Ομάδα Α05
Θεοδώρου Αλέξανδρος 03114710
Μαππούρα Ελευθερία 03114706

Λειτουργικά Συστήματα

Άσκηση 2: Διαχείριση Διεργασιών και Διαδιεργασιακή Επικοινωνία

1.1 Δημιουργίαδεδομένουδέντρουδιεργασιών

```
#include <unistd.h>

#include <stdlib.h>

#include <stdio.h>

#include <signal.h>

#include <assert.h>


#include <sys/types.h>

#include <sys/wait.h>

#include "proc-common.h"


#define SLEEP_PROC_SEC 5

#define SLEEP_TREE_SEC  2


/*

 * Create this process tree:
```

```
 * A-+-B---D
 *   `-C
 */

void fork_procsC(void)
{
change_pname("c");

      sleep(SLEEP_PROC_SEC);

printf("C: Exiting...\n");
exit(17);
}

void fork_procsD(void)
{
change_pname("D");

      sleep(SLEEP_PROC_SEC);
```

```
printf("D: Exiting...\n");
exit(13);
}

void fork_procsB(void)
{
change_pname("B");
pid_tpid;
int status;

pid = fork();
	if (pid< 0) {
perror("main: fork");
exit(1);
	}
	if (pid == 0) {
fork_procsD();
exit(1);
```

```
        }



pid = wait(&status);

explain_wait_status(pid, status);


printf("B: Exiting...\n");

exit(19);

}


void fork_procs(void)

{


change_pname("A");

pid_t pid1,pid2;

int status1,status2;


    pid1 = fork();

        if (pid1 < 0) {
```

```
perror("main: fork");
exit(1);
	}
	if (pid1 == 0) {
fork_procsB();
exit(1);
	}

   pid2 = fork();
	if (pid2 < 0) {
perror("main: fork");
exit(1);
	}
	if (pid2 == 0) {
fork_procsC();
exit(1);
	}

	pid1 = wait(&status1);
```

```
explain_wait_status(pid1, status1);


   pid2 = wait(&status2);
explain_wait_status(pid2, status2);


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


int main(void)
{
pid_tpid;
int status;


pid = fork();
      if (pid< 0) {
perror("main: fork");
exit(1);
      }
```

```
	if (pid == 0) {
		/* Child */
fork_procs();
exit(1);
	}

	sleep(SLEEP_TREE_SEC);

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

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

return 0;
}
```

Ομάδα Α05
Θεοδώρου Αλέξανδρος 03114710
Μαππούρα Ελευθερία 03114706

Παρακάτω φαίνεται η έξοδος του προγράμματος

```
oslaba05@os-node1:~/lab2$ ./ask1,1


A(19675)─┬─B(19676)──D(19678)
         └─c(19677)


C: Exiting...
D: Exiting...
My PID = 19676: Child PID = 19678 terminated normally, exit status = 13
My PID = 19675: Child PID = 19677 terminated normally, exit status = 17
B: Exiting...
My PID = 19675: Child PID = 19676 terminated normally, exit status = 19
A: Exiting...
My PID = 19674: Child PID = 19675 terminated normally, exit status = 16
```

Απαντήσεις:

1. Τερματίζοντας μια διεργασία, έστω την “A” που έχει παιδιά, με “kill –Kill<pid>” τότε οι θυγατρικές διεργασίες ανατίθενται στην “init”. Η διεργασία που δημιούργησε την “A” δεν ενημερώνεται για τον τερματισμό της. Αυτό γίνεται διότι το σήμα –Kill (-9) προέρχεται από τον πυρήνα και δεν μπορεί μία διεργασία να το χειριστεί. Η διεργασία που δημιούργησε την “A” περιμένει για κάποιο χρονικό διάστημα μέχρι το wait() να γίνει time-out, δηλαδή να σταματήσει να περιμένει καθώς η διεργασία “A” δεν βρέθηκε.

