МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

## Факультет информационных технологий и робототехники

Кафедра программного обеспечения информационных систем и технологий

**Отчет по лабораторной работе № 12**

по дисциплине:”Системное программирование”

на тему: “ Сигналы в Linux”

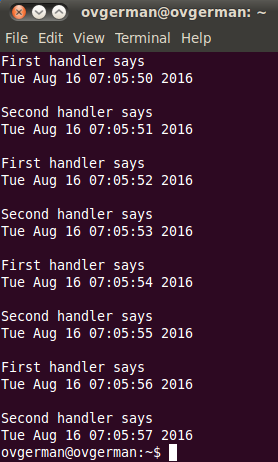
Выполнил**:** студент группы 10701321

Грудинский К. А.

Принял**:** Кондратёнок Е.В.

Минск 2023

# Лабораторная работа № 12. Сигналы в Linux.

**Задание 1**. Создайте два обработчика сигнала SIGALRM.

Здесь должны быть объявлены два обработчика для сигнала типа SIGALRM. Логика работы обоих обработчиков одна и та же. Скриншот дан.

**Решение**

#include <iostream>

#include <csignal>

#include <ctime>

#include <unistd.h>

void alarmHandler1(int signum) {

time\_t currentTime = time(nullptr);

tm\* localTime = localtime(&currentTime);

std::cout << "First handler says\n"

<< asctime(localTime)<<std::endl;

}

void alarmHandler2(int signum) {

time\_t currentTime = time(nullptr);

tm\* localTime = localtime(&currentTime);

std::cout << "Second handler says\n"

<< asctime(localTime)<<std::endl;

}

int main() {

while (true) {

signal(SIGALRM, alarmHandler1);

alarm(1);

sleep(1);

signal(SIGALRM, alarmHandler2);

alarm(1);

sleep(1);

}

return 0;

}

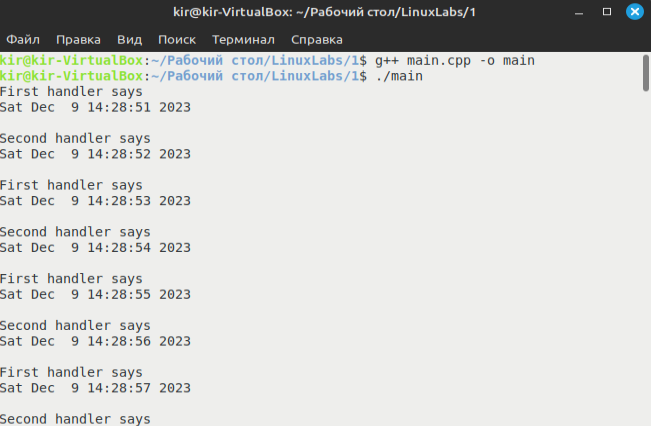


Рисунок 1 – результат выполнения задания 1

**Задание** 2. Измените последнюю программу так, чтобы она выдавала не время и дату, а счет в игре двух человек. Первый обработчик играет за первого человека (условно говоря). Второй обработчик играет за второго человека. Первый обработчик добавляет случайное число к общей переменной, второй – отнимает. Играют до 10 раз. Если в конце игры на счету будет число больше нуля – то выиграл первый игрок, иначе выиграл второй игрок. Случайное число разыгрывается таким образом:

Для получения случайных чисел можно использовать устройство-генератор случайных чисел Linux или библиотечные функции. Последняя возможность иллюстрируется следующей программой:

#include "stdafx.h"

#include <stdio.h>

#include <time.h>

#include <stdlib.h>

#include <conio.h>

int \_tmain(int argc, \_TCHAR\* argv[])

{

int player [5] = {0,0,0,0,0};

srand(time(NULL)); //инициализация случайного генератора

for (int i = 0; i <5 ; i++ ){

player[i] =rand() % 100;

}

for (int i = 0; i <5 ;i++ ){

printf ("Player[%d] - %d\n",i, player[i]);

}

getch();

return 0;

}

Функция srand устанавливает начальное состояние генератора случайных чисел. Собственно, случайные числа в диапазоне от 1 до 100 получаются в команде player[i] =rand () % 100;

Функция возвращает случайное число от 1 до 100. Просто rand() возвращает достаточно большое случайное число, ограниченно системной константой.

