ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
& ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Λειτουργικά Συστήματα

7ο Εξάμηνο

Άσκηση 2

Διαχείριση Διεργασιών και Διαδιεργασιακή Επικοινωνία

Εργαστηριακή Ομάδα Δ01

Σταυρακάκης Δημήτριος
ΑΜ: 03112017

Αθανασίου Νικόλαος
ΑΜ: 03112074

Άσκηση 2
Διαχείριση Διεργασιών και Διαδιεργασιακή Επικοινωνία

1.1 Δημιουργία δεδομένου δέντρου διεργασιών

Στη συγκεκριμένη άσκηση θα γράψουμε έναν κώδικα ο οποίος θα δημιουργεί και θα εμφανίζει ένα δέντρο διεργασιών όπως αυτό περιγράφεται στην εκφώνηση. Προκειμένου να επιτευχθεί αυτό, δημιουργούμε τις διεργασίες, τους δίνουμε τα κατάλληλα ονόματα (change_pname()), τις βάζουμε για ύπνο (system call : sleep) για δεδομένο χρονικό διάστημα, που μας δίνεται, και τυπώνουμε το δέντρο διεργασιών που έχει ως ρίζα τη διεργασία που δημιουργείται με fork() από τη main μας. Για την εμφάνιση του δέντρου χρησιμοποιούμε τη συνάρτηση show_pstree που μας δίνεται. Επίσης, κάθε κόμβος-γονιός αναμένει να τερματίσει το παιδί του για να τερματιστεί ο ίδιος. Σε κάθε τέτοια περίπτωση τυπώνεται κατάλληλο μήνυμα (με την explain_wait_status). Έτσι έχουμε τη δυνατότητα να τυπώσουμε ορθά το δέντρο διεργασιών. Για να δούμε ότι όλα πήγαν καλά ορίζουμε διαφορετικό exit status για την κάθε διεργασία.
Ο κώδικας που υλοποιεί όσα περιγράφηκαν παραπάνω είναι ο ακόλουθος και τα αποτελέσματα του φαίνονται στο screenshot που παραθέτουμε στη συνέχεια:

```
#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
 */
/*
 * 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) {
	/* A */
	//fork_procs();
	pid=fork(); //fork for B
	if (pid<0){
		perror("main: fork");
	}
	if (pid == 0){
		/* B */
		//fork_procs();
		pid=fork();//fork for D
		if (pid<0){
			perror("main: fork");
		}
		if (pid == 0){
			/* D */
			change_pname("D");
			printf("D: Sleeping...\n");
			sleep(SLEEP_PROC_SEC);
			//pid = wait(&status);
			//explain_wait_status(pid,status);
			printf("D: Exiting...\n");
			exit(13);

		}
		change_pname("B");
		printf("B: Waiting\n");
		pid = wait(&status); //B waits for D to end
		explain_wait_status(pid, status);
		printf("B: Exiting...\n");
		exit(19);

	}
	/* A continues here */
	pid=fork();//fork for C
	if (pid<0){
	     perror("main: fork");
	}
	if (pid == 0){
		/* C */
		change_pname("C");
		printf("C: Sleeping...\n");
		sleep(SLEEP_PROC_SEC);
		// pid = wait(&status);
		//explain_wait_status(pid,status);
		printf("C: Exiting...\n");
		exit(17);
```

```
        }
        change_pname("A");
        printf("A: Waiting\n");
        pid = wait(&status);
        explain_wait_status(pid,status);
        pid = wait(&status);
        explain_wait_status(pid,status);//A waits for its children B C to end
        printf("A: Exiting...\n");
        exit(16);
    }

    /*
     * 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);
    //printf("\n");
    //printf("\n");
    //printf("\n");
    //show_pstree(getpid());
    /* 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;
}
```

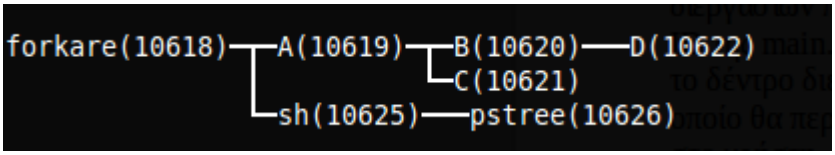
```
oslabd01@poros:~/Askhsh2/Askhsh1.1$ ./forkare
A: Waiting
B: Waiting
C: Sleeping...
D: Sleeping...


A(10595)─┬─B(10596)──D(10598)
         └─C(10597)


D: Exiting...
C: Exiting...
My PID = 10596: Child PID = 10598 terminated normally, exit status = 13
B: Exiting...
My PID = 10595: Child PID = 10597 terminated normally, exit status = 17
My PID = 10595: Child PID = 10596 terminated normally, exit status = 19
A: Exiting...
My PID = 10594: Child PID = 10595 terminated normally, exit status = 16
```

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

Η διεργασία Α έχει κάποια παιδια (όπως φαίνεται και παραπάνω). Αν τη “σκοτώσουμε” πρόωρα (με την παραπάνω εντολή), πριν δηλαδή προλάβουν να τερματιστούν επιτυχώς τα παιδιά της (ή έστω κάποια απο αυτά), αυτά θα υιοθετηθούν από την init.(αυτό συμβαίνει και σε γενική περίπτωση αν γονιός τερματίσει πριν τερματίσει το παιδί του) Η init είναι η αρχική διεργασία του συστήματος. Ο Process ID της είναι 1. Αυτή η διεργασία με τη σειρά της θα περιμένει μέχρι να τερματιστούν οι διεργασίες-παιδιά που υιοθέτησε (κάνει wait δηλαδή). Σε περιπτώσεις σαν αυτή, τα παιδιά-διεργασίες των οποιών ο γονιός “θανατώθηκε βίαια”,πριν αυτά “πεθάνουν”, ονομάζονται zombies.

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

Η συνάρτηση show_pstree() στην ουσία εμφανίζει το δέντρο διεργασιών που έχει ώς ρίζα τη διεργασία με pid την παράμετρο που της δίνεται. Γι'αυτό και με το να κάνουμε show_pstree(pid) εμφανίζεται το δέντρο διεργασιών με ρίζα την διεργασία Α, καθώς η fork() αυτό το pid επιστρέφει στην main() (αφού η Α είναι παιδί της main) . Επομένως αν δώσουμε στην show_pstree() ως παράμετρο την τιμή που επιστρέφει η συνάρτηση getpid(), θα εμφανίσει ένα διαφορετικό δέντρο διεργασιών που θα έχει ως διεργασία-ρίζα τη main μας αφού η getpid() θα επιστρέψει το Process ID της main. Στο δέντρο αυτό , η main θα έχει ως παιδί την Α και γενικά θα περιέχει ως υποδέντρο το δέντρο διεργασιών που δημιουργήσαμε με τον κώδικα μας, καθώς επίσης ένα άλλο υπόδεντρο το οποίο θα περιέχει τις διεργασίες που χρησιμοποιούνται για την εμφάνιση του δέντρου διεργασιών στο χρήστη. Οπτικά:



Σημείωση: Διεργασία sh είναι ο bash.

