ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ

ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ

ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ 4Η

Εργαστηριακή ομάδα: oslabb25

Ιωάννης Μιχαήλ Καζελίδης – 03117885

Μάριος Κερασιώτης - 03117890

Περιεχόμενα

Για τις ασκήσεις χρησιμοποιήσαμε το παρακάτω makefile:

makefile

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

CC = gcc
CFLAGS = -Wall -O2 -g

all: scheduler scheduler-shell scheduler-shell-priority shell prog execve-example strace-test sigchld-example

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

queue-shell.o: queue-shell.c
	$(CC) $(CFLAGS) -o queue-shell.o -c queue-shell.c
```

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

Για την άσκηση 1.1 χρησιμοποιούμε τη παρακάτω δομή δεδομένων:

queue.h
```h
#ifndef QUEUE_H
#define QUEUE_H

#include <unistd.h>

typedef struct process_s {
  unsigned id;
  pid_t pid;
  char *name;
  struct process_s *next;
} process;

void *safe_malloc(size_t size);
void enqueue(pid_t pid, char *name);
void dequeue(pid_t pid);
void rotate_queue();
#endif
```

queue.c
```c
#include "queue.h"

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

process* head;
process* tail;
unsigned queue_length;
unsigned queue_max;

void* safe_malloc(size_t size) {
  void* p;

  if ((p = malloc(size)) == NULL) {
    fprintf(stderr, "Out of memory, failed to allocate %zd bytes\n", size);
    exit(1);
  }
  return p;
}

void enqueue(pid_t pid, char* name) {
  queue_length++;
  queue_max++;
  process* new_node = safe_malloc(sizeof(process));
  new_node->pid = pid;
  new_node->id = queue_max;
  new_node->name = name;
  process* temp = head;
  while (temp->next != NULL) temp = temp->next;

  temp->next = new_node;
  tail = new_node;
}

void dequeue(pid_t pid) {
  process* temp = head;
  while (temp->next->pid != pid) temp = temp->next;

  process* to_delete = temp->next;
  free(to_delete);
  temp->next = temp->next->next;
  queue_length--;
  if (queue_length == 0) {
```

```
    printf("Done!\n");
    exit(10);
  }
}

void rotate_queue() {
  head = head->next;
  tail = tail->next;
}
```

Τέλος στο αρχείο prog.c αλλάξαμε τον αριθμό των μηνυμάτων από 200 σε 40 ώστε να έχουμε μικρότερες εξόδους των προγραμμάτων μας και να έχει μικρότερη έκταση η αναφορά μας.

Άσκηση 1.1

Για την άσκηση 1.1 τροποποιήσαμε τον κώδικα του scheduler.c όπως παρακάτω:

scheduler.c

```
#include <assert.h>
#include <errno.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

#include "proc-common.h"
#include "queue.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 */

process *head, *tail;
unsigned queue_length;

/*
 * SIGALRM handler
 */
static void sigalrm_handler(int signum) {
  if (signum != SIGALRM) {
    fprintf(stderr, "Internal error: Called for signum %d, not SIGALRM\n",
            signum);
    exit(1);
  }

  // kill the proccess
  if (kill(head->pid, SIGSTOP) < 0) {
    perror("kill");
    exit(1);
  }
}

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

  if (signum != SIGCHLD) {
    fprintf(stderr, "Internal error: Called for signum %d, not SIGCHLD\n",
            signum);
    exit(1);
  }

  for (;;) {
```

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

    explain_wait_status(p, status);

    if (WIFEXITED(status) || WIFSIGNALED(status)) {
      /* A child has died */
      printf("Parent: Received SIGCHLD, child is dead.\n");

      dequeue(head->pid);

      rotate_queue();

      fprintf(stderr, "Proccess with pid=%ld is about to begin...\n",
              (long int)head->pid);

      if (kill(head->pid, SIGCONT) < 0) {
        perror("Continue to process");
        exit(1);
      }
      /* Setup the alarm again */
      if (alarm(SCHED_TQ_SEC) < 0) {
        perror("alarm");
        exit(1);
      }
    }
    if (WIFSTOPPED(status)) {
      /* A child has stopped due to SIGSTOP/SIGTSTP, etc... */

      // rotate queue
      rotate_queue();

      fprintf(stderr, "Proccess with pid=%ld is about to begin...\n",
              (long int)head->pid);
      if (kill(head->pid, SIGCONT) < 0) {
        perror("Continue to process");
        exit(1);
      }

      /* Setup the alarm again */
      if (alarm(SCHED_TQ_SEC) < 0) {
        perror("alarm");
        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);
  }
}

void child(char *name) {
  char *newargv[] = {name, NULL, NULL, NULL};
  char *newenviron[] = {NULL};

  printf("I am %s, PID = %ld\n", name, (long)getpid());
  printf("About to replace myself with the executable %s...\n", name);
  sleep(2);
  raise(SIGSTOP);
  execve(name, newargv, newenviron);

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

int main(int argc, char *argv[]) {
  int nproc;
  pid_t pid;
  queue_length = 0;
  /*
   * For each of argv[1] to argv[argc - 1],
   * create a new child process, add it to the process list.
   */

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

  head = safe_malloc(sizeof(process));
  head->next = NULL;

  for (int i = 1; i < nproc; i++) {
    printf("Parent: Creating child...\n");
    pid = fork();

    if (pid < 0) {
      perror("fork");
      exit(1);
    } else if (pid == 0) {
      fprintf(stderr, "A new proccess is created with pid=%ld \n",
              (long int)getpid());

      child(argv[i]);
      assert(0);
    } else {
      enqueue(pid, argv[i]);
      printf(
          "Parent: Created child with PID = %ld, waiting for it to "
          "terminate...\n",
          (long)pid);
    }
  }

  // make the queue circular
  head = head->next;
  free(tail->next);
  tail->next = head;
```

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

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

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

  fprintf(stderr, "Proccess with pid=%ld is about to begin...\n",
          (long int)head->pid);

  if (kill(head->pid, SIGCONT) < 0) {
    perror("First child error with continuing");
    exit(1);
  }

