ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

http://www.cslab.ece.ntua.gr

Πέτρου Νικόλας	oslabb13	03117714
Πεγειώτη Νάταλυ		03117707

Λειτουργικά Συστήματα

6ο εξάμηνο, Ακαδημαϊκή περίοδος 2019-2020

Άσκηση 2
Διαχείριση Διεργασιών και Διαδιεργασιακή Επικοινωνία

1 Ασκήσεις

1.1 Δημιουργία δεδομένου δέντρου διεργασιών

Ερωτήσεις

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

Το πρόγραμμα θα τερματίσει όταν ο πατέρας της Α θα πάρει το status του τερματισμού της.
Τα παιδιά της Α θα γίνουν zombies, αφού ο πατέρας τους έχει πεθάνει, θα τα υιοθετήσει η init και θα συνεχίσουν την λειτουργία τους μέχρι να πεθάνουν.
Αυτό δε σημαίνει όμως ότι το πρόγραμμα θα δουλέψει σωστά.

```
oslabb13@os-node1:~/lab2$ ./ask2-fork
A: PID = 29640: Creating child C...
C: PID = 29641:
A: Waiting for child C to terminate...
C: Sleeping...
A: PID = 29640: Creating child B...
B: PID = 29642:
A: Waiting for child B to terminate...
B: PID = 29642: Creating child D...
D: PID = 29643:
B: Waiting for child D to terminate...
D: Sleeping...


A(29640)─┬─B(29642)──D(29643)
         └─C(29641)


My PID = 29639: Child PID = 29640 was terminated by a signal, signo = 9
```

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 εντολή show_pstree() παρουσιάζει το δέντρο διεργασιών ξεκινώντας από την ρίζα που του δίνεται σαν όρισμα.
Στην περίπτωση που δίνεται σαν όρισμα το getpid() αντί του pid, δίνεται σαν ρίζα ο πατέρας του A, οπότε στο δέντρο φαίνονται και τα υπόλοιπα παιδιά της κύριας διεργασίας η οποία δημιουργήθηκε με την εκτέλεση του προγράμματος.
Εκτός από αυτήν φαίνεται το παιδί της, η sh η οποία δημιουργείται λόγω του τρόπου υλοποίησης της δοσμένης συνάρτησης “show_pstree”, η οποία χρησιμοποιεί το shell για να καλέσει την pstree, που επίσης δημιουργείται σαν παιδί της sh.

show_pstree(pid) :

```
oslabb13@os-node1:~/lab2$ ./ask2-fork
A: PID = 12494: Creating child C...
C: PID = 12495:
A: Waiting for child C to terminate...
C: Sleeping...
A: PID = 12494: Creating child B...
B: PID = 12496:
A: Waiting for child B to terminate...
B: PID = 12496: Creating child D...
D: PID = 12497:
B: Waiting for child D to terminate...
D: Sleeping...


A(12494)─┬─B(12496)──D(12497)
         └─C(12495)


C: Exiting...
My PID = 12494: Child PID = 12495 terminated normally, exit status = 17
D: Exiting...
My PID = 12496: Child PID = 12497 terminated normally, exit status = 13
B: Exiting...
My PID = 12494: Child PID = 12496 terminated normally, exit status = 19
A: Exiting...
My PID = 12493: Child PID = 12494 terminated normally, exit status = 16
```

show_pstree(getpid()):

```
oslabb13@os-node1:~/lab2$ ./ask2-fork
A: PID = 12465: Creating child C...
C: PID = 12466:
A: Waiting for child C to terminate...
C: Sleeping...
A: PID = 12465: Creating child B...
B: PID = 12467:
A: Waiting for child B to terminate...
B: PID = 12467: Creating child D...
D: PID = 12468:
B: Waiting for child D to terminate...
D: Sleeping...


ask2-fork(12464)─┬─A(12465)─┬─B(12467)──D(12468)
                 │          └─C(12466)
                 └─sh(12469)──pstree(12470)


C: Exiting...
My PID = 12465: Child PID = 12466 terminated normally, exit status = 17
D: Exiting...
My PID = 12467: Child PID = 12468 terminated normally, exit status = 13
B: Exiting...
My PID = 12465: Child PID = 12467 terminated normally, exit status = 19
A: Exiting...
My PID = 12464: Child PID = 12465 terminated normally, exit status = 16
```

