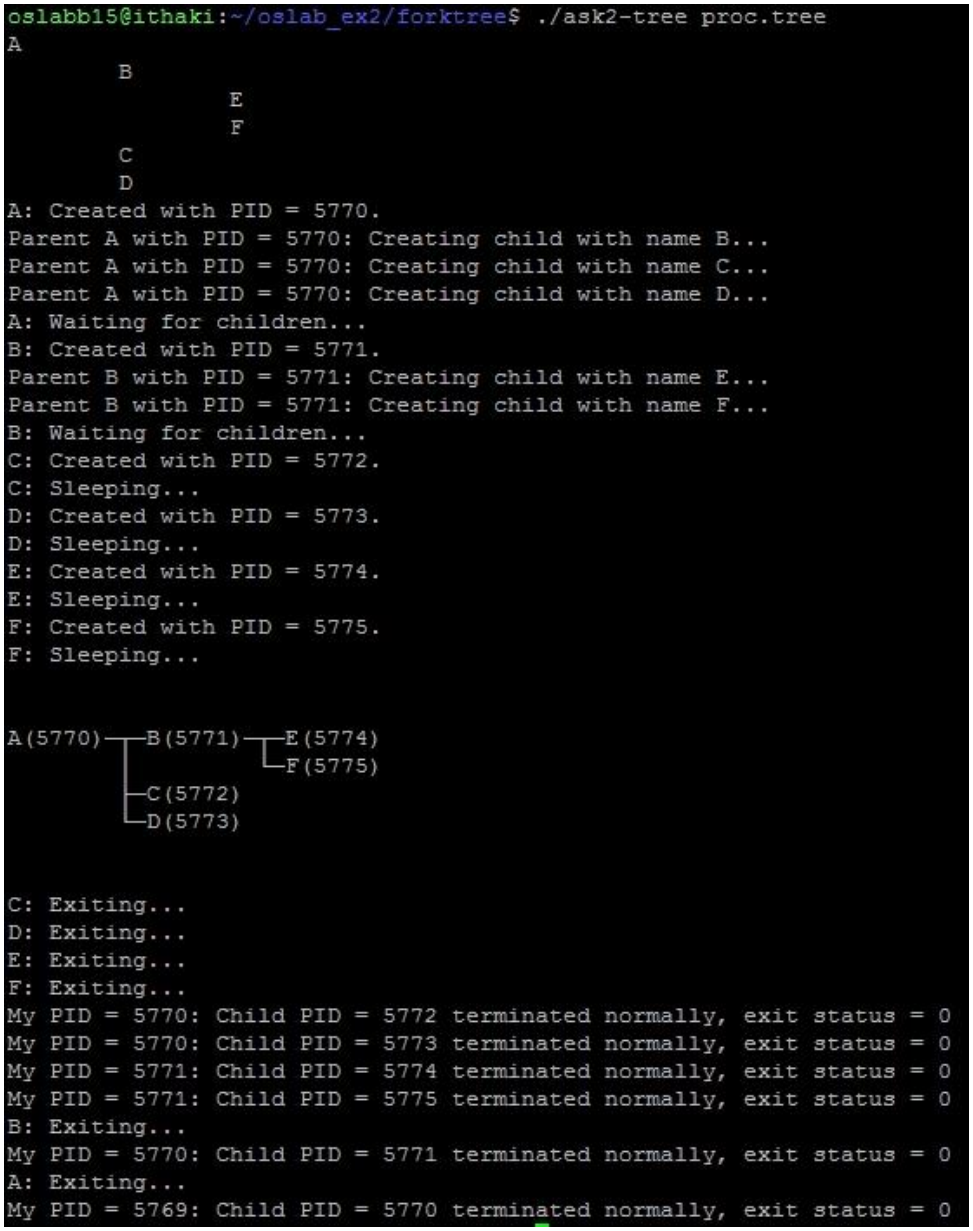
```
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <sys/types.h>
#include <sys/wait.h>

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

#define SLEEP_PROC_SEC  8
#define SLEEP_TREE_SEC  3

void fork_procs(struct tree_node* root)
{
        /*
         * initial process is root
         */

        pid_t pid_child;
        int i, status;
        pid_t* child;

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

        child = (pid_t*)malloc(sizeof(pid_t)*root->nr_children);
        if (child == NULL){
                fprintf(stderr,"allocate children failed\n");
                exit(1);
        }

        //if root has childen then forking is necessary
        for (i = 0; i < root->nr_children; i++){
                fprintf(stderr, "Parent %s with PID = %ld: Creating child with name
%s...\n", root->name, (long)getpid(), root->children[i].name);
                pid_child = fork();
                if (pid_child < 0) {
                        /* fork failed */
                        perror("fork");
                        exit(1);
                }
                if (pid_child == 0) {
                        /* In child process */
                        fork_procs(&root->children[i]);
                }
                //Parent saves child's PID
                child[i] = pid_child;
```

```
        }
        if (root->nr_children == 0){
                printf("%s: Sleeping...\n", root->name);
                sleep(SLEEP_PROC_SEC);
                printf("%s: Exiting...\n", root->name);
        }
                /*change_pname("A");
                 *
                 * In parent process. Wait for the child to terminate
                 * and report its termination status.
                 */
        else{
                printf("%s: Waiting for children...\n", root->name);
                for (i = 0; i < root->nr_children; ++i){
                        child[i] = wait(&status);
                        explain_wait_status(child[i], status);
                }
                printf("%s: Exiting...\n", root->name);
                free(child);
        }
        exit(0);
}

/*
 * The initial process forks the root of the process tree,
 * waits for the process tree to be completely created,
 * then takes a photo of it using show_pstree().
 *
 * How to wait for the process tree to be ready?
 * In ask2-{fork, tree}:
 *      wait for a few seconds, hope for the best.
 * In ask2-signals:
 *      use wait_for_ready_children() to wait until
 *      the first process raises SIGSTOP.
 */
int main(int argc, char *argv[])
{
        pid_t pid;
        int status;
        struct tree_node *root;

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

        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);
        }

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

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


        /* 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 */
        pid = wait(&status);
        explain_wait_status(pid, status);

        return 0;
}
```

