ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ

ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΛΕΙΤΟΥΡΓΙΚΑ ΣΥΣΤΗΜΑΤΑ

ΕΡΓΑΣΤΗΡΙΑΚΗ ΑΣΚΗΣΗ 4^{H}

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 queue.o

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

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

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

Για τις ασκήσεις 1.1 και 1.2 χρησιμοποιούμε τη παρακάτω δομή δεδομένων:

queue.h

```c
#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

```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;
```

```c
  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;
  /* 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 την άσκηση 1.2 τροποποιήσαμε τον κώδικα του scheduler-shell.c όπως παρακάτω:

scheduler-shell.c
```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 */
#define SHELL_EXECUTABLE_NAME "shell" /* executable for shell */

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

/* Print a list of all tasks currently being scheduled.  */
static void sched_print_tasks(void) {
  process *temp = head;
  printf("Current Process: ");
  for (int i = 0; i < queue_length + 1; i++) {
    if (temp != head) printf("                 ");
    printf("ID: %d, PID: %d, NAME: %s\n", temp->id, temp->pid, temp->name);
    temp = temp->next;
  }
}

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

  while (temp->id != id) {
    temp = temp->next;
    if (temp == head) {
      printf("ID not in process queue\n");
      return -1;
    }
  }
  printf("ID: %d, PID: %d, NAME: %s is being killed\n", temp->id, temp->pid,
         temp->name);

  if (kill(temp->pid, SIGKILL) < 0) {
    perror("kill");
  }

  head = temp;
  return 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);
}

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

  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(executable);
    assert(0);
  } else {
    show_pstree(getpid());
    tail->next = NULL;  // make queue linear for enqueue
    char *t_name;
    t_name = safe_malloc(sizeof(executable));
    strcpy(t_name, executable);
    enqueue(pid, t_name);
    tail->next = head;  // make queue circular again
    printf(
        "Parent: Created child with PID = %ld, waiting for it to "
        "terminate...\n",
        (long)pid);
  }
}

/* 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;
  }
}
```

```c
/*
 * 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);
      }
    }
  }
}

/* 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);
  }
```

```c
}

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
  head = (process *)malloc(sizeof(process));
  head->id = 0;
  head->pid = p;
  head->name = SHELL_EXECUTABLE_NAME;
  tail = head;
  tail->next = NULL;

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

  /* 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.
   */
  queue_length = 0;
  queue_max = 0;
  nproc = argc; /* number of proccesses goes here */

  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

  free(tail->next);
  tail->next = head;

  /* Wait for all children to raise SIGSTOP before exec()ing. */
  wait_for_ready_children(nproc);
  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(head->pid, SIGCONT) < 0) {
    perror("First child error with continuing");
    exit(1);
  }

  if (alarm(SCHED_TQ_SEC) < 0) {
    perror("alarm");
    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: Creating child...
Parent: Created child with PID = 16444, waiting for it to
terminate...
Parent: Creating child...
Parent: Created child with PID = 16445, waiting for it to
terminate...
A new proccess is created with pid=16444
My PID = 16442: Child PID = 16443 has been stopped by a
signal, signo = 19
I am prog, PID = 16444
About to replace myself with the executable prog...
A new proccess is created with pid=16445
I am prog, PID = 16445
About to replace myself with the executable prog...
My PID = 16442: Child PID = 16444 has been stopped by a
signal, signo = 19
My PID = 16442: Child PID = 16445 has been stopped by a
signal, signo = 19



scheduler-shell(16442)─┬─scheduler-shell(16443)
                       ├─scheduler-shell(16444)
                       ├─scheduler-shell(16445)
                       └─sh(16451)───pstree(16452)



This is the Shell. Welcome.

Shell> p
Shell: issuing request...
Shell: receiving request return value...
Current Process: ID: 0, PID: 16443, NAME: shell
         ID: 1, PID: 16444, NAME: prog
         ID: 2, PID: 16445, NAME: prog
Shell> e prog
Shell: issuing request...
Shell: receiving request return value...
A new proccess is created with pid=16454
I am prog, PID = 16454
About to replace myself with the executable prog...
```

```

scheduler-shell(16442)─┬─scheduler-shell(16444)
