ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ

ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΝΑΦΟΡΑ 2Η

Από τους:

Ιωάννης Μιχαήλ Καζελίδης
Α.Μ.: 03117885

Μάριος Κερασιώτης
Α.Μ.: 03117890

1 ΑΣΚΗΣΕΙΣ

1.1 Δημιουργία δεδομένου δέντρου διεργασιών

Για αυτό το ερώτημα χρησιμοποιούμε τον παρακάτω πηγαίο κώδικα. Ταυτόχρονα χρησιμοποιούμε τα αρχεία *proc-common.{c, h}* που μας είναι δοσμένα.

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

#include "proc-common.h"

#define SLEEP_PROC_SEC 10
#define SLEEP_TREE_SEC 3

#define EXIT_A 16
#define EXIT_B 19
#define EXIT_C 17
#define EXIT_D 13

/*
 * Create this process tree:
 * A-+-B---D
 *   `-C
 */
void fork_procs(void) {
  /*
   * initial process is A.
   */
  pid_t pid_b, pid_c, pid_d;
  int status;

  change_pname("A");
  printf("Proccess A has PID = %ld\n", (long)getpid());

  // Code bellow is for child B
  fprintf(stderr, "Parent A, PID = %ld: Creating child B\n", (long)getpid());
  pid_b = fork();
  if (pid_b < 0) {
    /* fork failed */
    perror("B: fork");
    exit(1);
  }

  if (pid_b == 0) {
    change_pname("B");
    printf("Proccess B has PID = %ld\n", (long)getpid());
    fprintf(stderr, "Parent B, PID = %ld: Creating child D\n", (long)getpid());
    pid_d = fork();
    if (pid_d < 0) {
      /* fork failed */
```

```
    perror("D: fork");
    exit(1);
  }

  if (pid_d == 0) {
    change_pname("D");
    printf("Proccess D has PID = %ld\n", (long)getpid());
    printf("D: Sleeping...\n");
    sleep(SLEEP_PROC_SEC);
    printf("D: Done Sleeping...\n");
    printf("D with PID = %ld is ready to terminate...\nD: Exiting...\n",
           (long)getpid());
    exit(EXIT_D);
  }

  printf(
      "Parent B, PID = %ld: Created child D with PID = %ld, waiting for it "
      "to terminate...\n",
      (long)getpid(), (long)pid_d);
  pid_d = wait(&status);
  explain_wait_status(pid_d, status);
  printf("B with PID = %ld is ready to terminate...\nB: Exiting...\n",
         (long)getpid());
  exit(EXIT_B);
}

// Code bellow is for child C
fprintf(stderr, "Parent A, PID = %ld: Creating child C\n", (long)getpid());
pid_c = fork();
if (pid_c < 0) {
  /* fork failed */
  perror("C: fork");
  exit(1);
}

if (pid_c == 0) {
  change_pname("C");
  printf("Proccess C has PID = %ld\n", (long)getpid());
  printf("C: Sleeping...\n");
  sleep(SLEEP_PROC_SEC);
  printf("C: Done Sleeping...\n");
  printf("C with PID = %ld is ready to terminate...\nC: Exiting...\n",
         (long)getpid());
  exit(EXIT_C);
}

printf(
    "Parent A, PID = %ld: Created child C with PID = %ld, waiting for it to "
    "terminate...\n",
    (long)getpid(), (long)pid_c);
pid_c = wait(&status);
explain_wait_status(pid_c, status);
```

```c
  printf(
      "Parent A, PID = %ld: Created child B with PID = %ld, waiting for it to "
      "terminate...\n",
      (long)getpid(), (long)pid_b);
  pid_b = wait(&status);
  explain_wait_status(pid_b, status);
  printf("A with PID = %ld is ready to terminate...\nA: Exiting...\n",
         (long)getpid());
  exit(EXIT_A);
}

/*
 * 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(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-{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;
}
```

Και τρέχοντας το πρόγραμμα λαμβάνουμε έξοδο:

```
giannis@pop-os:~/Desktop/OS/NTUA-Operating-Systems/Lab2/lab2/forktree$ ./ask2-fork
Proccess A has PID = 6432
Parent A, PID = 6432: Creating child B
Parent A, PID = 6432: Creating child C
Parent A, PID = 6432: Created child C with PID = 6434, waiting for it to terminate...
Proccess B has PID = 6433
Parent B, PID = 6433: Creating child D
Parent B, PID = 6433: Created child D with PID = 6435, waiting for it to terminate...
Proccess D has PID = 6435
D: Sleeping...
Proccess C has PID = 6434
C: Sleeping...


