ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Λειτουργικά Συστήματα

7ο εξάμηνο, Ακαδημαϊκή περίοδος 2017-2018

Άσκηση 4: Χρονοδρομολόγηση

	Team: oslaba27
Ονοματεπώνυμο	Αριθμός Μητρώου
Κερασιώτης Ιωάννης	03114951
Ραφτόπουλος Ευάγγελος	03114743

Πηγαίος Κώδικας:

Για την παρακάτω άσκηση δημιουργήθηκε το header file, list.h, με σκοπό την διαχείριση των λιστών διεργασιών.

Ο κώδικας παρατήθηκε παρακάτω:

```
#ifndef PROC_LIST_H_
#define PROC_LIST_H_

#define LOW 0
#define HIGH 1

/*
 * A struct which represent
 * every node of the list.
 */

struct process {
	int pNumber;
	int priority;
	pid_t pPid;
	char* execName;
	struct process* next;
};

/*
 * A struct which is used
 * for construction of list
 */

struct processList{
	int cnt;
	struct process* head;
};

/*
 * Type Definition
 */

typedef struct process* processT;
typedef struct processList* processListT;

/*
 * A function which initialize a process
 */
```

```
processT initProc(char* execName, int N){

	processT proc = malloc(sizeof(struct process));
	if(proc == NULL){
		perror("");
		exit(-1);
	}

	proc->pNumber = N;
	proc->priority = LOW;
	proc->pPid = -1;
	proc->execName = malloc((strlen(execName)+1) * sizeof(char));
	strcpy(proc->execName, execName);
	proc->next = NULL;

	return proc;
}

/*
 * A function which initialize a list
 */

processListT initList(){

	processListT list = malloc(sizeof(struct processList));
	if(list == NULL){
		perror("");
		exit(-1);
	}

	list->cnt = 0;
	list->head = NULL;

	return list;

}

/*
 * Usage of this function is to
 * add a proccess node to the list
 */

void addToList(processListT list, processT proc){

	list->cnt++;
	if(list->head == NULL) list->head = proc;
	else{
```

```
                processT tmp = list->head;
                while(tmp->next != NULL) tmp = tmp->next;
                tmp->next = proc;
        }

}

/*
 * Usage of this function is to
 * remove a proccess node to the list
 */

void removeFromList(processListT list, processT proc){

        processT temp = list->head;
        processT prev = NULL;

        list->cnt--;

        if(list->cnt == 0) list->head = NULL;
        else{
                while(temp != proc){
                        prev = temp;
                        temp = temp->next;
                }
                if (prev){
                        prev->next = temp->next;
                        temp->pPid = -1;
                }
                else{
                        list->head = temp->next;
                        temp->pPid = -1;
                }
        }
}

/*
 * Find a new ID for a new task
 * that is created
 */

int findNewID(processListT procList){

        int id = 1;

        while(1){
                processT temp = procList->head;
```

```
                int flag=1;
                while(temp != NULL){
                        if(temp->pNumber == id){ flag=0; break;}
                        temp = temp->next;
                }
                if(flag == 1) return id;
                id++;
        }
}

/*
 * A simple function that
 * find a node with given ID
 */

processT findProcID(processListT list, int id, int * flag){

        processT temp = list->head;

        while(temp != NULL){
                if(temp->pNumber == id) {*flag = 1; break;}
                temp = temp->next;
        }

        return temp;

}

#endif
```

Άσκηση 1.1

Πηγαίος Κώδικας:

scheduler.c

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

#include <sys/wait.h>
#include <sys/types.h>

#include "proc-common.h"
#include "request.h"
#include "list.h"

processT curr;
processListT procList;


/* Compile-time parameters. */
#define SCHED_TQ_SEC 2                /* time quantum */
#define TASK_NAME_SZ 60               /* maximum size for a task's name */


/*
 * SIGALRM handler
 */
static void
sigalrm_handler(int signum)
{
        if (signum != SIGALRM) {
                fprintf(stderr, "Internal error: Called for signum %d, not SIGALRM\n",
                        signum);
                exit(1);
        }
        fprintf(stderr,"Alarm!!! -.- \n");
        kill(curr->pPid, SIGSTOP);
```

```
}

/*
 * SIGCHLD handler
 */
static void sigchld_handler(int signum)
{
	pid_t p;
	int status;

	if (signum != SIGCHLD) {
		fprintf(stderr, "Internal error: Called for signum %d, not SIGCHLD\n",
			signum);
		exit(1);
	}

	/*
	 * Something has happened to one of the children.
	 * We use waitpid() with the WUNTRACED flag, instead of wait(), because
	 * SIGCHLD may have been received for a stopped, not dead child.
	 *
	 * A single SIGCHLD may be received if many processes die at the same time.
	 * We use waitpid() with the WNOHANG flag in a loop, to make sure all
	 * children are taken care of before leaving the handler.
	 */
	for(;;) {
		p = waitpid(-1, &status, WUNTRACED | WNOHANG);
		if (p < 0) {
			perror("waitpid");
			exit(1);
		}

		if (p == 0)
			break;

		explain_wait_status(p, status);

		if (WIFEXITED(status) || WIFSIGNALED(status)) {
		/* A child has died */
			struct process* temp = curr;
			curr = curr->next;
			fprintf(stderr,"Proccess with pid=%ld terminated...\n", (long int)temp->pPid);
			removeFromList(procList, temp);
			if(procList->head == NULL) {
				printf("All children finished. Exiting...\n");
				exit(EXIT_SUCCESS);
			}
```

```
                        if(curr == NULL)
                                curr = procList->head;

                        /* start alarm */
                        if (alarm(SCHED_TQ_SEC) < 0) {
                                perror("alarm");
                                exit(1);
                        }

                        fprintf(stderr,"Proccess with pid=%ld is about to begin...\n", (long int)curr->pPid);
                        kill(curr->pPid, SIGCONT);
                }

                if (WIFSTOPPED(status)) {
                        /* A child has stopped due to SIGSTOP/SIGTSTP, etc... */
                        curr = curr->next;
                        if (curr == NULL)
                                curr = procList->head;

                        /* start alarm */
                        if (alarm(SCHED_TQ_SEC) < 0) {
                                perror("alarm");
                                exit(1);
                        }
                        fprintf(stderr,"Proccess with pid=%ld is about to begin...\n", (long int)curr->pPid);
                        kill(curr->pPid, SIGCONT);
                }
        }
}

/* Install two signal handlers.
 * One for SIGCHLD, one for SIGALRM.
 * Make sure both signals are masked when one of them is running.
 */
static void
install_signal_handlers(void)
{
        sigset_t sigset;
        struct sigaction sa;

        sa.sa_handler = sigchld_handler;
        sa.sa_flags = SA_RESTART;
        sigemptyset(&sigset);
        sigaddset(&sigset, SIGCHLD);
        sigaddset(&sigset, SIGALRM);
```

```
        sa.sa_mask = sigset;
        if (sigaction(SIGCHLD, &sa, NULL) < 0) {
                perror("sigaction: sigchld");
                exit(1);
        }

        sa.sa_handler = sigalrm_handler;
        if (sigaction(SIGALRM, &sa, NULL) < 0) {
                perror("sigaction: sigalrm");
                exit(1);
        }

        /*
         * Ignore SIGPIPE, so that write()s to pipes
         * with no reader do not result in us being killed,
         * and write() returns EPIPE instead.
         */
        if (signal(SIGPIPE, SIG_IGN) < 0) {
                perror("signal: sigpipe");
                exit(1);
        }
}

void executeProg(processT proc){

        char *newargv[] = { proc->execName, NULL, NULL, NULL };
        char *newenviron[] = { NULL };

        execve(proc->execName, newargv, newenviron);

        perror("Execve:");
        exit(-1);
}

int main(int argc, char *argv[])
{
        int nproc;
        pid_t pid;
        int i;

        /*
         * For each of argv[1] to argv[argc - 1],
         * create a new child process, add it to the process list.
         */

        procList = initList();
        nproc = argc - 1; /* number of proccesses goes here */
```

