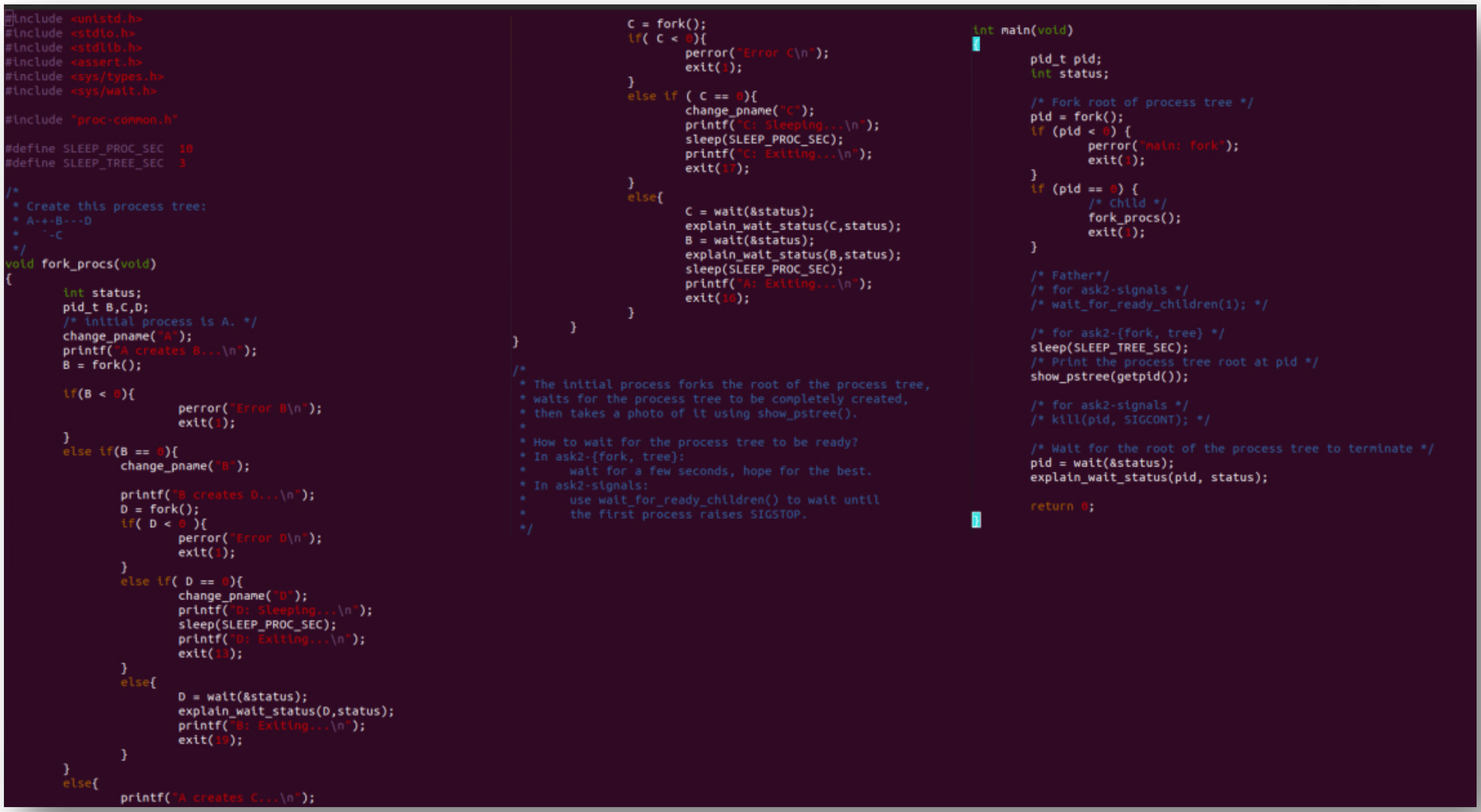
ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ

2η Άσκηση: Διαχείρηση Διεργασιών και Διαδιεργασιακή Επικοινωνία

ΦΟΙΤΗΤΕΣ: ΚΥΡΙΑΚΟΥ ΔΗΜΗΤΡΗΣ 03117601
ΧΑΤΖΗΧΡΙΣΤΟΦΗ ΧΡΙΣΤΟΣ 03117711
ΟΜΑΔΑ: OSLABC16
ΕΞΑΜΗΝΟ: 6ο

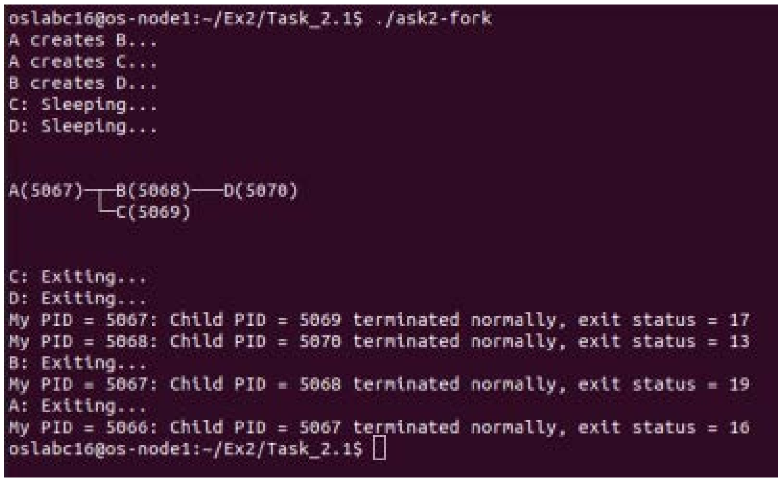
1.1 Δημιουργία δεδομένου δέντρου διεργασιών

Πηγαίος κώδικας:



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

#include "proc-common.h"

#define SLEEP_PROC_SEC  10
#define SLEEP_TREE_SEC  3

/*
 * Create this process tree:
 * A-+-B---D
 *    `-C
 */
void fork_procs(void)
{
        int status;
        pid_t B,C,D;
        /* initial process is A. */
        change_pname("A");
        printf("A creates B...\n");
        B = fork();

        if(B < 0){
                        perror("Error B\n");
                        exit(1);
        }
        else if(B == 0){
                change_pname("B");

                printf("B creates D...\n");
                D = fork();
                if( D < 0 ){
                        perror("Error D\n");
                        exit(1);
                }
                else if( D == 0){
                        change_pname("D");
                        printf("D: Sleeping...\n");
                        sleep(SLEEP_PROC_SEC);
                        printf("D: Exiting...\n");
                        exit(13);
                }
                else{
                        D = wait(&status);
                        explain_wait_status(D,status);
                        printf("B: Exiting...\n");
                        exit(19);
                }
        }
        else{
                printf("A creates C...\n");
                C = fork();
                if( C < 0){
                        perror("Error C\n");
                        exit(1);
                }
                else if ( C == 0){
                        change_pname("C");
                        printf("C: Sleeping...\n");
                        sleep(SLEEP_PROC_SEC);
                        printf("C: Exiting...\n");
                        exit(17);
                }
                else{
                        C = wait(&status);
                        explain_wait_status(C,status);
                        B = wait(&status);
                        explain_wait_status(B,status);
                        sleep(SLEEP_PROC_SEC);
                        printf("A: Exiting...\n");
                        exit(16);
                }
        }
}

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

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

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

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

        /* for ask2-{fork, tree} */
        sleep(SLEEP_TREE_SEC);
        /* Print the process tree root at pid */
        show_pstree(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;
}
```

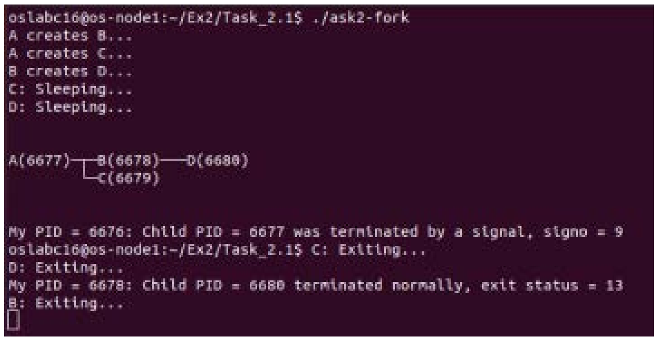
Έξοδος εκτέλεσης:



```
oslabc16@os-node1:~/Ex2/Task_2.1$ ./ask2-fork
A creates B...
A creates C...
B creates D...
C: Sleeping...
D: Sleeping...


