ΑΝΑΦΟΡΑ ΑΣΚΗΣΗΣ 2

Λειτουργικά Συστήματα 6ο εξάμηνο
Ακαδημαϊκή περίοδος 2019-2020
Ομάδα: oslabc18
Φοιτητές:Αθανασίου Ιωάννης Α.Μ.:03117041
Καραβαγγέλης Αθανάσιος Α.Μ.:03117022

ΑΣΚΗΣΗ 2.1

- ***Source code:***

```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <signal.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 B, C, D;

	/* Root A */
	change_pname("A");
	printf("Parent created root (A) with PID = %ld , waiting for it to terminate...\n", (long)getpid());
	// printf("A: Sleeping...\n");
	// sleep(SLEEP_PROC_SEC);
	// kill -KILL <getpid()>
	// fprintf(stderr, "Parent (A), PID = %ld: Trying to creat child (B)...\n", (long)getpid());
	B = fork();
	if (B<0) {
		perror("fork");
		exit(1);
	}
	if (B==0) {
		/* Child B */
		change_pname("B");
		printf("Parent (A), created child (B) with PID = %ld, waiting for it to terminate...\n", (long)B);
		// printf("B: Sleeping...\n");
		// sleep(SLEEP_PROC_SEC);
	  // fprintf(stderr, "Parent (B), PID = %ld: Trying to creat child (D)...\n", (long)getpid());
		D = fork();
		if (D<0) {
		   perror("fork");
		   exit(1);
		}
		if (D==0) {
		   /*Child D*/
		   change_pname("D");
		   printf("Parent (B), created child (D) with PID = %ld, waiting for it to terminate...\n", (long)D);
		   printf("D: Sleeping...\n");
		   sleep(SLEEP_PROC_SEC);
		   printf("D: Exiting...\n");
		   exit(13);
		}
		/* back to root B */
		// printf("B: Sleeping...\n");
		// sleep(SLEEP_PROC_SEC);
```

```
            D = wait(&status);
            explain_wait_status(D, status);
            printf("Parent B: All done, exiting...\n");
            exit(19);
        }
        /* back to root A*/
        // fprintf(stderr, "Parent (A), PID = %ld: Trying to creat child (C)...\n", (long)getpid());
        C = fork();
        if (C<0) {
           perror("fork");
           exit(1);
        }
        if (C==0) {
        /* child C */
        change_pname("C");
        printf("Parent (A), created child (C) with PID = %ld, waiting for it to terminate...\n", (long)C);
        printf("C: Sleeping...\n");
        sleep(SLEEP_PROC_SEC);
        printf("C: Exiting...\n");
        exit(17);
        }
        /* back to root A */
        /* Wait for the root of the process tree to terminate */
        C = wait(&status);
        explain_wait_status(C, status);
        B = wait(&status);
        explain_wait_status(B, status);
        printf("A: Exiting...\n");
        exit(16);
}

int main(void) {
        pid_t A;
        int status;
  //    fprintf(stderr, "Parent, PID = %ld: Trying to creating root (A)...\n", (long)getpid());
        /* Fork root of process tree */
        A = fork();
        if (A < 0) {
            perror("main: fork");
            exit(1);
        }
        if (A == 0) {

            /* trying to create the other children with function fork_procs */
            fork_procs();
            // exit(1);
        }

        /*
         * Father
         */

        sleep(SLEEP_TREE_SEC);

        /* Print the process tree root at pid */
        show_pstree(A);
        /* Wait for the root of the process tree to terminate */

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

        return 0;
}
```

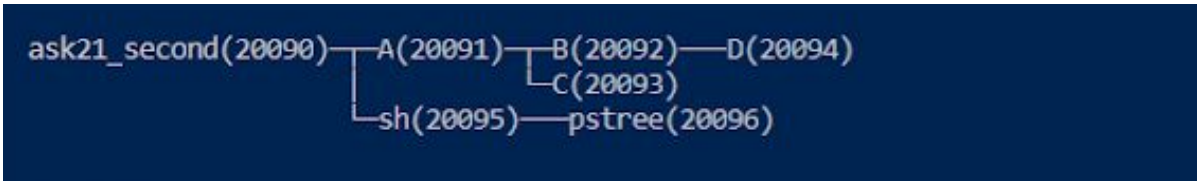
- **Η έξοδος του προγράμματος μας είναι:**

```
Parent created root (A) with PID = 27069 , waiting for it to terminate...
Parent (A), created child (B) with PID = 27070, waiting for it to terminate...
Parent (A), created child (C) with PID = 27071, waiting for it to terminate...
C: Sleeping...
Parent (B), created child (D) with PID = 27072, waiting for it to terminate...
D: Sleeping...


A(27069)─┬─B(27070)──D(27072)
         └─C(27071)


C: Exiting...
D: Exiting...
My PID = 27069: Child PID = 27071 terminated normally, exit status = 17
My PID = 27070: Child PID = 27072 terminated normally, exit status = 13
B: Exiting...
My PID = 27069: Child PID = 27070 terminated normally, exit status = 19
A: Exiting...
My PID = 27068: Child PID = 27069 terminated normally, exit status = 16
```

- **Ερωτήσεις:**

1) Αν τερματίσουμε πρόωρα την διεργασία Α, τότε όλες οι διεργασίες παιδιά της θα αποτελούν πλέον παιδιά της πρώτης διεργασίας Init, η οποία έχει PID=1 και κάνει συνεχώς wait.

