ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

Λειτουργία Συστήματα Υπολογιστών

Αναφορά για την 4η Εργαστηριακή Άσκηση: Χρονοδρομολόγηση

Ον/μο :Μεσσής Κωνσταντίνος
Α.Μ.: :03111122

Ον/μο :Μπερέτσος Θεόδωρος
Α.Μ.: :03111612

Ημερομηνία παράδοσης: 18/3/2015

4.1 Υλοποίηση χρονοδρομολογητή κυκλικής επαναφοράς στο χώρο χρήστη

-Πηγαίος κώδικας (source code)

scheduler_solution.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"

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


//////////////////////
//struct definition//
//////////////////////

typedef struct node {
       pid_t pid;
       int id;
       char procname[TASK_NAME_SZ];
       struct node *previous;
       struct node *next;
}queue;

//global variables
queue *baton;
int remain_proc;

queue *insert_to_queue(pid_t pid, int id, queue *last, char *execname) {
       if (last->next == NULL) {
               last->previous = last;
               last->next = last;
         }
       else{
              queue *new = (queue*)malloc(sizeof(queue));
                     if (!new){
                     perror("queue : no memory!");
                     exit(1);
              }
              queue *temp = last->previous;
               //connect new node with the last double-connected
               //          _____________________________
               // [new]--'->[last]<->....<->[head]<-'
               new->next = last;
               // connect the last double-connected with the new node
               //          _____________________________
              // [new]<-'->[last]<-->...<->[head]<-'
```

```
                last->previous = new;
                // connect new node with head i.e. close the cycle
                //  ____________________________________
               // '->[new]<-->[last]<-->...<->[head]<-'
                new->previous = temp;
                temp->next = new;
               // connection done ! make new node the last connected
                last = new;
        }
        // write process p(id) [same for both cases]
        last->pid = pid;
        last->id = id;
        strcpy(last->procname, execname);
        return last;
}

queue *remove_from_queue(queue *pointer) {
    if (pointer == pointer->next) {

    }
    queue *temp = pointer->previous;
    (pointer->previous)->next = pointer->next;
    (pointer->next)->previous = pointer->previous;
    pointer->next = NULL;
    pointer->previous = NULL;
    free(pointer);
    return temp;
}

queue *find_pid_queue(queue *pointer ,int pid) {
    queue *temp = pointer;
    while (temp->pid != pid) {
       temp = temp->next;
    }
    return temp;
}

/*
 * SIGALRM handler
 */
static void
sigalrm_handler(int signum)
{
        if (signum != SIGALRM) {
                fprintf(stderr, "Internal error: Called for signum %d, not
SIGALRM\n",
                  signum);
                exit(1);
        }
        if (baton) {
                kill(baton->pid, SIGSTOP);
                printf("\n");
                printf("\t [Alarm]: Time is Up!Quantum has elapsed!\n\n");
        }

//apomama     assert(0 && "Please fill me!");
}

void update(int msg) {
    remain_proc -= 1;
    baton = remove_from_queue(baton);
    switch (msg) {
```

```
        case 0 :
               printf("successfully terminated within time quantum !\n");
               break;
            case 1 :
               printf("and has been removed from RR-Queue!\n");
               break;
            default:
               printf(" unreachable switch case option\n");
               exit(1);
        }
    if (remain_proc) {
         kill(baton->pid, SIGCONT);
         alarm(SCHED_TQ_SEC);
         printf("\t [Scheduler]: Advancing to next process...\n");
    }
    else {
         printf("\t [Scheduler]: All tasks finished. Exiting...\n");
         exit(1);
    }
}

/*
 * SIGCHLD handler
 */
static void
sigchld_handler(int signum)
{
        pid_t p;
        int status;
        queue *temp ;
        if (signum != SIGCHLD) {
               fprintf(stderr, "Internal error: Called for signum %d, not
SIGCHLD\n",
                 signum);
               exit(1);
        }
        p = waitpid(-1, &status, WUNTRACED|WCONTINUED);
        if (p < 0) {
               perror("sigchld handler: waitpid");
                exit(1);
        }
        temp = find_pid_queue(baton,p);
        printf("\t [Scheduler]: Process PNAME: %s  PID: %d  ID: %d ", temp-
>procname, temp->pid, temp->id);
        if (WIFCONTINUED(status)) {
               printf("continues!\n\n");
               return;
        }
        if (WIFEXITED(status)) {
               /*  Current process has finished within time */
                update(0);
        }
        else if (WIFSIGNALED(status)) {
               if (WTERMSIG(status) == 9) {
                      printf("was killed ");
                       if (baton->pid == p) {
                             update(1);
                      }
                      else {
                             remain_proc -= 1;
                             temp = remove_from_queue(temp);
                             printf("and has been removed from RR-Queue!\n");
```

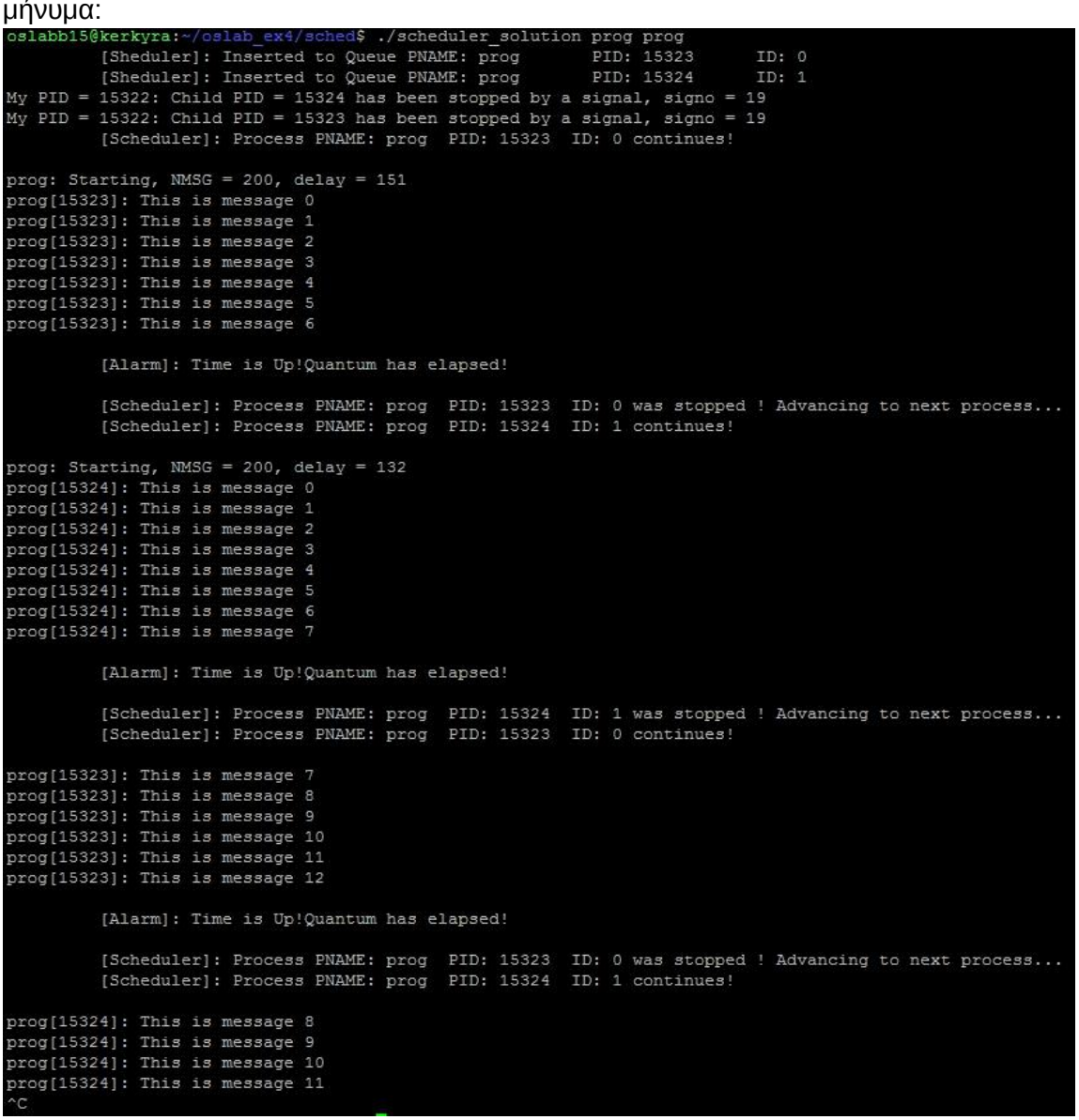
```
                        }
                }
        }
        else if (WIFSTOPPED(status)) {
                /* Current process needs more time */
                 baton = baton->previous;
                 kill(baton->pid, SIGCONT);
                alarm(SCHED_TQ_SEC);
                printf("was stopped ! Advancing to next process...\n");
        }