3. Σε υπολογιστικά συστήματα πολλαπλών χρηστών, πολλές φορές ο διαχειριστής θέτει όρια στον αριθμό των διεργασιών που μπορεί να δημιουργήσει ένας χρήστης. Γιατί;
Τα όρια αυτά τίθενται έτσι ώστε να μην υπάρχει η δυνατότητα για κανένα χρήστη να κατέχει όλους τους πόρους του συστήματος, είτε επιτηδευμένα είτε από κάποιο σφάλμα.

1.2 Δημιουργία αυθαίρετου δέντρου διεργασιών

Ερωτήσεις

1. Με ποια σειρά εμφανίζονται τα μηνύματα έναρξης και τερματισμού των διεργασιών; γιατί;
Τα μηνύματα έναρξης και τερματισμου δεν εμφανίζονται πάντα με την ίδια σειρά καθώς εξαρτώνται από τη σειρά που θα τρέξουν οι διεργασίες.
Η σειρά εκτέλεσης των διεργασιών εξαρτάται από τον Χρονοδρομολογητή, ο οποίος επιλέγει με δικά του κριτήρια.

Η έξοδος του προγράμματος για το αρχείο tree1.2 που δημιουργήσαμε σαν test είναι η ακόλουθη:

```
oslabb13@os-node2:~/lab2$ ./prog1.2 tree1.2
A
        B
                C
                D
        E
                F
        G
Parent, PID = 3163: Creating child...
I am A, with PID = 3164 and Parent with PID = 3163
I am B, with PID = 3165 and Parent with PID = 3164
I am E, with PID = 3166 and Parent with PID = 3164
I am G, with PID = 3167 and Parent with PID = 3164
I am C, with PID = 3168 and Parent with PID = 3165
I am F, with PID = 3169 and Parent with PID = 3166
I am D, with PID = 3170 and Parent with PID = 3165


A(3164)─┬─B(3165)─┬─C(3168)
        │         └─D(3170)
        ├─E(3166)───F(3169)
        └─G(3167)


My PID = 3164: Child PID = 3167 terminated normally, exit status = 3
My PID = 3166: Child PID = 3169 terminated normally, exit status = 3
My PID = 3165: Child PID = 3168 terminated normally, exit status = 3
My PID = 3165: Child PID = 3170 terminated normally, exit status = 3
My PID = 3164: Child PID = 3166 terminated normally, exit status = 2
My PID = 3164: Child PID = 3165 terminated normally, exit status = 2
My PID = 3163: Child PID = 3164 terminated normally, exit status = 2
```

1.3 Αποστολή και χειρισμός σημάτων

Ερωτήσεις

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

Με τη συνάρτηση sleep() ουσιαστικά γινόταν προσπάθεια για τον υπολογισμό του χρόνου εκτέλεσης των διεργασιών έτσι ώστε να επιτευχθεί ο συγχρονισμός τους.
Αυτό δεν είναι αποδοτικό για πολλές διεργασίες καθώς δημιουργούνται καθυστερήσεις σε σημεία που δεν πρέπει και επιπλέον οι χρόνοι sleep() που δουλεύουν για ένα μέγεθος δέντρου είναι πολύ πιθανό να μην δουλεύουν για ένα μεγαλύτερο δέντρο, με αποτέλεσμα να μην υπάρχει συγχρονισμός.
Με τη χρήση σημάτων υπάρχει καλύτερος έλεγχος, λόγω της απευθείας επικοινωνίας μεταξύ των διεργασιών οι οποίες μπορούν να στείλουν σήματα η μία στην άλλη ανάλογα με το σημείο εκτέλεσης στο οποίο βρίσκονται.