2) Παρατηρούμε ότι όταν αλλάζουμε τον κώδικα κατ'αυτόν τον τρόπο αλλάζει το δένδρο διεργασιών που εμφανίζεται. Μέσω της εντολής show_pstree(getpid()), ζητάμε να εκτυπωθεί στην οθόνη το δένδρο διεργασιών με ρίζα την διεργασία-πατέρα της Α και όχι την ίδια την Α. Η νέα ρίζα εμφανίζεται άρα με το όνομα του εκτελέσιμου αρχείου που είναι αυτήν την στιγμή υπό εκτέλεση. Αυτή η διεργασία έχει πλέον ως παιδιά, εκτός από όλες τις άλλες, και τις διεργασίες με ονόματα "sh" και "pstree". Η διεργασία "sh", όπως δίνεται και με την εντολή "man sh", αποτελεί τον διερμηνέα της γραμμής εντολών, ένα πρόγραμμα που "τρέχει" όποτε κάποιος χρήστης συνδέεται στον υπολογιστή και επομένως είναι αναμενόμενη η εμφάνιση του στο δένδρο διεργασιών. Ομοίως βρίσκουμε (man pstree) ότι η εντολή "pstree" έχει ως αποτέλεσμα την εκτύπωση του δένδρου διεργασιών που προσδιορίζεται με βάση περιορισμούς που μπορούμε να εισάγουμε κατά την κλήση της. Είναι αναμενόμενο η διεργασία να είναι ενεργή αφού χρησιμοποιείται από την συνάρτηση "show_pstree" που χρησιμοποιούμε στον κώδικα. Έτσι λοιπόν κάνοντας τη συγκεκριμένη αντικατάσταση θα βλέπαμε στην οθόνη μας την εξής έξοδο:



3) Σε συστήματα πολλών χρηστών, παρομοίως με το δένδρο διεργασιών του παραδείγματος μας, κάθε διεργασία θα περιμένει να ολοκληρωθούν όλες οι διεργασίες

παιδιά της προκειμένου να συνεχίσει τη λειτουργία της. Θεωρούμε ότι οι διεργασίες κάθε χρήστη ανήκουν σε ένα διαφορετικό υπόδενδρο του δένδρου διεργασιών. Τότε, οι διεργασίες που βρίσκονται σε υψηλότερο επίπεδο από τη ρίζα αυτού του δένδρου θα πρέπει να περιμένουν την ολοκλήρωση όλων των διεργασιών του δένδρου για να συνεχιστούν. Έτσι, η ανεξέλεγκτη αύξηση του αριθμού των διεργασιών που εκτελεί ένας χρήστης (που ισοδυναμεί με την αύξηση του μεγέθους του υποδένδρου εργασιών του) θα μπορούσε να προκαλέσει σημαντικές καθυστερήσεις στην λειτουργία των κύριων δραστηριοτήτων του συστήματος και να καθυστερήσει ενδεχομένως και τους υπόλοιπους χρήστες.