//apomama     assert(0 && "Please fill me!");
}

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

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

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

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


void create_execve(char *exec) {
        char *newargv[] = {exec, NULL, NULL, NULL };
        char *newenviron[] = {NULL};
        raise(SIGSTOP);
        execve(exec, newargv, newenviron);
        /* execve() only returns on error */
        perror("execve");
        exit(1);
}
```

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

	int i;
	pid_t p;
	queue *head = (queue *)malloc(sizeof(queue));
	baton = head;
	head->next = NULL;
	head->previous = NULL;

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

//=>apomama   nproc = 0; /* number of proccesses goes here */
	nproc = argc - 1;
	remain_proc = nproc; /*number of remaining proccesses goes here*/


	for (i = 0; i < nproc; i++) {
		 if ((p = fork()) < 0){
			perror("[Scheduler]: fork could not be done");
			exit(1);
		}
		if (p == 0) {
			/* Child Code : Process */
			create_execve(argv[i + 1]);
			fprintf(stderr,"Error ! unreachable point in child's
code\n");
			exit(0);
		}
		 /* Parent's Code : Scheduler */
		baton = insert_to_queue(p, i, baton, argv[i+1]);
		printf("\t [Sheduler]: Inserted to Queue PNAME: %s\t PID: %d\t ID:
%d\n",baton->procname,baton->pid,baton->id);
	}

	if (nproc == 0) {
		printf("\t [Scheduler]: No tasks. Exiting...\n");
		exit(1);
	}

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

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

	baton = head;
	kill(baton->pid, SIGCONT);
	alarm(SCHED_TQ_SEC);

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

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

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

-Έξοδο εκτέλεσης του προγράμματος

Η εντολή `./scheduler_solution prog prog` έχει ως έξοδο στην οθόνη το εξής μήνυμα:



-Σύντομες απαντήσεις στις ερωτήσεις

Ερώτηση1: 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; Υπόδειξη: μελετήστε τη συνάρτηση `install_signal_handlers()` που δίνεται.

Απάντηση:

Αν παρατηρήσει κανείς τον κώδικα της `install_signal_handlers` θα δει πως δηλώνεται ένα `struct sa` τύπου `sigaction`. Τα `sigaction structs` μας επιτρέπουν παραπάνω λειτουργίες στον χειρισμό σημάτων. Μία από αυτές είναι ο `sa_handler` ο οποίος αποτελεί ένα δείκτη στην συνάρτηση χειρισμού του σήματος και μια άλλη είναι η `sa_mask` η οποία περιέχει ένα σετ σημάτων που δεν θα εξυπηρετηθούν όσο τρέχει η συνάρτηση που πιάνει το 1ο σήμα που πρέπει να εξυπηρετηθεί. Έτσι στην υλοποίησή μας κάθε `signal handler` προστατεύεται από το άλλο σήμα που εμφανίζεται στην άσκηση. Άρα όταν καταφθάνει ένα εκ των SIGCHLD, SIGALRM και εξυπηρετείται ένα άλλο, τότε αυτό που ήρθε τελευταίο δε θα εξυπηρετηθεί.

Ένας πραγματικός χρονοδρομολογητής (π.χ. των Linux με τις συναρτήσεις `signal_handle()` και `do_signal()`) κάνει το ίδιο πράγμα μπλοκάρωντας σήματα που έρχονται, ενώ εξυπηρετείται κάποιο άλλο, αλλά ταυτόχρονα προστατεύει το αποτέλεσμα σε κλήσεις συστήματος που πιθανώς έτρεχαν όταν ήρθε το σήμα προς εξυπηρέτηση.

Ερώτηση 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;

Απάντηση:

Όταν τερματίζεται μια διεργασία αποστέλλεται στον πατέρα αυτής SIGCHLD. Έτσι ο πατέρας (ο χρονοδρομολογητής εδώ) πιάνει το σήμα και εξυπηρετεί την συνάρτηση στην οποία δείχνει ο sig_handler της SIGCHLD . Το λογικό είναι αυτό να αναφέρεται σε τερματισμό είτε βεβιασμένο είτε φυσιολογικό της τρέχουσας διεργασίας.