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

Η συνάρτηση wait_for_ready_children() καλείται από τον πατέρα για να σιγουρευτεί ότι όλα τα παιδιά του έχουν αναστείλει την λειτουργία τους. Η συνάρτηση εξασφαλίζει ότι ο πατέρας θα περιμένει όλα του τα παιδιά να ολοκληρώσουν την εκτέλεση τους μέχρι το σημείο όπου θα αναστείλουν τη λειτουργία τους, για να αναστείλει τότε και αυτός την δικιά του, δηλαδή χρησιμοποιείται για τον συγχρονισμό του πατέρα με τα παιδιά.
Αν αυτή παραλειφθεί θα υπάρξουν προβλήματα καθώς ο πατέρας θα αναστείλει την λειτουργία του πριν να είναι σίγουρος ότι τα παιδιά έχουν αναστείλει την δικιά τους. Έτσι δε θα μπορούμε να γνωρίζουμε αν το δέντρο διεργασιών έχει δημιουργηθεί σωστά, τουλάχιστον μέχρι να το τυπώσουμε.
Επιπλέον μετά εκτύπωση του δέντρου, στην οποία θα αναγνωρίσουμε ότι η κατασκευή του είναι λανθασμένη, οι γονείς θα αρχίσουν να στέλνουν σήματα SIGCONT στα παιδιά, αλλά σε περίπτωση που κάποιο παιδί δε θα έχει φτάσει ακόμη στο σημείο αναστολής της λειτουργίας του, δε θα λάβει ποτέ το σήμα από τον πατέρα του, ακόμα και όταν το ίδιο στείλει στον πατέρα του SIGSTOP.

Η έξοδος του προγράμματος για το αρχείο tree1.2 είναι η ακόλουθη:

```
oslabb13@os-node2:~/lab2$ ./prog1.3 tree1.2
I am A, with PID = 3377 and Parent with PID = 3376
I am B, with PID = 3378 and Parent with PID = 3377
I am E, with PID = 3379 and Parent with PID = 3377
A is waiting for his children to sleep
I am G, with PID = 3380 and Parent with PID = 3377
G: Sleeping...
I am C, with PID = 3381 and Parent with PID = 3378
C: Sleeping...
My PID = 3377: Child PID = 3380 has been stopped by a signal, signo = 19
B is waiting for his children to sleep
E is waiting for his children to sleep
I am F, with PID = 3382 and Parent with PID = 3379
F: Sleeping...
My PID = 3378: Child PID = 3381 has been stopped by a signal, signo = 19
I am D, with PID = 3383 and Parent with PID = 3378
D: Sleeping...
My PID = 3379: Child PID = 3382 has been stopped by a signal, signo = 19
My PID = 3378: Child PID = 3383 has been stopped by a signal, signo = 19
My PID = 3377: Child PID = 3378 has been stopped by a signal, signo = 19
My PID = 3377: Child PID = 3379 has been stopped by a signal, signo = 19
My PID = 3376: Child PID = 3377 has been stopped by a signal, signo = 19


A(3377)─┬─B(3378)─┬─C(3381)
        │         └─D(3383)
        ├─E(3379)──F(3382)
        └─G(3380)


PID = 3377, name = A is awake
PID = 3378, name = B is awake
PID = 3381, name = C is awake
My PID = 3378: Child PID = 3381 terminated normally, exit status = 3
PID = 3383, name = D is awake
My PID = 3378: Child PID = 3383 terminated normally, exit status = 3
My PID = 3377: Child PID = 3378 terminated normally, exit status = 2
PID = 3379, name = E is awake
PID = 3382, name = F is awake
My PID = 3379: Child PID = 3382 terminated normally, exit status = 3
My PID = 3377: Child PID = 3379 terminated normally, exit status = 2
PID = 3380, name = G is awake
My PID = 3377: Child PID = 3380 terminated normally, exit status = 3
My PID = 3376: Child PID = 3377 terminated normally, exit status = 2
```

1.4 Παράλληλος υπολογισμός αριθμητικής έκφρασης

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;

Αν η διεργασία είναι φύλλο τότε δεν χρειάζεται κάποια άλλη επιπλέον σωλήνωση εκτός αυτής που θα έχει ήδη δημιουργήσει ο πατέρας της. Άρα ουσιαστικά χρειάζεται μία σωλήνωση ανά ζεύγος πατέρα-παιδιού.

