ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

Σχολη ΗΜ&ΜΥ
Λειτουργικά Συστήματα
Άσκηση 2
Ακ. Έτος 2012-13

Παπαδιάς Σεραφείμ | ΑΜ: 03109193
Παπαδημητρίου Κων/νος | ΑΜ: 03108769
Ημερομηνία: 09/01/2013

Άσκηση 1.1 Δημιουργία δεδομένου δέντρου διεργασιών

Ο κώδικας που γράψαμε, καθώς και η έξοδος του προγράμματος, βρίσκονται στα παραρτήματα, στο τέλος αυτής της αναφοράς.

Ερωτησεις

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

Αν τερματίσουμε τη διεργασία Α, τότε τα παιδία της (διεργασίες Β και C) θα συνεχίσουν να εκτελούνται και θα υιοθετηθούν από την init.

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

Στη μεταβλητή pid έχει αποθηκευτεί το Process ID της διεργασίας Α. Έτσι όταν κάνουμε show_pstree(pid) τυπώνεται το (υπο)δέντρο διεργασιών με ρίζα τη διεργασία Α. Αν αντί γι αυτό κάνουμε show_pstree(getpid()), θα τυπωθεί το (υπο)δέντρο διεργασιών με ρίζα το αρχικό μας πρόγραμμα. Δηλαδή θα φαίνεται και η διεργασία-πατέρας της Α (το ask2-fork).

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

Όταν υπάρχουν πολλές διεργασίες σε ένα σύστημα, μοιράζονται τους πόρους του (CPU, μνήμες, κλπ). Οπότε όσο περισσότερες διεργασίες υπάρχουν, τόσο μεγαλύτερος είναι ο χρόνος αναμονής για πολλές από αυτές. Έτσι, αν ο κάθε χρήστης έχει τη δυνατότητα να δημιουργήσει απεριόριστο αριθμό διεργασιών, το σύστημα μπορεί να καταστεί μη λειτουργικό, κάτι που πρέπει ο διαχειριστής να αποφύγει.

Άσκηση 1.2 Δημιουργία αυθαίρετου δέντρου διεργασιών

Ο κώδικας που γράψαμε, καθώς και η έξοδος του προγράμματος, βρίσκονται στα παραρτήματα, στο τέλος αυτής της αναφοράς.

Ερωτησεις

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

Οι διεργασίες δημιουργούνται κατα επίπεδο. Η σειρά εκτέλεσής τους όμως δε μπορεί να εξασφαλιστεί, γιατί αυτό εξαρτάται από τη χρονοδρομολόγηση.

Άσκηση 1.3 Αποστολή και χειρισμός σημάτων

Ο κώδικας που γράψαμε, καθώς και η έξοδος του προγράμματος, βρίσκονται στα παραρτήματα, στο τέλος αυτής της αναφοράς.

Ερωτησεις

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

Προφανώς είναι ποιό αξιόπιστη η χρήση σημάτων, καθώς δεν στηριζόμαστε στη χρονοδρομολόγηση και δεν χρειάζεται να περιμένουμε "τυχαίο" (και αρκετά μεγάλο) αριθμό δευτερολέπτων.

2. *Ποιός ο ρόλος της wait_for_ready_children(); Τί εξασφαλίζει η χρήση της και τί πρόβλημα θα δημιουργούσε η παράληψή της;*

Με τη wait_for_ready_children(), πριν στείλουμε σήματα στα παιδιά, έχουμε βεβαιωθεί πως αυτά είναι σε ready state για να δεχθούν σήμα. Αν παραλήψουμε να χρησιμοποιήσουμε τη wait_for_ready_children(), τα σήματα που θα στείλουμε θα αγνοηθούν. Έτσι περιμένουμε όλα τα παιδιά να κάνουν raise(SIGSTOP) ώστε να δεχθούν το SIGCONT.

Άσκηση 1.4 Παράλληλος υπολογισμός αριθμητικής έκφρασης

Ο κώδικας που γράψαμε, καθώς και η έξοδος του προγράμματος, βρίσκονται στα παραρτήματα, στο τέλος αυτής της αναφοράς.