A(5067)─┬─B(5068)───D(5070)
        └─C(5069)


C: Exiting...
D: Exiting...
My PID = 5067: Child PID = 5069 terminated normally, exit status = 17
My PID = 5068: Child PID = 5070 terminated normally, exit status = 13
B: Exiting...
My PID = 5067: Child PID = 5068 terminated normally, exit status = 19
A: Exiting...
My PID = 5066: Child PID = 5067 terminated normally, exit status = 16
oslabc16@os-node1:~/Ex2/Task_2.1$ 
```

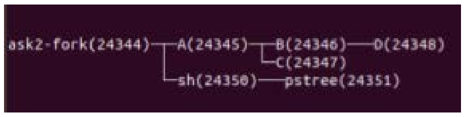
Ερωτήσεις

1. Στην περίπτωση που δίνεται kill για τη διεργασία A αυτή τερματίζεται βίαια και επιστρέφει signal 9 (μήνυμα τερματισμού me kill) σε αντίθεση με το μήνυμα terminated normally που εμφάνιζε πριν. Το πιο πάνω μήνυμα επιστρέφεται από τη διεργασία ρίζας του προγράμματος (πατέρας της διεργασίας A). Τα παιδιά της διεργασίας A γίνονται παιδιά της διεργασίας ρίζας του προγράμματος συνεχίζουν και τερματίζουν κανονικά όμως χωρίς να εμφανίζονται τα μηνύματα τερματισμού αφού η διεργασία πατέρας τους (διεργασία A) τερματίστηκε. Από την άλλη, η διεργασία D εμφανίζει μήνυμα αφού η διεργασία πατέρας της είναι η B.

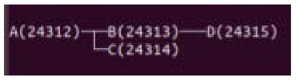


```
oslabc16@os-node1:~/Ex2/Task_2.1$ ./ask2-fork
A creates B...
A creates C...
B creates D...
C: Sleeping...
D: Sleeping...


A(6677)─┬─B(6678)───D(6680)
        └─C(6679)


My PID = 6676: Child PID = 6677 was terminated by a signal, signo = 9
oslabc16@os-node1:~/Ex2/Task_2.1$ C: Exiting...
D: Exiting...
My PID = 6678: Child PID = 6680 terminated normally, exit status = 13
B: Exiting...
```

2. Στην περίπτωση που στη συνάρτηση show_pstree δίδεται σαν παράμετρος το getpid() αντί του pid, κατά την εκτύπωση του δέντρου εκτυπώνονται οι διεργασίες του προγράμματος (A,B,C,D), η διεργασία φλοιού και η διεργασία που δείχνει τις τρέχουσες διεργασίες σε μορφή δέντρου.



```
ask2-fork(24344)─┬─A(24345)─┬─B(24346)───D(24348)
                 │          └─C(24347)
                 └─sh(24350)───pstree(24351)
```

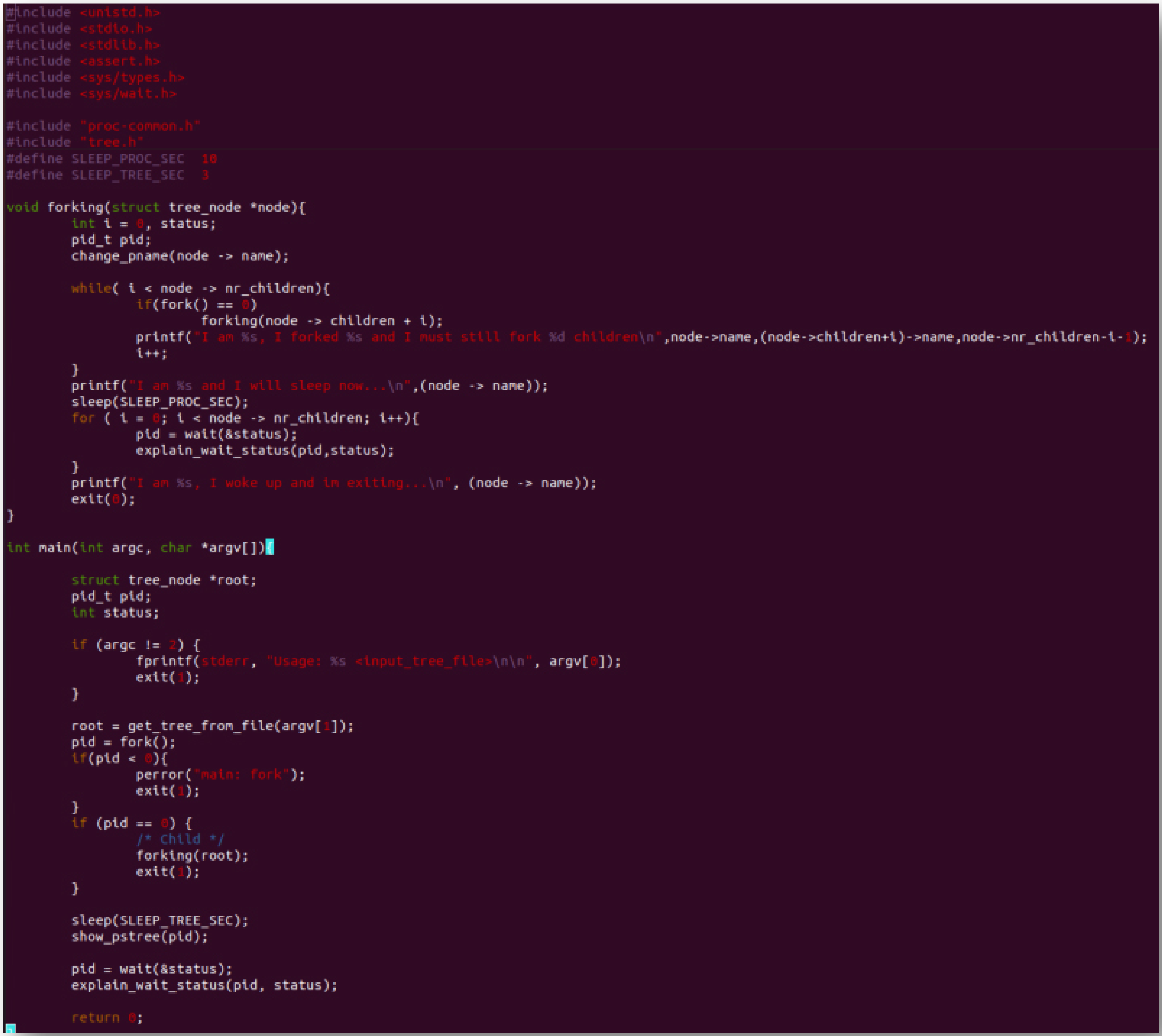


```
A(24312)─┬─B(24313)───D(24315)
         └─C(24314)
```

3. Τα όρια τίθενται από τους διαχειριστές διότι υπάρχει ο κίνδυνος υπερφόρτωσης του υπολογιστικού συστήματος με τεράστιο αριθμό διεργασιών το οποίο μπορεί να έχει ως αποτέλεσμα την κατάληψη αρκετής μνήμης ή και να επηρεάσει την ταχύτητα του.

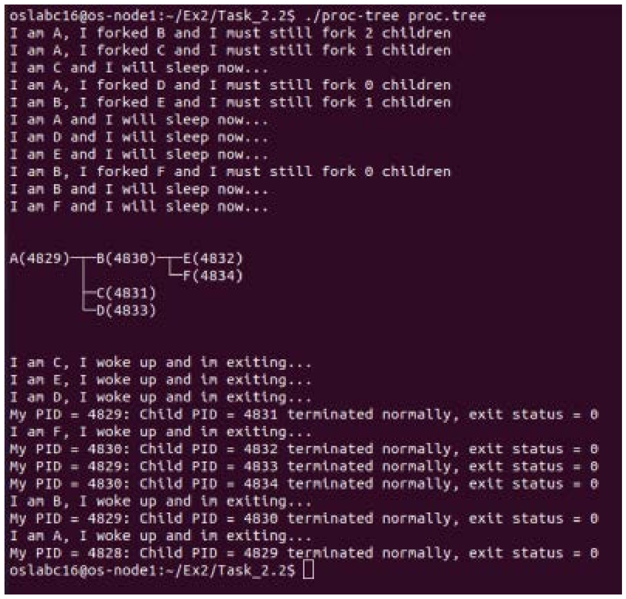
1.2 Δημιουργία αυθαίρετου δέντρου διεργασιών

Πηγαίος κώδικας:



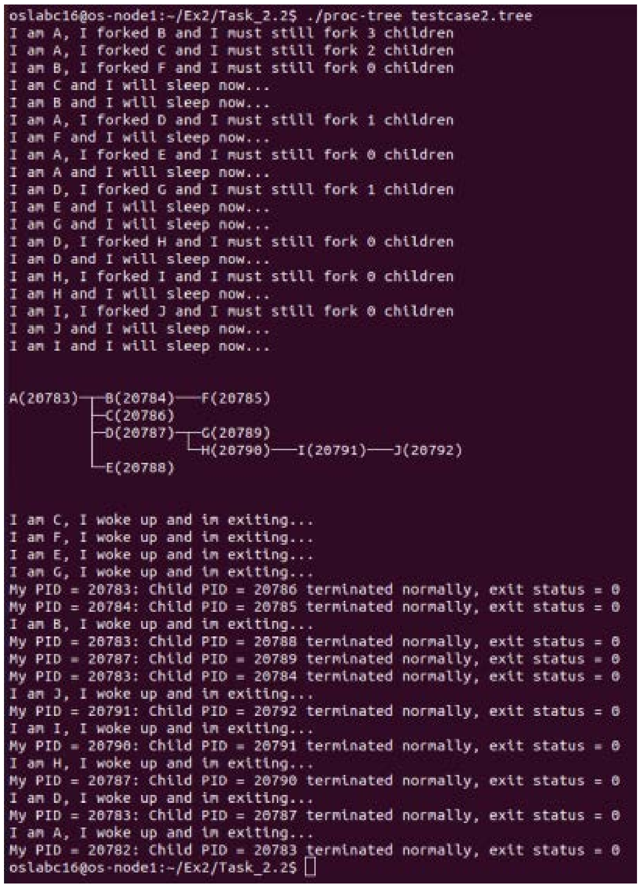
```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <sys/types.h>
#include <sys/wait.h>

