

МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра Вычислительной техники

ОТЧЁТ

по лабораторной работе №6

по дисциплине «Организация процессов и программирования в среде Linux»

Тема: ОРГАНИЗАЦИЯ ПЕРИОДИЧЕСКИХ ПРОЦЕССОВ

Студент гр. 9308

Преподаватель

Соболев М.С.

Разумовский Г.В.

Санкт-Петербург,

2022

Оглавление

1. Введение.....	3
1.1. Введение.....	3
1.2. Порядок выполнения работы.....	3
1.3. Содержание отчёта.....	4
2. Текст периодической программы.....	5
3. Скриншот экрана результатов работы периодической программы после всех её перезапусков.....	8
4. Вывод.....	10
5. Список использованных источников.....	11

1. Введение

1.1. Введение

Тема работы: Организация периодических процессов.

Цель работы: Использование сервиса cron, механизма сигналов и интервальных таймеров для организации периодических процессов.

1.2. Порядок выполнения работы

1. Создать пользовательский файл конфигурации сервиса cron, в котором содержатся команды периодического запуска одной из программ, разработанных в предыдущих лабораторных работах. Результаты работы этой программы должны выводиться или переадресовываться в файл.

2. После нескольких запусков программы удалить пользовательский файл конфигурации.

3. Написать периодическую программу, в которой период запуска и количество запусков должны задаваться в качестве её параметров. При каждом очередном запуске программа должна порождать новый процесс, который выводит на экран свой идентификатор, дату и время старта. Программа и её дочерний процесс должны быть заблокированы от завершения при нажатии клавиши Ctrl/z. После завершения дочернего процесса программа должна вывести на экран информацию о времени своей работы и дочернего процесса.

4. Откомпилировать программу и запустить её несколько раз с разным периодом запуска и количеством повторений.

Выбранные задания: 3, 4.

1.3. Содержание отчёта

Отчёт по лабораторной работе должен содержать:

1. Цель и задание.
2. Распечатки файлов расписания и результатов работы программы.
3. Текст периодической программы.
4. Скриншот экрана результатов работы периодической программы после всех её перезапусков.

2. Текст периодической программы

```
// start program
// ./main <launch period> <number of launches>
// <launch period> -- launch period in seconds (time between every launch)
// <number of launches> -- number of program launches (how many times program will launch)

#include <iostream>
#include <signal.h>
#include <sys/wait.h>
#include <sys/time.h>
#include <unistd.h>

using namespace std;

void processFunction (int local_int); // function for repeat
float finalTime (float local_start_time); // counting final time

int main(int argc, char *argv[])
{
    int number_period = 0; // launch period, 0 by default
    int number_launch = 0; // number of launches, 0 by default
    float start_time; // parent process start time
    struct itimerval value; // for timer structure, new time set
    struct itimerval old_value; // for timer structure, old timer dropped here
    struct sigaction sigact;

    number_period = atoi(argv[1]); // convert input data to numeric
    number_launch = atoi(argv[2]); // convert input data to numeric

    //start_time = (float)clock() / CLOCKS_PER_SEC; // measuring the current time

    sigact.sa_handler = processFunction; // setting handling function
    sigemptyset(&sigact.sa_mask); // cleaning "set" from all signals

    sigaddset(&sigact.sa_mask, SIGTSTP); // SIGTSTP signal adding to mask
    sigprocmask(SIG_BLOCK, &sigact.sa_mask, NULL); // SIGTSTP signal blocking
    sigact.sa_flags = 0; // no flags added
```

```

sigaction(SIGALRM, &sigact, NULL); // setting response to SIGALRM signal

// settings for 1st launch, because time (10 us) will be left very fast
value.it_value.tv_sec = 0; // setting 0 seconds. only useconds
value.it_value.tv_usec = 10; // setting 10 useconds for fast timer counting --> faster the signal will be sended

// setting new interval
value.it_interval.tv_sec = number_period; // interval s, launch period
value.it_interval.tv_usec = 0; // interval us, launch period

setitimer(ITIMER_REAL, &value, &old_value); // ITIMER_REAL means always

for (int i = 0; i < number_launch; i++)
{
    start_time = (float)clock() / CLOCKS_PER_SEC; // measuring the current time
    pause(); // SIGALRM signal waiting
    cout << "Parent process' work time (seconds): " << finalTime(start_time) << "\n\n";
}

return 0;
}

void processFunction (int local_int) // function for repeat
{
    int local_status; // status for "waitpid" func
    float local_start_time; // initilaizing
    pid_t local_pid = fork(); // creating child process

    if (local_pid == 0) // if child is created
    {
        time_t local_seconds = time (NULL); // seconds since
        sigset_t local_set; // signal set
        struct tm* local_time = localtime (&local_seconds); // time date
        local_start_time = (float)clock() / CLOCKS_PER_SEC; // counting current time since starting
program

        sigemptyset(&local_set); // setting emty signal set