A(6432)─┬─B(6433)──D(6435)
        └─C(6434)


D: Done Sleeping...
D with PID = 6435 is ready to terminate...
D: Exiting...
C: Done Sleeping...
C with PID = 6434 is ready to terminate...
C: Exiting...
My PID = 6432: Child PID = 6434 terminated normally, exit status = 17
Parent A, PID = 6432: Created child B with PID = 6433, waiting for it to terminate...
My PID = 6433: Child PID = 6435 terminated normally, exit status = 13
B with PID = 6433 is ready to terminate...
B: Exiting...
My PID = 6432: Child PID = 6433 terminated normally, exit status = 19
A with PID = 6432 is ready to terminate...
A: Exiting...
My PID = 6431: Child PID = 6432 terminated normally, exit status = 16
```

1.1.1 Ερώτηση 1η

Η διεργασία A είναι πατέρας των B, C. Εάν την τερματίσουμε πρόωρα δηλαδή την τερματίσουμε πριν οι διεργασίες B, C ολοκληρωθούν τότε οι διεργασίες αυτές θα γίνουν “zombie” διεργασίες και θα γίνουν παιδιά της init (PID = 1) που εκτελεί συνεχώς wait().

1.1.2 Ερώτηση 2η

Το αποτέλεσμα της *show_pstree(getpid())* φαίνεται παρακάτω:

```
giannis@pop-os:~/Desktop/OS/NTUA-Operating-Systems/Lab2/lab2/forktree$ ./ask2-fork
Proccess A has PID = 8151
Parent A, PID = 8151: Creating child B
Parent A, PID = 8151: Creating child C
Parent A, PID = 8151: Created child C with PID = 8153, waiting for it to terminate...
Proccess B has PID = 8152
Parent B, PID = 8152: Creating child D
Parent B, PID = 8152: Created child D with PID = 8154, waiting for it to terminate...
Proccess D has PID = 8154
D: Sleeping...
Proccess C has PID = 8153
C: Sleeping...


ask2-fork(8150)─┬─A(8151)─┬─B(8152)──D(8154)
                │         └─C(8153)
                └─sh(8158)──pstree(8159)


D: Done Sleeping...
D with PID = 8154 is ready to terminate...
D: Exiting...
C: Done Sleeping...
C with PID = 8153 is ready to terminate...
C: Exiting...
My PID = 8151: Child PID = 8153 terminated normally, exit status = 17
Parent A, PID = 8151: Created child B with PID = 8152, waiting for it to terminate...
My PID = 8152: Child PID = 8154 terminated normally, exit status = 13
B with PID = 8152 is ready to terminate...
B: Exiting...
My PID = 8151: Child PID = 8152 terminated normally, exit status = 19
A with PID = 8151 is ready to terminate...
A: Exiting...
My PID = 8150: Child PID = 8151 terminated normally, exit status = 16
```

Από τα man pages μπορούμε να καταλάβουμε ότι η getpid() επιστρέφει το PID της καλούμενης διεργασίας που στη περίπτωση μας η διεργασία αυτή είναι η ask2-fork (PID=8150). Ως αποτέλεσμα η show_pstree() τυπώνει ένα δέντρο που έχει ρίζα όχι τη διεργασία A αλλά τη διεργασία μέσα στην οποία βρισκόμαστε τώρα, την ask2-fork. Αξιοσημείωτο είναι ότι μέσα στο δέντρο προστέθηκε η διεργασία sh (PID=8158) και η pstree (PID=8159), που είναι παιδί της. Αυτό γίνεται διότι η show_pstree για να λειτουργήσει καλεί μέσω της διεργασίας του bash (sh (PID = 8158)) την pstree (PID = 8159). Για αυτόν τον λόγο επομένως η ask2-fork καλεί την διεργασία του bash, η οποία με την σειρά της καλεί την pstree, δημιουργώντας έτσι το παραπάνω σχήμα.

1.1.3 Ερώτηση 3η

Ο εκάστοτε διαχειριστής ενός συστήματος πρέπει να εξασφαλίσει ότι κανένας χρήστης δεν θα μπορεί να δημιουργήσει αυθαίρετο αριθμό από διεργασίες, αφήνοντας έτσι ανοικτή την περίπτωση εκούσια (π.χ. κάποιος ιός) ή ακούσια (π.χ. ατελείωτη αναδρομή) να κάνει κατάχρηση των διαθέσιμων πόρων του συστήματος με αποτέλεσμα το σύστημα να γίνει ασταθές.

Ένα χαρακτηριστικό παράδειγμα εκούσιας κατάχρησης πόρων αποτελεί η επίθεση forkbomb η οποία δημιουργεί συνεχώς αντίγραφα του εαυτού της με σκοπό να κάνει ένα σύστημα να καταρρεύσει (crash-άρει) (DoS : Denial of Service attack).

1.2 Δημιουργία αυθαίρετου δέντρου διεργασιών

Για αυτό το ερώτημα χρησιμοποιούμε τον παρακάτω πηγαίο κώδικα. Ταυτόχρονα χρησιμοποιούμε τα αρχεία *proc-common.{c, h}, tree.{c, h}* που μας είναι δοσμένα.

```c
#include <unistd.h>
#include <stdio.h>
```

```c
#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 fork_procs(struct tree_node *node)
{
    pid_t child;
    int status;
    int i;

    change_pname(node->name);
    printf("Proccess %s has PID = %ld and %d children\n", node-
>name, (long)getpid(), node->nr_children);

    for (i = 0; i < node->nr_children; ++i)
    {
        fprintf(stderr, "Parent %s, PID = %ld: Creating child %s\n", node-
>name, (long)getpid(), node->children[i].name);
        child = fork();
        if (child < 0)
        {
            /* fork failed */
            perror("Error at children");
            exit(1);
        }

        if (child == 0)
        {
            fork_procs(&node->children[i]);
        }
    }

    for (i = 0; i < node->nr_children; ++i)
    {
        child = wait(&status);
        explain_wait_status(child, status);
    }

    if (node->nr_children == 0)
    {
        printf("%s: Sleeping...\n", node->name);
        sleep(SLEEP_PROC_SEC);
        printf("%s: Done Sleeping...\n", node->name);
    }
    printf("%s with PID = %ld is ready to terminate...\n%s: Exiting...\n", node-
>name, (long)getpid(), node->name);
```