```
if (WTERMSIG(status) == 9) { //SIGKILL == no.9//
            printf("was killed ");
            if (baton->pid == p){
                  update(1);
            }
            else {
                  temp = remove_from_queue(temp);
                  printf("and has been removed from RR-Queue!\n");
            }
 }
```

Από το παραπάνω κομμάτι κώδικα που εμπεριέχεται στον handler της SIGCHLD φαίνεται ότι αν η κλήση της συνάρτησης έχει προέλθει από εξωτερικό τερματισμό διεργασίας διαφορετικής της τρέχουσας εμφανίζεται κατάλληλο μήνυμα και αυτή αφαιρείται από τη λίστα . Έπειτα συνεχίζει να εξυπηρετείται η τρέχουσα διεργασία.

Ερώτηση 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; Υπόδειξη: Η παραλαβή του σήματος SIGCHLD εγγυάται ότι η τρέχουσα διεργασία έλαβε το σήμα SIGSTOP και έχει σταματήσει.

Απάντηση:

Το σήμα SIGALRM αποστέλλεται κάθε φορά που τελειώνει το κβάντο χρόνου. Στην δική μας υλοποίηση ο χειρισμός του σήματος αυτού περιλαμβάνει τερματισμό της τρέχουσας διεργασίας με SIGSTOP και η εφαρμογή περεταίρω κινήσεων για την εμφάνιση μηνυμάτων και την εκκίνηση της επόμενης διεργασίας με αποστολή σήματος SIGCONT ανατίθεται στην συνάρτηση χειρισμού του SIGCHLD. Δεδομένου ότι η προκαθορισμένη ενέργεια του πατέρα στο τελευταίο σήμα είναι να το αγνοήσει αν δεν υπήρχε συνάρτηση χειρισμού για αυτό, δεν θα υπήρχε και εναλλαγή διεργασιών . Πέραν όμως του προβλήματος αυτού που συμβαίνει στην δική μας υλοποίηση στην γενική περίπτωση αν μια διεργασία τερματίσει τη λειτουργία της πριν εκπνεύσει το κβάντο χρόνου τότε αποστέλλεται με τη μια SIGCHLD πριν το SIGALRM . Στην περίπτωση αυτή έχει νόημα άμεσος χειρισμός του σήματος αυτού τόσο για εξοικονόμηση χρόνου στον χρονοδρομολογητή (χειρισμός άμεσα της επόμενης διεργασίας πριν εκπνεύσει το κβάντο χρόνου) όσο και για αποφυγή ενδεχόμενης αποστολής SIGSTOP σε ήδη τερματισμένη διεργασία που θα επέφερε προβλήματα.

4.2 Έλεγχος λειτουργίας χρονοδρομολογητή μέσω φλοιού

-Πηγαίος κώδικας (source code)

scheduler-shell_solution.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"

/* 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 definition//
////////////////////

typedef struct node{
```

```
    pid_t pid ;
    int id, priority;
    char procname [TASK_NAME_SZ];
    struct node *previous;
    struct node *next;
}queue;

//global variables
queue *baton;
int remain_proc, num_high = 0, max_proc;
pid_t pid_shell;
const int PRIORITY_LOW = 0;
const int PRIORITY_HIGH = 1;
//global end

queue *insert_to_queue(pid_t pid, int id, int priority, queue *last, char
*execname) {

    if (last->next == NULL) {
              //insertion of the first process : create self-loop cycle
        last->previous = last;
        last->next = last ;
    }
    //queue has at least one process already inserted!
    else {
       queue *new = (queue *)malloc(sizeof(queue));
        if (!new) {
              perror("queue : no memory!");
              exit(1);
              }
       queue *temp = last->previous;
        //connect new node with the last double-connected
        //         _____________________________
        // [new]--'->[last]<->....<->[head]<-'
        new->next = last;
        // connect the last double-connected with the new node
        //               _____________________________
        // [new]<-'->[last]<-->...<->[head]<-'
        last->previous = new;
        // connect new node with head i.e. close the cycle
        //  ____________________________________
        // '->[new]<-->[last]<-->...<->[head]<-'
        new->previous = temp;
        temp->next = new;
        // connection done ! make new node the last connected
        last = new;
    }
    // write process p(id) [same for both cases]
    last->pid = pid;
    last->id = id ;
    strcpy(last->procname, execname);
    return last;
}

queue *remove_from_queue(queue *pointer) {
//    if (pointer == pointer->next) {
//
//    }
    queue *temp = pointer->previous;
    (pointer->previous)->next = pointer->next;
    (pointer->next)->previous = pointer->previous;
    pointer->next = NULL;
```

```
    pointer->previous = NULL;
    if (pointer->priority == PRIORITY_HIGH)
       num_high -= 1;
    free(pointer);
    remain_proc -= 1;
    return temp;
}

queue *find_pid_queue(queue *pointer, int pid) {
       queue *temp = pointer;
       while (temp->pid != pid) {
              temp = temp->next;
       }
    return temp;
}

void print_queue(queue *head, int num){
    int i = 0;
    queue *temp = head;
    for (i = 0; i < num; i++) {
        printf("PID:%d\tID:%d\n", temp->pid, temp->id);
        temp = temp->next ;
    }
}

/*Print a list of all tasks currently being scheduled.  */
static void
sched_print_tasks(void)
{
       printf("\t [Scheduler]: The following processes are currently running:\n");
       print_queue(baton, remain_proc + 1);
       printf("\t The ongoin process has process id: %d",baton->id);

//     assert(0 && "Please fill me!");
}

/* Send SIGKILL to a task determined by the value of its
 * scheduler-specific id.
 */
static int
sched_kill_task_by_id(int id)
{
       queue *temp = baton;
       int id2 = temp->id;
       while (temp->id != id) {
              temp = temp->next;
              if (temp->id == id2)
                     break;
       }
       if (temp->id == id) {
              printf("\t Killing process with id %i.\n", id);
              kill(temp->id, SIGKILL);
              remove_from_queue(temp);
       }
       return 0;

//ssert(0 && "Please fill me!");
//eturn -ENOSYS;
}

static void
sched_high_priority(int id)
```

```
{
	if (id != 0){
		queue *temp = baton;
		int id2 = temp->id;
		while (temp->id != id) {
				temp = temp->next;
			if (temp->id == id2)
				break;
		}
		if ((temp->id == id) && (temp->priority == PRIORITY_LOW)) {
			printf("\t UPgrading the priority of process with id %i.\n",
id);
			num_high += 1;
			temp->priority = PRIORITY_HIGH;

		}
	}
}

static void
sched_low_priority(int id)
{
	queue *temp = baton;
	int id2 = temp->id;
	while (temp->id != id) {
			temp = temp->next;
		if (temp->id == id2)
			break;
	}
	if ((temp->id == id) && (temp->priority == PRIORITY_HIGH)) {
		printf("\t  DOWNgrading the priority of process with id %i.\n", id);
		num_high -= 1;
		 temp->priority = PRIORITY_LOW;
	}
}


/* Create a new task.  */
static void
sched_create_task(char *executable)
{
	char *newargv[] = {executable, NULL, NULL, NULL};
	char *newenviron[] = {NULL};
	pid_t pid;
	pid = fork();
	if (pid < 0) {
		perror("Create_new_task: fork could not be done!");