```

```
cout << "Child process' PID: " << getpid() << "\n"; // get child process' parent ID
```

```
cout << "Parent process' work start time: " << asctime(local_time); //<< "\n";
```

```
sigaddset(&local_set, SIGTSTP); // adding signal SIGTSTP (anti ctrl-z) to the current process
```

```
sigprocmask(SIG_BLOCK, &local_set, NULL); // adding blocked signals to the set, SIG_BLOCK
```

means blocked signals are current set + set argument in function

```
//cout << "Child process' work time (seconds): " << finalTime(local_start_time) << "\n"; // cout child
```

process time

```
exit(EXIT_SUCCESS);
```

```
}
```

```
else // else if it is not child or child is not created
```

```
{
```

```
waitpid(local_pid, &local_status, 0); // wait until child is terminate its work
```

```
cout << "Child process' work time (seconds): " << finalTime(local_start_time) << "\n"; // cout child
```

process time

```
}
```

```
}
```

```
float finalTime (float local_start_time) // counting current time since starting prohran & last function
```

```
{
```

```
    //float local_end_time = 0;
```

```
    //local_end_time = ((float)clock()) / CLOCKS_PER_SEC;
```

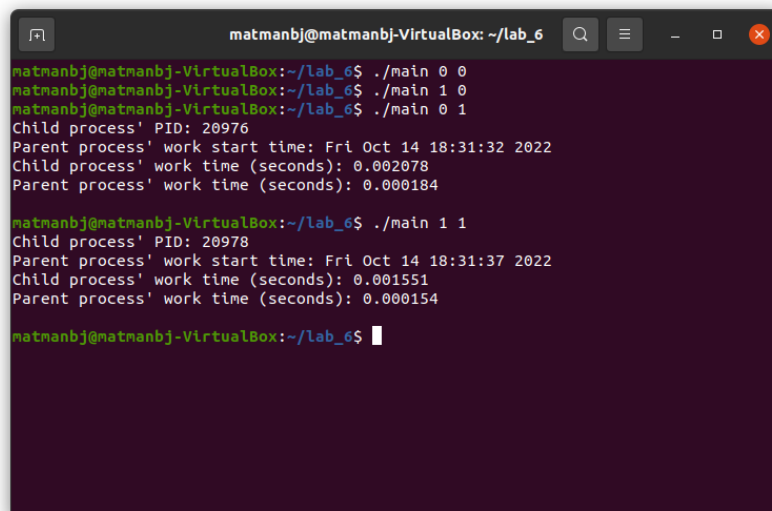
```
    return (((float)clock()) / CLOCKS_PER_SEC) - local_start_time;
```

```
}
```

3. Скриншот экрана результатов работы периодической программы после всех её перезапусков

Программа запускается с помощью команды «./main <период запуска> <количество запусков>», где <период запуска> – это период, который проходит между запусками программы и который измеряется в секундах, и где <количество запусков> – это количество раз, которое запускается программа.