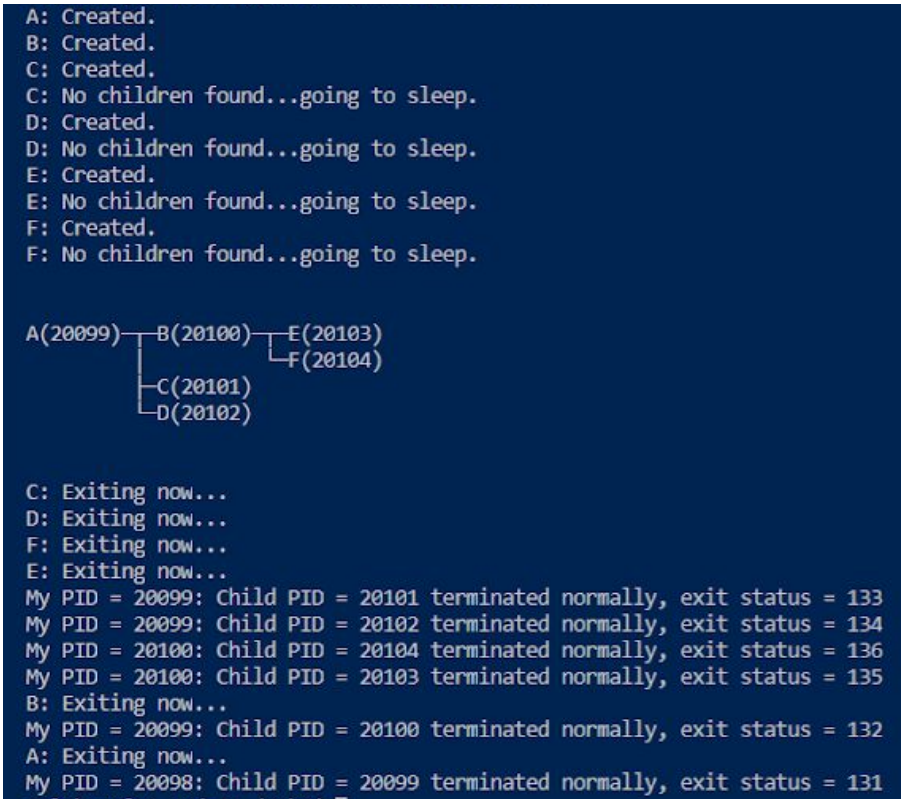
ΑΣΚΗΣΗ 2.2

- ***Source code:***

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

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

#define SLEEP_PROC_SEC  10
#define SLEEP_TREE_SEC  5

typedef struct tree_node* Tree;

void fork_newtree(Tree FirstNode) {
        unsigned children_nmbr = FirstNode->nr_children ;

  if (children_nmbr != 0) {
        pid_t new_pid;
            int i = 0 ;
        int status ;
        for(i=0; i<children_nmbr; i++) {
                new_pid = fork();
                if (new_pid < 0) {
                        perror("Error with children.");
                        exit(1);
                }
                else if ( new_pid == 0 ) {
                        change_pname((FirstNode->children+i)->name);
                        printf("%s: Created.\n", (FirstNode->children[i]).name );
                        fork_newtree(FirstNode->children+i);
                }
        }
        for (i=0; i<children_nmbr; i++) {
                new_pid = wait(&status);
           explain_wait_status(new_pid, status);
        }
        printf("%s: Exiting now...\n",FirstNode->name);
        exit( getpid() );
  }
        else {
```

```
                printf("%s: No children found...going to sleep.\n",(FirstNode->name));
                sleep(SLEEP_PROC_SEC);
                printf("%s: Exiting now...\n",(FirstNode->name));
                exit( getpid() );
            }
}
int main(int argc, char *argv[]) {
    if (argc != 2) {
        fprintf(stderr, "Usage: %s <input_tree_file>\n\n", argv[0]);
        exit(1);
    }
    pid_t  rootpid;
    Tree root = get_tree_from_file(argv[1]);
    int status;
    rootpid = fork() ;
    if (rootpid < 0 ) {
        perror("Error with root.\n");
        exit(1);
    }
    else if (rootpid==0) {
        change_pname(root->name);
        printf("%s: Created.\n", root->name);
        fork_newtree(root);
        exit(getpid());
    }

    sleep(SLEEP_TREE_SEC);

    show_pstree(rootpid);

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

    return 0 ;
}
```

- **Η έξοδος μας για την είσοδο που παίρνουμε από το αρχείο “proc.tree” :**



- **Ερωτήσεις:**

1) Παρατηρούμε ότι η σειρά με την οποία εμφανίζονται τα μηνύματα έναρξης και τερματισμού των διεργασιών εμφανίζονται κατά επίπεδο του δένδρου, αλλά με τυχαιότητα όσον αφορά την σειρά μεταξύ των διεργασιών του ίδιου επιπέδου. Εκτελώντας πολλές φορές τον ίδιο κώδικα παρατηρούμε ότι τα μηνύματα έναρξης εμφανίζονται ξεκινώντας από το επίπεδο της ρίζας και συνεχίζοντας προς τα χαμηλότερα επίπεδα. Αυτό είναι αναμενόμενο εφόσον η δημιουργία κάθε διεργασίας απαιτεί πρώτα την δημιουργία του γονέα της. Ομοίως, τα μηνύματα τερματισμού τον διεργασιών εμφανίζονται από το επίπεδο των φύλλων προς το επίπεδο της ρίζας. Το γεγονός αυτό είναι σε συμφωνία με τον περιορισμό κάθε διεργασία να περιμένει τον τερματισμό όλων των παιδιών της προτού τερματιστεί.

ΑΣΚΗΣΗ 2.3

- ***Source code:***

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

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

```
#define SLEEP_TREE_SEC  5

typedef struct tree_node* Tree;

void fork_newtree(Tree FirstNode) {
	unsigned children_nmbr = FirstNode->nr_children ;
	int status;


		pid_t new_pid;
		int i=0;
		for(i=0; i<children_nmbr; i++) { //if it has children then make them all sleep

			new_pid = fork();
			if (new_pid < 0) {

				perror("Error with children.");
				exit(1);
			}
			else if ( new_pid == 0 ) {

				change_pname((FirstNode->children+i)->name);
				printf("%s: Created.\n", (FirstNode->children+i)->name);
				fork_newtree(FirstNode->children+i);
				 printf("PID = %ld, name = %s is going to stop now.\n",(long)getpid(),
                                                        (FirstNode->children+i)->name);
				raise (SIGSTOP);