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

Αυτό συμβαίνει για την ασφάλεια του συστήματος. Αν κάθε χρήστης είχε τη δυνατότητα να δημιουργεί όσες διεργασίες επιθυμεί, το μεγάλο πλήθος διεργασιών θα ήταν πρόβλημα. Αυτό γιατί, όλες οι διεργασίες διαμοιράζονται μέσα στο διαθέσιμο χρόνο με αποτέλεσμα ο μεγάλος αριθμός να οδηγεί στο ότι κάθε διεργασία θα έχει ελάχιστο χρόνο στη διάθεσή της προκειμένου να εκτελέσει τη λειτουργία της. Αυτό ισχύει και για τις διεργασίες του συστήματος. Επομένως, με αυτή τη λογική το σύστημα θα ήταν πιθανό να μη μπορεί να ανταπεξέλθει κάποια στιγμή και να καταρρεύσει πράγμα που εμείς δεν επιθυμούμε. Γι'αυτό το λόγο θέτουμε αυτόν τον περιορισμό στον αριθμό των διεργασιών που μπορεί ένας χρήστης να δημιουργήσει.

1.2 Δημιουργία αυθαίρετου δέντρου διεργασιών

Στην άσκηση αυτή κάνουμε παρόμοια διαδικασία με την προηγούμενη μόνο που τώρα το δέντρο που καλούμαστε να φτιάξουμε δεν είναι δεδομένο αλλά διαβάζουμε τη μορφή του από αρχείο εισόδου. Χρησιμοποιώντας τη συνάρτηση get_tree_from_file παίρνουμε το δέντρο από το αρχείο. (η συνάρτηση διαβάζει το αρχείο, κατασκευάζει την αναπαράσταση του δέντρου στη μνήμη και επιστρέφει δείκτη στη ρίζα) Αφού λοιπόν διαβάσουμε καλούμαστε να δημιουργήσουμε το δέντρο. Θα το κάνουμε με αναδρομική κλήση της συνάρτησης let_it_fork (δική μας συνάρτηση), η οποία στην ουσία καλείται για κάθε διεργασία και ανάλογα με την ιδιότητα της υλοποιεί τα εξής:

- Αν η διεργασία έχει παιδιά με for loop τα δημιουργεί
- Αν η διεργασία δεν έχει παιδια (είναι φύλλο) κάνει sleep() συγκεκριμενο χρόνο.

Κάθε διεργασία τερματίζεται αφού τερματίσουν όλα τα παιδιά της. (κάνει wait())
Ο κώδικας που υλοποιεί τα παραπάνω είναι ο ακόλουθος:

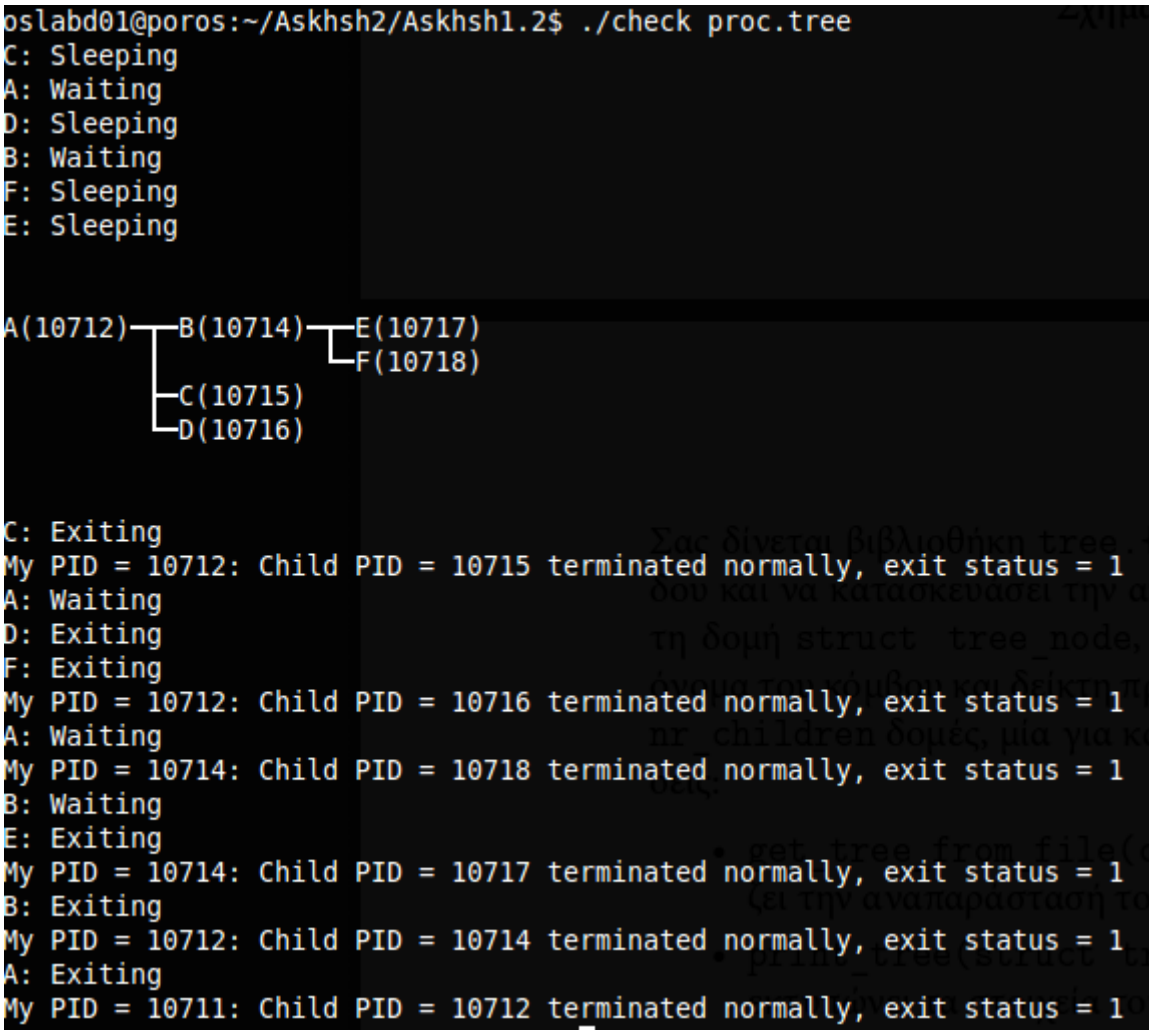
```
#include <stdio.h>
#include <stdlib.h>

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

int sleeping=10;
int flag=0;
//pid_t headp=0;

void let_it_fork(struct tree_node * );

int main(int argc, char *argv[])
{       pid_t pid;
        struct tree_node *root;
        int status;
        if (argc != 2) {
                fprintf(stderr, "Usage: %s <input_tree_file>\n\n", argv[0]);
                exit(1);
        }

        root = get_tree_from_file(argv[1]);
        pid = fork();
        if (pid<0){
                perror("main: fork\n");
                exit(1);
        }
        if (pid==0){
                //call my void for the first child to get its pid for the tree
                let_it_fork(root);
                exit(1);
        }

        //show_pstree(pid);
        //printf("%d\n",headp);
        //sleep(3);
        //if (flag!=0)
        //      show_pstree(headp);
        //else
```

```
	//	show_pstree(getpid());

	show_pstree(pid);

	//sleep(3);
	pid = wait(&status);
	//printf("%d\n",root);

	explain_wait_status(pid,status);
	//print_tree(root);
	//show_pstree(pid);
	return 0;
}

void let_it_fork(struct tree_node * root)
{
	change_pname(root->name); //gia na allaksoume to id ka8e diadikasias sto epi8ymhto onoma
	int i;
	int status;
//	printf("Working with %s node \n" , root->name);
	if (root->nr_children == 0){
		printf("%s: Sleeping\n",root->name);
		sleep(sleeping);//periptwsh pou einai fyllo
		printf("%s: Exiting\n",root->name);
		exit(1);
	}
	else{
		//periptwsh pou exoume paidia, prepei na ta kanoume fork
		pid_t pid;

		for (i=0;i<root->nr_children;i++){
			pid=fork();
			//if (flag==0){
			//	headp=pid;//take the head pid
			//	flag++;
			//}
			if (pid<0){
				perror("problem with fork\n");
				exit(1);
			}
			if (pid == 0){
				let_it_fork(root->children+i);		//child, recursive call gia ka8e pointer se
									//child
				exit(1);
			}
		}

		//edw synexizei mono o goneas (pid>0)
		//kai prepei na perimenei ta paidia tou
		for (i=0;i<root->nr_children;i++){
			printf("%s: Waiting\n",root->name);
			pid=wait(&status);
```

```
                explain_wait_status(pid,status); //wait for every child and diplay appropriate
                                                 //message
            }
        }
        printf("%s: Exiting\n",root->name);
}
```

