Εθνικό Μετσόβιο Πολυτεχνείο

Σχολή Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών

Λειτουργικά Συστήματα , Ροή Υ

6ο εξάμηνο, Ακαδημαϊκή περίοδος 2019-2020

Άσκηση 4: Χρονοδρομολόγηση

Στοιχεία σπουδαστών:

Όνομα: Μαντζούτας Ανδρέας Α.Μ. : 03117108

Όνομα: Τσιτσής Αντώνιος Α.Μ. : 03117045

Ομάδα εργαστηρίου: Oslabc23

Εξάμηνο: 6^{ο}

Ημερομηνία: 02/06/2020

1.1 Υλοποίηση χρονοδρομολογητή κυκλικής επαναφοράς στο χώρο χρήστη

Στην άσκηση αυτή κληθήκαμε να υλοποιήσουμε έναν χρονοδρομολογητή κυκλικής επαναφοράς, ο οποίος θα διαμοιράζει τον υπολογιστικό χρόνο στις διεργασίες-παιδιά.

Για να επιτευχθεί αυτό, δημιουργήσαμε μία κυκλική, απλά συνδεδεμένη λίστα. Επίσης, δημιουργήσαμε δύο pointer head και end, που δείχνουν στην κορυφή και στο τέλος της λίστας αντίστοιχα. Έπειτα, δημιουργήσαμε τις απαραίτητες διεργασίες και τις τοποθετήσαμε στη λίστα. Με τη συνάρτηση wait_for_ready_children περιμένουμε την δημιουργία όλων των διεργασιών, άρα ουσιαστικά τη δημιουργία της λίστας, και όταν ολοκληρωθεί, στέλνουμε σήμα SIGCONT στην κεφαλή της λίστας για να ξεκινήσει η πρώτη διεργασία.

Τέλος, χρησιμοποιήσαμε δύο ρουτίνες sighandler. Η μια είναι υπεύθυνη για να στείλει το σήμα SIGSTOP στην διεργασία όταν εκπνεύσει το κβάντο χρόνου, ενώ η δεύτερη ενεργοποιείται όταν τερματιστεί μια διεργασία. Πιο συγκεκριμένα, αν μια διεργασία τερματιστεί, τότε ο δεύτερος sighandler την αφαιρεί από τη λίστα, ενώ αν εκπνεύσει το κβάντο χρόνου, την τοποθετεί στο τέλος της λίστας, δηλαδή εκεί που δείχνει ο pointer end.

Παραθέτουμε παρακάτω τον κώδικα του προγράμματος, καθώς και μερικά παραδείγματα εκτέλεσής του. Σημειώνουμε ότι στο πρόγραμμα prog.c μειώνουμε τον αριθμό των μηνυμάτων σε 20, ώστε να τερματιστεί γρηγορότερα το πρόγραμμα.

```
#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 */

typedef struct Node{
      pid_t pid;
      char *name;
      struct Node *next;
      } Node;
```

```
typedef Node * nodeptr;

nodeptr head = NULL;
nodeptr end = NULL; //arxika einai keni i lista

void insert( pid_t pid, const char *name){

	int flag= 0;
	if (head == NULL){
		flag = 1;
	}

	nodeptr new = (Node *) malloc(sizeof(Node));
	new->pid = pid;
	new->name= strdup(name);
	new->next = NULL;
	if (flag == 1){
		head = new;
		end = new;
	}
	else{
		end->next = new;
		end = new;
	}
}

void delete (pid_t pid) {
	nodeptr temp = head;
	if (head->pid==pid){
		head=temp->next;
		free(temp);
	}
	else if (end->pid==pid){
		while(temp->next!=end){
			temp=temp->next;
		}
		end=temp;
		temp=temp->next;
		end->next=NULL;
		free(temp);
	}

	else{
		while(temp->next->pid!=pid){
			temp=temp->next;
		}
		nodeptr del= temp->next;
		temp->next=temp->next->next;
		free(del);
	}
	if (head == NULL){
		printf("List empty! Everything done!\n");
		exit(0);}
}
```

```
/*
 * SIGALRM handler
 */
static void
sigalrm_handler(int signum)
{
	if (kill(head -> pid, SIGSTOP)<0){
		perror("sigalrm");
		exit(10);
	}
}

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

	for(;;){
		id = waitpid(-1, &status, WUNTRACED| WNOHANG);

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

		explain_wait_status(id,status);
		if (WIFEXITED(status) | WIFSIGNALED(status)){
	/* termatistike i diergasia, ara tin aferoume*/
			delete(id);
		}
		if (WIFSTOPPED(status)){
		/*den termatistike, ara topothetoume to head sto telos*/
			end->next=head;
			end=head;
			nodeptr temp=head;
			head=head->next;
			temp->next=NULL;

		}

		alarm(SCHED_TQ_SEC);

		if((kill(head->pid,SIGCONT))<0){
			perror("kill");
			exit(5);
		}
	}
}

/* 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,i;
	pid_t pid;
	if (argc < 2){
		perror("arguments");
		exit(1);
	}
	nproc = argc-1;


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

	for(i =0; i<nproc; i++){
		pid = fork();
		if (pid<0){
			perror("main:fork");
		}
		if (pid == 0) {
			raise(SIGSTOP);
			printf("I am starting. My PID is %ld\n", (long)getpid());
```

```
                sleep(2);
                execve(executable, newargv, newenviron);
            }
            else{
                insert(pid, argv[i+1]);

            }
        }

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

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

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

        kill(head->pid, SIGCONT);
        /* loop forever  until we exit from inside a signal handler. */
        while (pause())
            ;

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

