ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
& ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Λειτουργικά Συστήματα

7ο Εξάμηνο

Άσκηση 4

Χρονοδρομολόγηση

Εργαστηριακή Ομάδα Δ01

Σταυρακάκης Δημήτριος
ΑΜ: 03112017

Αθανασίου Νικόλαος
ΑΜ: 03112074

Άσκηση 3:
Χρονοδρομολόγηση

1.1 Υλοποίηση χρονοδρομολογητή κυκλικής επαναφοράς στο χώρο χρήστη

Στόχος της άσκησης αυτής είναι η υλοποίηση ενός χρονοδρομολογητή κυκλικής επαναφοράς (Round-Robin) ο οποίος εκτελείται ως διεργασία γονέας στο userspace και χρονοδρομολογεί διεργασίες παιδιά.Οι διεργασίες χρονοδρομολογούνται κυκλικά με ανάθεση ίσων κβάντων χρόνου στην καθεμία(tq) με του οποίου την εκπνοή η διεργασία διακόπτεται -αν δεν έχει τερματιστεί μέχρι την εκπνοή – με την αποστολή σήματος SIGSTOP και ενεργοποιεί την επόμενη στη λίστα με την αποστολή σήματος SIGCONT.
Αρχικά,δημιουργούμε μία ουρά διεργασιών με απλά συνδεδεμένη λίστα κρατώντας την κεφαλή και το τελευταίο στοιχείο της σε δύο μεταβλητές το head και το last αντίστοιχα.
Αρχικά δημιουργούμε τις διεργασίες παιδιά από αυτές που δίνονται σαν ορίσματα στο πρόγραμμα μας (prog) και στέλνουμε σε κάθεμια SIGSTOP για να περιμένουμε τη δημιουργία ολόκληρης της λίστας και αφού ολοκληρωθεί η δημιουργία της, στέλνουμε SIGCONT στην κεφαλή της λίστας για να ξεκινήσει να εκτελείται. Μια διεργασία μπορεί:

- Να τερματιστεί εντός του κβάντου χρόνου της
- Να εκπνεύσει το κβάντο χρόνου της πριν τερματιστεί
- Να δεχτεί σήμα τύπου kill κατά την εκτέλεση του προγράμματος

Για να έχουμε ενα συνεπές και ορθό πρόγραμμα χρησιμοποιούμε δυο handlers, τον sigalrm_handler και τον sigchld_handler. Ο πρώτος εκτελείται όταν ληφθεί σήμα SIGALRM, το οποίο συμβαίνει μόλις εκπνεύσει ο χρόνος που έχουμε ορίσει στην εντολή alarm. Η μόνη λειτουργία που επιτελεί είναι να σταματά (στέλνει σήμα SIGSTOP) στη διεργασία που εκείνη την ώρα εκτελείται. Το τμήμα κώδικα του δεύτερου handler εκτελείται μολίς σταλέι σήμα SIGCHLD. Αυτό συμβαίνει όταν κάποιο παιδί μεταβάλλει την κατάστασή του(σταματήσει, φάει kill ή τερματιστεί). Στον handler αυτόν το μόνο που κάνουμε είναι να φτιάχνουμε τη λίστα μας για να συνεχίζει κανονικά η χρονοδρομολόγηση. Αν μια διεργασία τερματιστεί ή “σκοτωθεί” τότε την αφαιρούμε απο τη λίστα ενώ αν απλά έχει διακοπεί λόγο του χρονοδρομολογητή τη βάζουμε στο τελος και συνεχίζουμε με το head της λίστας. (που είναι η επόμενη διεργασία προς εκτέλεση στη σειρά)
Ο κώδικας του προγράμματος scheduler.c είναι ο ακόλουθος:

```
#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"

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


struct process{
```

```
	pid_t mypid;
	struct process* next;
	char *name;
	int data;
};
/*Definition of my list nodes*/
struct process *head=NULL;
struct process *newnode=NULL;
//struct process *delete=NULL;
struct process *last=NULL;
//struct process *running=NULL;
/*
 * SIGALRM handler
 */
static void
sigalrm_handler(int signum)
{
	//assert(0 && "Please fill me!");
	kill(head->mypid,SIGSTOP);
	//alarm(SCHED_TQ_SEC);

}

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

	while(1){
		p=waitpid(-1,&status,WNOHANG|WUNTRACED);
		if (p<0){
			perror("waitpid");
			exit(1);
		}
		if(p==0) break;			////////

		explain_wait_status(p,status);
		//p has the pid of our child that has changed its status

		if (WIFEXITED(status) || WIFSIGNALED(status)){//if terminated or killed by a
signal remove it from the list
			struct process* delete=NULL;
			struct process * temp;
			temp=head;

			while(temp!=NULL){
				if (temp->mypid==p && temp==head){ //if i got to delete the head
```

```
                                        if (temp->next == NULL){
                                                free(temp);
                                                printf("I am done\n");
                                                exit(0);
                                        }
                                        else{
                                                head=temp->next;
                                                free(temp);
                                        }
                                }
                                else if (temp->mypid==p && temp==last){
                                                //if i got to delete the last element of the list
                                        last=delete;
                                        last->next=NULL;
                                        free(temp);
                                }
                                else if(temp->mypid==p){ //if i delete a random element in the list
                                        delete->next=temp->next;
                                        free(temp);
                                }
                                else{
                        //if i got to continue searching , will never access any node after the last element
                                        delete=temp;
                                        temp=temp->next;
                                        continue;
                                }
                                break;
                        }



                }

                if (WIFSTOPPED(status)){
                        //if our process is stopped by the scheduler, continue with the next one
                        last->next=head;
                        last=head;
                        struct process * temp;
                        temp=head;
                        head=head->next;
                        temp->next=NULL;
                }
                alarm(SCHED_TQ_SEC);
                //printf("%s\n",running->name);
                kill(head->mypid,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);
        }
}



int main(int argc, char *argv[])
{
        int nproc;
        /*
         * For each of argv[1] to argv[argc - 1],
         * create a new child process, add it to the process list.
         */
        char executable[] = "prog";
      char *newargv[] = { executable, NULL, NULL, NULL };
      char *newenviron[] = { NULL };

        nproc = argc-1; /* number of proccesses goes here */
        if (nproc == 0) {
```

```
        fprintf(stderr, "Scheduler: No tasks. Exiting...\n");
        exit(1);
}

   pid_t mypid;
   int i;
   /*Fork and create the list*/
   for (i=0;i<nproc;i++){
            mypid=fork();
            if (mypid<0){
                    printf("Error with forks\n");
            }
            if(mypid==0){
                    raise(SIGSTOP);
                    printf("I am %s, PID = %ld\n",
            argv[0], (long)getpid());
            printf("About to replace myself with the executable %s...\n",
                  executable);
            sleep(2);

            execve(executable, newargv, newenviron);

            /* execve() only returns on error */
            perror("execve");
            exit(1);
                    }
            else{
                    if (i==0){
                            head=(struct process *)malloc(sizeof(struct process));
                            if (head==NULL) printf("Error with malloc\n");
                            head->mypid=mypid;
                            head->next=NULL;
                            head->data=i+1;
                            head->name=argv[i+1];
                            last=head;

                    }
                    else{
                            newnode=(struct process *)malloc(sizeof(struct process));
                            if (newnode==NULL) printf("Error with malloc\n");
                            newnode->mypid=mypid;
                            newnode->next=NULL;
                            newnode->data=i+1;
                            newnode->name=argv[i+1];
                            last->next=newnode;
                            last=newnode;
                    }

            }


   }
```