Ερωτησεις

1. *Πόσες σωληνώσεις χρειάζονται στη συγκεκριμένη άσκηση ανά διεργασία; Θα μπορούσε κάθε γονική διεργασία να χρησιμοποιεί μόνο μία σωλήνωση για όλες τις διεργασίες παιδιά; Γενικά, μπορεί για κάθε αριθμητικό τελεστή να χρησιμοποιηθεί μόνο μια σωλήνωση;*

Για κάθε διεργασία-φύλλο του δέντρου χρειάζεται μόνο μια σωλήνωση· με τη χρήση της οποίας θα περάσουμε την τιμή του φύλλου στον πατέρα. Για τις διεργασίες-κόμβους χρησιμοποιούμε 3 σωληνώσεις· 2 για να πάρουμε τα αποτελέσματα από τα παιδιά και μια για να περάσουμε τη τιμή της πράξης που εκτελέσαμε στον πατέρα. Στη συγκεκριμένη άσκηση θα μπορούσαμε να χρησιμοποιήσουμε μόνο μια σωλήνωση και για τα 2 παιδιά, καθώς καί η πρόσθεση καί ο πολλαπλασιασμός είναι αντιμεταθετικές πράξεις. Έτσι όποιο αποτέλεσμα και να έρθει πρώτο, η τιμή που θα δώσουμε στον πατέρα είναι η ίδια. Η αφαίρεση από την άλλη δεν μπορεί να υποστηρίξει κάτι τέτοιο και χρειάζεται ξεχωριστή σωλήνωση για κάθε παιδί. (Το ίδιο και για τη διαίρεση)

2. *Σε ένα συστημα πολλαπλών επεξεργαστών, μπορούν να εκτελούνται παραπάνω από μια διεργασίες παράλληλα. Σε ένα τέτοιο σύστημα, τι πλεονέκτημα μπορεί να έχει η αποτίμηση της έκφρασης από δέντρο διεργασιών, έναντι της αποτίμησης από μία μόνο διεργασία;*

Σε ένα σύστημα πολλαπλών επεξεργαστών, η αποτίμηση από μία μόνο διεργασία δεν μπορεί να εκμεταλλευτεί πλήρως το hardware. Και αυτό συμβαίνει, επειδή πολλές πράξεις (όπως αυτές που βρίσκονται στο ίδιο επίπεδο) είναι εντελώς ανεξάρτητες μεταξύ τους, οπότε μπορούν να εκτελεστούν παράλληλα. Κάτι που επιταχύνει σημαντικά την αποτίμηση της έκφρασης.

ΠΑΡΑΡΤΗΜΑ 1

1. Κώδικας που γράψαμε για την άσκηση 1

```
// file: ask2-fork.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
#define EXIT_A                  16      /* A exit status */
#define EXIT_B                  19      /* B exit status */
#define EXIT_C                  17      /* C exit status */
#define EXIT_D                  13      /* D exit status */

void call_B(void);      // code for process B
void call_C(void);      // code for process C
void call_D(void);      // code for process D

/*
 * Create this process tree:
 * A-+-B---D
 *   `-C
 */

void
call_B(void){
  change_pname("B");
  pid_t d;
  /* create process D */
  if ((d = fork()) < 0){
    perror("fork");
    exit (1);
  }
  if (d == 0){
    /* in process D */
    call_D();
  }

  /* in process B */
  int status;
  d = wait(&status);
  explain_wait_status(d, status);
  printf("B: Exiting...\n");
  exit (EXIT_B);
}

void
call_C(void){
  change_pname("C");
```

```
  printf("C: Sleeping\n");
  sleep(SLEEP_PROC_SEC);
  printf("C: Exiting\n");
  exit (EXIT_C);
}

void
call_D(void){
  change_pname("D");
  printf("D: Sleeping\n");
  sleep(SLEEP_PROC_SEC);
  printf("D: Exiting\n");
  exit(EXIT_D);
}

void
fork_procs(void)
{
        /*
         * initial process is A.
         */

        change_pname("A");
        int status;
        pid_t p;

        /* create B */
        if ((p = fork()) < 0){
          perror("fork");
          exit (1);
        }
        if (p == 0){
          /* in process B */
          call_B();
        }
        /* in process A */

        /* continue with creating C */
        if ((p = fork()) < 0){
          perror("fork");
          exit (-1);
        }
        if (p==0){
          /* in process C */
          call_C();
        }

        /*in process A */
        p = wait(&status);
        explain_wait_status(p, status);
        p = wait(&status);
        explain_wait_status(p, status);
        printf("A: Exiting...\n");
        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.
 * 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(pid);

        /* 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;
}
```

