Λειτουργικά Συστήματα

Άσκηση 4: Χρονοδρομολόγηση

Ομάδα: oslabd43
Ο/Ε: Αντώνης Παπαοικονόμου 03115140
Γιάννης Πιτόσκας 03115077

1.1 Υλοποίηση χρονοδρομολογητή κυκλικής επαναφοράς στο χώρο χρήστη

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 "queue.h"

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

Queue q;

typedef struct PCB {
        pid_t pid;
        int id;
        char* exec;

}PCB;

PCB current;

/*
 * SIGALRM handler
 */
static void
sigalrm_handler(int signum)
{
        // assert(0 && "Please fill me!");

        if (kill(current.pid, SIGSTOP) < 0) {
                perror("kill");
                exit(1);
        }
}

/*
 * SIGCHLD handler
 */
static void
sigchld_handler(int signum)
{
        // assert(0 && "Please fill me!");

        pid_t p;
        int status;

        for (;;) {
```

```
                p = waitpid(-1, &status, WUNTRACED | WNOHANG);

                if (p < 0) {
                        perror("waitpid");
                        exit(1);
                }
                if (p == 0)
                        break;

                explain_wait_status(p, status);

                /* A child has died */
                if (WIFEXITED(status) || WIFSIGNALED(status)){
                        PCB temp;
                        /* Existence is pain for Meeseeks... */
                        printf("Mr Meeseeks %d: Stops existing! *POOF!* \n", current.id);
                        if (getQueueSize(&q) == 0){
                                fprintf(stderr, "Rick (Scheduler): All your tasks seem to be complete, now give me
back the Meeseeks Box. Exiting...\n");
                                exit(42);
                        }
                        dequeue(&q, &temp);
                        current = temp;
                }
                /* A child has stopped due to SIGSTOP/SIGTSTP, etc */
                if (WIFSTOPPED(status)){
                        PCB temp;
                        printf("Mr Meeseeks %d: I need more time to fullfill the request!\n", current.id);
                        enqueue(&q, &current);
                        dequeue(&q, &temp);
                        current = temp;
                }
                if (kill(current.pid, SIGCONT) < 0) {
                        perror("kill");
                        exit(1);
                }
                else{
                        printf("Mr Meeseeks %d: Back to work again!\n", current.id);
                        alarm(SCHED_TQ_SEC);
                }
        }
}

/* 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);
        }
}

int main(int argc, char *argv[])
{
        pid_t pid;
        int nproc = argc - 1; /* number of proccesses goes here */
        char *newenviron[] = { NULL };

        queueInit(&q, sizeof(PCB));

        if (nproc == 0) {
                fprintf(stderr, "Rick (Scheduler): You don't seem to have any requests, why did you ask for a Meeseeks
Box? Exiting...\n");
                exit(2);
        }
        else {
                fprintf(stderr, "Rick (Scheduler): This is a Meeseeks Box, let me show you how it works. You press
this...\n");
        }
        /*
         * For each of argv[1] to argv[argc - 1],
         * create a new child process, add it to the process list.
         */

        int i;
        for (i = 1; i <= nproc; i++) {

                char *newargv[] = { argv[i], NULL };
                PCB camtono;
                pid = fork();

                if (pid > 0){
                        printf("HEY, I'm Mr Meeseeks %d LOOK AT ME and my Meeseeks PID is %ld. \n", i, (long)pid);
                        printf("Rick (Scheduler): Mr Meeseeks %d, run process named %s. \n", i, argv[i]);
                        printf("Mr Meeseeks %d: YES SIR YI. \n", i);
                        camtono.pid = pid;
                        camtono.id = i;
                        camtono.exec = argv[i];
                        enqueue(&q, &camtono);

                }

                if (pid == 0) {
                        raise(SIGSTOP);
                        execve(argv[i], newargv, newenviron);
                        exit(0);
                }

                if (pid < 0) {
                        perror("Fork");
                }
    }

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

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

        dequeue(&q, &current);
        printf("First Process PID: %ld, Meeseeks ID: %d\n", (long)current.pid, current.id);
        alarm(SCHED_TQ_SEC);
        kill(current.pid, SIGCONT);

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

        /* Unreachable */
        fprintf(stderr, "HEY Morty, what are you doing here?! You are not supposed to be down here!\n");
        return 1;
```

```
}
```

Έξοδος για 4 διεργασίες prog:

```
oslabd43@orion:~/Ask4$ ./scheduler prog prog prog prog
Rick (Scheduler): This is a Meeseeks Box, let me show you how it works. You press this...
HEY, I'm Mr Meeseeks 1 LOOK AT ME and my Meeseeks PID is 31362.
Rick (Scheduler): Mr Meeseeks 1, run process named prog.
Mr Meeseeks 1: YES SIR YI.
HEY, I'm Mr Meeseeks 2 LOOK AT ME and my Meeseeks PID is 31363.
Rick (Scheduler): Mr Meeseeks 2, run process named prog.
Mr Meeseeks 2: YES SIR YI.
HEY, I'm Mr Meeseeks 3 LOOK AT ME and my Meeseeks PID is 31364.
Rick (Scheduler): Mr Meeseeks 3, run process named prog.
Mr Meeseeks 3: YES SIR YI.
HEY, I'm Mr Meeseeks 4 LOOK AT ME and my Meeseeks PID is 31365.
Rick (Scheduler): Mr Meeseeks 4, run process named prog.
Mr Meeseeks 4: YES SIR YI.
My PID = 31361: Child PID = 31365 has been stopped by a signal, signo = 19
My PID = 31361: Child PID = 31364 has been stopped by a signal, signo = 19
My PID = 31361: Child PID = 31362 has been stopped by a signal, signo = 19
My PID = 31361: Child PID = 31363 has been stopped by a signal, signo = 19
First Process PID: 31362, Meeseeks ID: 1
prog: Starting, NMSG = 20, delay = 93
prog[31362]: This is message 0
prog[31362]: This is message 1
prog[31362]: This is message 2
prog[31362]: This is message 3
My PID = 31361: Child PID = 31362 has been stopped by a signal, signo = 19
Mr Meeseeks 1: I need more time to fullfill the request!
Mr Meeseeks 2: Back to work again!
prog: Starting, NMSG = 20, delay = 139
prog[31363]: This is message 0
prog[31363]: This is message 1
prog[31363]: This is message 2
My PID = 31361: Child PID = 31363 has been stopped by a signal, signo = 19
Mr Meeseeks 2: I need more time to fullfill the request!
Mr Meeseeks 3: Back to work again!
prog: Starting, NMSG = 20, delay = 56
prog[31364]: This is message 0
prog[31364]: This is message 1
prog[31364]: This is message 2
prog[31364]: This is message 3
prog[31364]: This is message 4
prog[31364]: This is message 5
prog[31364]: This is message 6
prog[31364]: This is message 7
My PID = 31361: Child PID = 31364 has been stopped by a signal, signo = 19
Mr Meeseeks 3: I need more time to fullfill the request!
Mr Meeseeks 4: Back to work again!
prog: Starting, NMSG = 20, delay = 102
prog[31365]: This is message 0
prog[31365]: This is message 1
prog[31365]: This is message 2
prog[31365]: This is message 3
My PID = 31361: Child PID = 31365 has been stopped by a signal, signo = 19
Mr Meeseeks 4: I need more time to fullfill the request!
Mr Meeseeks 1: Back to work again!
prog[31362]: This is message 4
prog[31362]: This is message 5
prog[31362]: This is message 6
prog[31362]: This is message 7
prog[31362]: This is message 8
My PID = 31361: Child PID = 31362 has been stopped by a signal, signo = 19
Mr Meeseeks 1: I need more time to fullfill the request!
Mr Meeseeks 2: Back to work again!
prog[31363]: This is message 3
prog[31363]: This is message 4
prog[31363]: This is message 5
My PID = 31361: Child PID = 31363 has been stopped by a signal, signo = 19
Mr Meeseeks 2: I need more time to fullfill the request!
Mr Meeseeks 3: Back to work again!
prog[31364]: This is message 8
prog[31364]: This is message 9
```