```
for(i=1; i<argc; i++){
        curr = initProc(argv[i], i);
        addToList(procList,curr);
}

curr = procList->head;
for(i=0; i<nproc; i++){
        pid = fork();
        if(pid < 0){
                perror("");
                exit(-1);
        }
        if(pid == 0){
                fprintf(stderr,"A new proccess is created with pid=%ld \n",(long int)getpid());
                raise(SIGSTOP);
                executeProg(curr);
        }
        if(pid > 0){
                curr->pPid = pid;
                curr = curr->next;
        }
}

/* Wait for all children to raise SIGSTOP before exec()ing. */
wait_for_ready_children(nproc);

/* Install SIGALRM and SIGCHLD handlers. */
install_signal_handlers();

if (nproc == 0) {
        fprintf(stderr, "Scheduler: No tasks. Exiting...\n");
        exit(1);
}

curr = procList->head;

if (alarm(SCHED_TQ_SEC) < 0) {
        perror("alarm");
        exit(-1);
}

fprintf(stderr,"Proccess with pid=%ld is about to begin...\n", (long int)curr->pPid);
kill(curr->pPid, SIGCONT);

/* loop forever  until we exit from inside a signal handler. */
while (pause())
```

```
                ;

        /* Unreachable */
        fprintf(stderr, "Internal error: Reached unreachable point\n");
        return 1;
}
```

Έξοδος εκτέλεσης προγράμματος:

Μια τυπική έξοδος του προγράμματος με κλήση προγράμματος :
$ ./schelduler prog prog prog prog

Είναι η παρακάτω:

A new proccess is created with pid=8453
A new proccess is created with pid=8454
A new proccess is created with pid=8455
My PID = 8452: Child PID = 8453 has been stopped by a signal, signo = 19
My PID = 8452: Child PID = 8454 has been stopped by a signal, signo = 19
My PID = 8452: Child PID = 8455 has been stopped by a signal, signo = 19
A new proccess is created with pid=8456
My PID = 8452: Child PID = 8456 has been stopped by a signal, signo = 19
Proccess with pid=8453 is about to begin...
prog: Starting, NMSG = 20, delay = 142
prog[8453]: This is message 0
prog[8453]: This is message 1
prog[8453]: This is message 2
prog[8453]: This is message 3
prog[8453]: This is message 4
prog[8453]: This is message 5
prog[8453]: This is message 6
Alarm!!! -.-
My PID = 8452: Child PID = 8453 has been stopped by a signal, signo = 19
Proccess with pid=8454 is about to begin...
prog: Starting, NMSG = 20, delay = 59
prog[8454]: This is message 0
prog[8454]: This is message 1
prog[8454]: This is message 2
prog[8454]: This is message 3
prog[8454]: This is message 4
prog[8454]: This is message 5
prog[8454]: This is message 6
prog[8454]: This is message 7
prog[8454]: This is message 8

prog[8454]: This is message 9
prog[8454]: This is message 10
prog[8454]: This is message 11
prog[8454]: This is message 12
prog[8454]: This is message 13
prog[8454]: This is message 14
Alarm!!! -.-
My PID = 8452: Child PID = 8454 has been stopped by a signal, signo = 19
Proccess with pid=8455 is about to begin...
prog: Starting, NMSG = 20, delay = 105
prog[8455]: This is message 0
prog[8455]: This is message 1
prog[8455]: This is message 2
prog[8455]: This is message 3
prog[8455]: This is message 4
prog[8455]: This is message 5
prog[8455]: This is message 6
prog[8455]: This is message 7
prog[8455]: This is message 8
Alarm!!! -.-
My PID = 8452: Child PID = 8455 has been stopped by a signal, signo = 19
Proccess with pid=8456 is about to begin...
prog: Starting, NMSG = 20, delay = 152
prog[8456]: This is message 0
prog[8456]: This is message 1
prog[8456]: This is message 2
prog[8456]: This is message 3
prog[8456]: This is message 4
prog[8456]: This is message 5
Alarm!!! -.-
My PID = 8452: Child PID = 8456 has been stopped by a signal, signo = 19
Proccess with pid=8453 is about to begin...
prog[8453]: This is message 7
prog[8453]: This is message 8
prog[8453]: This is message 9
prog[8453]: This is message 10
prog[8453]: This is message 11
prog[8453]: This is message 12
Alarm!!! -.-
My PID = 8452: Child PID = 8453 has been stopped by a signal, signo = 19
Proccess with pid=8454 is about to begin...
prog[8454]: This is message 15
prog[8454]: This is message 16
prog[8454]: This is message 17
prog[8454]: This is message 18
prog[8454]: This is message 19
My PID = 8452: Child PID = 8454 terminated normally, exit status = 0

Proccess with pid=8454 terminated...
Proccess with pid=8455 is about to begin...
prog[8455]: This is message 9
prog[8455]: This is message 10
prog[8455]: This is message 11
prog[8455]: This is message 12
prog[8455]: This is message 13
prog[8455]: This is message 14
prog[8455]: This is message 15
prog[8455]: This is message 16
Alarm!!! -.-
My PID = 8452: Child PID = 8455 has been stopped by a signal, signo = 19
Proccess with pid=8456 is about to begin...
prog[8456]: This is message 6
prog[8456]: This is message 7
prog[8456]: This is message 8
prog[8456]: This is message 9
prog[8456]: This is message 10
prog[8456]: This is message 11
Alarm!!! -.-
My PID = 8452: Child PID = 8456 has been stopped by a signal, signo = 19
Proccess with pid=8453 is about to begin...
prog[8453]: This is message 13
prog[8453]: This is message 14
prog[8453]: This is message 15
prog[8453]: This is message 16
prog[8453]: This is message 17
prog[8453]: This is message 18
prog[8453]: This is message 19
Alarm!!! -.-
My PID = 8452: Child PID = 8453 has been stopped by a signal, signo = 19
Proccess with pid=8455 is about to begin...
prog[8455]: This is message 17
prog[8455]: This is message 18
prog[8455]: This is message 19
My PID = 8452: Child PID = 8455 terminated normally, exit status = 0
Proccess with pid=8455 terminated...
Proccess with pid=8456 is about to begin...
prog[8456]: This is message 12
prog[8456]: This is message 13
prog[8456]: This is message 14
prog[8456]: This is message 15
prog[8456]: This is message 16
prog[8456]: This is message 17
Alarm!!! -.-
My PID = 8452: Child PID = 8456 has been stopped by a signal, signo = 19
Proccess with pid=8453 is about to begin...

My PID = 8452: Child PID = 8453 terminated normally, exit status = 0
Proccess with pid=8453 terminated...
Proccess with pid=8456 is about to begin...
prog[8456]: This is message 18
prog[8456]: This is message 19
My PID = 8452: Child PID = 8456 terminated normally, exit status = 0
Proccess with pid=8456 terminated...
All children finished. Exiting…

Ερωτήσεις:

1. **Τι συμβαίνει αν το σήμα SIGALRM έρθει ενώ εκτελείται η συνάρτηση χειρισμού του σήματος SIGCHLD ή το αντίστροφο; Πώς αντιμετωπίζει ένας πραγματικός χρονο-δρομολογητής χώρου πυρήνα ανάλογα ενδεχόμενα και πώς η δική σας υλοποίηση; Υπόδειξη: μελετήστε τη συνάρτηση install_signal_handlers() που δίνεται.**

   Η διαδικασία που θα ακολουθηθεί στην περίπτωση αυτή είναι όταν εκτελείται ο handler της SIGCHLD, τότε δεν εκτελείται η συνάρτηση χειρισμού του SIGALRM ακόμη και αν ληφθεί τέτοιο σήμα. Αυτο συμβαίνει καθώς στη δοσμένη install_signal_hanlders() έχουμε ορίσει μέσω μασκας να μπλοκάρεται το σήμα SIGALRM όταν εκτελείται το τμήμα κώδικα του SIGCHLD handler. Αντίστοιχα, έχει οριστεί και στην αντίθετη περίπτωση, όταν δηλαδή εκτελείται ο SIGALRM handler και ληφθεί σήμα SIGCHLD. Όσον αφορά έναν πραγματικό χρονοδρομολογητή, αυτός λειτουργεί με διακοπές και δε βασίζεται σε σήματα (μπορεί να εννίοτε αναξιόπιστα). Με αυτή την τακτική, έχουμε καλύτερη και πιο άμεση απόκριση, αφού με το που γίνει μια διακοπή ΑΜΕΣΑ θα εκτελεστεί η ρουτίνα εξυπηρέτησής της, ενώ στη περίπτωσή μας, τα σήματα ενδέχεται να έχουν καθυστερήσεις καθώς ακόμη και τα σήματα χρονοδρομολογούνται. Αυτός είναι ο κύριος λόγος που χρησιμοποιούμε διακοπές αντί για σήματα στους πραγματικούς χρονοδρομολογητές