```
oslabc23@os-node1:~/andr/4/gia_paradosi$ ./scheduler prog prog
My PID = 14875: Child PID = 14876 has been stopped by a signal, signo = 19
My PID = 14875: Child PID = 14877 has been stopped by a signal, signo = 19
I am starting. My PID is 14876
My PID = 14875: Child PID = 14876 has been stopped by a signal, signo = 19
I am starting. My PID is 14877
My PID = 14875: Child PID = 14877 has been stopped by a signal, signo = 19
prog: Starting, NMSG = 20, delay = 60
prog[14876]: This is message 0
prog[14876]: This is message 1
prog[14876]: This is message 2
prog[14876]: This is message 3
prog[14876]: This is message 4
prog[14876]: This is message 5
prog[14876]: This is message 6
prog[14876]: This is message 7
prog[14876]: This is message 8
prog[14876]: This is message 9
prog[14876]: This is message 10
prog[14876]: This is message 11
My PID = 14875: Child PID = 14876 has been stopped by a signal, signo = 19
prog: Starting, NMSG = 20, delay = 42
prog[14877]: This is message 0
prog[14877]: This is message 1
prog[14877]: This is message 2
prog[14877]: This is message 3
prog[14877]: This is message 4
prog[14877]: This is message 5
prog[14877]: This is message 6
prog[14877]: This is message 7
prog[14877]: This is message 8
prog[14877]: This is message 9
prog[14877]: This is message 10
prog[14877]: This is message 11
prog[14877]: This is message 12
prog[14877]: This is message 13
prog[14877]: This is message 14
prog[14877]: This is message 15
My PID = 14875: Child PID = 14877 has been stopped by a signal, signo = 19
prog[14876]: This is message 12
prog[14876]: This is message 13
prog[14876]: This is message 14
prog[14876]: This is message 15
prog[14876]: This is message 16
prog[14876]: This is message 17
prog[14876]: This is message 18
prog[14876]: This is message 19
My PID = 14875: Child PID = 14876 terminated normally, exit status = 0
prog[14877]: This is message 16
prog[14877]: This is message 17
prog[14877]: This is message 18
prog[14877]: This is message 19
My PID = 14875: Child PID = 14877 terminated normally, exit status = 0
List is empty. Everything's done!
```

Ερωτήσεις:

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

Όπως έχουμε ορίσει στη συνάρτηση install_signal_handlers(), σε περίπτωση που έρθει ένα σήμα SIGALRM, ενώ εκτελείται η συνάρτηση σήματος SIGCHLD, το πρώτο σήμα μπλοκάρεται. Το ίδιο συμβαίνει και στην αντίθετη περίπτωση. Προφανώς αυτό σημαίνει ότι αν ληφθεί κάποιο σήμα τη στιγμή που εκτελείται η ρουτίνα εξυπηρέτησης του άλλου σήματος, τότε το πρώτο δεν θα εξυπηρετηθεί.

Αντίθετα, ένας πραγματικός χρονοδρομολογητής χώρου πυρήνα δε λειτουργεί με σήματα, αλλά με διακοπές. Αυτό σημαίνει ότι με το που συμβεί μία διακοπή, τότε εξυπηρετείται αμέσως η ρουτίνα εξυπηρέτησης αυτής της διακοπής, με αποτέλεσμα να μεγιστοποιείται η αποκρισιμότητα του συστήματος.

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

Ο χρονοδρομολογητής λαμβάνει σήμα SIGCHLD σε περίπτωση που κάποιο παιδί πεθάνει, τερματιστεί με “kill” ή εκπνεύσει το κβάντο χρόνου του. Στη ρουτίνα χειρισμού του σήματος SIGCHLD, με τη χρήση των συναρτήσεων waitpid() και explain_wait_status() λαμβάνουμε τις απαραίτητες πληροφορίες σχετικά με την κατάσταση του παιδιού. Σε περίπτωση που η διεργασία-παιδί είχε τερματιστεί αναπάντεχα, τότε ο handler θα την αφαιρέσει από την λίστα.

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.

Όπως γνωρίζουμε, τα σήματα δεν είναι πάντα ασφαλή και αξιόπιστα. Σε περίπτωση που χρησιμοποιούσαμε μόνο SIGALRM, τότε θα ήταν πιθανό να παραδοθεί ένα σήμα SIGSTOP για τερματισμό της διεργασίας, μετά την παράδοση του σήματος SIGCONT στην επόμενη διεργασία, με αποτέλεσμα να εκτελούνται δύο διεργασίες ταυτόχρονα.

Στην περίπτωση μας, που χρησιμοποιούμε δύο σήματα, βεβαιωνόμαστε πως μόνο μία διεργασία θα είναι ενεργοποιημένη κάθε φορά. Αυτό επιτυγχάνεται επειδή για να ενεργοποιηθεί μια διεργασία, θα πρέπει να παραλάβει SIGCHLD, γεγονός που μας εγγυάται ότι η προηγούμενη διεργασία θα έχει λάβει σήμα SIGSTOP και έχει σταματήσει.

1.2 Έλεγχος λειτουργίας χρονοδρομολογητή μέσω φλοιού

Για την υλοποίηση αυτής της άσκησης, κινηθήκαμε σε γενικές γραμμές όπως και στην προηγούμενη. Η κύρια διαφορά ήταν ότι θέλαμε να είναι δυνατός ο έλεγχος της λειτουργίας μέσω του φλοιού. Για την επίτευξη αυτού, κατά τη δημιουργία της λίστας, τοποθετούμε σαν κεφαλή, με σειριακό αριθμό 0, τον φλοιό. Έτσι, κάθε φορά που έρχεται η «σειρά» του για να εκτελέσει τη λειτουργία του, αναμένει λίγη ώρα για να λάβει κάποια εντολή, είτε για προσθήκη/τερματισμό διεργασίας, είτε για να εκτυπώσει τις ενεργές διεργασίες, είτε για να τερματιστεί ο ίδιος. Άμα δεν λάβει κάποια εντολή μέχρι να εκπνεύσει το κβάντο χρόνου του, θα τοποθετηθεί τελευταίος στη λίστα, και θα χρονοδρομολογηθεί η επόμενη διεργασία.

Σας παραθέτουμε τον επεκταμένο κώδικα του χρονοδρομολογητή.

