ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ
ΥΠΟΛΟΓΙΣΤΩΝ ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ
ΥΠΟΛΟΓΙΣΤΩΝ ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

Άσκηση 4

Χρονοδρομολόγηση

Λειτουργικά Συστήματα

Εργαστηρική Ομάδα Δ02

Βαβουλιώτης Γεώργιος
Α.Μ. : 03112083
7ο εξάμηνο

Γιαννούλας Βασίλειος
Α.Μ. : 03112117
7ο εξάμηνο

Ακ. Έτος
2015-2016

1.1 Υλοποίηση χρονοδρομολογητή κυκλικής επαναφοράς στο χώρο χρήστη

Στόχος της άσκησης αυτής είναι η υλοποίηση ενός χρονοδρομολογητή κυκλικής επαναφοράς (Round-Robin) ο οποίος εκτελείται ως γονική διεργασία στο χώρο χρήστη, κατανέμοντας τον υπολογιστικό χρόνο σε διεργασίες-παιδιά. Θα πρέπει να τονιστεί ότι κάθε διεργασία εκτελείται το πολύ για χρονικό διάστημα ίσο με το κβάντο χρόνου tq. Αν τώρα το κβάντο χρόνου λήξει, ο scheduler σταματάει την διεργασία με χρήση του σήματος SIGSTOP και ενεργοποιεί την επόμενη διεργασία με χρήση του σήματος SIGCONT(παρακάτω εξηγώ αναλυτικά τι θα συμβεί).

Λεπτομέρειες υλοποίησης **:** Η ουρά διεργασιών έχει υλοποιηθεί με χρήση απλά συνδεδεμένης λίστας. Υπάρχει ενας pointer με όνομα head, ο οποίος δείχνει κάθε φορά στην αρχή της λίστας και ενας pointer με όνομα end, ο οποίος δείχνει κάθε φορά στο τέλος της λίστας.

Επεξήγηση ιδέας **:** Αρχικά δημιουργώ την λίστα με τις διεργασίες που καλούμαι να χρονοδρομολογήσω, κάνοντας fork() για κάθε ένα απο τα προγράμματα prog που έδωσα σαν όρισμα στο εκτελέσιμο της άσκησης(ουσιαστικά καλούμαι να χρονοδρομολογήσω τα προγράμματα prog που έδωσα σαν όρισμα). Σε κάθε διεργασία που δημιουργώ στέλνω σήμα SIGSTOP ώστε όταν τελικά ολοκληρωθεί η δημιουργία της λίστας όλες οι διεργασίες να είναι σταματημένες. Η συνάρτηση που περιμένει να ολοκληρωθεί η δημιουργία της λίστας είναι η wait_for_ready_children(). Στο σημείο αυτό στέλνω σήμα SIGCONT στο head της λίστας ώστε να ενεργοποιηθεί η πρώτη διεργασία. Κάθε διεργασία μπορεί είτε να μην τελειώσει μέσα στο κβάντο χρόνου της είτε να τερματίσει μέσα στο κβάντο χρόνου της. Πιθανό είναι επίσης κάποια διεργασία να ‘φάει’ σήμα kill κατά την εκτέλεση του προγράμματος. Για να έχουμε ενα συνεπές και ορθό πρόγραμμα χρησιμοποιούμε δυο handlers, τον sigalrm_handler και τον sigchld_handler. Ο πρώτος handler ενεργοποιείται όταν τελειώσει το κβάντο χρόνου μιας διεργασίας(δηλαδή όταν έρθει σήμα SIGALRM) και αυτό που κάνει είναι να στέλνει σήμα SIGSTOP στην διεργασία που έτρεχε. Ο δεύτερος handler ενεργοποιείται όταν προκύψει κάποια αλλαγή στην κατάσταση κάποιας διεργασίας(δηλαδή όταν έρθει σήμα SIGCHLD). Αυτο λοιπόν που κάνουμε είναι όταν μια διεργασία ‘φάει’ σήμα kill ή τερματίσει την λειτουργία της κανονικά να την αφαιρούμε απο την λίστα ενώ όταν μια διεργασία δεν καταφέρει να ολοκληρώσει τη λειτουργία της μέσα στο κβάντο χρόνου την βάζουμε στο τέλος της λίστας. Θα πρέπει επίσης να τονιστεί πως η διεργασία που είναι ενεργή κάθε φορά είναι αυτή που βρίσκεται στο head της λίστας μας.

Ο κώδικας του προγράμματος scheduler.c παρατίθεται στη συνέχεια:

```
#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 id; //my pid
 char * name;
 int data;
 struct Node *  next;
}Node;
typedef Node * nodeptr;

nodeptr head = NULL;
nodeptr end = NULL;

/*
 * SIGALRM handler
 */
static void
sigalrm_handler(int signum)
{
 kill(head->id,SIGSTOP);
}

/*
 * 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->id == p && current == head){
   /* I have to delete the head of my list */
    if (current->next == NULL){
     /*I come here when i have only one node.i delete it and i am done*/
      free(current);
      printf("My list is empty...I don't have any more work to do\n");
      exit(0);
    }
    else{
     head=current->next;
     free(current);
    }
  }
  else if (current->id == p && current == end){
   /* I have to delete the last node of my list */
   end=previous;
   end->next=NULL;
   free(current);
  }
  else if(current->id == p){
   /* 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 m process is stopped */
   /*Transfer this process at the end of list and continue with the next*/
   nodeptr temp;
   end->next=head;
   end=head;
   temp=head;
   head=head->next;
   temp->next=NULL;
   /* Only here i have to set the alarm again */
   alarm(SCHED_TQ_SEC);
  }

  /* Alarm Set */
  //alarm(SCHED_TQ_SEC);

  /* I always send SIGCONT to the head of my list */
  kill(head->id,SIGCONT);
 }
}

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

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

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

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



int main(int argc, char *argv[])
{
 /*
  * For each of argv[1] to argv[argc - 1],
  * create a new child process, add it to the process list.
  */
 /* nproc tells us how many processes i gave us arguments */
 int nproc = argc-1, i;
 pid_t p;
 nodeptr newnode = NULL;
 char executable[] = "prog";
        char *newargv[] = { executable, NULL, NULL, NULL };
        char *newenviron[] = { NULL };

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

 /*Fork and create the list*/
 for (i=0;i<nproc;i++){
  p=fork();
  if (p<0){
   perror("fork");
  }
  if(p == 0){
   /* Child Space */
   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() only returns on error */
   execve(executable, newargv, newenviron);
   perror("execve");
   exit(1);
  }
```

```
 else{
  /* Father Space */
  if( i == 0){
   /* I have to create the head of the list */
   head=(nodeptr)malloc(sizeof(Node));
   if(head == NULL){
     perror("malloc");
     exit(2);
   }
   head->id=p;
   head->next=NULL;
   head->data=i+1;
   head->name=argv[i+1];
   end=head; // i do that beacause is the first node ==> head == end

  }
  else{
   /*Create a new node of the list and put it at the end of the list */
   newnode = (nodeptr)malloc(sizeof(Node));
   if(newnode == NULL){
    perror("malloc");
    exit(2);
   }
   newnode->id=p;
   newnode->next=NULL;
   newnode->data=i+1;
   newnode->name=argv[i+1];
   end->next=newnode;
   end=newnode;
  }

 }


}



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

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

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

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

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

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

Ακολουθεί στιγμιότυπο για κλήση του scheduler με δημιουργία και χρονοδρομολόγηση δυο προγραμμάτων prog.c(για να τερματίζει κάθε διεργασία σχετικά γρήγορα ώστε να μπορέσουμε να δούμε το πρόγραμμα να τελειώνει αλλάζουμε την εντολή #define NMSG 200 του αρχείου prog.c με την εντολή #define NMSG 50):