2. **Κάθε φορά που ο χρονοδρομολογητής λαμβάνει σήμα SIGCHLD, σε ποια διεργασία-παιδί περιμένετε να αναφέρεται αυτό; Τι συμβαίνει αν λόγω εξωτερικού παράγοντα (π.χ. αποστολή SIGKILL) τερματιστεί αναπάντεχα μια οποιαδήποτε διεργασία-παιδί;**

   Το σήμα SIGCHLD το λαμβάνει ο χρονοδρομολογητής μας, όταν αλλάξει κάποιο παιδί την κατάστασή του. Στην περίπτωσή μας αυτό συμβαίνει όταν ένα παιδί δεχθεί σήμα SIGSTOP(από το χρονοδρομολογητή) ή τερματιστεί κανονικά. (σε αυτή την άσκηση δεν έχουμε Kill,αν και ελέγχουμε και αυτή την κατάσταση γιατί ενδέχεται να μπορεί να γίνει μέσω άλλου terminal) Το τι έπαθε το παιδί το μαθαίνουμε με χρήση των waitpid() και explain_wait_status(). Αφου ενημερωθούμε λοιπόν για αυτό, ανάλογα με την περίπτωση κάνουμε και τις κατάλληλες ενέργειες για τη λίστα μας (αφαίρεση κόμβου,κυκλική περιστροφή όπως εξηγήθηκαν και παραπάνω). Το πιο λογικό θα ήταν τέτοιο σήμα να ληφθεί απο την κεφαλή της λίστας ωστόσο σήμα KILL μπορεί να λάβει οποιαδήποτε από τις διεργασίες μας, οπότε διατρέχουμε τη λίστα και βλέπουμε κάθε φορά ποια είναι η διεργασία για την οποία λήφθηκε το σήμα SIGCHLD. Έτσι, όποια διεργασία και να πάθει το οτιδήποτε, ο χρονοδρομολογητής μας θα λειτουργεί σωστά.

3. ***Γιατί χρειάζεται ο χειρισμός δύο σημάτων για την υλοποίηση του χρονοδρομολογητή;Θα μπορούσε ο χρονοδρομολογητής να χρησιμοποιεί μόνο το σήμα SIGALRM για να σταματά την τρέχουσα διεργασία και να ξεκινά την επόμενη; Τι ανεπιθύμητη συμπεριφορά θα μπορούσε να εμφανίζει μια τέτοια υλοποίηση;***

   Ο λόγος που χρησιμοποιούμε 2 διαφορετικά σήματα είναι αυτό που προαναφέραμε, ότι μπορεί να υπάρχουν καθυστερήσεις μεταξύ της αποστολής και λήψης των σημάτων. Για παράδειγμα, αν χρησιμοποιούσαμε μόνο handler για το SIGALRM θα ήταν πιθανό ένα SIGSTOP να σταλεί σε μια διεργασία και αμέσως μετά ένα SIGCONT, ωστόσο η 2η διεργασία να λάβει πρώτη το SIGCONT πριν καν σταματήσει η πρώτη, γεγονός που θα έκανε το χρονοδρομολογητή μας

ελαττωματικό , αφού θα ξεκινούσε έτσι η επόμενη διεργασία πριν σταματήσει η προηγούμενη της, κι έτσι τότε θα έτρεχαν δύο διεργασίες την ίδια στιγμή. Όμως τώρα με τους 2 handlers είμαστε σίγουροι ότι ο scheduler μας θα τρέχει σωστά αφου οταν έρθει σήμα SIGARLM στέλνουμε SIGSTOP στη διεργασία που εκτελείται και αναμένουμε να μας έρθει σήμα SIGCHLD (η επιβεβαίωση στην ουσία ότι σταμάτησε) από τη διεργασία και αφου μας έρθει ελέγχουμε τι της συνέβη (αν σταμάτησε επιτυχώς) και μετά από αυτή τη διαδικασία στέλνουμε SIGCONT στην επόμενη που έχει σειρά να ενεργοποιηθεί. Έτσι αποφεύγουμε όλες τις ανεπιθύμητες περιπτώσεις που μπορεί να προέκυπταν λόγω των καθυστερήσεων των σημάτων.

Άσκηση 1.2

Πηγαίος Κώδικας:

scheduler-shell.c

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

#include <sys/wait.h>
#include <sys/types.h>

#include "proc-common.h"
#include "request.h"
#include "list.h"

processT curr;
processListT procList;

int exceptions [2] = {0,0};
pid_t  exceptionsID [2];

/* Compile-time parameters. */
#define SCHED_TQ_SEC 2              /* time quantum */
#define TASK_NAME_SZ 60             /* maximum size for a task's name */
#define SHELL_EXECUTABLE_NAME "shell" /* executable for shell */

void executeProg(processT proc){

        char *newargv[] = { proc->execName, NULL, NULL, NULL };
        char *newenviron[] = { NULL };

        execve(proc->execName, newargv, newenviron);

        perror("Execve:");
        exit(-1);
}
```

```
/* Print a list of all tasks currently being scheduled.  */
static void
sched_print_tasks(void)
{
	processT temp = procList->head;

	while(temp != NULL) {
		if (temp == curr) {
			printf("Currently running: \n");
		}

		printf("process ID: %d, "
			"process PID: %ld, "
			"executable name: %s\n",
			temp->pNumber, (long)temp->pPid, temp->execName);

			temp = temp->next;
	}
}

/* Send SIGKILL to a task determined by the value of its
 * scheduler-specific id.
 */
static int
sched_kill_task_by_id(int id)
{
	int flag = 0;
	processT temp = findProcID(procList, id, &flag);
	if (flag) {

		kill(temp->pPid, SIGKILL);
		fprintf(stderr,"Proccess with pid=%ld died...\n", (long int)temp->pPid);

		exceptions[0] = 1;
		exceptionsID[0] = temp->pNumber;

		return 0;
	}
	else{
		fprintf(stderr,"No such Proccess");
		return -ENOSYS;
	}
}


/* Create a new task.  */
static void
```

```
sched_create_task(char *executable)
{
        processT temp;
        int id = findNewID(procList);
        pid_t p;

        temp = initProc(executable,id);
        addToList(procList,temp);

        p = fork();

        if(p < 0) {
                /* fork failed */
                perror("");
                exit(-1);
        }
        if(p == 0) {
                fprintf(stderr,"A new proccess is created with pid=%ld \n",(long int)getpid());
                raise(SIGSTOP);
                executeProg(temp);
        }
        if(p>0){
                temp->pPid = p;
                exceptions[1] = 1;
                exceptionsID[1] = temp->pNumber;

        }

}

/* Process requests by the shell.  */
static int
process_request(struct request_struct *rq)
{
        switch (rq->request_no) {
                case REQ_PRINT_TASKS:
                        sched_print_tasks();
                        return 0;

                case REQ_KILL_TASK:
                        return sched_kill_task_by_id(rq->task_arg);

                case REQ_EXEC_TASK:
                        sched_create_task(rq->exec_task_arg);
                        return 0;

                default:
```

```
                return -ENOSYS;
        }
}

/*
 * SIGALRM handler
 */
static void
sigalrm_handler(int signum)
{
        if (signum != SIGALRM) {
                fprintf(stderr, "Internal error: Called for signum %d, not SIGALRM\n",
                        signum);
                exit(1);
        }
        fprintf(stderr,"Alarm! -.- \n");
        kill(curr->pPid, SIGSTOP);
}

/*
 * SIGCHLD handler
 */
static void
sigchld_handler(int signum)
{
        pid_t p;
        int status;

        if (signum != SIGCHLD) {
                fprintf(stderr, "Internal error: Called for signum %d, not SIGCHLD\n",
                        signum);
                exit(1);
        }

        /*
         * Something has happened to one of the children.
         * We use waitpid() with the WUNTRACED flag, instead of wait(), because
         * SIGCHLD may have been received for a stopped, not dead child.
         *
         * A single SIGCHLD may be received if many processes die at the same time.
         * We use waitpid() with the WNOHANG flag in a loop, to make sure all
         * children are taken care of before leaving the handler.
         */
        for(;;) {
                p = waitpid(-1, &status, WUNTRACED | WNOHANG);
                if (p < 0) {
                        perror("waitpid");
```

```
                exit(1);
        }

        if (p == 0)
                break;

        if(procList->cnt > 1) explain_wait_status(p, status);

        if (WIFEXITED(status) || WIFSIGNALED(status)) {
                /* A child has died */

                /* The child died because we terminated it with SIGKILL */
                if (exceptions[0] == 1) {
                        processT temp;
                        exceptions[0] = 0;
                        int flag=0;
                        temp = findProcID(procList, exceptionsID[0],&flag);
                        if(flag) removeFromList(procList, temp);
                        if(procList->head == NULL) {
                                printf("All children finished. Exiting...\n");
                                exit(EXIT_SUCCESS);