Γενικά δεν είναι καλή πρακτική να προσπαθήσουμε να χρησιμοποιήσουμε μία σωλήνωση από μια γονική διεργασία προς όλα της τα παιδιά καθώς δεν υπάρχει η βεβαιότητα ότι σε κάθε write ή read θα γράψουμε ή θα διαβάσουμε όλα όσα πρέπει. Αν υπάρχουν για παράδειγμα 3 διεργασίες παιδιά που θέλουν να γράψουν στο σωλήνα, δεν είναι σίγουρο ότι θα γράψουν το ένα μετά το άλλο όλα όσα θέλουν, με αποτέλεσμα αυτά που θα διαβάσει ο πατέρας να μην έχουν νόημα. Επιπλέον, ακόμα και αν εξασφαλίσουμε ότι τα παιδιά γράφουν σωστά στη σωλήνωση (όπως κάνουμε στην άσκηση) δεν μπορούμε να εξασφαλίσουμε την σειρά με την οποία θα γράψουν, οπότε σε αριθμητικούς τελεστές όπου η σειρά παίζει ρόλο (π.χ. αφαίρεση), δεν εξασφαλίζεται το σωστό αποτέλεσμα.

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;

Αφού οι διεργασίες μπορούν να εκτελούνται παράλληλα, πράξεις που χρειάζονται αρκετούς υπολογισμούς μπορούν να εκτελεστούν πολύ πιο γρήγορα από ένα δέντρο διεργασιών, παρά από μία μόνο διεργασία, δηλαδή σειριακή εκτέλεση.

Η έξοδος του προγράμματος για το αρχείο expr.tree είναι η ακόλουθη:

```
oslabb13@os-node2:~/lab2$ ./prog1.4 expr.tree
Parent: Creating pipe...
+
        10
        *
                +
                        5
                        7
                4
Parent, PID = 3677: Creating child...
Parent: My PID is 3677, Receiving a value from the child.
I am +, with PID = 3678 and Parent with PID = 3677
+: Creating pipe...
+: Creating pipe...
I am 10, with PID = 3679 and Parent with PID = 3678
+: My PID is 3678. Receiving int values from the children.
+: received value 10 from the pipe.
I am *, with PID = 3680 and Parent with PID = 3678
*: Creating pipe...
*: Creating pipe...
I am +, with PID = 3681 and Parent with PID = 3680
+: Creating pipe...
*: My PID is 3680. Receiving int values from the children.
I am 4, with PID = 3682 and Parent with PID = 3680
+: Creating pipe...
I am 5, with PID = 3683 and Parent with PID = 3681
+: My PID is 3681. Receiving int values from the children.
+: received value 5 from the pipe.
I am 7, with PID = 3684 and Parent with PID = 3681
+: received value 7 from the pipe. Will now compute!
+: Writing to pipe
*: received value 12 from the pipe.
*: received value 4 from the pipe. Will now compute!
*: Writing to pipe
+: received value 48 from the pipe. Will now compute!
+: Writing to pipe
My PID = 3680: Child PID = 3682 terminated normally, exit status = 3
Parent: received value 58 from the pipe. Will now compute.
The final result is:58
My PID = 3678: Child PID = 3679 terminated normally, exit status = 3
My PID = 3681: Child PID = 3683 terminated normally, exit status = 3
My PID = 3681: Child PID = 3684 terminated normally, exit status = 3
My PID = 3680: Child PID = 3681 terminated normally, exit status = 2
My PID = 3678: Child PID = 3680 terminated normally, exit status = 2
My PID = 3677: Child PID = 3678 terminated normally, exit status = 2
```