                       ├─scheduler-shell(16445)
                       ├─scheduler-shell(16454)
                       ├─sh(16455)───pstree(16456)
                       └─shell(16443)


Parent: Created child with PID = 16454, waiting for it to
terminate...
Shell> My PID = 16442: Child PID = 16443 has been stopped
by a signal, signo = 19
Proccess with pid=16444 is about to begin...
prog: Starting, NMSG = 40, delay = 118
prog[16444]: This is message 0
prog[16444]: This is message 1
prog[16444]: This is message 2
prog[16444]: This is message 3
prog[16444]: This is message 4
prog[16444]: This is message 5
pprog[16444]: This is message 6
prog[16444]: This is message 7

prog[16444]: This is message 8
My PID = 16442: Child PID = 16454 has been stopped by a
signal, signo = 19
Proccess with pid=16445 is about to begin...
prog: Starting, NMSG = 40, delay = 34
prog[16445]: This is message 0
prog[16445]: This is message 1
prog[16445]: This is message 2
prog[16444]: This is message 9
prog[16445]: This is message 3
prog[16445]: This is message 4
prog[16445]: This is message 5
prog[16444]: This is message 10
prog[16445]: This is message 6
prog[16445]: This is message 7
prog[16445]: This is message 8
prog[16445]: This is message 9
prog[16444]: This is message 11
prog[16445]: This is message 10
prog[16445]: This is message 11
```

```
prog[16445]: This is message 12
prog[16444]: This is message 12
prog[16445]: This is message 13
prog[16445]: This is message 14
prog[16445]: This is message 15
prog[16445]: This is message 16
prog[16444]: This is message 13
prog[16445]: This is message 17
prog[16445]: This is message 18
prog[16445]: This is message 19
prog[16444]: This is message 14
prog[16445]: This is message 20
prog[16445]: This is message 21
prog[16445]: This is message 22
prog[16445]: This is message 23
prog[16444]: This is message 15
prog[16445]: This is message 24
prog[16445]: This is message 25
prog[16445]: This is message 26
prog[16444]: This is message 16
prog[16445]: This is message 27
prog[16445]: This is message 28
prog[16445]: This is message 29
My PID = 16442: Child PID = 16445 has been stopped by a
signal, signo = 19
Proccess with pid=16454 is about to begin...
prog: Starting, NMSG = 40, delay = 129
prog[16454]: This is message 0
prog[16444]: This is message 17
prog[16454]: This is message 1
prog[16444]: This is message 18
prog[16454]: This is message 2
prog[16444]: This is message 19
prog[16444]: This is message 20
prog[16454]: This is message 3
prog[16444]: This is message 21
prog[16454]: This is message 4
prog[16444]: This is message 22
prog[16454]: This is message 5
prog[16444]: This is message 23
prog[16454]: This is message 6
prog[16444]: This is message 24
prog[16454]: This is message 7
prog[16444]: This is message 25
My PID = 16442: Child PID = 16454 has been stopped by a
signal, signo = 19
Proccess with pid=16443 is about to begin...
Shell: issuing request...
Shell: receiving request return value...
```

```
Current Process: ID: 0, PID: 16443, NAME: shell
            ID: 1, PID: 16444, NAME: prog
            ID: 2, PID: 16445, NAME: prog
            ID: 3, PID: 16454, NAME: prog
Shell> prog[16444]: This is message 26
prog[16444]: This is message 27
k
prog[16444]: This is message 28
 3prog[16444]: This is message 29

Shell: issuing request...
Shell: receiving request return value...
ID: 3, PID: 16454, NAME: prog is being killed
Shell> My PID = 16442: Child PID = 16454 was terminated by
a signal, signo = 9
Parent: Received SIGCHLD, child is dead.
Proccess with pid=16443 is about to begin...
prog[16444]: This is message 30
prog[16444]: This is message 31
prog[16444]: This is message 32
prog[16444]: This is message 33
prog[16444]: This is message 34
prog[16444]: This is message 35
prog[16444]: This is message 36
prog[16444]: This is message 37
prog[16444]: This is message 38
My PID = 16442: Child PID = 16443 has been stopped by a
signal, signo = 19
Proccess with pid=16444 is about to begin...
prog[16444]: This is message 39
My PID = 16442: Child PID = 16444 terminated normally, exit
status = 0
Parent: Received SIGCHLD, child is dead.
Proccess with pid=16445 is about to begin...
prog[16445]: This is message 30
prog[16445]: This is message 31
prog[16445]: This is message 32
prog[16445]: This is message 33
prog[16445]: This is message 34
prog[16445]: This is message 35
prog[16445]: This is message 36
prog[16445]: This is message 37
prog[16445]: This is message 38
prog[16445]: This is message 39
My PID = 16442: Child PID = 16445 terminated normally, exit
status = 0
Parent: Received SIGCHLD, child is dead.
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 και χρησιμοποιήσαμε τα αρχεία queue-priority.c και queue-priority.h όπως φαίνονται παρακάτω

queue-priority.h
```c
#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;