Σημείωση: Οι εντολές-σχόλια χρησιμοποιήθηκα για debugging

Ο παραπάνω κώδικας, όταν λάβει σαν όρισμα το αρχείο proc.tree που περιέχει ένα δέντρο δίνει τα παρακάτω:



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

Η σειρά εμφάνισης των μηνυμάτων σε αυτή την άσκηση είναι τυχαία. Μπορεί να αλλάζει απο εκτέλεση σε εκτέλεση. Αυτό συμβαίνει γιατί οι διεργασίες δε δημιουργούνται με κάποια καθορισμένη σειρά. Κάθε μία ξεκινάει και φτιάχνει τα παιδιά της, τα παιδιά της δημιουργούν τα δικά τους κοκ, και όλα αυτά γίνονται παράλληλα οπότε δε μπορούμε να είμαστε σίγουροι για το ποιο θα δημιουργηθεί πρώτα (έτσι εξηγείται η τυχαία σειρά μηνμυμάτων έναρξης) και αντίστοιχα ποιο θα πεθάνει πρώτα. Το μόνο που γνωρίζουμε είναι ότι μια διεργασία τυπώνει μήνυμα και τερματίζει όταν τερματιστούν όλα της τα παιδιά (κανει wait() για κάθε παιδί). Επομένως, επειδή δε γνωρίζουμε τη σειρά με την οποία θα συμβεί αυτό, και τα μηνύματα τερματισμού είναι τυχαία. (σίγουρα βέβαια τα παιδιά θα βγάλουν μήνυμα τερματισμού νωρίτερα από τους γονείς τους)
Πιο συγκεκριμένα, στο δικό μας δέντρο:

- σίγουρα πριν τερματίσει ο B θα εμφανίσουν μήνυμα τερματισμού ο F και ο E
- σίγουρα πριν τερματίσει ο A θα εμφανίσουν μήνυμα τερματισμού ο A ο B και ο C.

1.3 Αποστολή και Χειρισμός Σημάτων

Στην άσκηση αυτή θα υλοποιήσουμε παρόμοιο πρόγραμμα με το προηγούμενο μόνο που εδώ το δέντρο θα δημιουργείται με DFS σειρά καθορισμένα. Δε θα υπάρχει δηλαδή τίποτα στην τύχη. Για να το επιτύχουμε αυτό θα χρησιμοποιήσουμε σήματα. Πιο συγκεκριμένα:

- οι εργασίες φύλλα κάνουν raise SIGSTOP , που θα πει ο,τι περιμένουν σήμα SIGCONT για να συνεχίσουν τη λειτουργία τους και να τερματίσουν.
- οι διεργασίες που έχουν παιδιά δημιουργούν τις διεργασίες-παιδιά τους και κάνουν raise SIGSTOP περιμένοντας απο το δικό τους γονιό σήμα SIGCONT και μόλις το λάβουν στέλνουν σε κάθε μία διεργασία-παιδί τους διαδοχικα σήμα SIGCONT , ώστε να τερματισει (αφού τερματίσουν με τον ίδιο τρόπο-αναδρομικά- τα δικά της παιδιά) και στη συνέχεια για να ολοκληρωθούν και αυτές(αφού κάνουν wait)