2. Κώδικας που γράψαμε για την άσκηση 2

```
// file: ask2-tree.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"
#include "tree.h"

#define SLEEP_PROC_SEC  10
#define SLEEP_TREE_SEC  3

/*
 * Create this process tree:
 * A-+-B---D
 *   `-C
 */
void
fork_procs(struct tree_node *me)
{
        pid_t pid;
        int status, i;
        change_pname( me->name );
        // printf("%s: Just started\n",me->name);
        /* create children and call recursively
         * fork_procs().
         */
        for (i=0; i<(me->nr_children); i++){
          if ((pid = fork()) < 0){
            perror("fork");
            exit (1);
          }

          if (pid == 0){
            /* child process */
            me = me->children+i;
            fork_procs(me);
            exit (0);
          }
        }

        /* if i haven't any kids, i'm going to sleep */
        if ((me->nr_children) == 0){
          printf("%s: Going for a nap...\n",me->name);
          sleep(SLEEP_PROC_SEC);
        }

        /* wait for all children to die and
         * print the result of their exit status
         */
        for (i=0; i<(me->nr_children); i++){
          pid_t child= wait(&status);
          explain_wait_status(child, status);
        }

        /* Exit */
        printf("%s: Exiting...\n",me->name);
        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.
 */
int main(int argc, char **argv)
{
        pid_t pid;
        int status;
        struct tree_node * root;
        if (argc != 2){
          fprintf(stderr,"Usage: ./ask2-fork tree-file\n");
          exit (1);
        }
        root = get_tree_from_file(argv[1]);

        /* Fork root of process tree */
        if ((pid = fork()) < 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 */
        pid = wait(&status);
        explain_wait_status(pid, status);

        return 0;
}
```

3. Κώδικας που γράψαμε για την άσκηση 3

```
// file: ask2-signals.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
         */
        pid_t *ch_pids;             // array with root's children pids
        pid_t pid;
        int i,status;

        ch_pids = malloc(root->nr_children * sizeof(pid_t));      // allocate ch_pids

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

        /* create children */
        for (i=0; i<root->nr_children; i++){
          if ((pid = fork()) < 0){
            perror("fork");
            exit(1);
          }

          if (pid == 0){
            /* child */
            root = root->children+i;
            fork_procs(root);
            exit(0);
          }
          *(ch_pids+i) = pid;
        }

        /* wait for children to pause */
        wait_for_ready_children(root->nr_children);

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

        /* wake up and send SIGCONT to all of your children
         * (wait for one to exit and then send to an other)
         */
        for (i=0; i<root->nr_children; i++){
```

```
            kill(*(ch_pids+i), SIGCONT);
            pid = wait(&status);
            explain_wait_status(pid, status);
        }

        /* Exit */
        printf("%s: Exiting\n",root->name);
        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.
 */

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;
}
```