		exit(1);
	}
	if (pid == 0) {
		/* Child */
		raise(SIGSTOP);
		execve(executable, newargv, newenviron);
		exit(1);
	}
	remain_proc += 1;
	max_proc += 1;
	insert_to_queue(pid, max_proc, 0, baton, executable);

//	assert(0 && "Please fill me!");
}
```

```c
/* 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:
                     printf("Killing in the name of...\n");
                     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_high_priority(rq->task_arg);
                      return 0;

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

              default:
                     return -ENOSYS;
       }
}

/*
 * SIGALRM handler
 */
static void
sigalrm_handler(int signum)
{
    if (signum != SIGALRM) {
              fprintf(stderr, "Internal error: Called for signum %d, not
SIGALRM\n", signum);
              exit(1);
       }
    if (baton) {
        kill(baton->pid, SIGSTOP);
        printf("\n [Alarm]: Time is Up!Quantum has elapsed!\n\n");
       alarm(SCHED_TQ_SEC);
    }

//     assert(0 && "Please fill me!");
}

void update(int msg){
//    remain_proc -= 1;
    queue *temp2;
    baton = remove_from_queue(baton);
    switch (msg) {
           case 0 :
                     printf("successfully terminated within time quantum !\n");
                     break;
           case 1 :
                     printf("and has been removed from RR-Queue!\n");
                     break;
           default:
```

```
                    printf(" unreachable switch case option\n");
                    exit(1);
    }
    if (remain_proc >= 0) {
       temp2 = baton;
       while((temp2->id !=0) && (temp2->priority != PRIORITY_HIGH) && (num_high !=
0)){
             temp2 = temp2->next;
       }
        kill(baton->pid, SIGCONT);
        alarm(SCHED_TQ_SEC);
        printf("[Scheduler]: Advancing to next process...\n");
    }
    else {
        printf("[Scheduler]: All tasks finished. Exiting...\n");
        exit(1);
    }
}

/*
 * SIGCHLD handler
 */
static void
sigchld_handler(int signum)
{
       pid_t p;
       int status;
       queue *temp, *temp2;
       if (signum != SIGCHLD) {
              fprintf(stderr, "Internal error: Called for signum %d, not
SIGCHLD\n", signum);
              exit(1);
       }
       p = waitpid(-1, &status, WUNTRACED | WCONTINUED);
       if (p < 0) {
              perror("sigchld handler: waitpid");
               exit(1);
       }
       temp = find_pid_queue(baton, p);
       printf(" [Scheduler]: Process PNAME: %s  PID: %d  ID: %d ", temp-
>procname,temp->pid,temp->id);
       if (WIFCONTINUED(status)) {
              printf("continues!\n\n");
              return;
       }
       if (WIFEXITED(status)) {
              /*  Current process has finished within time */
               update(0);
       }
       else if (WIFSIGNALED(status)) {
              if (WTERMSIG(status) == 9) {
                     printf("was killed ");
                       if (baton->pid == p){
                            update(1);
                            }
                     else {
       //            remain_proc -= 1;
                     temp = remove_from_queue(temp);
                     printf("and has been removed from RR-Queue!\n");
                     }
              }
       }
```

```
        else if (WIFSTOPPED(status)) {
                /* Current process needs more time */
                temp2 = baton;
                temp2 = temp2->next;
                while((temp2->id !=0) && (temp2->priority != PRIORITY_HIGH) &&
(num_high != 0)){
                        temp2 = temp2->next;
                }
                baton = temp2;
                 kill(baton->pid, SIGCONT);
                 alarm(SCHED_TQ_SEC);
                 printf("was stopped ! Advancing to next process...\n");
        }


//      assert(0 && "Please fill me!");
}

/* 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 */
		//pid_shell = getpid();

		close(pfds_rq[0]);
		close(pfds_ret[1]);
		do_shell(executable, pfds_rq[1], pfds_ret[0]);
		assert(0);
	}
	pid_shell = getpid();
	/* Parent */
	close(pfds_rq[1]);
	close(pfds_ret[0]);
	*request_fd = pfds_rq[0];
	*return_fd = pfds_ret[1];

	return p;
}

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

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

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

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

void create_execve(char *exec){
	char *newargv[] = {exec, NULL, NULL, NULL };
	char *newenviron[] = {NULL};
	raise(SIGSTOP);
	execve(exec, newargv, newenviron);
	/* execve() only returns on error */
	perror("execve");
```

```
        exit(1);
}


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

        int i;
        pid_t p;
        queue *head = (queue *)malloc(sizeof(queue));
        baton = head;
        head->next = NULL;
        head->previous = NULL;
        /* Two file descriptors for communication with the shell */
        static int request_fd, return_fd;

        /* Create the shell. */
        p = sched_create_shell(SHELL_EXECUTABLE_NAME, &request_fd, &return_fd);
        /* TODO: add the shell to the scheduler's tasks */
        baton = insert_to_queue(p, 0, PRIORITY_LOW, baton, "shell");
        printf(" [Sheduler]: Inserted to Queue PNAME: shell\t PID: %d\t ID: 0\n",
p);

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

//apomama     nproc = 0; /* number of proccesses goes here */
        nproc = argc - 1;
        remain_proc = nproc;
        max_proc = nproc;

        /* Wait for all children to raise SIGSTOP before exec()ing. */
        for (i = 0; i < nproc ; i++) {
                if ((p = fork()) < 0) {
                        perror("[Scheduler]: fork could not be done");
                        exit(1);
                }
                if (p == 0) {
                        /* Child Code : Process */
                        create_execve(argv[i+1]);
                        fprintf(stderr,"Error ! unreachable point in child's
code\n");
                        exit(0);
                }
                 /* Parent's Code : Scheduler */
                baton = insert_to_queue(p, i + 1, PRIORITY_LOW, baton, argv[i+1]);
                printf(" [Sheduler]: Inserted to Queue PNAME: %s\t PID: %d\t ID:
%d\n", baton->procname, baton->pid, baton->id);
        }

        wait_for_ready_children(nproc);

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

        baton = head;
        kill(baton->pid, SIGCONT);
        alarm(SCHED_TQ_SEC);
```