```c
    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[])
{
    pid_t pid;
    int status;
    struct tree_node *root;

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

    root = get_tree_from_file(argv[1]);
    printf("Constructing the following process tree:\n");
    print_tree(root);

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

```
}
```

Τρέχοντας το πρόγραμμα με είσοδο το αρχείο *proc.tree* που μας δίνεται λαμβάνουμε έξοδο:

```
giannis@pop-os:~/Desktop/OS/NTUA-Operating-Systems/Lab2/lab2/forktree$ ./ask2-tree proc.tree
Constructing the following process tree:
A
        B
                E
                F
        C
        D
Proccess A has PID = 11515 and 3 children
Parent A, PID = 11515: Creating child B
Parent A, PID = 11515: Creating child C
Parent A, PID = 11515: Creating child D
Proccess B has PID = 11516 and 2 children
Parent B, PID = 11516: Creating child E
Parent B, PID = 11516: Creating child F
Proccess E has PID = 11519 and 0 children
E: Sleeping...
Proccess D has PID = 11518 and 0 children
D: Sleeping...
Proccess F has PID = 11520 and 0 children
F: Sleeping...
Proccess C has PID = 11517 and 0 children
C: Sleeping...


A(11515)─┬─B(11516)─┬─E(11519)
         │          └─F(11520)
         ├─C(11517)
         └─D(11518)


E: Done Sleeping...
E with PID = 11519 is ready to terminate...
E: Exiting...
D: Done Sleeping...
D with PID = 11518 is ready to terminate...
D: Exiting...
My PID = 11516: Child PID = 11519 terminated normally, exit status = 0
My PID = 11515: Child PID = 11518 terminated normally, exit status = 0
F: Done Sleeping...
F with PID = 11520 is ready to terminate...
F: Exiting...
My PID = 11516: Child PID = 11520 terminated normally, exit status = 0
B with PID = 11516 is ready to terminate...
B: Exiting...
My PID = 11515: Child PID = 11516 terminated normally, exit status = 0
C: Done Sleeping...
C with PID = 11517 is ready to terminate...
C: Exiting...
My PID = 11515: Child PID = 11517 terminated normally, exit status = 0
A with PID = 11515 is ready to terminate...

A: Exiting...
My PID = 11514: Child PID = 11515 terminated normally, exit status = 0
```

Τρέχοντας το πρόγραμμα με είσοδο το αρχείο *proc2.tree* που δημιουργήσαμε λαμβάνουμε έξοδο:

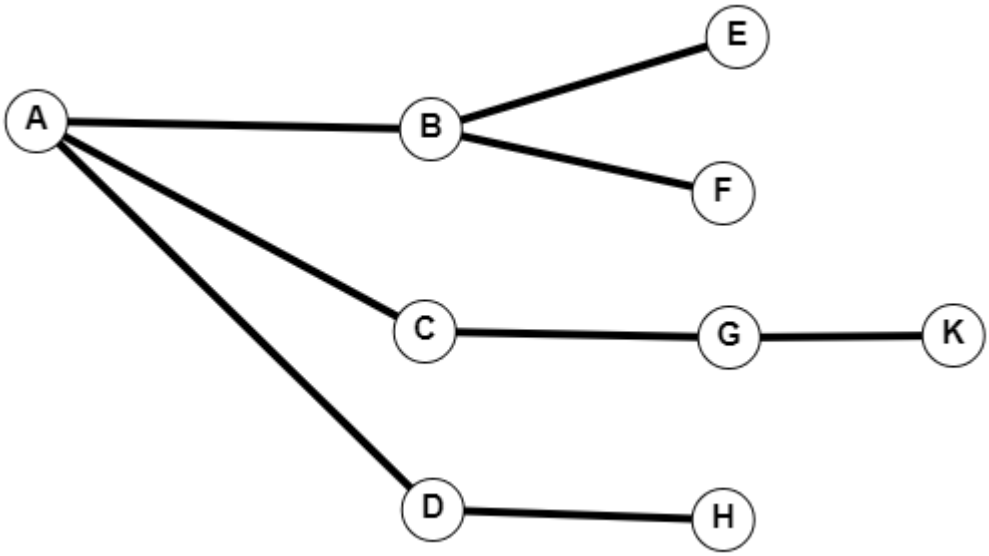
```
giannis@pop-os:~/Desktop/OS/NTUA-Operating-Systems/Lab2/lab2/forktree$ ./ask2-tree proc2.tree
Constructing the following process tree:
A
        B
                E
                F
        C
                G
                        K
        D
                H
Proccess A has PID = 12902 and 3 children
Parent A, PID = 12902: Creating child B
Parent A, PID = 12902: Creating child C
Proccess B has PID = 12903 and 2 children
Parent A, PID = 12902: Creating child D
Parent B, PID = 12903: Creating child E
Parent B, PID = 12903: Creating child F
Proccess C has PID = 12904 and 1 children
Parent C, PID = 12904: Creating child G
Proccess F has PID = 12907 and 0 children
F: Sleeping...
Proccess G has PID = 12908 and 1 children
Parent G, PID = 12908: Creating child K
Proccess K has PID = 12909 and 0 children
K: Sleeping...
Proccess D has PID = 12905 and 1 children
Parent D, PID = 12905: Creating child H
Proccess H has PID = 12910 and 0 children
H: Sleeping...
Proccess E has PID = 12906 and 0 children
E: Sleeping...


A(12902)─┬─B(12903)─┬─E(12906)
         │          └─F(12907)
         ├─C(12904)──G(12908)──K(12909)
         └─D(12905)──H(12910)


F: Done Sleeping...
F with PID = 12907 is ready to terminate...
F: Exiting...
My PID = 12903: Child PID = 12907 terminated normally, exit status = 0
K: Done Sleeping...
K with PID = 12909 is ready to terminate...
K: Exiting...
My PID = 12908: Child PID = 12909 terminated normally, exit status = 0
G with PID = 12908 is ready to terminate...
G: Exiting...
My PID = 12904: Child PID = 12908 terminated normally, exit status = 0
```

```
C with PID = 12904 is ready to terminate...
C: Exiting...
My PID = 12902: Child PID = 12904 terminated normally, exit status = 0
H: Done Sleeping...
H with PID = 12910 is ready to terminate...
H: Exiting...
My PID = 12905: Child PID = 12910 terminated normally, exit status = 0
D with PID = 12905 is ready to terminate...
D: Exiting...
My PID = 12902: Child PID = 12905 terminated normally, exit status = 0
E: Done Sleeping...
E with PID = 12906 is ready to terminate...
E: Exiting...
My PID = 12903: Child PID = 12906 terminated normally, exit status = 0
B with PID = 12903 is ready to terminate...
B: Exiting...
My PID = 12902: Child PID = 12903 terminated normally, exit status = 0
A with PID = 12902 is ready to terminate...
A: Exiting...
My PID = 12901: Child PID = 12902 terminated normally, exit status = 0
```