4. Κώδικας που γράψαμε για την άσκηση 4

```
// file: ask2-pipes.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"
#include "tree.h"

void
fork_procs(struct tree_node *me, int ffd)	//ffd -> father's pipe file descriptor.. I will write there
{
	pid_t pid;
	int status, i;
	int pipes[4];	// pipe to child_1 -> pipes[0..1] // pipe to child_2 -> pipes[2..3]
	int res[2];	// value_of_child_i -> res[i]
	int my_value;

	change_pname(me->name);

	/* create pipes */
	/* pipe_1 -> pipes[0..1] */
	if (pipe(pipes) != 0){
	  perror("pipe");
	  exit(-1);
	}
	/* pipe_2 -> pipes[2..3] */
	if (pipe(pipes+2) != 0){
	  perror("pipe");
	  exit(-1);
	}

	/* create children and call recursively
	 * fork_procs().
	 */
	for (i=0; i<(me->nr_children); i++){
	  if ((pid = fork()) < 0){
	    perror("fork");
	    exit (-1);
	  }
```

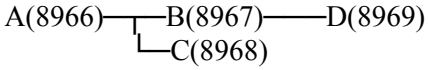
```
  if (pid == 0){
    /* child process */
    me = me->children+i;
    close(pipes[2*i]);                    // close read
    fork_procs(me, pipes[2*i+1]);         // give to child_1 pipe[1] and child_2 pipe[3]
    exit (0);
  }
}

// FATHER

// close write
if (close(pipes[1]) < 0){
  perror("close");
  exit(-1);
}
if (close(pipes[3]) < 0){
  perror("close");
  exit(-1);
}

/* search for the results in the pipes
 * put the result in res[i]
 */
for (i=0; i<me->nr_children; i++){
  if (read(pipes[2*i], res+i, sizeof(int)) != sizeof(int)){
    perror("read");
    exit (-1);
  }
  pid = wait(&status);
  explain_wait_status(pid, status);
  printf("%s(%d): child %d, gave %d\n", me->name, getpid(), i+1,  res[i]);
}


switch (*(me->name))
{
  case '*':{
    my_value = res[0] * res[1];
    break;
  }
  case '+':{
    my_value = res[0] + res[1];
    break;
  }
  default:{
    my_value = atoi(me->name);
//    sscanf(me->name, "%d", &my_value);
    break;
  }
}
printf("%s: My value: %d\n", me->name, my_value);
if (write(ffd, &my_value, sizeof(my_value)) != sizeof(my_value)){
  perror("write");
  exit(-1);
}
```

```
        printf("%s: Exiting\n",me->name);
        exit (0);
}

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

        struct tree_node * root;
        if (argc != 2){
          fprintf(stderr,"Usage: ./ask2-fork tree-file\n");
          exit (1);
        }
        root = get_tree_from_file(argv[1]);

        printf("=========================================\n");
        print_tree(root);
        printf("=========================================\n");

        if (pipe(fd) != 0){
          perror("pipe");
          exit(-1);
        }

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

        if (pid == 0) {
                /* Child */
                close(fd[0]);                   // close read
                fork_procs(root, fd[1]);
                exit(1);
        }
        close(fd[1]);                           // close write
        if (read(fd[0], &result, sizeof(int)) != sizeof(int)){
          perror("read");
          exit (-1);
        }

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

        printf("result = %d\n", result);
        return 0;
}
```

ΠΑΡΑΡΤΗΜΑ 2

1. Αποτελέσματα από την άσκηση 1

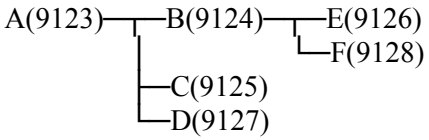
C: Sleeping
D: Sleeping



C: Exiting
D: Exiting
My PID = 8966: Child PID = 8968 terminated normally, exit status = 17
My PID = 8967: Child PID = 8969 terminated normally, exit status = 13
B: Exiting...
My PID = 8966: Child PID = 8967 terminated normally, exit status = 19
A: Exiting...
My PID = 8965: Child PID = 8966 terminated normally, exit status = 16

2. Αποτελέσματα από την άσκηση 2

D: Going for a nap...
F: Going for a nap...
C: Going for a nap...
E: Going for a nap...