Функция random\_ возвращает целое случайное число в диапазоне от min до max. Устройство для получения случайных чисел открывается в команде

dev\_random = open (“/dev/random”, O\_RDONLY);

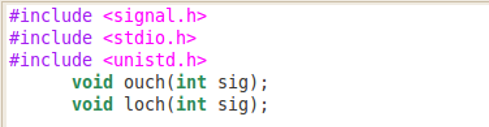
Считывание случайного числа выполняется в команде

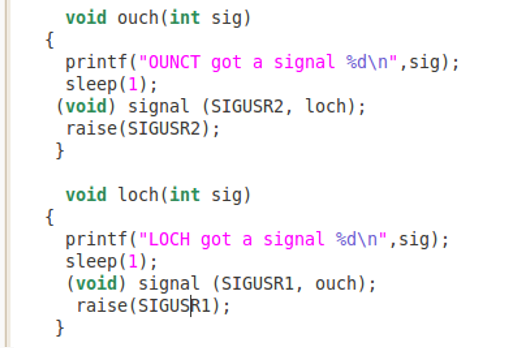
bytes\_read = read (dev\_random, newbyte, sizeof (random\_value));

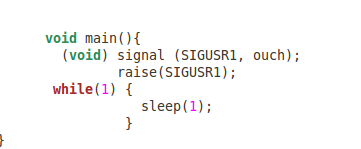
Случайное число помещается в переменную newbyte. Обычно случайные числа, генерируемые машиной, довольно большие. Чтобы уложить их в нужный диапазон, используем преобразование: newbyte % (max-min). Операция % соответствует взятию остатка от деления. Остаток от деления не может превосходить делимое, т.е. max-min.

Теперь познакомимся с сигналами пользователя – SIGUSR1 и SIGUSR2.

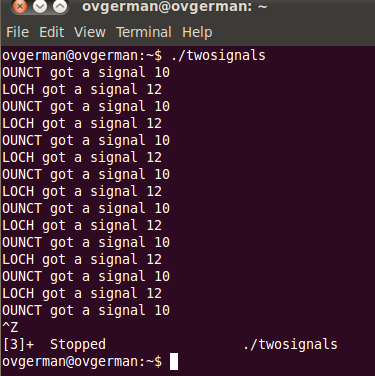
Рассмотрим следующий пример







Отметим, что команда (void) signal (…) не посылает сигнал, а связывает сигнал с обработчиком. Сигнал зажигается по команде raise(…). В нашем примере использованы два сигнала пользователя и два обработчика.

Работу их легко понять из порождаемого скриншота.

Происходит обмен сигналами между функциями с поочередным выводом на экран.

(цикл закрыть сигналом килл)

**Решение**

#include <iostream>

#include <csignal>

#include <ctime>

#include <cstdlib>

#include <unistd.h>

int score = 0; // Общий счет в игре

void alarmHandler1(int signum) {

int randomScore = rand() % 10 + 1;

score += randomScore;

std::cout << "Первый игрок добавил " << randomScore << ". Общий счет: " << score << std::endl;

}

void alarmHandler2(int signum) {

int randomScore = rand() % 10 + 1;

score -= randomScore;

std::cout << "Второй игрок отнял " << randomScore << ". Общий счет: " << score << std::endl;

}

int main() {

srand(time(nullptr)); // Инициализация генератора случайных чисел

int moves = 10; // Количество ходов

while (moves > 0) {

signal(SIGALRM, alarmHandler1);

alarm(1);

sleep(1);

signal(SIGALRM, alarmHandler2);

alarm(1);

sleep(1);

moves--;

}

// Определение победителя

if (score > 0) {

std::cout << "Выиграл первый игрок!" << std::endl;

} else if (score < 0) {

std::cout << "Выиграл второй игрок!" << std::endl;

} else {

std::cout << "Ничья! Оба игрока набрали одинаковое количество очков." << std::endl;

}

return 0;

}

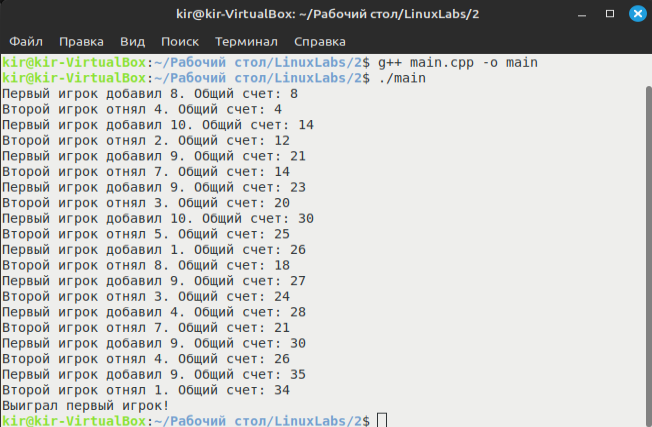


Рисунок 2 – выполнение задания 2

**Задание 3**. Представим, что есть клиент банка, который справляется о состоянии своего счета. Клиент посылает запрос (сигнал), банк возвращает ответ (сумма). Допустим, что сумма постепенно растет с каждым вызовом по формуле

X= X+0.1\*X

Написать соответствующий код.

