Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Операционные среды и системное программирование

ОТЧЁТ

к лабораторной работе №4

на тему

**УПРАВЛЕНИЕ ПРОЦЕССАМИ И ВЗАИМОДЕЙСТВИЕ**

**ПРОЦЕССОВ**

|  |
| --- |
| Выполнил: студент гр. 253503  Кудош А.С. |
| Проверил: ассистент кафедры информатики Гриценко Н.Ю. |

Минск 2025

**СОДЕРЖАНИЕ**

[1 Постановка задачи и программа для её решения 3](#_Toc193113658)

[1.1 Постановка задачи 3](#_Toc193113659)

[1.2 Демонстрация работы программы 3](#_Toc193113660)

[Заключение 5](#_Toc193113661)

[Список использованных источников 6](#_Toc193113662)

[Приложение А (справочное) Исходный код 7](#_Toc193113663)

**1 ПОСТАНОВКА ЗАДАЧИ И ПРОГРАММА ДЛЯ ЕЁ РЕШЕНИЯ**

**1.1 Постановка задачи**

Целью данной лабораторной работы является изучение основных особенностей подсистемы управления процессами и средств взаимодействия процессов в операционной системе *Unix*. В рамках работы предполагается практическое проектирование, реализация и отладка программного комплекса, состоящего из нескольких взаимодействующих процессов. Основное внимание уделяется таким аспектам, как порождение процессов с использованием системных вызовов *fork*, *exec* и их комбинации *fork-exec*, завершение процессов и анализ кодов завершения, а также работа с сигналами *Unix*, включая надежные и ненадежные сигналы [1]. Кроме того, рассматриваются базовые средства межпроцессного взаимодействия (*IPC*), такие как каналы, семафоры, очереди сообщений и разделяемая память. В процессе выполнения работы необходимо изучить типичные задачи, проблемы и подходы к их решению, а также модели организации взаимодействия процессов.

Практическая часть работы заключается в разработке программы, которая преобразуется в демон и выполняет функцию приема и протоколирования заданных сигналов. Список сигналов, которые должны быть зарегистрированы, задается в файле конфигурации и считывается процессом при запуске, а также при получении сигнала *SIGHUP*. Сигналы *SIGHUP* и *SIGTERM* сохраняют свои стандартные эффекты (реконфигурация и завершение процесса соответственно), но также могут быть зарегистрированы в лог-файле наравне с другими сигналами [2]. Дополнительно предусмотрена возможность выгрузки демона из памяти по команде, передаваемой при запуске исполняемого файла с соответствующей опцией, например, *mydaemon -q* (или *--quit*). Для взаимодействия между процессами используется сигнал *SIGTERM*, который передается демону для завершения его работы. Программа должна быть спроектирована таким образом, чтобы обеспечивать корректное создание и удаление файлов конфигурации, логов и *PID*-файлов, а также корректную обработку ошибок, связанных с правами доступа и другими системными ограничениями. В процессе выполнения работы необходимо обеспечить тестирование и демонстрацию работы программы, включая проверку корректности обработки сигналов, создания и удаления файлов, а также завершения работы демона. Для управления проектом рекомендуется использовать утилиту *make* и сценарии *makefile*, что позволяет автоматизировать процесс сборки и тестирования программы.

**1.2 Демонстрация работы программы**

Программный код предоставлен в приложении А.

Для реализации поставленной задачи была создана программа-демон, принимающая сигналы, которые описаны в файле конфигурации. Для отправки сигнала следует взять *PID* демона из соответствующего файла (рисунок 1.1). После получения сигнала программа-демон запишет лог о нём в файл (рисунок 1.2).