2. Με την show_pstree(getpid()), θα έχουμε ως κεφαλή του δέντρου μας την final()(όνομα εκτελέσημου αρχείου, στη περίπτωσή μας ask1,1) αντί τη διεργασία “A”. Οπότε θα εμφανιστούν στο δέντρο διεργασιών εκτός από τις διεργασίες παιδία που δημιουργήσαμε και οι διεργασίες “sh” και “pstree” οι οποίες θα είναι επίσης παιδιά της final(). Ο λόγος είναι γιατί καλώντας με κεφαλή το PID της main, η show_pstree θα κάνει κάποια

systemcalls, τα οποία θα δημιουργήσουν τις παραπάνω διεργασίες και οπότε θα είναι παιδιά της final().

3. Αυτό γίνεται διότι υπάρχει η πιθανότητα ένας χρήστης να δημιουργήσει μια διεργασία που να δημιουργεί με την σειρά της ανεξέλεγκτα πολλές διεργασίες. Έτσι θέτοντας όριο στις διεργασίες που δημιουργούνται από κάθε χρήστη ο διαχειριστής επιτυγχάνει με αυτό το τρόπο ένα είδος μηχανισμού ελέγχου και περιορισμού της υπολογιστικής ισχύος που χρησιμοποιείται από κάθε χρήστη με αποτέλεσμα μια πιο δίκαιη κατανομή της ισχύος.

Ομάδα Α05
Θεοδώρου Αλέξανδρος 03114710
Μαππούρα Ελευθερία 03114706

1.2Δημιουργίααυθαίρετουδέντρουδιεργασιών

Ο κώδικας φαίνεται πιο κάτω:

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

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

#define SLEEP_PROC_SEC 10
#define SLEEP_TREE_SEC  4

void fork_procs(struct tree_node *root)
{
        change_pname(root->name);    //dinoumeonomastindiergasia
   if (root->nr_children> 0)        //an exeipediaksekinanakanei fork
   {
pid_tpid[root->nr_children];
int status[root->nr_children],i;
```

Ομάδα Α05
Θεοδώρου Αλέξανδρος 03114710
Μαππούρα Ελευθερία 03114706

```
    for (i=0; i< root->nr_children; i++)
      {
pid[i] = fork();
            if (pid[i] < 0) {
                perror("main: fork");
                exit(1);
              }
              if (pid[i] == 0) {
                fork_procs(root->children + i);
        //arxizoumetidimiourgiadiergasiwn
                exit(1);
              }

           }
      for (i=0; i< root->nr_children; i++)
      {
pid[i] = wait(&status[i]);  //perimeno ton prwtokomvo
        explain_wait_status(pid[i], status[i]);
      }
   }
   else sleep(SLEEP_PROC_SEC);          //idergasiakanei sleep k termatizei
exit(10);
```

```
}


intmain(intargc, char *argv[])
{
        struct tree_node *root;


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


        root = get_tree_from_file(argv[1]);     //an iparxe, dimiourgoume to
structure k pernoumetiriza


pid_tpid;
        int status;


        pid = fork();           //dimiourgiaprwtoukomvou
        if (pid< 0) {
                perror("main: fork");
                exit(1);
        }
```

```
        if (pid == 0) {
                fork_procs(root);   //arxizoumetidimiourgiadiergasiwn
                exit(1);
        }

    sleep(SLEEP_TREE_SEC);        //parousiasidentroudiergasiwn
        show_pstree(pid);

        pid = wait(&status);        //perimeno ton prwtokomvo
        explain_wait_status(pid, status);

        return 0;
}
```

Ομάδα Α05
Θεοδώρου Αλέξανδρος 03114710
Μαππούρα Ελευθερία 03114706

Παρακάτω φαίνεται η έξοδος του προγράμματος

```
oslaba05@os-node1:~/lab2$ ./ask1,2 input.txt
PID = 19768, name A, starting...
PID = 19769, name B, starting...
PID = 19770, name C, starting...
PID = 19771, name D, starting...