Όπου *proc2.tree*:



1.2.1 Ερώτηση 1η

Το πρόγραμμα αρχικά δημιουργεί τους κόμβους αρχίζοντας από τους γονείς και εκ των υστέρων δημιουργεί και τα παιδιά-φύλλα, επομένως η δημιουργία γίνεται με top-down λογική (η δημιουργία των διεργασιών ακολουθεί δηλαδή τον αλγόριθμο BFS καθώς διατρέχουμε το δέντρο διεργασιών εισόδου κατά επίπεδα). Αντίθετα, τερματισμός γίνεται με bottom-up λογική, καθώς θέλουμε να τερματιστούν πρώτα τα φύλλα και μετά οι εσωτερικοί κόμβοι, διότι θέλουμε να πεθάνουν ομαλά τα παιδιά και να μην μείνουν ορφανά.

1.3 Αποστολή και χειρισμός σημάτων

Για αυτό το ερώτημα χρησιμοποιούμε τον παρακάτω πηγαίο κώδικα. Ταυτόχρονα χρησιμοποιούμε τα αρχεία *proc-common.{c, h}*, *tree.{c, h}* που μας είναι δοσμένα.

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

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

void fork_procs(struct tree_node *root)
{
    /*
     * Start
     */
    int i;
    int status[root->nr_children];
    pid_t child[root->nr_children];

    printf("PID = %ld, name %s, starting...\n",
           (long)getpid(), root->name);
    change_pname(root->name);

    if (root->nr_children == 0)
    {                   // is leaf
        raise(SIGSTOP); // stop what you are doing until parent requests it
        printf("PID = %ld, name = %s is awake\n",
               (long)getpid(), root->name);
    }
    else
    {
        for (i = 0; i < root->nr_children; ++i)
        {
            fprintf(stderr, "Parent %s, PID = %ld: Creating child %s\n", root-
>name, (long)getpid(), root->children[i].name);

            child[i] = fork();
            if (child[i] < 0)
            {
                /* fork failed */
                perror("Error at children");
                exit(1);
            }

            if (child[i] == 0)
            {
                fork_procs(&root->children[i]);
                exit(1);
            }

            wait_for_ready_children(1);
        }

        /*
        * Suspend Self
        */
        raise(SIGSTOP);
        printf("PID = %ld, name = %s is awake\n",
               (long)getpid(), root->name);
```

```c
        for (i = 0; i < root->nr_children; ++i)
        {
            kill(child[i], SIGCONT);
            waitpid(child[i], &status[i], 0);
            explain_wait_status(child[i], status[i]);
        }
    }

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

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);
    }
```

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

    /* 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;
}
```