                        }
                }

                /* other reasons */
                else {
                        processT temp = curr;
                        curr = curr->next;
                        fprintf(stderr,"Proccess with pid=%ld terminated...\n", (long int)temp-
>pPid);
                        removeFromList(procList, temp);

                        if(procList->head == NULL) {
                                printf("All children finished. Exiting...\n");
                                exit(EXIT_SUCCESS);
                        }
                        if(curr == NULL)
                                        curr = procList->head;

                        /* start alarm */
                        if (alarm(SCHED_TQ_SEC) < 0) {
                                perror("alarm");
                                exit(1);
                        }

                        fprintf(stderr,"Proccess with pid=%ld is about to begin...\n", (long
int)curr->pPid);
```

```
                        kill(curr->pPid, SIGCONT);
                }
        }
        if (WIFSTOPPED(status)) {
                /* A child has stopped due to SIGSTOP/SIGTSTP, etc... */
                curr = curr->next;
                if (curr == NULL)
                        curr = procList->head;

                /* start alarm */
                //      printf("%d\n", current->pNumber);
                        if (alarm(SCHED_TQ_SEC) < 0) {
                                perror("alarm");
                                exit(1);
                        }

                fprintf(stderr,"Proccess with pid=%ld is about to begin...\n", (long int)curr->pPid);
                kill(curr->pPid, SIGCONT);
        }
    }
}

/* Disable delivery of SIGALRM and SIGCHLD. */
static void
signals_disable(void)
{
    sigset_t sigset;

    sigemptyset(&sigset);
    sigaddset(&sigset, SIGALRM);
    sigaddset(&sigset, SIGCHLD);
    if (sigprocmask(SIG_BLOCK, &sigset, NULL) < 0) {
        perror("signals_disable: sigprocmask");
        exit(1);
    }
}

/* Enable delivery of SIGALRM and SIGCHLD.  */
static void
signals_enable(void)
{
    sigset_t sigset;

    sigemptyset(&sigset);
    sigaddset(&sigset, SIGALRM);
    sigaddset(&sigset, SIGCHLD);
```

```
	if (sigprocmask(SIG_UNBLOCK, &sigset, NULL) < 0) {
		perror("signals_enable: sigprocmask");
		exit(1);
	}
}


/* Install two signal handlers.
 * One for SIGCHLD, one for SIGALRM.
 * Make sure both signals are masked when one of them is running.
 */
static void
install_signal_handlers(void)
{
	sigset_t sigset;
	struct sigaction sa;

	sa.sa_handler = sigchld_handler;
	sa.sa_flags = SA_RESTART;
	sigemptyset(&sigset);
	sigaddset(&sigset, SIGCHLD);
	sigaddset(&sigset, SIGALRM);
	sa.sa_mask = sigset;
	if (sigaction(SIGCHLD, &sa, NULL) < 0) {
		perror("sigaction: sigchld");
		exit(1);
	}

	sa.sa_handler = sigalrm_handler;
	if (sigaction(SIGALRM, &sa, NULL) < 0) {
		perror("sigaction: sigalrm");
		exit(1);
	}

	/*
	 * Ignore SIGPIPE, so that write()s to pipes
	 * with no reader do not result in us being killed,
	 * and write() returns EPIPE instead.
	 */
	if (signal(SIGPIPE, SIG_IGN) < 0) {
		perror("signal: sigpipe");
		exit(1);
	}
}

static void
do_shell(char *executable, int wfd, int rfd)
```

```
{
        char arg1[10], arg2[10];
        char *newargv[] = { executable, NULL, NULL, NULL };
        char *newenviron[] = { NULL };

        sprintf(arg1, "%05d", wfd);
        sprintf(arg2, "%05d", rfd);
        newargv[1] = arg1;
        newargv[2] = arg2;

        raise(SIGSTOP);
        execve(executable, newargv, newenviron);

        /* execve() only returns on error */
        perror("scheduler: child: execve");
        exit(1);
}

/* Create a new shell task.
 *
 * The shell gets special treatment:
 * two pipes are created for communication and passed
 * as command-line arguments to the executable.
 */
static pid_t
sched_create_shell(char *executable, int *request_fd, int *return_fd)
{
        pid_t p;
        int pfds_rq[2], pfds_ret[2];

        if (pipe(pfds_rq) < 0 || pipe(pfds_ret) < 0) {
                perror("pipe");
                exit(1);
        }

        p = fork();
        if (p < 0) {
                perror("scheduler: fork");
                exit(1);
        }

        if (p == 0) {
                /* Child */
                close(pfds_rq[0]);
                close(pfds_ret[1]);
                do_shell(executable, pfds_rq[1], pfds_ret[0]);
                assert(0);
```

```
        }
        /* Parent */
        close(pfds_rq[1]);
        close(pfds_ret[0]);
        *request_fd = pfds_rq[0];
        *return_fd = pfds_ret[1];

        return p;
}

static void
shell_request_loop(int request_fd, int return_fd)
{
        int ret;
        struct request_struct rq;

        /*
         * Keep receiving requests from the shell.
         */
        for (;;) {
                if (read(request_fd, &rq, sizeof(rq)) != sizeof(rq)) {
                        perror("scheduler: read from shell");
                        fprintf(stderr, "Scheduler: giving up on shell request processing.\n");
                        break;
                }

                signals_disable();
                ret = process_request(&rq);
                signals_enable();

                if (write(return_fd, &ret, sizeof(ret)) != sizeof(ret)) {
                        perror("scheduler: write to shell");
                        fprintf(stderr, "Scheduler: giving up on shell request processing.\n");
                        break;
                }
        }
}

int main(int argc, char *argv[])
{
        int nproc;
        pid_t pid;
        int i;

        procList = initList();

        /* Two file descriptors for communication with the shell */
```

```
static int request_fd, return_fd;

/* Create the shell. */
pid = sched_create_shell(SHELL_EXECUTABLE_NAME, &request_fd, &return_fd);
/* TODO: add the shell to the scheduler's tasks */

curr = initProc(SHELL_EXECUTABLE_NAME, 0);
curr->pPid = pid;
addToList(procList, curr);

for(i=1; i<argc; i++){
        curr = initProc(argv[i], i);
        addToList(procList,curr);
}

/*
 * For each of argv[1] to argv[argc - 1],
 * create a new child process, add it to the process list.
 */

nproc = argc-1; /* number of proccesses goes here */

curr = procList->head->next;
for(i=0; i<nproc; i++){
        pid = fork();
        if(pid<0){
                perror("");
                exit(-1);
        }
        if(pid==0){
                fprintf(stderr,"A new proccess is created with pid=%ld \n",(long int)getpid());
                raise(SIGSTOP);
                executeProg(curr);
        }
        if(pid>0){
                curr->pPid = pid;
                curr = curr->next;
        }
}


/* Wait for all children to raise SIGSTOP before exec()ing. */
wait_for_ready_children(nproc);

/* Install SIGALRM and SIGCHLD handlers. */
install_signal_handlers();
```

```
        curr = procList->head;
        if (alarm(SCHED_TQ_SEC) < 0) {
                perror("alarm");
                exit(1);
        }

        fprintf(stderr,"Proccess with pid=%ld is about to begin...\n", (long int)curr->pPid);
        kill(curr->pPid,SIGCONT);

        shell_request_loop(request_fd, return_fd);

        /* Now that the shell is gone, just loop forever
         * until we exit from inside a signal handler.
         */
        while (pause())
                ;

        /* Unreachable */
        fprintf(stderr, "Internal error: Reached unreachable point\n");
        return 1;
}
```