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

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

        /*Set the alarm on*/
        alarm(SCHED_TQ_SEC);

        /*Start the first process*/
        kill(head->mypid,SIGCONT);

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

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

Κάποια screenshots για την εκτέλεση του προγράμματος:

```
dimstav23@Spam ~/Desktop/Λειτουργικά Άσκηση 4/Askhsh4/Askhsh1.1 $ ./scheduler prog prog prog
My PID = 4056: Child PID = 4057 has been stopped by a signal, signo = 19
My PID = 4056: Child PID = 4058 has been stopped by a signal, signo = 19
My PID = 4056: Child PID = 4059 has been stopped by a signal, signo = 19
I am ./scheduler, PID = 4057
About to replace myself with the executable prog...
My PID = 4056: Child PID = 4057 has been stopped by a signal, signo = 19
I am ./scheduler, PID = 4058
About to replace myself with the executable prog...
My PID = 4056: Child PID = 4058 has been stopped by a signal, signo = 19
I am ./scheduler, PID = 4059
About to replace myself with the executable prog...
My PID = 4056: Child PID = 4059 has been stopped by a signal, signo = 19
prog: Starting, NMSG = 200, delay = 85
prog[4057]: This is message 0
prog[4057]: This is message 1
prog[4057]: This is message 2
prog[4057]: This is message 3
prog[4057]: This is message 4
prog[4057]: This is message 5
prog[4057]: This is message 6
prog[4057]: This is message 7
prog[4057]: This is message 8
My PID = 4056: Child PID = 4057 has been stopped by a signal, signo = 19
prog: Starting, NMSG = 200, delay = 131
prog[4058]: This is message 0
prog[4058]: This is message 1
prog[4058]: This is message 2
prog[4058]: This is message 3
prog[4058]: This is message 4
prog[4058]: This is message 5
My PID = 4056: Child PID = 4058 has been stopped by a signal, signo = 19
prog: Starting, NMSG = 200, delay = 113
prog[4059]: This is message 0
prog[4059]: This is message 1
prog[4059]: This is message 2
prog[4059]: This is message 3
prog[4059]: This is message 4
prog[4059]: This is message 5
prog[4059]: This is message 6
My PID = 4056: Child PID = 4059 has been stopped by a signal, signo = 19
```

```
My PID = 4056: Child PID = 4059 has been stopped by a signal, signo = 19
prog[4057]: This is message 192
prog[4057]: This is message 193
prog[4057]: This is message 194
prog[4057]: This is message 195
prog[4057]: This is message 196
prog[4057]: This is message 197
prog[4057]: This is message 198
prog[4057]: This is message 199
My PID = 4056: Child PID = 4057 has been stopped by a signal, signo = 19
prog[4058]: This is message 124
prog[4058]: This is message 125
prog[4058]: This is message 126
prog[4058]: This is message 127
prog[4058]: This is message 128
prog[4058]: This is message 129
My PID = 4056: Child PID = 4058 has been stopped by a signal, signo = 19
prog[4059]: This is message 146
prog[4059]: This is message 147
prog[4059]: This is message 148
prog[4059]: This is message 149
prog[4059]: This is message 150
prog[4059]: This is message 151
My PID = 4056: Child PID = 4059 has been stopped by a signal, signo = 19
My PID = 4056: Child PID = 4057 terminated normally, exit status = 0
prog[4058]: This is message 130
prog[4058]: This is message 131
prog[4058]: This is message 132
prog[4058]: This is message 133
prog[4058]: This is message 134
My PID = 4056: Child PID = 4058 has been stopped by a signal, signo = 19
prog[4059]: This is message 152
prog[4059]: This is message 153
prog[4059]: This is message 154
prog[4059]: This is message 155
prog[4059]: This is message 156
prog[4059]: This is message 157
prog[4059]: This is message 158
My PID = 4056: Child PID = 4059 has been stopped by a signal, signo = 19
prog[4058]: This is message 135
prog[4058]: This is message 136
```

```
prog[4058]: This is message 170
prog[4058]: This is message 171
prog[4058]: This is message 172
prog[4058]: This is message 173
prog[4058]: This is message 174
My PID = 4056: Child PID = 4058 has been stopped by a signal, signo = 19
prog[4059]: This is message 198
prog[4059]: This is message 199
My PID = 4056: Child PID = 4059 terminated normally, exit status = 0
prog[4058]: This is message 175
prog[4058]: This is message 176
prog[4058]: This is message 177
prog[4058]: This is message 178
prog[4058]: This is message 179
My PID = 4056: Child PID = 4058 has been stopped by a signal, signo = 19
prog[4058]: This is message 180
prog[4058]: This is message 181
prog[4058]: This is message 182
prog[4058]: This is message 183
prog[4058]: This is message 184
prog[4058]: This is message 185
My PID = 4056: Child PID = 4058 has been stopped by a signal, signo = 19
prog[4058]: This is message 186
prog[4058]: This is message 187
prog[4058]: This is message 188
prog[4058]: This is message 189
prog[4058]: This is message 190
My PID = 4056: Child PID = 4058 has been stopped by a signal, signo = 19
prog[4058]: This is message 191
prog[4058]: This is message 192
prog[4058]: This is message 193
prog[4058]: This is message 194
prog[4058]: This is message 195
prog[4058]: This is message 196
My PID = 4056: Child PID = 4058 has been stopped by a signal, signo = 19
prog[4058]: This is message 197
prog[4058]: This is message 198
prog[4058]: This is message 199
My PID = 4056: Child PID = 4058 terminated normally, exit status = 0
I am done
```

