Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών

Λειτουργικά Συστήματα Υπολογιστών

2η Εργαστηριακή Άσκηση

Διαχείριση Διεργασιών και Διαδιεργασιακή Επικοινωνία

Ροή: Υ

Ομάδα: Α05

Μέλη: Μιχαήλ Σωτήρης *03113719*

Νικολαΐδης Ανδρέας *03113715*

Εξάμηνο: 7ο

Ακαδ. έτος: 2016-2017

Άσκηση 1.1 – : Δημιουργία δεδομένου δέντρου εργασιών

```
#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.
         */
        int status;

        change_pname("A");          //Ονομασία διεργασίας A
        printf("A CREATED WITH PID = %ld\n",(long)getpid());

        pid_t childB = fork();          //Δημιουργία παιδιού B. Η συνάρτηση fork επιστρέφει 0
        στη διεγασία του παιδιού, το PID του παιδιού στη διεργασία του πατέρα και αρνητικό
        αριθμό σε περίπτωση σφάλματος.


        if (childB < 0) {               //Λάθος στη δημιουργία του B
                perror("main: fork");
                exit(1);
        }
        if (childB == 0) {              //Το κομμάτι αυτό θα εκτελεστεί από τη διεργασία B.
                change_pname("B");  //Ονομασία διεργασίας B
                printf("B CREATED WITH PID = %ld\n",(long)getpid());
                //Δημιουργία παιδιούD
                pid_t childD = fork ();
                if (childD < 0) {       //Σφάλμα στη  fork
```

```
                perror("main: fork");
                exit(1);
                }
        if (childD == 0) {     //Το κομμάτι αυτό θα εκτελεστεί από τη διεργασία D.
                change_pname("D");
                printf("D CREATED WITH PID = %ld\n",(long)getpid());
                printf("D: Sleeping...\n");
                sleep(SLEEP_PROC_SEC);
                printf("D: Exiting...\n");
                exit(13);
        }
        printf("Parent, PID = %ld: Created child with PID = %ld, waiting for it to
terminate...\n",(long)getpid(), (long)childD);

        printf("B: Sleeping...\n");
        sleep(SLEEP_PROC_SEC);
        childD = wait(&status); //Αναμονή για τερματισμό του D
        explain_wait_status(childD, status);
        printf("B: Exiting...\n");
        exit(19);
}
//Το κομμάτι αυτό θα εκτελεστεί από τον A
printf("Parent, PID = %ld: Created child with PID = %ld, waiting for it to
terminate...\n",(long)getpid(), (long)childB);

pid_t childC = fork(); //Δημιουργία του C
if (childC < 0) { //Σφάλμα στη fork
        perror("main: fork");
        exit(1);
}
if (childC == 0) {
        change_pname("C"); //Ονομασία του C
        printf("C CREATED WITH PID = %ld\n",(long)getpid());
        printf("C: Sleeping...\n");
        sleep(SLEEP_PROC_SEC);
        printf("C: Exiting...\n");
        exit(17);
        }
printf("Parent, PID = %ld: Created child with PID = %ld, waiting for it to
terminate...\n",(long)getpid(), (long)childC);

printf("A: Sleeping...\n");
sleep(SLEEP_PROC_SEC);
```

```
	//Αναμονή για τερματισμό των παιδιών
	childC = wait(&status);
	explain_wait_status(childC, status);
	childB = wait(&status);
	explain_wait_status(childB, status);

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

int main(void)
{
	pid_t pid;
	int status;

	/* Fork root of process tree */
	pid = fork(); //Δημιουργία του A
	if (pid < 0) {
		perror("main: fork");
		exit(1);
	}
	if (pid == 0) { //Αυτό θα εκτελεστεί από την A
		/* Child */
		fork_procs();
		exit(1);
	}


	/* waits for the process tree to be completely created*/
	sleep(SLEEP_TREE_SEC);

	/* takes a photo of the tree*/
	show_pstree(pid);

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

	return 0;
}
```

Ερωτήσεις:

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

   Αν τερματιστεί πρόωρα η διεργασία Α, οι διεργασίες-παιδιά της θα γίνουν παιδιά της init η οποία βρίσκεται συνεχώς σε κατάσταση αναμονής εκτελώντας συνεχώς wait(). Έτσι έχουμε τις ακόλουθες επιπτώσεις ανάλογα με το τι πρόλαβε να εκτελέσει η Α προτού τερματιστεί:
   ->Αν η διεργασία Α δεν έχει προλάβει να κάνει fork τις διεργασίες Β, C δε θα γεννηθεί ούτε και η D και ούτε θα τυπωθούν τα μηνύματα αυτών.
   ->Αν η διεργασία Α πρόλαβε να κάνει fork την Β αλλά όχι την C τότε θα τυπωθούν τα μηνύματα της Β και κατ' επέκταση της D, αλλά δε θα γεννηθεί και δε θα τυπώσει μηνύματα η C. Ακόμη, όταν η Α πεθάνει, η Β θα υιοθετηθεί από την init(1) και αντί να επιστρέψει στην Α, θα επιστρέψει στο νέο της πατέρα. Η D δεν αντιλαμβάνεται κάποια διαφορά από το θάνατο της Α καθώς έχει να κάνει μόνο με το δικό της πατέρα, την Β.
   ->Αν η διεργασία πρόλαβε να κάνει fork και τη Β και τη C τότε θα γεννηθεί και η D. Θα τυπωθούν τα μηνύματα και των τριών διεργασιών. Όμως όταν η Α πεθαίνει , τα παιδιά της (Β και C) υιοθετούνται από την init(1) και επιστρέφουν σε αυτή αντί της Α. Η D και πάλι δεν αντιλαμβάνεται κάποια διαφορά.

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

   Τώρα κάνουμε show_pstree(pid) το οποίο εκτελείται από τη main. Η main έχει στη μεταβλητή pid το processid του παιδιού της, δηλαδή της Α. Η κλήση show_pstree(pid) θα μας δείξει το δέντρο διεργασιών με ρίζα το Α. Δεν μας δείχνει τυχόν άλλα πράγματα που εκτελεί η main και τις αντίστοιχες διεργασίες.

   A(5996)─┬─B(5997)──D(5999)
   └─C(5998)

   Αν τώρα κάνουμε show_pstree(getpid()) τότε θα δούμε το δέντρο διεργασιών με ρίζα αυτό που επιστρέφει η getpid() , δηλαδή το ίδιο το πρόγραμμα που τρέχουμε ή αλλιώς το ask2_1a. Στη προκειμένη , θα δούμε το ask2_1a να έχει ως παιδί το Α και αυτό ακολούθως τα Β, C, D ως γνωστόν , αλλά επίσης η ask2_1a θα έχει ακόμη ένα παιδί, που αντιστοιχεί στη διεργασία που δημιουργήθηκε με σκοπό να εκτελεστεί η pstree συνάρτηση από το proc-common. Και επειδή αυτό αποτελεί κλήση , η ask2_1a έχει ως

παιδί μια διεργασία με όνομα "sh" δηλαδή shell (φλοιός) η οποία με τη σειρά της δημιουργεί μια διεργασία-κλήση της pstree().

```
ask2_1a (5995)─┬─A(5996)─┬─B(5997)──D(5999)
               |         └─C(5998)
               └─sh(6002)──pstree(6003)
```

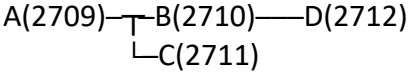
3. ***Σε υπολογιστικά συστήματα πολλαπλών χρηστών, πολλές φορές ο διαχειριστής θέτει όρια στον αριθμό των διεργασιών που μπορεί να δημιουργήσει ένας χρήστης. Γιατί;***

Θέτοντας όριο στις διεργασίες που δημιουργούνται από κάθε χρήστη ο διαχειριστής επιτυγχάνει με αυτό το τρόπο ένα είδος μηχανισμού ελέγχου και περιορισμού της υπολογιστικής ισχύος που χρησιμοποιείται από κάθε χρήστη με αποτέλεσμα μια πιο δίκαιη κατανομή της ισχύος.