Έξοδος εκτέλεσης προγράμματος:

Μια τυπική έξοδος του προγράμματος με κλήση προγράμματος :
$ ./schelduler-shell prog prog

A new proccess is created with pid=23449
My PID = 23447: Child PID = 23448 has been stopped by a signal, signo = 19
My PID = 23447: Child PID = 23449 has been stopped by a signal, signo = 19
A new proccess is created with pid=23450
Proccess with pid=23448 is about to begin...
My PID = 23447: Child PID = 23450 has been stopped by a signal, signo = 19
Proccess with pid=23449 is about to begin...

This is the Shell. Welcome.

Shell> prog: Starting, NMSG = 50, delay = 110
prog[23449]: This is message 0
prog[23449]: This is message 1
prog[23449]: This is message 2
prog[23449]: This is message 3
prog[23449]: This is message 4
prog[23449]: This is message 5
prog[23449]: This is message 6
Alarm! -.-
My PID = 23447: Child PID = 23449 has been stopped by a signal, signo = 19
Proccess with pid=23450 is about to begin...
prog: Starting, NMSG = 50, delay = 158
prog[23450]: This is message 0
prog[23450]: This is message 1
prog[23450]: This is message 2
prog[23450]: This is message 3
prog[23450]: This is message 4
Alarm! -.-
My PID = 23447: Child PID = 23450 has been stopped by a signal, signo = 19
Proccess with pid=23448 is about to begin...
e progAlarm! -.-
My PID = 23447: Child PID = 23448 has been stopped by a signal, signo = 19
Proccess with pid=23449 is about to begin...

prog[23449]: This is message 7
prog[23449]: This is message 8
prog[23449]: This is message 9
prog[23449]: This is message 10
prog[23449]: This is message 11
prog[23449]: This is message 12

Alarm! -.-
My PID = 23447: Child PID = 23449 has been stopped by a signal, signo = 19
Proccess with pid=23450 is about to begin...
prog[23450]: This is message 5
prog[23450]: This is message 6
pprog[23450]: This is message 7

prog[23450]: This is message 8
Alarm! -.-
My PID = 23447: Child PID = 23450 has been stopped by a signal, signo = 19
Proccess with pid=23448 is about to begin...
Shell: issuing request...
Shell: receiving request return value...
Shell> Shell: issuing request...
Shell: receiving request return value...
Currently running:
process ID: 0, process PID: 23448, executable name: shell
process ID: 1, process PID: 23449, executable name: prog
process ID: 2, process PID: 23450, executable name: prog
process ID: 3, process PID: 23451, executable name: prog
A new proccess is created with pid=23451
Shell> My PID = 23447: Child PID = 23451 has been stopped by a signal, signo = 19
Proccess with pid=23449 is about to begin...
prog[23449]: This is message 13
prog[23449]: This is message 14
prog[23449]: This is message 15
prog[23449]: This is message 16
prog[23449]: This is message 17
prog[23449]: This is message 18
Alarm! -.-
My PID = 23447: Child PID = 23449 has been stopped by a signal, signo = 19
Proccess with pid=23450 is about to begin...
prog[23450]: This is message 9
prog[23450]: This is message 10
prog[23450]: This is message 11
prog[23450]: This is message 12
Alarm! -.-
My PID = 23447: Child PID = 23450 has been stopped by a signal, signo = 19
Proccess with pid=23451 is about to begin...
prog: Starting, NMSG = 50, delay = 74
prog[23451]: This is message 0
prog[23451]: This is message 1
prog[23451]: This is message 2
prog[23451]: This is message 3
prog[23451]: This is message 4
prog[23451]: This is message 5
prog[23451]: This is message 6

prog[23451]: This is message 7
prog[23451]: This is message 8
Alarm! -.-
My PID = 23447: Child PID = 23451 has been stopped by a signal, signo = 19
Proccess with pid=23448 is about to begin...
Alarm! -.-
My PID = 23447: Child PID = 23448 has been stopped by a signal, signo = 19
Proccess with pid=23449 is about to begin...
prog[23449]: This is message 19
prog[23449]: This is message 20
prog[23449]: This is message 21
prog[23449]: This is message 22
prog[23449]: This is message 23
prog[23449]: This is message 24
Alarm! -.-
My PID = 23447: Child PID = 23449 has been stopped by a signal, signo = 19
Proccess with pid=23450 is about to begin...
prog[23450]: This is message 13
prog[23450]: This is message 14
prog[23450]: This is message 15
prog[23450]: This is message 16
Alarm! -.-
My PID = 23447: Child PID = 23450 has been stopped by a signal, signo = 19
Proccess with pid=23451 is about to begin...
prog[23451]: This is message 9
prog[23451]: This is message 10
kprog[23451]: This is message 11
prog[23451]: This is message 12
prog[23451]: This is message 13
prog[23451]: This is message 14
3prog[23451]: This is message 15
prog[23451]: This is message 16

prog[23451]: This is message 17
Alarm! -.-
My PID = 23447: Child PID = 23451 has been stopped by a signal, signo = 19
Proccess with pid=23448 is about to begin...
Shell: issuing request...
Shell: receiving request return value...
Proccess with pid=23451 died...
Shell> My PID = 23447: Child PID = 23451 was terminated by a signal, signo = 9
Alarm! -.-
My PID = 23447: Child PID = 23448 has been stopped by a signal, signo = 19
Proccess with pid=23449 is about to begin...
prog[23449]: This is message 25
prog[23449]: This is message 26
prog[23449]: This is message 27

prog[23449]: This is message 28
prog[23449]: This is message 29
prog[23449]: This is message 30
Alarm! -.-
My PID = 23447: Child PID = 23449 has been stopped by a signal, signo = 19
Proccess with pid=23450 is about to begin...
prog[23450]: This is message 17
prog[23450]: This is message 18
prog[23450]: This is message 19
prog[23450]: This is message 20
prog[23450]: This is message 21
Alarm! -.-
My PID = 23447: Child PID = 23450 has been stopped by a signal, signo = 19
Proccess with pid=23448 is about to begin...
p
Shell: issuing request...
Shell: receiving request return value...
Currently running:
process ID: 0, process PID: 23448, executable name: shell
process ID: 1, process PID: 23449, executable name: prog
process ID: 2, process PID: 23450, executable name: prog
Shell> Alarm! -.-
My PID = 23447: Child PID = 23448 has been stopped by a signal, signo = 19
Proccess with pid=23449 is about to begin...
prog[23449]: This is message 31
prog[23449]: This is message 32
prog[23449]: This is message 33
prog[23449]: This is message 34
prog[23449]: This is message 35
prog[23449]: This is message 36
kAlarm! -.-
My PID = 23447: Child PID = 23449 has been stopped by a signal, signo = 19
Proccess with pid=23450 is about to begin...
1prog[23450]: This is message 22

prog[23450]: This is message 23
prog[23450]: This is message 24
prog[23450]: This is message 25
Alarm! -.-
My PID = 23447: Child PID = 23450 has been stopped by a signal, signo = 19
Proccess with pid=23448 is about to begin...
Shell: issuing request...
Shell: receiving request return value...
Proccess with pid=23449 died...
Shell> My PID = 23447: Child PID = 23449 was terminated by a signal, signo = 9
p
Shell: issuing request...