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;

Η διαδικασία που θα ακολουθηθεί στην περίπτωση αυτή είναι:
όταν εκτελείται ο hanler 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 χρησιμοποιύσαμε 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.

1.2 Έλεγχος λειτουργίας χρονοδρομολογητή μέσω φλοιού

Στο δεύτερο μέρος της άσκησης επεκτείνεται ο χρονοδρομολογητής του προηγούμενου ερωτήματος ώστε να συμπεριλαμβάνεται και ο φλοιός στη χρονοδρομολόγηση σαν μια απλά

διεργασία που χρονοδρομολογείται κανονικά απλά αποστέλλει αιτήσεις στο χρονοδρομολογητή μας για τύπωμα των διεργασιών διαγραφή, δημιουργία διεργασίας και εντολή τεραμτισμού του εαυτού του. Αναλυτικότερα δέχεται τις εξής εντολές :

• **Εντολή'p'**:Ο χρονοδρομολογητής εκτυπώνει για κάθε διεργασία που βρίσκεται στη λίστα εκείνη τη στιγμή το όνομα της το pid της το id της το όνομα της.

• **Εντολή 'k'**: Δέχεται όρισμα το id μιας διεργασίας και σημαίνει τον τερματισμό της.

• **Εντολή 'e'**: Δέχεται όρισμα το όνομα ενός εκτελέσιμου στον τρέχοντα κατάλογο, και ζητά τη δημιουργία μιας νέας διεργασίας από τον χρονοδρομολογητή,με το δοθέν όνομα.

• **Εντολή 'q'**: Ο φλοιός τερματίζει τη λειτουργία του.

Ακολουθείται ο ίδιος σκελετός με την προηγούμενη άσκηση απλά προσθέτεται ο shell σαν κανονική διεργασία στην κεφαλή της λίστας. Επίσης υλοποιήθηκαν οι sched_kill_task_by_id() και sched_create_task(). Η πρώτη αφορά τη διαχείριση της εντολής του shell k <id> για την οποία ψάχνει να βρει τη διεργασία με το δοσμένο id και της στέλνει σήμα SIGKILL.

Η δεύτερη συνάρτηση ενεργοποιείται όταν πατήσουμε στο shell e <proc_name> και δημιουργεί μαι νέα διεργασία με το δοθέν όνομα που δέχεται ως παράμετρο.Στη συνέχεια παρατίθεται ο κώδικας:

```
#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"

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


struct process{
        pid_t mypid;
        struct process* next;
        char *name;
        int data;
};
/*Definition of my list nodes*/
struct process *head=NULL;
struct process *newnode=NULL;
//struct process *delete=NULL;
struct process *last=NULL;
//struct process *running=NULL;
/*
 * SIGALRM handler
 */
static void
sigalrm_handler(int signum)
```

```
{
	//assert(0 && "Please fill me!");
	kill(head->mypid,SIGSTOP);
	//alarm(SCHED_TQ_SEC);

}

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

	while(1){
		p=waitpid(-1,&status,WNOHANG|WUNTRACED);
		if (p<0){
			perror("waitpid");
			exit(1);
		}
		if(p==0) break;			////////

		explain_wait_status(p,status);
		//p has the pid of our child that has changed its status

		if (WIFEXITED(status) || WIFSIGNALED(status)){//if terminated or killed by a
signal remove it from the list
			struct process* delete=NULL;
			struct process * temp;
			temp=head;

			while(temp!=NULL){
				if (temp->mypid==p && temp==head){ //if i got to delete the head
					if (temp->next == NULL){
						free(temp);
						printf("I am done\n");
						exit(0);
					}
					else{
						head=temp->next;
						free(temp);
					}
				}
				else if (temp->mypid==p && temp==last){ //if i got to delete the last
element of the list
					last=delete;
					last->next=NULL;
					free(temp);
				}
```

```
                                else if(temp->mypid==p){ //if i delete a random element in the list
                                        delete->next=temp->next;
                                        free(temp);
                                }
                                else{ //if i got to continue searching , will never access any node after
the last element
                                        delete=temp;
                                        temp=temp->next;
                                        continue;
                                }
                                break;
                        }



                }

                if (WIFSTOPPED(status)){
                        //if our process is stopped by the scheduler, continue with the next one
                        last->next=head;
                        last=head;
                        struct process * temp;
                        temp=head;
                        head=head->next;
                        temp->next=NULL;
                }
                alarm(SCHED_TQ_SEC);
                //printf("%s\n",running->name);
                kill(head->mypid,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);
        }
}



int main(int argc, char *argv[])
{
        int nproc;
        /*
         * For each of argv[1] to argv[argc - 1],
         * create a new child process, add it to the process list.
         */
        char executable[] = "prog";
      char *newargv[] = { executable, NULL, NULL, NULL };
      char *newenviron[] = { NULL };

        nproc = argc-1; /* number of proccesses goes here */
        if (nproc == 0) {
            fprintf(stderr, "Scheduler: No tasks. Exiting...\n");
            exit(1);
      }

        pid_t mypid;
        int i;
        /*Fork and create the list*/
        for (i=0;i<nproc;i++){
                mypid=fork();
                if (mypid<0){
                        printf("Error with forks\n");
                }
                if(mypid==0){
                        raise(SIGSTOP);
                        printf("I am %s, PID = %ld\n",
                 argv[0], (long)getpid());
```

```
		printf("About to replace myself with the executable %s...\n",
			executable);
		sleep(2);

		execve(executable, newargv, newenviron);

		/* execve() only returns on error */
		perror("execve");
		exit(1);
			}
		else{
			if (i==0){
				head=(struct process *)malloc(sizeof(struct process));
				if (head==NULL) printf("Error with malloc\n");
				head->mypid=mypid;
				head->next=NULL;
				head->data=i+1;
				head->name=argv[i+1];
				last=head;

			}
			else{
				newnode=(struct process *)malloc(sizeof(struct process));
				if (newnode==NULL) printf("Error with malloc\n");
				newnode->mypid=mypid;
				newnode->next=NULL;
				newnode->data=i+1;
				newnode->name=argv[i+1];
				last->next=newnode;
				last=newnode;
			}

		}


}


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

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

/*Set the alarm on*/
alarm(SCHED_TQ_SEC);

/*Start the first process*/
kill(head->mypid,SIGCONT);

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