Παράδειγμα εκτέλεσης:

Εντολή: ./ask2_1a

A CREATED WITH PID = 2709
Parent, PID = 2709: Created child with PID = 2710, waiting for it to terminate...
Parent, PID = 2709: Created child with PID = 2711, waiting for it to terminate...
B CREATED WITH PID = 2710
A: Sleeping...
C CREATED WITH PID = 2711
Parent, PID = 2710: Created child with PID = 2712, waiting for it to terminate...
C: Sleeping...
B: Sleeping...
D CREATED WITH PID = 2712
D: Sleeping...



C: Exiting...
D: Exiting...
My PID = 2709: Child PID = 2711 terminated normally, exit status = 17
My PID = 2710: Child PID = 2712 terminated normally, exit status = 13
B: Exiting...
My PID = 2709: Child PID = 2710 terminated normally, exit status = 19
A: Exiting...
My PID = 2708: Child PID = 2709 terminated normally, exit status = 16

Άσκηση 1.2 – Δημιουργία αυθαίρετου δέντρου εργασιών

Κώδικας:

```
#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)
{
	printf("PROCESS %s CREATED WITH PID = %ld\n",root->name,(long)getpid());
	change_pname(root->name); //Ονομασία κόμβου
	int i;
	for (i=0; i < root->nr_children; i++){ //Επανέλαβε για όλα τα παιδιά του κόμβου
		pid_t child = fork();// Δημιουργία i παιδιού
		if (child < 0) {
			perror("main: fork");
			exit(1);
		}
		if (child == 0) {//Αναδρομή για κάθε παιδί του κόμβου
			fork_procs(root->children+i);
			exit(1);
		}
		printf("Parent, PID = %ld: Created child with PID = %ld, waiting for it to
terminate...\n",(long)getpid(), (long)child);
	}
	sleep(SLEEP_PROC_SEC);
	wait_for_dead_children(root->nr_children);//Αναμονή για τα τον τερματισμό όλων των
παιδιών του κόμβου

	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.
 */

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

        if (root==NULL)
                exit(1);

        pid_t pid = fork();       //Δημιουργία της ρίζας του δέντρου
        if (pid < 0) {
                perror("main: fork");
                exit(1);
        }
        if (pid == 0) {
                fork_procs(root);
                exit(1);
        }

        /* 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;
}
```

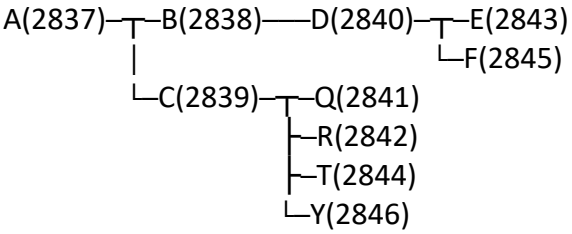
Ερωτήσεις:

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

   Με κάθε δημιουργία διεργασίας-παιδιού, η διεργασία-παιδί μπαίνει στην ουρά έτοιμων διεργασιών και ανταγωνίζεται τον πατέρα για την προτεραιότητα στη CPU. Το λειτουργικό σύστημα κατανέμει τον υπολογιστικό χρόνο ανάμεσα στις διεργασίες ανάλογα με την προτεραιότητα τους και έτσι δεν είμαστε σίγουροι για τη σειρά εμφάνισης των μηνυμάτων έναρξης και τερματισμού των διεργασιών.