  if (alarm(SCHED_TQ_SEC) < 0) {
    perror("alarm");
    exit(1);
  }

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

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

Ενδεικτική έξοδος της εντολής: $ ./scheduler prog prog prog prog:

```
Parent: Creating child...
Parent: Created child with PID = 15254, waiting
for it to terminate...
Parent: Creating child...
A new proccess is created with pid=15254
I am prog, PID = 15254
Parent: Created child with PID = 15255, waiting
for it to terminate...
About to replace myself with the executable
prog...
Parent: Creating child...
Parent: Created child with PID = 15256, waiting
for it to terminate...
Parent: Creating child...
A new proccess is created with pid=15256
I am prog, PID = 15256
About to replace myself with the executable
prog...
Parent: Created child with PID = 15257, waiting
for it to terminate...
A new proccess is created with pid=15257
I am prog, PID = 15257
About to replace myself with the executable
prog...
A new proccess is created with pid=15255
I am prog, PID = 15255
About to replace myself with the executable
prog...
My PID = 15253: Child PID = 15254 has been
stopped by a signal, signo = 19
My PID = 15253: Child PID = 15257 has been
stopped by a signal, signo = 19
My PID = 15253: Child PID = 15256 has been
stopped by a signal, signo = 19
My PID = 15253: Child PID = 15255 has been
stopped by a signal, signo = 19
```

```
Proccess with pid=15254 is about to begin...
prog: Starting, NMSG = 40, delay = 121
prog[15254]: This is message 0
prog[15254]: This is message 1
prog[15254]: This is message 2
prog[15254]: This is message 3
prog[15254]: This is message 4
prog[15254]: This is message 5
prog[15254]: This is message 6
prog[15254]: This is message 7
prog[15254]: This is message 8
My PID = 15253: Child PID = 15254 has been
stopped by a signal, signo = 19
Proccess with pid=15255 is about to begin...
prog: Starting, NMSG = 40, delay = 103
prog[15255]: This is message 0
prog[15255]: This is message 1
prog[15255]: This is message 2
prog[15255]: This is message 3
prog[15255]: This is message 4
prog[15255]: This is message 5
prog[15255]: This is message 6
prog[15255]: This is message 7
prog[15255]: This is message 8
prog[15255]: This is message 9
prog[15255]: This is message 10
My PID = 15253: Child PID = 15255 has been
stopped by a signal, signo = 19
Proccess with pid=15256 is about to begin...
prog: Starting, NMSG = 40, delay = 85
prog[15256]: This is message 0
prog[15256]: This is message 1
prog[15256]: This is message 2
prog[15256]: This is message 3
prog[15256]: This is message 4
prog[15256]: This is message 5
```

```
prog[15256]: This is message 6
prog[15256]: This is message 7
prog[15256]: This is message 8
prog[15256]: This is message 9
prog[15256]: This is message 10
prog[15256]: This is message 11
prog[15256]: This is message 12
My PID = 15253: Child PID = 15256 has been
stopped by a signal, signo = 19
Proccess with pid=15257 is about to begin...
prog: Starting, NMSG = 40, delay = 131
prog[15257]: This is message 0
prog[15257]: This is message 1
prog[15257]: This is message 2
prog[15257]: This is message 3
prog[15257]: This is message 4
prog[15257]: This is message 5
prog[15257]: This is message 6
prog[15257]: This is message 7
prog[15257]: This is message 8
My PID = 15253: Child PID = 15257 has been
stopped by a signal, signo = 19
Proccess with pid=15254 is about to begin...
prog[15254]: This is message 9
prog[15254]: This is message 10
prog[15254]: This is message 11
prog[15254]: This is message 12
prog[15254]: This is message 13
prog[15254]: This is message 14
prog[15254]: This is message 15
prog[15254]: This is message 16
prog[15254]: This is message 17
My PID = 15253: Child PID = 15254 has been
stopped by a signal, signo = 19
Proccess with pid=15255 is about to begin...
prog[15255]: This is message 11
prog[15255]: This is message 12
prog[15255]: This is message 13
prog[15255]: This is message 14
prog[15255]: This is message 15
prog[15255]: This is message 16
prog[15255]: This is message 17
prog[15255]: This is message 18
prog[15255]: This is message 19
prog[15255]: This is message 20
My PID = 15253: Child PID = 15255 has been
stopped by a signal, signo = 19
Proccess with pid=15256 is about to begin...
prog[15256]: This is message 13
prog[15256]: This is message 14
prog[15256]: This is message 15
prog[15256]: This is message 16
prog[15256]: This is message 17
prog[15256]: This is message 18
prog[15256]: This is message 19
prog[15256]: This is message 20
prog[15256]: This is message 21
prog[15256]: This is message 22
prog[15256]: This is message 23
prog[15256]: This is message 24
My PID = 15253: Child PID = 15256 has been
stopped by a signal, signo = 19
Proccess with pid=15257 is about to begin...
prog[15257]: This is message 9
prog[15257]: This is message 10
prog[15257]: This is message 11
prog[15257]: This is message 12
prog[15257]: This is message 13
prog[15257]: This is message 14
prog[15257]: This is message 15
prog[15257]: This is message 16
```

```
My PID = 15253: Child PID = 15257 has been
stopped by a signal, signo = 19
Proccess with pid=15254 is about to begin...
prog[15254]: This is message 18
prog[15254]: This is message 19
prog[15254]: This is message 20
prog[15254]: This is message 21
prog[15254]: This is message 22
prog[15254]: This is message 23
prog[15254]: This is message 24
prog[15254]: This is message 25
prog[15254]: This is message 26
My PID = 15253: Child PID = 15254 has been
stopped by a signal, signo = 19
Proccess with pid=15255 is about to begin...
prog[15255]: This is message 21
prog[15255]: This is message 22
prog[15255]: This is message 23
prog[15255]: This is message 24
prog[15255]: This is message 25
prog[15255]: This is message 26
prog[15255]: This is message 27
prog[15255]: This is message 28
prog[15255]: This is message 29
prog[15255]: This is message 30
My PID = 15253: Child PID = 15255 has been
stopped by a signal, signo = 19
Proccess with pid=15256 is about to begin...
prog[15256]: This is message 25
prog[15256]: This is message 26
prog[15256]: This is message 27
prog[15256]: This is message 28
prog[15256]: This is message 29
prog[15256]: This is message 30
prog[15256]: This is message 31
prog[15256]: This is message 32
prog[15256]: This is message 33
prog[15256]: This is message 34
prog[15256]: This is message 35
My PID = 15253: Child PID = 15256 has been
stopped by a signal, signo = 19
Proccess with pid=15257 is about to begin...
prog[15257]: This is message 17
prog[15257]: This is message 18
prog[15257]: This is message 19
prog[15257]: This is message 20
prog[15257]: This is message 21
prog[15257]: This is message 22
prog[15257]: This is message 23
prog[15257]: This is message 24
My PID = 15253: Child PID = 15257 has been
stopped by a signal, signo = 19
Proccess with pid=15254 is about to begin...
prog[15254]: This is message 27
prog[15254]: This is message 28
prog[15254]: This is message 29
prog[15254]: This is message 30
prog[15254]: This is message 31
prog[15254]: This is message 32
prog[15254]: This is message 33
prog[15254]: This is message 34
prog[15254]: This is message 35
My PID = 15253: Child PID = 15254 has been
stopped by a signal, signo = 19
Proccess with pid=15255 is about to begin...
prog[15255]: This is message 31
prog[15255]: This is message 32
prog[15255]: This is message 33
prog[15255]: This is message 34
prog[15255]: This is message 35
prog[15255]: This is message 36
prog[15255]: This is message 37
```

```
prog[15255]: This is message 38
prog[15255]: This is message 39
My PID = 15253: Child PID = 15255 has been
stopped by a signal, signo = 19
Proccess with pid=15256 is about to begin...
prog[15256]: This is message 36
prog[15256]: This is message 37
prog[15256]: This is message 38
prog[15256]: This is message 39
My PID = 15253: Child PID = 15256 terminated
normally, exit status = 0
Parent: Received SIGCHLD, child is dead.
Proccess with pid=15257 is about to begin...
prog[15257]: This is message 25
prog[15257]: This is message 26
prog[15257]: This is message 27
prog[15257]: This is message 28
prog[15257]: This is message 29
prog[15257]: This is message 30
prog[15257]: This is message 31
My PID = 15253: Child PID = 15257 has been
stopped by a signal, signo = 19
Proccess with pid=15254 is about to begin...
prog[15254]: This is message 36
prog[15254]: This is message 37
```

```
prog[15254]: This is message 38
prog[15254]: This is message 39
My PID = 15253: Child PID = 15254 terminated
normally, exit status = 0
Parent: Received SIGCHLD, child is dead.
Proccess with pid=15255 is about to begin...
My PID = 15253: Child PID = 15255 terminated
normally, exit status = 0
Parent: Received SIGCHLD, child is dead.
Proccess with pid=15257 is about to begin...
prog[15257]: This is message 32
prog[15257]: This is message 33
prog[15257]: This is message 34
prog[15257]: This is message 35
prog[15257]: This is message 36
prog[15257]: This is message 37
prog[15257]: This is message 38
prog[15257]: This is message 39
My PID = 15253: Child PID = 15257 has been
stopped by a signal, signo = 19
Proccess with pid=15257 is about to begin...
My PID = 15253: Child PID = 15257 terminated
normally, exit status = 0
Parent: Received SIGCHLD, child is dead.
Done!
```

(Η έξοδος διαβάζεται σε κάθε σελίδα ξεχωριστά από τα αριστερά προς τα δεξιά.)