```
#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 */

typedef struct Node{
      pid_t pid;
      char* name;
      int id;
      struct Node * next;
}Node;
typedef Node * nodeptr;

/*Definition of my list nodes*/
nodeptr head=NULL;
nodeptr end=NULL;

int serial_number;

/* Print a list of all tasks currently being scheduled.  */
static void
sched_print_tasks(void)
{
      nodeptr temp=head;
      if (temp->next == NULL){
            printf("ID: %d PID: %i Name: %s \n",temp->id,temp->pid,temp->name);
      }
```

```
	else{
		while (temp!=NULL){
			printf("ID: %d PID: %i Name: %s \n",temp->id,temp->pid,temp->name);
			temp = temp->next;
		}

	}
}

/* Send SIGKILL to a task determined by the value of its
 * scheduler-specific id.
 */

static int
sched_kill_task_by_id(int id)
{
	nodeptr temp=head;
	while (temp!=NULL){
		if (temp->id == id ){
			kill(temp->pid,SIGKILL);
			return id;
		}
		else
			if (temp == end && end->id != id) break;
			else temp = temp->next;
	}
	return -ENOSYS;
}


/* Create a new task.  */
static void
sched_create_task(char *executable)//
{
		pid_t mypid;
		mypid=fork();
		if (mypid < 0){
			perror("fork");
		}
		if(mypid == 0){

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

			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 process to the list */
			nodeptr newnode=(nodeptr)malloc(sizeof(Node));
```

```
                if (newnode == NULL){
                        perror("malloc");
                        exit(2);
                }
                newnode->pid=mypid;
                newnode->next=NULL;
                newnode->id=serial_number;
                newnode->name= strdup(executable);
                end->next=newnode;
                end=newnode;
                serial_number ++;
        }

}

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

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

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

                default:
                        return -ENOSYS;
        }
}

/*
 * SIGALRM handler
 */
static void
sigalrm_handler(int signum)
{
        if ((kill(head->pid,SIGSTOP))<0){
                perror("sigalrm: kill");
                exit(51);
        }
}


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

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

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

        /* a child has changed his status, lets check what happened */
        explain_wait_status(p,status);


        if (WIFEXITED(status) || WIFSIGNALED(status)){
            /* If i am here means that a child is terminated OR killed by a signal */
            /* Because of that i have to remove it from the list */
            nodeptr previous = NULL;
            nodeptr current = head;

            while(current != NULL){
                if (current->pid == p && current == head){
                    /* I have to delete the head of my list */
                        if (current->next == NULL){
                            /* I will come here only once...when i have only one node... i delete it
and i am done */
                             free(current);
                             printf("List is empty! Everything done correctly!\n");
                             exit(0);
                        }
                        else{
                            head=current->next;
                            free(current);
                        }
                }
                else if (current->pid == p && current == end){
                    /* I have to delete the last node of my list */
                    end=previous;
                    end->next=NULL;
                    free(current);
                }
                else if(current->pid == p){
                    /* I have to delete a random node of the list but sure its not head or tail */
                    previous->next=current->next;
                    free(current);
                }
                else{
                    /* I will continue searching*/
                    previous=current;
                    current=current->next;
                    continue;
                }
                break;
            }
```

```
            }

            if (WIFSTOPPED(status)){
                /* I am here if a process is stopped */
                /* I transfer this process at the end of list and continue with the next one */
                nodeptr temp;
                end->next=head;
                end=head;
                temp=head;
                head=head->next;
                temp->next=NULL;

            }


            if(head->id==0){
                printf("I am in Shell again:\n");
                alarm(5*SCHED_TQ_SEC);
            }
            else alarm(SCHED_TQ_SEC);

            /* Send SIGCONT to the head of the list */
            if ((kill(head->pid,SIGCONT))<0){
                perror("sigchl: kill");
                exit(15);
            }


        }

}


/* 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);
```

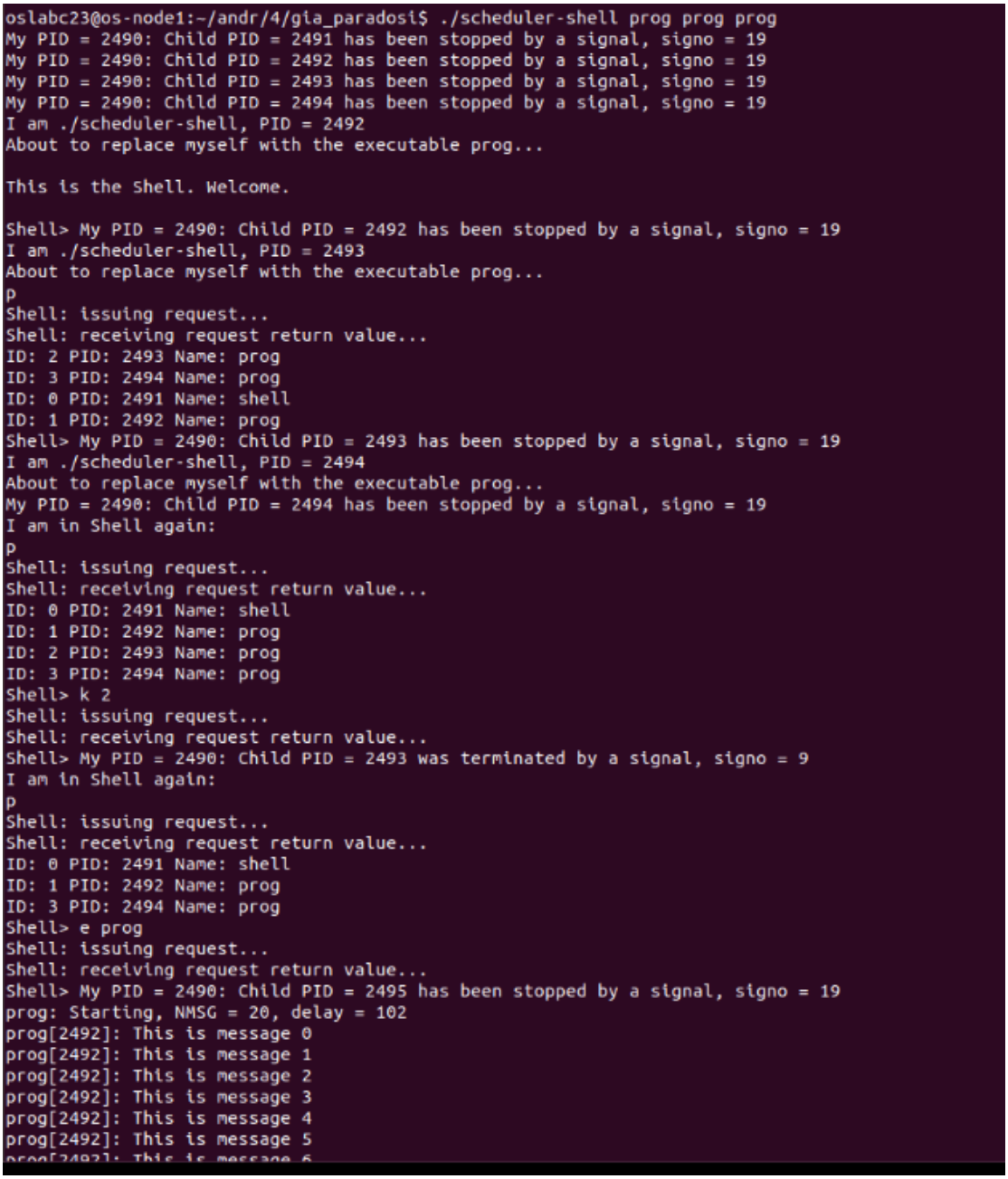
```
	if (sigprocmask(SIG_UNBLOCK, &sigset, NULL) < 0) {
		perror("signals_enable: sigprocmask");
		exit(1);
	}
}