```
                ;

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

Παρατίθενται και κάποια screenshots από την εκτέλεση του προγράμματος όπου είναι εμφανείς και οι λειτουργίες του φλοιού:

```
oslabd01@skopelos:~/Askhsh4/Askhsh1.2$ ./scheduler-shell prog prog
My PID = 6643: Child PID = 6645 has been stopped by a signal, signo = 19
My PID = 6643: Child PID = 6646 has been stopped by a signal, signo = 19
My PID = 6643: Child PID = 6644 has been stopped by a signal, signo = 19
I am ./scheduler-shell, PID = 6645
About to replace myself with the executable prog...
My PID = 6643: Child PID = 6645 has been stopped by a signal, signo = 19
I am ./scheduler-shell, PID = 6646
About to replace myself with the executable prog...
My PID = 6643: Child PID = 6646 has been stopped by a signal, signo = 19
****Welcome to the shell again****

This is the Shell. Welcome.

Shell> My PID = 6643: Child PID = 6644 has been stopped by a signal, signo = 19
prog: Starting, NMSG = 200, delay = 148
prog[6645]: This is message 0
prog[6645]: This is message 1
prog[6645]: This is message 2
prog[6645]: This is message 3
prog[6645]: This is message 4
prog[6645]: This is message 5
prog[6645]: This is message 6
My PID = 6643: Child PID = 6645 has been stopped by a signal, signo = 19
prog: Starting, NMSG = 200, delay = 130
prog[6646]: This is message 0
prog[6646]: This is message 1
prog[6646]: This is message 2
prog[6646]: This is message 3
prog[6646]: This is message 4
prog[6646]: This is message 5
prog[6646]: This is message 6
prog[6646]: This is message 7
My PID = 6643: Child PID = 6646 has been stopped by a signal, signo = 19
****Welcome to the shell again****
```

```
****Welcome to the shell again****
p
Shell: issuing request...
Shell: receiving request return value...
ID: 0 PID: 6644 Name: shell
ID: 1 PID: 6645 Name: prog
ID: 2 PID: 6646 Name: prog
Shell> My PID = 6643: Child PID = 6644 has been stopped by a signal, signo = 19
prog[6645]: This is message 7
prog[6645]: This is message 8
prog[6645]: This is message 9
prog[6645]: This is message 10
prog[6645]: This is message 11
prog[6645]: This is message 12
prog[6645]: This is message 13
My PID = 6643: Child PID = 6645 has been stopped by a signal, signo = 19
prog[6646]: This is message 8
prog[6646]: This is message 9
prog[6646]: This is message 10
prog[6646]: This is message 11
prog[6646]: This is message 12
prog[6646]: This is message 13
prog[6646]: This is message 14
prog[6646]: This is message 15
My PID = 6643: Child PID = 6646 has been stopped by a signal, signo = 19
```

```
****Welcome to the shell again****
e prog
Shell: issuing request...
Shell: receiving request return value...
Shell> My PID = 6643: Child PID = 6647 has been stopped by a signal, signo = 19
prog[6645]: This is message 14
prog[6645]: This is message 15
prog[6645]: This is message 16
prog[6645]: This is message 17
prog[6645]: This is message 18
prog[6645]: This is message 19
My PID = 6643: Child PID = 6645 has been stopped by a signal, signo = 19
prog[6646]: This is message 16
prog[6646]: This is message 17
prog[6646]: This is message 18
prog[6646]: This is message 19
prog[6646]: This is message 20
prog[6646]: This is message 21
prog[6646]: This is message 22
My PID = 6643: Child PID = 6646 has been stopped by a signal, signo = 19
I am prog, PID = 6647
About to replace myself with the executable prog...
My PID = 6643: Child PID = 6647 has been stopped by a signal, signo = 19
****Welcome to the shell again****
p
Shell: issuing request...
Shell: receiving request return value...
ID: 0 PID: 6644 Name: shell
ID: 1 PID: 6645 Name: prog
ID: 2 PID: 6646 Name: prog
ID: 3 PID: 6647 Name: prog
Shell> My PID = 6643: Child PID = 6644 has been stopped by a signal, signo = 19
prog[6645]: This is message 20
prog[6645]: This is message 21
prog[6645]: This is message 22
prog[6645]: This is message 23
prog[6645]: This is message 24
prog[6645]: This is message 25
prog[6645]: This is message 26
```

```
****Welcome to the shell again****
k 3
Shell: issuing request...
Shell: receiving request return value...
Shell> My PID = 6643: Child PID = 6647 was terminated by a signal, signo = 9
p
Shell: issuing request...
Shell: receiving request return value...
ID: 0 PID: 6644 Name: shell
ID: 1 PID: 6645 Name: prog
ID: 2 PID: 6646 Name: prog
Shell> My PID = 6643: Child PID = 6644 has been stopped by a signal, signo = 19
prog[6645]: This is message 27
prog[6645]: This is message 28
prog[6645]: This is message 29
prog[6645]: This is message 30
prog[6645]: This is message 31
prog[6645]: This is message 32
My PID = 6643: Child PID = 6645 has been stopped by a signal, signo = 19
prog[6646]: This is message 31
prog[6646]: This is message 32
prog[6646]: This is message 33
prog[6646]: This is message 34
prog[6646]: This is message 35
prog[6646]: This is message 36
prog[6646]: This is message 37
My PID = 6643: Child PID = 6646 has been stopped by a signal, signo = 19
****Welcome to the shell again****
q
Shell: Exiting. Goodbye.
My PID = 6643: Child PID = 6644 terminated normally, exit status = 0
scheduler: read from shell: Success
Scheduler: giving up on shell request processing.
prog[6645]: This is message 33
prog[6645]: This is message 34
prog[6645]: This is message 35
prog[6645]: This is message 36
prog[6645]: This is message 37
prog[6645]: This is message 38
```

```
My PID = 6643: Child PID = 6646 has been stopped by a signal, signo = 19
prog[6645]: This is message 174
prog[6645]: This is message 175
prog[6645]: This is message 176
prog[6645]: This is message 177
prog[6645]: This is message 178
prog[6645]: This is message 179
prog[6645]: This is message 180
My PID = 6643: Child PID = 6645 has been stopped by a signal, signo = 19
prog[6646]: This is message 196
prog[6646]: This is message 197
prog[6646]: This is message 198
prog[6646]: This is message 199
My PID = 6643: Child PID = 6646 terminated normally, exit status = 0
prog[6645]: This is message 181
prog[6645]: This is message 182
My PID = 6643: Child PID = 6645 has been stopped by a signal, signo = 19
prog[6645]: This is message 183
prog[6645]: This is message 184
prog[6645]: This is message 185
prog[6645]: This is message 186
prog[6645]: This is message 187
prog[6645]: This is message 188
prog[6645]: This is message 189
My PID = 6643: Child PID = 6645 has been stopped by a signal, signo = 19
prog[6645]: This is message 190
prog[6645]: This is message 191
prog[6645]: This is message 192
prog[6645]: This is message 193
prog[6645]: This is message 194
prog[6645]: This is message 195
My PID = 6643: Child PID = 6645 has been stopped by a signal, signo = 19
prog[6645]: This is message 196
prog[6645]: This is message 197
prog[6645]: This is message 198
prog[6645]: This is message 199
My PID = 6643: Child PID = 6645 terminated normally, exit status = 0
I am done. Waiting for new processes
```