Έτσι επιτυγχάνεται και η αφύπνιση των εργασιών σε DFS σειρά.
Ο κώδικας που υλοποιεί όσα περιγράφησαν είναι ο ακόλουθος:

```
#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"
void let_it_fork(struct tree_node * root);
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 */
		/* root of my tree */
		/*for every child I enter this if and
		decide in let_it_fork if it is a father of another child
		to fork again until I reach leaves*/
		let_it_fork(root);
		exit(0);
	}
	/*
	 * 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);
	kill(pid, SIGCONT);
	/* Wait for the root of the process tree to terminate */
	wait(&status);
	explain_wait_status(pid, status);
	return 0;
}
void let_it_fork(struct tree_node * root)
{
	change_pname(root->name); //gia na allaksoume to id ka8e diadikasias sto epi8ymhto onoma
	int i;
	int status;
	pid_t p[root->nr_children];
//create an array to hold all my children's PIDs so as to send SIGCONT
	if (root->nr_children == 0){
		printf("Process %s started. It's a leaf\n",root->name);
		printf("Process %s: Waiting for SIGCONT\n",root->name);
		raise(SIGSTOP);
		printf("PID = %ld, name = %s is awake\n",
		(long)getpid(), root->name);
		printf("Process %s: Exiting\n",root->name);
		exit(0);
	}
	else{
		//periptwsh pou exoume paidia, prepei na ta kanoume fork
		pid_t pid;
		printf("Process %s started creating children\n",root->name);
		for (i=0;i<root->nr_children;i++){
		      pid=fork();
		       if (pid<0){
				perror("problem with fork\n");
				exit(1);
		       }
		       if (pid == 0){
				let_it_fork(root->children+i);		//child, recursive call gia ka8e pointer se
									//child
				printf("Process %s : Exiting\n",root->name);
			       exit(0);
		       }
		       else{   //so as to enter only for the parent
				p[i]=pid;
		       }
		       wait_for_ready_children(1); //waits for every child
		}
		//wait_for_ready_children(root->nr_children);
		printf("PID= %ld, name = %s is waiting for SIGCONT\n",(long)getpid(),root->name);
		raise(SIGSTOP); //waiting for SIGCONT
```

```
            printf("PID = %ld, name = %s is awake\n",
            (long)getpid(), root->name);
            //edw prepei na steilei SIGCONT diadoxika sta paidia ths
            for(i=0;i<root->nr_children;i++){
                  kill(p[i],SIGCONT);
                  wait(&status);
                  explain_wait_status(p[i],status);
            }
//          pid=wait(&status);
//          explain_wait_status(pid,status);
            exit(0);
      }
}
```

Ο παραπάνω κώδικας, όταν λάβει σαν όρισμα το αρχείο proc.tree που περιέχει ένα δέντρο δίνει τα παρακάτω:

```
oslabd01@zakynthos:~/Askhsh2/Askhsh1.3$ ./check2 proc.tree
Process A started creating children
Process B started creating children
Process E started. It's a leaf
Process E: Waiting for SIGCONT
My PID = 15442: Child PID = 15443 has been stopped by a signal, signo = 19
Process F started. It's a leaf
Process F: Waiting for SIGCONT
My PID = 15442: Child PID = 15444 has been stopped by a signal, signo = 19
PID= 15442, name = B is waiting for SIGCONT
My PID = 15441: Child PID = 15442 has been stopped by a signal, signo = 19
Process C started. It's a leaf
Process C: Waiting for SIGCONT
My PID = 15441: Child PID = 15445 has been stopped by a signal, signo = 19
Process D started. It's a leaf
Process D: Waiting for SIGCONT
My PID = 15441: Child PID = 15446 has been stopped by a signal, signo = 19
PID= 15441, name = A is waiting for SIGCONT
My PID = 15440: Child PID = 15441 has been stopped by a signal, signo = 19


A(15441)─┬─B(15442)─┬─E(15443)
         │          └─F(15444)
         ├─C(15445)
         └─D(15446)


PID = 15441, name = A is awake
PID = 15442, name = B is awake
PID = 15443, name = E is awake
Process E: Exiting
My PID = 15442: Child PID = 15443 terminated normally, exit status = 0
PID = 15444, name = F is awake
Process F: Exiting
My PID = 15442: Child PID = 15444 terminated normally, exit status = 0
My PID = 15441: Child PID = 15442 terminated normally, exit status = 0
PID = 15445, name = C is awake
Process C: Exiting
My PID = 15441: Child PID = 15445 terminated normally, exit status = 0
PID = 15446, name = D is awake
Process D: Exiting
My PID = 15441: Child PID = 15446 terminated normally, exit status = 0
My PID = 15440: Child PID = 15441 terminated normally, exit status = 0
```

1. Στις προηγούμενες ασκήσεις χρησιμοποιήσαμε τη sleep() για το συγχρονισμό των διεργασιών. Τι πλεονεκτήματα έχει η χρήση σημάτων;

Η λειτουργία sleep() στην ουσία αυτό που κάνει είναι να “κοιμίζει” τις διεργασίες-φύλλα για δεδομένο χρονικό διάστημα (που της δίνεται σαν όρισμα). Αυτό έχει ως αποτέλεσμα στην ουσία να μη μπορούμε να ελέγξουμε επακριβώς τη σειρά με την οποία “αφυπνούνται” οι διεργασίες. Εδώ λοιπόν υπάρχει τυχαιότητα. Στον αντίποδα αυτού, με τη χρήση σημάτων γίνονται όλα ντετερμινιτικά. Ο χρήστης είναι αυτός που καθορίζει τη σειρά δημιουργίας και αφυπνισης των διεργασιών. Μπορεί έτσι να είναι σίγουρος για τα πάντα. Αυτό είναι το βασικό πλεονέκτημα της χρήσης σημάτων. Αυτός ο συγχρονισμός με σήματα είναι ιδιαίτερα χρήσιμος σε πολλές περιπτώσεις, όπως για παράδειγμα αν είναι αναγκαίο μια διεργασία να εκκινήσει ή να πεθάνει πριν εκκινήσει ή πεθάνει κάποια άλλη (αν θέλουμε συγκεκριμένη σειρά δηλαδή) ή αν η λειτουργία μιας διαδικασίες απαιτεί την ύπαρξη κάποιας άλλης ώστε να λειουργήσει σωστά (αναμένει αυτή τη διεργασία δηλαδή).