-Έξοδο εκτέλεσης του προγράμματος

Η εντολή `./ask2-tree proc.tree` έχει ως έξοδο στην οθόνη το εξής μήνυμα:



-Σύντομες απαντήσεις στις ερωτήσεις

Ερώτηση 1: Με ποια σειρά εμφανίζονται τα μηνύματα έναρξης και τερματισμού των διεργασιών; γιατί;

Απάντηση:

Αν κανείς παρατηρήσει τον κώδικα θα δει ότι δεν υπάρχει κάποιος έλεγχος ως προς την σειρά με την οποία δημιουργούνται και τρέχουν οι διεργασίες και τα μηνύματά τους. Το μόνο που κάνουμε είναι wait για κάθε διεργασία ώστε να περιμένει τα παιδιά της να κλείσουν και sleep στα φύλλα . Είναι ξεκάθαρα τυχαία η σειρά με την οποία θα εμφανιστούν τα μηνύματα αφού δεν εξασφαλίζεται κάπως η depth-first σειρά στο τρέξιμο των διεργασιών , παρά μόνο στο κλείσιμό τους (με την wait).

1.3 Αποστολή και χειρισμός σημάτων

-Πηγαίος κώδικας (source code)

ask2-signals.c

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

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

#define SLEEP_PROC_SEC  8
#define SLEEP_TREE_SEC  3

void fork_procs(struct tree_node* root)
{
        /*
         * initial process is root
         */

        pid_t pid_child;
        int i, status;
        pid_t* child;

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

        child = (pid_t*)malloc(sizeof(pid_t)*root->nr_children);
        if (child == NULL){
                fprintf(stderr,"allocate children failed\n");
                exit(1);
        }

        //if root has childen then forking is necessary
        for (i = 0; i < root->nr_children; i++){
                fprintf(stderr, "Parent %s with PID = %ld: Creating child with name
%s...\n", root->name, (long)getpid(), root->children[i].name);
                pid_child = fork();
                if (pid_child < 0) {
                        /* fork failed */
                        perror("fork");
```

```
                        exit(1);
                }
                if (pid_child == 0) {
                        /* In child process */
                        fork_procs(&root->children[i]);
                }
                //Parent saves child's PID
                child[i] = pid_child;
        }
        printf("%s: Waiting for children...\n", root->name);
        wait_for_ready_children(root->nr_children);
        raise(SIGSTOP);
        printf("%s:Awake...\n",root->name);
        for (i = 0; i < root->nr_children; ++i){
                printf("%s: Waking child %s...\n", root->name, root-
>children[i].name);//?
                kill(child[i], SIGCONT);
                child[i] = wait(&status);
                explain_wait_status(child[i], status);
        }
        printf("%s: Exiting...\n", root->name);
        free(child);
        exit(0);
}

/*
 * The initial process forks the root of the process tree,
 * waits for the process tree to be completely created,
 * then takes a photo of it using show_pstree().
 *
 * How to wait for the process tree to be ready?
 * In ask2-{fork, tree}:
 *      wait for a few seconds, hope for the best.
 * In ask2-signals:
 *      use wait_for_ready_children() to wait until
 *      the first process raises SIGSTOP.
 */
int main(int argc, char *argv[])
{
        pid_t pid;
        int status;
        struct tree_node *root;

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

        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);
        }

        /*
         * Father
         */
```

```
        /* for ask2-signals */
        /* wait_for_ready_children(1); */
        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 */
        pid = wait(&status);
        explain_wait_status(pid, status);

        return 0;
}
```

-Έξοδο εκτέλεσης του προγράμματος

Η εντολή `./ask2-signals proc.tree` έχει ως έξοδο στην οθόνη το εξής μήνυμα:

```
oslabb15@ithaki:~/oslab_ex2/forktree$ ./ask2-expr proc.tree
A
        B
                E
                F
        C
        D
PID=5779, name A,starting...
Parent A with PID = 5779: Creating child with name B...
Parent A with PID = 5779: Creating child with name C...
Parent A with PID = 5779: Creating child with name D...
A: Waiting for children...
PID=5780, name B,starting...
Parent B with PID = 5780: Creating child with name E...
Parent B with PID = 5780: Creating child with name F...
B: Waiting for children...
PID=5781, name C,starting...
C: Waiting for children...
My PID = 5779: Child PID = 5781 has been stopped by a signal, signo = 19
PID=5782, name D,starting...
D: Waiting for children...
My PID = 5779: Child PID = 5782 has been stopped by a signal, signo = 19
PID=5783, name E,starting...
E: Waiting for children...
My PID = 5780: Child PID = 5783 has been stopped by a signal, signo = 19
PID=5784, name F,starting...
F: Waiting for children...
My PID = 5780: Child PID = 5784 has been stopped by a signal, signo = 19
My PID = 5779: Child PID = 5780 has been stopped by a signal, signo = 19
My PID = 5778: Child PID = 5779 has been stopped by a signal, signo = 19


A(5779)─┬─B(5780)─┬─E(5783)
        │         └─F(5784)
        ├─C(5781)
        └─D(5782)


