Харківський національний університет імені В.Н. Каразіна
Факультет комп’ютерних наук

Звіт
з лабораторної роботи №9
дисципліна: «Операційні системи»

Виконала: студентка групи КС34
Євстігнєєва Марія Олексіївна
Перевірив: Паршинцев Богдан Володимирович

Харків 2022

Лабораторное занятие №11

«Процессы. Сигналы»

На данном занятии следует познакомиться с сигналами — традиционным механизмом общения ядра и процессов пользователя, который может быть использован для организации простейшего межпроцессного взаимодействия.

Основные задания

Задание №1

Напишите программу, в которой определите пользовательскую реакцию процесса на сигналы SIGINT, SIGTERM (вывод информации о захвате соответствующего сигнала) и SIGUSR1 (завершение работы программы), восстановите поведение по умолчанию для сигнала SIGPROF и игнорируйте сигнал SIGHUP. Для задержки выполнения процесса используйте функцию pause() в бесконечном цикле. Убедитесь в работоспособности программы (с помощью команды kill оболочки отошлите процессу нужный сигнал и проследите за реакцией). Реализуйте две версии программы: с помощью функции signal и с помощью функции sigaction.

Задание №2

Напишите программу «*будильник*»: программа через командную строку получает требуемый интервал времени (в секундах, для удобства отладки и демонстрации) и текстовое сообщение; завершает основной процесс, а в дочернем с помощью функций alarm и pause через заданное время выводит в стандартный поток вывода заданный текст.

Задание №3

Напишите программу, в которой создаются два процесса (родительский и дочерний). Эти процессы должны поочередно, синхронизировано выводить сообщения в стандартный поток вывода. Процессы синхронизируют свою работу, посылая друг другу сигнал SIGUSR1 при помощи вызова kill.

Задание №4

Напишите программу, в которой создаются два процесса (родительский и дочерний). Родительский процесс заданное количество раз посылает дочернему процессу сигнал SIGUSR1 (можно взять один из сигналов реального времени) с дополнительной информацией (целое число - номер вызова). Дочерний процесс обрабатывает сигнал и в стандартный поток вывода выводит полученный номер, текстовое представление сигнала и дополнительную информацию. Затем основной процесс посылает дочернему сигнал SIGTERM, завершающие его работу, ожидает завершение дочернего процесса и завершает работу сам.

Результати:

1 версія (sigaction):

```
Process ID: 88
Сигнал переривання (SIGINT) зловлено!
```

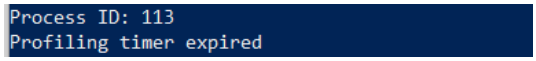
(1) Результати виконання програми

```
Process ID: 104
Сигнал завершення (SIGTERM) зловлено!
```

(2) Результати виконання програми

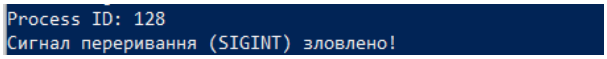
```
Process ID: 105
^[[A^[[B^[[B^[[B^[[B^[[BСигнал 1, визначений користувачем (SIGUSR1), зловлено!
```

(3) Результати виконання програми

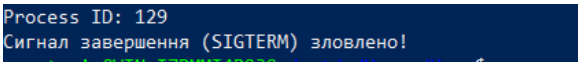


(4) Результати виконання програми

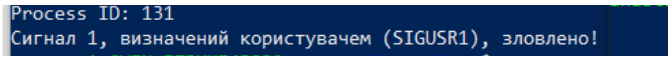
2 версія (signal):



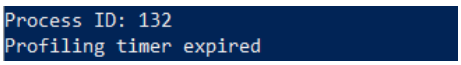
(1) Результати виконання програми



(2) Результати виконання програми



(3) Результати виконання програми



(4) Результати виконання програми

(5) Результати виконання програми

Код:

1 версія (sigaction):

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

/* обробник для SIGINT */
static void signal_handler(int signo) {
    switch (signo) {
        case SIGINT:
            // Usage: kill -2 PID
            fprintf(stdout, "Сигнал переривання (SIGINT) зловлено!\n");
            break;
        case SIGTERM:
            // Usage: kill -15 PID
            fprintf(stdout, "Сигнал завершення (SIGTERM) зловлено!\n");
            break;
//            User-defined signal 1
        case SIGUSR1:
            // Usage: kill -10 PID
            fprintf(stdout, "Сигнал 1, визначений користувачем (SIGUSR1),
зловлено!\n");
            exit(0);
        case SIGHUP:
            // Usage: kill -1 PID
            // do nothing
            break;
```