Указание – использовать предыдущую программу.

**Решение**

*#include <iostream>*

*#include <csignal>*

*#include <ctime>*

*#include <cstdlib>*

*#include <unistd.h>*

*double accountBalance = 100.0;*

*void updateBalance(int signum) {*

*accountBalance\*=1.1;*

*std::cout<<"Обновлённый баланс: "<<accountBalance<<std::endl;*

*}*

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

*if (argc != 2) {*

*std::cerr << "Неверное количество параметров" << std::endl;*

*return 1;*

*}*

*int moves = std::stoi(argv[1]);*

*signal(SIGALRM, updateBalance);*

*while (moves > 0) {*

*alarm(1);*

*sleep(1);*

*moves--;*

*}*

*std::cout<<"Конченый баланс: "<<accountBalance;*

*return 0;*

*}*

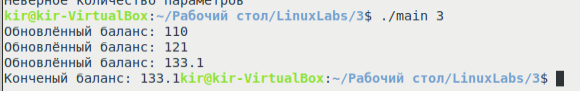


Рисунок 3 – выполнение задания 3

**Дополнение к заданию 3.** Перестройте программу так чтобы сигналы передавались между отдельными процессами (программами), а не функциями одной программы.

**Ркшение**

*Клиент*

#include <iostream>

#include <csignal>

#include <unistd.h>

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

if (argc != 2) {

std::cerr << "Неверное количество параметров" << std::endl;

return 1;

}

int moves = std::stoi(argv[1]);

pid\_t serverPid = fork();

if (serverPid == -1) {

perror("Ошибка при создании процесса сервера");

return 1;

} else if (serverPid == 0) {

execl("./server", "server", nullptr);

perror("Ошибка при запуске сервера");

return 1;

} else {

sleep(1);

while(moves>0)

{

kill(serverPid, SIGUSR1);

sleep(2);

moves--;

}

// Завершаем сервер после отправки сигнала

kill(serverPid, SIGTERM);

}

return 0;

}

*Сервер.*

#include <iostream>

#include <csignal>

#include <unistd.h>

double accountBalance = 100.0;

void updateBalance(int signum) {

accountBalance \*= 1.1;

std::cout << "Обновлённый баланс: " << accountBalance << std::endl;

}

int main() {

signal(SIGUSR1, updateBalance);

while (true) {

pause();

}

return 0;

}



Рисунок 3 – выполнение дополнения к заданию 3

Вывод*:* В ходе выполнения лабораторной работы был освоена работа с сигналами в Linux, были созданы программы использующие механизм сигналов.

Контрольные вопросы

1. Что может отправлять сигналы в системе? Что может являться отправителем сигнала?

В системе Linux сигналы используются для управления процессами. Сигналы могут быть отправлены различными сущностями в системе, такими как:

* Другие процессы: Процессы могут посылать сигналы другим процессам для уведомления их о различных событиях или для запроса определенных действий.
* Ядро операционной системы: Ядро может посылать сигналы процессам для оповещения о различных событиях в системе, таких как ошибка деления на ноль, обращение к недопустимой памяти и т.д.
* Пользователь: Пользователь может отправлять сигналы процессам с помощью команд в командной строке или с использованием утилит, таких как kill.

Некоторые общие сигналы, которые могут быть отправлены в системе Linux, включают в себя:

* SIGTERM (15): Этот сигнал отправляется для запроса завершения процесса. Процесс имеет возможность выполнить дополнительные действия перед завершением.
* SIGKILL (9): Этот сигнал используется для немедленного завершения процесса. Процесс не имеет возможности обработать или игнорировать этот сигнал.
* SIGINT (2): Этот сигнал отправляется при нажатии комбинации клавиш Ctrl+C в терминале. Обычно используется для прерывания выполнения процесса в терминале.
* SIGHUP (1): Этот сигнал отправляется при закрытии терминала или при отключении пользователя. Некоторые демоны используют SIGHUP для перезапуска.
* SIGUSR1 (10) и SIGUSR2 (12): Эти сигналы являются пользовательскими и могут быть использованы процессами для своих нужд.