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;

Πάντα όταν εκτελούμε την εντολή 'p' ως τρέχουσα διεργασία εμφανίζεται να είναι ο φλοιός (id 0). Αυτό είναι απόλυτα φυσιολογικό, καθώς η εκτύπωση των διεργασιών γίνεται μόνο όταν τρέχουσα διεργασία είναι ο φλοιός. Δηλαδή δεν θα μπορούσε να έχουμε άλλη διεργασία ως τρέχουσα διεργασία στη λίστα διεργασιών καθώς η εντολή 'p' δίνεται μόνο όταν head της λίστας μας είναι ο φλοιός.Εκείνη τη στιγμή έχουν απενεργοποιηθεί τα υπόλοιπα σήματα.

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

Οι συναρτήσεις signals_disable() και signals_enable() χρησιμοποιούνται για την απενεργοποίηση και ενεργοποίηση των σημάτων αντίστοιχα (δε γίνεται ο χειρισμός τους), όπως δηλώνεται και στο όνομά τους. Είναι απαραίτητο να χρησιμοποιηθούν αυτές οι συναρτήσεις ώστε να είμαστε σίγουροι ότι όσο εξυπηρετούνται οι αιτήσεις που κάνουμε στο φλοιό δε θα γίνει χειρισμός άλλου σήματος. 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.