```
oslabd02@poros:~/Exercise4/Askhsh1.1$ ./scheduler prog prog
My PID = 17151: Child PID = 17152 has been stopped by a signal, signo = 19
My PID = 17151: Child PID = 17153 has been stopped by a signal, signo = 19
I am ./scheduler, PID = 17152
About to replace myself with the executable prog...
My PID = 17151: Child PID = 17152 has been stopped by a signal, signo = 19
I am ./scheduler, PID = 17153
About to replace myself with the executable prog...
My PID = 17151: Child PID = 17153 has been stopped by a signal, signo = 19
prog: Starting, NMSG = 50, delay = 148
prog[17152]: This is message 0
prog[17152]: This is message 1
prog[17152]: This is message 2
prog[17152]: This is message 3
prog[17152]: This is message 4
prog[17152]: This is message 5
My PID = 17151: Child PID = 17152 has been stopped by a signal, signo = 19
prog: Starting, NMSG = 50, delay = 64
prog[17153]: This is message 0
prog[17153]: This is message 1
prog[17153]: This is message 2
prog[17153]: This is message 3
prog[17153]: This is message 4
prog[17153]: This is message 5
prog[17153]: This is message 6
prog[17153]: This is message 7
prog[17153]: This is message 8
prog[17153]: This is message 9
prog[17153]: This is message 10
prog[17153]: This is message 11
prog[17153]: This is message 12
prog[17153]: This is message 13
My PID = 17151: Child PID = 17153 has been stopped by a signal, signo = 19
prog[17152]: This is message 6
prog[17152]: This is message 7
prog[17152]: This is message 8
prog[17152]: This is message 9
prog[17152]: This is message 10
prog[17152]: This is message 11
My PID = 17151: Child PID = 17152 has been stopped by a signal, signo = 19
prog[17153]: This is message 14
prog[17153]: This is message 15
prog[17153]: This is message 16
prog[17153]: This is message 17
prog[17153]: This is message 18
prog[17153]: This is message 19
prog[17153]: This is message 20
prog[17153]: This is message 21
prog[17153]: This is message 22
prog[17153]: This is message 23
prog[17153]: This is message 24
prog[17153]: This is message 25
```