A:Awake...
A: Waking child B...
B:Awake...
B: Waking child E...
E:Awake...
E: Exiting...
My PID = 5780: Child PID = 5783 terminated normally, exit status = 0
B: Waking child F...
F:Awake...
F: Exiting...
My PID = 5780: Child PID = 5784 terminated normally, exit status = 0
B: Exiting...
My PID = 5779: Child PID = 5780 terminated normally, exit status = 0
A: Waking child C...
C:Awake...
C: Exiting...
My PID = 5779: Child PID = 5781 terminated normally, exit status = 0
A: Waking child D...
D:Awake...
D: Exiting...
My PID = 5779: Child PID = 5782 terminated normally, exit status = 0
A: Exiting...
My PID = 5778: Child PID = 5779 terminated normally, exit status = 0
```

-Σύντομες απαντήσεις στις ερωτήσεις

Ερώτηση1: Στις προηγούμενες ασκήσεις χρησιμοποιήσαμε τη `sleep()` για τον συγχρονισμό των διεργασιών. Τι πλεονεκτήματα έχει η χρήση σημάτων;

Απάντηση:

Με τη χρήση μηνυμάτων και της `wait_for_ready_children` εξασφαλίζουμε την depth-first εκκίνηση και λειτουργία των διεργασιών αντίθετα με την προηγούμενη υλοποίηση που το έκανε τυχαία.

Ερώτηση2: Ποιος ο ρόλος της `wait_for_ready_children();` Τι εξασφαλίζει η χρήση της και τι πρόβλημα θα δημιουργούσε η παράλειψή της;

Απάντηση:

Η συνάρτηση `wait_for_ready_children` δέχεται έναν αριθμό παιδιών ως όρισμα. Ειδοποιεί την διεργασία που την κάλεσε ότι ο συγκεκριμένος αριθμός παιδιών έχει στείλει σήμα `raise(sigstop)`.Η χρήση της είναι και αυτή που εξασφαλίζει την depth-first λειτουργία των διεργασιών μας. Χωρίς αυτή θα χανόταν ο επιθυμητός συγχρονισμός διεργασιών και η σειρά λειτουργίας τους θα αφηνόταν τυχαία.

```
void
wait_for_ready_children(int cnt)
{
        int i;
        pid_t p;
        int status;

        for (i = 0; i < cnt; i++) {
                /* Wait for any child, also get status for stopped children */
                p = waitpid(-1, &status, WUNTRACED);
                explain_wait_status(p, status);
                if (!WIFSTOPPED(status)) {
                        fprintf(stderr, "Parent: Child with PID %ld has died
unexpectedly!\n",
                                        (long)p);
                        exit(1);
                }
        }
}
```

1.4 Παράλληλος υπολογισμός αριθμητικής έκφρασης

-Πηγαίος κώδικας (source code)

`ask2-praks.c`

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

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

```c
#define SLEEP_PROC_SEC  8
#define SLEEP_TREE_SEC  3

void fork_procs(struct tree_node* root,int fd)
{
        /*
         * initial process is root
         */

        int pfd[2][2];
        int value;
        int value1,value2;
        pid_t pid_child;
        int i, status;
        pid_t* child;

        change_pname(root->name);
        printf("%s: Created with PID = %ld,and given fd= %d.\n", root->name,
(long)getpid(),fd);

        child = (pid_t*)malloc(sizeof(pid_t)*root->nr_children);
        if (child == NULL){
                fprintf(stderr,"allocate children failed\n");
                exit(1);
        }

        if(root->nr_children==0){
                value=atoi(root->name);
                if(write(fd,&value,sizeof(value)) !=sizeof(value)){
                        perror("write to pipe");
                        exit(1);
                }
                exit(0);
        }

        //if root has childen then forking is necessary
        for (i = 0; i < root->nr_children; i++){

                if (pipe(pfd[i])<0){
                        perror("pipe");
                        exit(1);
                }

                pid_child = fork();
                if (pid_child < 0) {
                        /* fork failed */
                        perror("fork");
                        exit(1);
                }
                if (pid_child == 0) {
                        /* In child process */
                        close(pfd[i][0]);
                        fork_procs(&root->children[i],pfd[i][1]);
                }
        }
        pid_child = wait(&status);
        explain_wait_status(pid_child,status);
        pid_child = wait(&status);
        explain_wait_status(pid_child,status);

        if (read(pfd[0][0], &value1, sizeof(value1)) != sizeof(value1)) {
                perror("child: read from pipe");
                exit(1);
        }
        if (read(pfd[1][0], &value2, sizeof(value2)) != sizeof(value2)) {
                perror("child: read from pipe");
```