1.3 Υλοποίηση προτεραιοτήτων στο χρονοδρομολογητή

Στην άσκηση αυτή απλά επεκτείνεται ο χρονοδρομολογητής του προηγούμενου ερωτήματος ώστε να υποστηρίζονται προτεραιότητες HIGH και LOW. Αν υπάρχουν διεργασίες HIGH διαφορετικά χρονοδρομολογούνται οι LOW διεργασίες και στις δύο περιπτώσεις με κυκλική επαναφορά.
Ακολουθήθηκε ο σκελετός της προηγούμενης άσκησης με τη διαφορά ότι προσθέσαμε ενα νέο πεδίο στο struct κάθε διεργασίες με όνομα priority
ώστε κάθε διεργασία να έχει την δική της προτεραιότητα η οποία είναι είτε 1 για HIGH είτε 0 για LOW Θα πρέπει να τονιστεί πως αρχικά όλες οι διεργασίες που είναι μέσα στη λίστα έχουν priority = 0 όπως επίσης και ο shell. Με την εντολή h <id> θέτω την το priority κάποιες διεργασίας στο 1(HIGH) διότι καλείται η συνάρτηση sched_set_high_p() η οποία ψάχνει να ταυτίσει το id που τις δώσαμε με το αντίσοιχο πεδίο καοιας διεργασίας και βάζει στο πεδίο priority το 1.
Αντίστοιχα η εντολή l <id> κάνει το αντίστοιχο αλλά θέτει το priority στο 0.
Επίσης κάθε ο φλιοός επιτελεί και τις λειτουργίες της προηγούμενης άσκησης με της διεργασίες που δημιουργούνται αν είναι αρχικά LOW. Επιπλέον στη συνάρτηση process_request() προστέθηκαν άλλες δυο περιπτώσεις ώστε να υποστηρίζονται οι εντολές h και l.Η σημαντικότερη αλλαγή που κάνεμε είναι στη συνάρτηση διαχείρισης του σήματος SIGCHILD την sigchld_handler() πλέον η εκπνοή κβάντου χρόνου ή ο τερματισμός διεργασίες διαχειρίζεται ως εξής :

1. Βάζουμε αυτή που έλαβε SIGSTOP στο τέλος της λίστας.

2. Ψανουμε στην λίστα να βρούμε διεργασία με priority = 1 και όσο δεν βρίσκουμε διεργασία βάζουμε κάθε διεργασία(LOW) στο τέλος της λίστας.

3. Αν βρούμε διεργασία με priority = 1 , θέτουμε τον timer και στέλνουμε SIGCONT στη διεργασία αυτή, αλλά λόγω RR στρατηγικής (εισαγωγής στο τέλος) διατηρείται η σωστή σειρά εκτέλεσης των διεργασιών.