/* 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];

	/* I have to insert the Shell as the head of my list */
	head->id=0;
	head->pid=p;
	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;

        /* Initialize the head and the end of the list to put the Shell on the head and the end as well
because at the start Shell will be the only node-process */
        head=(nodeptr)malloc(sizeof(Node));
        if(head == NULL){
                perror("malloc");
                exit(2);
        }
        end=head;

        /* Create the Shell and put Shell at the head of the list*/

        sched_create_shell(SHELL_EXECUTABLE_NAME, &request_fd, &return_fd);
        int nproc = argc-1, i;

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

        nodeptr newnode;
        pid_t mypid;

        /*Fork and create the list*/
        for (i=1;i<=nproc;i++){
                mypid=fork();
                if (mypid<0){
                        perror("main: fork");
                }
                if(mypid==0){
                        char executable[] = "prog";
```

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


                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{
                newnode=(nodeptr)malloc(sizeof(Node));
                if (newnode == NULL){
                      perror("malloc");
                      exit(2);
                }
                newnode->pid=mypid;
                newnode->next=NULL;
                newnode->id=i;
                newnode->name=argv[i];
                end->next=newnode;
                end=newnode;
          }
     }
     serial_number = i;
     /* 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->pid,SIGCONT); //wake up the shell

     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;
}
```

Παρακάτω, σας δίνουμε ένα παράδειγμα εκτέλεσης, όπου καλούμε το scheduler-shell με ορίσματα 3 φορές το prog. Έπειτα, προσθέτουμε κάποιες διεργασίες επιπλέον, καλούμε τον φλοιό να εκτυπώσει τη λίστα, και τέλος τερματίζουμε όλες τις διεργασίες εκτός από μία και καλούμε τον φλοιό να τερματιστεί, ώστε νε τερματιστεί γρήγορα και το πρόγραμμα. Σας παραθέτουμε τα screenshot.



```
prog[2492]: This is message 6
My PID = 2490: Child PID = 2492 has been stopped by a signal, signo = 19
prog: Starting, NMSG = 20, delay = 66
prog[2494]: This is message 0
prog[2494]: This is message 1
prog[2494]: This is message 2
prog[2494]: This is message 3
prog[2494]: This is message 4
prog[2494]: This is message 5
prog[2494]: This is message 6
prog[2494]: This is message 7
prog[2494]: This is message 8
prog[2494]: This is message 9
prog[2494]: This is message 10
My PID = 2490: Child PID = 2494 has been stopped by a signal, signo = 19
I am prog, PID = 2495
About to replace myself with the executable prog...
My PID = 2490: Child PID = 2495 has been stopped by a signal, signo = 19
I am in Shell again:
p
Shell: issuing request...
Shell: receiving request return value...
ID: 0 PID: 2491 Name: shell
ID: 1 PID: 2492 Name: prog
ID: 3 PID: 2494 Name: prog
ID: 4 PID: 2495 Name: prog
Shell> e prog
Shell: issuing request...
Shell: receiving request return value...
Shell> My PID = 2490: Child PID = 2496 has been stopped by a signal, signo = 19
prog[2492]: This is message 7
prog[2492]: This is message 8
prog[2492]: This is message 9
prog[2492]: This is message 10
prog[2492]: This is message 11
prog[2492]: This is message 12
prog[2492]: This is message 13
My PID = 2490: Child PID = 2492 has been stopped by a signal, signo = 19
prog[2494]: This is message 11
prog[2494]: This is message 12
prog[2494]: This is message 13
prog[2494]: This is message 14
prog[2494]: This is message 15
prog[2494]: This is message 16
prog[2494]: This is message 17
prog[2494]: This is message 18
prog[2494]: This is message 19
My PID = 2490: Child PID = 2494 terminated normally, exit status = 0
prog: Starting, NMSG = 20, delay = 113
prog[2495]: This is message 0
prog[2495]: This is message 1
prog[2495]: This is message 2
prog[2495]: This is message 3
prog[2495]: This is message 4
```

```
prog[2495]: This is message 4
prog[2495]: This is message 5
My PID = 2490: Child PID = 2495 has been stopped by a signal, signo = 19
I am prog, PID = 2496
About to replace myself with the executable prog...
My PID = 2490: Child PID = 2496 has been stopped by a signal, signo = 19
I am in Shell again:
p
Shell: issuing request...
Shell: receiving request return value...
ID: 0 PID: 2491 Name: shell
ID: 1 PID: 2492 Name: prog
ID: 4 PID: 2495 Name: prog
ID: 5 PID: 2496 Name: prog
Shell> k 1
Shell: issuing request...
Shell: receiving request return value...
Shell> My PID = 2490: Child PID = 2492 was terminated by a signal, signo = 9
I am in Shell again:
k 4
Shell: issuing request...
Shell: receiving request return value...
Shell> My PID = 2490: Child PID = 2495 was terminated by a signal, signo = 9
I am in Shell again:
p
Shell: issuing request...
Shell: receiving request return value...
ID: 0 PID: 2491 Name: shell
ID: 5 PID: 2496 Name: prog
Shell> q
Shell: Exiting. Goodbye.
My PID = 2490: Child PID = 2491 terminated normally, exit status = 0
scheduler: read from shell: Success
Scheduler: giving up on shell request processing.
prog: Starting, NMSG = 20, delay = 159
prog[2496]: This is message 0
prog[2496]: This is message 1
prog[2496]: This is message 2
prog[2496]: This is message 3
prog[2496]: This is message 4
My PID = 2490: Child PID = 2496 has been stopped by a signal, signo = 19
prog[2496]: This is message 5
prog[2496]: This is message 6
prog[2496]: This is message 7
prog[2496]: This is message 8
My PID = 2490: Child PID = 2496 has been stopped by a signal, signo = 19
prog[2496]: This is message 9
prog[2496]: This is message 10
prog[2496]: This is message 11
prog[2496]: This is message 12
My PID = 2490: Child PID = 2496 has been stopped by a signal, signo = 19
prog[2496]: This is message 13
prog[2496]: This is message 14
```

```
p
Shell: issuing request...
Shell: receiving request return value...
ID: 0 PID: 2491 Name: shell
ID: 1 PID: 2492 Name: prog
ID: 4 PID: 2495 Name: prog
ID: 5 PID: 2496 Name: prog
Shell> k 1
Shell: issuing request...
Shell: receiving request return value...
Shell> My PID = 2490: Child PID = 2492 was terminated by a signal, signo = 9
I am in Shell again:
k 4
Shell: issuing request...
Shell: receiving request return value...
Shell> My PID = 2490: Child PID = 2495 was terminated by a signal, signo = 9
I am in Shell again:
p
Shell: issuing request...
Shell: receiving request return value...
ID: 0 PID: 2491 Name: shell
ID: 5 PID: 2496 Name: prog
Shell> q
Shell: Exiting. Goodbye.
My PID = 2490: Child PID = 2491 terminated normally, exit status = 0
scheduler: read from shell: Success
Scheduler: giving up on shell request processing.
prog: Starting, NMSG = 20, delay = 159
prog[2496]: This is message 0
prog[2496]: This is message 1
prog[2496]: This is message 2
prog[2496]: This is message 3
prog[2496]: This is message 4
My PID = 2490: Child PID = 2496 has been stopped by a signal, signo = 19
prog[2496]: This is message 5
prog[2496]: This is message 6
prog[2496]: This is message 7
prog[2496]: This is message 8
My PID = 2490: Child PID = 2496 has been stopped by a signal, signo = 19
prog[2496]: This is message 9
prog[2496]: This is message 10
prog[2496]: This is message 11
prog[2496]: This is message 12
My PID = 2490: Child PID = 2496 has been stopped by a signal, signo = 19
prog[2496]: This is message 13
prog[2496]: This is message 14
prog[2496]: This is message 15
prog[2496]: This is message 16
My PID = 2490: Child PID = 2496 has been stopped by a signal, signo = 19
prog[2496]: This is message 17
prog[2496]: This is message 18
prog[2496]: This is message 19
My PID = 2490: Child PID = 2496 terminated normally, exit status = 0
List is empty! Everything done correctly!
oslabc23@os-node1:~/andr/4/gia_paradosi$
```