Shell: receiving request return value...
Currently running:
process ID: 0, process PID: 23448, executable name: shell
process ID: 2, process PID: 23450, executable name: prog
Shell> Alarm! -.-
My PID = 23447: Child PID = 23448 has been stopped by a signal, signo = 19
Proccess with pid=23450 is about to begin...
prog[23450]: This is message 26
q
prog[23450]: This is message 27
prog[23450]: This is message 28
prog[23450]: This is message 29
Alarm! -.-
My PID = 23447: Child PID = 23450 has been stopped by a signal, signo = 19
Proccess with pid=23448 is about to begin...
Shell: Exiting. Goodbye.
My PID = 23447: Child PID = 23448 terminated normally, exit status = 0
Proccess with pid=23448 terminated...
Proccess with pid=23450 is about to begin...
scheduler: read from shell: Success
Scheduler: giving up on shell request processing.
prog[23450]: This is message 30
prog[23450]: This is message 31
prog[23450]: This is message 32
prog[23450]: This is message 33
Alarm! -.-
Proccess with pid=23450 is about to begin...
prog[23450]: This is message 34
prog[23450]: This is message 35
prog[23450]: This is message 36
prog[23450]: This is message 37
Alarm! -.-
Proccess with pid=23450 is about to begin...
prog[23450]: This is message 38
prog[23450]: This is message 39
prog[23450]: This is message 40
prog[23450]: This is message 41
prog[23450]: This is message 42
Alarm! -.-
Proccess with pid=23450 is about to begin...
prog[23450]: This is message 43
prog[23450]: This is message 44
prog[23450]: This is message 45
prog[23450]: This is message 46
Alarm! -.-
Proccess with pid=23450 is about to begin...
prog[23450]: This is message 47

prog[23450]: This is message 48
prog[23450]: This is message 49
Proccess with pid=23450 terminated...
All children finished. Exiting...

Ερωτήσεις:

1. **Όταν και ο φλοιός υφίσταται χρονοδρομολόγηση, ποια εμφανίζεται πάντοτε ως τρέχουσα διεργασία στη λίστα διεργασιών (εντολή 'p'); Θα μπορούσε να μη συμβαίνει αυτό; Γιατί;**

   Από την έξοδο του προγράμματος παρατηρείται ότι το process ID που ισούται με 0 είναι η τρέχουσα διεργασία, δηλαδή η διεργασία του φλοιού. Αυτό είναι λογικό, καθώς η εκτύπωση γίνεται μόνο όταν η τρεχουσα διεργασία είναι ο φλοιός. Οπότε δεν θα μπορούσε να μην συμβαίνει αυτό.

2. **Γιατί είναι αναγκαίο να συμπεριλάβετε κλήσεις signals_disable(), _enable() γύρω από την συνάρτηση υλοποίησης αιτήσεων του φλοιού;**

   Οι χρήσεις των συναρτήσεων χρησιμοποιούνται αντίστοιχα για την απενεργοποίηση και ενεργοποίηση των σημάτων (να μην γίνεται ή να γίνεται αντίστοιχα ο χειρισμός τους). Η χρήση τους είναι ώστε να περιορίσουμε τα σήματα κατά την διάρκεια που εξυπηρετόνται οι αιτήσεις στον φλοιό. Έτσι εξασφαλίζουμε ότι δε θα πειραχτούν οι δομές που χρησιμοποιούνται εκείνη τη στιγμή. Αν αφήναμε να γίνονται κανονικά οι χειρισμοί σημάτων, με άλλα λόγια, θα ήταν πιθανό να τροποποιηθεί η λίστα διεργασιών μας και να κατέληγε αυτό σε λάθος αποτέλεσμα για το χρονοδρομολογητή μας.

Άσκηση 1.3

Πηγαίος Κώδικας:

scheduler-priorities.c

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

#include <sys/wait.h>
#include <sys/types.h>

#include "proc-common.h"
#include "request.h"
#include "list.h"

/* Compile-time parameters. */
#define SCHED_TQ_SEC 2                /* time quantum */
#define TASK_NAME_SZ 60               /* maximum size for a task's name */
#define SHELL_EXECUTABLE_NAME "shell" /* executable for shell */

processListT procList;
processT curr;

int exceptions [2] = {0,0};
pid_t  exceptionsID [2];

void executeProg(processT proc){

        char *newargv[] = { proc->execName, NULL, NULL, NULL };
        char *newenviron[] = { NULL };

        execve(proc->execName, newargv, newenviron);

        perror("Execve:");
        exit(-1);
}
```

```c
/* Print a list of all tasks currently being scheduled.  */
static void
sched_print_tasks(void)
{
	processT temp = procList->head;

	while(temp != NULL) {
		if (temp == curr) {
			printf("Currently running: \n");
		}

		printf("process ID: %d, "
			"process PID: %ld, "
			"executable name: %s"
			"priority: %d\n",
			temp->pNumber, (long)temp->pPid, temp->execName, temp->priority);

			temp = temp->next;
	}
}

/* Send SIGKILL to a task determined by the value of its
 * scheduler-specific id.
 */
static int
sched_kill_task_by_id(int id)
{
	int flag = 0;
	processT temp = findProcID(procList, id, &flag);
	if (flag) {
		kill(temp->pPid, SIGKILL);
		fprintf(stderr,"Proccess with pid=%ld died...\n", (long int)temp->pPid);

		exceptions[0] = 1;
		exceptionsID[0] = temp->pNumber;

		return 0;
	}
	else{
		fprintf(stderr,"No such Proccess");
		return -ENOSYS;
	}
}


/* Create a new task.  */
static void
```

```
sched_create_task(char *executable)
{
	processT temp;
	int id = findNewID(procList);
	pid_t p;

	temp = initProc(executable,id);
	addToList(procList,temp);

	p = fork();

	if(p < 0) {
		/* fork failed */
		perror("");
		exit(-1);
	}
	if(p == 0) {
		fprintf(stderr,"A new proccess is created with pid=%ld",(long int)temp->pPid);
		raise(SIGSTOP);
		executeProg(temp);
	}
	if(p>0){
		temp->pPid = p;
		exceptions[1] = 1;
		exceptionsID[1] = temp->pNumber;

	}
}

static int
sched_increase_priority(int id)
{
	processT temp = procList->head;
	int flag = 0;
	temp=findProcID(procList, id, &flag);
	if(flag) {temp->priority = HIGH; fprintf(stderr,"The Process with id=%d and pid=%ld is
HIGH!", temp->pNumber, (long int)temp->pPid);}
	else fprintf(stderr,"There is no such process!");
	return id;
}

static int
sched_decrease_priority(int id)
{
	processT temp = procList->head;
	int flag = 0;
	temp=findProcID(procList, id, &flag);
```

```
        if(flag) {temp->priority = LOW; fprintf(stderr,"The Process with id=%d and pid=%ld is
LOW!", temp->pNumber, (long int)temp->pPid);}
        else fprintf(stderr,"There is no such process!");
        return id;
}

/* Process requests by the shell.  */
static int
process_request(struct request_struct *rq)
{
        switch (rq->request_no) {
                case REQ_PRINT_TASKS:
                        sched_print_tasks();
                        return 0;

                case REQ_KILL_TASK:
                        return sched_kill_task_by_id(rq->task_arg);

                case REQ_EXEC_TASK:
                        sched_create_task(rq->exec_task_arg);
                        return 0;

                case REQ_HIGH_TASK:
                        return sched_increase_priority(rq->task_arg);

                case REQ_LOW_TASK:
                        return sched_decrease_priority(rq->task_arg);

                default:
                        return -ENOSYS;
        }
}

/*
 * SIGALRM handler
 */
static void
sigalrm_handler(int signum)
{
        if (signum != SIGALRM) {
                fprintf(stderr, "Internal error: Called for signum %d, not SIGALRM\n",
                        signum);
                exit(1);
        }

        printf("Alarm! -.- \n");
        kill(curr->pPid, SIGSTOP);
```

```
}

/*
 * SIGCHLD handler
 */
static void
sigchld_handler(int signum)
{
	pid_t p;
	int status;

	if (signum != SIGCHLD) {
		fprintf(stderr, "Internal error: Called for signum %d, not SIGCHLD\n",
			signum);
		exit(1);
	}

	/*
	 * Something has happened to one of the children.
	 * We use waitpid() with the WUNTRACED flag, instead of wait(), because
	 * SIGCHLD may have been received for a stopped, not dead child.
	 *
	 * A single SIGCHLD may be received if many processes die at the same time.
	 * We use waitpid() with the WNOHANG flag in a loop, to make sure all
	 * children are taken care of before leaving the handler.
	 */
	for(;;) {
		p = waitpid(-1, &status, WUNTRACED | WNOHANG);
		if (p < 0) {
			perror("waitpid");
			exit(1);
		}

		if (p == 0)
			break;

		if(procList->cnt > 1) explain_wait_status(p, status);

		if (WIFEXITED(status) || WIFSIGNALED(status)) {
			/* A child has died */

			/* The child died because we terminated it with SIGKILL */
			if (exceptions[0] == 1) {
				processT temp;
				exceptions[0] = 0;
				int flag=0;
				temp = findProcID(procList, exceptionsID[0],&flag);
```