Ερώτηση αναφοράς 1.1.1

Στην συνάρτηση install_signal_handlers() που μας δίνεται ουσιαστικά δημιουργούμε μια μάσκα στην δομή sa που είναι μια δομή sigaction. Σε αυτή βάζουμε ένα σύνολο από σήματα sigset (αφού πρώτα έχουμε αρχικοποιήσει το σύνολο ώστε να είναι άδειο), στο οποίο σύνολο έχουμε προσθέσει τα σήματα SIGALRM και SIGCHLD. Αν λοιπόν πρώτα έρθει ένα σήμα SIGCHLD, η sigaction θα μας στείλει στον sigchld_handler και αυτός θα αρχίσει να εκτελεί τις λειτουργίες του. Παράλληλα όμως με την κλήση του sigchld_handler, η μάσκα αντιλαμβάνεται ότι έχει έρθει ένα σήμα που περιέχει στο σύνολό της και επομένως απαγορεύει την πρόσληψη οποιουδήποτε άλλου σήματος εντός του συνόλου της. Με αυτόν τον τρόπο, αν ο sigchld_handler εκτελείται και έρθει SIGALRM, η μάσκα δεν θα επιτρέψει στην sigaction να ενεργοποιήσει τον sigalrm_handler, καθώς βρίσκεται σε λειτουργία άλλος handler σήματος του συνόλου της. Έτσι περιμένει μέχρι να τελειώσει ο sigchld_handler και στην συνέχεια αποδεσμεύεται και επιτρέπει στον sigalrm_handler να κληθεί, δεσμεύοντας πάλι το σήμα SIGALRM του sigset της. Επομένως εμείς στην υλοποίησή μας χρησιμοποιούμε ένα σύνολο και μια μάσκα που κάνει αποκλεισμό των υπόλοιπων σημάτων μέχρι να τελειώσει ο εν ενεργεία sighandler. Δηλαδή χρησιμοποιούμε σήματα. Εν αντιθέσει, ένας πραγματικός χρονοδρομολογητής σε χώρο πυρήνα θα χρησιμοποιούσε κάποια hardware interrupts (διακοπές). Έτσι ο χρονοδρομολογητής θα δέχεται ένα σήμα SIGCHLD από το χώρο χρήστη, θα μεταφέρεται σε χώρο πυρήνα, θα κάνει τις απαραίτητες αλλαγές και ενώ είμαστε σε χώρο πυρήνα, αν έρθει άλλο σήμα SIGALRM, τότε το υλικό δεν θα του επιτρέψει να σταλθεί σε αυτόν το σήμα, αφού το υλικό μας είναι “πιασμένο” από ένα άλλο σήμα. Με αυτόν τον τρόπο, έχουμε καλύτερη και πιο άμεση απόκριση, σε αντίθεση με την περίπτωσή μας που τα σήματα ενδέχεται να έχουν καθυστερήσεις, καθώς ακόμη και τα σήματα χρονοδρομολογούνται. Για αυτό, λοιπόν, κιόλας χρησιμοποιούμε διακοπές αντί για σήματα στους πραγματικούς χρονοδρομολογητές.

Ερώτηση αναφοράς 1.1.2

Κάθε φορά που ο χρονοδρομολογητής λαμβάνει σήμα SIGCHLD, περιμένουμε προφανώς να αναφέρεται στην διεργασία που εκείνη την ώρα βρίσκεται υπό εκτέλεση στο συγκεκριμένο κβάντο χρόνου του χρονοδρομολογητή. Αυτό συμβαίνει διότι η μοναδική διεργασία που υφίσταται αλλαγές, όπως δηλαδή να πάρει SIGSTOP λόγω alarm είτε να τελειώσει έχοντας ολοκληρώσει τις λειτουργίες της, είναι η διεργασία που τρέχει αυτή την στιγμή. Βέβαια, αν λόγω κάποιου εξωτερικού παράγοντα (π.χ. αποστολή SIGKILL) τερματιστεί αναπάντεχα μια οποιαδήποτε διεργασία-παιδί, τότε η waitpid (που λόγω -1, ενημερώνεται για κάθε διεργασία-παιδί) ενημερώνει το status και η WIFSIGNALED δίνει true και τότε αφαιρείται από την λίστα (αφού τερματίστηκε λόγω σήματος) η εν λόγω διεργασία.

Ερώτηση αναφοράς 1.1.3

Ο λόγος που χρησιμοποιούμε 2 διαφορετικά σήματα έγκειται ουσιαστικά στο γεγονός ότι μπορεί να υπάρχουν καθυστερήσεις μεταξύ της αποστολής και λήψης των σημάτων. Αυτό θα το καταλάβουμε καλύτερα με ένα συγκεκριμένο παράδειγμα. Αν χρησιμοποιούσαμε μόνο handler για το SIGALRM θα ήταν πιθανό ένα SIGSTOP να σταλεί σε μια διεργασία και αμέσως μετά ένα SIGCONT σε μια άλλη, ωστόσο η 2η διεργασία να λάβει πρώτη το SIGCONT πριν καν σταματήσει η πρώτη, γεγονός που θα έκανε το χρονοδρομολογητή μας να λειτουργεί λανθασμένα, αφού θα ξεκινούσε έτσι η επόμενη διεργασία πριν σταματήσει η προηγούμενή της, κι έτσι τότε θα έτρεχαν δύο διεργασίες ταυτόχρονα. Όμως τώρα με τους 2 handlers είμαστε σίγουροι ότι ο scheduler μας θα τρέχει σωστά αφού όταν έρθει σήμα SIGARLM στέλνουμε SIGSTOP στη διεργασία που εκτελείται και αναμένουμε να μας έρθει σήμα SIGCHLD (δηλαδή η επιβεβαίωση ότι σταμάτησε) από τη διεργασία και αφού μας έρθει ελέγχουμε τι της συνέβη (δηλαδή αν σταμάτησε επιτυχώς) και μετά από αυτή τη διαδικασία στέλνουμε SIGCONT στην επόμενη που έχει σειρά να ενεργοποιηθεί. Έτσι αποφεύγουμε όλες τις ανεπιθύμητες περιπτώσεις που ενδεχομένως να προκύψουν λόγω των καθυστερήσεων των σημάτων.

Άσκηση 1.2

Για τις ασκήσεις 1.2 και 1.3 χρησιμοποιούμε τη παρακάτω δομή δεδομένων:

queue-shell.h

```h
#ifndef QUEUE_H_
#define QUEUE_H_

#include <sys/types.h>
#include <unistd.h>

#define true 1
#define false 0

/* Define colors for output */
#define GREEN "\033[0;32m"
#define RED "\033[0;31m"
#define BLUE "\033[0;34m"
#define MAGENTA "\033[0;35m"
#define CYAN "\033[0;36m"
#define RST "\033[0m"

typedef int bool;

typedef struct process_s {
  pid_t pid;
  unsigned id;
  char* name;
  struct process_s* next;
} process;

typedef struct queue_s {
  process* head;
  process* tail;
  unsigned size;
} queue;

void* safe_malloc(size_t size);
queue* initialize_queue(void);
process* initialize_process(pid_t pid, char* name, unsigned id);
bool is_empty(queue* q);
unsigned get_size(queue* q);
void enqueue(queue* q, process* new_proc, pid_t pid, char* name);
void dequeue(queue* q, pid_t pid);
void print_queue(queue* q, bool add_space);
process* get_process_by_id(queue* q, unsigned r_id);
void rotate_queue(queue* q);
void rotate_queue_new(queue* q);
#endif  // QUEUE_H_
```