При запуске программы с количество запусков, равным 0, программа не запустится. При запуске программы с периодом запуска, равным 0, программа будет запускаться.



```
matmanbj@matmanbj-VirtualBox: ~/lab_6
matmanbj@matmanbj-VirtualBox:~/lab_6$ ./main 0 0
matmanbj@matmanbj-VirtualBox:~/lab_6$ ./main 1 0
matmanbj@matmanbj-VirtualBox:~/lab_6$ ./main 0 1
Child process' PID: 20976
Parent process' work start time: Fri Oct 14 18:31:32 2022
Child process' work time (seconds): 0.002078
Parent process' work time (seconds): 0.000184

matmanbj@matmanbj-VirtualBox:~/lab_6$ ./main 1 1
Child process' PID: 20978
Parent process' work start time: Fri Oct 14 18:31:37 2022
Child process' work time (seconds): 0.001551
Parent process' work time (seconds): 0.000154

matmanbj@matmanbj-VirtualBox:~/lab_6$
```

Рисунок 1. Запуск программы с периодом запуска и количеством раз, равными «0 0», «1 0», «0 1» и «1 1» соответственно


```
matmanbj@matmanbj-VirtualBox: ~/lab_6
matmanbj@matmanbj-VirtualBox:~/lab_6$ ./main 3 5
Child process' PID: 21048
Parent process' work start time: Fri Oct 14 18:32:28 2022
Child process' work time (seconds): 0.001934
Parent process' work time (seconds): 0.000165

^Z^ZChild process' PID: 21049
Parent process' work start time: Fri Oct 14 18:32:31 2022
Child process' work time (seconds): 0.002272
Parent process' work time (seconds): 0.000338

^Z^Z^ZChild process' PID: 21050
Parent process' work start time: Fri Oct 14 18:32:34 2022
Child process' work time (seconds): 0.002581
Parent process' work time (seconds): 0.000276

Child process' PID: 21051
Parent process' work start time: Fri Oct 14 18:32:37 2022
Child process' work time (seconds): 0.002865
Parent process' work time (seconds): 0.000275

Child process' PID: 21052
Parent process' work start time: Fri Oct 14 18:32:40 2022
Child process' work time (seconds): 0.003147
```

Рисунок 2. Запуск программы с периодом запуска и количеством раз, равными «3 5» соответственно, с нажатием клавиш «Ctrl-Z»

```
matmanbj@matmanbj-VirtualBox: ~/lab_6
Child process' work time (seconds): 0.001934
Parent process' work time (seconds): 0.000165

^Z^ZChild process' PID: 21049
Parent process' work start time: Fri Oct 14 18:32:31 2022
Child process' work time (seconds): 0.002272
Parent process' work time (seconds): 0.000338

^Z^Z^ZChild process' PID: 21050
Parent process' work start time: Fri Oct 14 18:32:34 2022
Child process' work time (seconds): 0.002581
Parent process' work time (seconds): 0.000276

Child process' PID: 21051
Parent process' work start time: Fri Oct 14 18:32:37 2022
Child process' work time (seconds): 0.002865
Parent process' work time (seconds): 0.000275

Child process' PID: 21052
Parent process' work start time: Fri Oct 14 18:32:40 2022
Child process' work time (seconds): 0.003147
Parent process' work time (seconds): 0.000275
matmanbj@matmanbj-VirtualBox:~/lab_6$
```

Рисунок 3. Запуск программы с периодом запуска и количеством раз, равными «3 5» соответственно, с нажатием клавиш «Ctrl-Z»

4. Вывод

В ходе выполнения лабораторной работы №6 «Организация периодических процессов» были изучены системные функции, которые позволяли периодически запускать программу, была изучена возможность устанавливать пользователем количество запусков и период в секундах, через который эти запуски будут повторяться. Таким образом и было изучено использование сервиса cron, механизма сигналов и интервальных таймеров для организации периодических процессов.

5. Список использованных источников

1. Онлайн-курс «Организация процессов и программирование в среде Linux» в LMS Moodle [сайт]. URL: <https://vec.etu.ru/moodle/course/view.php?id=9703>.

2. Разумовский Г.В. Организация процессов и программирование в среде Linux: учебно-методическое пособие. СПб.: Изд-во СПбГЭТУ «ЛЭТИ», 2018. 40с.