typedef struct list_s {
  process *head;
  process *tail;
  unsigned queue_length;
} list;

void *safe_malloc(size_t size);
void enqueue(list *queue, pid_t pid, char *name, int has_id);
void dequeue(list *queue, pid_t pid);
void rotate_queue(list *queue);
process *search_by_id(list *queue, int id);
void print_queue(list *queue);
#endif
```

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

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

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(list *queue, pid_t pid, char *name, int has_id) {
  process *new_node = safe_malloc(sizeof(process));
  queue->queue_length++;
  new_node->pid = pid;
  if (has_id == -1) {
    queue_max++;
    new_node->id = queue_max;
  } else
    new_node->id = has_id;
  new_node->name = name;

  if (queue->queue_length == 1) {
    queue->head = new_node;
    queue->tail = queue->head;
    queue->tail->next = queue->head;
  } else {
    queue->tail->next = new_node;
    queue->tail = new_node;
    queue->tail->next = queue->head;
  }
}

void dequeue(list *queue, pid_t pid) {
  if (queue->queue_length == 1) {
    process *temp = queue->head;
    queue->head = NULL;
    queue->tail = NULL;
    free(temp);
    queue->queue_length--;
    return;
  }

  process *temp = queue->head;
  while (temp->next->pid != pid) temp = temp->next;

  process *to_delete = temp->next;
  temp->next = temp->next->next;
  free(to_delete);

  queue->queue_length--;
}

process *search_by_id(list *queue, int id) {
  if (queue->queue_length == 0) return NULL;
  process *temp = queue->head;
  for (int i = 0; i < queue->queue_length; i++) {
    if (temp == NULL) return NULL;
    if (temp->id == id) {
      return temp;
    }
    temp = temp->next;
  }
  return NULL;
}

void rotate_queue(list *queue) {
  if (queue->queue_length <= 1) return;
  queue->head = queue->head->next;
  queue->tail = queue->tail->next;
}
```

scheduler-shell-priority.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-priority.h"
#include "request.h"
```

```c
/* Compile-time parameters. */
#define SCHED_TQ_SEC 5                /* time quantum */
#define TASK_NAME_SZ 60               /* maximum size for a task's name */
#define SHELL_EXECUTABLE_NAME "shell" /* executable for shell */

list *hp_queue, *lp_queue;
unsigned queue_max;

/* Print a list of all tasks currently being scheduled.  */
static void sched_print_tasks(void) {
  process *temp = hp_queue->head;
  if (hp_queue->queue_length > 0) {
    printf("                 ");
    printf("HIGH priority list with size = %d\n", hp_queue->queue_length);
    printf("Current Process: ");
  }
  for (int i = 0; i < hp_queue->queue_length; i++) {
    if (temp != hp_queue->head) printf("                 ");
    printf("ID: %d, PID: %d, NAME: %s\n", temp->id, temp->pid, temp->name);
    temp = temp->next;
  }

  temp = lp_queue->head;
  printf("                 ");
  printf("LOW priority list with size = %d\n", lp_queue->queue_length);
  if (hp_queue->queue_length == 0) printf("Current Process: ");
  for (int i = 0; i < lp_queue->queue_length; i++) {
    if (temp != lp_queue->head) printf("                 ");
    printf("ID: %d, PID: %d, NAME: %s\n", temp->id, temp->pid, temp->name);
    temp = temp->next;
  }
}