```
prog[31364]: This is message 10
prog[31364]: This is message 11
prog[31364]: This is message 12
prog[31364]: This is message 13
prog[31364]: This is message 14
My PID = 31361: Child PID = 31364 has been stopped by a signal, signo = 19
Mr Meeseeks 3: I need more time to fullfill the request!
Mr Meeseeks 4: Back to work again!
prog[31365]: This is message 4
prog[31365]: This is message 5
prog[31365]: This is message 6
My PID = 31361: Child PID = 31365 has been stopped by a signal, signo = 19
Mr Meeseeks 4: I need more time to fullfill the request!
Mr Meeseeks 1: Back to work again!
prog[31362]: This is message 9
prog[31362]: This is message 10
prog[31362]: This is message 11
prog[31362]: This is message 12
My PID = 31361: Child PID = 31362 has been stopped by a signal, signo = 19
Mr Meeseeks 1: I need more time to fullfill the request!
Mr Meeseeks 2: Back to work again!
prog[31363]: This is message 6
prog[31363]: This is message 7
My PID = 31361: Child PID = 31363 has been stopped by a signal, signo = 19
Mr Meeseeks 2: I need more time to fullfill the request!
Mr Meeseeks 3: Back to work again!
prog[31364]: This is message 15
prog[31364]: This is message 16
prog[31364]: This is message 17
prog[31364]: This is message 18
prog[31364]: This is message 19
My PID = 31361: Child PID = 31364 terminated normally, exit status = 0
Mr Meeseeks 3: Stops existing! *POOF!*
Mr Meeseeks 4: Back to work again!
prog[31365]: This is message 7
prog[31365]: This is message 8
prog[31365]: This is message 9
prog[31365]: This is message 10
My PID = 31361: Child PID = 31365 has been stopped by a signal, signo = 19
Mr Meeseeks 4: I need more time to fullfill the request!
Mr Meeseeks 1: Back to work again!
prog[31362]: This is message 13
prog[31362]: This is message 14
prog[31362]: This is message 15
prog[31362]: This is message 16
My PID = 31361: Child PID = 31362 has been stopped by a signal, signo = 19
Mr Meeseeks 1: I need more time to fullfill the request!
Mr Meeseeks 2: Back to work again!
prog[31363]: This is message 8
prog[31363]: This is message 9
prog[31363]: This is message 10
My PID = 31361: Child PID = 31363 has been stopped by a signal, signo = 19
Mr Meeseeks 2: I need more time to fullfill the request!
Mr Meeseeks 4: Back to work again!
prog[31365]: This is message 11
prog[31365]: This is message 12
prog[31365]: This is message 13
prog[31365]: This is message 14
My PID = 31361: Child PID = 31365 has been stopped by a signal, signo = 19
Mr Meeseeks 4: I need more time to fullfill the request!
Mr Meeseeks 1: Back to work again!
prog[31362]: This is message 17
prog[31362]: This is message 18
prog[31362]: This is message 19
My PID = 31361: Child PID = 31362 terminated normally, exit status = 0
Mr Meeseeks 1: Stops existing! *POOF!*
Mr Meeseeks 2: Back to work again!
prog[31363]: This is message 11
prog[31363]: This is message 12
prog[31363]: This is message 13
My PID = 31361: Child PID = 31363 has been stopped by a signal, signo = 19
Mr Meeseeks 2: I need more time to fullfill the request!
Mr Meeseeks 4: Back to work again!
prog[31365]: This is message 15
prog[31365]: This is message 16
prog[31365]: This is message 17
```