Παράδειγμα εκτέλεσης:
Εντολή: ./ask2_1b dentraki

```
A
        B
                D
                        E
                        F
        C
                Q
                R
                T
                Y
PROCESS A CREATED WITH PID = 2837
Parent, PID = 2837: Created child with PID = 2838, waiting for it to terminate...
PROCESS B CREATED WITH PID = 2838
Parent, PID = 2837: Created child with PID = 2839, waiting for it to terminate...
PROCESS C CREATED WITH PID = 2839
Parent, PID = 2838: Created child with PID = 2840, waiting for it to terminate...
PROCESS D CREATED WITH PID = 2840
Parent, PID = 2839: Created child with PID = 2841, waiting for it to terminate...
PROCESS Q CREATED WITH PID = 2841
Parent, PID = 2839: Created child with PID = 2842, waiting for it to terminate...
Parent, PID = 2840: Created child with PID = 2843, waiting for it to terminate...
PROCESS E CREATED WITH PID = 2843
Parent, PID = 2839: Created child with PID = 2844, waiting for it to terminate...
PROCESS R CREATED WITH PID = 2842
Parent, PID = 2840: Created child with PID = 2845, waiting for it to terminate...
PROCESS T CREATED WITH PID = 2844
PROCESS F CREATED WITH PID = 2845
Parent, PID = 2839: Created child with PID = 2846, waiting for it to terminate...
PROCESS Y CREATED WITH PID = 2846
```



My PID = 2839: Child PID = 2841 terminated normally, exit status = 0
My PID = 2839: Child PID = 2842 terminated normally, exit status = 0
My PID = 2840: Child PID = 2843 terminated normally, exit status = 0
My PID = 2839: Child PID = 2844 terminated normally, exit status = 0
My PID = 2840: Child PID = 2845 terminated normally, exit status = 0
My PID = 2839: Child PID = 2846 terminated normally, exit status = 0
My PID = 2838: Child PID = 2840 terminated normally, exit status = 0
My PID = 2837: Child PID = 2839 terminated normally, exit status = 0
My PID = 2837: Child PID = 2838 terminated normally, exit status = 0
My PID = 2836: Child PID = 2837 terminated normally, exit status = 0

Άσκηση 1.3– Αποστολή και χειρισμός σημάτων

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

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



void fork_procs(struct tree_node *root)
{

	printf("PROCESS %s CREATED WITH PID = %ld\n",root->name,(long)getpid());
	change_pname(root->name);//Ονομασία διεργασίας
	pid_t* pidOfChilds = malloc(sizeof(pid_t)*(root->nr_children));
	int i;
	for (i=0; i < root->nr_children; i++){
		pid_t child = fork();	//Δημιουργία παιδιού
		if (child < 0) {
			perror("main: fork");
			exit(1);
		}
		if (child == 0) {
			fork_procs(root->children+i); //Αναδρομή για κάθε παιδί
			exit(1);
		}
		printf("Parent, PID = %ld: Created child with PID = %ld, waiting for it to
terminate...\n",(long)getpid(), (long)child);
		pidOfChilds[i]=child;	//Αποθήκευση του PID κάθε παιδιού
	}

	wait_for_ready_children(root->nr_children); //Αναμονή μέχρι όλα τα παιδιά να
αναστείλουν τη λειτουργία τους
	raise(SIGSTOP);		//Αναστολή λειτουργίας
	printf("PID = %ld, name = %s is awake\n",(long)getpid(), root->name);
```

```
	int status;
	pid_t pid;
	for (i=0; i < root->nr_children; i++){
		kill(pidOfChilds[i],SIGCONT);  //Ξύπνημα παιδιού
		pid=wait(&status);		//Αναμονή μέχρι να τερματιστεί το παιδί
		explain_wait_status(pid, status);
		}

	exit(0);
}

/*
 * In ask2-signals:
 *     use wait_for_ready_children() to wait until
 *     the first process raises SIGSTOP.
 */

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

	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]);
	print_tree(root);

	if (root==NULL)
		exit(1);

	pid_t pid = fork();	//Δημιουργία διεργασίας που είναι η κορυφή του δέντρου
	if (pid < 0) {
		perror("main: fork");
		exit(1);
	}
	if (pid == 0) {
		printf("ROOT_PROCESS CREATED WITH PID = %ld\n",(long)getpid());
		fork_procs(root);
		exit(1);
	}
```

```
        //Αναμονή μέχρι να αναστείλει τη λειτουργία η διεργασία
        wait_for_ready_children(1);


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

        //Ξύπνημα της διεργασίας
        kill(pid, SIGCONT);