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

static int sched_kill_task_by_id(int id) {
  int priority_queue = 1;

  process *temp = search_by_id(hp_queue, id);
  if (temp == NULL) {
    temp = search_by_id(lp_queue, id);
    priority_queue = 0;
  }

  if (temp != NULL) {
    printf("ID: %d, PID: %d, NAME: %s is being killed\n", temp->id, temp->pid,
           temp->name);

    // if (strcmp(lp_queue->head->name, SHELL_EXECUTABLE_NAME) == 0) return -1;
    if (kill(temp->pid, SIGKILL) < 0) {
      perror("kill");
    } else {
      if (priority_queue)
        dequeue(hp_queue, temp->pid);
      else
        dequeue(lp_queue, temp->pid);
    }
    return 1;
  }

  printf("ID not in process queues\n");
  return -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);
}

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

  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(executable);
    assert(0);
  } else {
    show_pstree(getpid());
    char *t_name;
    t_name = safe_malloc(sizeof(executable));
    strcpy(t_name, executable);
    enqueue(lp_queue, pid, t_name, -1);
    printf(
        "Parent: Created child with PID = %ld, waiting for it to "
        "terminate...\n",
        (long)pid);
  }
}

static int sched_set_high_p(int id) {
  process *temp;
  temp = search_by_id(lp_queue, id);
  fprintf(stderr, "->>>>>>>>>>>>>>>>>>>> pid=%ld\n", (long int)temp->pid);

  if (temp != NULL) {
    // if (hp_queue->queue_length == 0) {
    //   process *tt = search_by_name(lp_queue, SHELL_EXECUTABLE_NAME);
    //   enqueue(hp_queue, tt->pid, SHELL_EXECUTABLE_NAME, tt->id);
    //   dequeue(lp_queue, tt->pid);
    // }
    enqueue(hp_queue, temp->pid, temp->name, id);
    printf("ID: %d, PID: %d, NAME: %s has HIGH priority now\n", temp->id,
           temp->pid, temp->name);
    dequeue(lp_queue, temp->pid);
    return 1;
  } else
    temp = search_by_id(hp_queue, id);

  if (temp != NULL) {
    printf("Process already has HIGH priority\n");
    return 1;
  } else {
    printf("Could not find process\n");
    return -1;
  }
}

static int sched_set_low_p(int id) {
  process *temp = search_by_id(hp_queue, id);

  if (temp != NULL) {
    // if (lp_queue->queue_length == 2) {
    //   process *tt = search_by_name(lp_queue, SHELL_EXECUTABLE_NAME);
    //   enqueue(lp_queue, tt->pid, SHELL_EXECUTABLE_NAME, tt->id);
    //   dequeue(hp_queue, tt->pid);
    // }

    enqueue(lp_queue, temp->pid, temp->name, id);
    printf("ID: %d, PID: %d, NAME: %s has LOW priority now\n", temp->id,
           temp->pid, temp->name);
    dequeue(hp_queue, temp->pid);
    return 1;
  } else
    temp = search_by_id(lp_queue, id);

  if (temp != NULL) {
    printf("Process already has LOW priority\n");
    return 1;
  } else {
    printf("Could not find process\n");
    return -1;
  }
}

/* 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) {
  if (signum != SIGALRM) {
    fprintf(stderr, "Internal error: Called for signum %d, not SIGALRM\n",
            signum);
    exit(1);
  }
  // kill the proccess
  if (hp_queue->queue_length > 0) {
    if (kill(hp_queue->head->pid, SIGSTOP) < 0) {
      perror("kill");
      exit(1);
    }
  } else {
    if (kill(lp_queue->head->pid, SIGSTOP) < 0) {
      perror("kill");
      exit(1);
    }
  }
}

// kill the proccess
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 */

      if (hp_queue->queue_length) {
        printf("Parent: Received SIGCHLD, child is dead.\n");
        if (hp_queue->queue_length == 1) {
          // if (strcmp(hp_queue->head->name, SHELL_EXECUTABLE_NAME) == 0) {
          //   process *tt = hp_queue->head;
          //   dequeue(hp_queue, hp_queue->head->pid);
          //   enqueue(lp_queue, tt->pid, tt->name, tt->id);
          //   return;
          // }
        }