Τρέχοντας το πρόγραμμα με είσοδο το αρχείο *proc.tree* που μας δίνεται λαμβάνουμε έξοδο:

```
giannis@pop-os:~/Desktop/OS/NTUA-Operating-Systems/Lab2/lab2/forktree$ ./ask2-signals proc.tree
PID = 14167, name A, starting...
Parent A, PID = 14167: Creating child B
PID = 14168, name B, starting...
Parent B, PID = 14168: Creating child E
PID = 14169, name E, starting...
My PID = 14168: Child PID = 14169 has been stopped by a signal, signo = 19
Parent B, PID = 14168: Creating child F
PID = 14170, name F, starting...
My PID = 14168: Child PID = 14170 has been stopped by a signal, signo = 19
My PID = 14167: Child PID = 14168 has been stopped by a signal, signo = 19
Parent A, PID = 14167: Creating child C
PID = 14171, name C, starting...
My PID = 14167: Child PID = 14171 has been stopped by a signal, signo = 19
Parent A, PID = 14167: Creating child D
PID = 14172, name D, starting...
My PID = 14167: Child PID = 14172 has been stopped by a signal, signo = 19
My PID = 14166: Child PID = 14167 has been stopped by a signal, signo = 19


A(14167)─┬─B(14168)─┬─E(14169)
         │          └─F(14170)
         ├─C(14171)
         └─D(14172)


PID = 14167, name = A is awake
PID = 14168, name = B is awake
PID = 14169, name = E is awake
My PID = 14168: Child PID = 14169 terminated normally, exit status = 0
PID = 14170, name = F is awake
My PID = 14168: Child PID = 14170 terminated normally, exit status = 0
My PID = 14167: Child PID = 14168 terminated normally, exit status = 0
PID = 14171, name = C is awake
My PID = 14167: Child PID = 14171 terminated normally, exit status = 0
PID = 14172, name = D is awake
My PID = 14167: Child PID = 14172 terminated normally, exit status = 0
My PID = 14166: Child PID = 14167 terminated normally, exit status = 0
```

Τρέχοντας το πρόγραμμα με είσοδο το αρχείο *proc2.tree* που δημιουργήσαμε λαμβάνουμε έξοδο:

```
giannis@pop-os:~/Desktop/OS/NTUA-Operating-Systems/Lab2/lab2/forktree$ ./ask2-signals proc2.tree
PID = 14319, name A, starting...
Parent A, PID = 14319: Creating child B
PID = 14320, name B, starting...
Parent B, PID = 14320: Creating child E
PID = 14321, name E, starting...
My PID = 14320: Child PID = 14321 has been stopped by a signal, signo = 19
Parent B, PID = 14320: Creating child F
PID = 14322, name F, starting...
My PID = 14320: Child PID = 14322 has been stopped by a signal, signo = 19
My PID = 14319: Child PID = 14320 has been stopped by a signal, signo = 19
Parent A, PID = 14319: Creating child C
PID = 14323, name C, starting...
Parent C, PID = 14323: Creating child G
PID = 14324, name G, starting...
Parent G, PID = 14324: Creating child K
PID = 14325, name K, starting...
My PID = 14324: Child PID = 14325 has been stopped by a signal, signo = 19
My PID = 14323: Child PID = 14324 has been stopped by a signal, signo = 19
My PID = 14319: Child PID = 14323 has been stopped by a signal, signo = 19
Parent A, PID = 14319: Creating child D
PID = 14326, name D, starting...
Parent D, PID = 14326: Creating child H
PID = 14327, name H, starting...
My PID = 14326: Child PID = 14327 has been stopped by a signal, signo = 19
My PID = 14319: Child PID = 14326 has been stopped by a signal, signo = 19
My PID = 14318: Child PID = 14319 has been stopped by a signal, signo = 19


A(14319)─┬─B(14320)─┬─E(14321)
         │          └─F(14322)
         ├─C(14323)──G(14324)──K(14325)
         └─D(14326)──H(14327)


PID = 14319, name = A is awake
PID = 14320, name = B is awake
PID = 14321, name = E is awake
My PID = 14320: Child PID = 14321 terminated normally, exit status = 0
PID = 14322, name = F is awake
My PID = 14320: Child PID = 14322 terminated normally, exit status = 0
My PID = 14319: Child PID = 14320 terminated normally, exit status = 0
PID = 14323, name = C is awake
PID = 14324, name = G is awake
PID = 14325, name = K is awake
My PID = 14324: Child PID = 14325 terminated normally, exit status = 0
My PID = 14323: Child PID = 14324 terminated normally, exit status = 0
My PID = 14319: Child PID = 14323 terminated normally, exit status = 0
PID = 14326, name = D is awake
PID = 14327, name = H is awake
My PID = 14326: Child PID = 14327 terminated normally, exit status = 0

My PID = 14319: Child PID = 14326 terminated normally, exit status = 0
My PID = 14318: Child PID = 14319 terminated normally, exit status = 0
```

1.3.1 Ερώτηση 1η

Στις προηγούμενες ασκήσεις αδρανοποιούσαμε τα παιδιά ώστε να εξασφαλίσουμε χρόνο πριν τερματιστούν ώστε να προλάβουν να αποτυπωθούν από την pstree που καλείται από την show_pstree(). Με την χρήση σημάτων αποφεύγουμε να ρυθμίζουμε αυθαίρετα τον χρόνο όπως κάναμε στην 1.2 με την sleep(), πετυχαίνοντας καλύτερο συγχρονισμό. Συγκεκριμένα μόλις “κοιμηθούν” με την raise(SIGSTOP) και αφού εμφανίσει το δέντρο στέλνει μήνυμα στα παιδιά με το kill(child_pid, SIGCONT) να ξυπνήσουν και να συνεχίσουν έως ότου τερματιστούν.