```
                        if(flag) {removeFromList(procList, temp);}
                        else fprintf(stderr, "No such proccess!");


                        if(procList->head == NULL) {
                                printf("All children finished. Exiting...\n");
                                exit(EXIT_SUCCESS);
                        }
                }

                /* other reasons */
                else {
                        processT temp = curr;
                        curr = curr->next;
                        fprintf(stderr,"Proccess with pid=%ld terminated...\n", (long int)temp-
>pPid);
                        removeFromList(procList, temp);


                        if(procList->head == NULL) {
                                printf("All children finished. Exiting...\n");
                                exit(EXIT_SUCCESS);
                        }
                        if(curr == NULL)
                                        curr = procList->head;

                        temp = curr;
                        if(procList->head != NULL){
                                while(temp->priority != HIGH){
                                        if(temp->next == curr || (temp->next == NULL && curr
== procList->head)) { temp = curr; break;}
                                        if(temp->next == NULL) {temp = procList->head;
continue;}
                                        temp = temp->next;
                                }
                        }

                        curr = temp;

                        /* start alarm */
                        if (alarm(SCHED_TQ_SEC) < 0) {
                                perror("alarm");
                                exit(1);
                        }

                        kill(curr->pPid, SIGCONT);
                }
```

```
                }
                if (WIFSTOPPED(status)) {
                        /* A child has stopped due to SIGSTOP/SIGTSTP, etc... */
                        curr = curr->next;
                        if (curr == NULL)
                                curr = procList->head;

                        processT temp = curr;
                        if(procList->head != NULL){
                                while(temp->priority != HIGH){
                                        if(temp->next == curr || (temp->next == NULL && curr
== procList->head)) { temp = curr; break;}
                                        if(temp->next == NULL) {temp = procList->head;
continue;}
                                        temp = temp->next;
                                }
                        }

                        curr = temp;

                                /* start alarm */
                        //      printf("%d\n", current->pNumber);
                                if (alarm(SCHED_TQ_SEC) < 0) {
                                        perror("alarm");
                                        exit(1);
                                }
                        kill(curr->pPid, SIGCONT);
                }
        }
}

/* Disable delivery of SIGALRM and SIGCHLD. */
static void
signals_disable(void)
{
        sigset_t sigset;

        sigemptyset(&sigset);
        sigaddset(&sigset, SIGALRM);
        sigaddset(&sigset, SIGCHLD);
        if (sigprocmask(SIG_BLOCK, &sigset, NULL) < 0) {
                perror("signals_disable: sigprocmask");
                exit(1);
        }
}

/* Enable delivery of SIGALRM and SIGCHLD.  */
```

```
static void
signals_enable(void)
{
	sigset_t sigset;

	sigemptyset(&sigset);
	sigaddset(&sigset, SIGALRM);
	sigaddset(&sigset, SIGCHLD);
	if (sigprocmask(SIG_UNBLOCK, &sigset, NULL) < 0) {
		perror("signals_enable: sigprocmask");
		exit(1);
	}
}


/* Install two signal handlers.
 * One for SIGCHLD, one for SIGALRM.
 * Make sure both signals are masked when one of them is running.
 */
static void
install_signal_handlers(void)
{
	sigset_t sigset;
	struct sigaction sa;

	sa.sa_handler = sigchld_handler;
	sa.sa_flags = SA_RESTART;
	sigemptyset(&sigset);
	sigaddset(&sigset, SIGCHLD);
	sigaddset(&sigset, SIGALRM);
	sa.sa_mask = sigset;
	if (sigaction(SIGCHLD, &sa, NULL) < 0) {
		perror("sigaction: sigchld");
		exit(1);
	}

	sa.sa_handler = sigalrm_handler;
	if (sigaction(SIGALRM, &sa, NULL) < 0) {
		perror("sigaction: sigalrm");
		exit(1);
	}

	/*
	 * Ignore SIGPIPE, so that write()s to pipes
	 * with no reader do not result in us being killed,
	 * and write() returns EPIPE instead.
	 */
```

```
	if (signal(SIGPIPE, SIG_IGN) < 0) {
		perror("signal: sigpipe");
		exit(1);
	}
}

static void
do_shell(char *executable, int wfd, int rfd)
{
	char arg1[10], arg2[10];
	char *newargv[] = { executable, NULL, NULL, NULL };
	char *newenviron[] = { NULL };

	sprintf(arg1, "%05d", wfd);
	sprintf(arg2, "%05d", rfd);
	newargv[1] = arg1;
	newargv[2] = arg2;

	raise(SIGSTOP);
	execve(executable, newargv, newenviron);

	/* execve() only returns on error */
	perror("scheduler: child: execve");
	exit(1);
}

/* Create a new shell task.
 *
 * The shell gets special treatment:
 * two pipes are created for communication and passed
 * as command-line arguments to the executable.
 */
static pid_t
sched_create_shell(char *executable, int *request_fd, int *return_fd)
{
	pid_t p;
	int pfds_rq[2], pfds_ret[2];

	if (pipe(pfds_rq) < 0 || pipe(pfds_ret) < 0) {
		perror("pipe");
		exit(1);
	}

	p = fork();
	if (p < 0) {
		perror("scheduler: fork");
		exit(1);
```

```
	}

	if (p == 0) {
		/* Child */
		close(pfds_rq[0]);
		close(pfds_ret[1]);
		do_shell(executable, pfds_rq[1], pfds_ret[0]);
		assert(0);
	}
	/* Parent */
	close(pfds_rq[1]);
	close(pfds_ret[0]);
	*request_fd = pfds_rq[0];
	*return_fd = pfds_ret[1];

	return p;
}

static void
shell_request_loop(int request_fd, int return_fd)
{
	int ret;
	struct request_struct rq;

	/*
	 * Keep receiving requests from the shell.
	 */
	for (;;) {
		if (read(request_fd, &rq, sizeof(rq)) != sizeof(rq)) {
			perror("scheduler: read from shell");
			fprintf(stderr, "Scheduler: giving up on shell request processing.\n");
			break;
		}

		signals_disable();
		ret = process_request(&rq);
		signals_enable();

		if (write(return_fd, &ret, sizeof(ret)) != sizeof(ret)) {
			perror("scheduler: write to shell");
			fprintf(stderr, "Scheduler: giving up on shell request processing.\n");
			break;
		}
	}
}

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

```
{
        int nproc;
        pid_t pid;
        int i;

        procList = initList();

        /* Two file descriptors for communication with the shell */
        static int request_fd, return_fd;

        /* Create the shell. */
        pid = sched_create_shell(SHELL_EXECUTABLE_NAME, &request_fd, &return_fd);

        /* TODO: add the shell to the scheduler's tasks */
        curr = initProc(SHELL_EXECUTABLE_NAME, 0);
        curr->pPid = pid;
        addToList(procList, curr);

        for(i=1; i<argc; i++){
                curr = initProc(argv[i], i);
                addToList(procList,curr);
        }

        /*
         * For each of argv[1] to argv[argc - 1],
         * create a new child process, add it to the process list.
         */

        nproc = argc-1; /* number of proccesses goes here */

        curr = procList->head->next;
        for(i=0; i<nproc; i++){
                pid = fork();
                if(pid<0){
                        perror("");
                        exit(-1);
                }
                if(pid==0){
                        fprintf(stderr,"A new proccess is created with pid=%ld \n",(long int)getpid());
                        raise(SIGSTOP);
                        executeProg(curr);
                }
                if(pid>0){
                        curr->pPid = pid;
                        curr = curr->next;
                }
        }
```

```
        /* Wait for all children to raise SIGSTOP before exec()ing. */
        wait_for_ready_children(nproc);

        /* Install SIGALRM and SIGCHLD handlers. */
        install_signal_handlers();

        curr = procList->head;
        if (alarm(SCHED_TQ_SEC) < 0) {
                perror("alarm");
                exit(1);
        }

        kill(curr->pPid,SIGCONT);

        shell_request_loop(request_fd, return_fd);

        /* Now that the shell is gone, just loop forever
         * until we exit from inside a signal handler.
         */
        while (pause())
                ;

        /* Unreachable */
        fprintf(stderr, "Internal error: Reached unreachable point\n");
        return 1;
}
```

Έξοδος εκτέλεσης προγράμματος:

Μια τυπική έξοδος του προγράμματος με κλήση προγράμματος :
$ ./schelduler-priorities prog prog

A new proccess is created with pid=23542
My PID = 23540: Child PID = 23541 has been stopped by a signal, signo = 19
My PID = 23540: Child PID = 23542 has been stopped by a signal, signo = 19
A new proccess is created with pid=23543
My PID = 23540: Child PID = 23543 has been stopped by a signal, signo = 19

This is the Shell. Welcome.