Ο κώδικας της άσκησης αυτής φαίνεται παρακάτω :

```
#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"

/* 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 */

struct process{
        pid_t mypid;
        struct process* next;
        char* name;
        int data;
           int priority;
};
/*Definition of my list nodes*/
struct process *head=NULL;
struct process *newnode=NULL;
//struct process *delete=NULL;
struct process *last=NULL;
//struct process *running=NULL;
int i=0;
int nproc;
/* Print a list of all tasks currently being scheduled.  */
static void
sched_print_tasks(void)
{
          //assert(0 && "Please fill me!");
          struct process * temp=head;
          while (temp!=NULL){
                    printf("ID: %d PID: %i Name: %s Priority:%d\n",temp->data,temp->mypid,temp-
>name,temp->priority);
                    if (temp->next != NULL)
                              temp=temp->next;
                    else
                              break;
          }

}

/* 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!");
          struct process * temp=head;
          while (temp!=NULL){
                    if (temp->data == id ){
```

```
                        kill(temp->mypid,SIGKILL);
                        return 0;
                }
                else
                        if(temp->next!=NULL)
                                temp=temp->next;
                        else
                                break;
        }
        return -ENOSYS;
}

/*Fix the priorities*/

static int
sched_set_high_p(int id){
        struct process * temp=head;
        //struct process * prev=NULL;
        //set the priority of a desired process to high and put it in the head of my list
      while (temp!=NULL){
            if (temp->data == id ){
                  temp->priority=1;
                        return 0;
            }
            else
                  if(temp->next!=NULL){
                        temp=temp->next;
                  }
                        else{
                        break;
                        }
      }
      return -ENOSYS;
}

static int
sched_set_low_p(int id){
        struct process * temp=head;
      while (temp!=NULL){
            if (temp->data == id ){
                 temp->priority=0;
                 return 0;
            }
            else
                  if(temp->next!=NULL)
                        temp=temp->next;
                  else
                        break;
      }
      return -ENOSYS;
```

```
}



/* Create a new task.  */
static void
sched_create_task(char *executable)//
{
                //assert(0 && "Please fill me!");
                pid_t mypid;
                mypid=fork();

                //char executable[] = "prog";
        char *newargv[] = { executable, NULL, NULL, NULL };
             char *newenviron[] = { NULL };

           if (mypid<0){
                printf("Error with forks\n");
           }
           if(mypid==0){
                raise(SIGSTOP);
                printf("I am %s, PID = %ld\n",
                executable, (long)getpid());
                printf("About to replace myself with the executable %s...\n",
                      executable);
                sleep(2);

                execve(executable, newargv, newenviron);

                /* execve() only returns on error */
                perror("execve");
                exit(1);
                }
           else{
                        //insert the new node
                        newnode=(struct process *)malloc(sizeof(struct process));
                if (newnode==NULL) printf("Error with malloc\n");
                newnode->mypid=mypid;
                newnode->next=NULL;
                newnode->data=i+1;
                        newnode->priority=0;
                //strcpy(newnode->name,executable);
                newnode->name=(char*)malloc(strlen(executable)+1);
                        strcpy(newnode->name,executable);
                last->next=newnode;
                last=newnode;
                nproc++;
                i++;
           }

}
```

```
/* 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_p(rq->task_arg);
			return 0;

		case REQ_LOW_TASK:
			sched_set_low_p(rq->task_arg);
			return 0;
//
		default:
			return -ENOSYS;
	}
}

/*
 * SIGALRM handler
 */
static void
sigalrm_handler(int signum)
{
	   //assert(0 && "Please fill me!");
	kill(head->mypid,SIGSTOP);
	//alarm(SCHED_TQ_SEC);
}

/*
 * SIGCHLD handler
 */
static void
sigchld_handler(int signum)
{
	//assert(0 && "Please fill me!");
	pid_t p;
	int status;
	   int flag=0;
	while(1){
```

```
p=waitpid(-1,&status,WNOHANG|WUNTRACED);
if (p<0){
	perror("waitpid");
	exit(1);
}
if(p==0) break;		////////

explain_wait_status(p,status);
//p has the pid of our child that has changed its status

if (WIFEXITED(status) || WIFSIGNALED(status)){//if terminated or killed by a signal remove it from the list
	struct process* delete=NULL;
	struct process * temp;
	temp=head;

	while(temp!=NULL){
		if (temp->mypid==p && temp==head){ //if i got to delete the head
			if (temp->next == NULL){
				free(temp);
				printf("I am done. Waiting for new processes\n");
				//exit(0);
			}
			else{
				head=temp->next;
				free(temp);
			}
		}
		else if (temp->mypid==p && temp==last){ //if i got to delete the last element of the list
			last=delete;
			last->next=NULL;
			free(temp);
		}
		else if(temp->mypid==p){ //if i delete a random element in the list
			delete->next=temp->next;
			free(temp);

		}
		else{ //if i got to continue searching , will never access any node after the last element
			delete=temp;
			temp=temp->next;
			continue;
		}
		break;
	}
	  temp=head;
	  struct process *current = head;
	  //struct process * prev = NULL;
	//search for high priorities
	while (temp!=NULL){
```

```
                    if (temp->priority!=1){ //move this node to the end
                    last->next=head;
                        last=head;
                        head=head->next;
                        last->next=NULL;
                        temp=head;
                        if (temp == current) break;
                              continue;
                        }
                        else{
                              flag=1;
                              break;
                      }
                    }


        }

        if (WIFSTOPPED(status)){
              //if our process is stopped by the scheduler, continue with the next one according to
priorities
                    if (head==NULL) {printf("Empty list\n"); exit(0);}
                    if (head->next != NULL){
                              last->next=head;//send my current process to the end
                    last=head;
                              head=head->next;
                              last->next=NULL;
                              struct process * current = head;
                    struct process * temp = head;
                              //struct process * prev = NULL;
                              //search for high priorities
                              while (temp!=NULL){
                                        if (temp->priority!=1){ //move this node to the end
                                                  last->next=head;
                                                  last=head;
                                                  head=head->next;
                                                  last->next=NULL;
                                                  temp=head;
                                                  if (temp == current) break;
                                                  continue;
                                        }
                                        else{
                                                  flag=1;
                                                  break;
                                        }


                              }
                    }


        }
```

```
                if (WIFSTOPPED(status)){
                        if(head->data==0 && (head->priority == 1 || flag == 0)){
                                printf("****Welcome to the shell again****\n");
                                //printf("Shell>");
                                alarm(6);
                        }
                        else
                        alarm(SCHED_TQ_SEC);
        }
           kill(head->mypid,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 void
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];
	//insert shell as head of my list
	head->data=0;
	head->mypid=p;
	head->priority=0;
	head->name="shell";
	head->next=NULL;
}

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[])
{

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

        head=(struct process *)malloc(sizeof(struct process));
        last=head;
        /* Create the shell. */
        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 */

        char executable[] = "prog";
      char *newargv[] = { executable, NULL, NULL, NULL };
      char *newenviron[] = { NULL };

      nproc = argc-1; /* number of proccesses goes here */
      if (nproc < 0) {
            fprintf(stderr, "Scheduler: No tasks. Exiting...\n");
            exit(1);
      }

      pid_t mypid;
```

```
//int i;
/*Fork and create the list*/
for (i=0;i<nproc;i++){
	mypid=fork();
	if (mypid<0){
		printf("Error with forks\n");
	}
	if(mypid==0){
		raise(SIGSTOP);
		printf("I am %s, PID = %ld\n",
		argv[0], (long)getpid());
		printf("About to replace myself with the executable %s...\n",
			executable);
		sleep(2);

		execve(executable, newargv, newenviron);

		/* execve() only returns on error */
		perror("execve");
		exit(1);
		}
	else{
		if (head==NULL){//will never enter though
			//printf("lala\n");
				head=(struct process *)malloc(sizeof(struct process));
			if (head==NULL) printf("Error with malloc\n");
			head->mypid=mypid;
			head->next=NULL;
			head->data=i+1;
				head->priority=0;
			head->name=argv[i+1];
				last=head;
		}
		   else{
			newnode=(struct process *)malloc(sizeof(struct process));
			if (newnode==NULL) printf("Error with malloc\n");
			newnode->mypid=mypid;
			newnode->next=NULL;
			newnode->data=i+1;
				newnode->priority=0;
			newnode->name=argv[i+1];
			last->next=newnode;
			last=newnode;
		}
	}


}


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

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

    /*Set the alarm on*/
    alarm(SCHED_TQ_SEC);

    /*Start the first process*/
        kill(head->mypid,SIGCONT); //wake up the shell

        shell_request_loop(request_fd, return_fd);

        //kill(head->mypid,SIGCONT);

        /* 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;
}
```