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

        return 0;
}
```

Ερωτήσεις:

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

   Με τη χρήση σημάτων επιτυγχάνεται ο πλήρης έλεγχος μιας διεργασίας όπως ο καθορισμός της χρονική στιγμής που θα ξεκινήσει μια διεργασία, πότε θα αναστείλει τη λειτουργία της, πότε θα ενεργοποιηθεί ξανά, πότε θα τερματιστεί. Επομένως, με αυτό τον τρόπο μπορούμε να αποφύγουμε την τυχαία σειρά εκτέλεσης των διεργασιών και να ελέγξουμε πότε και με ποια σειρά θα εκτελεστούν συγκεκριμένα τμήματα των διεργασιών.

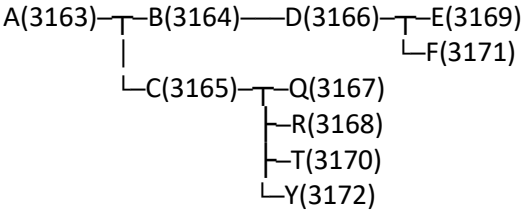
2. ***Ποιος ο ρόλος της wait_for_ready_children(); Τι εξασφαλίζει η χρήση της και τι πρόβλημα θα δημιουργούσε η παράλειψή της;***

   Η διεργασία wait_for_ready_children() χρησιμοποιείται για να βεβαιωθεί μία διεργασία πριν αναστείλει τη λειτουργία της ότι όλα τα παιδιά της έχουν αναστείλει τη δική τους λειτουργία. Με την παράλειψη της wait_for_ready_children() θα μπορούσε μία διεργασία να προσπαθήσει να ενεργοποιήσει παιδί της το οποίο ακόμη δεν έχει αναστείλει τη λειτουργία του αφού δεν υπάρχει η επιβεβαίωση ότι το παιδί ανέστειλε τη λειτουργία του πριν από τη γονική διεργασία.

Παράδειγμα εκτέλεσης:

Εντολή: ./ask2-signals dentraki

```
A
        B
                D
                        E
                        F
        C
                Q
                R
                T
                Y
ROOT_PROCESS CREATED WITH PID = 3163
PROCESS A CREATED WITH PID = 3163
Parent, PID = 3163: Created child with PID = 3164, waiting for it to terminate...
Parent, PID = 3163: Created child with PID = 3165, waiting for it to terminate...
PROCESS B CREATED WITH PID = 3164
PROCESS C CREATED WITH PID = 3165
Parent, PID = 3164: Created child with PID = 3166, waiting for it to terminate...
Parent, PID = 3165: Created child with PID = 3167, waiting for it to terminate...
PROCESS D CREATED WITH PID = 3166
PROCESS Q CREATED WITH PID = 3167
Parent, PID = 3165: Created child with PID = 3168, waiting for it to terminate...
Parent, PID = 3166: Created child with PID = 3169, waiting for it to terminate...
Parent, PID = 3165: Created child with PID = 3170, waiting for it to terminate...
PROCESS R CREATED WITH PID = 3168
PROCESS E CREATED WITH PID = 3169
Parent, PID = 3166: Created child with PID = 3171, waiting for it to terminate...
PROCESS T CREATED WITH PID = 3170
Parent, PID = 3165: Created child with PID = 3172, waiting for it to terminate...
My PID = 3166: Child PID = 3169 has been stopped by a signal, signo = 19
My PID = 3165: Child PID = 3167 has been stopped by a signal, signo = 19
My PID = 3165: Child PID = 3168 has been stopped by a signal, signo = 19
PROCESS Y CREATED WITH PID = 3172
PROCESS F CREATED WITH PID = 3171
My PID = 3165: Child PID = 3170 has been stopped by a signal, signo = 19
My PID = 3165: Child PID = 3172 has been stopped by a signal, signo = 19
My PID = 3166: Child PID = 3171 has been stopped by a signal, signo = 19
My PID = 3163: Child PID = 3165 has been stopped by a signal, signo = 19
My PID = 3164: Child PID = 3166 has been stopped by a signal, signo = 19
My PID = 3163: Child PID = 3164 has been stopped by a signal, signo = 19
My PID = 3162: Child PID = 3163 has been stopped by a signal, signo = 19
```