Παράρτημα Κώδικα

1.1 Δημιουργία δεδομένου δέντρου διεργασιών

```
#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)
{
	int status;
	pid_t pid;
	/*
	 * initial process is A.
	 */

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

	pid = fork();
	if (pid < 0)
	{
		perror("fork");
		exit(1);
	}
	if (pid == 0)
	{
		printf("A: PID = %ld: Creating child C...\n", (long)getppid());
		change_pname("C");
		printf("C: PID = %ld: \n", (long)getpid());
		printf("A: Waiting for child C to terminate...\n");
		printf("C: Sleeping...\n");
		sleep(SLEEP_PROC_SEC);
		printf("C: Exiting...\n");
		exit(17);
	}
```

```
else
{
        pid = fork();
        if (pid < 0)
        {
                perror("fork");
                exit(1);
        }
        if (pid == 0)
        {
                printf("A: PID = %ld: Creating child B...\n", (long)getppid());
                change_pname("B");
                printf("B: PID = %ld: \n", (long)getpid());
                printf("A: Waiting for child B to terminate...\n");
                pid = fork();

                if (pid < 0)
                {
                        perror("fork");
                        exit(1);
                }
                if (pid == 0)
                {
                        printf("B: PID = %ld: Creating child D...\n", (long)getppid());
                        change_pname("D");
                        printf("D: PID = %ld: \n", (long)getpid());
                        printf("B: Waiting for child D to terminate...\n");
                        printf("D: Sleeping...\n");
                        sleep(SLEEP_PROC_SEC);
                        printf("D: Exiting...\n");
                        exit(13);
                }
                else
                {
                        pid = wait(&status);
                        explain_wait_status(pid,status);
                        printf("B: Exiting...\n");
                        exit(19);
                }
        }
        else
        {
                sleep(SLEEP_PROC_SEC);
                pid = wait(&status);
                explain_wait_status(pid,status);
                pid = wait(&status);
                explain_wait_status(pid,status);
        }
}
/* ... */
```

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

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

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

	/*
	 * Father
	 */
	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;
}
```

1.2 Δημιουργία αυθαίρετου δέντρου διεργασιών

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.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 create_proc(struct tree_node *root){
	int status;
	int i,j;
	pid_t pid;
	change_pname(root->name);
	printf("I am %s, with PID = %ld and Parent with PID = %ld\n",root->name,(long)getpid(),
(long)getppid());

	for(i=0; i<root->nr_children; i++){
		pid = fork();
		if(pid == 0){
			if((root->children+i)->nr_children==0){
				change_pname((root->children+i)->name);
				printf("I am %s, with PID = %ld and Parent with PID = %ld\n",(root-
>children+i)->name,(long)getpid(),(long)getppid());
				sleep(SLEEP_PROC_SEC);
			exit(3);
			}
			else{
			create_proc(root->children+i);
			}
		}
	}

	for(j=0; j<root->nr_children; j++){
		pid=wait(&status);
		explain_wait_status(pid,status);
	}
	exit(2);
}

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

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

	pid = fork();
	if (pid < 0){
		/*fork failed*/
		perror("main: fork");
		exit(1);
	}
	if (pid == 0){
		/*child*/
		create_proc(root);
		exit(1);
	}

	sleep(SLEEP_TREE_SEC);

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

	/*wait for the root of the process tree to terminate*/
	wait(&status);
	explain_wait_status(pid,status);
	return 0;
}
```

1.3 Αποστολή και χειρισμός σημάτων

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

void create_proc(struct tree_node *root){
	int status;
	int i,j;
	pid_t pid[root->nr_children];
	change_pname(root->name);
	printf("I am %s, with PID = %ld and Parent with PID = %ld\n",root->name,(long)getpid(),
(long)getppid());