```
prog[17153]: This is message 26
prog[17153]: This is message 27
My PID = 17151: Child PID = 17153 has been stopped by a signal, signo = 19
prog[17152]: This is message 12
prog[17152]: This is message 13
prog[17152]: This is message 14
prog[17152]: This is message 15
prog[17152]: This is message 16
prog[17152]: This is message 17
My PID = 17151: Child PID = 17152 has been stopped by a signal, signo = 19
prog[17153]: This is message 28
prog[17153]: This is message 29
prog[17153]: This is message 30
prog[17153]: This is message 31
prog[17153]: This is message 32
prog[17153]: This is message 33
prog[17153]: This is message 34
prog[17153]: This is message 35
prog[17153]: This is message 36
prog[17153]: This is message 37
prog[17153]: This is message 38
prog[17153]: This is message 39
prog[17153]: This is message 40
prog[17153]: This is message 41
My PID = 17151: Child PID = 17153 has been stopped by a signal, signo = 19
prog[17152]: This is message 18
prog[17152]: This is message 19
prog[17152]: This is message 20
prog[17152]: This is message 21
prog[17152]: This is message 22
prog[17152]: This is message 23
My PID = 17151: Child PID = 17152 has been stopped by a signal, signo = 19
prog[17153]: This is message 42
prog[17153]: This is message 43
prog[17153]: This is message 44
prog[17153]: This is message 45
prog[17153]: This is message 46
prog[17153]: This is message 47
prog[17153]: This is message 48
prog[17153]: This is message 49
My PID = 17151: Child PID = 17153 terminated normally, exit status = 0
prog[17152]: This is message 24
prog[17152]: This is message 25
prog[17152]: This is message 26
My PID = 17151: Child PID = 17152 has been stopped by a signal, signo = 19
prog[17152]: This is message 27
prog[17152]: This is message 28
prog[17152]: This is message 29
prog[17152]: This is message 30
prog[17152]: This is message 31
prog[17152]: This is message 32
My PID = 17151: Child PID = 17152 has been stopped by a signal, signo = 19
prog[17152]: This is message 33
prog[17152]: This is message 34
prog[17152]: This is message 35
prog[17152]: This is message 36
prog[17152]: This is message 37
prog[17152]: This is message 38
My PID = 17151: Child PID = 17152 has been stopped by a signal, signo = 19
prog[17152]: This is message 39
prog[17152]: This is message 40
prog[17152]: This is message 41
prog[17152]: This is message 42
prog[17152]: This is message 43
prog[17152]: This is message 44
My PID = 17151: Child PID = 17152 has been stopped by a signal, signo = 19
prog[17152]: This is message 45
prog[17152]: This is message 46
prog[17152]: This is message 47
prog[17152]: This is message 48
prog[17152]: This is message 49
My PID = 17151: Child PID = 17152 terminated normally, exit status = 0
My list is empty...I don't have any more work to do
oslabd02@poros:~/Exercise4/Askhsh1.1$
```

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;

Αν υποθέσουμε οτι εκτελείται η συνάρτηση χειρισμού του σήματος SIGCHLD και έρθει σήμα SIGALRM τότε δεν εκτελείται η συνάρτηση χειρισμού του σήματος SIGALRM. Αυτό γίνεται διότι η δοσμένη συνάρτηση *install_signal_handlers()* ορίζει το μπλοκάρισμα του σήματος SIGALRM με χρήση κατάλληλης μάσκας, όταν εκτελείται η συνάρτηση χειρισμού του σήματος SIGCHLD. Προφανώς το ίδιο συμβαίνει και στην περίπτωση όπου εκτελείται η συνάρτηση χειρισμού του σήματος SIGALRM και έρθει σήμα SIGCHLD.
Τώρα ένας πραγματικός χρονοδρομολογητής χώρου πυρήνα λειτουργεί με διακοπές και όχι με σήματα. Μ'αυτό πετυχαίνει μεγαλύτερη αποκρισιμότητα διότι όταν μια διακοπή ενεργοποιηθεί τότε απ'ευθείας ενεργοποιείται ρουτίνα εξυπηρέτηση της αντίστοιχης διακοπής. Αντίθετα, στη περίπτωσή μας μπορεί να υπάρξει καθυστέρηση στην ενεργοποίηση της συνάρτησης χειρισμού κάποιου σήματος διότι ακόμα και τα σήματα δρομολογούνται. Κάτι τέτοιο δεν είναι επιθυμητό και γι'αυτό το λόγο οι χρονοδρομολογητές πυρήνα επιλέγουν τη χρήση διακοπών απο τη χρήση σημάτων.

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;

Αρχικά πρέπει να τονιστεί οτι ο χρονοδρομολογητής λαμβάνει σήμα SIGCHLD όταν έχει αλλάξει η κατάσταση κάποιου παιδιού, δηλαδή όταν κάποιο παιδί έχει πεθάνει κανονικά ή όταν το παιδί 'φάει' κάποιο σήμα kill ή έχει σταματήσει λόγω σήματος SIGSTOP από τον χρονοδρομολογητή(τελείωσε το κβάντο χρόνου του). Όπως φαίνεται και στο δοσμένο κώδικα ο διαχωρισμός αυτών των περιπτώσεων γίνεται μέσα στο συνάρτηση χειρισμού του σήματος SIGCHLD, με χρήση αρχικά της συνάρτησης *waitpid(),* η οποία μας λέει ποίο είναι το παιδί του οποίου άλλαξε η κατάσταση και μετά η συνάρτηση *explain_wait_status()* μας δίνει την πληροφορία σχετικά με το τι έπαθε το παιδί και άλλαξε η κατάσταση του. Στη συνέχεια με διαδοχικές εντολές *if-else* ανάλογα με το τι έπαθε το κάθε παιδί κάνουμε τις κατάλληλες ενέργειες ώστε να παραμένει το πρόγραμμα ορθό(οι ενέργειες αυτές αναφέρθηκαν παραπάνω στην επεξήγηση της ιδέας).
Όπως αναφέρθηκε και παραπάνω αν λόγω εξωτερικού παράγοντα τερματιστεί αναπάντεχα μια οποιαδήποτε διεργασία(πχ στέλνω σήμα SIGKILL), τότε ο scheduler θα λάβει σήμα SIGCHLD με αποτέλεσμα να χειριστεί την περίπτωση αυτή όπως πρέπει, δηλαδή βρίσκει το παιδί του δέχτηκε το σήμα αυτό, το αφαιρεί απο τη λίστα και το αποτέλεσμα είναι συνεπές.

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;

Η υλοποίηση του χρονοδρομολογητή με χειρισμό δύο σημάτων είναι αναγκαία στην άσκηση αυτή για λόγους συγχρονισμού και ορθότητας. Αν υποθέσουμε ότι ο scheduler χρησιμοποιούσε μόνο το σήμα SIGALRM για την εναλλαγή διεργασιών, τότε όπως ανέφερα και σε παραπάνω ερώτηση θα μπορούσε το σήμα SIGSTOP που στέλνεται στην διεργασία που θα πρέπει να σταματήσει, να παραδοθεί μετά απο το σήμα SIGCONT το οποίο λαμβάνεται από τη διεργασία που θα πρέπει να ενεργοποιηθεί αφού όμως σταματήσει η προηγούμενη. Απο τη τελευταία πρόταση καταλαβαίνουμε πως μπορεί να προκύψει μη έγκυρο στιγμιότυπο για τον scheduler, δηλαδή να είναι ενεργοποιημένες δυο διεργασίες ταυτόχρονα, γεγονός το οποίο δεν θέλουμε.
Στην υλοποίηση με χειρισμό δυο σημάτων είμαστε σίγουροι πως μόνο μια διεργασία θα είναι ενεργοποιημένη κάθε φορά αφού δεν υπάρχει ενδεχόμενο να προκύψει ποτέ η παραπάνω περίπτωση διότι όταν έρθει σήμα SIGALRM στέλνουμε σήμα SIGSTOP στη αντίστοιχη διεργασία που οφείλει να σταματήσει και έπειτα αποστέλλεται σήμα SIGCHLD το οποίο το αντιλαμβάνεται ο χρονοδρομολογητής και στη συνέχεια στέλνει το σήμα SIGCONT στη διεργασία που πρόκειται να ενεργοποιηθεί. Επομένως η υλοποίηση με χειρισμό δυο σημάτων εγγυάται τη διακοπή της μίας διεργασίας πριν τη συνέχιση της λειτουργίας της επόμενης αφού χρονοδρομολογούνται ακόμα και τα σήματα .

1.2 Έλεγχος λειτουργίας χρονοδρομολογητή μέσω φλοιού

Στην άσκηση αυτή απλά επεκτείνεται ο χρονοδρομολογητής του προηγούμενου ερωτήματος ώστε να υποστηρίζεται ο έλεγχος της λειτουργίας του μέσω προγράμματος φλοιού. Ο φλοιός δρομολογείται κανονικά μαζί με τις υπόλοιπες διεργασίες, δέχεται εντολές από το χρήση και κατασκευάζει αιτήσεις κατάλληλης μορφής τις οποίες αποστέλλει προς τον χρονοδρομολογητή. Οι εντολές του φλοιού είναι οι εξής :

• Εντολή'p':Οχρονοδρομολογητήςεκτυπώνειστηνέξοδολίσταμετιςυπόεκτέλεση διεργασίες, στον οποίο φαίνεται ο σειριακός αριθμός id της διεργασίας, το PID και το όνομα της. Επιπλέον, επισημαίνεται η τρέχουσα διεργασία.

• Εντολή 'k': Δέχεται όρισμα το id μιας διεργασίας (προσοχή: όχι το PID) και ζητά από το χρονοδρομολογητή τον τερματισμό της.

• Εντολή 'e': Δέχεται όρισμα το όνομα ενός εκτελέσιμου στον τρέχοντα κατάλογο, π.χ. prog2 και ζητά τη δημιουργία μιας νέας διεργασίας από τον χρονοδρομολο- γητή, στην οποία θα τρέχει αυτό το εκτελέσιμο.

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

Η λογική πάνω στην οποία στηρίχτηκε ο κώδικας της άσκησης αυτής(τον οποίο σας τον παραθέτω παρακάτω) είναι η ακόλουθη: Η συλλογιστική πορεία είναι ίδια με αυτή της πρώτης άσκησης μόνο που πλέον κατά τη δημιουργία της λίστας βάζουμε το schell στο head της λίστας και τον χειριζόμαστε όπως τις άλλες διεργασίες που βρίσκονται στη λίστα. Επίσης υλοποιήσαμε τις συναρτήσεις *sched_kill_task_by_id()* και *sched_create_task()*. Η πρώτη ενεργοποιείται όταν όταν πατήσουμε στον shell **k <id_task>**(το οποίο σημαίνει οτι θέλουμε να 'σκοτώσουμε' την διεργασία με id to id_task) και αυτό που κάνει είναι να ψάχνει στην λίστα να βρεί αν υπάρχει η διεργασία με το συγκεκριμένο id και αν την βρεί τις στέλνει ενα σήμα SIGKILL(μετά θα σταλθεί σήμα SIGCHLD αφού άλλαξε η κατάσταση μιας διεργασίας και μετά κατά τα γνωστά θα φροντίσει η συνάρτηση χειρισμού του SIGCHLD να διαγράψει την διεργασία αυτή).

Η δέυτερη συνάρτηση ενεργοποιείται όταν πατήσουμε στο shell **e <process_name>**, δηλαδή όταν θέλουμε να δημιουργήσουμε, δυναμικά πλέον, μια νέα διεργασία. Αυτό που κάνει είναι να δημιουργεί και να τοποθετεί την νέα αυτη διεργασία στο τέλος της ήδη υπάρχουσας λίστας.

Το τροποποιημένο πρόγραμμα είναι το ακόλουθο:

```
#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 id;
        struct Node* next;
        char* name;
        int data;
}Node;
typedef Node * nodeptr;

/*Definition of my list nodes*/
nodeptr head=NULL;
nodeptr end=NULL;
int i=0,nproc;

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

}
```

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

static int
sched_kill_task_by_id(int id)
{
 nodeptr temp=head;
 while (temp!=NULL){
  if (temp->data == id ){
   kill(temp->id,SIGKILL);
   return 0;
  }
  else
   if(temp->next!=NULL) temp=temp->next;
   else break;
 }
 return -ENOSYS;
}

/* Create a new task.  */
static void
sched_create_task(char *executable)//
{
  nodeptr newnode;

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

  pid_t mypid;
  mypid=fork();
  if (mypid < 0){
     perror("fork");
  }
  if(mypid == 0){
     raise(SIGSTOP);
     printf("I am %s, PID = %ld\n",executable, (long)getpid());
     printf("About to replace myself with the executable %s…
            \n",executable);
     sleep(2);
     execve(executable, newargv, newenviron);
     /* execve() only returns on error */
     perror("execve");
     exit(1);
```

```
   }
   else{
   /* Insert the new process to the list */
        newnode=(nodeptr)malloc(sizeof(Node));
        if (newnode == NULL){
           perror("malloc");
           exit(2);
        }
          newnode->id=mypid;
          newnode->next=NULL;
          newnode->data=i+1;
          newnode->name=(char*)malloc(strlen(executable)+1);
          strcpy(newnode->name,executable);
          end->next=newnode;
          end=newnode;
          nproc++;
          i++;
    }
}

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

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

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

  default:
   return -ENOSYS;
 }
}
```

```
/*
 * SIGALRM handler
 */
static void
sigalrm_handler(int signum)
{
        kill(head->id,SIGSTOP);
}

/*
 * 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->id == p && current == head){
     /* I have to delete the head of my list */
      if (current->next == NULL){
       /* I come here when i have one node,delete it and i am done */
        free(current);
```

```
      printf("My list is empty...I don't have any more work to do\n");
      exit(0);
    }
    else{
     head=current->next;
     free(current);
    }
  }
  else if (current->id == p && current == end){
   /* I have to delete the last node of my list */
   end=previous;
   end->next=NULL;
   free(current);
  }
  else if(current->id == p){
   /*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 */
 /*Transfer this process at the end of list and continue with the next*/
 nodeptr temp;
 end->next=head;
 end=head;
 temp=head;
 head=head->next;
 temp->next=NULL;
 /* Only here i have to set the alarm again */
 alarm(SCHED_TQ_SEC);
}
```

```
  if(head->data==0){
   printf("I am in Shell again:\n");
    /* use alarm to stay in shell for some seconds */
   alarm(3*SCHED_TQ_SEC);
  }

  /* Send SIGCONT to the head of the list */
   kill(head->id,SIGCONT);
 }

}


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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

 nodeptr newnode;
 pid_t mypid;
```

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

                        execve(executable, newargv, newenviron);

                        /* execve() only returns on error */
                        perror("execve");
                        exit(1);
                        }
                else{
                    /* I dont have to check if my head is NULL because
Shell is in the head for sure */
                        newnode=(nodeptr)malloc(sizeof(Node));
                        if (newnode == NULL){
                             perror("malloc");
                             exit(2);
                        }
                        newnode->id=mypid;
                        newnode->next=NULL;
                        newnode->data=i+1;
                        newnode->name=argv[i+1];
                        end->next=newnode;
                        end=newnode;
                }
        }


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

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

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

 /*Start the first process(shell)*/
 kill(head->id,SIGCONT);
 shell_request_loop(request_fd, return_fd);

 /* Now that the shell is gone, just loop forever
  * until we exit from inside a signal handler.
  */
 while (pause())
  ;
 /* Unreachable */
 fprintf(stderr, "Internal error: Reached unreachable point\n");
 return 1;
}
```

Ακολουθεί στιγμιότυπο για κλήση του scheduler με δημιουργία και χρονοδρομολόγηση δυο προγραμμάτων prog.c(για να τερματίζει μια διεργασία σχετικά γρήγορα ώστε να μπορέσουμε να δούμε το πρόγραμμα να τελειώνει αλλάζουμε την εντολή #define NMSG 200 του αρχείου prog.c με την εντολή #define NMSG 30):

```
oslabd02@zakynthos:~/Exercise4/Askhsh1.2$ ./scheduler-shell prog prog
My PID = 7063: Child PID = 7065 has been stopped by a signal, signo = 19
My PID = 7063: Child PID = 7066 has been stopped by a signal, signo = 19
My PID = 7063: Child PID = 7064 has been stopped by a signal, signo = 19
I am ./scheduler-shell, PID = 7065
About to replace myself with the executable prog...
My PID = 7063: Child PID = 7065 has been stopped by a signal, signo = 19
I am ./scheduler-shell, PID = 7066
About to replace myself with the executable prog...
My PID = 7063: Child PID = 7066 has been stopped by a signal, signo = 19
I am in Shell again:

This is the Shell. Welcome.