2.Ποιος ο ρόλος της wait_for_ready_children(); Τι εξασφαλίζει η χρήση της και τι πρόβλημα θα δημιουργούσε η παράλειψή της;

Η λειτουργία της συνάρτησης αυτής είναι η εξής: Κάθε διεργασία-γονιός καλεί την wait_for_ready_children() και με αυτή την κλήση περιμένει όλα του τα παιδιά (ο αριθμός των οποίων δίνεται σαν όρισμα στην wait_for_ready_children(number of children)) να κάνουν raise SIGSTOP προκειμένου να συνεχίσει η διεργασία-γονιός να εκτελεί τις επόμενες εντολές της. Στο πρόγραμμά μας καλούμε την wait_for_ready_children() με όρισμα 1 καθώς είναι μέσα σε for_loop επομένως στην ουσία θα αναμείνει για raise SIGSTOP όσων παιδιών αυτή δημιουργήσει (το 1 μετά το άλλο θα κάνει raise SIGSTOP, αφού το κάθε ένα περιμένει τα δικά του παιδια να κάνουν raise SIGSTOP κοκ, έτσι έχουμε **δημιουργία σε DFS σειρά**). Με αυτήν την τακτική καταφέρνουμε και να αφυπνήσουμε τους κόμβους του δέντρου διεργασιών με DFS σειρά. Αυτό επιτυγχάνεται με το να κρατάμε τα pid των παιδιών κάθε κόμβου σε πίνακα και αφού ο γονιός δεχτεί σήμα SIGCONT από το δικό του γονιό, τότε και αυτός με τη σειρά του να στέλνει στις διεργασίες-παιδιά του (των οποιων το pid έχει κρατήσει στον πίνακα) με ένα for_loop σήμα SIGCONT. Έτσι γίνεται η DFS διάσχιση του δέντρου.

Αν παραλείψουμε αυτή τη συνάρτηση θα υπάρξει πρόβλημα. Υπάρχει περίπτωση κάποια διεργασία-πρόγονος να πεθάνει πριν τη διεργασία που την έχει ως πρόγονο ή ακομή και πριν τη δημιουργία αυτής. Δεν εξασφαλίζεται λοιπόν η DFS διάσχιση και επίσης καθίσταται αδύνατη η παρατήρηση του σωστού δέντρου διεργασιών καθώς πιθανότατα δε θα είναι διαθέσιμοι όλοι οι κόμβοι του δέντρου όταν καλέσουμε τη show_pstree().

1.4 Παράλληλος υπολογισμός αριθμητικής έκφρασης

Σε αυτή την άσκηση καλούμαστε να υπολογίσουμε την τιμή μιας αριθμητικής έκφρασης κάνοντας χρήση της σωλήνωσης. Οι αριθμοί και οι τελεστές βρίσκονται σε ένα δέντρο το οποίο διαβάζουμε με την ίδια συνάρτηση που χρησιμοποιήσαμε και στην προηγούμενη άσκηση. Διασχίζουμε, όπως και στην προηγούμενη άσκηση, το δέντρο και δημιουργούμε μαζί με τις διεργασίες και ένα pipe με το οποίο θα επικονωνεί ο πατέρας με το παιδί του. (1 pipe για κάθε παιδί, γίνεται λόγω των συνθηκών της άσκησης αν χρησιμοποιηθεί και 1 μόνο pipe και για τα 2 παιδιά, όπως εξηγείται και στην 1η ερώτηση) Στο pipe αυτό γράφει κάθε παιδί τον αριθμό που περιέχει (αν είναι φύλλο) ή το αποτέλεσμα που προκύπτει από την εκτέλεση της πράξης <1ο παιδι><τελεστής γονιού><2ο παιδί>. Αυτό γίνεται σε κάθε επίπεδο μέχρις ότου φτάσουμε στη ρίζα όπου και έχουμε το τελικό μας αποτέλεσμα. Αυτό γίνεται επιτυχώς, καθώς το pipe είναι FIFO δομή,επομένως ο πατέρας διαβάζει απο το pipe τους 2 αριθμούς που γράφουν τα παιδιά του, τον ένα μετά τον άλλο (ακόμη και request για read να κάνει ο γονιός και να είναι άδειο το pipe δε μας πειράζει καθώς όπως γνωρίζουμε δε θα αποτύχει απλά θα διαβάσει την 1η τιμή που θα μπει στο συγκεκριμένο pipe κάποια στιγμή, επομένως έτσι επιτυγχάνεται αυτόματα ο συγχρονισμός αφού το read απο το pipe δουλεύει κατά αυτόν τον τρόπο) και εκτελεί την κατάλληλη πράξη, στέλνει το αποτέλεσμα με τον ίδιο τρόπο στο δικό του γονιό κοκ. Επομένως όταν η ρίζα έχει το αποτέλεσμα, το γράφει στο δικό της pipe με το οποίο επικοινωνεί με τη main και εκείνη το τυπώνει. Με αυτή την ιδιότητα του pipe δεν είμαστε αναγκασμένοι να χρησιμοποιήσουμε σήματα στην άσκηση αυτή (παρόλ' αυτά θα παραθέσω στον τέλος και έναν κώδικα που υλοποιεί τα ίδια με τον παρακάτω μόνο που ο συγχρονισμός στηρίζεται στη χρήση σημάτων (δε απαιτούταν απο την άσκηση). Ο δεύτερος αυτός κώδικας βασίστηκε στην προηγούμενη άσκηση)

Ο κώδικας που υλοποιεί όσα περιγράφησαν είναι ο ακόλουθος:

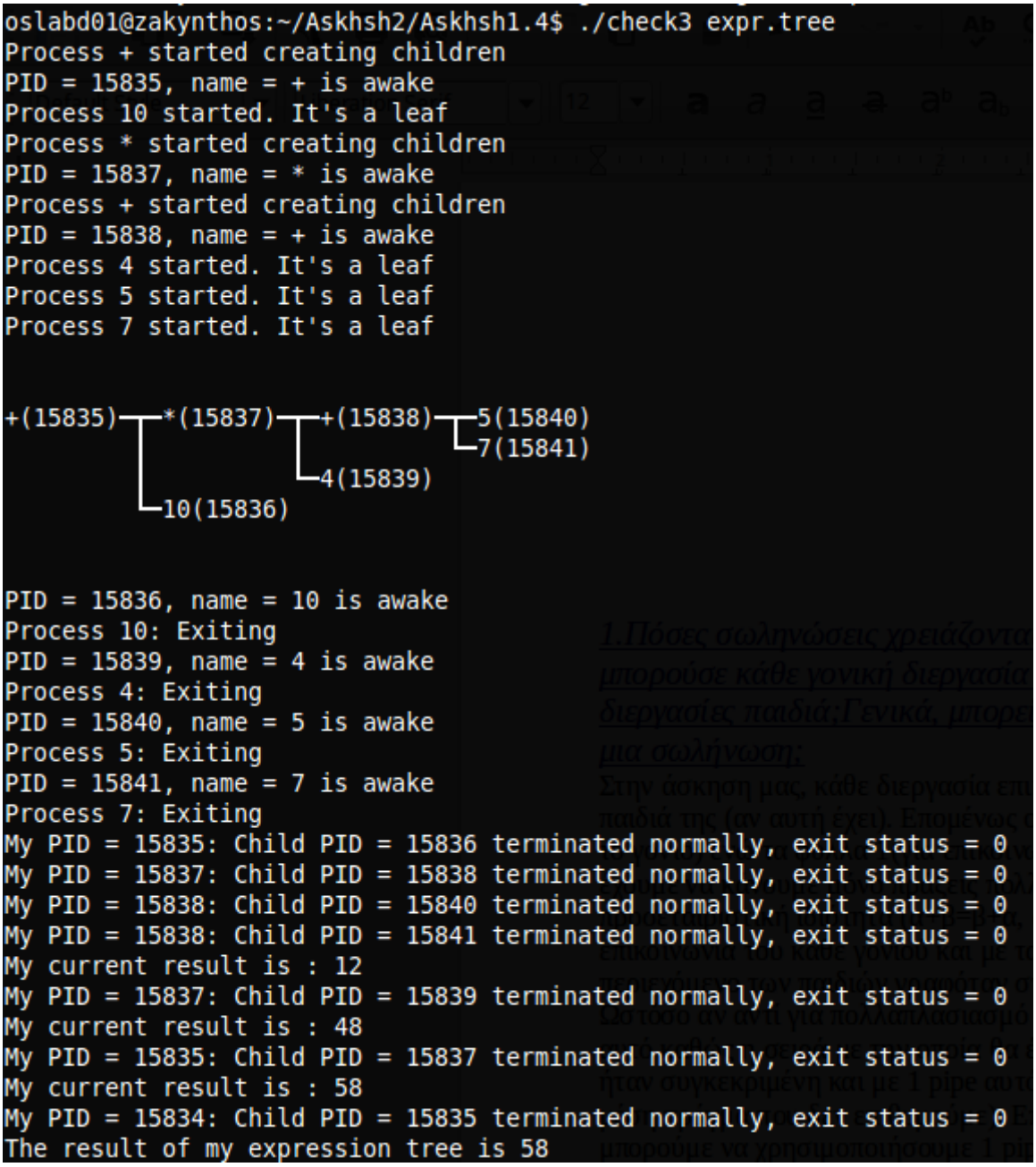
```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/wait.h>

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

void let_it_fork(struct tree_node * root,int []);

int main(int argc, char *argv[])
{
	int calc;
	pid_t pid;
	int status;
	struct tree_node *root;
	int filedes[2]; //variable for pipe
	//one cell to read one to write
	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]);
```

```
	//create the pipe
	if(pipe(filedes)<0){
		perror("pipe");
		exit(1);
	}


	/* Fork root of process tree */
	pid = fork();
	if (pid < 0) {
		perror("main: fork");
		exit(1);
	}
	if (pid == 0) {
		/* Child */
		/* root of my tree */
		close(filedes[0]); //child is write-only
		let_it_fork(root,filedes);
		exit(0);
	}
	close(filedes[1]); //father is read-only
	/*
	 * Father
	 */
	/* for ask2-signals */
	/////////wait_for_ready_children(1);
	/* for ask2-{fork, tree} */
	/* sleep(SLEEP_TREE_SEC); */
	/* Print the process tree root at pid */
	sleep(3);
	show_pstree(pid);
	////////kill(pid, SIGCONT);
	/* Wait for the root of the process tree to terminate */
	wait(&status);
	explain_wait_status(pid, status);
	if (read(filedes[0],&calc,sizeof(calc)) != sizeof(calc)){ //read from the pipe the final result
			perror("read from pipe\n");
			exit(1);
		}

	printf("The result of my expression tree is %d\n",calc);
	return 0;
}



void let_it_fork( struct tree_node * root, int* filedes )
{	int filedesnew[2];
	int res;
	int num1,num2; //variables to read numbers
	//definition of the necessary variables
	change_pname(root->name); //gia na allaksoume to id ka8e diadikasias sto epi8ymhto onoma
```

```
        int i;
        int status;
        pid_t p[root->nr_children];
//create an array to hold all my children's PIDs so as to send SIGCONT
        int num;
        if (root->nr_children == 0){
                printf("Process %s started. It's a leaf\n",root->name);
        ////    printf("Process %s: Waiting for SIGCONT\n",root->name);
               ////////////// raise(SIGSTOP);
                sleep(10);
                printf("PID = %ld, name = %s is awake\n",
                (long)getpid(), root->name);
                //computation with pipes
                //its a leaf so write to the pipe
                num =  atoi(root->name);//convert string to its equivalent integer
                if (write(filedes[1],&(num),sizeof(num)) != sizeof(num)) {
                        perror("write to pipe\n");
                        exit(1);
                }
                //printf("number %d\n",num);
                printf("Process %s: Exiting\n",root->name);
                exit(0);
        }
        else{
                //create the new pipe for parent and his children
                if(pipe(filedesnew)<0){
                        perror("pipe");
                        exit(1);
                }
                //periptwsh pou exoume paidia, prepei na ta kanoume fork
                pid_t pid;
                printf("Process %s started creating children\n",root->name);
                for (i=0;i<root->nr_children;i++){ //max na treksei 2 fores afou exoume 2  paidia  h
                                                   //kanena
                       pid=fork();
                        if (pid<0){
                                perror("problem with fork\n");
                                exit(1);
                        }
                        if (pid == 0){
                                close(filedesnew[0]); //child is write-only
                                let_it_fork(root->children+i,filedesnew);          //child, recursive call gia ka8e
                                                                                  //pointer se child
                                printf("Process %s : Exiting\n",root->name);
                               exit(0);
                        }
                        else{   //so as to enter only for the parent
                                p[i]=pid;
                        }
                        //////////wait_for_ready_children(1); //waits for every child
                }
                close(filedesnew[1]);//father is read-only
```

```
            //wait_for_ready_children(root->nr_children);
        //////  printf("PID= %ld, name = %s is waiting for SIGCONT\n",(long)getpid(),root->name);
            ////////////raise(SIGSTOP); //waiting for SIGCONT
            printf("PID = %ld, name = %s is awake\n",
            (long)getpid(), root->name);


            //edw prepei na steilei SIGCONT diadoxika sta paidia ths kai na perimenei na teleiwsoun
          //o,ti exoun na kanoun
            for(i=0;i<root->nr_children;i++){
                ////////////kill(p[i],SIGCONT);
                wait(&status);
                explain_wait_status(p[i],status);
            }
            //computations with pipes
            //read numbers pou exoun grapsei sto pipe ta paidia tou
            if (read(filedesnew[0], &num1 , sizeof(num1)) != sizeof(num1)){
                perror("read from pipe\n");
                exit(1);
            }
            if (read(filedesnew[0],&num2,sizeof(num2)) != sizeof(num2)){
                perror("read from pipe\n");
                exit(1);
            }
            //printf("\n\n\n %d,,,,,%d\n\n\n",num1,num2);
            if (strcmp(root->name,"*")==0){
                res=num1*num2;
                //(root->name)=itoa(res);

            }
            else if (strcmp(root->name,"+")==0){
                res=num1+num2;
            }
            printf("\ncurrent result: %d \n",res);
            //write to result tou computation sto pipe pou 8a dei o pateras sou
            if (write(filedes[1],&res,sizeof(res)) != sizeof(res)) {
                        perror("write to pipe\n");
                        exit(1);
            }

//          pid=wait(&status);
//          explain_wait_status(pid,status);
            exit(0);
        }
    }
```