Shell> prog: Starting, NMSG = 50, delay = 89
prog[23542]: This is message 0
prog[23542]: This is message 1
prog[23542]: This is message 2
prog[23542]: This is message 3
prog[23542]: This is message 4
prog[23542]: This is message 5
prog[23542]: This is message 6
prog[23542]: This is message 7
hAlarm! -.-
My PID = 23540: Child PID = 23542 has been stopped by a signal, signo = 19
prog: Starting, NMSG = 50, delay = 70
prog[23543]: This is message 0
prog[23543]: This is message 1
prog[23543]: This is message 2
2prog[23543]: This is message 3
prog[23543]: This is message 4
prog[23543]: This is message 5

Shell: issuing request...
Shell: receiving request return value...
The Process with id=2 and pid=23543 is HIGH!
Shell> prog[23543]: This is message 6
prog[23543]: This is message 7
prog[23543]: This is message 8
prog[23543]: This is message 9
Alarm! -.-
My PID = 23540: Child PID = 23543 has been stopped by a signal, signo = 19
prog[23543]: This is message 10
prog[23543]: This is message 11
prog[23543]: This is message 12
prog[23543]: This is message 13
prog[23543]: This is message 14

prog[23543]: This is message 15
prog[23543]: This is message 16
prog[23543]: This is message 17
prog[23543]: This is message 18
prog[23543]: This is message 19
Alarm! -.-
My PID = 23540: Child PID = 23543 has been stopped by a signal, signo = 19
prog[23543]: This is message 20
prog[23543]: This is message 21
pprog[23543]: This is message 22

Shell: issuing request...
Shell: receiving request return value...
process ID: 0, process PID: 23541, executable name: shellpriority: 0
process ID: 1, process PID: 23542, executable name: progpriority: 0
Currently running:
process ID: 2, process PID: 23543, executable name: progpriority: 1
Shell> prog[23543]: This is message 23
prog[23543]: This is message 24
prog[23543]: This is message 25
prog[23543]: This is message 26
prog[23543]: This is message 27
prog[23543]: This is message 28
Alarm! -.-
My PID = 23540: Child PID = 23543 has been stopped by a signal, signo = 19
prog[23543]: This is message 29
prog[23543]: This is message 30
prog[23543]: This is message 31
prog[23543]: This is message 32
prog[23543]: This is message 33
prog[23543]: This is message 34
prog[23543]: This is message 35
prog[23543]: This is message 36
prog[23543]: This is message 37
prog[23543]: This is message 38
Alarm! -.-
My PID = 23540: Child PID = 23543 has been stopped by a signal, signo = 19
prog[23543]: This is message 39
prog[23543]: This is message 40
prog[23543]: This is message 41
prog[23543]: This is message 42
prog[23543]: This is message 43
prog[23543]: This is message 44
prog[23543]: This is message 45
prog[23543]: This is message 46
prog[23543]: This is message 47
Alarm! -.-

My PID = 23540: Child PID = 23543 has been stopped by a signal, signo = 19
prog[23543]: This is message 48
prog[23543]: This is message 49
My PID = 23540: Child PID = 23543 terminated normally, exit status = 0
Proccess with pid=23543 terminated...
eAlarm! -.-
My PID = 23540: Child PID = 23541 has been stopped by a signal, signo = 19
prog[23542]: This is message 8
prog[23542]: This is message 9
prog[23542]: This is message 10
prog[23542]: This is message 11
pprog[23542]: This is message 12
rprog[23542]: This is message 13
oprog[23542]: This is message 14
gAlarm! -.-
My PID = 23540: Child PID = 23542 has been stopped by a signal, signo = 19

Shell: issuing request...
Shell: receiving request return value...
Shell> A new proccess is created with pid=-1My PID = 23540: Child PID = 23544 has been stopped by a signal, signo = 19
prog[23542]: This is message 15
prog[23542]: This is message 16
prog[23542]: This is message 17
prog[23542]: This is message 18
prog[23542]: This is message 19
prog[23542]: This is message 20
hprog[23542]: This is message 21
prog[23542]: This is message 22
Alarm! -.-
My PID = 23540: Child PID = 23542 has been stopped by a signal, signo = 19
Execve:: No such file or directory
My PID = 23540: Child PID = 23544 terminated normally, exit status = 255
Proccess with pid=23544 terminated...
0
Shell: issuing request...
Shell: receiving request return value...
The Process with id=0 and pid=23541 is HIGH!
Shell> Alarm! -.-
My PID = 23540: Child PID = 23541 has been stopped by a signal, signo = 19
Alarm! -.-
My PID = 23540: Child PID = 23541 has been stopped by a signal, signo = 19
Alarm! -.-
My PID = 23540: Child PID = 23541 has been stopped by a signal, signo = 19
Alarm! -.-
My PID = 23540: Child PID = 23541 has been stopped by a signal, signo = 19
p

Shell: issuing request...
Shell: receiving request return value...
Currently running:
process ID: 0, process PID: 23541, executable name: shellpriority: 1
process ID: 1, process PID: 23542, executable name: progpriority: 0
Shell> Alarm! -.-
My PID = 23540: Child PID = 23541 has been stopped by a signal, signo = 19
Alarm! -.-
My PID = 23540: Child PID = 23541 has been stopped by a signal, signo = 19
Alarm! -.-
My PID = 23540: Child PID = 23541 has been stopped by a signal, signo = 19
Alarm! -.-
My PID = 23540: Child PID = 23541 has been stopped by a signal, signo = 19
Alarm! -.-
My PID = 23540: Child PID = 23541 has been stopped by a signal, signo = 19
kAlarm! -.-
My PID = 23540: Child PID = 23541 has been stopped by a signal, signo = 19
1
Shell: issuing request...
Shell: receiving request return value...
Proccess with pid=23542 died...
Shell> My PID = 23540: Child PID = 23542 was terminated by a signal, signo = 9
Alarm! -.-
p
Shell: issuing request...
Shell: receiving request return value...
Currently running:
process ID: 0, process PID: 23541, executable name: shellpriority: 1
Shell> Alarm! -.-
q
Shell: Exiting. Goodbye.
Proccess with pid=23541 terminated...
All children finished. Exiting…

Ερωτήσεις:

1. **Περιγράψτε ένα σενάριο δημιουργίας λιμοκτονίας.**

   Ένα πιθανό σενάριο λιμοκτονίας για το δρομολογητή είναι το γεγονός πως αν έχουμε μια διαδικασία που είναι πάντα low και δεν αλλάξει ποτέ η προτεραιότητα, ενώ πάντα υπάρχουν διεργασίες με προτεραιότητα high. Σε αυτή την περίπτωση η διεργασία δεν πρόκειται να εκτελεστεί ποτέ, αφού θα εκτελούνται οι άλλες διεργασίες που έχουν ψηλη προτεραιότητα. Αυτό πολλές φορές δεν είναι επιθυμητό. Για να το αποτρέψουμε αυτό μπορούμε να προσθέσουμε στους κόμβους τις λίστας ένα πεδίο που θα ορίζει μια τιμή “ηλικίας” τις διεργασίας και μια τιμή “γηραιότητας”. Κάθε φορά που να στέλνεται alarm να αυξάνεται η “ηλικία” και μόλις ξεπεράσει την τιμή της “γηταιότητας” να εκτελείται η διεργασία. Κάπως έτσι μπορεί να αποφευχθει το σενάριο λιμοκτονίας.