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

        // if (hp_queue->queue_length == 1) {
        //   if (strcmp(hp_queue->head->name, SHELL_EXECUTABLE_NAME) == 0) {
        //     process *tt = hp_queue->head;
        //     dequeue(hp_queue, hp_queue->head->pid);
        //     enqueue(lp_queue, tt->pid, tt->name, tt->id);
        //     return;
        //   }
        // }
        if (hp_queue->queue_length == 0) return;
        if (hp_queue->queue_length > 1) rotate_queue(hp_queue);

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

        if (kill(hp_queue->head->pid, SIGCONT) < 0) {
          perror("Continue to process");
          exit(1);
        }
      } else {
        printf("Parent: Received SIGCHLD, child is dead.\n");
        if (lp_queue->queue_length == 0) {
          printf("done\n");
          exit(0);
        }
        dequeue(lp_queue, lp_queue->head->pid);

        rotate_queue(lp_queue);

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

        if (kill(lp_queue->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... */

      if (hp_queue->queue_length) {
        // if (hp_queue->queue_length != 1) rotate_queue(hp_queue);
        // fprintf(stderr, "Proccess with pid=%ld is about to begin...\n",
        //         (long int)hp_queue->head->pid);
        // if (kill(hp_queue->head->pid, SIGCONT) < 0) {
        //   perror("Continue to process");
        //   exit(1);
        // }
      } else {
        rotate_queue(lp_queue);
        fprintf(stderr, "Proccess with pid=%ld is about to begin...\n",
                (long int)lp_queue->head->pid);
        if (kill(lp_queue->head->pid, SIGCONT) < 0) {
          perror("Continue to process");
          exit(1);
        }
      }
      /* Setup the alarm again */
      if (alarm(SCHED_TQ_SEC) < 0) {
        perror("alarm");
        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 lp_queue with the shell process
```

```c
  enqueue(lp_queue, p, SHELL_EXECUTABLE_NAME, -1);

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

  /* Create the shell. */

  /*
   * For each of argv[1] to argv[argc - 1],
   * create a new child process, add it to the process list.
   */
  hp_queue = safe_malloc(sizeof(hp_queue));
  lp_queue = safe_malloc(sizeof(lp_queue));
  hp_queue->queue_length = 0;
  lp_queue->queue_length = 0;
  queue_max = -1;

  sched_create_shell(SHELL_EXECUTABLE_NAME, &request_fd, &return_fd);

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

  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(lp_queue, pid, argv[i], -1);
      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);
  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(lp_queue->head->pid, SIGCONT) < 0) {
    perror("First child error with continuing");
    exit(1);
  }

  if (alarm(SCHED_TQ_SEC) < 0) {
    perror("alarm");
    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: Creating child...
Parent: Created child with PID = 18494, waiting
for it to terminate...
Parent: Creating child...
Parent: Created child with PID = 18495, waiting
for it to terminate...
Parent: Creating child...
A new proccess is created with pid=18494
I am prog, PID = 18494
About to replace myself with the executable
prog...
Parent: Created child with PID = 18496, waiting
for it to terminate...
Parent: Creating child...
A new proccess is created with pid=18496
I am prog, PID = 18496
About to replace myself with the executable
prog...
Parent: Created child with PID = 18497, waiting
for it to terminate...
My PID = 18492: Child PID = 18493 has been
stopped by a signal, signo = 19
A new proccess is created with pid=18497
I am prog, PID = 18497
```

```
About to replace myself with the executable
prog...
A new proccess is created with pid=18495
I am prog, PID = 18495
About to replace myself with the executable
prog...
My PID = 18492: Child PID = 18494 has been
stopped by a signal, signo = 19
My PID = 18492: Child PID = 18496 has been
stopped by a signal, signo = 19
My PID = 18492: Child PID = 18497 has been
stopped by a signal, signo = 19
My PID = 18492: Child PID = 18495 has been
stopped by a signal, signo = 19


scheduler-shell(18492)─┬─scheduler-shell(18493)
                       ├─scheduler-shell(18494)
                       ├─scheduler-shell(18495)
                       ├─scheduler-shell(18496)
                       ├─scheduler-shell(18497)

└─sh(18500)───pstree(18501)
```

```


This is the Shell. Welcome.