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

        shell_request_loop(request_fd, return_fd);

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

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

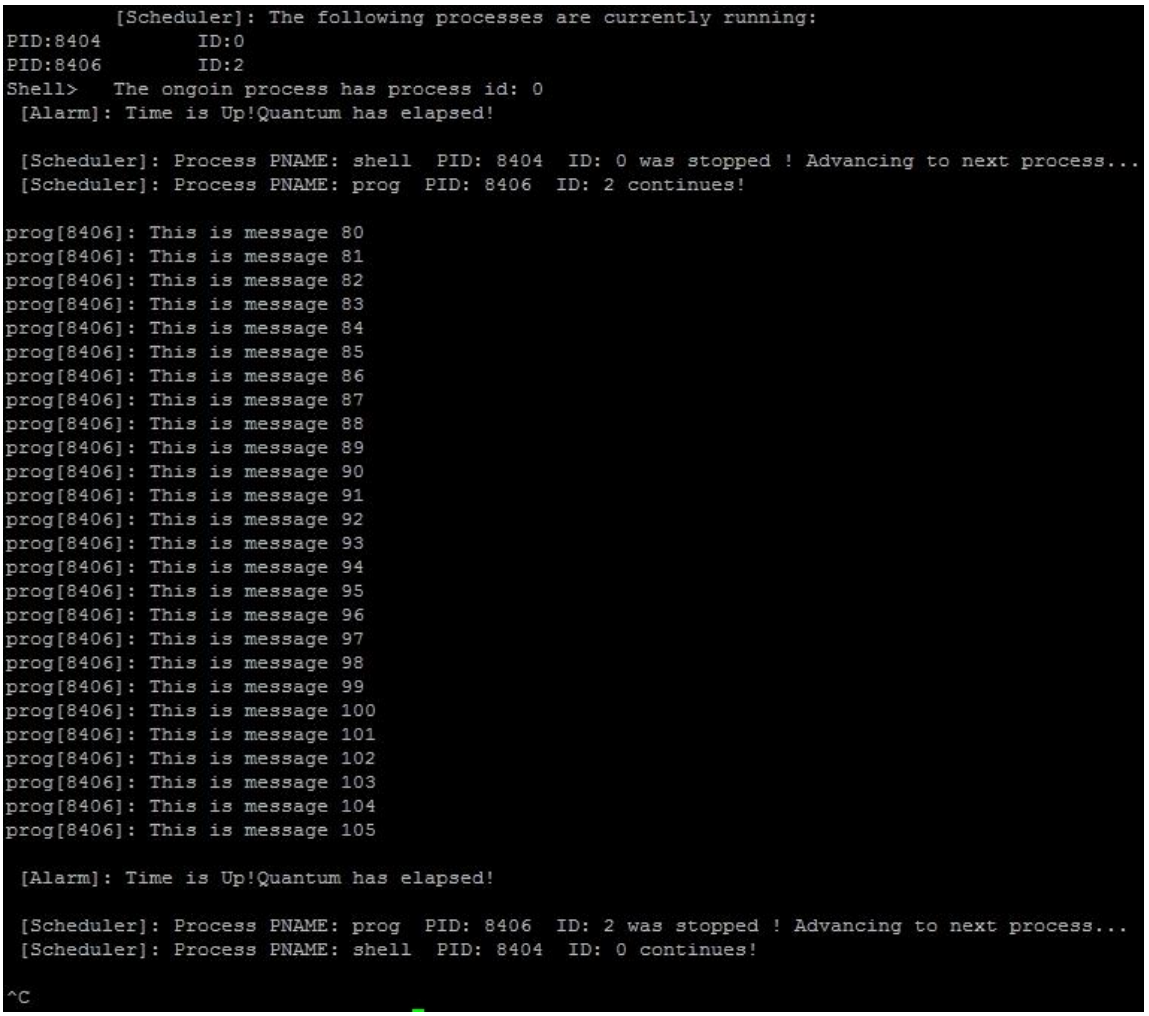
-Έξοδο εκτέλεσης του προγράμματος

Η εντολή `./scheduler-shell_solution prog prog` έχει ως έξοδο τα παρακάτω screenshots. Αυτό που παρουσιάζεται είναι η εκτύπωση αρχικά με την εντολή `p` στον shell των τρεχόντων διεργασιών. Στη συνέχεια τερματίζουμε μία διεργασία με την εντολή `k 1` και τέλος εκτυπώνουμε ξανά τις τρέχουσες διεργασίες.

```
oslabb15@anafi:~/oslab_ex4/sched$ ./scheduler-shell_solution prog prog
 [Sheduler]: Inserted to Queue PNAME: shell      PID: 8404       ID: 0
 [Sheduler]: Inserted to Queue PNAME: prog       PID: 8405       ID: 1
 [Sheduler]: Inserted to Queue PNAME: prog       PID: 8406       ID: 2
My PID = 8403: Child PID = 8405 has been stopped by a signal, signo = 19
My PID = 8403: Child PID = 8406 has been stopped by a signal, signo = 19
 [Scheduler]: Process PNAME: shell  PID: 8404  ID: 0 was stopped ! Advancing to next process...
 [Scheduler]: Process PNAME: prog  PID: 8406  ID: 2 continues!

prog: Starting, NMSG = 200, delay = 32
prog[8406]: This is message 0
prog[8406]: This is message 1
prog[8406]: This is message 2
prog[8406]: This is message 3
prog[8406]: This is message 4
prog[8406]: This is message 5
prog[8406]: This is message 6
prog[8406]: This is message 7
prog[8406]: This is message 8
prog[8406]: This is message 9
prog[8406]: This is message 10
prog[8406]: This is message 11
prog[8406]: This is message 12
prog[8406]: This is message 13
prog[8406]: This is message 14
prog[8406]: This is message 15
prog[8406]: This is message 16
prog[8406]: This is message 17
prog[8406]: This is message 18
prog[8406]: This is message 19
prog[8406]: This is message 20
prog[8406]: This is message 21
prog[8406]: This is message 22
prog[8406]: This is message 23
prog[8406]: This is message 24
prog[8406]: This is message 25
prog[8406]: This is message 26

 [Alarm]: Time is Up!Quantum has elapsed!

 [Scheduler]: Process PNAME: prog  PID: 8406  ID: 2 was stopped ! Advancing to next process...
 [Scheduler]: Process PNAME: prog  PID: 8405  ID: 1 continues!

prog: Starting, NMSG = 200, delay = 50
prog[8405]: This is message 0
prog[8405]: This is message 1
prog[8405]: This is message 2
prog[8405]: This is message 3
prog[8405]: This is message 4
prog[8405]: This is message 5
prog[8405]: This is message 6
prog[8405]: This is message 7
prog[8405]: This is message 8
prog[8405]: This is message 9
prog[8405]: This is message 10
prog[8405]: This is message 11
prog[8405]: This is message 12
prog[8405]: This is message 13
prog[8405]: This is message 14
prog[8405]: This is message 15
prog[8405]: This is message 16
```

```
prog[8405]: This is message 16
prog[8405]: This is message 17

 [Alarm]: Time is Up!Quantum has elapsed!

 [Scheduler]: Process PNAME: prog  PID: 8405  ID: 1 was stopped ! Advancing to next process...
 [Scheduler]: Process PNAME: shell  PID: 8404  ID: 0 continues!


This is the Shell. Welcome.

Shell> p
Shell: issuing request...
        [Scheduler]: The following processes are currently running:
PID:8404        ID:0
PID:8406        ID:2
PID:8405        ID:1
Shell: receiving request return value...
Shell>   The ongoin process has process id: 0
 [Alarm]: Time is Up!Quantum has elapsed!

 [Scheduler]: Process PNAME: shell  PID: 8404  ID: 0 was stopped ! Advancing to next process...
 [Scheduler]: Process PNAME: prog  PID: 8406  ID: 2 continues!

prog[8406]: This is message 27
prog[8406]: This is message 28
