ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ
http://www.cslab.ece.ntua.gr

Πέτρου Νικόλας	oslabb13	03117714
Πεγειώτη Νάταλυ		03117707

Λειτουργικά Συστήματα

6ο εξάμηνο, Ακαδημαϊκή περίοδος 2019-2020

Άσκηση 4
Χρονοδρομολόγηση

Άσκηση 1.1 – Ερωτήσεις

1. Τι συμβαίνει αν το σήμα SIGALRM έρθει ενώ εκτελείται η συνάρτηση χειρισμού του σήματος SIGCHLD ή το αντίστροφο; Πώς αντιμετωπίζει ένας πραγματικός χρονοδρομολογητής χώρου πυρήνα ανάλογα ενδεχόμενα και πώς η δική σας υλοποίηση;
Υπόδειξη: μελετήστε τη συνάρτηση install_signal_handlers() που δίνεται.
Η συνάρτηση install_signal_handlers() χρησιμοποιεί μία μάσκα η οποία δεν επιτρέπει και στους 2 handlers να τρέξουν μαζί και έτσι μπλοκάρει το ένα σήμα προσωρινά μέχρι την εξυπηρέτηση του τρέχοντος. Ένας πραγματικός χρονοδρομολογητής δουλεύει με ανάλογο τρόπο μπλοκάροντας ίδια σήματα και βάζοντας τα στην ουρά αναμονής προς εκτέλεση.

2. Κάθε φορά που ο χρονοδρομολογητής λαμβάνει σήμα SIGCHLD, σε ποια διεργασία-παιδί περιμένετε να αναφέρεται αυτό; Τι συμβαίνει αν λόγω εξωτερικού παράγοντα (π.χ. αποστολή SIGKILL) τερματιστεί αναπάντεχα μια οποιαδήποτε διεργασία-παιδί;
Κάθε φορά που ο χρονοδρομολογητής λαμβάνει σήμα SIGCHLD περιμένουμε να αναφέρεται στην τρέχουσα διεργασία. Αν όμως σταλεί κάποιο σήμα από εξωτερικό παράγοντα τότε θα ενεργοποιηθεί ο handler και εφόσον δεν έχουμε προνοήσει για κάτι τέτοιο ο χρονοδρομολογητής δεν θα λειτουργήσει σωστά.

3. Γιατί χρειάζεται ο χειρισμός δύο σημάτων για την υλοποίηση του χρονοδρομολογητή; θα μπορούσε ο χρονοδρομολογητής να χρησιμοποιεί μόνο το σήμα SIGALRM για να σταματά την τρέχουσα διεργασία και να ξεκινά την επόμενη; Τι ανεπιθύμητη συμπεριφορά θα μπορούσε να εμφανίζει μια τέτοια υλοποίηση; Υπόδειξη: Η παραλαβή του σήματος SIGCHLD εγγυάται ότι η τρέχουσα διεργασία έλαβε το σήμα SIGSTOP και έχει σταματήσει.
Χρειάζεται ο χειρισμός 2 σημάτων για τον λόγο ότι το σήμα SIGALRM δεν μας εξασφαλίζει ότι η τρέχουσα διεργασία έχει όντως σταματήσει παρά μόνο όταν λάβουμε το σήμα SIGCHLD. Έτσι έχουμε σίγουρα σωστή λειτουργεία αφού καμία διεργασία δεν μπορεί να ξεκινήσει χωρίς να έχει τελειώσει η τρέχουσα.