```
        default:
            fprintf(stderr, "Такого сигналу немає!!!");
            break;
    }
    exit(EXIT_SUCCESS);
}

int main(void) {
    printf("Process ID: %d\n", getpid());
    struct sigaction act;
    act.sa_handler = &signal_handler;
    sigfillset(&act.sa_mask);
    act.sa_flags = SA_RESTART;
    // Usage: kill -2 PID
    if (sigaction(SIGINT, &act, NULL) == -1) {
        perror("Неможливо обробити SIGINT!\n");
        exit(EXIT_FAILURE);
    }
    // Usage: kill -15 PID
    if (sigaction(SIGTERM, &act, NULL) == -1) {
        perror("Неможливо обробити SIGTERM!\n");
        exit(EXIT_FAILURE);
    }
    // Usage: kill -27 PID
    if (sigaction(SIGHUP, &act, NULL) == -1) {
        perror("Неможливо обробити SIGPROF!\n");
        exit(EXIT_FAILURE);
    }
    // Usage: kill -10 PID
    if (sigaction(SIGUSR1, &act, NULL) == -1) {
        perror("Неможливо обробити SIGUSR1!\n");
        exit(EXIT_FAILURE);
    }
    for (;;)
        pause();
    return 0;
}
```

2 версія (signal):

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

/* обробник для SIGINT */
static void signal_handler(int signo) {
    switch (signo) {
        case SIGINT:
            // Usage: kill -2 PID
            fprintf(stdout, "Сигнал переривання (SIGINT) зловлено!\n");
            break;
        case SIGTERM:
            // Usage: kill -15 PID
            fprintf(stdout, "Сигнал завершення (SIGTERM) зловлено!\n");
            break;
//            User-defined signal 1
        case SIGUSR1:
            // Usage: kill -10 PID
            fprintf(stdout, "Сигнал 1, визначений користувачем (SIGUSR1),
зловлено!\n");
```

```
            exit(0);
        case SIGHUP:
            // Usage: kill -1 PID
            // do nothing
            break;
        default:
            fprintf(stderr, "Такого сигналу немає!!!");
            break;
    }
    exit(EXIT_SUCCESS);
}

int main(void) {
    int p_id;
    p_id = getpid(); /*process id*/
    printf("Process ID: %d\n", p_id);

    if (signal(SIGINT, signal_handler) == SIG_ERR) {
        fprintf(stderr, "Неможливо обробити SIGINT!\n");
        exit(EXIT_FAILURE);
    }

    // ввічливий вихід
    if (signal(SIGTERM, signal_handler) == SIG_ERR) {
        fprintf(stderr, "Неможливо обробити SIGTERM!\n");
        exit(EXIT_FAILURE);
    }

    if (signal(SIGHUP, signal_handler) == SIG_ERR) {
        fprintf(stderr, "Неможливо скинути SIGPROF!\n");
        exit(EXIT_FAILURE);
    }

    if (signal(SIGUSR1, signal_handler) == SIG_ERR) {
        fprintf(stderr, "Неможливо ігнорувати SIGUSR1!\n");
        exit(EXIT_FAILURE);
    }

    for (;;)
        pause();
    return 0;
}
```

Завдання №2:

Результати:

```
[Child process] Alarm
standard message

[Parent process] Child with PID 164 finishes normally with code 0
```

(1) Результати виконання програми

```
[Child process] Alarm
привіт!

[Parent process] Child with PID 166 finishes normally with code 0
```