```
PID = 3163, name = A is awake
PID = 3164, name = B is awake
PID = 3166, name = D is awake
PID = 3169, name = E is awake
My PID = 3166: Child PID = 3169 terminated normally, exit status = 0
PID = 3171, name = F is awake
My PID = 3166: Child PID = 3171 terminated normally, exit status = 0
My PID = 3164: Child PID = 3166 terminated normally, exit status = 0
My PID = 3163: Child PID = 3164 terminated normally, exit status = 0
PID = 3165, name = C is awake
PID = 3167, name = Q is awake
My PID = 3165: Child PID = 3167 terminated normally, exit status = 0
PID = 3168, name = R is awake
My PID = 3165: Child PID = 3168 terminated normally, exit status = 0
PID = 3170, name = T is awake
My PID = 3165: Child PID = 3170 terminated normally, exit status = 0
PID = 3172, name = Y is awake
My PID = 3165: Child PID = 3172 terminated normally, exit status = 0
My PID = 3163: Child PID = 3165 terminated normally, exit status = 0
My PID = 3162: Child PID = 3163 terminated normally, exit status = 0
```

Άσκηση 1.4– Αποστολή και χειρισμός σημάτων

```
#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* pfd)
{
        printf("PROCESS %s CREATED WITH PID = %ld\n",root->name,(long)getpid());
        change_pname(root->name); //Αλλαγή ονόματος διεργασίας
        int i;
        int pfd_child[2];
        int num1,num2,answer;

        if ((root->nr_children)==0){//Εάν η διεργασία είναι φύλλο, δηλαδή είναι αριθμός
                        int num = atoi (root->name);  //Μετατροπή του string σε integer
                        if ((num==0)&& strcmp(root->name,"0")!=0){          //Εαν υπάρχει
σφάλμα στο αρχείο
                                exit(69);
                        }
                        //Εγγραφή της τιμής της διεργασίας στο pipe που δημιούργησε ο
πατέρας
                        if ( write(pfd[1],&num, sizeof(int)) != sizeof(int) ) {
                                perror("error: write to pipe");
                                exit(1);
                        }
                        sleep(SLEEP_PROC_SEC);
        }
        else{
                printf("PID = %ld: Creating pipe...\n",(long)getpid());
                //Δημιουργία pipe για επικοινωνία του πατέρα με τα παιδιά
                if (pipe(pfd_child) < 0) {
                        perror("error:creating pipe");
                        exit(1);
                }
```

```
for (i=0; i < 2; i++){
                        pid_t child = fork();     //Δημιουργία παιδιού
                        if (child < 0) {
                                perror("main: fork");
                                exit(1);
                        }
                        if (child == 0) {
                                fork_procs(root->children+i,pfd_child); //Αναδρομή για κάθε
παιδί. Ως δεύτερη παράμετρος δίνεται το pipe του πατέρα στο οποίο θα γράψει το παιδί
                                exit(1);
                        }
                        printf("Parent, PID = %ld: Created child with PID = %ld, waiting for it to
terminate...\n",(long)getpid(), (long)child);
                }


        wait_for_dead_children(2);    //Αναμονή για τερματισμό των παιδιών

        //Διάβασμα από το pipe των τιμών που έγραψαν τα παιδιά
        if (  read(pfd_child[0], &num1, sizeof(int) ) != sizeof(int) ) {
                perror("child: read from pipe");
                exit(1);
                }

        if ( read(pfd_child[0], &num2, sizeof(int) ) != sizeof(int) ) {
                perror("child: read from pipe");
                exit(1);
                }

        if (strcmp(root->name, "*")==0){//Εκτέλεση πολλαπλασιασμού
                answer=num1*num2;
                        printf("%d %d %d\n\n",answer,num1,num2);
        }
        else
        if (strcmp(root->name, "+")==0)
                answer=num1+num2;//Εκτέλεση πρόσθεσης
        else{
                exit(69);
        }
```