```
My PID = 31361: Child PID = 31365 has been stopped by a signal, signo = 19
Mr Meeseeks 4: I need more time to fullfill the request!
Mr Meeseeks 2: Back to work again!
prog[31363]: This is message 14
prog[31363]: This is message 15
My PID = 31361: Child PID = 31363 has been stopped by a signal, signo = 19
Mr Meeseeks 2: I need more time to fullfill the request!
Mr Meeseeks 4: Back to work again!
prog[31365]: This is message 18
prog[31365]: This is message 19
My PID = 31361: Child PID = 31365 terminated normally, exit status = 0
Mr Meeseeks 4: Stops existing! *POOF!*
Mr Meeseeks 2: Back to work again!
prog[31363]: This is message 16
prog[31363]: This is message 17
prog[31363]: This is message 18
My PID = 31361: Child PID = 31363 has been stopped by a signal, signo = 19
Mr Meeseeks 2: I need more time to fullfill the request!
Mr Meeseeks 2: Back to work again!
prog[31363]: This is message 19
My PID = 31361: Child PID = 31363 terminated normally, exit status = 0
Mr Meeseeks 2: Stops existing! *POOF!*
Rick (Scheduler): All your tasks seem to be complete, now give me back the Meeseeks Box. Exiting...
```

Ερωτήσεις 3.1:

1. Τι συμβαίνει αν το σήμα SIGALRM έρθει ενώ εκτελείται η συνάρτηση χειρισμού του σήματος SIGCHLD ή το αντίστροφο; Πώς αντιμετωπίζει ένας πραγματικός χρονοδρομολογητής χώρου πυρήνα ανάλογα ενδεχόμενα και πώς η δική σας υλοποίηση;

Ουσιαστικά, μέσα στην `install_signal_handlers` δημιουργείται μια μάσκα για τα σήματα SIGSTOP, SIGALRM. Με αυτόν τον τρόπο λοιπόν, όταν εκτελείται ένας εκ των signal handlers "`sigchld_handler()`", "`sigalrm_handler()`" μπλοκάρονται τα σήματα λόγω της μάσκας, οπότε με τους παραπάνω handlers στην δικιά μας υλοποίηση εξασφαλίζεται ότι όσο εκτελείται το ένα signal δεν θα διακοπεί για να εκτελεστεί το άλλο.
Στην πραγματικότητα ένας `scheduler` στον χώρο πυρήνα του λειτουργικού συστήματος θα εργαζόταν με System Calls ή Hardware Interrupts των οποίων η δουλεία θα ήταν η μετάβαση από τον χώρο χρήστη στον χώρο πυρήνα (kernel mode).

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 SIGCHLD που λαμβάνει ο scheduler (γονική διεργασία) λαμβάνεται κάθε φορά από οποιαδήποτε διεργασία που ήταν υπό εκτέλεση στείλει κάποιο `signal`, δηλαδή είτε κάνει `exit()` είτε στείλει SIGSTOP και όχι από κάποια συγκεκριμένη. Για τα ζητούμενα της συγκεκριμένης άσκησης δεν έχει ληφθεί κάποιος χειρισμός στην περίπτωση της αποστολής SIGKILL σε κάποια υπό εκτέλεση διεργασία από κάποιο άλλο τερματικό που σημαίνει ότι σε μία τέτοια περίπτωση η `kill()` θα επέστρεφε `error : no such process` καθώς δεν θα υπήρχε διεργασία με το PID που θα δεχόταν ως όρισμα.

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;

Προκειμένου να επιτυγχάνεται συγχρονισμός χρησιμοποιούνται δύο σήματα. Αυτό το κάνουμε επειδή τα σήματα δεν μας εξασφαλίζουν κάποια σειρά προτεραιότητας το καθένα από μόνο του και θα μπορούσε να εκτελείται πρώτα το SIGCONT σε σχέση με το SIGSTOP πράγμα που δεν είναι επιθυμητό (race condition). Χρησιμοποιώντας λοιπόν δύο σήματα μπορούμε να είμαστε βέβαιοι ότι θα εκτελεστούν με την σειρά που επιθυμούμε ώστε είμαστε σίγουροι ότι πάντα θα επιτελείται η σωστή τους λειτουργία. Φαίνεται λοιπόν ότι με αυτόν τον τρόπο ότι πρέπει πρώτα να έχει σταλεί στο παιδί SIGSTOP (μέσω του SIGALRM) προκειμένου να έρθει εν συνεχεία το σήμα SIGCHLD.

1.2 Έλεγχος λειτουργίας χρονοδρομολογητή μέσω φλοιού

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 "queue.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 */

Queue q;
int auto_incr = 0;       /* so that each id stays unique even if we add or kill process(es) */
char program_name[42];

typedef struct PCB {
        pid_t pid;
        int id;
        char* exec;

}PCB;

PCB shell;
PCB current;


/* Print a list of all tasks currently being scheduled.  */
static void
sched_print_tasks(void)
{
        /*assert(0 && "Please fill me!");*/

        node *temp = q.head;
        int i;
        for (i = 0; i < getQueueSize(&q); i++)
        {
                PCB *ptr = temp->data;
                PCB val = *ptr;
                printf("\nMr. Meeseeks %d with Meeseeks PID: %ld fulfills request: %s \n\n", val.id, (long)val.pid,
val.exec);
                temp=temp->next;
        }
        printf("Current running:\n \t\t id: %d\n \t\t PID: %ld\n \t\t Executable: %s\n", current.id,
(long)current.pid, current.exec);
}

/* Send SIGKILL to a task determined by the value of its
 * scheduler-specific id.
 */
static int
sched_kill_task_by_id(int id)
{
        /*assert(0 && "Please fill me!");*/
        int ret = 1;
        int i;
        int size = getQueueSize(&q);
        for (i = 0; i < size; i++)
        {
                PCB elem;
                dequeue(&q, &elem);
                if ( elem.id == id )
```

```
                {
                        ret = 0;
                        kill(elem.pid, SIGKILL);
                        printf("\nMeeseeks %d with Meeseeks PID: %ld just died. Cause of Death: Existence's Pain\n\n",
id, (long)elem.pid);
                }
                else
                {
                        enqueue(&q, &elem);
                }
        }
        if (ret==1)
                {
                        if (shell.id == id)
                        {
                                printf("\nCannot kill the Shell. Press <q> to kill Shell \n\n");
                        }
                        else
                        {
                                printf("\nError 404: Meeseeks not found\n\n");
                        }
                }
        return ret;
}


/* Create a new task.  */
static void
sched_create_task(char *executable)
{
        /*assert(0 && "Please fill me!");*/
        char *newargv[] = {executable, NULL};
        char *newenviron[] = { NULL };
        pid_t new_pid = fork();

        if ( new_pid > 0 )
        {
                int i=0;
                while (executable[i] != '\0'){
                        program_name[i] = executable[i];
                        i = i + 1;
                }
                program_name[i] = '\0';

                PCB task;
                auto_incr = auto_incr + 1;
                task.pid = new_pid;
                task.id = auto_incr;
                task.exec = program_name;
                printf("\nHEY, I'm new Mr Meeseeks %d LOOK AT ME and my Meeseeks PID is %ld. My Purpose is to execute:
%s\n\n", task.id, (long)task.pid, task.exec);
                enqueue(&q, &task);
        }
        if ( new_pid == 0 ) {
                raise(SIGSTOP);
                execve(executable, newargv, newenviron );
                exit(0);
        }
        if (new_pid < 0)
        {
                perror("fork");
        }
        wait_for_ready_children(1);
}

/* 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)
{
        /*assert(0 && "Please fill me!");*/
        if (kill(current.pid, SIGSTOP) < 0) {
                perror("kill");
                exit(1);
        }
}

/*
 * SIGCHLD handler
 */
static void
sigchld_handler(int signum)
{
        // assert(0 && "Please fill me!");

        pid_t p;
        int status;

        for (;;) {

                p = waitpid(-1, &status, WUNTRACED | WNOHANG);

                if (p < 0) {
                        perror("waitpid");
                        exit(1);
                }
                if (p == 0)
                        break;

                explain_wait_status(p, status);

                /* A child has died */
                if ((WIFEXITED(status) || WIFSIGNALED(status)) && (strcmp(current.exec, "shell") != 0)){
                        PCB temp;
                        /* Existence is pain for Meeseeks... */
                        printf("Mr Meeseeks %d: \t Stops existing! *POOF!* \n", current.id);
                        if (getQueueSize(&q) == 0){
                                fprintf(stderr, "Rick (Scheduler):\t All your tasks seem to be complete, now give me
back the Meeseeks Box. Exiting...\n");
                                exit(42);
                        }
                        dequeue(&q, &temp);
                        current = temp;
                }
                else if ((WIFEXITED(status) || WIFSIGNALED(status)) && (strcmp(current.exec, "shell") == 0)){
                        if (getQueueSize(&q) == 0){
                                fprintf(stderr, "Rick (Scheduler):\t All your tasks seem to be complete, now give me
back the Meeseeks Box. Exiting...\n");
                                exit(42);
                        }
                        PCB temp;
                        dequeue(&q, &temp);
                        current = temp;
                }
                /* A child has stopped due to SIGSTOP/SIGTSTP, etc */
                if (WIFSTOPPED(status)){
                        PCB temp;
                        if ( strcmp(current.exec, "shell") == 0 )
                        {
                                printf("This is the Shell bruv. Nothing to do here! \n");
```

```
                        }
                        else
                        {
                                printf("Mr Meeseeks %d: \t I need more time to fullfill the request!\n", current.id);
                        }
                        enqueue(&q, &current);
                        dequeue(&q, &temp);
                        current = temp;
                }
                if (kill(current.pid, SIGCONT) < 0) {
                        perror("kill");
                        exit(1);
                }
                else{
                        if ( strcmp(current.exec, "shell") == 0 )
                        {
                                printf("This is the Shell bruv. What up biatch?! \n");
                        }
                        else
                        {
                                printf("Mr Meeseeks %d: \t Back to work again!\n", current.id);
                        }
                        alarm(SCHED_TQ_SEC);
                }
        }
}

/* 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};
        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: \t giving up on shell request processing.\n");
                        break;
                }
        }
}

int main(int argc, char *argv[])
{
        int nproc;
        pid_t pid;
        char *newenviron[] = { NULL };
        queueInit(&q, sizeof(PCB));

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


        /* TODO: add the shell to the scheduler's tasks */

        /*
         * 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 */

        // if (nproc == 0) {
                // fprintf(stderr, "Rick (Scheduler): \t You don't seem to have any requests, why did you ask for a
Meeseeks Box? Exiting...\n");
                // exit(2);
        // }
        // else {
                // fprintf(stderr, "Rick (Scheduler): \t This is a Meeseeks Box, let me show you how it works. You
press this...\n");
        // }

        int i;
        for (i = 1; i <= nproc; i++) {

                char *newargv[] = { argv[i], NULL };
                PCB camtono;
                pid = fork();

                if (pid > 0){
                        auto_incr = auto_incr + 1;
                        printf("HEY, I'm Mr Meeseeks %d LOOK AT ME and my Meeseeks PID is %ld. \n", auto_incr,
(long)pid);
                        printf("Rick (Scheduler): \t Mr Meeseeks %d, run process named %s. \n", auto_incr, argv[i]);
                        printf("Mr Meeseeks %d: \t YES SIR YI. \n", auto_incr);
                        camtono.pid = pid;
                        camtono.id = auto_incr;
                        camtono.exec = argv[i];
```

```
                        enqueue(&q, &camtono);

                }

                if (pid == 0) {
                        raise(SIGSTOP);
                        execve(argv[i], newargv, newenviron);
                        exit(0);
                }

                if (pid < 0) {
                        perror("Fork");
                }
    }

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

        /* TODO: add the shell to the scheduler's tasks */
        auto_incr = auto_incr + 1;
        shell.pid = shell_pid;
        shell.id = auto_incr;
        shell.exec = "shell";
        enqueue(&q, &shell);



        /* Wait for all children to raise SIGSTOP before exec()ing. */
        wait_for_ready_children(nproc+1);

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

        dequeue(&q, &current);
        printf("First Process PID: %ld, Meeseeks ID: %d\n", (long)current.pid, current.id);
        alarm(SCHED_TQ_SEC);
        kill(current.pid, 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, "HEY Morty, what are you doing here?! You are not supposed to be down here!\n");
        return 1;

}
```