queue-shell.c

```c
#include "queue-shell.h"

#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>

unsigned id;

void *safe_malloc(size_t size) {
  void *p;

  if ((p = malloc(size)) == NULL) {
    fprintf(stderr, "Out of memory, failed to allocate %zd bytes\n", size);
    exit(1);
  }
  return p;
}

queue *initialize_queue(void) {
  queue *q = safe_malloc(sizeof(queue));
  q->head = NULL;
  q->tail = NULL;
  q->size = 0;
  return q;
}

process *initialize_process(pid_t pid, char *name, unsigned r_id) {
  process *p = safe_malloc(sizeof(process));
  p->pid = pid;
  p->id = r_id;
  p->name = name;
  p->next = NULL;
  return p;
}

bool is_empty(queue *q) { return (q->size == 0); }

unsigned get_size(queue *q) { return q->size; }

void enqueue(queue *q, process *new_proc, pid_t pid, char *name) {
  if (q == NULL) {
    printf("Queue not initialized\n");
    exit(1);
  }

  if (new_proc == NULL) {
    new_proc = initialize_process(pid, name, id);
    id++;
  }

  if (is_empty(q)) {
    q->head = new_proc;
    q->head->next = q->head;
    q->tail = q->head;
  } else if (get_size(q) == 1) {
    q->tail = new_proc;
    q->head->next = q->tail;
    q->tail->next = q->head;
  } else {
    q->tail->next = new_proc;
    q->tail = new_proc;
    q->tail->next = q->head;
  }

  q->size++;
}

void dequeue(queue *q, pid_t pid) {
  if (q == NULL) {
    printf("Queue not initialized\n");
    exit(1);
```

```
  }

  if (q->head == NULL) {
    printf("Cannot delete from an empty queue\n");
    return;
  }

  if (get_size(q) == 1) {  // if queue has only 1 process
    if (q->head->pid == pid) {
      free(q->head);
      q->head = NULL;
      q->tail = NULL;
      q->size = 0;
      return;
    } else {
      printf("Process not in queue\n");
      return;
    }
  }

  q->tail->next = NULL;

  process *curr = q->head;  // check if the process to be removed is in the head

  if (curr->pid == pid) {
    q->head = q->head->next;
    q->tail->next = q->head;
    free(curr);
    q->size--;
    return;
  }

  process *prev = NULL;

  while (curr != NULL && curr->pid != pid) {
    prev = curr;
    curr = curr->next;
  }

  if (curr == NULL) {
    printf("Process not in queue\n");
    q->tail->next = q->head;
    return;
  }

  prev->next = curr->next;
  free(curr);
  q->tail->next = q->head;
  q->size--;
}

void print_queue(queue *q, bool print_head_with_space) {
  if (q == NULL) {
    printf("Queue not initialized\n");
    exit(1);
  }

  if (is_empty(q)) {
    printf("Queue is empty\n");
    return;
  }

  process *p = q->head;
  for (int i = 0; i < q->size; i++) {
    if (p == NULL) continue;
    if (p != q->head || print_head_with_space) printf("                  ");
    printf("ID: %d, PID: %ld, NAME: %s\n" RST, p->id, (long)p->pid, p->name);
    p = p->next;
  }
}

process *get_process_by_id(queue *q, unsigned r_id) {
```

```c
  if (q == NULL) {
    printf("Queue not initialized\n");
    exit(1);
  }

  if (is_empty(q)) {
    printf("Queue is empty\n");
    return NULL;
  }

  process *p = q->head;
  for (int i = 0; i < q->size; i++) {
    if (p == NULL) continue;
    if (p->id == r_id) return p;
    p = p->next;
  }
  return NULL;
}

void rotate_queue(queue *q) {
  // if (q->size > 1) {
  //   process *curr = q->head;
  //   q->head = q->head->next;
  //   q->tail->next = curr;
  //   q->tail = curr;
  //   q->tail->next = q->head;
  // }
  if (q == NULL) {
    printf("Queue not initialized\n");
    exit(1);
  }

  if (is_empty(q)) {
    printf("Queue is empty\n");
    return;
  }

  process *temp = initialize_process(q->head->pid, q->head->name, q->head->id);
  dequeue(q, temp->pid);
  enqueue(q, temp, temp->pid, temp->name);
}

void rotate_queue_new(queue *q) {
  if (q->size > 1) {
    process *curr = q->head;
    q->head = q->head->next;
    q->tail->next = curr;
    q->tail = curr;
    q->tail->next = q->head;
  }
}
```

Για την άσκηση 1.2 τροποποιήσαμε τον κώδικα του scheduler-shell.c όπως παρακάτω:

scheduler-shell.c

```c
/*                          - Known Problems -
 * Although the program works fine and all the commands to the shell
 * (e,p,k,q) have the requested output, there rarely occurs a segmentation fault
 * while we are running it. The problem doesn't occur after a specific sequence
 * of commands or at a specific point in the program, it just appears (if it
 * appears) out of nowhere. (We can't recreate the bug although we have
 * pinpointed the seg-fault in sigchld_handler function).
 */

#include <assert.h>
#include <errno.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
```

```c
#include <sys/wait.h>
#include <unistd.h>

#include "proc-common.h"
#include "queue-shell.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 */

/* Define global variables */
unsigned id = 0;
queue *p_queue;

/* Print a list of all tasks currently being scheduled.  */
static void sched_print_tasks(void) {
  printf("-----------------------------------------------------\n");
  if (is_empty(p_queue)) {
    printf("THERE ARE NO PROCESSES TO PRINT\n");
    return;
  }
  printf("Queue has size: %u\n", get_size(p_queue));
  printf(BLUE "Current Process: ");
  print_queue(p_queue, false);
  printf("-----------------------------------------------------\n");
}

/* Send SIGKILL to a task determined by the value of its
 * scheduler-specific id.
 */
static int sched_kill_task_by_id(int id) {
  process *temp = get_process_by_id(p_queue, id);
  if (temp == NULL) {
    printf("Process not in queue\n");
    return -1;
  }

  pid_t r_pid = temp->pid;

  printf(CYAN "ID: %d, PID: %d, NAME: %s is being killed" RST "\n", temp->id,
         temp->pid, temp->name);

  if (kill(r_pid, SIGTERM) < 0) {
    perror("Kill proccess error- sched_kill_task_by_id");
    exit(1);
  }
  dequeue(p_queue, r_pid);
  return 1;
}

/* Create a new task.  */
static void sched_create_task(char *executable) {
  pid_t pid;
  char *new_name;
  new_name = safe_malloc(sizeof(executable));
  strcpy(new_name, executable);

  pid = fork();

  if (pid < 0) {
    perror("forking task- sched_kill_task_by_id");
    exit(1);
  } else if (pid == 0) {
    char *newargv[] = {new_name, NULL};
    char *newenviron[] = {NULL};
    raise(SIGSTOP);
    execve(new_name, newargv, newenviron);
  } else {
    // DEBUG:
    show_pstree(getpid());
```

```
    enqueue(p_queue, NULL, pid, new_name);
  }
}

/* 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 (signum != SIGALRM) {
    fprintf(stderr, "Internal error: Called for signum %d, not SIGALRM\n",
            signum);
    exit(1);
  }

  // kill the proccess
  if (kill(p_queue->head->pid, SIGSTOP) < 0) {
    perror("kill- sigalrm_handler");
    exit(1);
  }
}

/*
 * SIGCHLD handler
 */
static void sigchld_handler(int signum) {
  if (signum != SIGCHLD) {
    fprintf(stderr, "Internal error: Called for signum %d, not SIGCHLD\n",
            signum);
    exit(1);
  }

  pid_t p;
  int status;

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

    explain_wait_status(p, status);

    if (WIFEXITED(status) || WIFSIGNALED(status)) {
      /* A child has died */
      printf("Parent: Received SIGCHLD, child is dead.\n");
      // process *temp = p_queue->head->next;

      dequeue(p_queue, p_queue->head->pid);

      if (is_empty(p_queue)) {
        printf(GREEN "Job's Done!\n" RST);
```

```
        exit(0);
      } else {
        // rotate_queue(p_queue);

        fprintf(stderr, "Proccess with pid=%ld is about to begin...\n",
                (long int)p_queue->head->pid);

        if (kill(p_queue->head->pid, SIGCONT) < 0) {
          perror(
              "Continue to process- WIFEXITED-WIFSIGNALED - sigchld_handler");
          exit(1);
        }

        /* Setup the alarm again */
        if (alarm(SCHED_TQ_SEC) < 0) {
          perror("alarm- sigchld_handler");
          exit(1);
        }
      }
    }

    if (WIFSTOPPED(status)) {
      /* A child has stopped due to SIGSTOP/SIGTSTP, etc... */
      printf("Parent: Child has been stopped. Moving right along...\n");

      // rotate queue
      rotate_queue(p_queue);
      dequeue(p_queue, 0);  // maybe for debugging
      if (is_empty(p_queue)) {
        printf(GREEN "Job's Done!\n" RST);
        exit(0);
      }

      pid_t r_pid = p_queue->head->pid;

      fprintf(stderr, "Proccess with pid=%ld is about to begin...\n",
              (long int)r_pid);
      if (kill(r_pid, SIGCONT) < 0) {
        perror("Continue to process- WIFSTOPPED- sigchld_handler");
        exit(1);
      }

      /* Setup the alarm again */
      if (alarm(SCHED_TQ_SEC) < 0) {
        perror("alarm- WIFSTOPPED- sigchld_handler");
        exit(1);
      }
    }
  }
}

/* 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);
  }

  // initialize queue with the shell process
  char *new_name;
  new_name = safe_malloc(sizeof(executable));
  strcpy(new_name, executable);
  enqueue(p_queue, NULL, p, new_name);

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

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[]) {
  int nproc;
  pid_t pid;
  char *new_name;
  /* Two file descriptors for communication with the shell */
  static int request_fd, return_fd;

  p_queue = initialize_queue();  // to initialize the queue

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

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