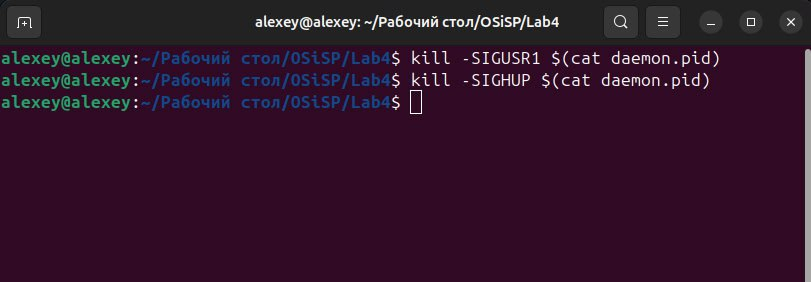


Рисунок 1.1 – Отправка сигналов

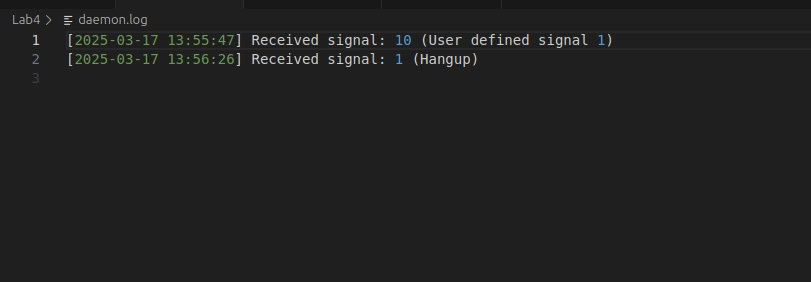


Рисунок 1.2 – Логирование полученных сигналов

**ЗАКЛЮЧЕНИЕ**

В ходе выполнения лабораторной работы были изучены основные механизмы управления процессами в операционной системе *Unix*, включая порождение процессов с использованием системных вызовов *fork* и *exec*, завершение процессов и обработку сигналов. Практическая часть работы заключалась в разработке программы, которая преобразуется в демона и выполняет функцию приема и протоколирования заданных сигналов. В процессе реализации были рассмотрены такие аспекты, как работа с файлами конфигурации, создание и удаление *PID*-файлов, а также логирование событий в файл.

Программа успешно продемонстрировала свою работоспособность: корректно создавала файлы *daemon.pid* и *daemon.log*, обрабатывала сигналы, указанные в конфигурационном файле, и записывала информацию о них в лог. Также была реализована возможность завершения работы демона как через отправку сигнала *SIGTERM*, так и через передачу специальной опции *-q* при запуске программы.

Важным аспектом работы стала обработка ошибок, связанных с правами доступа и системными ограничениями, что позволило обеспечить стабильную работу программы в различных условиях. Использование утилиты *make* для управления проектом упростило процесс сборки и тестирования программы.

Таким образом, поставленные задачи были выполнены, а полученные знания и навыки могут быть применены для разработки более сложных программных комплексов, взаимодействующих через механизмы межпроцессного взаимодействия в *Unix*-подобных системах.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1] Управление процессами в Linux [Электронный ресурс]. – Режим доступа: https://www.opennet.ru/docs/RUS/lnx\_process/process2.html.

[2] Как работают сигналы Linux: SIGINT, SIGTERM и SIGKILL [Электронный ресурс]. – Режим доступа: https://wiki.merionet.ru/articles/kak-rabotaiut-signaly-linux-sigint-sigterm-i-sigkill.

ПРИЛОЖЕНИЕ А  
(справочное)  
Исходный код

Листинг А.1 – Главная программа

#include <unistd.h>

#include <sys/stat.h>

#include <fcntl.h>

#include <signal.h>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <time.h>

#include <sys/types.h>

#include <sys/wait.h>

#define CONFIG\_FILE "daemon.conf"

#define LOG\_FILE "daemon.log"

#define PID\_FILE "daemon.pid"

sigset\_t signals\_to\_log;

typedef struct {

const char \*name;

int signo;

} SigInfo;

SigInfo signal\_table[] = {

{"SIGHUP", SIGHUP}, {"SIGINT", SIGINT}, {"SIGQUIT", SIGQUIT},

{"SIGILL", SIGILL}, {"SIGTRAP", SIGTRAP}, {"SIGABRT", SIGABRT},

{"SIGBUS", SIGBUS}, {"SIGFPE", SIGFPE}, {"SIGKILL", SIGKILL},

{"SIGUSR1", SIGUSR1}, {"SIGSEGV", SIGSEGV}, {"SIGUSR2", SIGUSR2},

{"SIGPIPE", SIGPIPE}, {"SIGALRM", SIGALRM}, {"SIGTERM", SIGTERM},

{NULL, 0}

};

int get\_signo(const char \*name) {

for (int i = 0; signal\_table[i].name != NULL; i++) {

if (strcmp(name, signal\_table[i].name) == 0) {

return signal\_table[i].signo;