Ο παραπάνω κώδικας, όταν λάβει σαν όρισμα το αρχείο expr.tree που περιέχει ένα δέντρο δίνει τα παρακάτω:



Παρατηρούμε λοιπόν το δεετρο διεργασιών, τη σειρά με την οποία γίνεται προσπέλαση στους κόμβους και όλα τα ενδιάμεσα αποτελέσματα της αριθμητικής μας έκφρασης όπως αυτά υπολογίζονται σε κάθε βήμα (12,48,58).

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

Στην άσκηση μας, κάθε διεργασία επικοινωνεί μέσω ενός pipe με το γονιό της και με άλλα 2 με τα παιδιά της (αν αυτή έχει). Επομένως οι ενδιάμεσοι κόμβοι απαιτούν 3 pipes(2 για τα παιδιά, 1 για το γονιό) ενώ τα φύλλα 1(για επικοινωνία με το γονιό τους). Ωστόσο, επειδή στην άσκησή μας έχουμε να κάνουμε μόνο πράξεις πολλαπλασιασμού και πρόσθεσης, για τις οποίες ισχύει η προσεταιριστική ιδιότητα (α+β=β+α, α*β=β*α), θα μπορούσαμε να χρησιμοποιήσουμε 1 pipe για επικοινωνία του κάθε γονιού και με τα 2 παιδιά του αφού ανεξάρτητα με τη σειρά που το περιεχόμενο των παιδιών γραφόταν στο pipe το αποτέλεσμα της πράξης θα ήταν τελικά το ίδιο. Ωστόσο αν αντί για πολλαπλασιασμό και προσθέσεις είχαμε πχ αφαίρεση ή διαίρεση, δε θα γινόταν αυτό καθώς η σειρά με την οποία θα έπρεπε να μπουν και να διαβαστούν οι αριθμοί στο pipe θα ήταν συγκεκριμένη και με 1 pipe αυτό δεν είναι εφικτό (θα έπαιζε πάλι ο παράγοντας τύχη στη μέση πράγμα που δεν επιθυμούμε). Επομένως για ένα δέντρο που περιέχει και τέτοιους operands δε μπορούμε να χρησιμοποιήσουμε 1 pipe για όλα τα παιδιά αλλά θέλουμε 1 για το καθένα προκειμένου να εκτελεστούν σωστά οι πράξεις.

2.Σε ένα σύστημα πολλαπλών επεξεργαστών, μπορούν να εκτελουνται 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;

Η αποτίμηση μιας έκφρασης μέσω δέντρου θα ήταν προφανώς καλύτερη σε ένα τέτοιο σύστημα. Αυτό ισχύει, γιατί με το να έχουμε στη διάθεσή μας πολλούς επεξεργαστές μπορούμε παράλληλα (χρονικά) να αναθέτουμε 1 κλάδο (ή υποκλάδο) ενός δέντρου σε κάθε επεξεργαστή και έτσι οι πράξεις κάθε υποδέντρου θα γίνονται παράλληλα. (και όχι σειριακά όπως θα γίνονταν απο μία μόνο διεργασία). Έτσι εξοικονομείται αρκετός χρόνος, ιδίως στην περίπτωση που έχουμε κάποια πολύ περίπλοκη έκφραση (που αποτελείται από διάφορους και μεγάλους κλάδους).

Ο κώδικας που υλοποιεί τα ίδια με τον παραπάνω αλλά επιτυγχάνει το συγχρονισμό με χρήση σημάτων (που πρακτικά δε χρειάζεται) είναι ο ακόλουθος:

```
#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"

void let_it_fork(struct tree_node * root,int []);

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

```
	int filedes[2]; //variable for pipe
	//one cell to read one to write
	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]);
	//create the pipe
	if(pipe(filedes)<0){
		perror("pipe");
		exit(1);
	}


	/* Fork root of process tree */
	pid = fork();
	if (pid < 0) {
		perror("main: fork");
		exit(1);
	}
	if (pid == 0) {
		/* Child */
		/* root of my tree */
		close(filedes[0]); //child is write-only
		let_it_fork(root,filedes);
		exit(0);
	}
	close(filedes[1]); //father is read-only
	/*
	 * Father
	 */
	/* for ask2-signals */
	wait_for_ready_children(1);
	/* for ask2-{fork, tree} */
	/* sleep(SLEEP_TREE_SEC); */
	///* Print the process tree root at pid */
	sleep(3);
	show_pstree(pid);
	kill(pid, SIGCONT);
	/* Wait for the root of the process tree to terminate */
	wait(&status);
	explain_wait_status(pid, status);
	if (read(filedes[0],&calc,sizeof(calc)) != sizeof(calc)){ //read from the pipe the final result
			perror("read from pipe\n");
			exit(1);
		}