#include "proc-common.h"
#include "tree.h"
#define SLEEP_PROC_SEC  10
#define SLEEP_TREE_SEC  3

void forking(struct tree_node *node){
        int i = 0, status;
        pid_t pid;
        change_pname(node -> name);

        while( i < node -> nr_children){
                if(fork() == 0)
                        forking(node -> children + i);
                printf("I am %s, I forked %s and I must still fork %d children\n",node->name,(node->children+i)->name,node->nr_children-i-1);
                i++;
        }
        printf("I am %s and I will sleep now...\n",(node -> name));
        sleep(SLEEP_PROC_SEC);
        for ( i = 0; i < node -> nr_children; i++){
                pid = wait(&status);
                explain_wait_status(pid,status);
        }
        printf("I am %s, I woke up and im exiting...\n", (node -> name));
        exit(0);
}

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

        struct tree_node *root;
        pid_t pid;
        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");
                exit(1);
        }
        if (pid == 0) {
                /* Child */
                forking(root);
                exit(1);
        }

        sleep(SLEEP_TREE_SEC);
        show_pstree(pid);

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

        return 0;
```

Έξοδος εκτέλεσης 1:



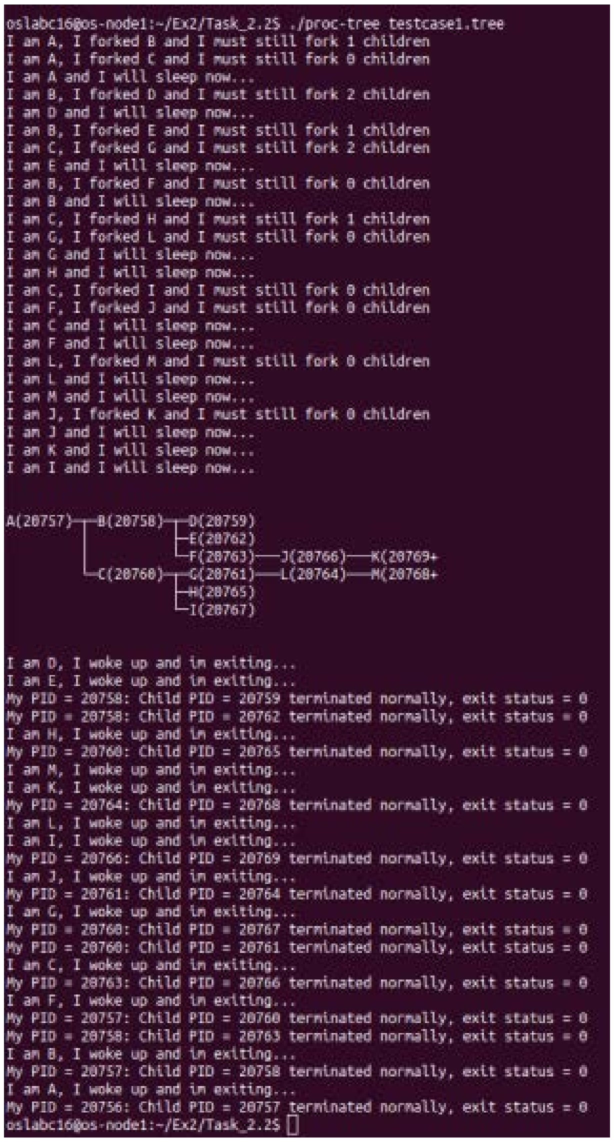
```
oslabc16@os-node1:~/Ex2/Task_2.2$ ./proc-tree proc.tree
I am A, I forked B and I must still fork 2 children
I am A, I forked C and I must still fork 1 children
I am C and I will sleep now...
I am A, I forked D and I must still fork 0 children
I am B, I forked E and I must still fork 1 children
I am A and I will sleep now...
I am D and I will sleep now...
I am E and I will sleep now...
I am B, I forked F and I must still fork 0 children
I am B and I will sleep now...
I am F and I will sleep now...


A(4829)─┬─B(4830)─┬─E(4832)
        │         └─F(4834)
        ├─C(4831)
        └─D(4833)


I am C, I woke up and im exiting...
I am E, I woke up and im exiting...
I am D, I woke up and im exiting...
My PID = 4829: Child PID = 4831 terminated normally, exit status = 0
I am F, I woke up and im exiting...
My PID = 4830: Child PID = 4832 terminated normally, exit status = 0
My PID = 4829: Child PID = 4833 terminated normally, exit status = 0
My PID = 4830: Child PID = 4834 terminated normally, exit status = 0
I am B, I woke up and im exiting...
My PID = 4829: Child PID = 4830 terminated normally, exit status = 0
I am A, I woke up and im exiting...
My PID = 4828: Child PID = 4829 terminated normally, exit status = 0
oslabc16@os-node1:~/Ex2/Task_2.2$ 
```

Έξοδος εκτέλεσης 2:



```
oslabc16@os-node1:~/Ex2/Task_2.2$ ./proc-tree testcase2.tree
I am A, I forked B and I must still fork 3 children
I am A, I forked C and I must still fork 2 children
I am B, I forked F and I must still fork 0 children
I am C and I will sleep now...
I am B and I will sleep now...
I am A, I forked D and I must still fork 1 children
I am F and I will sleep now...
I am A, I forked E and I must still fork 0 children
I am A and I will sleep now...
I am D, I forked G and I must still fork 1 children
I am E and I will sleep now...
I am G and I will sleep now...
I am D, I forked H and I must still fork 0 children
I am D and I will sleep now...
I am H, I forked I and I must still fork 0 children
I am H and I will sleep now...
I am I, I forked J and I must still fork 0 children
I am J and I will sleep now...
I am I and I will sleep now...


A(20783)─┬─B(20784)───F(20785)
         ├─C(20786)
         ├─D(20787)─┬─G(20789)
         │          └─H(20790)───I(20791)───J(20792)
         └─E(20788)