```
                exit(1);
        }
        printf("%s: Read from pipes values %d & %d \n",root->name,value1,value2);
        if (strcmp(root->name,"+") == 0)
                value = value1 + value2;
        else
                value = value1 * value2;
        if (write(fd, &value, sizeof(value)) != sizeof(value)) {
                perror("parent: write to pipe");
                exit(1);
        }
        close(fd);
        printf("%s: Wrote to pipe result: %d!\n",root->name,value);
        printf("%s: Exiting...\n",root->name);
        free(child);


        exit(7);
}

/*
 * The initial process forks the root of the process tree,
 * waits for the process tree to be completely created,
 * then takes a photo of it using show_pstree().
 *
 * How to wait for the process tree to be ready?
 * In ask2-{fork, tree}:
 *      wait for a few seconds, hope for the best.
 * In ask2-signals:
 *      use wait_for_ready_children() to wait until
 *      the first process raises SIGSTOP.
 */
int main(int argc, char *argv[])
{
        int fd[2];
        pid_t pid;
        int status,value;
        struct tree_node *root;

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

        root = get_tree_from_file(argv[1]);
        print_tree(root);
        if(pipe(fd)<0){
                perror("pipe");
                exit(1);
        }

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

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

        if(read(fd[0],&value,sizeof(value))!=sizeof(value)){
                perror("read from pipe");
                exit(1);
        }
        close(fd[0]);


        printf("FINAL RESULT IS %d!!\n",value);
        return 0;
}
```

-Έξοδο εκτέλεσης του προγράμματος

Η εντολή `./ask2-praks expr.tree` έχει ως έξοδο στην οθόνη το εξής μήνυμα:

```
oslabb15@ithaki:~/oslab_ex2/forktree$ ./ask2-praks expr.tree
+
        10
        *
                +
                        5
                        7
                4
+: Created with PID = 5788,and given fd= 4.
10: Created with PID = 5789,and given fd= 5.
My PID = 5788: Child PID = 5789 terminated normally, exit status = 0
*: Created with PID = 5790,and given fd= 7.
+: Created with PID = 5791,and given fd= 8.
4: Created with PID = 5792,and given fd= 10.
My PID = 5790: Child PID = 5792 terminated normally, exit status = 0
5: Created with PID = 5793,and given fd= 9.
My PID = 5791: Child PID = 5793 terminated normally, exit status = 0
7: Created with PID = 5794,and given fd= 11.
My PID = 5791: Child PID = 5794 terminated normally, exit status = 0
+: Read from pipes values 5 & 7
+: Wrote to pipe result: 12!
+: Exiting...
My PID = 5790: Child PID = 5791 terminated normally, exit status = 7
*: Read from pipes values 12 & 4
*: Wrote to pipe result: 48!
*: Exiting...
My PID = 5788: Child PID = 5790 terminated normally, exit status = 7
+: Read from pipes values 10 & 48
+: Wrote to pipe result: 58!
+: Exiting...
My PID = 5787: Child PID = 5788 terminated normally, exit status = 7
FINAL RESULT IS 58!!
```

-Σύντομες απαντήσεις στις ερωτήσεις

Ερώτηση1: Πόσες σωληνώσεις χρειάζονται στη συγκεκριμένη άσκηση ανά διεργασία; Θα μπορούσε κάθε γονική διεργασία να χρησιμοποιεί μόνο μία σωλήνωση για όλες τις διεργασίες παιδιά; Γενικά, μπορεί για κάθε αριθμητικό τελεστή να χρησιμοποιηθεί μόνο μια σωλήνωση;

Απάντηση:

Στην δική μας υλοποίηση χρησιμοποιήσαμε δύο σωληνώσεις μια για κάθε παιδί. Στην πραγματικότητα δεν είναι αναγκαίο αυτό για το συγκεκριμένο πρόβλημα και ίσως θα ήταν προτιμότερο να το είχαμε αποφύγει και να είχαμε χρησιμοποιήσει μόνο μια σωλήνωση σε περίπτωση που το πρόβλημα ήταν ίδιο αλλά η κάθε διεργασία πράξης είχε μεγαλύτερο αριθμό παιδιών ώστε να μην δημιουργούμε πολλές σωληνώσεις για εξοικονόμηση. Παρ' όλα αυτά η υλοποίηση με μια σωλήνωση είναι εφικτή μόνο επειδή η πρόσθεση και ο πολλαπλασιασμός που ζητά η άσκηση είναι πράξεις που έχουν την αντιμεταθετική ιδιότητα. Σε πράξεις που δεν την έχουν όπως η αφαίρεση και η διαίρεση όπου θα ήταν απαραίτητη η γνώση της σειράς με την οποία έρχονται τα αποτελέσματα θα ήταν επιβεβλημένη η χρήση μιας σωλήνωσης για κάθε παιδί.

Ερώτηση2: Σε ένα σύστημα πολλαπλών επεξεργαστών, μπορούν να εκτελούνται παραπάνω από μια διεργασίες παράλληλα. Σε ένα τέτοιο σύστημα, τι πλεονέκτημα μπορεί να έχει η αποτίμηση της έκφρασης από δέντρο διεργασιών, έναντι της αποτίμησης από μία μόνο διεργασία;

Απάντηση:

Με τη χρήση συστήματος πολλαπλών επεξεργαστών και δένδρου διεργασιών εκμεταλλευόμαστε την δυνατότητα του συστήματος να εκτελούνται παραπάνω από μια διεργασίες παράλληλα και πετυχαίνουμε ταχύτερη εκτέλεση των διεργασιών οι οποίες πλέον δεν εκτελούνται σειριακά (η μια μετά την άλλη) όπως στη χρήση μιας μόνο διεργασίας.

-Διαδικασία μεταγλώττισης και σύνδεσης

Αλλάξαμε κατάλληλα το Makefile που μας δίνετε ώστε να δημιουργεί τα εκτελέσιμα όλες τις ασκήσεις όπως φαίνεται παρακάτω:

```
all: fork-example tree-example ask2-fork ask2-signals ask2-tree ask2-expr ask2-praks


CC = gcc
CFLAGS = -g -Wall -O2
SHELL= /bin/bash

tree-example: tree-example.o tree.o
    $(CC) $(CFLAGS) $^ -o $@

fork-example: fork-example.o proc-common.o
    $(CC) $(CFLAGS) $^ -o $@

ask2-fork: ask2-fork.o proc-common.o
    $(CC) $(CFLAGS) $^ -o $@

ask2-tree: ask2-tree.o proc-common.o tree.o
    $(CC) $(CFLAGS) $^ -o $@

ask2-signals: ask2-signals.o proc-common.o tree.o
    $(CC) $(CFLAGS) $^ -o $@

ask2-expr: ask2-expr.o proc-common.o tree.o
    $(CC) $(CFLAGS) $^ -o $@

ask2-praks: ask2-praks.o proc-common.o tree.o
```

```
    $(CC) $(CFLAGS) $^ -o $@

%.s: %.c
    $(CC) $(CFLAGS) -S -fverbose-asm $<

%.o: %.c
    $(CC) $(CFLAGS) -c $<

%.i: %.c
    gcc -Wall -E $< | indent -kr > $@

clean:
    rm -f *.o tree-example fork-example pstree-this ask2-
{fork,tree,signals,pipes,expr,praks}
```