prog[8406]: This is message 29
prog[8406]: This is message 30
prog[8406]: This is message 31
prog[8406]: This is message 32
prog[8406]: This is message 33
prog[8406]: This is message 34
prog[8406]: This is message 35
prog[8406]: This is message 36
prog[8406]: This is message 37
prog[8406]: This is message 38
prog[8406]: This is message 39
prog[8406]: This is message 40
prog[8406]: This is message 41
prog[8406]: This is message 42
prog[8406]: This is message 43
prog[8406]: This is message 44
prog[8406]: This is message 45
prog[8406]: This is message 46
prog[8406]: This is message 47
prog[8406]: This is message 48
prog[8406]: This is message 49
prog[8406]: This is message 50
prog[8406]: This is message 51
prog[8406]: This is message 52

 [Alarm]: Time is Up!Quantum has elapsed!

 [Scheduler]: Process PNAME: prog  PID: 8406  ID: 2 was stopped ! Advancing to next process...
 [Scheduler]: Process PNAME: prog  PID: 8405  ID: 1 continues!

prog[8405]: This is message 18
prog[8405]: This is message 19
prog[8405]: This is message 20
prog[8405]: This is message 21
prog[8405]: This is message 22
prog[8405]: This is message 23
```

```
prog[8405]: This is message 24
prog[8405]: This is message 25
prog[8405]: This is message 26
prog[8405]: This is message 27
prog[8405]: This is message 28
prog[8405]: This is message 29
prog[8405]: This is message 30
prog[8405]: This is message 31
prog[8405]: This is message 32
prog[8405]: This is message 33

 [Alarm]: Time is Up!Quantum has elapsed!

 [Scheduler]: Process PNAME: prog  PID: 8405  ID: 1 was stopped ! Advancing to next process...
 [Scheduler]: Process PNAME: shell  PID: 8404  ID: 0 continues!

k 1
Shell: issuing request...
Shell: receiving request return value...
Killing in the name of...
        Killing process with id 1.
Shell>
 [Alarm]: Time is Up!Quantum has elapsed!

 [Scheduler]: Process PNAME: shell  PID: 8404  ID: 0 was stopped ! Advancing to next process...
 [Scheduler]: Process PNAME: prog  PID: 8406  ID: 2 continues!

prog[8406]: This is message 53
prog[8406]: This is message 54
prog[8406]: This is message 55
prog[8406]: This is message 56
prog[8406]: This is message 57
prog[8406]: This is message 58
prog[8406]: This is message 59
prog[8406]: This is message 60
prog[8406]: This is message 61
prog[8406]: This is message 62
prog[8406]: This is message 63
prog[8406]: This is message 64
prog[8406]: This is message 65
prog[8406]: This is message 66
prog[8406]: This is message 67
prog[8406]: This is message 68
prog[8406]: This is message 69
prog[8406]: This is message 70
prog[8406]: This is message 71
prog[8406]: This is message 72
prog[8406]: This is message 73
prog[8406]: This is message 74
prog[8406]: This is message 75
prog[8406]: This is message 76
prog[8406]: This is message 77
prog[8406]: This is message 78
prog[8406]: This is message 79

 [Alarm]: Time is Up!Quantum has elapsed!

 [Scheduler]: Process PNAME: prog  PID: 8406  ID: 2 was stopped ! Advancing to next process...
 [Scheduler]: Process PNAME: shell  PID: 8404  ID: 0 continues!

p
Shell: issuing request...
Shell: receiving request return value...
```



-Σύντομες απαντήσεις στις ερωτήσεις

Ερώτηση 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;

Απάντηση:

Αυτό που μας εξασφαλίζει ο χρονοδρομολογητής είναι η εξυπηρέτηση κάθε διεργασίας ξεχωριστά. Για να εξυπηρετηθεί ένα αίτημα της μορφής 'p' πρέπει η διεργασία που τρέχει να είναι ο φλοιός αλλιώς το αίτημα δεν εξυπηρετείται (βλ και Ερώτηση 2 με `signals_disable()` και `_enable()`). Έτσι ως τρέχουσα διεργασία βλέπουμε πάντα τον φλοιό στην απάντηση της 'p' και όχι κάποια άλλη διεργασία.