Ερωτήσεις:

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

Την εντολή 'p', όπως και οποιαδήποτε άλλη εντολή, μπορούμε να την πληκτρολογήσουμε μόνο όταν τρέχουσα διεργασία είναι ο φλοιός. Επομένως, όπως είναι λογικό πάντα η τρέχουσα διεργασία, δηλαδή η πάνω πάνω, θα είναι ο φλοιός. Θεωρητικά, αυτό δε θα μπορούσε να μη συμβαίνει, καθώς κατά τη διάρκεια της εξυπηρέτησης του φλοιού έχουμε μπλοκάρει όλα τα υπόλοιπα σήματα. Ωστόσο, κατά τη «δοκιμή» του προγράμματος, παρατηρήσαμε κάποιες φορές το φαινόμενο να εμφανιστεί κάποια άλλη διεργασία ως τρέχουσα, όταν πληκτρολογούσαμε την εντολή 'p' ακριβώς τη στιγμή που εξέπνεε το κβάντο χρόνου του φλοιού, άρα και ενεργοποιούνταν τα σήματα, με αποτέλεσμα όταν γινόταν η εξυπηρέτηση του αιτήματος, να βρίσκεται η επόμενη διεργασία ως τρέχουσα.

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.

Η λειτουργία των παραπάνω κλήσεων είναι η απενεργοποίηση και ενεργοποίηση των σημάτων αντίστοιχα. Τις συμπεριλαμβάνουμε γύρω από τη συνάρτηση υλοποίηση αιτήσεων του φλοιού, με σκοπό να μπλοκάρουμε τα σήματα κατά την διάρκεια που εκτελεί τη λειτουργία του. Άμα δεν το κάναμε αυτό, θα υπήρχε περίπτωση κάποια άλλη διεργασία να τροποποιήσει μια κοινή δομή, όπως η ουρά εκτέλεσης των διεργασιών.

1.3 Υλοποίηση προτεραιοτήτων στο χρονοδρομολογητή

Αυτή τη φορά κληθήκαμε να επεκτείνουμε ακόμα περισσότερο το πρόγραμμα του χρονοδρομολογητή, ώστε να υποστηρίζει και λειτουργίες προτεραιότητας. Έτσι, στους κόμβους της λίστας προσθέσαμε ακόμα ένα πεδίο, που αφορούσε την προτεραιότητα της διεργασίας.
Η κύρια διαφορά με το προηγούμενο ζήτημα, ήταν ότι αυτή τη φορά όταν μία διεργασία τερματιζόταν ή εξέπνεε το κβάντο χρόνου της, δεν «ξυπνούσαμε» την αμέσως επόμενη, αλλά ψάχναμε στη λίστα για την πρώτη με προτεραιότητα high. Σε περίπτωση που υπήρχε, της στέλναμε σήμα SIGCONT.
Για την επίτευξη αυτού, ουσιαστικά περιστρέφαμε τη λίστα(βάζαμε την κεφαλή στο τέλος και ορίζαμε ως κεφαλή την αμέσως επόμενη διεργασία), έως ότου βρίσκαμε κάποια διεργασία που πληρούσε τις προϋποθέσεις. Σε περίπτωση που ξαναφτάναμε στην αρχική κεφαλή, σημαίνει ότι δεν υπήρχε κάποια με υψηλή προτεραιότητα, και επομένως συνεχίζαμε κανονικά την «κυκλική λειτουργία».
Τέλος, υλοποιήσαμε δύο επιπλέον συναρτήσεις, για να θέτουμε την προτεραιότητα μιας διεργασίας high ή low, καθώς και συμπεριλάβαμε τις επιπλέον περιπτώσεις στη συνάρτηση εξυπηρέτησης των αιτημάτων του φλοιού.