A(19768)─┬─B(19769)──D(19771)
         └─C(19770)


My PID = 19768: Child PID = 19770 terminated normally, exit status = 10
My PID = 19769: Child PID = 19771 terminated normally, exit status = 10
My PID = 19768: Child PID = 19769 terminated normally, exit status = 10
My PID = 19767: Child PID = 19768 terminated normally, exit status = 10
```

Απαντήσεις:

1. Η σειρά εμφάνισης των μηνυμάτων έναρξης και τερματισμού, όπως φαίνεται και πιο πάνω, γίνεται με τυχαία σειρά. Το μόνο σίγουρο είναι ότι πριν τερματιστεί μια διεργασία έχουν προηγουμένως τερματιστεί όλες οι διεργασίες-παιδια της.

Ομάδα Α05
Θεοδώρου Αλέξανδρος 03114710
Μαππούρα Ελευθερία 03114706

1.3Αποστολή και χειρισμός σημάτων

Ο κώδικας φαίνεται πιο κάτω:

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

   if (root->nr_children> 0)
   {
pid_tpid[root->nr_children];
```

```
int status[root->nr_children];


    for (i=0; i< root->nr_children; i++)
      {
pid[i] = fork();
        if (pid[i] < 0) {
            perror("main: fork");
            exit(1);
          }
          if (pid[i] == 0) {
            fork_procs(root->children + i);
            exit(1);
          }
printf("Parent, PID = %ld: Created child with PID = %ld, waiting
for it to terminate...\n",(long)getpid(),(long)pid[i]);
wait_for_ready_children(1);
   }
      raise(SIGSTOP);


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


   for (i=0; i< root->nr_children; i++)
```

```
    {
        kill(pid[i],SIGCONT);
pid[i] = wait(&status[i]);
    explain_wait_status(pid[i], status[i]);
    }


    }
    else {raise(SIGSTOP);  printf("PID = %ld, name = %s is awake\n",
            (long)getpid(), root->name); }


    exit(0);
}


intmain(intargc, char *argv[])
{
    pid_tpid;
    int status;
    struct tree_node *root;


    //elexosotiiparxeiarxio
    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]);

pid = fork();  //dimiourgiaprwtoukomvou
if (pid< 0) {
        perror("main: fork");
        exit(1);
}
if (pid == 0) {
        /* Child */
        fork_procs(root);   //arxizoumetidimiourgiadiergasiwn
        exit(1);
}

wait_for_ready_children(1);             //perimeneina alla3ei
                //katastasi to paidi tis (diladinakanei stop i root)

show_pstree(pid);                       //etsiexoun
//dimiourgitheiola ta paidia kai tipwnw to dentro

kill(pid, SIGCONT);                     //stelnwshmastopaidi
                ///gianasinexistoun oi diergasies
```

```
    wait(&status);                          //perimeno ton
                                    //prwtokomvonapethanei

    explain_wait_status(pid, status);


    return 0;

}
```

Ομάδα Α05
Θεοδώρου Αλέξανδρος 03114710
Μαππούρα Ελευθερία 03114706

Παρακάτω φαίνεται η έξοδος του προγράμματος

```
oslaba05@os-node1:~/lab2$ ./ask1,3 input.txt
PID = 19798, name A, starting...
Parent, PID = 19798: Created child with PID = 19799, waiting for it to terminate...
PID = 19799, name B, starting...
Parent, PID = 19799: Created child with PID = 19800, waiting for it to terminate...
PID = 19800, name D, starting...
My PID = 19799: Child PID = 19800 has been stopped by a signal, signo = 19
My PID = 19798: Child PID = 19799 has been stopped by a signal, signo = 19
Parent, PID = 19798: Created child with PID = 19801, waiting for it to terminate...
PID = 19801, name C, starting...
My PID = 19798: Child PID = 19801 has been stopped by a signal, signo = 19
My PID = 19797: Child PID = 19798 has been stopped by a signal, signo = 19


A(19798)─┬─B(19799)──D(19800)
         └─C(19801)


PID = 19798, name = A is awake
PID = 19799, name = B is awake
PID = 19800, name = D is awake
My PID = 19799: Child PID = 19800 terminated normally, exit status = 0
My PID = 19798: Child PID = 19799 terminated normally, exit status = 0
PID = 19801, name = C is awake
My PID = 19798: Child PID = 19801 terminated normally, exit status = 0
My PID = 19797: Child PID = 19798 terminated normally, exit status = 0
```

Απαντήσεις:

1. Χρησιμοποιώντας την sleep() μια διεργασία “κοιμάται” για προκαθορισμένο χρονικό διάστημα. Σε κάθε περίπτωση η διεργασία θα γίνει “ready” μόνο όταν περάσει αυτό το χρονικό διάστημα. Η χρήση των σημάτων μας επιτρέπει να καθορίσουμε εμείς σε ποιο σημείο της εκτέλεσης του κώδικα θα γίνει η διεργασία “ready”. Με αυτό τον τρόπο μια διεργασία δεν πρόκειται να γίνει “ready” νωρίτερα από ότι χρειάζεται για την ορθή εκτέλεση του προγράμματος. Επίσης δεν πρόκειται να παραμείνει σε κατάσταση αναμονής για περισσότερο χρόνο από όσο χρειάζεται.

Ομάδα Α05
Θεοδώρου Αλέξανδρος 03114710
Μαππούρα Ελευθερία 03114706

2. Ο ρόλος της wait_for_ready_children(n) είναι να περιμένει μέχρι n-παιδιά της διεργασίας που την καλεί, να στείλουν μήνυμα SIGSTOP. Τελικά, με την χρήση της συνάρτησης αυτής εξασφαλίζουμε ότι μια διεργασία – πατέρας θα περιμένει όλα τα παιδιά του να δημιουργηθούν και να μπουν σε κατάσταση αναμονής. Με τον τρόπο που την χρησιμοποιούμε στον κώδικά μας εξασφαλίζουμε, επίσης, και το Depth-First. Η τυχόν παράλειψη αυτής της εντολής θα είχε την εξής συνέπεια. Κάθε διεργασία-πατέρας θα δημιουργούσε τα παιδιά της και πιθανότατα θα τερμάτιζε πριν προλάβουν τα παιδιά της τα τεραματιστούν. Τελικά, το δέντρο διεργασιών δεν θα ήταν το επιθυμητό. Κάποιες διεργασίες θα ανατίθονταν στην “init”, ενώ, δεν θα μπορούσαμε να εξασφαλίσουμε ούτε διάσχυση κατά βάθος.

Ομάδα Α05
Θεοδώρου Αλέξανδρος 03114710
Μαππούρα Ελευθερία 03114706

1.4Παράλληλος υπολογισμός αριθμητικής έκφρασης

Ο κώδικας φαίνεται πιο κάτω:

```
#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, int *pfd)
{

        printf("PID = %ld, name %s, starting...\n",
                        (long)getpid(), root->name);
        change_pname(root->name);           //dinwonomasti
                                            //diergasia
intnum = atoi (root->name);
```

Ομάδα Α05
Θεοδώρου Αλέξανδρος 03114710
Μαππούρα Ελευθερία 03114706

```
   if (root->nr_children> 0)        //an exeipediaksekinana
                                    //kanei fork
   {
pid_tpid[2];
int status[2],ch_pfd[2],i,num1,num2,ans;;


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


    for (i=0; i< 2; i++)
    {
pid[i] = fork();
            if (pid[i] < 0) {
                perror("main: fork");
                exit(1);
              }
              if (pid[i] == 0) {
                fork_procs(root->children + i,ch_pfd);
                exit(1);
              }
   }
```

Ομάδα A05
Θεοδώρου Αλέξανδρος 03114710
Μαππούρα Ελευθερία 03114706

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

pid[i] = wait(&status[i]);
        explain_wait_status(pid[i], status[i]);
    }

   if (read(ch_pfd[0], &num1, sizeof(int) ) != sizeof(int) ) {
                perror("read from pipe");
                exit(1);
                }

        if (read(ch_pfd[0], &num2, sizeof(int) ) != sizeof(int) ) {
                perror("read from pipe");
                exit(1);
                }
int x = root->name[0];
   if (x ==  42) ans = (num1)*(num2);
        else if (x ==  43) ans = num1+num2;
        else exit(1);

printf("%d %d %d\n\n",ans,num1,num2);
```

Ομάδα Α05
Θεοδώρου Αλέξανδρος 03114710
Μαππούρα Ελευθερία 03114706

```
    if (write(pfd[1],&ans, sizeof(int)) != sizeof(int) ) {
                                perror("write to pipe");
                                exit(1);
                        }