Ερώτηση 2: Γιατί είναι αναγκαίο να συμπεριλάβετε κλήσεις `signals_disable()`, `_enable()` γύρω από την συνάρτηση υλοποίησης αιτήσεων του φλοιού; Υπόδειξη: Η συνάρτηση υλοποίησης αιτήσεων του φλοιού μεταβάλλει δομές όπως η ουρά εκτέλεσης των διεργασιών.

Απάντηση:

Στον κώδικα παρατηρεί κανείς τις κλήσεις `signals_disable()` και `_enable()` γύρω από τον χειρισμό μιας αίτησης προς το φλοιό. Ουσιαστικά είναι σαν να προστατεύουμε μια αίτηση προς το φλοιό ως κρίσιμο τμήμα. Αυτό συμβαίνει επειδή αιτήσεις προς τον φλοιό σημαίνουν και μεταβολή της λίστας των διεργασιών. Ενδεχόμενο εξωτερικό σήμα θα μπορούσε να μας στείλει σε συνάρτηση χειρισμού του

και καταστροφή των δομών εκείνων που μας επιτρέπουν να χειριστούμε τη λίστα (αφού τη στιγμή που θα το λαμβάναμε θα την επεξεργαζόμασταν).

4.3 Υλοποίηση προτεραιοτήτων στο χρονοδρομολογητή

-Πηγαίος κώδικας (source code)

`scheduler-shell.c`

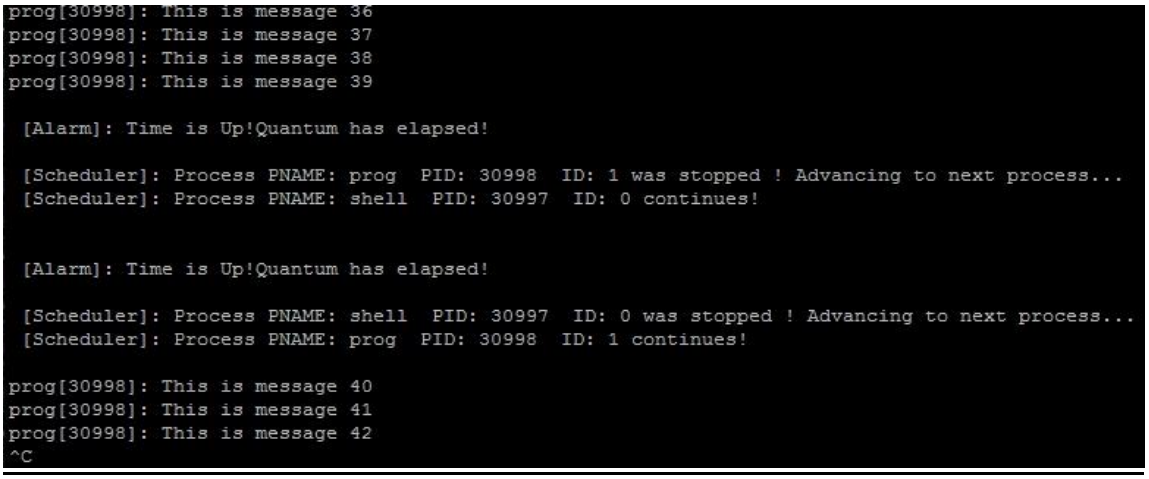
Ίδιος με 4.2.