Έξοδος για 3 διεργασίες prog:

```
oslabd43@orion:~/Ask4$ ./scheduler-shell prog prog prog
HEY, I'm Mr Meeseeks 1 LOOK AT ME and my Meeseeks PID is 21497.
Rick (Scheduler):        Mr Meeseeks 1, run process named prog.
Mr Meeseeks 1:   YES SIR YI.
HEY, I'm Mr Meeseeks 2 LOOK AT ME and my Meeseeks PID is 21498.
Rick (Scheduler):        Mr Meeseeks 2, run process named prog.
Mr Meeseeks 2:   YES SIR YI.
HEY, I'm Mr Meeseeks 3 LOOK AT ME and my Meeseeks PID is 21499.
Rick (Scheduler):        Mr Meeseeks 3, run process named prog.
Mr Meeseeks 3:   YES SIR YI.
My PID = 21496: Child PID = 21497 has been stopped by a signal, signo = 19
My PID = 21496: Child PID = 21498 has been stopped by a signal, signo = 19
My PID = 21496: Child PID = 21500 has been stopped by a signal, signo = 19
My PID = 21496: Child PID = 21499 has been stopped by a signal, signo = 19
First Process PID: 21497, Meeseeks ID: 1
prog: Starting, NMSG = 100, delay = 103
prog[21497]: This is message 0
prog[21497]: This is message 1
prog[21497]: This is message 2
```

```
prog[21497]: This is message 3
My PID = 21496: Child PID = 21497 has been stopped by a signal, signo = 19
Mr Meeseeks 1:   I need more time to fullfill the request!
Mr Meeseeks 2:   Back to work again!
prog: Starting, NMSG = 100, delay = 151
prog[21498]: This is message 0
p
prog[21498]: This is message 1
prog[21498]: This is message 2
My PID = 21496: Child PID = 21498 has been stopped by a signal, signo = 19
Mr Meeseeks 2:   I need more time to fullfill the request!
Mr Meeseeks 3:   Back to work again!
prog: Starting, NMSG = 100, delay = 131
prog[21499]: This is message 0
prog[21499]: This is message 1
prog[21499]: This is message 2
My PID = 21496: Child PID = 21499 has been stopped by a signal, signo = 19
Mr Meeseeks 3:   I need more time to fullfill the request!
This is the Shell bruv. What up biatch?!

This is the Shell. Welcome.

Shell>
 Shell: issuing request...
Shell: receiving request return value...

Mr. Meeseeks 1 with Meeseeks PID: 21497 fulfills request: prog


Mr. Meeseeks 2 with Meeseeks PID: 21498 fulfills request: prog


Mr. Meeseeks 3 with Meeseeks PID: 21499 fulfills request: prog

Current running:
                 id: 4
                 PID: 21500
                 Executable: shell
Shell>
 My PID = 21496: Child PID = 21500 has been stopped by a signal, signo = 19
This is the Shell bruv. Nothing to do here!
Mr Meeseeks 1:   Back to work again!
prog[21497]: This is message 4
prog[21497]: This is message 5
e pprog[21497]: This is message 6
rogprog[21497]: This is message 7

My PID = 21496: Child PID = 21497 has been stopped by a signal, signo = 19
Mr Meeseeks 1:   I need more time to fullfill the request!
Mr Meeseeks 2:   Back to work again!
prog[21498]: This is message 3
prog[21498]: This is message 4
prog[21498]: This is message 5
My PID = 21496: Child PID = 21498 has been stopped by a signal, signo = 19
Mr Meeseeks 2:   I need more time to fullfill the request!
Mr Meeseeks 3:   Back to work again!
prog[21499]: This is message 3
prog[21499]: This is message 4
prog[21499]: This is message 5
prog[21499]: This is message 6
My PID = 21496: Child PID = 21499 has been stopped by a signal, signo = 19
Mr Meeseeks 3:   I need more time to fullfill the request!
This is the Shell bruv. What up biatch?!
Shell: issuing request...
Shell: receiving request return value...

HEY, I'm new Mr Meeseeks 5 LOOK AT ME and my Meeseeks PID is 21501. My Purpose is to execute: prog

My PID = 21496: Child PID = 21501 has been stopped by a signal, signo = 19
Shell>
 My PID = 21496: Child PID = 21500 has been stopped by a signal, signo = 19
This is the Shell bruv. Nothing to do here!
Mr Meeseeks 1:   Back to work again!
prog[21497]: This is message 8
prog[21497]: This is message 9
prog[21497]: This is message 10
```

```
My PID = 21496: Child PID = 21497 has been stopped by a signal, signo = 19
Mr Meeseeks 1:   I need more time to fullfill the request!
Mr Meeseeks 2:   Back to work again!
prog[21498]: This is message 6
k prog[21498]: This is message 7
2My PID = 21496: Child PID = 21498 has been stopped by a signal, signo = 19
Mr Meeseeks 2:   I need more time to fullfill the request!
Mr Meeseeks 3:   Back to work again!

prog[21499]: This is message 7
prog[21499]: This is message 8
prog[21499]: This is message 9
My PID = 21496: Child PID = 21499 has been stopped by a signal, signo = 19
Mr Meeseeks 3:   I need more time to fullfill the request!
Mr Meeseeks 5:   Back to work again!
prog: Starting, NMSG = 100, delay = 95
prog[21501]: This is message 0
prog[21501]: This is message 1
prog[21501]: This is message 2
prog[21501]: This is message 3
My PID = 21496: Child PID = 21501 has been stopped by a signal, signo = 19
Mr Meeseeks 5:   I need more time to fullfill the request!
This is the Shell bruv. What up biatch?!
Shell: issuing request...
Shell: receiving request return value...

Meeseeks 2 with Meeseeks PID: 21498 just died. Cause of Death: Existence's Pain

Shell>
 My PID = 21496: Child PID = 21498 was terminated by a signal, signo = 9
Mr Meeseeks 1:   Back to work again!
prog[21497]: This is message 11
prog[21497]: This is message 12
prog[21497]: This is message 13
prog[21497]: This is message 14
My PID = 21496: Child PID = 21497 has been stopped by a signal, signo = 19
Mr Meeseeks 1:   I need more time to fullfill the request!
Mr Meeseeks 3:   Back to work again!
prog[21499]: This is message 10
prog[21499]: This is message 11
prog[21499]: This is message 12
My PID = 21496: Child PID = 21499 has been stopped by a signal, signo = 19
Mr Meeseeks 3:   I need more time to fullfill the request!
Mr Meeseeks 5:   Back to work again!
prog[21501]: This is message 4
prog[21501]: This is message 5
prog[21501]: This is message 6
prog[21501]: This is message 7
My PID = 21496: Child PID = 21501 has been stopped by a signal, signo = 19
Mr Meeseeks 5:   I need more time to fullfill the request!
Mr Meeseeks 1:   Back to work again!
prog[21497]: This is message 15
prog[21497]: This is message 16
prog[21497]: This is message 17
My PID = 21496: Child PID = 21497 has been stopped by a signal, signo = 19
Mr Meeseeks 1:   I need more time to fullfill the request!
Mr Meeseeks 3:   Back to work again!
prog[21499]: This is message 13
prog[21499]: This is message 14
My PID = 21496: Child PID = 21499 has been stopped by a signal, signo = 19
Mr Meeseeks 3:   I need more time to fullfill the request!
Mr Meeseeks 5:   Back to work again!
prog[21501]: This is message 8
prog[21501]: This is message 9
prog[21501]: This is message 10
prog[21501]: This is message 11
My PID = 21496: Child PID = 21501 has been stopped by a signal, signo = 19
Mr Meeseeks 5:   I need more time to fullfill the request!
Mr Meeseeks 1:   Back to work again!
prog[21497]: This is message 18
prog[21497]: This is message 19
prog[21497]: This is message 20
prog[21497]: This is message 21
prog[21497]: This is message 22
My PID = 21496: Child PID = 21497 has been stopped by a signal, signo = 19
Mr Meeseeks 1:   I need more time to fullfill the request!
```

```
Mr Meeseeks 3:   Back to work again!
prog[21499]: This is message 15
prog[21499]: This is message 16
prog[21499]: This is message 17
prog[21499]: This is message 18
qMy PID = 21496: Child PID = 21499 has been stopped by a signal, signo = 19
Mr Meeseeks 3:   I need more time to fullfill the request!
Mr Meeseeks 5:   Back to work again!

Shell: Exiting. Goodbye.
My PID = 21496: Child PID = 21500 terminated normally, exit status = 0
Mr Meeseeks 5:   Stops existing! *POOF!*
Mr Meeseeks 1:   Back to work again!
scheduler: read from shell: Success
Scheduler: giving up on shell request processing.
```