Shell> e prog
Shell: issuing request...
Shell: receiving request return value...
Shell> My PID = 7063: Child PID = 7067 has been stopped by a signal, signo = 19
prog: Starting, NMSG = 30, delay = 151
prog[7065]: This is message 0
prog[7065]: This is message 1
prog[7065]: This is message 2
prog[7065]: This is message 3
prog[7065]: This is message 4
prog[7065]: This is message 5
prog[7065]: This is message 6
My PID = 7063: Child PID = 7065 has been stopped by a signal, signo = 19
prog: Starting, NMSG = 30, delay = 132
prog[7066]: This is message 0
prog[7066]: This is message 1
prog[7066]: This is message 2
prog[7066]: This is message 3
prog[7066]: This is message 4
prog[7066]: This is message 5
prog[7066]: This is message 6
prog[7066]: This is message 7
My PID = 7063: Child PID = 7066 has been stopped by a signal, signo = 19
I am prog, PID = 7067
About to replace myself with the executable prog...
My PID = 7063: Child PID = 7067 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: 7064 Name: shell
ID: 1 PID: 7065 Name: prog
ID: 2 PID: 7066 Name: prog
ID: 3 PID: 7067 Name: prog
Shell> k 1
Shell: issuing request...
Shell: receiving request return value...
Shell> My PID = 7063: Child PID = 7065 was terminated by a signal, signo = 9
```

```
I am in Shell again:
q
Shell: Exiting. Goodbye.
My PID = 7063: Child PID = 7064 terminated normally, exit status = 0
scheduler: read from shell: Success
Scheduler: giving up on shell request processing.
prog[7066]: This is message 8
prog[7066]: This is message 9
prog[7066]: This is message 10
prog[7066]: This is message 11
prog[7066]: This is message 12
prog[7066]: This is message 13
prog[7066]: This is message 14
prog[7066]: This is message 15
prog[7066]: This is message 16
prog[7066]: This is message 17
prog[7066]: This is message 18
prog[7066]: This is message 19
prog[7066]: This is message 20
prog[7066]: This is message 21
prog[7066]: This is message 22
prog[7066]: This is message 23
prog[7066]: This is message 24
prog[7066]: This is message 25
prog[7066]: This is message 26
prog[7066]: This is message 27
prog[7066]: This is message 28
prog[7066]: This is message 29
My PID = 7063: Child PID = 7066 terminated normally, exit status = 0
prog: Starting, NMSG = 30, delay = 114
prog[7067]: This is message 0
prog[7067]: This is message 1
prog[7067]: This is message 2
prog[7067]: This is message 3
prog[7067]: This is message 4
prog[7067]: This is message 5
prog[7067]: This is message 6
prog[7067]: This is message 7
prog[7067]: This is message 8
prog[7067]: This is message 9
My PID = 7063: Child PID = 7067 has been stopped by a signal, signo = 19
prog[7067]: This is message 10
prog[7067]: This is message 11
prog[7067]: This is message 12
prog[7067]: This is message 13
prog[7067]: This is message 14
prog[7067]: This is message 15
prog[7067]: This is message 16
prog[7067]: This is message 17
prog[7067]: This is message 18
My PID = 7063: Child PID = 7067 has been stopped by a signal, signo = 19
prog[7067]: This is message 19
prog[7067]: This is message 20
prog[7067]: This is message 21
prog[7067]: This is message 22
prog[7067]: This is message 23
prog[7067]: This is message 24
prog[7067]: This is message 25
prog[7067]: This is message 26
My PID = 7063: Child PID = 7067 has been stopped by a signal, signo = 19
prog[7067]: This is message 27
prog[7067]: This is message 28
prog[7067]: This is message 29
My PID = 7063: Child PID = 7067 terminated normally, exit status = 0
My list is empty...I don't have any more work to do
oslabd02@zakynthos:~/Exercise4/Askhsh1.2$
```

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;**

Προφανώς κάθε φορά που πατάμε την εντολή 'p' για να δούμε την λίστα με τις διεργασίες, ως τρέχουσα διεργασία εμφανίζεται πάντα η διεργασία με id 0 δηλαδή ο φλοιός. Κάτι τέτοιο είναι απόλυτα λογικό διότι η εκτύπωση των διεργασιών είναι δυνατόν να γίνει μόνο όταν τρέχουσα διεργασία είναι ο φλοιός. Δηλαδή δεν θα μπορούσε να φαίνεται άλλη διεργασία ως τρέχουσα διεργασία στη λίστα διεργασιών καθώς η εντολή 'p' δίνεται μόνο από το φλοιό και εκείνη τη στιγμή έχουν απενεργοποιηθεί τα υπόλοιπα σήματα.

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

Οι κλήσεις *signals_disable()* και *signals_enable()* είναι αναγκαίο να χρησιμοποιηθούν γύρω από τη συνάρτηση υλοποίησης αιτήσεων του φλοιού για να εξασφαλιστεί η συνέπεια στα δεδομένα της βάσης. Με την προσθήκη των συναρτήσεων αυτών εξασφαλίζεται ότι όσο εξυπηρετείται μία αίτηση του φλοιού δεν υπάρχει περίπτωση να γίνει χειρισμός άλλου σήματος που θα τροποποιήσει τις δομές που χρησιμοποιούνται. Αν δεν κάναμε τις κλήσεις αυτές τότε κατά τη διάρκεια εξυπηρέτησης αιτήσεων του φλοιού θα ήταν δυνατόν μία διεργασία να τροποποιήσει μία κοινή δομή όπως η λίστα με τις διεργασίες και να προκύψει μη αναμενόμενο αποτέλεσμα.

1.3 Υλοποίηση προτεραιοτήτων στο χρονοδρομολογητή

Στην άσκηση αυτή απλά επεκτείνεται ο χρονοδρομολογητής του προηγούμενου ερωτήματος ώστε να υποστηρίζονται δύο κλάσεις προτεραιότητας HIGH και LOW. Αν υπάρχουν διεργασίες προτεραιότητας HIGH εκτελούνται μόνο αυτές με κυκλική επαναφορά, διαφορετικά χρονοδρομολογούνται οι LOW διεργασίες πάλι με κυκλική επαναφορά.

Η λογική πάνω στην οποία στηρίχτηκε ο κώδικας της άσκησης αυτής(τον οποίο σας τον παραθέτω παρακάτω) είναι η ακόλουθη: Η συλλογιστική πορεία είναι όμοια με αυτή που χρησιμοποιήσαμε για την άσκηση 1.2 μόνο που έγιναν κάποιες προσθήκες ώστε να υποστηριχθούν οι κλάσεις προτεραιότητας HIGH και LOW. Οι προσθήκες είναι οι ακόλουθες : Αρχικά προσθέσαμε ενα νέο πεδίο στο struct κάθε διεργασίες με όνομα priority ώστε κάθε διεργασία να έχει την δική της προτεραιότητα. Όμοια με πριν βάλαμε τον shell στο head της λίστας. Θα πρέπει να τονιστεί πως αρχικά όλες οι διεργασίες που είναι μέσα στη λίστα έχουν priority = 0 όπως επίσης **και ο shell έχει priority = 0.** Επομένως εφόσον αρχικά όλες οι διεργασίες έχουν priority = 0 θα γίνεται αυτό που γινόταν και στις παραπάνω ασκήσεις με την κυκλική εναλλαγή διεργασιών. Αν τώρα γράψω την εντολή **h <id_task>** θέτω την το priority κάποιες διεργασίας στο 1 διότι καλείται η συνάρτηση sched_set_high_p() η οποία ψάχνει μέσα στη λίστα με τις διεργασίες να βρει την διεργασία με το id που παίρνει σαν όρισμα και βάζει στο πεδίο priority το 1.
Αντίστοιχα η εντολή **l <id_task>** κάνει το ίδιο ακριβώς αλλά θέτει το πεδίο priority στο 0. Επίσης κάθε νέα διεργασία που δημιουργούμε δυναμικά(με την εντολή **e <process_name>**), το οποίο γίνεται μέσω της συνάρτησης sched_create_task() την οποία εξήγησα προηγουμένως, έχει αρχικά priority = 0. Επίσης στη συνάρτηση process_request() προσθέθηκαν αλλά δυο cases ώστε να υποστηρίζονται και οι εντολές h και l. Η τελευταία και η πλέον σημαντική αλλαγή έγινε είναι αυτή που κάναμε στην sigchld_handler() : πλέον όταν μια διεργασία σταματήσει λόγω τερματισμού του κβάντου χρόνου τότε για να ενεργοποιήσουμε την επόμενη κάνουμε τις εξής ενέργειες :

- Αρχικά βάζω την διεργασία που έφαγε SIGSTOP στο τέλος της λίστας.
- Μετά ψάχνω στην λίστα να βρω κάποια διεργασία με priority = 1 .
- Όσο δεν βρίσκω διεργασία με priority = 1 βάζω κάθε διεργασία που δεν μου κάνει στο τέλος της λίστας.
- Αν τελικά βρω μια διεργασία με priority = 1 δεν ψάχνω άλλο, θέτω τον timer και στέλνω SIGCONT στη διεργασία αυτή, αλλά ταυτόχρονα με την κυκλική εναλλαγή κατάφερα να διατηρήσω την σειρά με την οποία πρέπει να εκτελεστούν οι διεργασίες.

Ο κώδικας της άσκησης αυτής φαίνεται παρακάτω :