```c
  nproc = argc; /* number of proccesses goes here */

  for (int i = 1; i < nproc; i++) {
    new_name = safe_malloc(sizeof(argv[i]));
    strcpy(new_name, argv[i]);

    pid = fork();

    if (pid < 0) {
      perror("forking task- sched_create_shell");
      exit(1);
    } else if (pid == 0) {
      char *newargv[] = {new_name, NULL};
      char *newenviron[] = {NULL};
      raise(SIGSTOP);
      execve(new_name, newargv, newenviron);
    } else {
      enqueue(p_queue, NULL, pid, new_name);
      printf(
          "Parent: Created child with PID = %ld, waiting for it to "
          "terminate...\n",
          (long)pid);
    }
  }

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

  // DEBUG:
  show_pstree(getpid());

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

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

  if (kill(p_queue->head->pid, SIGCONT) < 0) {
    perror("First child error with continuing - main");
    exit(1);
  }

  if (alarm(SCHED_TQ_SEC) < 0) {
    perror("alarm - main");
    exit(1);
  }

  shell_request_loop(request_fd, return_fd);

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

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

Ενδεικτική έξοδος της εντολής: $ ./scheduler-shell prog prog με χρήση των εντολών “p”, “k”, “e”, “q”

```
Parent: Created child with PID = 20969, waiting
for it to terminate...
Parent: Created child with PID = 20970, waiting
for it to terminate...
My PID = 20967: Child PID = 20968 has been
stopped by a signal, signo = 19
```

```
My PID = 20967: Child PID = 20970 has been
stopped by a signal, signo = 19


scheduler-shell(20967)─┬─scheduler-shell(20968)
                       ├─scheduler-shell(20969)
```

```
                        ├─scheduler-shell(20970)

└─sh(20971)───pstree(20972)


My PID = 20967: Child PID = 20969 has been
stopped by a signal, signo = 19
Parent: Child has been stopped. Moving right
along...
Process not in queue
Proccess with pid=20969 is about to begin...

This is the Shell. Welcome.

Shell> prog: Starting, NMSG = 40, delay = 169
prog[20969]: This is message 0
prog[20969]: This is message 1
p
Shell: issuing request...
Shell: receiving request return value...
------------------------------------------------
-
Queue has size: 3
Current Process: ID: 1, PID: 20969, NAME: prog
                 ID: 2, PID: 20970, NAME: prog
                 ID: 0, PID: 20968, NAME: shell
------------------------------------------------
-
Shell> prog[20969]: This is message 2
eprog[20969]: This is message 3
 prprog[20969]: This is message 4
og
Shell: issuing request...
Shell: receiving request return value...


scheduler-shell(20967)─┬─prog(20969)
                       ├─scheduler-shell(20970)
                       ├─scheduler-shell(20974)

├─sh(20975)───pstree(20976)
                       └─shell(20968)


My PID = 20967: Child PID = 20974 has been
stopped by a signal, signo = 19
Parent: Child has been stopped. Moving right
along...
Process not in queue
Proccess with pid=20970 is about to begin...
Shell> prog: Starting, NMSG = 40, delay = 311
prog[20970]: This is message 0
prog[20969]: This is message 5
prog[20969]: This is message 6
prog[20970]: This is message 1
prog[20969]: This is message 7
prog[20969]: This is message 8
prog[20970]: This is message 2
prog[20969]: This is message 9
p
Shell: issuing request...
Shell: receiving request return value...
------------------------------------------------
-
Queue has size: 4
Current Process: ID: 2, PID: 20970, NAME: prog
                 ID: 0, PID: 20968, NAME: shell
                 ID: 3, PID: 20974, NAME: prog
                 ID: 1, PID: 20969, NAME: prog
------------------------------------------------
-
Shell> prog[20969]: This is message 10
prog[20970]: This is message 3
My PID = 20967: Child PID = 20970 has been
stopped by a signal, signo = 19
Parent: Child has been stopped. Moving right
along...
Process not in queue
Proccess with pid=20968 is about to begin...
prog[20969]: This is message 11
prog[20969]: This is message 12
k prog[20969]: This is message 13
3
Shell: issuing request...
Shell: receiving request return value...
ID: 3, PID: 20974, NAME: prog is being killed
Shell> prog[20969]: This is message 14
pprog[20969]: This is message 15

Shell: issuing request...
Shell: receiving request return value...
------------------------------------------------
-
Queue has size: 3
Current Process: ID: 0, PID: 20968, NAME: shell
                 ID: 1, PID: 20969, NAME: prog
                 ID: 2, PID: 20970, NAME: prog
------------------------------------------------
-
Shell> prog[20969]: This is message 16
My PID = 20967: Child PID = 20968 has been
stopped by a signal, signo = 19
Parent: Child has been stopped. Moving right
along...
Process not in queue
Proccess with pid=20969 is about to begin...
prog[20969]: This is message 17
prog[20969]: This is message 18
k 2
prog[20969]: This is message 19
pprog[20969]: This is message 20

prog[20969]: This is message 21
prog[20969]: This is message 22
My PID = 20967: Child PID = 20969 has been
stopped by a signal, signo = 19
Parent: Child has been stopped. Moving right
along...
Process not in queue
Proccess with pid=20970 is about to begin...
prog[20970]: This is message 4
prog[20970]: This is message 5
prog[20970]: This is message 6
My PID = 20967: Child PID = 20970 has been
stopped by a signal, signo = 19
Parent: Child has been stopped. Moving right
along...
Process not in queue
Proccess with pid=20968 is about to begin...
Shell: issuing request...
Shell: receiving request return value...
ID: 2, PID: 20970, NAME: prog is being killed
Shell> Shell: issuing request...
Shell: receiving request return value...
------------------------------------------------
-
Queue has size: 2
Current Process: ID: 0, PID: 20968, NAME: shell
                 ID: 1, PID: 20969, NAME: prog
------------------------------------------------
-
Shell> k 1
Shell: issuing request...
Shell: receiving request return value...
```

```
ID: 1, PID: 20969, NAME: prog is being killed
Shell> p
Shell: issuing request...
Shell: receiving request return value...
-----------------------------------------------
-
Queue has size: 1
Current Process: ID: 0, PID: 20968, NAME: shell
-----------------------------------------------
-
Shell> My PID = 20967: Child PID = 20968 has been
stopped by a signal, signo = 19
```

```
Parent: Child has been stopped. Moving right
along...
Process not in queue
Proccess with pid=20968 is about to begin...
q
Shell: Exiting. Goodbye.
My PID = 20967: Child PID = 20968 terminated
normally, exit status = 0
Parent: Received SIGCHLD, child is dead.
Job's Done!
```

Ερώτηση αναφοράς 1.2.1

Κάθε φορά που εκτελούμε 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 το οποίο είναι απολύτως 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.

Ερώτηση αναφοράς 1.2.2

Η συνάρτηση shell_request_loop() τρέχει σε όλη τη διάρκεια εκτέλεσης του χρονοδρομολογητή και ουσιαστικά μας δίνει τη δυνατότητα να πληκτρολογήσουμε οποιαδήποτε στιγμή στο shell μια εντολή (ακόμη και αν μια άλλη διεργασία τρέχει εκείνη τη στιγμή) και αυτή η εντολή να αποθηκευτεί στον buffer για να δοθεί ως εντολή στο πρόγραμμα shell. Οι συναρτήσεις signal_disable() και signal_enable() (απενεργοποίηση και ενεργοποίηση σημάτων αντίστοιχα) χρησιμοποιούνται στην συγκεκριμένη συνάρτηση, ώστε μόλις δοθεί μια εντολή και περαστεί στον buffer, να γίνει αποκλεισμός των σημάτων SIGCHLD και SIGALRM με την signal_disable() ώστε να περαστεί η εντολή που κάναμε request στο πρόγραμμα shell. Ύστερα προφανώς, ξανακάνουμε signal_enable() για να συνεχιστεί η ροή του προγράμματος. Η χρήση αυτών των συναρτήσεων έχει μεγάλη σημασία, διότι όταν δίνουμε μια εντολή στο shell του λέμε να κάνει κάποια μεταβολή στην ουρά του χρονοδρομολογητή με βάση κάποια εντολή εισόδου. Ωστόσο, αν ο χρονοδρομολογητής παράλληλα δεχτεί μια εντολή για να μετατρέψει μια διεργασία της ουράς, τότε θα έχουμε μια παράλληλη επεξεργασία της λίστας, το οποίο μπορεί να προκαλέσει σοβαρά προβλήματα στο πρόγραμμά μας. Με άλλα λόγια, θέλουμε η επεξεργασία της λίστας να γίνεται κάθε φορά ατομικά.