(2) Результати виконання програми

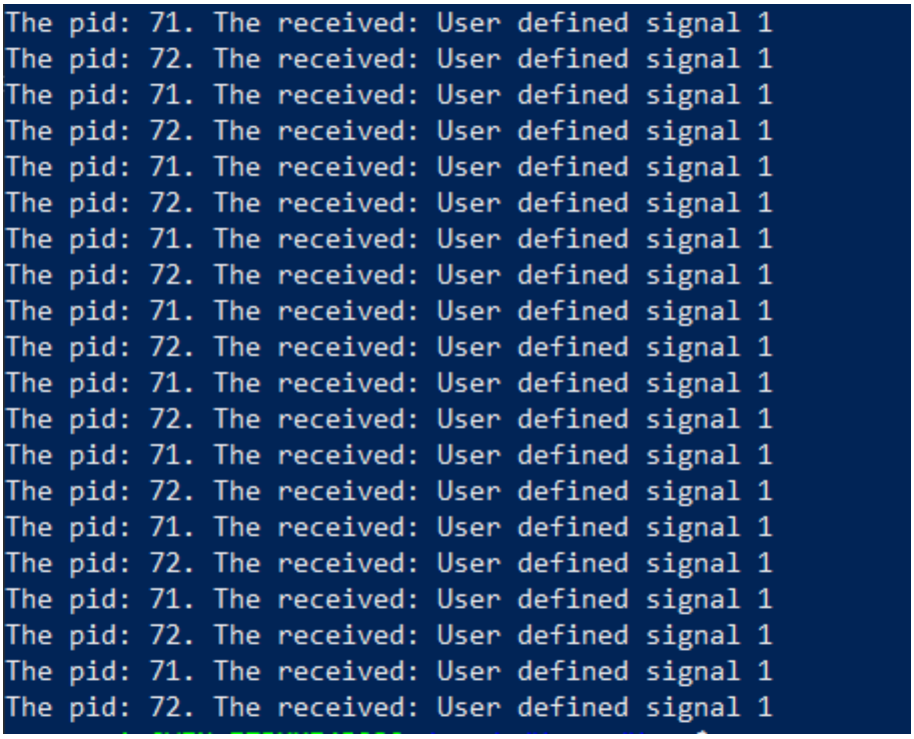
Код:

```c
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <signal.h>
#include <pthread.h>
#include <wait.h>

struct toProcess {
    int seconds;
    char* msg;
};

struct toProcess data;

sig_atomic_t alarm_flag = 0;

void setflag(int sig) {
    alarm_flag = 1;
}

int main(int argc, char *argv[]) {
    if (argc == 1) {
        data.seconds = 2;
        data.msg = "standard message";
    } else if (argc != 3) {
        printf("Вам необхідно ввести 2 аргументи!");
    } else {
        data.seconds = atoi(argv[1]);
        data.msg = argv[2];
    }

    pid_t pid = fork();

    if (pid > 0) {
        int status;
        pid = wait(&status);
        printf("\n[Parent process] Child with PID %d finishes ", (int)pid);
        if (WIFEXITED(status)) {
            printf("normally with code %d\n", WEXITSTATUS(status));
        } else if (WIFSIGNALED(status)) {
            printf("due to signal\n");
        }
    } else if (pid == 0) {
        static struct sigaction act;

        act.sa_handler = setflag;
        sigaction(SIGALRM, &act, NULL);
        alarm(data.seconds);
```

```
        pause();
        if (alarm_flag == 1) {
            printf("\n[Child process] ");
            printf("Alarm\n");
            printf("%s\n", data.msg);
        }
    } else {
        exit(EXIT_FAILURE);
    }

    return 0;
}
```

Завдання №3:

Результати:



```
The pid: 71. The received: User defined signal 1
The pid: 72. The received: User defined signal 1
The pid: 71. The received: User defined signal 1
The pid: 72. The received: User defined signal 1
The pid: 71. The received: User defined signal 1
The pid: 72. The received: User defined signal 1
The pid: 71. The received: User defined signal 1
The pid: 72. The received: User defined signal 1
The pid: 71. The received: User defined signal 1
The pid: 72. The received: User defined signal 1
The pid: 71. The received: User defined signal 1
The pid: 72. The received: User defined signal 1
The pid: 71. The received: User defined signal 1
The pid: 72. The received: User defined signal 1
The pid: 71. The received: User defined signal 1
The pid: 72. The received: User defined signal 1
The pid: 71. The received: User defined signal 1
The pid: 72. The received: User defined signal 1
The pid: 71. The received: User defined signal 1
The pid: 72. The received: User defined signal 1
```

Результати виконання програми

Код:

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

#define ITERATION 10

void handler(int sig) {
    fprintf(stderr, "The pid: %d. ", (int)getpid());
    psignal(sig, "The received");
}

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

```
    static struct sigaction act;
    sigset_t set1, set2;

    sigfillset(&set1);
    sigdelset(&set1, SIGUSR1);
    sigdelset(&set1, SIGINT);

    sigfillset(&set2);
    sigdelset(&set2, SIGINT);
    sigprocmask(SIG_SETMASK, &set2, NULL);

    act.sa_handler = handler;
    sigfillset(&act.sa_mask);
    sigdelset(&act.sa_mask, SIGUSR1);
    act.sa_flags = SA_RESTART;
    sigaction(SIGUSR1, &act, NULL);

    pid_t pid = fork();

    if (pid > 0) {
        for(int i = 0; i < ITERATION; i++) {
            sigsuspend(&set1);
            kill(pid, SIGUSR1);
        }
    } else if (pid == 0) {
        for(int i = 0; i < ITERATION; i++) {
            kill(getppid(), SIGUSR1);
            sigsuspend(&set1);
        }
    } else {
        exit(EXIT_FAILURE);
    }

    return 0;
}
```