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

ОТЧЕТ

к лабораторной работе №4

на тему

**УПРАВЛЕНИЕ ПРОЦЕССАМИ И ВЗАИМОДЕЙСТВИЕ ПРОЦЕССОВ.**

Студент А. В. Скворцов

Преподаватель Н. Ю. Гриценко

Минск 2024

СОДЕРЖАНИЕ

[1 Цель работы 3](#_Toc146752068)

[2 Теоретические сведения 4](#_Toc146752069)

[3 Результат выполнения 5](#_Toc146752070)

[Заключение 6](#_Toc146752071)

[Список использованных источников 7](#_Toc146752072)

[Приложение А (обязательное) Листинг кода 8](#_Toc146752073)

1. **ЦЕЛЬ РАБОТЫ**

Изучение основных особенностей подсистемы управления процессами и средств взаимодействия процессов в Unix.

Практическое проектирование, реализация и отладка программных комплексов из нескольких взаимодействующих процессов.

1. **ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Демон Linux – это программа, у которой есть определённая уникальная цель. Обычно, это служебные программы, которые незаметно работают в фоновом режиме для того, чтобы отслеживать состояние и обслуживать определённые подсистемы и гарантировать правильную работу всей операционной системы в целом. Например, демон принтера, отслеживает состояние служб печати, а сетевой демон управляет сетевыми подключениями и следит за их состоянием.

Демоны выполняют определённые действия в запланированное время или в зависимости от определённых событий. В системе Linux работает множество демонов, и каждый из них предназначен для того, чтобы следить за своей небольшой частью операционной системы. Поскольку они не находятся под непосредственным контролем пользователя, они фактически невидимы, но тем не менее необходимы. Поскольку демоны выполняют большую часть своей работы в фоновом режиме, они могут казаться загадочными [1].

Makefile — это файл, который хранится вместе с кодом в репозитории. Его обычно помещают в корень проекта. Он выступает и как документация, и как исполняемый код. Мейкфайл скрывает за собой детали реализации и раскладывает "по полочкам" команды, а утилита make запускает их из того мейкфайла, который находится в текущей директории.

Изначально make предназначалась для автоматизации сборки исполняемых программ и библиотек из исходного кода. Она поставлялась по умолчанию в большинство \*nix дистрибутивов, что и привело к её широкому распространению и повсеместному использованию. Позже оказалось, что данный инструмент удобно использовать и при разработке любых других проектов, потому что процесс в большинстве своём сводится к тем же задачам – автоматизация и сборка приложений.

Компилятор превращает набор кода в объектные файлы, или модули. С помощью линковщика они соединяет все воедино с учетом зависимостей и связей между исходниками. Результат — готовый исполняемый файл [2].

1. **РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ**

В результате работы был разработан процесс-демон, выполняющий функцию приема и протоколирования (записи в файл) заданных сигналов.

Список протоколируемых сигналов задается файлом конфигурации, считывается процессом при запуске и по сигналу SIGHUP.

Сигналы SIGHUP и SIGTERM сохраняют свой эффект для процесса (реконфигурация и завершение соответственно), но также могут и протоколироваться наравне с прочими.

Процесс запускается как обычный исполняемый файл Linux, сам превращается в процесс-демон и работает в фоне, пока пользователь принудительно не завершит процесс (рисунок 1).



Рисунок 1 – Запущенный процесс-демон

В процессе работы процесс полностью протоколирует все действия, относящиеся к нему (рисунок 2).

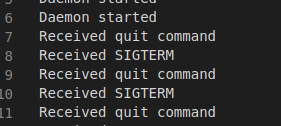


Рисунок 2 – Логирование действий

# ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы был разработан процесс-демон, выполняющий функцию приема и протоколирования (записи в файл) заданных сигналов. Также были изучены основные принципы работы процессов-демонов, история появления такого названия и взаимодействие процессов с помощью сигналов.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Что такое демоны в Linux [Электронный ресурс]. – Режим доступа: https://losst.pro/chto-takoe-demony-v-linux. – Дата доступа: 19.03.2024.
2. Makefile [Электронный ресурс]. – Режим доступа: https://blog.skillfactory.ru/glossary/makefile/. – Дата доступа: 28.02.2024.

# ПРИЛОЖЕНИЕ А (обязательное) Листинг кода

Листинг 1 – daemon.c

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <signal.h>

#include <string.h>

#include <fcntl.h>

#include <sys/stat.h>

#include <sys/types.h>

const char\* CONFIG\_FILE\_PATH = "../config.txt";

const char\* LOG\_FILE\_PATH = "../logfile.txt";

volatile sig\_atomic\_t config\_reload = 0;

void sighup\_handler(int sig) {

config\_reload = 1;

}

void sigterm\_handler(int sig) {

FILE\* log\_file = fopen(LOG\_FILE\_PATH, "a");

if (log\_file != NULL) {

fprintf(log\_file, "Received SIGTERM\n");

fclose(log\_file);

}

exit(0);

}

void daemonize() {

pid\_t pid, sid;

pid = fork();

if (pid < 0) {

exit(1);

}

if (pid > 0) {

exit(0);

}

sid = setsid();

if (sid < 0) {

exit(1);

}

close(STDIN\_FILENO);

close(STDOUT\_FILENO);

close(STDERR\_FILENO);

FILE\* log\_file = fopen(LOG\_FILE\_PATH, "a");

if (log\_file != NULL) {

fprintf(log\_file, "Daemon started\n");

fclose(log\_file);

}

}

void read\_config\_file() {

FILE\* config\_file = fopen(CONFIG\_FILE\_PATH, "r");

if (config\_file == NULL) {

return;

}

char line[256];

while (fgets(line, sizeof(line), config\_file) != NULL) {

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

int sig = atoi(line);

if (signal(sig, SIG\_IGN) == SIG\_ERR) {

// Обработка ошибки установки обработчика сигнала

}

}

fclose(config\_file);

}

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

signal(SIGHUP, sighup\_handler);

signal(SIGTERM, sigterm\_handler);

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

FILE\* log\_file = fopen(LOG\_FILE\_PATH, "a");

if (log\_file != NULL) {

fprintf(log\_file, "Received quit command\n");

fclose(log\_file);

}

kill(getpid(), SIGTERM);

exit(0);

}

daemonize();

while (1) {

if (config\_reload) {

FILE\* log\_file = fopen(LOG\_FILE\_PATH, "a");

if (log\_file != NULL) {

fprintf(log\_file, "Reloaded configuration\n");

fclose(log\_file);

}

config\_reload = 0;

}

sleep(1);

}

return 0;

}