Άσκηση 1.3

Για την άσκηση 1.3 τροποποιήσαμε τον κώδικα του scheduler-shell.c της άσκησης 1.2 στο αρχείο scheduler-shell-priority.c όπως φαίνεται παρακάτω

scheduler-shell-priority.c

```c
/*                          - Known Problems -
 * We built this program based on the scheduler-shell.c code, so the seg-fault
 * carries through this program. Although the program again works fine and all
 * the commands to the shell (e,p,k,q,h,l) have the requested output, sometimes
 * while we are running it a segmentation fault occurs (it may never happen, it
 * may happen just when we start running the program) or a "suspended signal"
 * error. Both of these problems don't occur after a specific sequence of
 * commands or at a specific point in the program, they just appear (if they
 * appear) out of nowhere. (We can't recreate the bug although we have
 * pinpointed the seg-fault in sigchld_handler function- WIFSTOPPED branch -
 * current_process setting)
 */

#include <assert.h>
#include <errno.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/wait.h>
```

```c
#include <unistd.h>

#include "proc-common.h"
#include "queue-shell.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 */

/* Define global variables */
unsigned id = 0;
queue *lp_queue;
queue *hp_queue;
queue *current_queue;
process current_process;

/* Print a list of all tasks currently being scheduled.  */
static void sched_print_tasks(void) {
  printf("----------------HIGH PRIORITY QUEUE----------------\n");
  if (is_empty(hp_queue)) {
    printf("THERE ARE NO PROCESSES TO PRINT\n");
  } else {
    printf("Queue has size: %u\n", get_size(hp_queue));
    printf(BLUE "Current Process: ");
    print_queue(hp_queue, false);
  }
  // printf("-------------------------------------------------\n");

  printf("\n----------------LOW PRIORITY QUEUE----------------\n");
  if (is_empty(lp_queue)) {
    printf("THERE ARE NO PROCESSES TO PRINT\n");
  } else {
    printf("Queue has size: %u\n", get_size(lp_queue));
    if (is_empty(hp_queue)) {
      printf(BLUE "Current Process: ");
    }
    print_queue(lp_queue, !is_empty(hp_queue));
  }
  printf("-------------------------------------------------\n");
  return;
}

/* Send SIGKILL to a task determined by the value of its
 * scheduler-specific id.
 */
static int sched_kill_task_by_id(int id) {
  bool is_in_lp_queue = true;

  process *temp = get_process_by_id(lp_queue, id);
  if (temp == NULL) {
    is_in_lp_queue = false;
    temp = get_process_by_id(hp_queue, id);
  }

  if (temp == NULL) {
    printf("Process not in queue\n");
    return -1;
  }

  pid_t r_pid = temp->pid;

  printf(CYAN "ID: %d, PID: %d, NAME: %s is being killed" RST "\n", temp->id,
         temp->pid, temp->name);

  if (kill(r_pid, SIGTERM) < 0) {
    perror("Kill proccess error- sched_kill_task_by_id");
    exit(1);
  }

  if (is_in_lp_queue)
```

```
    dequeue(lp_queue, r_pid);
  else
    dequeue(hp_queue, r_pid);

  return 1;
}

/* Create a new task.  */
static void sched_create_task(char *executable) {
  pid_t pid;
  char *new_name;
  new_name = safe_malloc(sizeof(executable));
  strcpy(new_name, executable);

  pid = fork();

  if (pid < 0) {
    perror("forking task- sched_kill_task_by_id");
    exit(1);
  } else if (pid == 0) {
    char *newargv[] = {new_name, NULL};
    char *newenviron[] = {NULL};
    raise(SIGSTOP);
    execve(new_name, newargv, newenviron);
  } else {
    show_pstree(getpid());
    enqueue(lp_queue, NULL, pid, new_name);
  }
}

static void sched_set_high(int id) {
  process *temp = NULL;
  if (!is_empty(lp_queue)) {
    temp = get_process_by_id(lp_queue, id);
  }
  if (temp == NULL) {
    temp = get_process_by_id(hp_queue, id);
    if (temp == NULL) {
      printf(RED "Process not in queues" RST "\n");
    } else {
      printf("Process already has " MAGENTA "HIGH " RST "priority\n");
    }
  } else {
    process *new_node = initialize_process(temp->pid, temp->name, temp->id);
    enqueue(hp_queue, new_node, new_node->pid, new_node->name);
    dequeue(lp_queue, temp->pid);
    printf("Process now has " MAGENTA "HIGH " RST "priority\n");
  }
}

static void sched_set_low(int id) {
  process *temp = NULL;
  if (!is_empty(hp_queue)) {
    temp = get_process_by_id(hp_queue, id);
  }
  if (temp == NULL) {
    temp = get_process_by_id(lp_queue, id);
    if (temp == NULL) {
      printf(RED "Process not in queues" RST "\n");
    } else {
      printf("Process already has " MAGENTA "LOW " RST "priority\n");
    }
  } else {
    process *new_node = initialize_process(temp->pid, temp->name, temp->id);
    enqueue(lp_queue, new_node, new_node->pid, new_node->name);
    dequeue(hp_queue, temp->pid);
    printf("Process now has " MAGENTA "LOW " RST "priority\n");
  }
}

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

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

    default:
      return -ENOSYS;
  }
}

/*
 * SIGALRM handler
 */
static void sigalrm_handler(int signum) {
  // printf(RED "IN SIGALARM" RST "\n");
  if (signum != SIGALRM) {
    fprintf(stderr, "Internal error: Called for signum %d, not SIGALRM\n",
            signum);
    exit(1);
  }

  // kill the proccess
  if (kill(current_process.pid, SIGSTOP) < 0) {
    perror("kill- sigalrm_handler");
    exit(1);
  }
  // printf(RED "OUT SIGALARM" RST "\n");
}

/*
 * SIGCHLD handler
 */
static void sigchld_handler(int signum) {
  // printf(RED "IN sigchld_handler" RST "\n");
  if (signum != SIGCHLD) {
    fprintf(stderr, "Internal error: Called for signum %d, not SIGCHLD\n",
            signum);
    exit(1);
  }

  if (is_empty(hp_queue)) {
    // printf(RED "IN hp empty" RST "\n");
    current_queue = lp_queue;
  } else {
    // printf(RED "IN hp not empty" RST "\n");
    current_queue = hp_queue;
  }

  pid_t p;
  int status;

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

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

    explain_wait_status(p, status);

    if (WIFEXITED(status) || WIFSIGNALED(status)) {
      /* A child has died */
      printf("Parent: Received SIGCHLD, child is dead.\n");
      if (!is_empty(current_queue))
        dequeue(current_queue, current_queue->head->pid);
      // get in if elements have high priority
      if (!is_empty(hp_queue)) {
        // printf(RED "11" RST "\n");
        rotate_queue_new(hp_queue);
        current_process = *hp_queue->head;

        fprintf(stderr, "Proccess with pid=%ld is about to begin...\n",
                (long int)current_process.pid);

        if (kill(current_process.pid, SIGCONT) < 0) {
          perror(
              "Continue to process- WIFEXITED-WIFSIGNALED -sigchld_handler1");
          exit(1);
        }
      } else  // get in if there are no high priority elements
      {
        // printf(RED "12" RST "\n");
        if (is_empty(lp_queue)) {  // both queues are empty
          // printf(RED "13" RST "\n");
          printf(GREEN "Job's Done!\n" RST);
          exit(0);
        } else {
          // printf(RED "14" RST "\n");
          if (current_queue == hp_queue) {  // previous queue is high queue
            current_process = *lp_queue->head;
            if (kill(current_process.pid, SIGCONT) < 0) {
              perror(
                  "Continue to process-WIFEXITED-WIFSIGNALED-sigchld_handler2");
              exit(1);
            }
          } else {  // continue to low queue
            // printf(RED "15" RST "\n");
            rotate_queue_new(lp_queue);
            current_process = *lp_queue->head;
            if (kill(current_process.pid, SIGCONT) < 0) {
              perror(
                  "Continue to process-WIFEXITED-WIFSIGNALED-sigchld_handler3");
              exit(1);
            }
          }
        }
      }
      /* Setup the alarm again */
      if (alarm(SCHED_TQ_SEC) < 0) {
        perror("alarm- sigchld_handler");
        exit(1);
      }
    }
    if (WIFSTOPPED(status)) {
      // printf(RED "IN WIFSTOPPED" RST "\n");
      /* A child has stopped due to SIGSTOP/SIGTSTP, etc... */
      printf("Parent: Child has been stopped. Moving right along...\n");

      if (is_empty(current_queue)) {  // both queues are empty
        // printf(RED "13" RST "\n");
        printf(GREEN "Job's Done!\n" RST);
        exit(0);
      }

      // rotate queue
      rotate_queue_new(current_queue);
      // dequeue(current_queue, 0);
      current_process = *current_queue->head;
```

```
      pid_t r_pid = current_queue->head->pid;

      fprintf(stderr, "Proccess with pid=%ld is about to begin...\n",
              (long int)r_pid);

      if (kill(r_pid, SIGCONT) < 0) {
        perror("Continue to process- WIFSTOPPED- sigchld_handler");
        exit(1);
      }

      /* Setup the alarm again */
      if (alarm(SCHED_TQ_SEC) < 0) {
        perror("alarm- WIFSTOPPED- sigchld_handler");
        exit(1);
      }
    }
  }
}

/* Disable elivery 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);
  }

  // initialize queue with the shell process
  char *new_name;
  new_name = safe_malloc(sizeof(executable));
  strcpy(new_name, executable);
  enqueue(lp_queue, NULL, p, new_name);

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

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