				exit(0);
			}
			(FirstNode->children+i)->pid = new_pid ;
		}
		 wait_for_ready_children(children_nmbr);
		raise(SIGSTOP);

		printf("PID = %ld, name = %s is now awake... \n" ,(long)getpid(), FirstNode->name );

		for ( i=0; i < children_nmbr; i++) { //every node sends a "wake up" signal to all of its child nodes with this loop

			kill((FirstNode->children+i)->pid , SIGCONT);
			int killedpid = wait(&status);
			explain_wait_status(killedpid , status);
			printf("\n");
		}

		printf("%s: Exiting now...\n",(FirstNode->name));
		exit( getpid() );
	}


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

	pid_t  rootpid;
```

```
    Tree root = get_tree_from_file(argv[1]);
    int status;

    rootpid = fork() ;
    if (rootpid < 0 ) {
        perror("Error with root.\n");
        exit(1);
    }
    else if (rootpid==0) {
        change_pname(root->name);
        printf("%s: Created.\n", root->name);
        fork_newtree(root);
        exit(getpid());
    }
    //father's code
    wait_for_ready_children(1);
    show_pstree(rootpid);
    kill(rootpid,SIGCONT);
    rootpid = wait(&status);
    explain_wait_status(rootpid,status);
    return 0 ;

}
```

- **Η έξοδος μας για το αρχείο εισόδου “proc.tree”:**

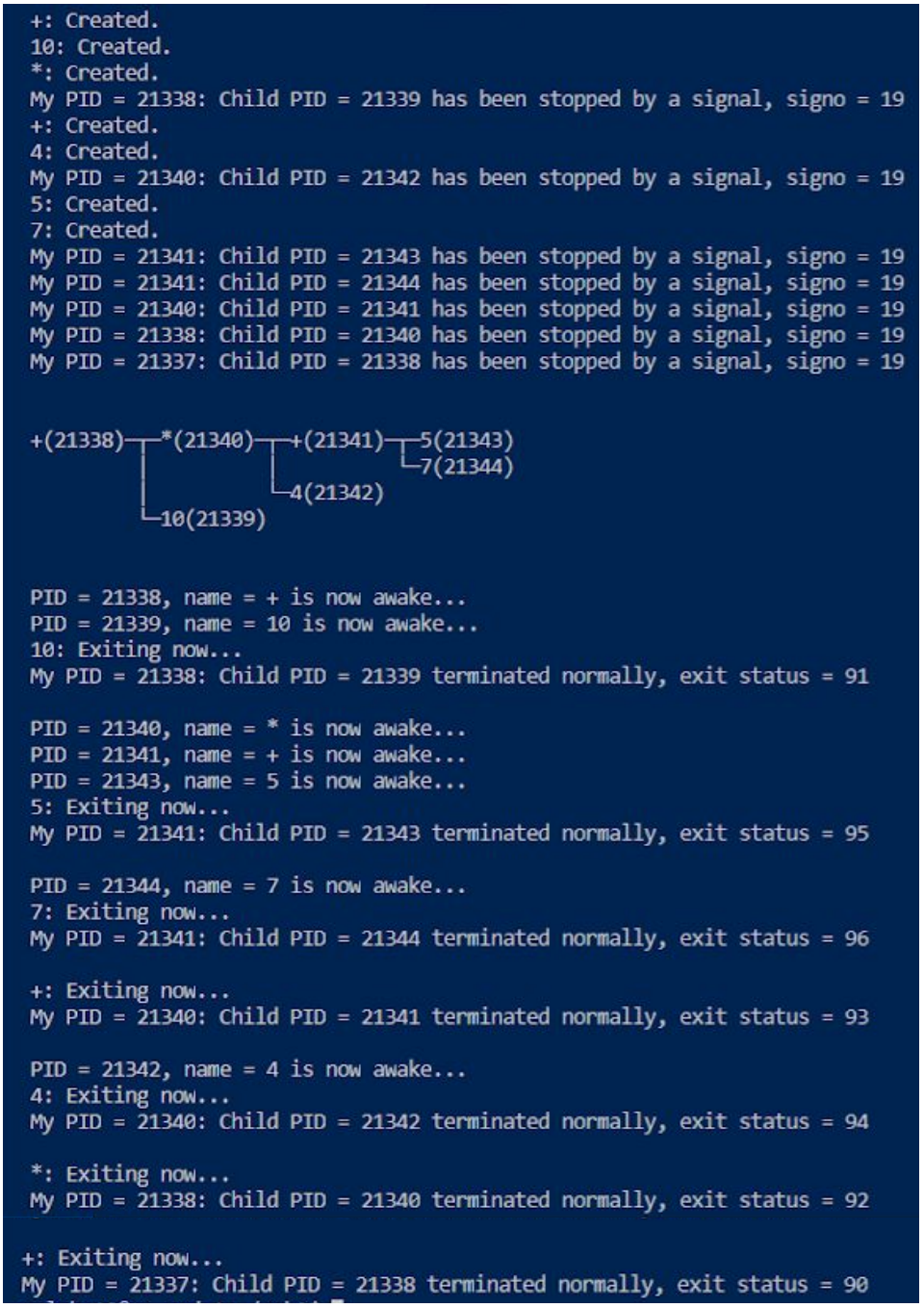
```
A: Created.
B: Created.
C: Created.
D: Created.
My PID = 20931: Child PID = 20933 has been stopped by a signal, signo = 19
My PID = 20931: Child PID = 20934 has been stopped by a signal, signo = 19
E: Created.
F: Created.
My PID = 20932: Child PID = 20935 has been stopped by a signal, signo = 19
My PID = 20932: Child PID = 20936 has been stopped by a signal, signo = 19
My PID = 20931: Child PID = 20932 has been stopped by a signal, signo = 19
My PID = 20930: Child PID = 20931 has been stopped by a signal, signo = 19