Παραθέτουμε τον επεκταμένο κώδικα scheduler-shell2.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 */

typedef struct Node{
      pid_t pid;
      int prio;
      char* name;
      int id;
      struct Node * next;
}Node;
typedef Node * nodeptr;

/*Definition of my list nodes*/
nodeptr head=NULL;
nodeptr end=NULL;

int serial_number;
```

```
static int
sched_set_high_prio(int id)
{
	nodeptr temp = head;
	while (temp!=NULL){
		if (temp->id == id){
			temp->prio = 1;
			return id;
		}
		else{
			if (temp == end && end->id!=id) break;
			else temp= temp->next;
		}
	}
	return -ENOSYS;
}


static int
sched_set_low_prio(int id)
{
	nodeptr temp = head;
	while (temp!=NULL){
		if (temp->id == id){
			temp->prio = 0;
			return id;
		}
		else{
			if (temp == end && end->id!=id) break;
			else temp= temp->next;
		}
	}
	return -ENOSYS;
}




/* Print a list of all tasks currently being scheduled.*/
static void
sched_print_tasks(void)
{
	nodeptr temp=head;
	if (temp->next == NULL){
		if (temp->prio == 0){
			printf("ID: %d PID: %i Name: %s Priority: Low \n",temp->id,temp->pid,temp->name);
		}
		else {
			printf("ID: %d PID: %i Name: %s Priority: High \n",temp->id,temp->pid,temp->name);
		}
	}

	else{
		while (temp!=NULL){
			if (temp->prio == 0){
```

```
                    printf("ID: %d PID: %i Name: %s Priority: Low \n",temp->id,temp->pid,temp-
>name);
                }
                else {
                    printf("ID: %d PID: %i Name: %s Priority: High \n",temp->id,temp->pid,temp-
>name);
                }
                temp = temp->next;
            }

        }
}

/* Send SIGKILL to a task determined by the value of its
 * scheduler-specific id.
 */

static int
sched_kill_task_by_id(int id)
{
    nodeptr temp=head;
    while (temp!=NULL){
        if (temp->id == id ){
            kill(temp->pid,SIGKILL);
            return id;
        }
        else
            if (temp == end && end->id != id) break;
            else temp = temp->next;
    }
    return -ENOSYS;
}


/* Create a new task.  */
static void
sched_create_task(char *executable)//
{
        pid_t mypid;
        mypid=fork();
        if (mypid < 0){
            perror("fork");
        }
        if(mypid == 0){

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

            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 process to the list */
                nodeptr newnode=(nodeptr)malloc(sizeof(Node));
                if (newnode == NULL){
                    perror("malloc");
                    exit(2);
                }
                newnode->pid=mypid;
                newnode->next=NULL;
                newnode->id=serial_number;
                newnode->name= strdup(executable);
                end->next=newnode;
                end=newnode;
                newnode->prio = 0;
                serial_number ++;
            }

}

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

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

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

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

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

        default:
            return -ENOSYS;
    }
}

/*
 * SIGALRM handler
 */
static void
sigalrm_handler(int signum)
{
    if ((kill(head->pid,SIGSTOP))<0){
        perror("sigalrm: kill");
        exit(51);
    }
```

```
}


/*
 * SIGCHLD handler
 */
static void
sigchld_handler(int signum)
{
	pid_t p;
	int status;
	int flag = 0; //boool 0 = false/den uparxoun high priorities, 1= true/uparxoun high
	for(;;){
		p=waitpid(-1,&status,WNOHANG|WUNTRACED);

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

		/* a child has changed his status, lets check what happened */
		explain_wait_status(p,status);


		if (WIFEXITED(status) || WIFSIGNALED(status)){
			/* If i am here means that a child is terminated OR killed by a signal */
			/* Because of that i have to remove it from the list */
			nodeptr previous = NULL;
			nodeptr current = head;

			while(current != NULL){
				if (current->pid == p && current == head){
					/* I have to delete the head of my list */
						if (current->next == NULL){

							free(current);
							printf("The list is empty...I don't have any more work to do\n");
							exit(0);
						}
						else{
							head=current->next;
							free(current);
						}
				}
				else if (current->pid == p && current == end){
					/* I have to delete the last node of my list */
					end=previous;
					end->next=NULL;
					free(current);
				}
				else if(current->pid == p){
					/* I have to delete a random node of the list but sure its not head or tail */
					previous->next=current->next;
					free(current);
				}
				else{
```

```
                    /* I will continue searching*/
                    previous=current;
                    current=current->next;
                    continue;
                }
                break;
            }

            nodeptr temp = head;
            nodeptr hd=head;
            //kano rotate ti lista mexri na bro high priority kai na to fero sto head
            while (temp != NULL){
                    if (temp->prio!=1){
                        /* In this case i have to put the process at the end of the list */
                        end->next=head;
                        end=head;
                        head=head->next;
                        end->next=NULL;
                        temp=head;
                        if (temp == hd){
                            flag = 0;  //den uparxoun high priorities
                            break;
                        }
                    }
                    else{
                         flag = 1 ; //head->prio == 1
                         break;
                    }
            }

        }

        if (WIFSTOPPED(status)){
            /* I am here if a process is stopped */
            /* I have to put this process to the end of the list and continue with the next one process
according to the priorities */
            if (head == NULL) {
                printf("Empty list\n");
                exit(0);
            }
            if (head->next != NULL){ //an head->next == NULL, tote exo mono mia diergasia kai
sunexizo stin ektelesi tis
                //prota tha steilo to head sto telos kai epeita tha psakso an uparxoun high priorities
                //akolouthontas to idio skeptiko me prin
                end->next=head;
                end=head;
                head=head->next;
                end->next=NULL;
                nodeptr hd=head;
                nodeptr temp = head;

                while (temp != NULL){
                    if (temp->prio!=1){
                        /* In this case i have to put the process at the end of the list */
                        end->next=head;
                        end=head;
                        head=head->next;
```