1.3.2 Ερώτηση 2η

```c
void wait_for_ready_children(int cnt)
{
    int i;
    pid_t p;
```

```
    int status;

    for (i = 0; i < cnt; i++) {
        /* Wait for any child, also get status for stopped children */
        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);
        }
    }
}
```

Όπως βλέπουμε από τον παραπάνω κώδικα η wait_for_ready_children() είναι μια συνάρτηση που αναστέλλει την λειτουργία της μέχρι τα παιδιά της να αδρανοποιηθούν. Ελέγχει ουσιαστικά τη σημαία WIFSTOPPED, που δείχνει αν το παιδί έχει σταματήσει και σε αντίθετη περίπτωση εμφανίζει μήνυμα για τον «ξαφνικό» θάνατο του παιδιού και κάνει exit(). Η χρήση της εξασφαλίζει πως όλα τα παιδιά είναι ζωντανά και έχουν σταματήσει, άρα το δέντρο διεργασιών θα εμφανιστεί σωστά και επίσης ότι θα έχουμε πλήρη γνώση αν κάποιο παιδί πέθανε πριν την ώρα του. Αν δε τη χρησιμοποιήσουμε, τότε ο πατέρας αφού ξυπνήσει θα περιμένει το θάνατο του παιδιού του, που έχει ήδη προηγηθεί, επομένως θα παραμένει αδρανής η διεργασία.

1.4 Παράλληλος υπολογισμός αριθμητικής έκφρασης

Για αυτό το ερώτημα χρησιμοποιούμε τον παρακάτω πηγαίο κώδικα. Ταυτόχρονα χρησιμοποιούμε τα αρχεία *proc-common.{c, h}, tree.{c, h}* που μας είναι δοσμένα.

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

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

#define SLEEP_PROC_SEC 10
#define SLEEP_TREE_SEC 3

void fork_procs(struct tree_node *node, int pfd[])
{
    pid_t child;
    int status;
    int i;

    change_pname(node->name);
    printf("Proccess %s has PID = %ld\n", node->name, (long)getpid());

    if (node->nr_children == 0)
    {
        printf("%s: Sleeping...\n", node->name);
```

```
        sleep(SLEEP_PROC_SEC);
        printf("%s: Done Sleeping...\n", node->name);

        if (close(pfd[0]) < 0)
        {
            perror("closing pipe");
            exit(1);
        }

        int val = atoi(node->name);

        if (write(pfd[1], &val, sizeof(val)) != sizeof(int))
        {
            perror("writing in pipe");
            exit(1);
        }
    }
    else
    {
        int operators[2] = {0, 0};
        int newpipe[2];

        if (pipe(newpipe) < 0)
        {
            perror("pipe");
            exit(1);
        }

        for (i = 0; i < 2; ++i)
        {
            fprintf(stderr, "Parent %s, PID = %ld: Creating child %s\n", node-
>name, (long)getpid(), node->children[i].name);
            child = fork();
            if (child < 0)
            {
                /* fork failed */
                perror("Error at children");
                exit(1);
            }

            if (child == 0)
            {
                fork_procs(&node->children[i], newpipe);
            }
        }

        if (close(newpipe[1]) < 0)
        {
            perror("closing pipe");
            exit(1);
        }

        for (i = 0; i < 2; i++)
```

```
        {
            if (read(newpipe[0], &operators[i], sizeof(operators[i])) != sizeof(operator
s[i]))
            {
                perror("Reading pipe");
                exit(1);
            }
        }

        int res;
        if (!strcmp(node->name, "+"))
        {
            res = operators[0] + operators[1];
            printf("Current operation is: %d + %d = %d\n", operators[0], operators[1], r
es);
        }
        else
        {
            res = operators[0] * operators[1];
            printf("Current operation is: %d * %d = %d\n", operators[0], operators[1], r
es);
        }

        if (close(pfd[0]) < 0)
        {
            perror("closing pipe");
            exit(1);
        }

        if (write(pfd[1], &res, sizeof(res)) != sizeof(res))
        {
            perror("writing in pipe");
            exit(1);
        }

        for (i = 0; i < 2; ++i)
        {
            child = wait(&status);
            explain_wait_status(child, status);
        }
    }
    printf("%s with PID = %ld is ready to terminate...\n%s: Exiting...\n", node-
>name, (long)getpid(), node->name);
    exit(0);
}

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

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

    root = get_tree_from_file(argv[1]);
    printf("Constructing the following process tree:\n");
    print_tree(root);

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

    printf("Parent: Creating child...\n");
    pid = fork();
    if (pid < 0)
    {
        /* fork failed */
        perror("main: fork");
        exit(1);
    }
    if (pid == 0)
    {
        fork_procs(root, pfd);
        exit(1);
    }

    /*
     * In parent process.
     */
    sleep(SLEEP_TREE_SEC);
    show_pstree(pid);

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

    /* Print the process tree root at pid */

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

    printf("The final result is %d\n", res);

    return 0;
}
```

Τρέχοντας το πρόγραμμα με είσοδο το αρχείο *expr.tree* που μας δίνεται λαμβάνουμε έξοδο:

```
giannis@pop-os:~/Desktop/OS/NTUA-Operating-Systems/Lab2/lab2/forktree$ ./ask2-pipes expr.tree
Constructing the following process tree:
+
        10
        *
                +
                        5
                        7
                4
Parent: Creating pipe...
Parent: Creating child...
Proccess + has PID = 15763
Parent +, PID = 15763: Creating child 10
Parent +, PID = 15763: Creating child *
Proccess 10 has PID = 15764
10: Sleeping...
Proccess * has PID = 15765
Parent *, PID = 15765: Creating child +
Parent *, PID = 15765: Creating child 4
Proccess 4 has PID = 15767
4: Sleeping...
Proccess + has PID = 15766
Parent +, PID = 15766: Creating child 5
Parent +, PID = 15766: Creating child 7
Proccess 5 has PID = 15768
5: Sleeping...
Proccess 7 has PID = 15769
7: Sleeping...