```c
  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[]) {
  int nproc;
  pid_t pid;
  char *new_name;
  /* Two file descriptors for communication with the shell */
  static int request_fd, return_fd;

  lp_queue = initialize_queue();  // to initialize the low priority queue
  hp_queue = initialize_queue();  // to initialize the high priority queue

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

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

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

  for (int i = 1; i < nproc; i++) {
    new_name = safe_malloc(sizeof(argv[i]));
    strcpy(new_name, argv[i]);

    pid = fork();

    if (pid < 0) {
      perror("forking task- sched_create_shell");
      exit(1);
    } else if (pid == 0) {
      char *newargv[] = {new_name, NULL};
      char *newenviron[] = {NULL};
      raise(SIGSTOP);
      execve(new_name, newargv, newenviron);
    } else {
      enqueue(lp_queue, NULL, pid, new_name);
      printf(
          "Parent: Created child with PID = %ld, waiting for it to "
          "terminate...\n",
          (long)pid);
    }
  }

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

  // DEBUG:
  show_pstree(getpid());
```

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

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

  if (kill(lp_queue->head->pid, SIGCONT) < 0) {
    perror("First child error with continuing - main");
    exit(1);
  }

  if (alarm(SCHED_TQ_SEC) < 0) {
    perror("alarm - main");
    exit(1);
  }

  shell_request_loop(request_fd, return_fd);

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

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

Ενδεικτική έξοδος της εντολής: $ ./scheduler-shell-priority prog prog με χρήση των εντολών “p”, “k”, “e”, “q”, “h”, “l”:

```
Parent: Created child with PID = 22124, waiting
for it to terminate...
Parent: Created child with PID = 22125, waiting
for it to terminate...
My PID = 22122: Child PID = 22123 has been
stopped by a signal, signo = 19
My PID = 22122: Child PID = 22124 has been
stopped by a signal, signo = 19


scheduler-shell(22122)─┬─scheduler-shell(22123)
                       ├─scheduler-shell(22124)
                       ├─scheduler-shell(22125)

└─sh(22126)───pstree(22127)


My PID = 22122: Child PID = 22125 has been
stopped by a signal, signo = 19
Parent: Child has been stopped. Moving right
along...
Proccess with pid=22124 is about to begin...

This is the Shell. Welcome.

prog: Starting, NMSG = 40, delay = 340
prog[22124]: This is message 0
Shell> p
Shell: issuing request...
Shell: receiving request return value...
----------------HIGH PRIORITY QUEUE--------------
---
THERE ARE NO PROCESSES TO PRINT

----------------LOW PRIORITY QUEUE---------------
-
```

```
Queue has size: 3
Current Process: ID: 1, PID: 22124, NAME: prog
                 ID: 2, PID: 22125, NAME: prog
                 ID: 0, PID: 22123, NAME: shell
-------------------------------------------------
Shell> prog[22124]: This is message 1
e proprog[22124]: This is message 2
g
Shell: issuing request...
Shell: receiving request return value...


scheduler-shell(22122)─┬─prog(22124)
                       ├─scheduler-shell(22125)
                       ├─scheduler-shell(22129)

├─sh(22130)───pstree(22131)
                       └─shell(22123)


My PID = 22122: Child PID = 22129 has been
stopped by a signal, signo = 19
Parent: Child has been stopped. Moving right
along...
Proccess with pid=22125 is about to begin...
prog: Starting, NMSG = 40, delay = 81
prog[22125]: This is message 0
Shell> prog[22125]: This is message 1
prog[22125]: This is message 2
prog[22125]: This is message 3
prog[22124]: This is message 3
prog[22125]: This is message 4
hprog[22125]: This is message 5
 prog[22125]: This is message 6
3
Shell: issuing request...
```

```
Shell: receiving request return value...
Process now has HIGH priority
Shell> prog[22125]: This is message 7
prog[22124]: This is message 4
prog[22125]: This is message 8
prog[22125]: This is message 9
prog[22125]: This is message 10
prog[22125]: This is message 11
prog[22125]: This is message 12
prog[22124]: This is message 5
My PID = 22122: Child PID = 22125 has been
stopped by a signal, signo = 19
Parent: Child has been stopped. Moving right
along...
Proccess with pid=22129 is about to begin...
prog: Starting, NMSG = 40, delay = 256
prog[22129]: This is message 0
prog[22129]: This is message 1
prog[22124]: This is message 6
prog[22129]: This is message 2
prog[22124]: This is message 7
prog[22129]: This is message 3
My PID = 22122: Child PID = 22129 has been
stopped by a signal, signo = 19
Parent: Child has been stopped. Moving right
along...
Proccess with pid=22129 is about to begin...
prog[22124]: This is message 8
prog[22129]: This is message 4
prog[22129]: This is message 5
prog[22124]: This is message 9
prog[22129]: This is message 6
prog[22124]: This is message 10
prog[22129]: This is message 7
My PID = 22122: Child PID = 22129 has been
stopped by a signal, signo = 19
Parent: Child has been stopped. Moving right
along...
Proccess with pid=22129 is about to begin...
prog[22124]: This is message 11
prog[22129]: This is message 8
prog[22124]: This is message 12
prog[22129]: This is message 9
prog[22129]: This is message 10
prog[22124]: This is message 13
prog[22129]: This is message 11
My PID = 22122: Child PID = 22129 has been
stopped by a signal, signo = 19
Parent: Child has been stopped. Moving right
along...
Proccess with pid=22129 is about to begin...
pprog[22124]: This is message 14

Shell: issuing request...
Shell: receiving request return value...
----------------HIGH PRIORITY QUEUE--------------
---
Queue has size: 1
Current Process: ID: 3, PID: 22129, NAME: prog

----------------LOW PRIORITY QUEUE---------------
-
Queue has size: 3
                 ID: 2, PID: 22125, NAME: prog
                 ID: 0, PID: 22123, NAME: shell
                 ID: 1, PID: 22124, NAME: prog
------------------------------------------------
Shell> prog[22129]: This is message 12
prog[22124]: This is message 15
prog[22129]: This is message 13
prog[22124]: This is message 16
prog[22129]: This is message 14
```

```
h My PID = 22122: Child PID = 22129 has been
stopped by a signal, signo = 19
Parent: Child has been stopped. Moving right
along...
Proccess with pid=22129 is about to begin...
2
Shell: issuing request...
Shell: receiving request return value...
Process now has HIGH priority
Shell> prog[22124]: This is message 17
prog[22129]: This is message 15
prog[22129]: This is message 16
prog[22124]: This is message 18
p
Shell: issuing request...
Shell: receiving request return value...
----------------HIGH PRIORITY QUEUE--------------
---
Queue has size: 2
Current Process: ID: 3, PID: 22129, NAME: prog
                 ID: 2, PID: 22125, NAME: prog