}

}

return -1;

}

void read\_config() {

sigemptyset(&signals\_to\_log);

FILE \*fp = fopen(CONFIG\_FILE, "r");

if (!fp) return;

char line[256];

while (fgets(line, sizeof(line), fp)) {

line[strcspn(line, "\n")] = 0;

int signo = get\_signo(line);

if (signo != -1) {

sigaddset(&signals\_to\_log, signo);

}

}

fclose(fp);

}

void log\_signal(int sig) {

time\_t now = time(NULL);

char time\_buf[64];

strftime(time\_buf, sizeof(time\_buf), "%Y-%m-%d %H:%M:%S", localtime(&now));

int fd = open(LOG\_FILE, O\_WRONLY | O\_CREAT | O\_APPEND, 0644);

if (fd == -1) {

perror("Failed to open log file");

return;

}

char buf[128];

int len = snprintf(buf, sizeof(buf), "[%s] Received signal: %d (%s)\n",

time\_buf, sig, strsignal(sig));

write(fd, buf, len);

close(fd);

}

void handle\_sighup(int sig) {

if (sigismember(&signals\_to\_log, sig)) {

log\_signal(sig);

}

read\_config();

}

void handle\_sigterm(int sig) {

if (sigismember(&signals\_to\_log, sig)) {

log\_signal(sig);

}

remove(PID\_FILE);

exit(0);

}

void handle\_generic(int sig) {

log\_signal(sig);

struct sigaction sa;

sa.sa\_handler = SIG\_DFL;

sigemptyset(&sa.sa\_mask);

sa.sa\_flags = 0;

sigaction(sig, &sa, NULL);

kill(getpid(), sig);

}

void setup\_signal\_handlers() {

struct sigaction sa;

sa.sa\_flags = 0;

sigemptyset(&sa.sa\_mask);

sa.sa\_handler = handle\_sighup;

sigaction(SIGHUP, &sa, NULL);

sa.sa\_handler = handle\_sigterm;

sigaction(SIGTERM, &sa, NULL);

for (int signo = 1; signo < NSIG; signo++) {

if (signo == SIGHUP || signo == SIGTERM) continue;

if (sigismember(&signals\_to\_log, signo)) {

sa.sa\_handler = handle\_generic;

sigaction(signo, &sa, NULL);

} else {

sa.sa\_handler = SIG\_DFL;

sigaction(signo, &sa, NULL);

}

}

}

void daemonize() {

pid\_t pid = fork();

if (pid < 0) {

perror("fork");

exit(1);

}

if (pid > 0) exit(0);

setsid();

umask(0);

close(STDIN\_FILENO);

close(STDOUT\_FILENO);

close(STDERR\_FILENO);

}

void cleanup\_pid\_file() {

remove(PID\_FILE);

}

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

if (argc == 2 && (strcmp(argv[1], "-q") == 0)) {

FILE \*pid\_fp = fopen(PID\_FILE, "r");

if (!pid\_fp) {

perror("fopen");

exit(1);

}

pid\_t pid;

fscanf(pid\_fp, "%d", &pid);

fclose(pid\_fp);

kill(pid, SIGTERM);

exit(0);

}

daemonize();

FILE \*pid\_fp = fopen(PID\_FILE, "w");

if (!pid\_fp) {

perror("Failed to create PID file");

exit(1);

}

if (pid\_fp) {

fprintf(pid\_fp, "%d", getpid());

fclose(pid\_fp);

atexit(cleanup\_pid\_file);

}

read\_config();

setup\_signal\_handlers();

while (1) {

pause(); // Ожидание сигналов

}

return 0;

}

Листинг А.2 – Makefile

CC=gcc

CFLAGS=-Wall -Wextra

all: lab4

mydaemon: lab4.c

$(CC) $(CFLAGS) -o $@ $<

clean:

rm -f lab4