Ερωτήσεις 3.2:

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;

Τρέχοντας την εντολή `'p'`, εκτυπώνονται οι διεργασίες που βρίσκονται εκείνη την στιγμή υπό εκτέλεση και ως τρέχουσα διεργασία εκτυπώνεται πάντα η `shell`. Αυτό γίνεται καθώς η διευθέτηση και εκτέλεση του `'p'` γίνεται πάντα κατά την εκτέλεση του `shell`, όταν έρθει η σειρά του να εκτελεστεί σύμφωνα με την χρονοδρομολόγηση. Στην περίπτωση που επιθυμούσαμε να εκτυπώνουμε κάθε φορά την διεργασία που εκτελείται ακριβώς την στιγμή που στέλνουμε την εντολή `'p'` μέσω του `stdinput` , θα έπρεπε η διεργασία `shell` να εκτελείται συνεχώς στο παρασκήνιο και παράλληλα με την εκτέλεση όλων των διεργασιών στην ουρά, κάτι το οποίο σημαίνει όμως πως η `shell` δεν θα ακολουθούσε το σενάριο χρονοδρομολόγησης όπως οι υπόλοιπες διεργασίες στην ουρά.

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

Είναι αναγκαίο κατά την διάρκεια της εκτέλεσης των αιτήσεων του shell να διασφαλίσουμε ότι η ουρά μας δεν θα τροποποιηθεί από κάποιο εξωτερικό `signal` (πχ αφαίρεση μιας διεργασίας από την ουρά μας λόγω τερματισμού). Κάτι τέτοιο υλοποιείται με την χρήση της συνάρτησης `signals_disable()` πριν την εκτέλεση των αιτημάτων και με την `signals_enable()` στο τέλος αυτών, οι οποίες κάνουν block και unblock αντίστοιχα την δυνατότητα λήψης σημάτων `SIGALRM` και `SIGCHLD`.

1.3 Υλοποίηση προτεραιοτήτων στο χρονοδρομολογητή

Πηγαίος κώδικας scheduler-hilo.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 "queue.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 */

Queue q;
int auto_incr = 0;      /* so that each id stays unique even if we add or kill process(es) */
```

```
char program_name[42];

typedef struct PCB {
        pid_t pid;
        int id;
        char* exec;
        char* pr;
}PCB;

PCB shell;
PCB current;


/* Print a list of all tasks currently being scheduled.  */
static void
sched_print_tasks(void)
{
        /*assert(0 && "Please fill me!");*/

        node *temp = q.head;
        int i;
        for (i = 0; i < getQueueSize(&q); i++)
        {
                PCB *ptr = temp->data;
                PCB val = *ptr;
                printf("\nMr. Meseeks %d with Meeseeks PID: %ld and Priority: %s fulfills request: %s \n\n", val.id,
(long)val.pid, val.pr, val.exec);
                temp=temp->next;
        }
        printf("Current running:\n \t\t id: %d\n \t\t PID: %ld\n \t\t Executable: %s\n \t\t Priority: %s\n",
current.id, (long)current.pid, current.exec, current.pr );
}