A(20931)─┬─B(20932)─┬─E(20935)
         │          └─F(20936)
         ├─C(20933)
         └─D(20934)


PID = 20931, name = A is now awake...
PID = 20932, name = B is now awake...
PID = 20935, name = E is now awake...
E: Exiting now...
My PID = 20932: Child PID = 20935 terminated normally, exit status = 199

PID = 20936, name = F is now awake...
F: Exiting now...
My PID = 20932: Child PID = 20936 terminated normally, exit status = 200

B: Exiting now...
My PID = 20931: Child PID = 20932 terminated normally, exit status = 196

PID = 20933, name = C is now awake...
C: Exiting now...
My PID = 20931: Child PID = 20933 terminated normally, exit status = 197

PID = 20934, name = D is now awake...
D: Exiting now...
My PID = 20931: Child PID = 20934 terminated normally, exit status = 198

A: Exiting now...
My PID = 20930: Child PID = 20931 terminated normally, exit status = 195
```

| Δημιουργία των διαδικασιών
| και
| ενημέρωση ότι κάθε
| διαδικασία έλαβε το σήμα
| STOPSIG.

|Εκτύπωση του
δένδρου.

|Έξοδος κι επανεκκίνηση
|των διαδικασιών με
|Depth-First Traversal.

- **Η έξοδος μας για το αρχείο εισόδου “expr.tree” :**



- **Ερωτήσεις :**

1) Η συνάρτηση “sleep()” που χρησιμοποιήθηκε στα προηγούμενα ερωτήματα (πληροφορίες γι’αυτήν με “man sleep”) αναστέλει τη λειτουργία της διεργασίας για την οποία καλείται για χρονικό διάστημα ίσο με το όρισμα της (δίνεται η δυνατότητα να επιλέξουμε και αριθμητική τιμή και μονάδα μέτρησης). Χρησιμοποιώντας την συνάρτηση sleep() άρα είναι δύσκολο και ίσως μη αποδοτικό να συγχρονίσουμε τις διεργασίες, αφού προσπαθούμε να επιλέξουμε κατάλληλα χρονικά διστήματα αναμονής (πολλές φορές μεγαλύτερα από ότι χρειάζεται) τέτοια ώστε να μην υπάρχουν μη επιθυμητές “επικαλύψεις” μεταξύ διεργασιών που πρέπει να δράσουν σειριακά.

Αντιθέτως, η χρήση σημάτων μας επιτρέπει να “παγώνουμε” την εκτέλεση μίας διαδικασίας μέσω του σήματος “SIGSTOP” και να την συνεχίζουμε όταν κρίνουμε ότι έρθει η κατάλληλη στιγμή στέλνοντας της μέσω μίας άλλης διεργασίας το σήμα “SIGCONT”. Με αυτόν τον τρόπο υλοποιείται μία πιο αποτελεσματική επικοινωνία μεταξύ των διεργασίων. Οι διαδικασίες παιδιά τίθενται σε αναμονή και ενεργοποιούνται όταν χρειάζεται από τις διαδικασίες γονείς τους, χωρίς να χρειάζεται να θέτονται σε αναμονή για περισσότερο χρονικό διάστημα από το απολύτως αναγκαίο.

2) Η συνάρτηση “wait_for_children” που δίνεται στο αρχείο πηγαίου κώδικα “proc-common.c” εξασφαλίζει ότι μια διεργασία πατέρας περιμένει όλες οι διεργασίες παιδιά της να είναι σε αναμονή έχοντας δεχτεί το σήμα “SIGSTOP” προτού τις ενεργοποιήσουμε εμείς εκ νέου με το σήμα “SIGCONT”. ΜΕ αυτόν τον τρόπο και σε συνεργασία με τις ανταλλαγές των σημάτων εξασφαλίζουμε ότι υπάρχει στιγμή που όλες οι διαδικασίες είναι ενεργές με σκοπό να μπορούμε να σχεδιάσουμε το δένδρο. Ταυτόχρονα, ελέγχουμε και τη σειρά ενεργοποίησης των διεργασιών επιτυγχάνοντας αυτή να γίνεται με διάσχιση κατά βάθος.

ΑΣΚΗΣΗ 2.4

- ***Source code:***

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <signal.h>
#include <assert.h>

#include <sys/wait.h>
#include "tree.h"
#include "proc-common.h"

#define SLEEP_PROC_SEC 3


void my_pipe (struct tree_node* current, int fd) {

        printf("PID = %ld, name = %s, Starting...\n", (long)getpid(), current->name);

        change_pname(current->name);

        double val_to_write;

        if (current->nr_children!=0) {      // ean den einai termatikos komvos

      pid_t p1, p2;
      int status;
      int pfd0[2];
      double data1;
      printf("%s: Creating pipe...\n", current->name);
      if (pipe(pfd0)<0) {
            perror("pipe");
            exit(1);
      }

      p1=fork();                  // dhmiourgia 1ou paidiou
      if (p1<0) {
            /*fork failed*/
```

```
        perror("fork");
        exit(1);
}
if (p1==0) {                    // 1st child's code
        printf("%s: Created 1st child process with PID=%ld\n", current->name, (long)getpid());