Ακολουθεί ενδεικτική εκτέλεση του χρονοδρομολογητή μας με 4 διεργασίες prog.
Όπως έχει οριστεί NMSG=200 και T_Q = 2.

```
oslabb13@os-node1:~/lab4$ ./scheduler prog prog prog prog
I am ./scheduler, PID = 7146
About to replace myself with the executable prog...
I am ./scheduler, PID = 7147
About to replace myself with the executable prog...
I am ./scheduler, PID = 7148
About to replace myself with the executable prog...
I am ./scheduler, PID = 7149
About to replace myself with the executable prog...
My PID = 7145: Child PID = 7146 has been stopped by a signal, signo = 19
My PID = 7145: Child PID = 7147 has been stopped by a signal, signo = 19
My PID = 7145: Child PID = 7148 has been stopped by a signal, signo = 19
My PID = 7145: Child PID = 7149 has been stopped by a signal, signo = 19
7146
prog: Starting, NMSG = 200, delay = 155
prog[7146]: This is message 0
prog[7146]: This is message 1
prog[7146]: This is message 2
prog[7146]: This is message 3
prog[7146]: This is message 4
7146 is stopping
My PID = 7145: Child PID = 7146 has been stopped by a signal, signo = 19
7147 is starting
prog: Starting, NMSG = 200, delay = 71
prog[7147]: This is message 0
prog[7147]: This is message 1
prog[7147]: This is message 2
prog[7147]: This is message 3
prog[7147]: This is message 4
prog[7147]: This is message 5
prog[7147]: This is message 6
prog[7147]: This is message 7
prog[7147]: This is message 8
prog[7147]: This is message 9
7147 is stopping
My PID = 7145: Child PID = 7147 has been stopped by a signal, signo = 19
7148 is starting
prog: Starting, NMSG = 200, delay = 119
prog[7148]: This is message 0
prog[7148]: This is message 1
prog[7148]: This is message 2
prog[7148]: This is message 3
prog[7148]: This is message 4
prog[7148]: This is message 5
7148 is stopping
My PID = 7145: Child PID = 7148 has been stopped by a signal, signo = 19
7149 is starting
prog: Starting, NMSG = 200, delay = 99
prog[7149]: This is message 0
prog[7149]: This is message 1
prog[7149]: This is message 2
prog[7149]: This is message 3
prog[7149]: This is message 4
prog[7149]: This is message 5
prog[7149]: This is message 6
7149 is stopping
My PID = 7145: Child PID = 7149 has been stopped by a signal, signo = 19
7146 is starting
prog[7146]: This is message 5
prog[7146]: This is message 6
prog[7146]: This is message 7
prog[7146]: This is message 8
7146 is stopping
```

Λόγω του ότι η έξοδος της εκτέλεσης είναι τεράστια παραθέτουμε ένα κομμάτι από την αρχή της εξόδου, το οποίο φαίνεται πιο πάνω, καθώς και το τέλος της το οποίο ακολουθεί:

```
My PID = 7145: Child PID = 7146 has been stopped by a signal, signo = 19
7146 is starting
prog[7146]: This is message 185
prog[7146]: This is message 186
prog[7146]: This is message 187
prog[7146]: This is message 188
7146 is stopping
My PID = 7145: Child PID = 7146 has been stopped by a signal, signo = 19
7146 is starting
prog[7146]: This is message 189
prog[7146]: This is message 190
prog[7146]: This is message 191
prog[7146]: This is message 192
prog[7146]: This is message 193
7146 is stopping
My PID = 7145: Child PID = 7146 has been stopped by a signal, signo = 19
7146 is starting
prog[7146]: This is message 194
prog[7146]: This is message 195
prog[7146]: This is message 196
prog[7146]: This is message 197
7146 is stopping
My PID = 7145: Child PID = 7146 has been stopped by a signal, signo = 19
7146 is starting
prog[7146]: This is message 198
prog[7146]: This is message 199
My PID = 7145: Child PID = 7146 terminated normally, exit status = 0
All processes have been executed
```

Άσκηση 1.2 – Ερωτήσεις

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

Όταν και ο φλοιός υφίσταται χρονοδρομολόγηση, εμφανίζεται πάντοτε ως τρέχουσα διεργασία ο φλοιός. Δεν θα μπορούσε στην περίπτωση μας να μην συμβεί για το λόγο ότι ο φλοιός χρονοδρομολογείται μαζί με τις άλλες διεργασίες οπότε η εντολή 'p' εκτελείται μόνο όταν χρονοδρομολογείται η διεργασία shell.

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

Αυτές οι κλήσεις είναι αναγκαίες έτσι ώστε να είμαστε σίγουροι ότι καθώς εκτελείται ο φλοιός δεν θα έρθει κάποιο άλλο εξωτερικό σήμα που θα δημιουργήσει πρόβλημα στην λειτουργία του, αλλάζοντας για παράδειγμα την λίστα διεργασιών.

Άσκηση 1.3 – Ερωτήσεις

1. Περιγράψτε ένα σενάριο δημιουργίας λιμοκτονίας.