```
#include <errno.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <signal.h>
#include <string.h>
#include <assert.h>
#include <sys/wait.h>
#include <sys/types.h>
#include "proc-common.h"
#include "request.h"

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

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

/*Definition of my list nodes*/
nodeptr head=NULL;
nodeptr end=NULL;
int i=0,nproc;

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

}
```

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

static int
sched_kill_task_by_id(int id)
{
        nodeptr temp=head;
        while (temp!=NULL){
                if (temp->data == id ){
                        kill(temp->id,SIGKILL);
                        return 0;
                }
                else
                        if(temp->next!=NULL)  temp=temp->next;
                        else break;
        }
        return -ENOSYS;
}


/*Fix the priorities*/

static int
sched_set_high_p(int id){
 /* I will set the priority of the process to 1(high) */
 nodeptr temp=head;
        while (temp!=NULL){
                if (temp->data == id ){
                        temp->priority=1;
   return 0;
                }
                else
                        if(temp->next!=NULL)  temp=temp->next;
   else break;
        }
        return -ENOSYS;
}

static int
sched_set_low_p(int id){
 /* I will set the priority of the process to 0(low) */
  nodeptr temp=head;
        while (temp!=NULL){
```

```
                if (temp->data == id ){
                        temp->priority=0;
                        return 0;
                }
                else
                         if(temp->next!=NULL) temp=temp->next;
                         else break;
        }
        return -ENOSYS;
}



/* Create a new task.  */
static void
sched_create_task(char *executable)//
{
              /* Create a new task and set its priority = 0 , because a new
task has always priority = 0 */
    nodeptr newnode;

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

                pid_t mypid;
                mypid=fork();
                if (mypid < 0){
                        perror("fork");
                }
                if(mypid == 0){
                         raise(SIGSTOP);
                         printf("I am %s, PID = %ld\n",
                         executable, (long)getpid());
                         printf("About to replace myself with the
                                executable %s...\n",executable);
                         sleep(2);
                         execve(executable, newargv, newenviron);
                         /* execve() only returns on error */
                         perror("execve");
                         exit(1);
                         }
                else{
                         /* Insert the new process to the list */
                         newnode=(nodeptr)malloc(sizeof(Node));
```

```
                        if (newnode == NULL){
                                perror("malloc");
                                exit(2);
                        }
                        newnode->id=mypid;
                        newnode->next=NULL;
                        newnode->data=i+1;
                        newnode->priority=0;
                        newnode->name=(char*)malloc(strlen(executable)+1);
                        strcpy(newnode->name,executable);
                        end->next=newnode;
                        end=newnode;
                        nproc++;
                        i++;
                }

}


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

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

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

  case REQ_HIGH_TASK:
   sched_set_high_p(rq->task_arg);
   return 0;

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

  default:
```

```
   return -ENOSYS;
 }
}

/*
 * SIGALRM handler
 */
static void
sigalrm_handler(int signum)
{
        kill(head->id,SIGSTOP);
}

/*
 * SIGCHLD handler
 */
static void
sigchld_handler(int signum)
{
        pid_t p;
        int status;
 int flag = 0;
 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->id == p && current == head){
                                /* I have to delete the head
                                   of my list */
                                     if (current->next == NULL){
                                        /* I come here when
                                        i have one node,delete
                                        it and i am done */
                                              free(current);
                                              printf("The list
                                              is empty…I
                                              don't have any
                                              more work to\n");
                                              exit(0);
                                     }
                                     else{
                                             head=current->next;
                                             free(current);
                                     }
                        }
                        else if (current->id == p && current==end){
                                /* I have to delete the last
                                node of my list */
                                end=previous;
                                end->next=NULL;
                                free(current);
                        }
                        else if(current->id == 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;
}

/* Search if there are high priorities */
nodeptr temp1 = head;
nodeptr current2=head;
```

```
   while (temp1 != NULL){
                                        if (temp1->priority!=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;
                                                temp1=head;
                                                if (temp1 == current2)
                                                          break;
                                        }
                                        else{
                                                 flag = 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 */
 nodeptr current1,temp;

 if (head == NULL) {
  printf("Empty list\n");
  exit(0);
 }

 if (head->next != NULL){
  /* First i have to send the current process to the end of the list */
  end->next=head;
  end=head;
  head=head->next;
  end->next=NULL;
  current1=head;
  temp=head;
```

```
    /* Search at the list if there are high priorities */
    while (temp != NULL){
     if (temp->priority!=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 == current1)
       break;
     }
     else{
       flag = 1 ;
       break;
     }
     }
    alarm(SCHED_TQ_SEC);
   }

                }



  if( head->data == 0 && (flag == 0 || head->priority == 1)){
    printf("I am in Shell again:\n”);
    /* use alarm to stay in shell for some seconds */
    alarm(3*SCHED_TQ_SEC);
  }

  /* Send SIGCONT to the head of the list */
                kill(head->id,SIGCONT);
        }

}


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

 sigemptyset(&sigset);
```