+(15763)─┬─*(15765)─┬─+(15766)─┬─5(15768)
         │          │          └─7(15769)
         │          └─4(15767)
         └─10(15764)


10: Done Sleeping...
10 with PID = 15764 is ready to terminate...
10: Exiting...
4: Done Sleeping...
4 with PID = 15767 is ready to terminate...
4: Exiting...
5: Done Sleeping...
5 with PID = 15768 is ready to terminate...
5: Exiting...
7: Done Sleeping...
7 with PID = 15769 is ready to terminate...
7: Exiting...
Current operation is: 5 + 7 = 12
Current operation is: 4 * 12 = 48
Current operation is: 10 + 48 = 58

My PID = 15763: Child PID = 15764 terminated normally, exit status = 0
My PID = 15765: Child PID = 15767 terminated normally, exit status = 0
My PID = 15766: Child PID = 15768 terminated normally, exit status = 0
My PID = 15766: Child PID = 15769 terminated normally, exit status = 0
+ with PID = 15766 is ready to terminate...
+: Exiting...
My PID = 15765: Child PID = 15766 terminated normally, exit status = 0
* with PID = 15765 is ready to terminate...
*: Exiting...
My PID = 15763: Child PID = 15765 terminated normally, exit status = 0
+ with PID = 15763 is ready to terminate...
+: Exiting...
My PID = 15762: Child PID = 15763 terminated normally, exit status = 0
The final result is 58
```

Τρέχοντας το πρόγραμμα με είσοδο το αρχείο *expr2.tree* που δημιουργήσαμε λαμβάνουμε έξοδο:

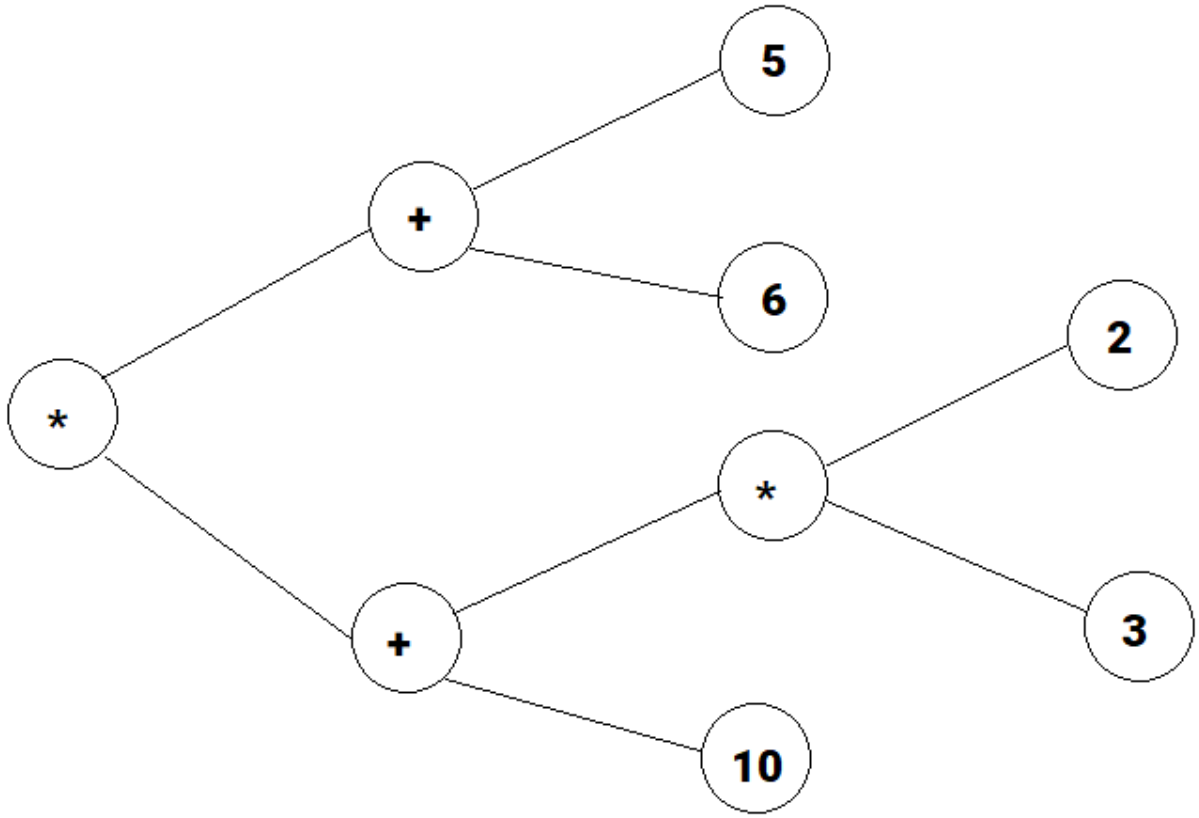
```
giannis@pop-os:~/Desktop/OS/NTUA-Operating-Systems/Lab2/lab2/forktree$ ./ask2-pipes expr2.tree
Constructing the following process tree:
*
        +
                5
                6
        +
                *
                        2
                        3
                10
Parent: Creating pipe...
Parent: Creating child...
Proccess * has PID = 16457
Parent *, PID = 16457: Creating child +
Parent *, PID = 16457: Creating child +
Proccess + has PID = 16458
Parent +, PID = 16458: Creating child 5
Parent +, PID = 16458: Creating child 6
Proccess 5 has PID = 16460
5: Sleeping...
Proccess + has PID = 16459
Parent +, PID = 16459: Creating child *
Parent +, PID = 16459: Creating child 10
Proccess * has PID = 16462
Parent *, PID = 16462: Creating child 2
Proccess 10 has PID = 16463
10: Sleeping...
Parent *, PID = 16462: Creating child 3
Proccess 2 has PID = 16464
2: Sleeping...
Proccess 3 has PID = 16465
3: Sleeping...
Proccess 6 has PID = 16461
6: Sleeping...