    }
    else {
       if (write(pfd[1],&num, sizeof(int)) != sizeof(int) ) {
         //writes to the pipe
                                perror("write to pipe");
                                exit(1);
                        }
       }


         exit(0);
}


intmain(intargc, char *argv[])
{
```

```
        pid_tpid;
        intstatus,ans,pfd[2];
        struct tree_node *root;

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

        root = get_tree_from_file(argv[1]);     //an iparxei
                                        //dimiourgoume to structure
print_tree(root);
if (root==NULL)                                         //an einai
                                                        //adeio exit
                exit(1);

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

        pid = fork();                   //dimiourgiaprwtoukomvou
        if (pid< 0) {
```

```
        perror("main: fork");
        exit(1);
    }
    if (pid == 0) {

        fork_procs(root,pfd);       //arxizoumeti
//dimiourgiadiergasiwn
        exit(1);
    }

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

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

    printf("The answer is: %d\n\n",ans);

    return 0;
}
```

Ομάδα Α05
Θεοδώρου Αλέξανδρος 03114710
Μαππούρα Ελευθερία 03114706

```
oslaba05@os-node1:~/lab2$ ./ask1,4 input2.txt
*
        +
                5
                6
        10
PID = 28718, name *, starting...
PID = 28719, name +, starting...
PID = 28720, name 10, starting...
PID = 28721, name 5, starting...
My PID = 28718: Child PID = 28720 terminated normally, exit status = 0
PID = 28722, name 6, starting...
My PID = 28719: Child PID = 28721 terminated normally, exit status = 0
My PID = 28719: Child PID = 28722 terminated normally, exit status = 0
11 5 6

My PID = 28718: Child PID = 28719 terminated normally, exit status = 0
110 10 11

My PID = 28717: Child PID = 28718 terminated normally, exit status = 0
The answer is: 110
```

Παρακάτω φαίνεται η έξοδος του προγράμματος

Απαντήσεις:

1. Στην άσκηση αυτή μπορούμε να έχουμε μέχρι και ένα pipe ανά κόμβο-τελεστή. Αυτό επιτρέπεται καθώς η πρόσθεση και ο πολλαπλασιασμός είναι αντιμεταθετικές πράξεις. Αυτό δεν μπορεί να ισχύει στην περίπτωση που είχαμε σαν τελεστές την αφαίρεση και την διαίρεση, διότι παίζει ρόλο η σειρά των πράξεων.

2. Το πλεονέκτημα στην περίπτωση αυτή είναι πως μπορούμε να έχουμε παράλληλη εκτέλεση των διεργασιών με αποτέλεσμα η τελική αποτίμηση να είναι ταχύτερη από ότι αν εκτελείτο από μία διεργασία.Όμως χρειάζεται

χρόνος για την δημιουργία κάθε διεργασίες και αυτό είναι το τίμημα που πληρώνουμε.

Ο κώδικας του **Makefile** φαίνεται πιο κάτω:

```
.PHONY: all clean
all: ask1,1 ask1,2 ask1,3 ask1,4
CC = gcc
CFLAGS = -g -Wall -O2
SHELL= /bin/bash

ask1,1: ask1,1.o  proc-common.o
	$(CC) $(CFLAGS) $^ -o $@

ask1,2: ask1,2.otree.o proc-common.o
	$(CC) $(CFLAGS) $^ -o $@

ask1,3: ask1,3.otree.o proc-common.o
	$(CC) $(CFLAGS) $^ -o $@

ask1,4: ask1,4.otree.o proc-common.o
	$(CC) $(CFLAGS) $^ -o $@



%.s: %.c
	$(CC) $(CFLAGS) -S -fverbose-asm $<

%.o: %.c
	$(CC) $(CFLAGS) -c $<

%.i: %.c
gcc -Wall -E $< | indent -kr> $@
```

Ομάδα Α05
Θεοδώρου Αλέξανδρος 03114710
Μαππούρα Ελευθερία 03114706

```
clean:
rm -f *.o ask1,1 ask1,2 ask1,3 ask1,4 -{fork,tree,signals,pipes}
```