```
                        end->next=NULL;
                        temp=head;
                        if (temp == hd){
                            flag = 0;
                            break;
                        }
                    }
                    else{
                        flag = 1 ;
                        break;
                    }
                }
            }
        }

    //gia na mpoume sto shell prepei na erthei i seira tou + na exei prio 1 i na mhn uparxei diergasia
me prio 1
        if( head->id == 0 && (flag == 0 || head->prio == 1)  ){
                printf("I am in Shell again:\n");
                alarm(5*SCHED_TQ_SEC);
        }
        else alarm(SCHED_TQ_SEC);

        /* Send SIGCONT to the head of the list */
        if ((kill(head->pid,SIGCONT))<0){
            perror("sigchld: kill");
            exit(3);
        }
    }

}


/* 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);
```

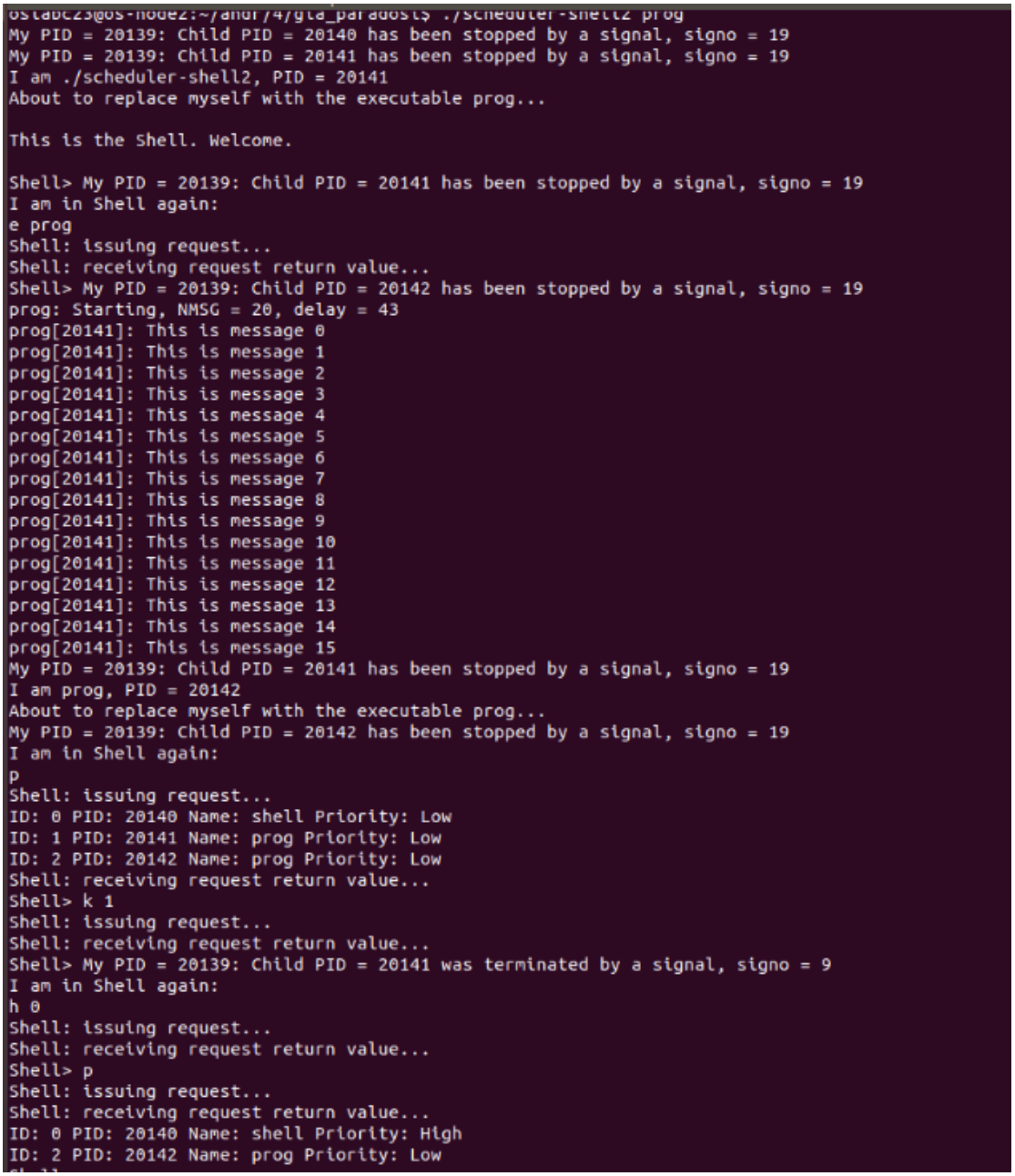
```
	if (sigprocmask(SIG_UNBLOCK, &sigset, NULL) < 0) {
		perror("signals_enable: sigprocmask");
		exit(1);
	}
}


/* 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];

	/* I have to insert the Shell as the head of my list */
	head->id=0;
	head->pid=p;
	head->prio = 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;

	/* Initialize the head and the end of the list to put the Shell on the head and the end as well
because at the start Shell will be the only node-process */
	head=(nodeptr)malloc(sizeof(Node));
	if(head == NULL){
		perror("malloc");
		exit(2);
	}
	end=head;

	/* Create the Shell and put Shell at the head of the list*/

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

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

```
    nodeptr newnode;
    pid_t mypid;

    for (i=1;i<=nproc;i++){
        mypid=fork();
        if (mypid<0){
            perror("main: fork");
        }
        if(mypid==0){
            char executable[] = "prog";
            char *newargv[] = {executable, NULL, NULL, NULL};
            char *newenviron[] = {NULL};

            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{
            newnode=(nodeptr)malloc(sizeof(Node));
            if (newnode == NULL){
                perror("malloc");
                exit(2);
            }
            newnode->pid=mypid;
            newnode->next=NULL;
            newnode->id=i;
            newnode->name=argv[i];
            newnode->prio = 0;
            end->next=newnode;
            end=newnode;
        }
    }
    serial_number = i;
    /* 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->pid,SIGCONT); //wake up the shell

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

Σας παραθέτουμε τα στιγμιότυπα ενός παραδείγματος εκτέλεσης του προγράμματος, στο οποίο χρησιμοποιήθηκαν όλες οι πιθανές εντολές του φλοιού.