----------------LOW PRIORITY QUEUE---------------
-
Queue has size: 2
                 ID: 0, PID: 22123, NAME: shell
                 ID: 1, PID: 22124, NAME: prog
------------------------------------------------
Shell> prog[22129]: This is message 17
prog[22124]: This is message 19
prog[22129]: This is message 18
My PID = 22122: Child PID = 22129 has been
stopped by a signal, signo = 19
Parent: Child has been stopped. Moving right
along...
Proccess with pid=22125 is about to begin...
prog[22125]: This is message 13
prog[22124]: This is message 20
prog[22125]: This is message 14
1 prog[22125]: This is message 15
3prog[22125]: This is message 16

Shell: issuing request...
Shell: receiving request return value...
Process now has LOW priority
Shell> prog[22125]: This is message 17
prog[22124]: This is message 21
prog[22125]: This is message 18
prog[22125]: This is message 19
prog[22125]: This is message 20
prog[22125]: This is message 21
prog[22125]: This is message 22
pprog[22124]: This is message 22
prog[22125]: This is message 23

Shell: issuing request...
Shell: receiving request return value...
----------------HIGH PRIORITY QUEUE--------------
---
Queue has size: 1
Current Process: ID: 2, PID: 22125, NAME: prog

----------------LOW PRIORITY QUEUE---------------
-
Queue has size: 3
                 ID: 0, PID: 22123, NAME: shell
                 ID: 1, PID: 22124, NAME: prog
                 ID: 3, PID: 22129, NAME: prog
------------------------------------------------
Shell> prog[22125]: This is message 24
My PID = 22122: Child PID = 22125 has been
stopped by a signal, signo = 19
```

```
Parent: Child has been stopped. Moving right
along...
Proccess with pid=22125 is about to begin...
prog[22125]: This is message 25
prog[22125]: This is message 26
prog[22124]: This is message 23
prog[22125]: This is message 27
prog[22125]: This is message 28
prog[22125]: This is message 29
prog[22124]: This is message 24
prog[22125]: This is message 30
prog[22125]: This is message 31
l prog[22125]: This is message 32
2prog[22125]: This is message 33

Shell: issuing request...
Shell: receiving request return value...
Process now has LOW priority
Shell> prog[22124]: This is message 25
prog[22125]: This is message 34
prog[22125]: This is message 35
My PID = 22122: Child PID = 22125 has been
stopped by a signal, signo = 19
Parent: Child has been stopped. Moving right
along...
Proccess with pid=22124 is about to begin...
pprog[22124]: This is message 26

Shell: issuing request...
Shell: receiving request return value...
----------------HIGH PRIORITY QUEUE--------------
---
THERE ARE NO PROCESSES TO PRINT

----------------LOW PRIORITY QUEUE---------------
-
Queue has size: 4
Current Process: ID: 1, PID: 22124, NAME: prog
                 ID: 3, PID: 22129, NAME: prog
                 ID: 2, PID: 22125, NAME: prog
                 ID: 0, PID: 22123, NAME: shell
------------------------------------------------
Shell> prog[22124]: This is message 27
prog[22124]: This is message 28
My PID = 22122: Child PID = 22124 has been
stopped by a signal, signo = 19
Parent: Child has been stopped. Moving right
along...
Proccess with pid=22129 is about to begin...
kprog[22129]: This is message 19
 2
Shell: issuing request...
Shell: receiving request return value...
ID: 2, PID: 22125, NAME: prog is being killed
Shell> prog[22129]: This is message 20
prog[22129]: This is message 21
p
Shell: issuing request...
Shell: receiving request return value...
----------------HIGH PRIORITY QUEUE--------------
---
THERE ARE NO PROCESSES TO PRINT

----------------LOW PRIORITY QUEUE---------------
-
Queue has size: 3
Current Process: ID: 3, PID: 22129, NAME: prog
                 ID: 0, PID: 22123, NAME: shell
                 ID: 1, PID: 22124, NAME: prog
------------------------------------------------
Shell> prog[22129]: This is message 22
```

```
My PID = 22122: Child PID = 22129 has been
stopped by a signal, signo = 19
Parent: Child has been stopped. Moving right
along...
Proccess with pid=22123 is about to begin...
k 1
Shell: issuing request...
Shell: receiving request return value...
ID: 1, PID: 22124, NAME: prog is being killed
Shell> My PID = 22122: Child PID = 22123 has been
stopped by a signal, signo = 19
Parent: Child has been stopped. Moving right
along...
Proccess with pid=22129 is about to begin...
prog[22129]: This is message 23
p
prog[22129]: This is message 24
prog[22129]: This is message 25
My PID = 22122: Child PID = 22129 has been
stopped by a signal, signo = 19
Parent: Child has been stopped. Moving right
along...
Proccess with pid=22123 is about to begin...
Shell: issuing request...
Shell: receiving request return value...
----------------HIGH PRIORITY QUEUE--------------
---
THERE ARE NO PROCESSES TO PRINT

----------------LOW PRIORITY QUEUE---------------
-
Queue has size: 2
Current Process: ID: 0, PID: 22123, NAME: shell
                 ID: 3, PID: 22129, NAME: prog
------------------------------------------------
Shell> My PID = 22122: Child PID = 22123 has been
stopped by a signal, signo = 19
Parent: Child has been stopped. Moving right
along...
Proccess with pid=22129 is about to begin...
k prog[22129]: This is message 26
3
prog[22129]: This is message 27
prog[22129]: This is message 28
prog[22129]: This is message 29
My PID = 22122: Child PID = 22129 has been
stopped by a signal, signo = 19
Parent: Child has been stopped. Moving right
along...
Proccess with pid=22123 is about to begin...
Shell: issuing request...
Shell: receiving request return value...
ID: 3, PID: 22129, NAME: prog is being killed
Shell> p
Shell: issuing request...
Shell: receiving request return value...
----------------HIGH PRIORITY QUEUE--------------
---
THERE ARE NO PROCESSES TO PRINT

----------------LOW PRIORITY QUEUE---------------
-
Queue has size: 1
Current Process: ID: 0, PID: 22123, NAME: shell
------------------------------------------------
Shell> q
Shell: Exiting. Goodbye.
My PID = 22122: Child PID = 22123 terminated
normally, exit status = 0
Parent: Received SIGCHLD, child is dead.
Job's Done!
```

Ερώτηση αναφοράς 1.3.1

Ουσιαστικά ένα πολύ μεγάλο ζήτημα λιμοκτονίας είναι όταν έχουμε HIGH προτεραιότητα σε διεργασίες που για να ολοκληρωθούν χρειάζονται πολλά κβάντα χρόνου, ενώ παράλληλα έχουμε σε priority LOW το shell. Στην περίπτωση αυτή, έχουμε αποκλείσει την δυνατότητα της δυναμικής επέμβασής μας στο χειρισμό των διεργασιών και το μόνο που μπορούμε να κάνουμε είναι να περιμένουμε τις διεργασίες υψηλής προτεραιότητας να ολοκληρωθούν, ώστε να δώσουμε την σκυτάλη στις χαμηλής προτεραιότητας διεργασίες και προφανώς και στο shell. Επίσης πρόβλημα δημιουργείται όταν για παράδειγμα έχουμε κάποιες διεργασίες χαμηλής προτεραιότητας και παράλληλα δημιουργούμε νέες διεργασίες μέσω του shell τις οποίες τις βάζουμε συνέχεια σε high priority. Τότε δημιουργείται ένας μεγάλος κύκλος εκτέλεσης διεργασιών υψηλής προτεραιότητας, με αποτέλεσμα οι διεργασίες χαμηλής προτεραιότητας να λιμοκτονούν μέχρι να λάβουν την σκυτάλη. Ένας πιθανός τρόπος αντιμετώπισης σε αυτό το πρόβλημα θα ήταν να υλοποιηθεί προτεραιότητα με γήρανση, δηλαδή να προστεθεί στο struct κάθε διεργασίας ένα νέο πεδίο που θα αναφέρεται στην «ηλικία» της διεργασίας, το οποίο αρχικά είναι μηδέν και κάθε φορά που επιλέγεται μία διεργασία το πεδίο αυτό όλων των άλλων διεργασιών θα αυξάνεται κατά 1 . Έτσι όταν το πεδίο της «ηλικίας» κάποιας διεργασίας ξεπεράσει μια προκαθορισμένη τιμή, τότε ανεξάρτητα από το αν η διεργασία έχει HIGH ή LOW priority θα εκτελεστεί. Με τον τρόπο αυτό, λοιπόν, η λιμοκτονία παύει να υπάρχει πια.