Shell> p
Shell: issuing request...
Shell: receiving request return value...
                 LOW priority list with size = 5
Current Process: ID: 0, PID: 18493, NAME: shell
                 ID: 1, PID: 18494, NAME: prog
                 ID: 2, PID: 18495, NAME: prog
                 ID: 3, PID: 18496, NAME: prog
                 ID: 4, PID: 18497, NAME: prog
Shell> k 3
ID: 3, PID: 18496, NAME: prog is being killed
Shell: issuing request...
My PID = 18492: Child PID = 18496 was terminated
by a signal, signo = 9
Parent: Received SIGCHLD, child is dead.
Proccess with pid=18494 is about to begin...
Shell: receiving request return value...
Shell> prog: Starting, NMSG = 40, delay = 1143
prog[18494]: This is message 0
prog[18494]: This is message 1
prog[18494]: This is message 2
My PID = 18492: Child PID = 18494 has been
stopped by a signal, signo = 19
Proccess with pid=18495 is about to begin...
prog: Starting, NMSG = 40, delay = 952
prog[18495]: This is message 0
prog[18495]: This is message 1
prog[18495]: This is message 2
My PID = 18492: Child PID = 18495 has been
stopped by a signal, signo = 19
Proccess with pid=18497 is about to begin...
prog: Starting, NMSG = 40, delay = 591
prog[18497]: This is message 0
prog[18497]: This is message 1
prog[18497]: This is message 2
prog[18497]: This is message 3
prog[18497]: This is message 4
My PID = 18492: Child PID = 18497 has been
stopped by a signal, signo = 19
Proccess with pid=18494 is about to begin...
prog[18494]: This is message 3
prog[18494]: This is message 4
My PID = 18492: Child PID = 18494 has been
stopped by a signal, signo = 19
Proccess with pid=18495 is about to begin...
prog[18495]: This is message 3
prog[18495]: This is message 4
prog[18495]: This is message 5
My PID = 18492: Child PID = 18495 has been
stopped by a signal, signo = 19
Proccess with pid=18497 is about to begin...
prog[18497]: This is message 5
prog[18497]: This is message 6
prog[18497]: This is message 7
prog[18497]: This is message 8
prog[18497]: This is message 9
My PID = 18492: Child PID = 18497 has been
stopped by a signal, signo = 19
Proccess with pid=18494 is about to begin...
prog[18494]: This is message 5
prog[18494]: This is message 6
My PID = 18492: Child PID = 18494 has been
stopped by a signal, signo = 19
Proccess with pid=18495 is about to begin...
prog[18495]: This is message 6
prog[18495]: This is message 7
prog[18495]: This is message 8
```