        my_pipe(current->children, pfd0[1]);       // kaloume anadromika thn my_pipe gia to 1o paidi
}


p2=fork();                      // dhmiourgia 2ou paidiou
if (p2<0) {
        /* fork failed...*/
        perror("fork");
        exit(1);
}
if (p2==0) {                    // 2nd child's code
        printf("%s: Created the 2nd child process with PID=%ld\n", current->name, (long)getpid());
        my_pipe(current->children+1, pfd0[1]);
}



         // wait for your 2 children
// waitpid(p1, &status, 0);
// explain_wait_status(p1, status);

// waitpid(p2, &status, 0);
// explain_wait_status(p2, status);


                p1=wait(&status);
          explain_wait_status(p1, status);

                p2=wait(&status);
          explain_wait_status(p2, status);



if (read (pfd0[0], &data1, sizeof(data1))!=sizeof(data1)) { // read from the first child
        perror("Read from pipe the 1st number");
        exit(1);
}
val_to_write=data1;

printf("%s: Read %f from 1st child process\n", current->name, data1);

if (read (pfd0[0], &data1, sizeof(data1))!=sizeof(data1)) { // read from the second child
        perror("Read from pipe the 2nd number");
        exit(1);
}
printf("%s: Read %f from 2nd child process\n", current->name, data1);

// compute the result acoording to current node's data
if ( *(current->name)=='+') {
        val_to_write = val_to_write+data1;
}
else {
        val_to_write = val_to_write*data1;
```

```
            }

            // write the result to the pipe connecting you with your parent node
            if (write(fd, &val_to_write, sizeof(val_to_write))!=sizeof(val_to_write)) {
                perror("write to pipe");
                exit(1);
            }
            printf("%s: Wrote %f to the pipe\n", current->name, val_to_write);

            // printf("%s: Sleeping...\n", current->name);
            // sleep(SLEEP_PROC_SEC);


                        printf("%s: All done, exiting...\n", current->name);
            exit(val_to_write);
        }

        else {              // leaf node, we just have to read a number

            val_to_write=atoi(current->name);               // read this number

                        if (write(fd, &val_to_write, sizeof(val_to_write))!=sizeof(val_to_write)) {      // send it to the parent
                perror(" write to pipe");
                exit(1);
            }
            printf("Leaf %s: Wrote %f to the pipe\n", current->name, val_to_write);

            printf("%s: Sleeping...\n", current->name);
            sleep(SLEEP_PROC_SEC);
            printf("%s: Exiting...\n", current->name);
            exit(val_to_write);
        }
}

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

        pid_t pid;
        int status;

        int pfd_main[2];  // dyo akra tou pipe

        double output;
        struct tree_node* root;

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

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

        root=get_tree_from_file(argv[1]);         // diavasma tou dendrou eisodou apo to orisma

        printf("Parent: Creating child process...\n");
        pid=fork();
```

```
    if (pid<0) {
        perror("main: fork");
        exit(1);
    }
    if (pid==0) {                       // root's code
        printf("Parent created the root with PID=%ld\n", (long)getpid());

        printf("Parent: Creating pipe...\n");
        my_pipe(root, pfd_main[1]);
        exit(1);
    }

    if (read(pfd_main[0], &output, sizeof(output))!=sizeof(output)) {          // when the result is ready
        perror("parent: Read from pipe");
        exit(1);
    }
    printf("The final result is: %f\n", output);

        // wait for the root to terminate
    pid=wait(&status);
    explain_wait_status(pid, status);

        printf("Parent: All done, exiting\n");
    return 0;

}
```