*(16457)─┬─+(16458)─┬─5(16460)
         │          └─6(16461)
         └─+(16459)─┬─*(16462)─┬─2(16464)
                    │          └─3(16465)
                    └─10(16463)


5: Done Sleeping...
5 with PID = 16460 is ready to terminate...
5: Exiting...
10: Done Sleeping...
10 with PID = 16463 is ready to terminate...
10: Exiting...
2: Done Sleeping...
```

```
2 with PID = 16464 is ready to terminate...
2: Exiting...
3: Done Sleeping...
3 with PID = 16465 is ready to terminate...
3: Exiting...
Current operation is: 2 * 3 = 6
Current operation is: 10 + 6 = 16
My PID = 16462: Child PID = 16464 terminated normally, exit status = 0
My PID = 16459: Child PID = 16463 terminated normally, exit status = 0
My PID = 16462: Child PID = 16465 terminated normally, exit status = 0
* with PID = 16462 is ready to terminate...
*: Exiting...
My PID = 16459: Child PID = 16462 terminated normally, exit status = 0
+ with PID = 16459 is ready to terminate...
+: Exiting...
6: Done Sleeping...
6 with PID = 16461 is ready to terminate...
6: Exiting...
Current operation is: 5 + 6 = 11
Current operation is: 16 * 11 = 176
My PID = 16458: Child PID = 16460 terminated normally, exit status = 0
My PID = 16458: Child PID = 16461 terminated normally, exit status = 0
My PID = 16457: Child PID = 16459 terminated normally, exit status = 0
+ with PID = 16458 is ready to terminate...
+: Exiting...
My PID = 16457: Child PID = 16458 terminated normally, exit status = 0
* with PID = 16457 is ready to terminate...
*: Exiting...
My PID = 16456: Child PID = 16457 terminated normally, exit status = 0
The final result is 176
```

Όπου είναι *expr2.tree* το:



1.4.1 Ερώτηση 1η

Στην άσκηση γίνεται η χρήση μία σωλήνωσης ανά διεργασία. Γενικά, αυτό είναι δυνατό γιατί τα writes μέσα στο pipe γίνονται ατομικά ,το οποίο σημαίνει ότι αν δύο η περισσότερα παιδιά προσπαθούν να γράψουν μέσα στο ίδιο pipe δεν θα υπάρχει περίπτωση να υπάρξει πρόβλημα με μπερδεμένα δεδομένα (εναλλάξ δεδομένα από τα διαφορετικά παιδιά). Συγκεκριμένα, είναι δυνατό διότι το δέντρο

διασχίζεται κατά βάθος και φτάνοντας στα φύλλα με το μεγαλύτερο βάθος αρχίζουν να γίνονται οι πράξεις της πρόσθεσης και του πολλαπλασιασμού, που είναι αντιμεταθετικές πράξεις. Στην περίπτωση που γινόντουσαν πράξεις όπως αφαίρεση και διαίρεση, που δεν ισχύει η αντιμεταθετική ιδιότητα, θα χρειαζόντουσαν τουλάχιστον δύο pipes.

1.4.2 Ερώτηση 2η

Σε ένα τέτοιο σύστημα μπορούν να τρέξουν ταυτόχρονα περισσότερες διεργασίες, αν αυτές δεν είναι αλληλεξαρτώμενες. Συγκεκριμένα σε αυτό το πρόγραμμα θα εκτελούνται παράλληλα οι διεργασίες παιδιά κάθε γονιού. Αυτό σημαίνει ότι μπορούμε στον ίδιο χρόνο να κάνουμε περισσότερες πράξεις και να λαμβάνουμε γρηγορότερα το τελικό αποτέλεσμα. Αν δεν έχουμε τέτοιο σύστημα αναγκαζόμαστε να ακολουθούμε το ρυθμό του επεξεργαστή και τη σειρά εντολών του κώδικα.