```
//Εγγραφή του αποτελέσματος στο pipe που δημιούργησε ο πατέρας της διεργασίας
	if (write(pfd[1],&(answer), sizeof(int)) != sizeof(int)) {
			perror("error: write to pipe");
			exit(1);
			}
	}

	exit(0);
}
int main(int argc, char *argv[])
{

	struct tree_node *root;
	int pfd[2];
	int status;
	int finalAnswer;

	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 (root==NULL)
		exit(1);
	//Δημιουργία pipe για επικοινωνία της main με την κορυφή του δέντρου
	printf("Parent: Creating pipe...\n");
	if (pipe(pfd) < 0) {
		perror("pipe");
		exit(1);
	}
		printf("MAIN PROCESS: Creating child...\n");
	pid_t pid = fork();//Δημιουργία της κορυφής του δέντρου
	if (pid < 0) {
		perror("main: fork");
		exit(1);
	}
	if (pid == 0) {
		printf("ROOT_PROCESS CREATED WITH PID = %ld\n",(long)getpid());
		fork_procs(root,pfd);
		exit(1);
	}
```

```
        show_pstree(pid);

        wait(&status); //Αναμονή για τερματισμό της διεργασίας
        explain_wait_status(pid, status);

        //Διάβασμα της τελική απάντησης από την κορυφή του δέντρου
        if (read(pfd[0], &finalAnswer, sizeof(finalAnswer)) != sizeof(finalAnswer)) {
                perror("child: read from pipe");
                exit(1);
                }
        printf("The final answer is: %d\n\n",finalAnswer);

        return 0;
}
```

Ερωτήσεις:

1. ***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;***

   Κάθε διεργασία που αντιστοιχεί σε αριθμό δεν δημιουργεί σωλήνωση αλλά γράφει μόνο στη σωλήνωση που δημιούργησε ο πατέρας, ενώ κάθε διεργασία που αντιστοιχεί σε αριθμητικό τελεστή δημιουργεί μία καινούρια σωλήνωση για να πάρει τα αποτελέσματα από τα παιδιά της. Επομένως, κάθε αριθμητικός τελεστής διαβάζει τα δύο αποτελέσματα των παιδιών του από τη σωλήνωση που δημιούργησε και γράφει το δικό του αποτέλεσμα στη σωλήνωση του πατέρα του.

2. ***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;***

   Το δέντρο διεργασιών δίνει το πλεονέκτημα αξιοποίησης των πολλών επεξεργαστών αφού οι διεργασίες μπορούν να εκτελούνται παράλληλα (το δέντρο μοιράζει την εργασία σε πολλές διεργασίες οι οποίες κατανέμονται στους πολλούς επεξεργαστές με αποτέλεσμα την μείωση στη συσσώρευση στην ουρά αναμονής) και να υπολογίζονται ταυτόχρονα τα αποτελέσματα διαφορετικών αριθμητικών τελεστών, εξοικονομώντας έτσι χρόνο σε σχέση με την αποτίμηση μιας μόνο διεργασίας στην οποία θα γίνουν όλα σειριακά.

Παράδειγμα εκτέλεσης:

```
Εντολή: ./ask2_4 equation
*
        +
                3
                8
        5
Parent: Creating pipe...
MAIN PROCESS: Creating child...
ROOT_PROCESS CREATED WITH PID = 3261
PROCESS * CREATED WITH PID = 3261
PID = 3261: Creating pipe...
Parent, PID = 3261: Created child with PID = 3263, waiting for it to terminate...
Parent, PID = 3261: Created child with PID = 3264, waiting for it to terminate...
PROCESS + CREATED WITH PID = 3263
PID = 3263: Creating pipe...
PROCESS 5 CREATED WITH PID = 3264
Parent, PID = 3263: Created child with PID = 3265, waiting for it to terminate...
PROCESS 3 CREATED WITH PID = 3265
Parent, PID = 3263: Created child with PID = 3266, waiting for it to terminate...
PROCESS 8 CREATED WITH PID = 3266


*(3261)─┬─+(3263)─┬─3(3265)
        │         └─8(3266)
        └─5(3264)