I am C, I woke up and im exiting...
I am F, I woke up and im exiting...
I am E, I woke up and im exiting...
I am G, I woke up and im exiting...
My PID = 20783: Child PID = 20786 terminated normally, exit status = 0
My PID = 20784: Child PID = 20785 terminated normally, exit status = 0
I am B, I woke up and im exiting...
My PID = 20783: Child PID = 20788 terminated normally, exit status = 0
My PID = 20787: Child PID = 20789 terminated normally, exit status = 0
My PID = 20783: Child PID = 20784 terminated normally, exit status = 0
I am J, I woke up and im exiting...
My PID = 20791: Child PID = 20792 terminated normally, exit status = 0
I am I, I woke up and im exiting...
My PID = 20790: Child PID = 20791 terminated normally, exit status = 0
I am H, I woke up and im exiting...
My PID = 20787: Child PID = 20790 terminated normally, exit status = 0
I am D, I woke up and im exiting...
My PID = 20783: Child PID = 20787 terminated normally, exit status = 0
I am A, I woke up and im exiting...
My PID = 20782: Child PID = 20783 terminated normally, exit status = 0
oslabc16@os-node1:~/Ex2/Task_2.2$ 
```

Έξοδος εκτέλεσης 3:



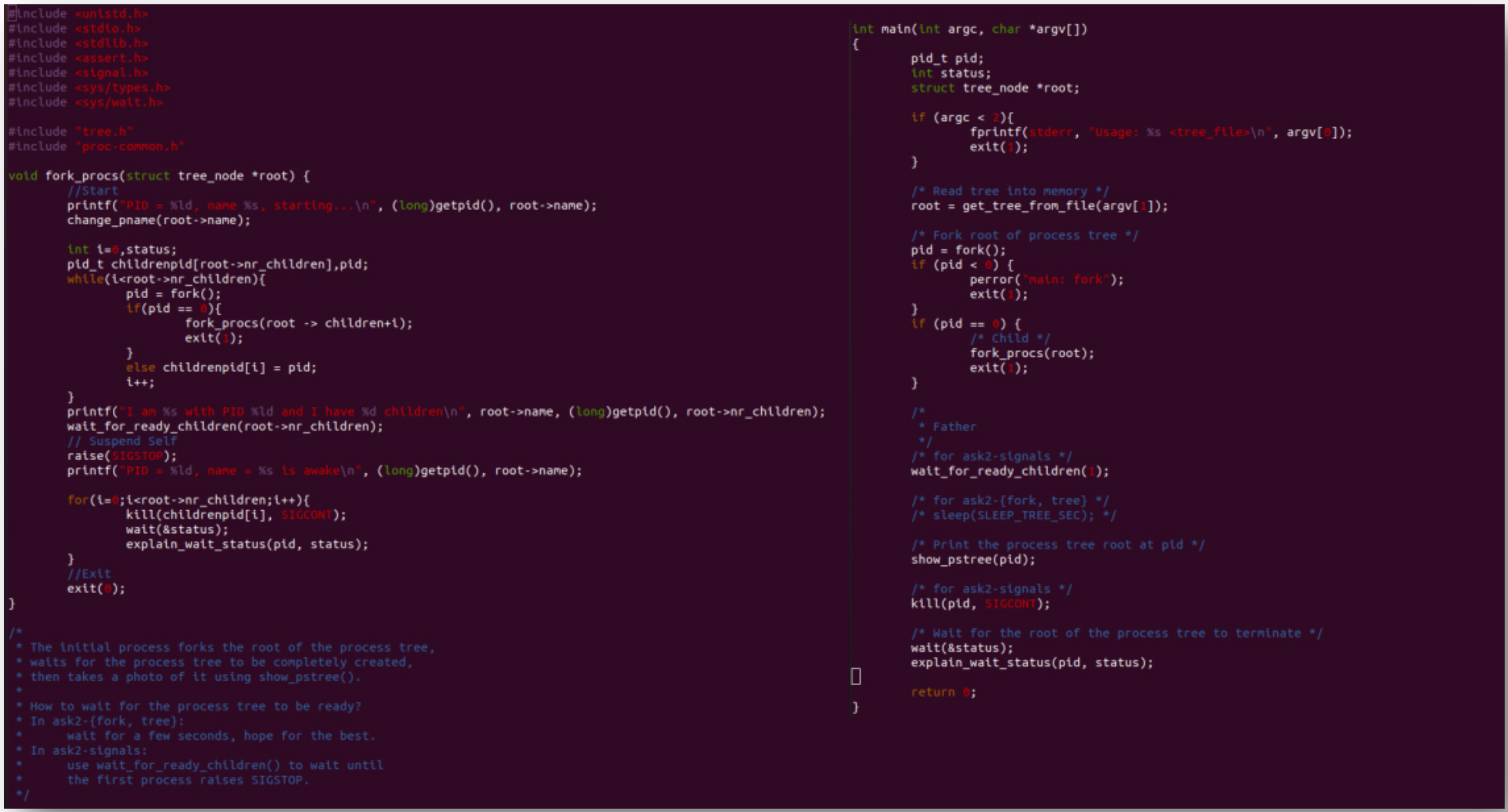
```
oslabc16@os-node1:~/Ex2/Task_2.2$ ./proc-tree testcase1.tree
I am A, I forked B and I must still fork 1 children
I am A, I forked C and I must still fork 0 children
I am A and I will sleep now...
I am B, I forked D and I must still fork 2 children
I am D and I will sleep now...
I am B, I forked E and I must still fork 1 children
I am C, I forked G and I must still fork 2 children
I am E and I will sleep now...
I am B, I forked F and I must still fork 0 children
I am B and I will sleep now...
I am C, I forked H and I must still fork 1 children
I am G, I forked L and I must still fork 0 children
I am G and I will sleep now...
I am H and I will sleep now...
I am C, I forked I and I must still fork 0 children
I am F, I forked J and I must still fork 0 children
I am C and I will sleep now...
I am F and I will sleep now...
I am L, I forked M and I must still fork 0 children
I am L and I will sleep now...
I am M and I will sleep now...
I am J, I forked K and I must still fork 0 children
I am J and I will sleep now...
I am K and I will sleep now...
I am I and I will sleep now...


A(20757)─┬─B(20758)─┬─D(20759)
         │          ├─E(20762)
         │          └─F(20763)───J(20766)───K(20769+
         └─C(20760)─┬─G(20761)───L(20764)───M(20768+
                    ├─H(20765)
                    └─I(20767)


I am D, I woke up and im exiting...
I am E, I woke up and im exiting...
My PID = 20758: Child PID = 20759 terminated normally, exit status = 0
My PID = 20758: Child PID = 20762 terminated normally, exit status = 0
I am H, I woke up and im exiting...
My PID = 20760: Child PID = 20765 terminated normally, exit status = 0
I am M, I woke up and im exiting...
I am K, I woke up and im exiting...
My PID = 20764: Child PID = 20768 terminated normally, exit status = 0
I am L, I woke up and im exiting...
I am I, I woke up and im exiting...
My PID = 20766: Child PID = 20769 terminated normally, exit status = 0
I am J, I woke up and im exiting...
My PID = 20761: Child PID = 20764 terminated normally, exit status = 0
I am G, I woke up and im exiting...
My PID = 20760: Child PID = 20767 terminated normally, exit status = 0
My PID = 20760: Child PID = 20761 terminated normally, exit status = 0
I am C, I woke up and im exiting...
My PID = 20763: Child PID = 20766 terminated normally, exit status = 0
I am F, I woke up and im exiting...
My PID = 20757: Child PID = 20760 terminated normally, exit status = 0
My PID = 20758: Child PID = 20763 terminated normally, exit status = 0
I am B, I woke up and im exiting...
My PID = 20757: Child PID = 20758 terminated normally, exit status = 0
I am A, I woke up and im exiting...
My PID = 20756: Child PID = 20757 terminated normally, exit status = 0
oslabc16@os-node1:~/Ex2/Task_2.2$ 
```

Ερωτήσεις

1. Τα μηνύματα έναρξης και τερματισμού των διεργασιών εμφανίζονται με γενικά απροσδιόριστη σειρά. Κατά την έναρξη παρατηρείται ότι κατά κανόνα τα αριστερότερα παιδιά δημιουργούνται πριν τα δεξιότερα όμως δεν μπορεί να χαρακτηριστεί ούτε bfs αλλά ούτε και dfs. Αυτό συμβαίνει διότι υπάρχουν περιπτώσεις που δημιουργούνται παιδιά αριστερότερων κόμβων πριν από τα δεξιότερα αδέλφια τους. Κατά τον τερματισμό οι κόμβοι χωρίς παιδιά τερματίζονται πρώτοι αλλά κατά τα άλλα δεν παρατηρείται κάποια άλλη τακτική συμπεριφορά.

1.3 Αποστολή και χειρισμός σημάτων

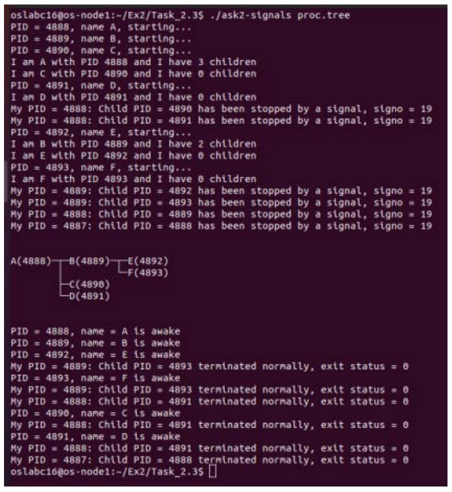
Πηγαίος κώδικας:



```c
#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 fork_procs(struct tree_node *root) {
        //Start
        printf("PID = %ld, name %s, starting...\n", (long)getpid(), root->name);
        change_pname(root->name);

        int i=0,status;
        pid_t childrenpid[root->nr_children],pid;
        while(i<root->nr_children){
                pid = fork();
                if(pid == 0){
                        fork_procs(root -> children+i);
                        exit(1);
                }
                else childrenpid[i] = pid;
                i++;
        }
        printf("I am %s with PID %ld and I have %d children\n", root->name, (long)getpid(), root->nr_children);
        wait_for_ready_children(root->nr_children);
        // Suspend Self
        raise(SIGSTOP);
        printf("PID = %ld, name = %s is awake\n", (long)getpid(), root->name);