/* Send SIGKILL to a task determined by the value of its
 * scheduler-specific id.
 */
static int
sched_kill_task_by_id(int id)
{
        /*assert(0 && "Please fill me!");*/
        int ret = 1;
        int i;
        int size = getQueueSize(&q);
        for (i = 0; i < size; i++)
        {
                PCB elem;
                dequeue(&q, &elem);
                if ( elem.id == id )
                {
                        ret = 0;
                        kill(elem.pid, SIGKILL);
                        printf("\nMeeseeks %d with Meeseeks PID: %ld and Priority: %s just died. Cause of Death:
Existence's Pain\n\n", id, (long)elem.pid, elem.pr);
                }
                else
                {
                        enqueue(&q, &elem);
                }
                printf("%d\n", 1);
        }
        if (ret==1)
                {
                        if (shell.id == id)
                        {
                                printf("\nCannot kill the Shell. Press <q> to kill Shell \n\n");
                        }
                        else
                        {
                                printf("\nError 404: Meeseeks not found\n\n");
                        }
                }
        return ret;
}


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

```
static void
sched_create_task(char *executable)
{
        /*assert(0 && "Please fill me!");*/
        char *newargv[] = {executable, NULL};
        char *newenviron[] = { NULL };

        pid_t new_pid = fork();

        if ( new_pid > 0 )
        {
                int i=0;
                while (executable[i] != '\0'){
                        program_name[i] = executable[i];
                        i = i + 1;
                }
                program_name[i] = '\0';

                auto_incr = auto_incr + 1;
                PCB task;
                task.pid = new_pid;
                task.id = auto_incr;
                task.exec = program_name;
                task.pr = "LOW";
                enqueue(&q, &task);
                printf("\nHEY, I'm  new Mr Meeseeks %d LOOK AT ME and my Meeseeks PID is %ld and Priority: %s. My
Purpose is to execute: %s \n\n", task.id, (long)task.pid, task.pr, task.exec);
        }
        if ( new_pid == 0 ) {
                raise(SIGSTOP);
                execve(executable, newargv, newenviron );
                exit(0);
        }
        if (new_pid < 0)
        {
                perror("fork");
        }
        wait_for_ready_children(1);
}

static int
sched_set_low_prio(int id)
{
        int ret = 1;
        int i;
        int size = getQueueSize(&q);
        PCB changed;
        for (i = 0; i < size; i++)
        {
                PCB elem;
                dequeue(&q, &elem);
                if ( elem.id == id )
                {
                        ret = 0;
                        elem.pr = "LOW";
                        changed = elem;
                        printf("\nMeeseeks %d with Meeseeks PID: %ld changed Priority status to %s.\n\n", id,
(long)elem.pid, elem.pr);
                }
                else{
                        enqueue(&q, &elem);
                }
        }

        if (!ret){
                enqueue(&q, &changed);
        }
        else
        {
                if (shell.id == id)
                {
                        printf("\nNo reason to change Priority status of Shell.\n\n");
                }
                else
                {
                        printf("\nError 404: Meeseeks not found\n\n");
```

```
                }
        }
        return ret;
}

static int
sched_set_high_prio(int id)
{
        int ret = 1;
        int i;
        int size = getQueueSize(&q);
        PCB changed;
        for (i = 0; i < size; i++)
        {
                PCB elem;
                dequeue(&q, &elem);
                if ( elem.id == id )
                {
                        ret = 0;
                        elem.pr = "HIGH";
                        changed = elem;
                        printf("\nMeeseeks %d with Meeseeks PID: %ld changed Priority status to %s.\n\n", id,
(long)elem.pid, elem.pr);
                }
                else{
                        enqueue(&q, &elem);
                }
        }

        if (!ret){
                enqueue(&q, &changed);
        }

        else
        {
                if (shell.id == id)
                {
                        printf("\nNo reason to change Priority status of Shell.\n\n");
                }
                else
                {
                        printf("\nError 404: Meeseeks not found\n\n");
                }
        }
        return ret;
}


/* This function find the next HIGH priority if exists and sets it as the next task to be fulfilled */
static void
find_HIGH_if_exists(void)
{
        int onHigh = 0;
        node *t = q.head;
        int i;
        /* find if there is any HIGH priority task in the queue */
        for (i = 0; i < getQueueSize(&q); i++)
        {
                PCB *ptr = t->data;
                PCB val = *ptr;
                if ((strcmp(val.pr,"HIGH") == 0) && (strcmp(val.exec, "shell") != 0)){
                        onHigh = 1;
                        /*break;*/
                }
                t=t->next;
        }       /* if onHigh == 0 it means we only have LOW priority tasks (except from shell which is always HIGH) */

        // if ((strcmp(current.pr,"HIGH") == 0) && (strcmp(current.exec, "shell") != 0))
        // {
                // onHigh = 1;
        // }

        if (onHigh == 1){
                for (i = 0; i < getQueueSize(&q); i++)
                {
                        PCB tempo;
```

```
                        node *hd = q.head;
                        PCB *pointer = hd->data;
                        PCB val = *pointer;
                        if (strcmp(val.pr,"HIGH") == 0){
                                break;
                        }
                        else {
                                dequeue(&q, &tempo);
                                enqueue(&q, &tempo);
                        }
                }
        }
}



/* 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:
                        sched_set_high_prio(rq->task_arg);
                        return 0;

                case REQ_LOW_TASK:
                        sched_set_low_prio(rq->task_arg);
                        return 0;

                default:
                        return -ENOSYS;
        }
}

/*
 * SIGALRM handler
 */
static void
sigalrm_handler(int signum)
{
        /*assert(0 && "Please fill me!");*/
        if (kill(current.pid, SIGSTOP) < 0) {
                perror("kill");
                exit(1);
        }
}

/*
 * SIGCHLD handler
 */
static void
sigchld_handler(int signum)
{
        // assert(0 && "Please fill me!");

        pid_t p;
        int status;

        for (;;) {

                p = waitpid(-1, &status, WUNTRACED | WNOHANG);

                if (p < 0) {
                        perror("waitpid");
                        exit(1);
```

```
                }
                if (p == 0)
                        break;

                explain_wait_status(p, status);

                /* if onHigh == 0 it means we only have LOW priority tasks (except from shell which is always HIGH) */


                /*      if we have HIGH priorities and the current is one of them we run the current
                        if we have HIGH priorities and the current is LOW then we dont run it
                        if we only have LOW priorities (except from shell) then we run the current regardless of
whether it is HIGH or LOW       */

                /* A child has died */
                if ((WIFEXITED(status) || WIFSIGNALED(status)) && (strcmp(current.exec, "shell") != 0)){
                        PCB temp;
                        /* Existence is pain for Meeseeks... */
                        printf("Mr Meeseeks %d: \t Stops existing! *POOF!* \n", current.id);
                        if (getQueueSize(&q) == 0){
                                fprintf(stderr, "Rick (Scheduler): \t All your tasks seem to be complete, now give me
back the Meeseeks Box. Exiting...\n");
                                exit(42);
                        }
                        find_HIGH_if_exists();
                        dequeue(&q, &temp);
                        current = temp;
                }
                else if ((WIFEXITED(status) || WIFSIGNALED(status)) && (strcmp(current.exec, "shell") == 0)){
                        if (getQueueSize(&q) == 0){
                                fprintf(stderr, "Rick (Scheduler):\t All your tasks seem to be complete, now give me
back the Meeseeks Box. Exiting...\n");
                                exit(42);
                        }
                        find_HIGH_if_exists();
                        PCB temp;
                        dequeue(&q, &temp);
                        current = temp;
                }

                /* A child has stopped due to SIGSTOP/SIGTSTP, etc */
                if (WIFSTOPPED(status)){
                        PCB temp;
                        if ( strcmp(current.exec, "shell") == 0 )
                        {
                                printf("This is the Shell bruv. Nothing to do here!\n");
                        }
                        else
                        {
                                printf("Mr Meeseeks %d: \t I need more time to fullfill the request!\n", current.id);
                        }
                        enqueue(&q, &current);
                        find_HIGH_if_exists();
                        dequeue(&q, &temp);
                        current = temp;
                }

                if (kill(current.pid, SIGCONT) < 0) {
                        perror("kill");
                        exit(1);
                }
                else{
                        if ( strcmp(current.exec, "shell") == 0 )
                        {
                                printf("This is the Shell bruv. What up biatch?! \n");
                        }
                        else
                        {
                                printf("Mr Meeseeks %d: \t Back to work again!\n", current.id);
                        }
                        alarm(SCHED_TQ_SEC);
                }
        }
}

/* 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: \t giving up on shell request processing.\n");
                        break;
                }
        }
```

```
}

int main(int argc, char *argv[])
{
        int nproc;
        pid_t pid;
        char *newenviron[] = { NULL };
        queueInit(&q, sizeof(PCB));

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

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

        /* TODO: add the shell to the scheduler's tasks */

        /*
         * 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 */

        // if (nproc == 0) {
                // fprintf(stderr, "Rick (Scheduler): \t You don't seem to have any requests, why did you ask for a
Meeseeks Box? Exiting...\n");
                // exit(2);
        // }
        // else {
                // fprintf(stderr, "Rick (Scheduler): \t This is a Meeseeks Box, let me show you how it works. You
press this...\n");
        // }

        int i;
        for (i = 1; i <= nproc; i++) {

                char *newargv[] = { argv[i], NULL };
                PCB camtono;
                pid = fork();

                if (pid > 0){
                        auto_incr = auto_incr + 1;
                        camtono.pid = pid;
                        camtono.id = auto_incr;
                        camtono.exec = argv[i];
                        camtono.pr = "LOW";
                        printf("HEY, I'm Mr Meeseeks %d LOOK AT ME with Priority: %s and my Meeseeks PID is %ld. \n",
auto_incr, camtono.pr, (long)pid);
                        printf("Rick (Scheduler): \t Mr Meeseeks %d, run process named %s. \n", auto_incr, argv[i]);
                        printf("Mr Meeseeks %d: \t YES SIR YI. \n", auto_incr);
                        enqueue(&q, &camtono);

                }

                if (pid == 0) {
                        raise(SIGSTOP);
                        execve(argv[i], newargv, newenviron);
                        exit(0);
                }

                if (pid < 0) {
                        perror("Fork");
                }
    }


        /* TODO: add the shell to the scheduler's tasks */
        auto_incr = auto_incr + 1;
        shell.pid = shell_pid;
        shell.id = auto_incr;
        shell.pr = "LOW";
        shell.exec = "shell";
        enqueue(&q, &shell);
```

```
        /* Wait for all children to raise SIGSTOP before exec()ing. */
        wait_for_ready_children(nproc+1);

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

        dequeue(&q, &current);
        printf("First Process PID: %ld, Meeseeks ID: %d\n", (long)current.pid, current.id);
        alarm(SCHED_TQ_SEC);
        kill(current.pid, 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, "HEY Morty, what are you doing here?! You are not supposed to be down here!\n");
        return 1;

}
```