My PID = 3261: Child PID = 3264 terminated normally, exit status = 0
My PID = 3263: Child PID = 3265 terminated normally, exit status = 0
My PID = 3263: Child PID = 3266 terminated normally, exit status = 0
My PID = 3261: Child PID = 3263 terminated normally, exit status = 0
55 5 11

My PID = 3260: Child PID = 3261 terminated normally, exit status = 0
The final answer is: 55
```

MakeFile:

```
all: ask2_4 ask2-signals ask2_1b ask2_1a

ask2_4: ask2_4.o proc-common.o tree.o
	gcc ask2_4.o proc-common.o tree.o -o ask2_4

ask2-signals: ask2-signals.o proc-common.o tree.o
	gcc ask2-signals.o proc-common.o tree.o -o ask2-signals

ask2_4.o: ask2_4.c
	gcc -Wall -c ask2_4.c

ask2-signals.o: ask2-signals.c
	gcc -Wall -c ask2-signals.c

ask2_1b: ask2_1b.o proc-common.o tree.o
	gcc ask2_1b.o proc-common.o tree.o -o ask2_1b

ask2_1b.o: ask2_1b.c
	gcc -Wall -c ask2_1b.c

ask2_1a: ask2_1a.o proc-common.o tree.o
	gcc ask2_1a.o proc-common.o tree.o -o ask2_1a

ask2_1a.o: ask2_1a.c
	gcc -Wall -c ask2_1a.c

proc-common.o:proc-common.c
	gcc -Wall -c proc-common.c

tree.o: tree.c
	gcc -Wall -c tree.c
```

Στο αρχείο proc-common.c συμπεριλάβαμε επίσης τη συνάρτηση :

```
wait_for_ready_children(int cnt)
{
	int i, status;
	pid_t p;
		for (i = 0; i < cnt; i++) {/* Wait for any child*/
		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);}
		}
}
```

Επίσης, τροποποιήσαμε τη συνάρτηση explain_wait_status έτσι ώστε όταν μία διεργασία τερματίσει με 69 ή 1 να εμφανίζεται μήνυμα ότι έχει δοθεί λανθασμένο αρχείο εισόδου ή ότι έχει γίνει κάποιο system error αντίστοιχα και στη συνέχεια να τερματίζονται διαδοχικά όλες οι γονικές διεργασίες.

```
void explain_wait_status(pid_t pid, int status)
{
	if (WEXITSTATUS(status)==69){ // Αν η διεργασία τερμάτισε με 69 τότε έχει δοθεί
//λανθασμένο input
		fprintf(stderr, "My PID = %ld: Child PID = %ld terminated unexpected because
wrong input, exit status = %d\n",
			(long)getpid(), (long)pid, WEXITSTATUS(status));
		exit(69); //Διαδοχικός τερματισμός όλων των διεργασιών
	}
	else if (WEXITSTATUS(status)==1){ // Αν η διεργασία τερμάτισε με 1 τότε έχει
// γίνει κάποιο system error

		fprintf(stderr, "My PID = %ld: Child PID = %ld terminated unexpected because
system error, exit status = %d\n",
			(long)getpid(), (long)pid, WEXITSTATUS(status));
		exit(1); //Διαδοχικός τερματισμός όλων των διεργασιών
	}
	else if (WIFEXITED(status))
		fprintf(stderr, "My PID = %ld: Child PID = %ld terminated normally, exit status =
%d\n",
			(long)getpid(), (long)pid, WEXITSTATUS(status));
	else if (WIFSIGNALED(status))
		fprintf(stderr, "My PID = %ld: Child PID = %ld was terminated by a signal, signo =
%d\n",
			(long)getpid(), (long)pid, WTERMSIG(status));
	else if (WIFSTOPPED(status))
		fprintf(stderr, "My PID = %ld: Child PID = %ld has been stopped by a signal, signo
= %d\n",
			(long)getpid(), (long)pid, WSTOPSIG(status));
	else {
		fprintf(stderr, "%s: Internal error: Unhandled case, PID = %ld, status = %d\n",
			__func__, (long)pid, status);
		exit(1);
	}
	fflush(stderr);
}
```