Χρησιμοποιώντας προτεραιότητες για τις διεργασίες είναι πολύ πιθανό μία διεργασία με χαμηλή προτεραιότητα να μην εκτελεστεί ποτέ αν συνεχίσουν να καταφθάνουν διεργασίες υψηλής προτεραιότητας. Σε τέτοια περίπτωση παραμελούνται οι εργασίες χαμηλής προτεραιότητας έναντι των υψηλής.

Παράρτημα Κώδικα

Άσκηση 1.1

```
#include <errno.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <signal.h>
#include <string.h>
#include <assert.h>

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

#include "proc-common.h"
#include "request.h"

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

struct node_t
{
        pid_t pid;
        int serial_id;
        struct node_t *next;
};
typedef struct node_t node;

struct list_t
{
        node *head;
        node *tail;
};
typedef struct list_t list;

list lista;
int status;
pid_t current_pid;

list createList(void)
{
        list result;
        result.head = NULL;
        result.tail = NULL;
        return result;
}
```

```
void add (list *l, pid_t x, int n)
{
	node *temp;
	temp=(node *)malloc(sizeof(node));
	temp->pid = x;
	temp->serial_id = n;

	if (l->head == NULL)
	{
		temp->next = temp;
		l->head = temp;
		l->tail = temp;
	}
	else
	{
		l->tail->next = temp;
		l->tail = temp;
		l->tail->next = l->head;
	}
}

pid_t get_head (list *l)
{
	pid_t result;
	if (l->head == NULL)
		return -1;
	result = l->head->pid;
	return result;
}

void move_head (list *l)
{
	l->head = l->head->next;
	l->tail = l->tail->next;
}

void remove_head (list *l)
{
	node *temp, *prev;
	temp = l->head;
	prev = l->tail;

	if (temp == prev)
	{
		l->head = NULL;
		l->tail = NULL;
	}
```

```
	else
	{
		l->head = temp->next;
		prev->next = l->head;
	}
	free(temp);
}

/*
 * SIGALRM handler
 */
static void
sigalrm_handler(int signum)
{
	//assert(0 && "Please fill me!");
	printf("%d is stopping\n", current_pid);
	kill(current_pid, SIGSTOP);
}

/*
 * SIGCHLD handler
 */
static void
sigchld_handler(int signum)
{
	//assert(0 && "Please fill me!");
	pid_t p;
	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*/
			alarm(0);
			remove_head(&lista);
			current_pid = get_head(&lista);
			if (current_pid < 0)
			{
				printf("All processes have been executed\n");
				exit(1);
			}
```

```
                        printf("%d is starting\n", current_pid);
                        kill(current_pid, SIGCONT);
                        alarm(SCHED_TQ_SEC);
                }
                /* A child has died*/
                if (WIFSTOPPED(status))
                {
                        move_head(&lista);
                        current_pid = get_head(&lista);
                        printf("%d is starting\n",current_pid);
                        kill(current_pid, SIGCONT);
                        alarm(SCHED_TQ_SEC);
                }
        }
}



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

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

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

        /*
         * Ignore SIGPIPE, so that write()s to pipes
         * with no reader do not result in us being killed,
         * and write() returns EPIPE instead.
```

```
	 */
	if (signal(SIGPIPE, SIG_IGN) < 0) {
		perror("signal: sigpipe");
		exit(1);
	}
}

int main(int argc, char *argv[])
{
	lista = createList();
	int nproc;
	int i;
	pid_t p;
	nproc = argc - 1;
	for (i=1; i<=nproc; i++)
	{
		p = fork();
		add(&lista, p , i);
		if (p == 0)
		{
			char *newargv[] = {argv[i], NULL, NULL, NULL};
			char *newenviron[] = {NULL};
			printf("I am %s, PID = %ld\n", argv[0], (long)getpid());
			printf("About to replace myself with the executable %s...\n", argv[i]);
			sleep(2);
			raise(SIGSTOP);
			execve(argv[i], newargv, newenviron);
			/*execve() only returns an error*/
			perror("execve");
			exit(1);
		}
	}
	/*
	 * For each of argv[1] to argv[argc - 1],
	 * create a new child process, add it to the process list.
	 */

	//nproc = 0; /* number of proccesses goes here */

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

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

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

```
	current_pid = get_head(&lista);
	printf("%d\n", current_pid);
	kill(current_pid, SIGCONT);
	alarm(SCHED_TQ_SEC);


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

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