Έξοδος για 3 διεργασίες prog:

```
oslabd43@orion:~/Ask4$ ./scheduler-hilo prog prog prog
HEY, I'm Mr Meeseeks 1 LOOK AT ME with Priority: LOW and my Meeseeks PID is 21552.
Rick (Scheduler):        Mr Meeseeks 1, run process named prog.
Mr Meeseeks 1:   YES SIR YI.
HEY, I'm Mr Meeseeks 2 LOOK AT ME with Priority: LOW and my Meeseeks PID is 21553.
Rick (Scheduler):        Mr Meeseeks 2, run process named prog.
Mr Meeseeks 2:   YES SIR YI.
HEY, I'm Mr Meeseeks 3 LOOK AT ME with Priority: LOW and my Meeseeks PID is 21554.
Rick (Scheduler):        Mr Meeseeks 3, run process named prog.
Mr Meeseeks 3:   YES SIR YI.
My PID = 21550: Child PID = 21554 has been stopped by a signal, signo = 19
My PID = 21550: Child PID = 21552 has been stopped by a signal, signo = 19
My PID = 21550: Child PID = 21551 has been stopped by a signal, signo = 19
My PID = 21550: Child PID = 21553 has been stopped by a signal, signo = 19
First Process PID: 21552, Meeseeks ID: 1
prog: Starting, NMSG = 100, delay = 131
prog[21552]: This is message 0
prog[21552]: This is message 1
prog[21552]: This is message 2
My PID = 21550: Child PID = 21552 has been stopped by a signal, signo = 19
Mr Meeseeks 1:   I need more time to fullfill the request!
Mr Meeseeks 2:   Back to work again!
prog: Starting, NMSG = 100, delay = 112
prog[21553]: This is message 0
prog[21553]: This is message 1
p
prog[21553]: This is message 2
My PID = 21550: Child PID = 21553 has been stopped by a signal, signo = 19
Mr Meeseeks 2:   I need more time to fullfill the request!
Mr Meeseeks 3:   Back to work again!
prog: Starting, NMSG = 100, delay = 159
prog[21554]: This is message 0
prog[21554]: This is message 1
prog[21554]: This is message 2
My PID = 21550: Child PID = 21554 has been stopped by a signal, signo = 19
Mr Meeseeks 3:   I need more time to fullfill the request!
This is the Shell bruv. What up biatch?!

This is the Shell. Welcome.

Shell>
 Shell: issuing request...
Shell: receiving request return value...

Mr. Meseeks 1 with Meeseeks PID: 21552 and Priority: LOW fulfills request: prog
```

```
Mr. Meseeks 2 with Meeseeks PID: 21553 and Priority: LOW fulfills request: prog


Mr. Meseeks 3 with Meeseeks PID: 21554 and Priority: LOW fulfills request: prog

Current running:
                  id: 4
                  PID: 21551
                  Executable: shell
                  Priority: HIGH
Shell>
 My PID = 21550: Child PID = 21551 has been stopped by a signal, signo = 19
This is the Shell bruv. Nothing to do here!
Mr Meeseeks 1:   Back to work again!
prog[21552]: This is message 3
h prog[21552]: This is message 4
2
prog[21552]: This is message 5
My PID = 21550: Child PID = 21552 has been stopped by a signal, signo = 19
Mr Meeseeks 1:   I need more time to fullfill the request!
Mr Meeseeks 2:   Back to work again!
prog[21553]: This is message 3
prog[21553]: This is message 4
prog[21553]: This is message 5
My PID = 21550: Child PID = 21553 has been stopped by a signal, signo = 19
Mr Meeseeks 2:   I need more time to fullfill the request!
Mr Meeseeks 3:   Back to work again!
prog[21554]: This is message 3
prog[21554]: This is message 4
My PID = 21550: Child PID = 21554 has been stopped by a signal, signo = 19
Mr Meeseeks 3:   I need more time to fullfill the request!
This is the Shell bruv. What up biatch?!
Shell: issuing request...
Shell: receiving request return value...

Meeseeks 2 with Meeseeks PID: 21553 changed Priority status to HIGH.

Shell>
 My PID = 21550: Child PID = 21551 has been stopped by a signal, signo = 19
This is the Shell bruv. Nothing to do here!
Mr Meeseeks 2:   Back to work again!
prog[21553]: This is message 6
prog[21553]: This is message 7
prog[21553]: This is message 8
prog[21553]: This is message 9
My PID = 21550: Child PID = 21553 has been stopped by a signal, signo = 19
Mr Meeseeks 2:   I need more time to fullfill the request!
This is the Shell bruv. What up biatch?!
My PID = 21550: Child PID = 21551 has been stopped by a signal, signo = 19
This is the Shell bruv. Nothing to do here!
Mr Meeseeks 2:   Back to work again!
prog[21553]: This is message 10
prog[21553]: This is message 11
prog[21553]: This is message 12
prog[21553]: This is message 13
My PID = 21550: Child PID = 21553 has been stopped by a signal, signo = 19
Mr Meeseeks 2:   I need more time to fullfill the request!
This is the Shell bruv. What up biatch?!
My PID = 21550: Child PID = 21551 has been stopped by a signal, signo = 19
This is the Shell bruv. Nothing to do here!
Mr Meeseeks 2:   Back to work again!
prog[21553]: This is message 14
prog[21553]: This is message 15
prog[21553]: This is message 16
My PID = 21550: Child PID = 21553 has been stopped by a signal, signo = 19
Mr Meeseeks 2:   I need more time to fullfill the request!
This is the Shell bruv. What up biatch?!
My PID = 21550: Child PID = 21551 has been stopped by a signal, signo = 19
This is the Shell bruv. Nothing to do here!
Mr Meeseeks 2:   Back to work again!
prog[21553]: This is message 17
l prog[21553]: This is message 18
2
prog[21553]: This is message 19
```

```
My PID = 21550: Child PID = 21553 has been stopped by a signal, signo = 19
Mr Meeseeks 2:   I need more time to fullfill the request!
This is the Shell bruv. What up biatch?!
Shell: issuing request...
Shell: receiving request return value...

Meeseeks 2 with Meeseeks PID: 21553 changed Priority status to LOW.

Shell>
 My PID = 21550: Child PID = 21551 has been stopped by a signal, signo = 19
This is the Shell bruv. Nothing to do here!
Mr Meeseeks 1:   Back to work again!
prog[21552]: This is message 6
prog[21552]: This is message 7
prog[21552]: This is message 8
My PID = 21550: Child PID = 21552 has been stopped by a signal, signo = 19
Mr Meeseeks 1:   I need more time to fullfill the request!
Mr Meeseeks 3:   Back to work again!
prog[21554]: This is message 5
prog[21554]: This is message 6
prog[21554]: This is message 7
My PID = 21550: Child PID = 21554 has been stopped by a signal, signo = 19
Mr Meeseeks 3:   I need more time to fullfill the request!
Mr Meeseeks 2:   Back to work again!
prog[21553]: This is message 20
prog[21553]: This is message 21
prog[21553]: This is message 22
My PID = 21550: Child PID = 21553 has been stopped by a signal, signo = 19
Mr Meeseeks 2:   I need more time to fullfill the request!
This is the Shell bruv. What up biatch?!
My PID = 21550: Child PID = 21551 has been stopped by a signal, signo = 19
This is the Shell bruv. Nothing to do here!
Mr Meeseeks 1:   Back to work again!
kprog[21552]: This is message 9
 1
prog[21552]: This is message 10
prog[21552]: This is message 11
My PID = 21550: Child PID = 21552 has been stopped by a signal, signo = 19
Mr Meeseeks 1:   I need more time to fullfill the request!
Mr Meeseeks 3:   Back to work again!
prog[21554]: This is message 8
prog[21554]: This is message 9
My PID = 21550: Child PID = 21554 has been stopped by a signal, signo = 19
Mr Meeseeks 3:   I need more time to fullfill the request!
Mr Meeseeks 2:   Back to work again!
prog[21553]: This is message 23
prog[21553]: This is message 24
prog[21553]: This is message 25
My PID = 21550: Child PID = 21553 has been stopped by a signal, signo = 19
Mr Meeseeks 2:   I need more time to fullfill the request!
This is the Shell bruv. What up biatch?!
Shell: issuing request...
Shell: receiving request return value...

Meeseeks 1 with Meeseeks PID: 21552 and Priority: LOW just died. Cause of Death: Existence's Pain

Shell>
 My PID = 21550: Child PID = 21552 was terminated by a signal, signo = 9
Mr Meeseeks 3:   Back to work again!
prog[21554]: This is message 10
prog[21554]: This is message 11
prog[21554]: This is message 12
My PID = 21550: Child PID = 21554 has been stopped by a signal, signo = 19
Mr Meeseeks 3:   I need more time to fullfill the request!
Mr Meeseeks 2:   Back to work again!
prog[21553]: This is message 26
prog[21553]: This is message 27
prog[21553]: This is message 28
prog[21553]: This is message 29
```