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

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

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


/* Install two signal handlers.
 * One for SIGCHLD, one for SIGALRM.
 * Make sure both signals are masked when one of them is running.
 */


static void
install_signal_handlers(void)
{
 sigset_t sigset;
 struct sigaction sa;

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

```
  exit(1);
 }

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

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

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

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

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

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

/* Create a new shell task.
 *
 * The shell gets special treatment:
 * two pipes are created for communication and passed
```

```
 * as command-line arguments to the executable.
 */
static void
sched_create_shell(char *executable, int *request_fd, int *return_fd)
{
 pid_t p;
 int pfds_rq[2], pfds_ret[2];

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

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

 if (p == 0) {
  /* Child */
  close(pfds_rq[0]);
  close(pfds_ret[1]);
  do_shell(executable, pfds_rq[1], pfds_ret[0]);
  assert(0);
 }
 /* Parent */
 close(pfds_rq[1]);
 close(pfds_ret[0]);
 *request_fd = pfds_rq[0];
 *return_fd = pfds_ret[1];
 //insert shell as head of my list
 head->data=0;
 head->id=p;
 head->priority=0;
 head->name="shell";
 head->next=NULL;
}

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

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

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

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



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

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

        /* 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.
         */

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

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

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

        nodeptr newnode;
        pid_t mypid;

        /*Fork and create the list*/

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

                        execve(executable, newargv, newenviron);

                        /* execve() only returns on error */
                        perror("execve");
                        exit(1);
                        }
```

```
        else{
            /* I dont have to check if my head is NULL
               because Shell is in the head for sure */
                newnode=(nodeptr)malloc(sizeof(Node));
                if (newnode == NULL){
                        perror("malloc");
                        exit(2);
                }
                newnode->id=mypid;
                newnode->next=NULL;
                newnode->data=i+1;
                newnode->priority=0;
                newnode->name=argv[i+1];
                end->next=newnode;
                end=newnode;
        }
 }

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

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


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

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

 shell_request_loop(request_fd, return_fd);

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

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

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

}

Ακολουθεί στιγμιότυπο για κλήση του scheduler με δημιουργία και χρονοδρομολόγηση τεσσάρων προγραμμάτων prog.c(για να τερματίζει μια διεργασία σχετικά γρήγορα ώστε να μπορέσουμε να δούμε το πρόγραμμα να τελειώνει αλλάζουμε την εντολή #define NMSG 200 του αρχείου prog.c με την εντολή #define NMSG 30):

```
oslabd02@ithaki:~/Exercise4/Askhsh1.3$ ./scheduler-shell prog prog prog prog
My PID = 27432: Child PID = 27434 has been stopped by a signal, signo = 19
My PID = 27432: Child PID = 27435 has been stopped by a signal, signo = 19
My PID = 27432: Child PID = 27436 has been stopped by a signal, signo = 19
My PID = 27432: Child PID = 27437 has been stopped by a signal, signo = 19
My PID = 27432: Child PID = 27433 has been stopped by a signal, signo = 19
I am ./scheduler-shell, PID = 27434
About to replace myself with the executable prog...
My PID = 27432: Child PID = 27434 has been stopped by a signal, signo = 19
I am ./scheduler-shell, PID = 27435
About to replace myself with the executable prog...
My PID = 27432: Child PID = 27435 has been stopped by a signal, signo = 19
I am ./scheduler-shell, PID = 27436
About to replace myself with the executable prog...
My PID = 27432: Child PID = 27436 has been stopped by a signal, signo = 19
I am ./scheduler-shell, PID = 27437
About to replace myself with the executable prog...
My PID = 27432: Child PID = 27437 has been stopped by a signal, signo = 19
I am in Shell again:

This is the Shell. Welcome.

Shell> p
Shell: issuing request...
ID: 0 PID: 27433 Name: shell Priority: 0
ID: 1 PID: 27434 Name: prog Priority: 0
ID: 2 PID: 27435 Name: prog Priority: 0
ID: 3 PID: 27436 Name: prog Priority: 0
ID: 4 PID: 27437 Name: prog Priority: 0
Shell: receiving request return value...
Shell> h 1
Shell: issuing request...
Shell: receiving request return value...
Shell> h 2
Shell: issuing request...
Shell: receiving request return value...
Shell> p
Shell: issuing request...
ID: 0 PID: 27433 Name: shell Priority: 0
ID: 1 PID: 27434 Name: prog Priority: 1
ID: 2 PID: 27435 Name: prog Priority: 1
ID: 3 PID: 27436 Name: prog Priority: 0
ID: 4 PID: 27437 Name: prog Priority: 0
Shell: receiving request return value...
Shell> My PID = 27432: Child PID = 27433 has been stopped by a signal, signo = 19
prog: Starting, NMSG = 30, delay = 129
prog[27434]: This is message 0
prog[27434]: This is message 1
prog[27434]: This is message 2
prog[27434]: This is message 3
prog[27434]: This is message 4
prog[27434]: This is message 5
prog[27434]: This is message 6
prog[27434]: This is message 7
My PID = 27432: Child PID = 27434 has been stopped by a signal, signo = 19
prog: Starting, NMSG = 30, delay = 112
prog[27435]: This is message 0
prog[27435]: This is message 1
prog[27435]: This is message 2
prog[27435]: This is message 3
prog[27435]: This is message 4
prog[27435]: This is message 5
prog[27435]: This is message 6
prog[27435]: This is message 7
prog[27435]: This is message 8
My PID = 27432: Child PID = 27435 has been stopped by a signal, signo = 19
prog[27434]: This is message 8
prog[27434]: This is message 9
prog[27434]: This is message 10
prog[27434]: This is message 11
prog[27434]: This is message 12
prog[27434]: This is message 13
prog[27434]: This is message 14
prog[27434]: This is message 15
My PID = 27432: Child PID = 27434 has been stopped by a signal, signo = 19
prog[27435]: This is message 9
prog[27435]: This is message 10
prog[27435]: This is message 11
prog[27435]: This is message 12
prog[27435]: This is message 13
prog[27435]: This is message 14
prog[27435]: This is message 15
prog[27435]: This is message 16
prog[27435]: This is message 17
My PID = 27432: Child PID = 27435 has been stopped by a signal, signo = 19
prog[27434]: This is message 16
prog[27434]: This is message 17
prog[27434]: This is message 18
prog[27434]: This is message 19
prog[27434]: This is message 20
prog[27434]: This is message 21
prog[27434]: This is message 22
My PID = 27432: Child PID = 27434 has been stopped by a signal, signo = 19
prog[27435]: This is message 18
prog[27435]: This is message 19
prog[27435]: This is message 20
prog[27435]: This is message 21
prog[27435]: This is message 22
prog[27435]: This is message 23
prog[27435]: This is message 24
prog[27435]: This is message 25
prog[27435]: This is message 26
My PID = 27432: Child PID = 27435 has been stopped by a signal, signo = 19
prog[27434]: This is message 23
```