	printf("The result of my expression tree is %d\n",calc);
	return 0;
}
```

```
void let_it_fork( struct tree_node * root, int* filedes )
{       int filedesnew[2];
        int res;
        int num1,num2; //variables to read numbers
        //definition of the necessary variables
        change_pname(root->name); //gia na allaksoume to id ka8e diadikasias sto epi8ymhto onoma
        int i;
        int status;
        pid_t p[root->nr_children];
//create an array to hold all my children's PIDs so as to send SIGCONT
        int num;
        if (root->nr_children == 0){
                printf("Process %s started. It's a leaf\n",root->name);
                printf("Process %s: Waiting for SIGCONT\n",root->name);
                //sleep(10);
                raise(SIGSTOP);
                printf("PID = %ld, name = %s is awake\n",
                (long)getpid(), root->name);
                //computation with pipes
                //its a leaf so write to the pipe
                num =  atoi(root->name);//convert string to its equivalent integer
                if (write(filedes[1],&(num),sizeof(num)) != sizeof(num)) {
                        perror("write to pipe\n");
                        exit(1);
                }
                //printf("number %d\n",num);
                printf("Process %s: Exiting\n",root->name);
                exit(0);
        }
        else{
                //create the new pipe for parent and his children
                if(pipe(filedesnew)<0){
                        perror("pipe");
                        exit(1);
                }
                //periptwsh pou exoume paidia, prepei na ta kanoume fork
                pid_t pid;
                printf("Process %s started creating children\n",root->name);
                for (i=0;i<root->nr_children;i++){ //max na treksei 2 fores afou exoume 2  paidia  h
                                                        //kanena
                      pid=fork();
                       if (pid<0){
                              perror("problem with fork\n");
                              exit(1);
                       }
                       if (pid == 0){
                              close(filedesnew[0]); //child is write-only
                              let_it_fork(root->children+i,filedesnew);         //child, recursive call gia ka8e
                                                                  //pointer se child
```

```
                printf("Process %s : Exiting\n",root->name);
                exit(0);
            }
            else{   //so as to enter only for the parent
                p[i]=pid;
            }
            wait_for_ready_children(1); //waits for every child
        }
        close(filedesnew[1]);//father is read-only
        //wait_for_ready_children(root->nr_children)
        printf("PID= %ld, name = %s is waiting for SIGCONT\n",(long)getpid(),root->name);
        raise(SIGSTOP); //waiting for SIGCONT
        printf("PID = %ld, name = %s is awake\n",
        (long)getpid(), root->name);


        //edw prepei na steilei SIGCONT diadoxika sta paidia ths kai na perimenei na teleiwsoun
            //o,ti exoun na kanoun
        for(i=0;i<root->nr_children;i++){
            kill(p[i],SIGCONT);
            wait(&status);
            explain_wait_status(p[i],status);
        }
        //computations with pipes
        //read numbers pou exoun grapsei sto pipe ta paidia tou
        if (read(filedesnew[0], &num1 , sizeof(num1)) != sizeof(num1)){
            perror("read from pipe\n");
            exit(1);
        }
        if (read(filedesnew[0],&num2,sizeof(num2)) != sizeof(num2)){
            perror("read from pipe\n");
            exit(1);
        }
        //printf("\n\n\n %d,,,,,%d\n\n\n\n",num1,num2);
        if (strcmp(root->name,"*")==0){
            res=num1*num2;
            //(root->name)=itoa(res);
        }
        else if (strcmp(root->name,"+")==0){
            res=num1+num2;
        }
        printf("My current result is : %d \n",res);
        //write to result tou computation sto pipe pou 8a dei o pateras sou
        if (write(filedes[1],&res,sizeof(res)) != sizeof(res)) {
                perror("write to pipe\n");
                exit(1);
        }

//          pid=wait(&status);
//          explain_wait_status(pid,status);
        exit(0);
    }
```

}
Σημείωση: η πληθώρα σχολίων χρησιμοποιήθηκαν για λόγους debugging

Αν εκτελέσουμε τον παραπάνω κώδικα με όρισμα το αρχείο expr.tree έχουμε τα αποτελέσματα:

```
oslabd01@zakynthos:~/Askhsh2/Askhsh1.4$ ./check4 expr.tree
Process + started creating children
Process 10 started. It's a leaf
Process 10: Waiting for SIGCONT
My PID = 15874: Child PID = 15875 has been stopped by a signal, signo = 19
Process * started creating children
Process + started creating children
Process 5 started. It's a leaf
Process 5: Waiting for SIGCONT
My PID = 15877: Child PID = 15878 has been stopped by a signal, signo = 19
Process 7 started. It's a leaf
Process 7: Waiting for SIGCONT
My PID = 15877: Child PID = 15879 has been stopped by a signal, signo = 19
PID= 15877, name = + is waiting for SIGCONT
My PID = 15876: Child PID = 15877 has been stopped by a signal, signo = 19
Process 4 started. It's a leaf
Process 4: Waiting for SIGCONT
My PID = 15876: Child PID = 15880 has been stopped by a signal, signo = 19
PID= 15876, name = * is waiting for SIGCONT
My PID = 15874: Child PID = 15876 has been stopped by a signal, signo = 19
PID= 15874, name = + is waiting for SIGCONT
My PID = 15873: Child PID = 15874 has been stopped by a signal, signo = 19


+(15874)─┬─*(15876)─┬─+(15877)─┬─5(15878)
         │          │          └─7(15879)
         │          └─4(15880)
         └─10(15875)


PID = 15874, name = + is awake
PID = 15875, name = 10 is awake
Process 10: Exiting
My PID = 15874: Child PID = 15875 terminated normally, exit status = 0
PID = 15876, name = * is awake
PID = 15877, name = + is awake
PID = 15878, name = 5 is awake
Process 5: Exiting
My PID = 15877: Child PID = 15878 terminated normally, exit status = 0
PID = 15879, name = 7 is awake
Process 7: Exiting
My PID = 15877: Child PID = 15879 terminated normally, exit status = 0
My current result is : 12
My PID = 15876: Child PID = 15877 terminated normally, exit status = 0
PID = 15880, name = 4 is awake
Process 4: Exiting
My PID = 15876: Child PID = 15880 terminated normally, exit status = 0
My current result is : 48
My PID = 15874: Child PID = 15876 terminated normally, exit status = 0
My current result is : 58
My PID = 15873: Child PID = 15874 terminated normally, exit status = 0
The result of my expression tree is 58
```

Είναι εμφανής η DFS αφύπνιση του δέντρου (όπως έγινε και στην άσκηση 3).
Το αποτέλεσμα και με αυτόν τον κώδικα είναι σωστό. Ωστόσο τα pipes μας κάνουν τη ζωή πιο εύκολη και δε χρειάζεται ο χειρσμός των διεργασιών μέσω σημάτων απλώς τον παρουσιάσαμε για λόγους πληρότητας.