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

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

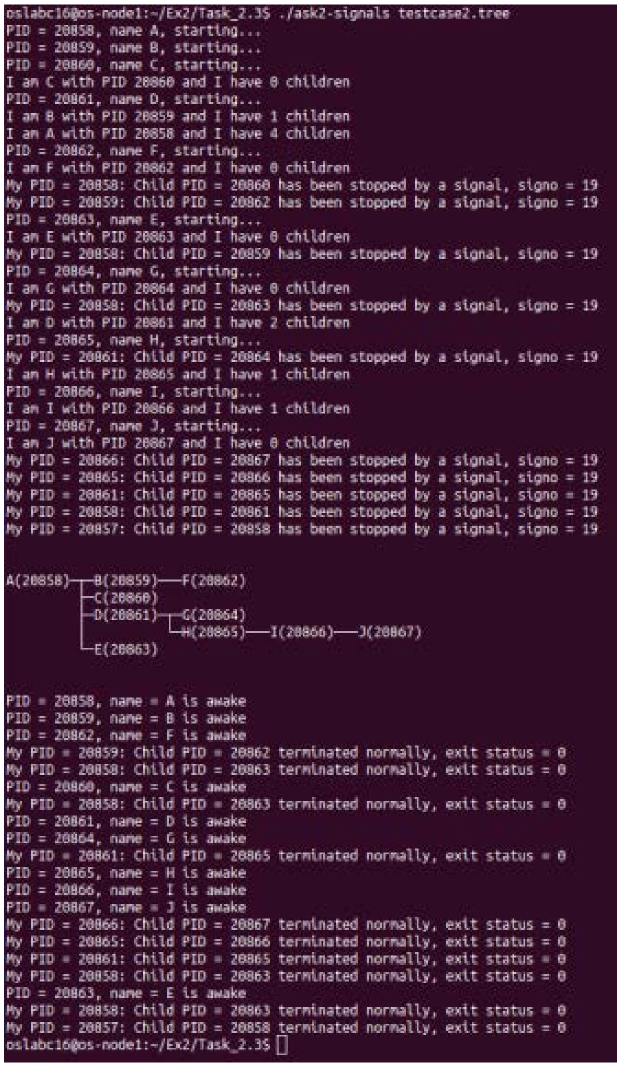
```c
int main(int argc, char *argv[])
{
        pid_t pid;
        int status;
        struct tree_node *root;

        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 */
                fork_procs(root);
                exit(1);
        }

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

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

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

        /* for ask2-signals */
        kill(pid, SIGCONT);

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

        return 0;

}
```

Έξοδος εκτέλεσης 1:



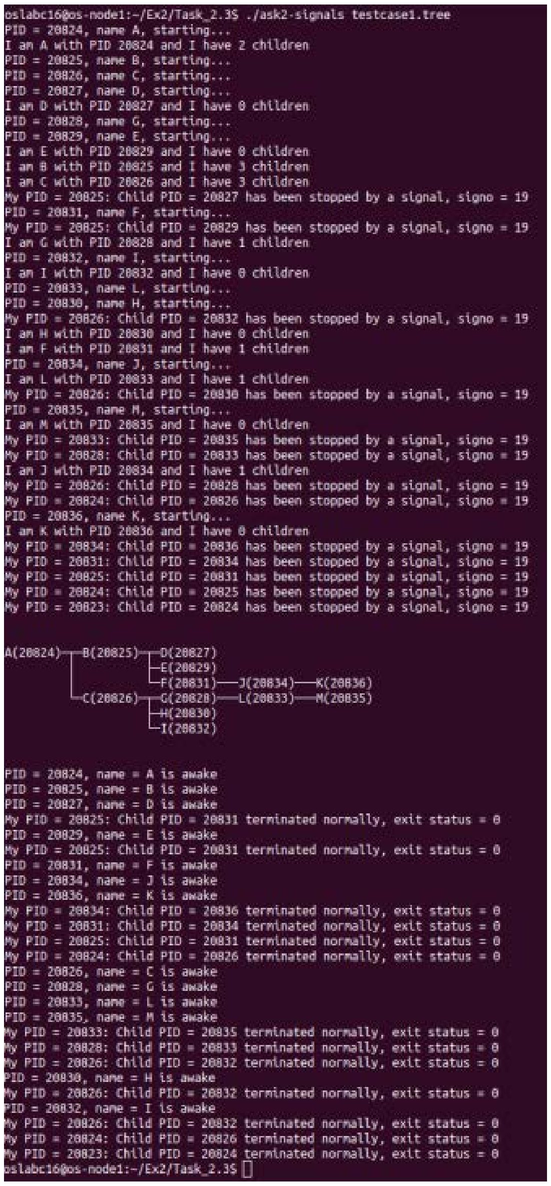
```
oslabc16@os-node1:~/Ex2/Task_2.3$ ./ask2-signals proc.tree
PID = 4888, name A, starting...
PID = 4889, name B, starting...
PID = 4890, name C, starting...
I am A with PID 4888 and I have 3 children
I am C with PID 4890 and I have 0 children
PID = 4891, name D, starting...
I am D with PID 4891 and I have 0 children
My PID = 4888: Child PID = 4890 has been stopped by a signal, signo = 19
My PID = 4888: Child PID = 4891 has been stopped by a signal, signo = 19
PID = 4892, name E, starting...
I am B with PID 4889 and I have 2 children
I am E with PID 4892 and I have 0 children
PID = 4893, name F, starting...
I am F with PID 4893 and I have 0 children
My PID = 4889: Child PID = 4892 has been stopped by a signal, signo = 19
My PID = 4889: Child PID = 4893 has been stopped by a signal, signo = 19
My PID = 4888: Child PID = 4889 has been stopped by a signal, signo = 19
My PID = 4887: Child PID = 4888 has been stopped by a signal, signo = 19


A(4888)─┬─B(4889)─┬─E(4892)
        │         └─F(4893)
        ├─C(4890)
        └─D(4891)


PID = 4888, name = A is awake
PID = 4889, name = B is awake
PID = 4892, name = E is awake
My PID = 4889: Child PID = 4893 terminated normally, exit status = 0
PID = 4893, name = F is awake
My PID = 4889: Child PID = 4893 terminated normally, exit status = 0
My PID = 4888: Child PID = 4891 terminated normally, exit status = 0
PID = 4890, name = C is awake
My PID = 4888: Child PID = 4891 terminated normally, exit status = 0
PID = 4891, name = D is awake
My PID = 4888: Child PID = 4891 terminated normally, exit status = 0
My PID = 4887: Child PID = 4888 terminated normally, exit status = 0
oslabc16@os-node1:~/Ex2/Task_2.3$ 
```

Έξοδος εκτέλεσης 2:



```
oslabc16@os-node1:~/Ex2/Task_2.3$ ./ask2-signals testcase2.tree
PID = 20858, name A, starting...
PID = 20859, name B, starting...
PID = 20860, name C, starting...
I am C with PID 20860 and I have 0 children
PID = 20861, name D, starting...
I am B with PID 20859 and I have 1 children
I am A with PID 20858 and I have 4 children
PID = 20862, name F, starting...
I am F with PID 20862 and I have 0 children
My PID = 20858: Child PID = 20860 has been stopped by a signal, signo = 19
My PID = 20859: Child PID = 20862 has been stopped by a signal, signo = 19
PID = 20863, name E, starting...
I am E with PID 20863 and I have 0 children
My PID = 20858: Child PID = 20859 has been stopped by a signal, signo = 19
PID = 20864, name G, starting...
I am G with PID 20864 and I have 0 children
My PID = 20858: Child PID = 20863 has been stopped by a signal, signo = 19
I am D with PID 20861 and I have 2 children
PID = 20865, name H, starting...
My PID = 20861: Child PID = 20864 has been stopped by a signal, signo = 19
I am H with PID 20865 and I have 1 children
PID = 20866, name I, starting...
I am I with PID 20866 and I have 1 children
PID = 20867, name J, starting...
I am J with PID 20867 and I have 0 children
My PID = 20866: Child PID = 20867 has been stopped by a signal, signo = 19
My PID = 20865: Child PID = 20866 has been stopped by a signal, signo = 19
My PID = 20861: Child PID = 20865 has been stopped by a signal, signo = 19
My PID = 20858: Child PID = 20861 has been stopped by a signal, signo = 19
My PID = 20857: Child PID = 20858 has been stopped by a signal, signo = 19


A(20858)─┬─B(20859)───F(20862)
         ├─C(20860)
         ├─D(20861)─┬─G(20864)
         │          └─H(20865)───I(20866)───J(20867)
         └─E(20863)


PID = 20858, name = A is awake
PID = 20859, name = B is awake
PID = 20862, name = F is awake
My PID = 20859: Child PID = 20862 terminated normally, exit status = 0
My PID = 20858: Child PID = 20863 terminated normally, exit status = 0
PID = 20860, name = C is awake
My PID = 20858: Child PID = 20863 terminated normally, exit status = 0
PID = 20861, name = D is awake
PID = 20864, name = G is awake
My PID = 20861: Child PID = 20865 terminated normally, exit status = 0
PID = 20865, name = H is awake
PID = 20866, name = I is awake
PID = 20867, name = J is awake
My PID = 20866: Child PID = 20867 terminated normally, exit status = 0
My PID = 20865: Child PID = 20866 terminated normally, exit status = 0
My PID = 20861: Child PID = 20865 terminated normally, exit status = 0
My PID = 20858: Child PID = 20863 terminated normally, exit status = 0
PID = 20863, name = E is awake
My PID = 20858: Child PID = 20863 terminated normally, exit status = 0
My PID = 20857: Child PID = 20858 terminated normally, exit status = 0
oslabc16@os-node1:~/Ex2/Task_2.3$ 
```