```
Shell> e prog
Shell: issuing request...
Shell: receiving request return value...
Shell> My PID = 20139: Child PID = 20143 has been stopped by a signal, signo = 19
I am in Shell again:
p
Shell: issuing request...
Shell: receiving request return value...
ID: 0 PID: 20140 Name: shell Priority: High
ID: 2 PID: 20142 Name: prog Priority: Low
ID: 3 PID: 20143 Name: prog Priority: Low
Shell> h 2
Shell: issuing request...
Shell: receiving request return value...
Shell> My PID = 20139: Child PID = 20140 has been stopped by a signal, signo = 19
prog: Starting, NMSG = 20, delay = 155
prog[20142]: This is message 0
prog[20142]: This is message 1
prog[20142]: This is message 2
prog[20142]: This is message 3
prog[20142]: This is message 4
My PID = 20139: Child PID = 20142 has been stopped by a signal, signo = 19
I am in Shell again:
My PID = 20139: Child PID = 20140 has been stopped by a signal, signo = 19
prog[20142]: This is message 5
prog[20142]: This is message 6
prog[20142]: This is message 7
prog[20142]: This is message 8
My PID = 20139: Child PID = 20142 has been stopped by a signal, signo = 19
I am in Shell again:
p
Shell: issuing request...
Shell: receiving request return value...
ID: 0 PID: 20140 Name: shell Priority: High
ID: 2 PID: 20142 Name: prog Priority: High
ID: 3 PID: 20143 Name: prog Priority: Low
Shell> l 2
Shell: issuing request...
Shell: receiving request return value...
Shell> p
Shell: issuing request...
Shell: receiving request return value...
ID: 0 PID: 20140 Name: shell Priority: High
ID: 2 PID: 20142 Name: prog Priority: Low
ID: 3 PID: 20143 Name: prog Priority: Low
Shell> My PID = 20139: Child PID = 20140 has been stopped by a signal, signo = 19
I am in Shell again:
l 0
Shell: issuing request...
Shell: receiving request return value...
Shell> p
Shell: issuing request...
Shell: receiving request return value...
ID: 0 PID: 20140 Name: shell Priority: Low
ID: 2 PID: 20142 Name: prog Priority: Low
```

```
ID: 2 PID: 20142 Name: prog Priority: Low
ID: 3 PID: 20143 Name: prog Priority: Low
Shell> My PID = 20139: Child PID = 20140 has been stopped by a signal, signo = 19
prog[20142]: This is message 9
prog[20142]: This is message 10
prog[20142]: This is message 11
prog[20142]: This is message 12
My PID = 20139: Child PID = 20142 has been stopped by a signal, signo = 19
I am prog, PID = 20143
About to replace myself with the executable prog...
My PID = 20139: Child PID = 20143 has been stopped by a signal, signo = 19
I am in Shell again:
My PID = 20139: Child PID = 20140 has been stopped by a signal, signo = 19
prog[20142]: This is message 13
prog[20142]: This is message 14
prog[20142]: This is message 15
prog[20142]: This is message 16
prog[20142]: This is message 17
My PID = 20139: Child PID = 20142 has been stopped by a signal, signo = 19
prog: Starting, NMSG = 20, delay = 137
prog[20143]: This is message 0
prog[20143]: This is message 1
prog[20143]: This is message 2
prog[20143]: This is message 3
prog[20143]: This is message 4
My PID = 20139: Child PID = 20143 has been stopped by a signal, signo = 19
I am in Shell again:
q
Shell: Exiting. Goodbye.
My PID = 20139: Child PID = 20140 terminated normally, exit status = 0
scheduler: read from shell: Success
Scheduler: giving up on shell request processing.
prog[20142]: This is message 18
prog[20142]: This is message 19
My PID = 20139: Child PID = 20142 terminated normally, exit status = 0
prog[20143]: This is message 5
prog[20143]: This is message 6
prog[20143]: This is message 7
prog[20143]: This is message 8
prog[20143]: This is message 9
My PID = 20139: Child PID = 20143 has been stopped by a signal, signo = 19
prog[20143]: This is message 10
prog[20143]: This is message 11
prog[20143]: This is message 12
prog[20143]: This is message 13
prog[20143]: This is message 14
My PID = 20139: Child PID = 20143 has been stopped by a signal, signo = 19
prog[20143]: This is message 15
prog[20143]: This is message 16
prog[20143]: This is message 17
prog[20143]: This is message 18
prog[20143]: This is message 19
My PID = 20139: Child PID = 20143 has been stopped by a signal, signo = 19
My PID = 20139: Child PID = 20143 terminated normally, exit status = 0
The list is empty...I don't have any more work to do
```

Ερωτήσεις:

1. Περιγράψτε ένα σενάριο δημιουργίας λιμοκτονίας.

Λιμοκτονία ονομάζεται το φαινόμενο μία διεργασία χαμηλής προτεραιότητας να μην εκτελείται ποτέ, επειδή πάντα υπάρχουν διεργασίας υψηλότερης προτεραιότητας που προηγούνται.
Αυτό στην περίπτωση μας θα μπορούσε να συμβεί, για παράδειγμα, αν δημιουργούσαμε συνεχώς διεργασίες και θέταμε την προτεραιότητα τους σε high, με αποτέλεσμα όσες έχουν low να μην εκτελούνται ποτέ. Ή και στην περίπτωση που θέταμε τον shell σε high priority, με αποτέλεσμα να χρονοδρομολογείται μόνο ο shell και καμία άλλη διεργασία.
Για να μην δώσουμε το περιθώριο να συμβεί κάτι τέτοιο, θα ήταν δυνατό να χρησιμοποιήσουμε τη μέθοδο της γήρανσης. Πιο συγκεκριμένα, θα μπορούσαμε να βάλουμε ένα επιπλέον πεδίο στα χαρακτηριστικά των διεργασιών, το οποίο θα είχε μια τιμή 0 στην αρχή. Κάθε φορά που εκτελούταν μια διεργασία, θα αυξάναμε το πεδίο κατά 1. Έτσι, όταν η τιμή του πεδίου αυτού φτάσει κάποιο προκαθορισμένο νούμερο, θα θέταμε το priority της σε high, με αποτέλεσμα να χρονοδρομολογηθεί με την πρώτη ευκαιρία.