Μερικά screenshots κατά την εκτέλεση:

```
oslabd01@skopelos:~/Askhsh4/Askhsh1.3$ ./scheduler-shell prog
My PID = 6664: Child PID = 6666 has been stopped by a signal, signo = 19
My PID = 6664: Child PID = 6665 has been stopped by a signal, signo = 19
I am ./scheduler-shell, PID = 6666
About to replace myself with the executable prog...
My PID = 6664: Child PID = 6666 has been stopped by a signal, signo = 19
****Welcome to the shell again****

This is the Shell. Welcome.

Shell> p
Shell: issuing request...
ID: 0 PID: 6665 Name: shell Priority:0
ID: 1 PID: 6666 Name: prog Priority:0
Shell: receiving request return value...
Shell> My PID = 6664: Child PID = 6665 has been stopped by a signal, signo = 19
prog: Starting, NMSG = 200, delay = 152
prog[6666]: This is message 0
prog[6666]: This is message 1
prog[6666]: This is message 2
prog[6666]: This is message 3
prog[6666]: This is message 4
prog[6666]: This is message 5
prog[6666]: This is message 6
My PID = 6664: Child PID = 6666 has been stopped by a signal, signo = 19
****Welcome to the shell again****
h 0
Shell: issuing request...
Shell: receiving request return value...
Shell> My PID = 6664: Child PID = 6665 has been stopped by a signal, signo = 19
****Welcome to the shell again****
h 1
Shell: issuing request...
Shell: receiving request return value...
Shell> p
Shell: issuing request...
Shell: receiving request return value...
ID: 0 PID: 6665 Name: shell Priority:1
ID: 1 PID: 6666 Name: prog Priority:1
```

```
Shell> My PID = 6664: Child PID = 6665 has been stopped by a signal, signo = 19
prog[6666]: This is message 7
prog[6666]: This is message 8
prog[6666]: This is message 9
prog[6666]: This is message 10
prog[6666]: This is message 11
prog[6666]: This is message 12
My PID = 6664: Child PID = 6666 has been stopped by a signal, signo = 19
****Welcome to the shell again****
l 1
Shell: issuing request...
Shell: receiving request return value...
Shell> p
Shell: issuing request...
Shell: receiving request return value...
ID: 0 PID: 6665 Name: shell Priority:1
ID: 1 PID: 6666 Name: prog Priority:0
Shell> My PID = 6664: Child PID = 6665 has been stopped by a signal, signo = 19
****Welcome to the shell again****
e prog
Shell: issuing request...
Shell: receiving request return value...
Shell> My PID = 6664: Child PID = 6667 has been stopped by a signal, signo = 19
****Welcome to the shell again****
p
Shell: issuing request...
Shell: receiving request return value...
ID: 0 PID: 6665 Name: shell Priority:1
ID: 1 PID: 6666 Name: prog Priority:0
ID: 2 PID: 6667 Name: prog Priority:0
Shell> k 1
Shell: issuing request...
Shell: receiving request return value...
Shell> My PID = 6664: Child PID = 6666 was terminated by a signal, signo = 9
p
Shell: issuing request...
Shell: receiving request return value...
ID: 0 PID: 6665 Name: shell Priority:1
ID: 2 PID: 6667 Name: prog Priority:0
Shell> My PID = 6664: Child PID = 6665 has been stopped by a signal, signo = 19
```

```
****Welcome to the shell again****
h 2
Shell: issuing request...
Shell: receiving request return value...
Shell> My PID = 6664: Child PID = 6665 has been stopped by a signal, signo = 19
I am prog, PID = 6667
About to replace myself with the executable prog...
My PID = 6664: Child PID = 6667 has been stopped by a signal, signo = 19
****Welcome to the shell again****
My PID = 6664: Child PID = 6665 has been stopped by a signal, signo = 19
prog: Starting, NMSG = 200, delay = 133
prog[6667]: This is message 0
prog[6667]: This is message 1
prog[6667]: This is message 2
prog[6667]: This is message 3
prog[6667]: This is message 4
prog[6667]: This is message 5
prog[6667]: This is message 6
prog[6667]: This is message 7
My PID = 6664: Child PID = 6667 has been stopped by a signal, signo = 19
****Welcome to the shell again****
p
Shell: issuing request...
Shell: receiving request return value...
ID: 0 PID: 6665 Name: shell Priority:1
ID: 2 PID: 6667 Name: prog Priority:1
Shell> l 2
Shell: issuing request...
Shell: receiving request return value...
Shell> p
Shell: issuing request...
Shell: receiving request return value...
ID: 0 PID: 6665 Name: shell Priority:1
ID: 2 PID: 6667 Name: prog Priority:0
Shell> kMy PID = 6664: Child PID = 6665 has been stopped by a signal, signo = 19
****Welcome to the shell again****
k 2
```

```
Shell: issuing request...
Shell: receiving request return value...
Shell> My PID = 6664: Child PID = 6667 was terminated by a signal, signo = 9
My PID = 6664: Child PID = 6665 has been stopped by a signal, signo = 19
****Welcome to the shell again****
q
Shell: Exiting. Goodbye.
My PID = 6664: Child PID = 6665 terminated normally, exit status = 0
I am done. Waiting for new processes
```

1. Περιγράψτε ένα σενάριο δημιουργίας λιμοκτονίας.

Ένα σεναριο λιμοκτονίας για το χρονοδρομολογητή μας είναι το εξής: Αν εχουμε μια διεργασία η οποία έχει low priority και δεν αλλάξει ΠΟΤΕ η προτεραιότητά της και πάντα έχουμε διεργασίες με high priority. Σε μια τετοια περίπτωση, σύμφωνα με το scheduler και την υλοποίηση που έχουμε κάνει η διεργασία αυτή δε θα εκτελεστεί ποτέ καθώς πάντα εκτελούνται (όσο υπάρχουν) διεργασίες με high priority. Κάτι τέτοιο δεν ειναι σε καμια περίπτωση επιθυμητό. Ένας τρόπος που θα μπορούσε να αποφευχθεί αυτό το ενδεχόμενο είναι να προσθέσουμε στο struct που αντιπροσωπεύει κάθε διεργασία ένα πεδίο age που θα εκφράζει πόσα κβάντα χρόνου η διεργασία αυτή ειναι σε αναμονή (κάθε φορα που επιλέγεται μια διεργασία προς εκτέλεση, το πεδίο αυτό των άλλων διεργασιών αυξάνεται κατά ένα και αν επιλεγεί μια διεργασία, το πεδίο αυτό_μηδενίζεται). Κατά αυτόν τον τρόπο όταν το πεδίο age μιας διεργασίας ξεπεράσει μια προκαθορισμένη τιμή τότε, χωρίς να λαμβάνεται υπόψην αν είναι high ή low η προτεραιότητά της, η εν λόγω διεργασία θα εκτελείται και θα πάψει λοιπόν να υπάρχει αυτό το σενάριο_λιμοκτονίας.