Έξοδος εκτέλεσης 3:



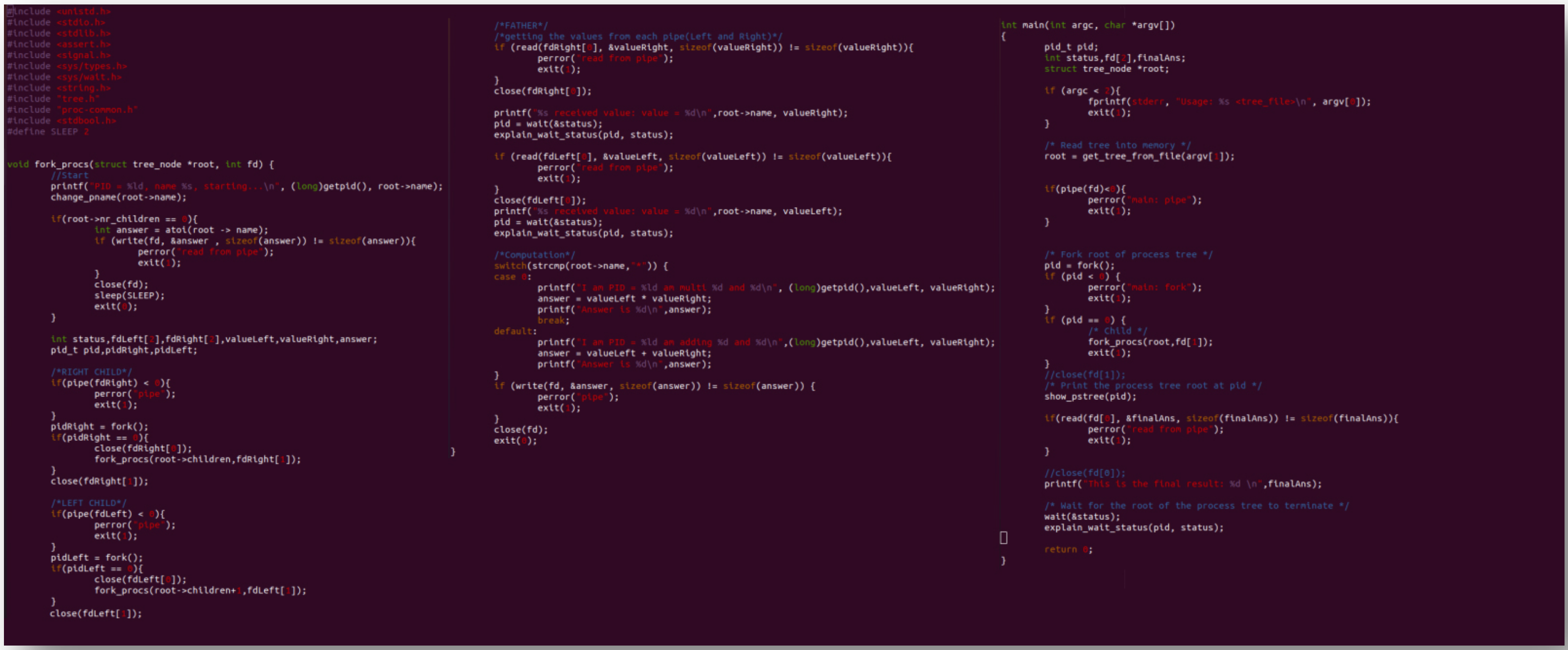
```
oslabc16@os-node1:~/Ex2/Task_2.3$ ./ask2-signals testcase1.tree
PID = 20824, name A, starting...
I am A with PID 20824 and I have 2 children
PID = 20825, name B, starting...
PID = 20826, name C, starting...
PID = 20827, name D, starting...
I am D with PID 20827 and I have 0 children
PID = 20828, name G, starting...
PID = 20829, name E, starting...
I am E with PID 20829 and I have 0 children
I am B with PID 20825 and I have 3 children
I am C with PID 20826 and I have 3 children
My PID = 20825: Child PID = 20827 has been stopped by a signal, signo = 19
PID = 20831, name F, starting...
My PID = 20825: Child PID = 20829 has been stopped by a signal, signo = 19
I am G with PID 20828 and I have 1 children
PID = 20832, name I, starting...
I am I with PID 20832 and I have 0 children
PID = 20833, name L, starting...
PID = 20830, name H, starting...
My PID = 20826: Child PID = 20832 has been stopped by a signal, signo = 19
I am H with PID 20830 and I have 0 children
I am F with PID 20831 and I have 1 children
PID = 20834, name J, starting...
I am L with PID 20833 and I have 1 children
My PID = 20826: Child PID = 20830 has been stopped by a signal, signo = 19
PID = 20835, name M, starting...
I am M with PID 20835 and I have 0 children
My PID = 20833: Child PID = 20835 has been stopped by a signal, signo = 19
My PID = 20828: Child PID = 20833 has been stopped by a signal, signo = 19
I am J with PID 20834 and I have 1 children
My PID = 20826: Child PID = 20828 has been stopped by a signal, signo = 19
My PID = 20824: Child PID = 20826 has been stopped by a signal, signo = 19
PID = 20836, name K, starting...
I am K with PID 20836 and I have 0 children
My PID = 20834: Child PID = 20836 has been stopped by a signal, signo = 19
My PID = 20831: Child PID = 20834 has been stopped by a signal, signo = 19
My PID = 20825: Child PID = 20831 has been stopped by a signal, signo = 19
My PID = 20824: Child PID = 20825 has been stopped by a signal, signo = 19
My PID = 20823: Child PID = 20824 has been stopped by a signal, signo = 19


A(20824)─┬─B(20825)─┬─D(20827)
         │          ├─E(20829)
         │          └─F(20831)───J(20834)───K(20836)
         └─C(20826)─┬─G(20828)───L(20833)───M(20835)
                    ├─H(20830)
                    └─I(20832)


PID = 20824, name = A is awake
PID = 20825, name = B is awake
PID = 20827, name = D is awake
My PID = 20825: Child PID = 20831 terminated normally, exit status = 0
PID = 20829, name = E is awake
My PID = 20825: Child PID = 20831 terminated normally, exit status = 0
PID = 20831, name = F is awake
PID = 20834, name = J is awake
PID = 20836, name = K is awake
My PID = 20834: Child PID = 20836 terminated normally, exit status = 0
My PID = 20831: Child PID = 20834 terminated normally, exit status = 0
My PID = 20825: Child PID = 20831 terminated normally, exit status = 0
My PID = 20824: Child PID = 20826 terminated normally, exit status = 0
PID = 20826, name = C is awake
PID = 20828, name = G is awake
PID = 20833, name = L is awake
PID = 20835, name = M is awake
My PID = 20833: Child PID = 20835 terminated normally, exit status = 0
My PID = 20828: Child PID = 20833 terminated normally, exit status = 0
My PID = 20826: Child PID = 20832 terminated normally, exit status = 0
PID = 20830, name = H is awake
My PID = 20826: Child PID = 20832 terminated normally, exit status = 0
PID = 20832, name = I is awake
My PID = 20826: Child PID = 20832 terminated normally, exit status = 0
My PID = 20824: Child PID = 20826 terminated normally, exit status = 0
My PID = 20823: Child PID = 20824 terminated normally, exit status = 0
oslabc16@os-node1:~/Ex2/Task_2.3$ 
```

Ερωτήσεις

1. Με τη χρήση σημάτων μπορεί να ρυθμιστεί με ακρίβεια η στιγμή πάυσης και επανεκκίνησης των διεργασιών και έτσι επιτυγχάνεται καλύτερα ο συγχρονισμός τους. Με τη χρήση της sleep() δεν υπάρχει τέτοια δυνατότητα και η σειρά αφίεται, σε κάποια σημεία της, στην τύχη. Θεωρητικά θα ήταν εφικτή η επίτευξη σωστής σειράς και με τη χρήση της sleep() όμως κάτι τέτοιο θα προϋπέθετε προσεκτική επιλογή του χρόνου καθυστέρησης της κάθε φορά.
2. Η wait_for_ready_children(); εξασφαλίζει ότι όλα τα παιδιά ενός πατέρα έχουν αναστείλει τη λειτουργία τους και έχουν στείλει το σήμα SIGSTOP. Τυχόν παράλειψη της μπορεί να οδηγούσε σε παύση του πατέρα πριν από παύση όλων των παιδιών του ή ακόμα και να μην σταματήσει καθόλου κάποιο παιδί.