Ερωτήσεις 3.3:

1. Περιγράψτε ένα σενάριο δημιουργίας λιμοκτονίας.

Στην υλοποίηση μας χρησιμοποιείται μια ουρά η οποία μπορεί να εκτελέσει τις κλασσικές λειτουργίες των Queues δηλαδή enqueue (τοποθέτηση στοιχείου στο τέλος της ουράς), dequeue (αφαίρεση του πρώτου στοιχείου της ουράς δηλαδή του στοιχείου στο οποίο δείχνει το head). Τώρα όσον αφορά την επιλογή της επόμενης διεργασίας που θα εκτελεστεί, αυτό το πραγματοποιούμε μέσω της συνάρτησης `find_HIGH_if_exists()` μέσω της οποίας γίνεται μια αναζήτηση στην ουρά για να διαπιστώσουμε εάν υπάρχουν σε αυτήν HIGH Priority στοιχεία. Εάν, υπάρχουν τότε θέτουμε ως current το πρώτο HIGH που βρίσκουμε και έπειτα επαναλαμβάνουμε την ίδια διαδικασία. Εάν δεν υπάρχουν HIGH Priority στοιχεία τότε αυτό σημαίνει ότι όλα είναι LOW Priority επομένως εκτελούμε απλά την πρώτη που θα βρούμε στην ουρά. Τώρα όσον αφορά στο ζήτημα της δημιουργίας σεναρίου λιμοκτονίας, ένα απλό παράδειγμα είναι η περίπτωση που θέτουμε μέσω του `shell` μια διεργασία που θα αργούσε πάρα πολύ (έστω το `prog` με πολύ μεγάλο NMSG) ως HIGH και ύστερα να τεθεί το `shell` ως LOW Priority (ώστε να μην μπορούμε να αλλάξουμε κανένα Priority) τότε οι υπόλοιπες διεργασίες που είναι LOW θα έπρεπε να περιμένουν πάρα πολύ για να εκτελεστούν καθώς θα πρέπει πρώτα να τελειώσει η λειτουργία του "χρονοβόρου" HIGH Priority task.