-Έξοδο εκτέλεσης του προγράμματος

Η εντολή `./scheduler-shell prog prog` έχει ως έξοδο τα παρακάτω screenshots. Αυτό που παρουσιάζεται είναι η εκτύπωση αρχικά με την εντολή `p` στον `shell` των τρεχόντων διεργασιών. Στη συνέχεια αναβαθμίζουμε την προτεραιότητα μιας διεργασίας με την εντολή `h 1` και τέλος παρατηρούμε ότι τρέχει μόνο αυτή και ο scheduler (κατόπιν της υπόθεσης που κάναμε ότι ο shell πρέπει να τρέχει πάντα, χωρίς να το υπογορεύει η εκφώνηση).

```
oslabb15@amorgos:~/oslab_ex4/sched$ ./scheduler-shell_solution prog prog
 [Sheduler]: Inserted to Queue PNAME: shell      PID: 30997      ID: 0
 [Sheduler]: Inserted to Queue PNAME: prog       PID: 30998      ID: 1
 [Sheduler]: Inserted to Queue PNAME: prog       PID: 30999      ID: 2
My PID = 30996: Child PID = 30998 has been stopped by a signal, signo = 19
My PID = 30996: Child PID = 30999 has been stopped by a signal, signo = 19
 [Scheduler]: Process PNAME: shell  PID: 30997  ID: 0 was stopped ! Advancing to next process...
 [Scheduler]: Process PNAME: prog  PID: 30999  ID: 2 continues!

prog: Starting, NMSG = 200, delay = 146
prog[30999]: This is message 0
prog[30999]: This is message 1
prog[30999]: This is message 2
prog[30999]: This is message 3
prog[30999]: This is message 4
prog[30999]: This is message 5
prog[30999]: This is message 6

 [Alarm]: Time is Up!Quantum has elapsed!

 [Scheduler]: Process PNAME: prog  PID: 30999  ID: 2 was stopped ! Advancing to next process...
 [Scheduler]: Process PNAME: prog  PID: 30998  ID: 1 continues!

prog: Starting, NMSG = 200, delay = 99
prog[30998]: This is message 0
prog[30998]: This is message 1
prog[30998]: This is message 2
prog[30998]: This is message 3
prog[30998]: This is message 4
prog[30998]: This is message 5
prog[30998]: This is message 6
prog[30998]: This is message 7
prog[30998]: This is message 8
prog[30998]: This is message 9

 [Alarm]: Time is Up!Quantum has elapsed!

 [Scheduler]: Process PNAME: prog  PID: 30998  ID: 1 was stopped ! Advancing to next process...
 [Scheduler]: Process PNAME: shell  PID: 30997  ID: 0 continues!


This is the Shell. Welcome.

Shell> p
Shell: issuing request...
Shell: receiving request return value...
         [Scheduler]: The following processes are currently running:
PID:30997       ID:0
PID:30999       ID:2
PID:30998       ID:1
Shell>   The ongoin process has process id: 0
 [Alarm]: Time is Up!Quantum has elapsed!

 [Scheduler]: Process PNAME: shell  PID: 30997  ID: 0 was stopped ! Advancing to next process...
 [Scheduler]: Process PNAME: prog  PID: 30999  ID: 2 continues!

prog[30999]: This is message 7
prog[30999]: This is message 8
prog[30999]: This is message 9
prog[30999]: This is message 10
prog[30999]: This is message 11
```

```
prog[30999]: This is message 12
prog[30999]: This is message 13

 [Alarm]: Time is Up!Quantum has elapsed!

 [Scheduler]: Process PNAME: prog  PID: 30999  ID: 2 was stopped ! Advancing to next process...
 [Scheduler]: Process PNAME: prog  PID: 30998  ID: 1 continues!

prog[30998]: This is message 10
prog[30998]: This is message 11
prog[30998]: This is message 12
prog[30998]: This is message 13
prog[30998]: This is message 14
prog[30998]: This is message 15
prog[30998]: This is message 16
prog[30998]: This is message 17
prog[30998]: This is message 18
prog[30998]: This is message 19

 [Alarm]: Time is Up!Quantum has elapsed!

 [Scheduler]: Process PNAME: prog  PID: 30998  ID: 1 was stopped ! Advancing to next process...
 [Scheduler]: Process PNAME: shell  PID: 30997  ID: 0 continues!

h 1
Shell: issuing request...
Shell: receiving request return value...
        UPgrading the priority of process with id 1.
Shell>
 [Alarm]: Time is Up!Quantum has elapsed!

 [Scheduler]: Process PNAME: shell  PID: 30997  ID: 0 was stopped ! Advancing to next process...
 [Scheduler]: Process PNAME: prog  PID: 30998  ID: 1 continues!

prog[30998]: This is message 20
prog[30998]: This is message 21
prog[30998]: This is message 22
prog[30998]: This is message 23
prog[30998]: This is message 24
prog[30998]: This is message 25
prog[30998]: This is message 26
prog[30998]: This is message 27
prog[30998]: This is message 28
prog[30998]: This is message 29

 [Alarm]: Time is Up!Quantum has elapsed!

 [Scheduler]: Process PNAME: prog  PID: 30998  ID: 1 was stopped ! Advancing to next process...
 [Scheduler]: Process PNAME: shell  PID: 30997  ID: 0 continues!


 [Alarm]: Time is Up!Quantum has elapsed!

 [Scheduler]: Process PNAME: shell  PID: 30997  ID: 0 was stopped ! Advancing to next process...
 [Scheduler]: Process PNAME: prog  PID: 30998  ID: 1 continues!

prog[30998]: This is message 30
prog[30998]: This is message 31
prog[30998]: This is message 32
prog[30998]: This is message 33
prog[30998]: This is message 34
prog[30998]: This is message 35
```



-Σύντομες απαντήσεις στις ερωτήσεις

Ερώτηση1: Περιγράψτε ένα σενάριο δημιουργίας λιμοκτονίας.

Απάντηση:

Σε περίπτωση που μια διεργασία μένει πάντα με low priority, ενώ συνεχώς έρχονται νέες και μπαίνουν σε high priority, η διεργασία αυτή δε θα εξυπηρετείται και δε θα λαμβάνει σήμα από το χρονοδρομολογητή για να συνεχίσει την εκτέλεσή της. Κοινώς θα λιμοκτονεί.

-Διαδικασία μεταγλώττισης και σύνδεσης

Αλλάξαμε κατάλληλα το Makefile που μας δίνετε ώστε να δημιουργεί τα εκτελέσιμα όλες τις ασκήσεις όπως φαίνεται παρακάτω:

```
#
# Makefile
#
# Operating Systems, Exercise 4
#

CC = gcc
#CFLAGS = -Wall -g
CFLAGS = -Wall -O2 -g

all: scheduler scheduler_solution scheduler-shell scheduler-
shell_solution shell prog execve-example strace-test sigchld-example

scheduler: scheduler.o proc-common.o
    $(CC) -o scheduler scheduler.o proc-common.o

scheduler_solution: scheduler_solution.o proc-common.o
    $(CC) -o scheduler_solution scheduler_solution.o proc-common.o

scheduler-shell: scheduler-shell.o proc-common.o
    $(CC) -o scheduler-shell scheduler-shell.o proc-common.o

scheduler-shell_solution: scheduler-shell_solution.o proc-common.o
    $(CC) -o scheduler-shell_solution scheduler-shell_solution.o proc-
common.o
```

```
shell: shell.o proc-common.o
	$(CC) -o shell shell.o proc-common.o

prog: prog.o proc-common.o
	$(CC) -o prog prog.o proc-common.o

execve-example: execve-example.o
	$(CC) -o execve-example execve-example.o

strace-test: strace-test.o
	$(CC) -o strace-test strace-test.o

sigchld-example: sigchld-example.o proc-common.o
	$(CC) -o sigchld-example sigchld-example.o proc-common.o

proc-common.o: proc-common.c proc-common.h
	$(CC) $(CFLAGS) -o proc-common.o -c proc-common.c

shell.o: shell.c proc-common.h request.h
	$(CC) $(CFLAGS) -o shell.o -c shell.c

scheduler.o: scheduler.c proc-common.h request.h
	$(CC) $(CFLAGS) -o scheduler.o -c scheduler.c

scheduler-shell.o: scheduler-shell.c proc-common.h request.h
	$(CC) $(CFLAGS) -o scheduler-shell.o -c scheduler-shell.c

prog.o: prog.c
	$(CC) $(CFLAGS) -o prog.o -c prog.c

execve-example.o: execve-example.c
	$(CC) $(CFLAGS) -o execve-example.o -c execve-example.c

strace-test.o: strace-test.c
	$(CC) $(CFLAGS) -o strace-test.o -c strace-test.c

sigchld-example.o: sigchld-example.c
	$(CC) $(CFLAGS) -o sigchld-example.o -c sigchld-example.c

clean:
	rm -f scheduler scheduler-shell shell prog execve-example strace-
test sigchld-example *.o
```