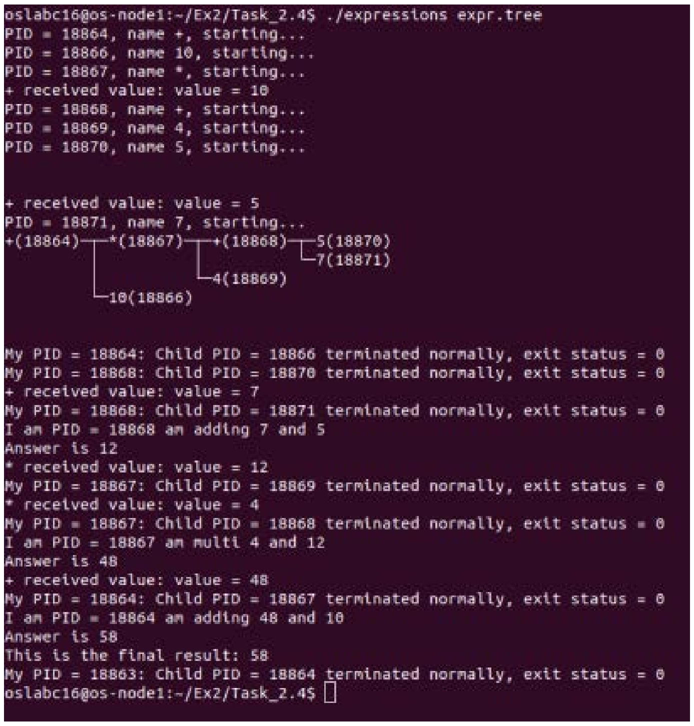
1.4 Παράλληλος υπολογισμός αριθμητικής έκφρασης

Πηγαίος κώδικας:



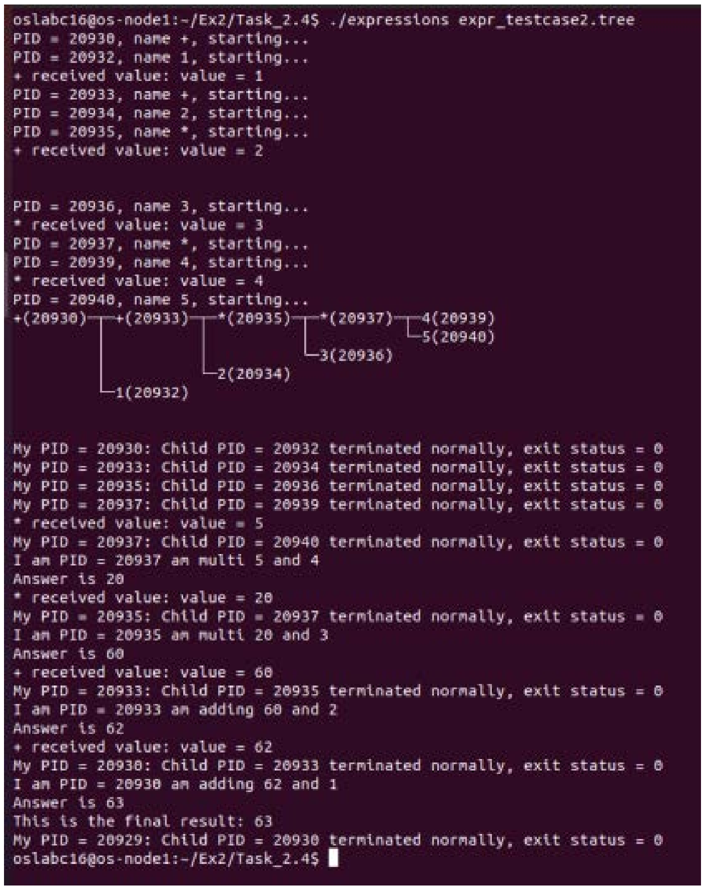
```
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <signal.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <string.h>
#include "tree.h"
#include "proc-common.h"
#include <stdbool.h>
#define SLEEP 2


void fork_procs(struct tree_node *root, int fd) {
        //Start
        printf("PID = %ld, name %s, starting...\n", (long)getpid(), root->name);
        change_pname(root->name);

        if(root->nr_children == 0){
                int answer = atoi(root -> name);
                if (write(fd, &answer , sizeof(answer)) != sizeof(answer)){
                        perror("read from pipe");
                        exit(1);
                }
                close(fd);
                sleep(SLEEP);
                exit(0);
        }

        int status,fdLeft[2],fdRight[2],valueLeft,valueRight,answer;
        pid_t pid,pidRight,pidLeft;

        /*RIGHT CHILD*/
        if(pipe(fdRight) < 0){
                perror("pipe");
                exit(1);
        }
        pidRight = fork();
        if(pidRight == 0){
                close(fdRight[0]);
                fork_procs(root->children,fdRight[1]);
        }
        close(fdRight[1]);

        /*LEFT CHILD*/
        if(pipe(fdLeft) < 0){
                perror("pipe");
                exit(1);
        }
        pidLeft = fork();
        if(pidLeft == 0){
                close(fdLeft[0]);
                fork_procs(root->children+1,fdLeft[1]);
        }
        close(fdLeft[1]);

        /*FATHER*/
        /*getting the values from each pipe(Left and Right)*/
        if (read(fdRight[0], &valueRight, sizeof(valueRight)) != sizeof(valueRight)){
                perror("read from pipe");
                exit(1);
        }
        close(fdRight[0]);

        printf("%s received value: value = %d\n",root->name, valueRight);
        pid = wait(&status);
        explain_wait_status(pid, status);

        if (read(fdLeft[0], &valueLeft, sizeof(valueLeft)) != sizeof(valueLeft)){
                perror("read from pipe");
                exit(1);
        }
        close(fdLeft[0]);
        printf("%s received value: value = %d\n",root->name, valueLeft);
        pid = wait(&status);
        explain_wait_status(pid, status);

        /*Computation*/
        switch(strcmp(root->name,"*")) {
        case 0:
                printf("I am PID = %ld am multi %d and %d\n", (long)getpid(),valueLeft, valueRight);
                answer = valueLeft * valueRight;
                printf("Answer is %d\n",answer);
                break;
        default:
                printf("I am PID = %ld am adding %d and %d\n",(long)getpid(),valueLeft, valueRight);
                answer = valueLeft + valueRight;
                printf("Answer is %d\n",answer);
        }
        if (write(fd, &answer, sizeof(answer)) != sizeof(answer)) {
                perror("pipe");
                exit(1);
        }
        close(fd);
        exit(0);
}

int main(int argc, char *argv[])
{
        pid_t pid;
        int status,fd[2],finalAns;
        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]);


        if(pipe(fd)<0){
                perror("main: pipe");
                exit(1);
        }


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

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

        //close(fd[0]);
        printf("This is the final result: %d \n",finalAns);

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

        return 0;
}
```

Έξοδος εκτέλεσης 1:



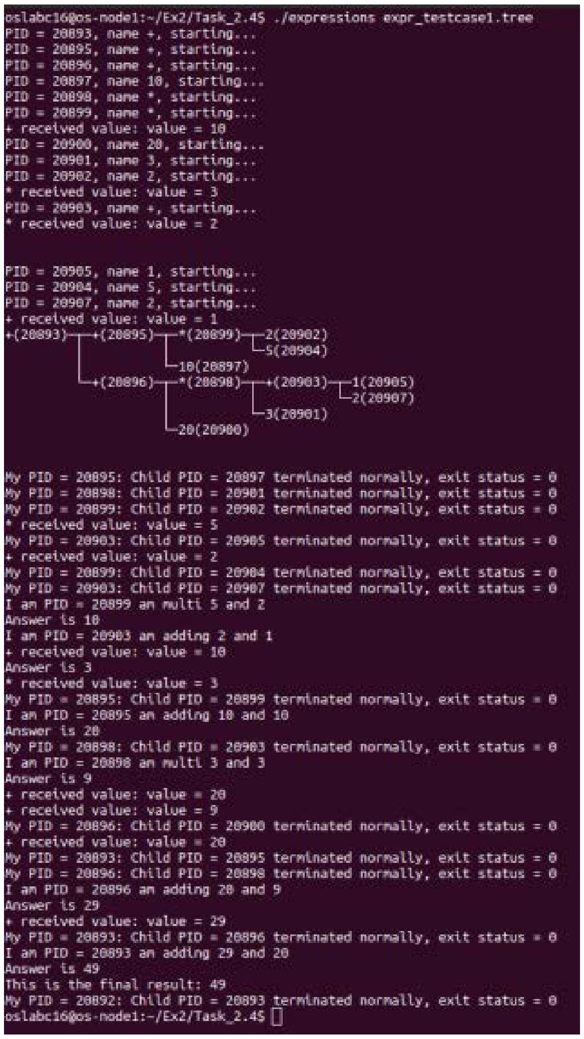
```
oslabc16@os-node1:~/Ex2/Task_2.4$ ./expressions expr.tree
PID = 18864, name +, starting...
PID = 18866, name 10, starting...
PID = 18867, name *, starting...
+ received value: value = 10
PID = 18868, name +, starting...
PID = 18869, name 4, starting...
PID = 18870, name 5, starting...


+ received value: value = 5
PID = 18871, name 7, starting...
+(18864)─┬─*(18867)─┬─+(18868)─┬─5(18870)
         │          │          └─7(18871)
         │          └─4(18869)
         └─10(18866)


My PID = 18864: Child PID = 18866 terminated normally, exit status = 0
My PID = 18868: Child PID = 18870 terminated normally, exit status = 0
+ received value: value = 7
My PID = 18868: Child PID = 18871 terminated normally, exit status = 0
I am PID = 18868 am adding 7 and 5
Answer is 12
* received value: value = 12
My PID = 18867: Child PID = 18869 terminated normally, exit status = 0
* received value: value = 4
My PID = 18867: Child PID = 18868 terminated normally, exit status = 0
I am PID = 18867 am multi 4 and 12
Answer is 48
+ received value: value = 48
My PID = 18864: Child PID = 18867 terminated normally, exit status = 0
I am PID = 18864 am adding 48 and 10
Answer is 58
This is the final result: 58
My PID = 18863: Child PID = 18864 terminated normally, exit status = 0
oslabc16@os-node1:~/Ex2/Task_2.4$ 
```

Έξοδος εκτέλεσης 2:



```
oslabc16@os-node1:~/Ex2/Task_2.4$ ./expressions expr_testcase2.tree
PID = 20930, name +, starting...
PID = 20932, name 1, starting...
+ received value: value = 1
PID = 20933, name +, starting...
PID = 20934, name 2, starting...
PID = 20935, name *, starting...
+ received value: value = 2


PID = 20936, name 3, starting...
* received value: value = 3
PID = 20937, name *, starting...
PID = 20939, name 4, starting...
* received value: value = 4
PID = 20940, name 5, starting...
+(20930)─┬─+(20933)─┬─*(20935)─┬─*(20937)─┬─4(20939)
         │          │          │          └─5(20940)
         │          │          └─3(20936)
         │          └─2(20934)
         └─1(20932)


My PID = 20930: Child PID = 20932 terminated normally, exit status = 0
My PID = 20933: Child PID = 20934 terminated normally, exit status = 0
My PID = 20935: Child PID = 20936 terminated normally, exit status = 0
My PID = 20937: Child PID = 20939 terminated normally, exit status = 0
* received value: value = 5
My PID = 20937: Child PID = 20940 terminated normally, exit status = 0
I am PID = 20937 am multi 5 and 4
Answer is 20
* received value: value = 20
My PID = 20935: Child PID = 20937 terminated normally, exit status = 0
I am PID = 20935 am multi 20 and 3
Answer is 60
+ received value: value = 60
My PID = 20933: Child PID = 20935 terminated normally, exit status = 0
I am PID = 20933 am adding 60 and 2
Answer is 62
+ received value: value = 62
My PID = 20930: Child PID = 20933 terminated normally, exit status = 0
I am PID = 20930 am adding 62 and 1
Answer is 63
This is the final result: 63
My PID = 20929: Child PID = 20930 terminated normally, exit status = 0
oslabc16@os-node1:~/Ex2/Task_2.4$ █
```

Έξοδος εκτέλεσης 3:



```
oslabc16@os-node1:~/Ex2/Task_2.4$ ./expressions expr_testcase1.tree
PID = 20893, name +, starting...
PID = 20895, name +, starting...
PID = 20896, name +, starting...
PID = 20897, name 10, starting...
PID = 20898, name *, starting...
PID = 20899, name *, starting...
+ received value: value = 10
PID = 20900, name 20, starting...
PID = 20901, name 3, starting...
PID = 20902, name 2, starting...
* received value: value = 3
PID = 20903, name +, starting...
* received value: value = 2


PID = 20905, name 1, starting...
PID = 20904, name 5, starting...
PID = 20907, name 2, starting...
+ received value: value = 1
+(20893)─┬─+(20895)─┬─*(20899)─┬─2(20902)
         │          │          └─5(20904)
         │          └─10(20897)
         └─+(20896)─┬─*(20898)─┬─+(20903)─┬─1(20905)
                    │          │          └─2(20907)
                    │          └─3(20901)
                    └─20(20900)


My PID = 20895: Child PID = 20897 terminated normally, exit status = 0
My PID = 20898: Child PID = 20901 terminated normally, exit status = 0
My PID = 20899: Child PID = 20902 terminated normally, exit status = 0
* received value: value = 5
My PID = 20903: Child PID = 20905 terminated normally, exit status = 0
+ received value: value = 2
My PID = 20899: Child PID = 20904 terminated normally, exit status = 0
My PID = 20903: Child PID = 20907 terminated normally, exit status = 0
I am PID = 20899 am multi 5 and 2
Answer is 10
I am PID = 20903 am adding 2 and 1
+ received value: value = 10
Answer is 3
* received value: value = 3
My PID = 20895: Child PID = 20899 terminated normally, exit status = 0
I am PID = 20895 am adding 10 and 10
Answer is 20
My PID = 20898: Child PID = 20903 terminated normally, exit status = 0
I am PID = 20898 am multi 3 and 3
Answer is 9
+ received value: value = 20
+ received value: value = 9
My PID = 20896: Child PID = 20900 terminated normally, exit status = 0
+ received value: value = 20
My PID = 20893: Child PID = 20895 terminated normally, exit status = 0
My PID = 20896: Child PID = 20898 terminated normally, exit status = 0
I am PID = 20896 am adding 20 and 9
Answer is 29
+ received value: value = 29
My PID = 20893: Child PID = 20896 terminated normally, exit status = 0
I am PID = 20893 am adding 29 and 20
Answer is 49
This is the final result: 49
My PID = 20892: Child PID = 20893 terminated normally, exit status = 0
oslabc16@os-node1:~/Ex2/Task_2.4$ ▯
```

Ερωτήσεις

1. Σε αυτή την άσκηση κάθε διεργασία φύλλο (αριθμός) έχει μία σωλήνωση με την οποία επικοινωνεί με τον πατέρα της. Κάθε άλλη διεργασία έχει 3 σωληνώσεις, μία για επικοινωνία με τον πατέρα της και δύο για επικοινωνία με το κάθε παιδί της. Στη συγκεκριμένη περίπτωση θα μπορούσε να χρησιμοποιηθεί κοινή σωλήνωση διότι ο πολ/σμος και η πρόσθεση έχουν την αντιμεταθετική ιδιότητα όμως δεν θα μπορούσε να γενικευτεί για κάθε τελεστή.
2. Η αποτίμηση της έκφρασης με δέντρο διεργασιών σε ένα σύστημα πολλαπλών επεξεργαστών έχει το πλεονέκτημα ότι υπάρχουν διεργασίες που μπορούν να εκτελεστούν ταυτόχρονα. Αυτό έχεις ως αποτέλεσμα την διεκπαιρέωση της συνολικής διαδικασίας σε λιγότερο χρόνο. Παρόλα αυτά είναι σημαντικό κατά το διαμοιρασμό των διεργασιών σε επεξεγαστές να εξασφαλίζεται ότι δεν θα εκτελεστούν διεργασίες-πατέρες πριν από τις διεργασίες-παιδιά τους.