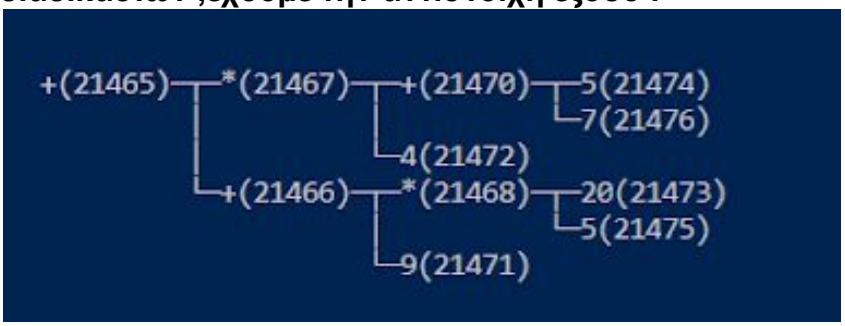
- **Η έξοδος μας για το αρχείο εισόδου “expr.tree” :**

```
Parent: Creating pipe...
Parent: Creating child process...
Parent created the root with PID=21371
Parent: Creating pipe...
PID = 21371, name = +, Starting...
+: Creating pipe...
+: Created 1st child process with PID=21372
PID = 21372, name = 10, Starting...
+: Created the 2nd child process with PID=21373
trying to write 10.000000 to the pipe...
PID = 21373, name = *, Starting...
Leaf 10: Wrote 10.000000 to the pipe
10: Sleeping...
*: Creating pipe...
*: Created 1st child process with PID=21374
PID = 21374, name = +, Starting...
+: Creating pipe...
*: Created the 2nd child process with PID=21375
PID = 21375, name = 4, Starting...
trying to write 4.000000 to the pipe...
Leaf 4: Wrote 4.000000 to the pipe
4: Sleeping...
+: Created 1st child process with PID=21376
PID = 21376, name = 5, Starting...
trying to write 5.000000 to the pipe...
Leaf 5: Wrote 5.000000 to the pipe
5: Sleeping...
+: Created the 2nd child process with PID=21377
PID = 21377, name = 7, Starting...
trying to write 7.000000 to the pipe...
Leaf 7: Wrote 7.000000 to the pipe
7: Sleeping...
10: Exiting...
My PID = 21371: Child PID = 21372 terminated normally, exit status = 10
4: Exiting...
5: Exiting...
7: Exiting...
My PID = 21374: Child PID = 21376 terminated normally, exit status = 5
My PID = 21374: Child PID = 21377 terminated normally, exit status = 7
+: Read 5.000000 from 1st child process
+: Read 7.000000 from 2nd child process
+: Wrote 12.000000 to the pipe
+: Sleeping...
+: All done, exiting...
My PID = 21373: Child PID = 21374 terminated normally, exit status = 12
My PID = 21373: Child PID = 21375 terminated normally, exit status = 4
*: Read 4.000000 from 1st child process
*: Read 12.000000 from 2nd child process
*: Wrote 48.000000 to the pipe
*: Sleeping...
*: All done, exiting...
My PID = 21371: Child PID = 21373 terminated normally, exit status = 48
+: Read 10.000000 from 1st child process
+: Read 48.000000 from 2nd child process
+: Wrote 58.000000 to the pipe
+: Sleeping...
The final result is: 58.000000
+: All done, exiting...
My PID = 21370: Child PID = 21371 terminated normally, exit status = 58
Parent: All done, exiting
```

|Για κάθε διαδικασία το exit status |είναι ο αριθμός ή το αποτέλεσμα |πράξης που μεταφέρει στη |διεργασία γονιό της.

|Εκτύπωση του τελικού |αποτελέσματος.(58)