	for(i=0; i< root->nr_children; i++){
		pid[i] = fork();
		if(pid[i] == 0){
			pid[i] = getpid();
			if((root->children + i)->nr_children==0){
				change_pname((root->children+i)->name);
				printf("I am %s, with PID = %ld and Parent with PID = %ld\n",(root-
>children+i)->name,(long)getpid(),(long)getppid());
				printf("%s: Sleeping...\n", (root->children+i)->name);
				raise(SIGSTOP);
				printf("PID = %ld, name = %s is awake\n", (long)getpid(), (root-
>children+i)->name);
				exit(3);
			}
			else{
				create_proc(root->children+i);
			}
		}
	}
	printf("%s is waiting for his children to sleep\n", root->name);
	wait_for_ready_children(root->nr_children);
	raise(SIGSTOP);
	printf("PID = %ld, name = %s is awake\n", (long)getpid(), root->name);

	for(j=0; j < root->nr_children; j++){
		kill(pid[j],SIGCONT);
		pid[i]=wait(&status);
		explain_wait_status(pid[i],status);
	}
	exit(2);
}
```

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

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

	/*read tree*/
	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){
		create_proc(root);
		exit(1);
	}

	wait_for_ready_children(1);

	/*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;
}
```

1.4 Παράλληλος υπολογισμός αριθμητικής έκφρασης

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

void create_proc(struct tree_node *root,int fd){
	int status;
	int i,j;
	int val;
	pid_t pid;
	int pfd[2];
	int read_pfd[2];
	int num1,num2,result;

	change_pname(root->name);
	printf("I am %s, with PID = %ld and Parent with PID = %ld\n",root->name,(long)getpid(),
(long)getppid());

	for(i=0; i<root->nr_children; i++){
		printf("%s: Creating pipe...\n",root->name);
		if(pipe(pfd) < 0){
			perror("pipe");
			exit(1);
		}
		read_pfd[i] = pfd[0];
		pid = fork();
		if(pid == 0){
			if((root->children + i)->nr_children == 0){
				change_pname((root->children + i)->name);
				printf("I am %s, with PID = %ld and Parent with PID = %ld\n",(root-
>children + i)->name, (long)getpid(),(long)getppid());
				val = atoi((root->children +i)->name);
				if(write(pfd[1],&val,sizeof(val)) != sizeof(val)){
					perror("Child: write to pipe");
					exit(1);
				}
				exit(3);
			}
			else{
				create_proc(root->children + i,pfd[1]);
			}
		}
	}
	printf("%s: My PID is %ld. Receiving int values from the children.\n",root->name,
(long)getpid());
```

```
	if(read(read_pfd[0], &num1, sizeof(num1)) != sizeof(num1)){
		perror("Parent: read from pipe");
		exit(1);
	}
	printf("%s: received value %d from the pipe.\n",root->name,num1);

	if(read(read_pfd[1], &num2, sizeof(num2)) != sizeof(num2)){
		perror("Parent: read from pipe");
		exit(1);
	}
	printf("%s: received value %d from the pipe. Will now compute!\n",root->name,num2);
	int symbol = *root->name;
	if((symbol) == '+'){
		result = num1 + num2;
	}
	else{
		result = num1 * num2;
	}
	printf("%s: Writing to pipe\n",root->name);
	if(write(fd,&result,sizeof(result)) != sizeof(result)){
		perror("child: write to pipe");
		exit(1);
	}

	for(j=0; j<root->nr_children; j++){
		pid=wait(&status);
		explain_wait_status(pid,status);
	}
	exit(2);
}

int main(int argc,char *argv[])
{
	struct tree_node *root;
	int status;
	int val;
	int pfd[2];
	pid_t pid;

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

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

	pid = fork();
	if (pid < 0){
		/*fork failed*/
		perror("main: fork");
		exit(1);
	}
	if (pid == 0){
		/*child*/
		create_proc(root,pfd[1]);
		exit(1);
	}

	printf("Parent: My PID is %ld, Receiving a value from the child.\n",(long)getpid());
	if(read(pfd[0],&val,sizeof(val)) != sizeof(val)){
		perror("parent: read from pipe");
		exit(1);
	}
	printf("Parent: received value %d from the pipe. Will now compute.\n",val);
	printf("The final result is:%d\n",val);
	/*wait for the root of the process tree to terminate*/
	wait(&status);
	explain_wait_status(pid,status);
	return 0;
}
```