```
prog[27434]: This is message 24
prog[27434]: This is message 25
prog[27434]: This is message 26
prog[27434]: This is message 27
prog[27434]: This is message 28
prog[27434]: This is message 29
My PID = 27432: Child PID = 27434 terminated normally, exit status = 0
My PID = 27432: Child PID = 27435 has been stopped by a signal, signo = 19
prog[27435]: This is message 27
prog[27435]: This is message 28
prog[27435]: This is message 29
My PID = 27432: Child PID = 27435 terminated normally, exit status = 0
prog: Starting, NMSG = 30, delay = 93
prog[27436]: This is message 0
prog[27436]: This is message 1
prog[27436]: This is message 2
prog[27436]: This is message 3
prog[27436]: This is message 4
prog[27436]: This is message 5
prog[27436]: This is message 6
My PID = 27432: Child PID = 27436 has been stopped by a signal, signo = 19
prog: Starting, NMSG = 30, delay = 139
prog[27437]: This is message 0
prog[27437]: This is message 1
prog[27437]: This is message 2
prog[27437]: This is message 3
prog[27437]: This is message 4
prog[27437]: This is message 5
prog[27437]: This is message 6
prog[27437]: This is message 7
My PID = 27432: Child PID = 27437 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: 27433 Name: shell Priority: 0
ID: 3 PID: 27436 Name: prog Priority: 0
ID: 4 PID: 27437 Name: prog Priority: 0
Shell> h 0
Shell: issuing request...
Shell: receiving request return value...
Shell> p
Shell: issuing request...
Shell: receiving request return value...
ID: 0 PID: 27433 Name: shell Priority: 1
ID: 3 PID: 27436 Name: prog Priority: 0
ID: 4 PID: 27437 Name: prog Priority: 0
Shell> l 0
Shell: issuing request...
Shell: receiving request return value...
Shell> My PID = 27432: Child PID = 27433 has been stopped by a signal, signo = 19
prog[27436]: This is message 7
prog[27436]: This is message 8
prog[27436]: This is message 9
prog[27436]: This is message 10
prog[27436]: This is message 11
prog[27436]: This is message 12
prog[27436]: This is message 13
prog[27436]: This is message 14
prog[27436]: This is message 15
prog[27436]: This is message 16
My PID = 27432: Child PID = 27436 has been stopped by a signal, signo = 19
prog[27437]: This is message 8
prog[27437]: This is message 9
prog[27437]: This is message 10
prog[27437]: This is message 11
prog[27437]: This is message 12
prog[27437]: This is message 13
prog[27437]: This is message 14
My PID = 27432: Child PID = 27437 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: 27433 Name: shell Priority: 0
ID: 3 PID: 27436 Name: prog Priority: 0
ID: 4 PID: 27437 Name: prog Priority: 0
Shell> k 3
Shell: issuing request...
Shell: receiving request return value...
Shell> My PID = 27432: Child PID = 27436 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 = 27432: Child PID = 27437 was terminated by a signal, signo = 9
I am in Shell again:
q
Shell: Exiting. Goodbye.
My PID = 27432: Child PID = 27433 terminated normally, exit status = 0
The list is empty...I don't have any more work to do
oslabd02@ithaki:~/Exercise4/Askhsh1.3$
```

Ερωτήσεις

1. Περιγράψτε ένα σενάριο δημιουργίας λιμοκτονίας.

Λιμοκτονία θα προκύψει αν μία διεργασία με priority = 0(LOW) δεν αλλάξει ποτέ προτεραιότητα, ενώ πάντα υπάρχουν διεργασίες με priority = 1(HIGH). Στη περίπτωση αυτή ο scheduler δε θα επιλέξει ποτέ τη διεργασία αυτή για να εκτελεστεί καθώς θα εκτελούνται μόνο οι διεργασίες με priority = 1(HIGH) με κυκλική αναφορά. Κάτι τέτοιο δεν είναι επιθυμητό προφανώς γι'αυτό θα μπορούσε να υλοποιηθεί προτεραιότητα με γήρανση, δηλαδή να προσθεθεί στο struct κάθε διεργασίας ένα νεό πεδίο με όνομα old, το οποίο αρχικά είναι μηδέν και κάθε φορά που επιλέγεται μία διεργασία το πεδίο old όλων των άλλων διεργασιών θα αυξάνεται κατά 1.Έτσι όταν το πεδίο old κάποιας διεργασίας ξεπεράσει μια προκαθορισμένη τιμή, τότε ανεξάρτητα απο το αν η διεργασία έχει priority = 1(HIGH) ή priority = 0(LOW) θα εκτελεστεί. Με τον τρόπο αυτό η λιμοκτονία παύει να υπάρχει πια.

Γενική Παρατήρηση

Σε όλες τις ασκήσεις θα πρέπει να τονιστεί ότι και αν μια διεργασία τελείωνε την δουλεία που είχε να κάνει πριν εκπνεύσει το κβάντο χρόνου της ή αν 'έτρωγε' σήμα kill ενώ εκτελούταν και δεν είχε εκπνεύσει το κβάντο χρόνου της δεν έθετα εκ νέου τον timer αλλά ενεργοποιούσα την επόμενη διεργασία που έπρεπε να τρέξει(η οποία προφανώς θα είχε πιο λίγο χρόνο να εκτελεστεί αφού δεν θα είχε στην διάθεση της ολόκληρο το κβάντο χρόνου tq). Τον timer τον έθετα μόνο όταν είχε σταματήσει μια διεργασία λόγω εκπνοής του κβάντου χρόνου. Επέλεξα αυτή τη λογική διότι θεώρησα πως έπρεπε να εκμεταλλευτώ όλο το διαθέσιμο χρόνο.