- **Έπειτα με κατάλληλο αρχείο εισόδου που παράγει το παρακάτω δένδρο διαδικασιών ,έχουμε την αντίστοιχη έξοδο :**




```
Parent: Creating pipe...
Parent: Creating child process...
Parent created the root with PID=21503
Parent: Creating pipe...
PID = 21503, name = +, Starting...
+: Creating pipe...
+: Created 1st child process with PID=21504
PID = 21504, name = +, Starting...
+: Creating pipe...
+: Created the 2nd child process with PID=21505
PID = 21505, name = *, Starting...
*: Creating pipe...
+: Created 1st child process with PID=21506
PID = 21506, name = *, Starting...
*: Creating pipe...
*: Created 1st child process with PID=21507
PID = 21507, name = +, Starting...
+: Created the 2nd child process with PID=21508
+: Creating pipe...
PID = 21508, name = 9, Starting...
*: Created the 2nd child process with PID=21509
PID = 21509, name = 4, Starting...
Leaf 9: Wrote 9.000000 to the pipe
9: Sleeping...
Leaf 4: Wrote 4.000000 to the pipe
4: Sleeping...
+: Created 1st child process with PID=21511
PID = 21511, name = 5, Starting...
*: Created the 2nd child process with PID=21512
*: Created 1st child process with PID=21510
PID = 21512, name = 5, Starting...
Leaf 5: Wrote 5.000000 to the pipe
PID = 21510, name = 20, Starting...
5: Sleeping...
+: Created the 2nd child process with PID=21513
PID = 21513, name = 7, Starting...
Leaf 5: Wrote 5.000000 to the pipe
Leaf 20: Wrote 20.000000 to the pipe
20: Sleeping...
Leaf 7: Wrote 7.000000 to the pipe
7: Sleeping...
5: Sleeping...
9: Exiting...
4: Exiting...
5: Exiting...
20: Exiting...
```

```
7: Exiting...
5: Exiting...
My PID = 21507: Child PID = 21511 terminated normally, exit status = 5
My PID = 21506: Child PID = 21510 terminated normally, exit status = 20
My PID = 21507: Child PID = 21513 terminated normally, exit status = 7
My PID = 21506: Child PID = 21512 terminated normally, exit status = 5
+: Read 5.000000 from 1st child process
+: Read 7.000000 from 2nd child process
*: Read 20.000000 from 1st child process
+: Wrote 12.000000 to the pipe
+: Sleeping...
*: Read 5.000000 from 2nd child process
*: Wrote 100.000000 to the pipe
*: Sleeping...
+: All done, exiting...
*: All done, exiting...
My PID = 21505: Child PID = 21507 terminated normally, exit status = 12
My PID = 21504: Child PID = 21506 terminated normally, exit status = 100
My PID = 21505: Child PID = 21509 terminated normally, exit status = 4
My PID = 21504: Child PID = 21508 terminated normally, exit status = 9
*: Read 4.000000 from 1st child process
*: Read 12.000000 from 2nd child process
*: Wrote 48.000000 to the pipe
*: Sleeping...
+: Read 9.000000 from 1st child process
+: Read 100.000000 from 2nd child process
+: Wrote 109.000000 to the pipe
+: Sleeping...
+: All done, exiting...
My PID = 21503: Child PID = 21504 terminated normally, exit status = 109
*: All done, exiting...
My PID = 21503: Child PID = 21505 terminated normally, exit status = 48
+: Read 48.000000 from 1st child process
+: Read 109.000000 from 2nd child process
+: Wrote 157.000000 to the pipe
+: Sleeping...
The final result is: 157.000000
+: All done, exiting...
My PID = 21502: Child PID = 21503 terminated normally, exit status = 157
Parent: All done, exiting
```

|Τελικό αποτέλεσμα:157

Ερωτήσεις:

1) Στην άσκηση αυτή επιτύχαμε την ανταλλαγή δεδομένων μεταξύ των διεργασιών με την χρήση σωληνώσεων(pipes). Χρησιμοποιήσαμε για κάθε διεργασία φύλλο ένα pipe στο οποίο γράφει το όνομά της, δηλαδή την τιμή που περιέχει. Για κάθε μη τερματικό κόμβο χρησιμοποιήσαμε δύο pipes, ένα από όπου διαβάζει τα δεδομένα που του στέλνουν τα δύο παιδιά του (και τα οποία επεξεργάζεται κατάλληλα με βάση τον τελεστή-όνομά του) και ένα όπου γράφει τα δεδομένα που θα στείλει προς τον πατέρα του. Γίνεται, επομένως, κάθε διεργασία να χρησιμοποιεί μία σωλήνωση για

όλες τις διεργασίες παιδιά της. Στην συγκεκριμένη εφαρμογή δεν γίνεται να χρησιμοποιηθεί μία μόνο σωλήνωση για κάθε τελεστή, δεδομένου ότι για την αποστολή των δεδομένων στην διεργασία-πατέρα του τελεστή χρειάζονται δύο βήματα (1:ο κατάλληλος συνδυασμός των δεδομένων των διεργασιών παιδιών και 2: η αποστολή τους) και κάθε ένα από αυτά απαιτεί μία σωλήνωση.

Ωστόσο θα μπορούσε να υλοποιηθεί σαν ιδέα ένα “globally ορισμένο” pipe το οποίο δε θα δίναμε σαν όρισμα στην αναδρομική συνάρτηση δημιουργίας διαδικασιών αλλά θα το χειριζόμασταν ως εξής. Αν υλοποιούσαμε με έναν DFS τρόπο τον υπολογισμό μας ,θα μπορούσαμε να χρησιμοποιούμε το ίδιο pipe για κάθε read & write αναδρομικά μέχρι να φτάναμε στη ρίζα του δέντρου. Φυσικά σε μία τέτοια επιλογή ελοχεύουν κίνδυνοι λάθος διαχείρισης των δεδομένων και εν τέλει κίνδυνος εξαγωγής λάθος αποτελέσματος.

2) Σε ένα σύστημα πολλαπλών επεξεργαστών η χρήση του δένδρου διεργασιών καθιστά εφικτό και αποδοτικό τον συνδυασμό των αποτελεσμάτων των επιμέρους διεργασιών με βάση τη δομή του δένδρου. Έτσι, αξιοποιούμε σε μεγαλύτερο βαθμό την δυνατότητα παράλληλης εκτέλεσης των διεργασιών, αφού η αποτίμηση μίας έκφρασης από μία μόνο διεργασία θα χρησιμοποιεί μικρό μέρος των δυνατοτήτων του υλικού(έναν επεξεργαστή).