```
My PID = 18492: Child PID = 18495 has been
stopped by a signal, signo = 19
Proccess with pid=18497 is about to begin...
prog[18497]: This is message 10
prog[18497]: This is message 11
prog[18497]: This is message 12
prog[18497]: This is message 13
My PID = 18492: Child PID = 18497 has been
stopped by a signal, signo = 19
Proccess with pid=18494 is about to begin...
prog[18494]: This is message 7
prog[18494]: This is message 8
My PID = 18492: Child PID = 18494 has been
stopped by a signal, signo = 19
Proccess with pid=18495 is about to begin...
prog[18495]: This is message 9
prog[18495]: This is message 10
prog[18495]: This is message 11
My PID = 18492: Child PID = 18495 has been
stopped by a signal, signo = 19
Proccess with pid=18497 is about to begin...
prog[18497]: This is message 14
prog[18497]: This is message 15
prog[18497]: This is message 16
prog[18497]: This is message 17
My PID = 18492: Child PID = 18497 has been
stopped by a signal, signo = 19
Proccess with pid=18494 is about to begin...
prog[18494]: This is message 9
prog[18494]: This is message 10
My PID = 18492: Child PID = 18494 has been
stopped by a signal, signo = 19
Proccess with pid=18495 is about to begin...
prog[18495]: This is message 12
prog[18495]: This is message 13
My PID = 18492: Child PID = 18495 has been
stopped by a signal, signo = 19
Proccess with pid=18497 is about to begin...
prog[18497]: This is message 18
prog[18497]: This is message 19
prog[18497]: This is message 20
prog[18497]: This is message 21
prog[18497]: This is message 22
My PID = 18492: Child PID = 18497 has been
stopped by a signal, signo = 19
Proccess with pid=18494 is about to begin...
prog[18494]: This is message 11
p
Shell: issuing request...
Shell: receiving request return value...
                 LOW priority list with size = 3
Current Process: ID: 1, PID: 18494, NAME: prog
                 ID: 2, PID: 18495, NAME: prog
                 ID: 4, PID: 18497, NAME: prog
Shell> prog[18494]: This is message 12
h My PID = 18492: Child PID = 18494 has been
stopped by a signal, signo = 19
Proccess with pid=18495 is about to begin...
prog[18495]: This is message 14
 4
Shell: issuing request...
Shell: receiving request return value...
->>>>>>>>>>>>>>>>>> pid=18497
ID: 4, PID: 18497, NAME: prog has HIGH priority
now
Shell> prog[18495]: This is message 15
p
Shell: issuing request...
Shell: receiving request return value...
                 HIGH priority list with size = 1
Current Process: ID: 4, PID: 18497, NAME: prog
                 LOW priority list with size = 2
```

```
ID: 2, PID: 18495, NAME: prog
                 ID: 1, PID: 18494, NAME: prog
Shell> prog[18495]: This is message 16
prog[18495]: This is message 17
l 4
Shell: issuing request...
Shell: receiving request return value...
ID: 4, PID: 18497, NAME: prog has LOW priority
now
Shell> prog[18495]: This is message 18
pprog[18495]: This is message 19

Shell: issuing request...
Shell: receiving request return value...
                 LOW priority list with size = 3
Current Process: ID: 2, PID: 18495, NAME: prog
                 ID: 4, PID: 18497, NAME: prog
ID: 2, PID: 18495, NAME: prog
Shell> prog[18495]: This is message 20
q
Shell: Exiting. Goodbye.
My PID = 18492: Child PID = 18493 terminated
normally, exit status = 0
Parent: Received SIGCHLD, child is dead.
Proccess with pid=18497 is about to begin...
scheduler: read from shell: Success
Scheduler: giving up on shell request processing.
prog[18495]: This is message 21
prog[18497]: This is message 23
prog[18497]: This is message 24
prog[18495]: This is message 22
prog[18497]: This is message 25
prog[18495]: This is message 23
prog[18497]: This is message 26
My PID = 18492: Child PID = 18497 has been
stopped by a signal, signo = 19
Proccess with pid=18497 is about to begin...
prog[18497]: This is message 27
```

```
prog[18495]: This is message 24
prog[18497]: This is message 28
prog[18495]: This is message 25
prog[18497]: This is message 29
prog[18497]: This is message 30
prog[18495]: This is message 26
My PID = 18492: Child PID = 18497 has been
stopped by a signal, signo = 19
Proccess with pid=18497 is about to begin...
prog[18497]: This is message 31
prog[18495]: This is message 27
prog[18497]: This is message 32
prog[18497]: This is message 33
prog[18495]: This is message 28
prog[18497]: This is message 34
My PID = 18492: Child PID = 18497 has been
stopped by a signal, signo = 19
Proccess with pid=18497 is about to begin...
prog[18495]: This is message 29
prog[18497]: This is message 35
prog[18497]: This is message 36
prog[18495]: This is message 30
prog[18497]: This is message 37
prog[18495]: This is message 31
prog[18497]: This is message 38
My PID = 18492: Child PID = 18497 has been
stopped by a signal, signo = 19
Proccess with pid=18497 is about to begin...
prog[18497]: This is message 39
prog[18495]: This is message 32
My PID = 18492: Child PID = 18497 terminated
normally, exit status = 0
Parent: Received SIGCHLD, child is dead.
Proccess with pid=22012 is about to begin...
Continue to process: No such process
```

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.