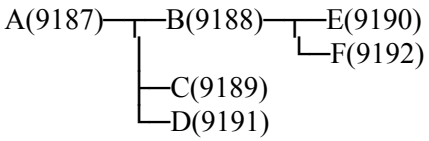


F: Exiting...
D: Exiting...
C: Exiting...
My PID = 9124: Child PID = 9128 terminated normally, exit status = 0
E: Exiting...
My PID = 9123: Child PID = 9125 terminated normally, exit status = 0
My PID = 9123: Child PID = 9127 terminated normally, exit status = 0
My PID = 9124: Child PID = 9126 terminated normally, exit status = 0
B: Exiting...
My PID = 9123: Child PID = 9124 terminated normally, exit status = 0
A: Exiting...
My PID = 9122: Child PID = 9123 terminated normally, exit status = 0

3. Αποτελέσματα από την άσκηση 3

PID = 9187, name A, starting...
PID = 9188, name B, starting...
PID = 9189, name C, starting...
PID = 9191, name D, starting...
C: Pausing..
D: Pausing..
My PID = 9187: Child PID = 9189 has been stopped by a signal, signo = 19
My PID = 9187: Child PID = 9191 has been stopped by a signal, signo = 19
PID = 9192, name F, starting...
F: Pausing..
PID = 9190, name E, starting...
E: Pausing..
My PID = 9188: Child PID = 9192 has been stopped by a signal, signo = 19
My PID = 9188: Child PID = 9190 has been stopped by a signal, signo = 19
B: Pausing..

My PID = 9187: Child PID = 9188 has been stopped by a signal, signo = 19
A: Pausing..
My PID = 9186: Child PID = 9187 has been stopped by a signal, signo = 19



PID = 9187, name = A is awake
PID = 9188, name = B is awake
PID = 9190, name = E is awake
E: Exiting
My PID = 9188: Child PID = 9190 terminated normally, exit status = 0
PID = 9192, name = F is awake
F: Exiting
My PID = 9188: Child PID = 9192 terminated normally, exit status = 0
B: Exiting
My PID = 9187: Child PID = 9188 terminated normally, exit status = 0
PID = 9189, name = C is awake
C: Exiting
My PID = 9187: Child PID = 9189 terminated normally, exit status = 0
PID = 9191, name = D is awake
D: Exiting
My PID = 9187: Child PID = 9191 terminated normally, exit status = 0
A: Exiting
My PID = 9186: Child PID = 9187 terminated normally, exit status = 0

4. Αποτελέσματα από την άσκηση 4

```
==========================================
+
      10
      *
            +
                  5
                  7
            4
==========================================
```

10: My value: 10
10: Exiting
My PID = 9227: Child PID = 9228 terminated normally, exit status = 0
+(9227): child 1, gave 10
4: My value: 4
4: Exiting
7: My value: 7
5: My value: 5
7: Exiting
5: Exiting
My PID = 9230: Child PID = 9233 terminated normally, exit status = 0
+(9230): child 1, gave 5
My PID = 9230: Child PID = 9232 terminated normally, exit status = 0
+(9230): child 2, gave 7
+: My value: 12
+: Exiting
My PID = 9229: Child PID = 9231 terminated normally, exit status = 0

```
*(9229): child 1, gave 12
My PID = 9229: Child PID = 9230 terminated normally, exit status = 0
*(9229): child 2, gave 4
*: My value: 48
*: Exiting
My PID = 9227: Child PID = 9229 terminated normally, exit status = 0
+(9227): child 2, gave 48
+: My value: 58
+: Exiting
My PID = 9226: Child PID = 9227 terminated normally, exit status = 0
result = 58
```

Υποσημείωση: Στις ασκήσεις 2, 3 και 4 δώσαμε τις εξόδους των προγραμμάτων δίνοντας είσοδο τα αρχεία που μας είχαν δωθεί (proc.tree για τις ασκήσεις 2, 3 και expr.tree για την άσκηση 4). Έχουμε επισυνάψει τα αρχεία του κώδικα που γράψαμε και χρησιμοποιήσαμε καθώς και ένα αρχείο με το δέντρο μιας έκφρασης που γράψαμε για δοκιμή (expr2.tree).