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 В. М. Вергасов

Преподаватель Н. Ю. Гриценко

Минск 2024

СОДЕРЖАНИЕ

[1 Цель работы 3](#_Toc146752068)

[2 Теоретические сведения 4](#_Toc146752069)

[3 Результат выполнения 6](#_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.

Аналог команды kill.

1. **ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**
2. Управление процессами в UNIX основано на концепции процессов, представляющих собой отдельные исполняемые единицы, работающие в операционной системе. Взаимодействие между процессами осуществляется через механизмы межпроцессного взаимодействия, такие как разделяемая память, сигналы, каналы и сокеты [1].
3. Управление памятью в UNIX реализовано с использованием виртуальной памяти, что обеспечивает изоляцию процессов друг от друга и защиту их данных. Планирование процессов осуществляется с использованием механизма приоритета и алгоритма планирования, например, Round Robin.
4. Каждый процесс имеет уникальный идентификатор (PID), состояние и контекст исполнения [2], что позволяет операционной системе управлять и контролировать их работу. Родительский процесс (процесс-предок) определяет начальные условия для нового процесса и передаёт ему информацию о среде выполнения.
5. Для управления процессами в UNIX используются различные команды и утилиты, такие как kill, top, ps и другие. Они позволяют управлять активностью процессов, контролировать использование ресурсов и взаимодействовать с процессами в системе.
6. **РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ**

В результате работы была создана программа, пример результата запуска программы без аргументов представлен на рисунке 1:

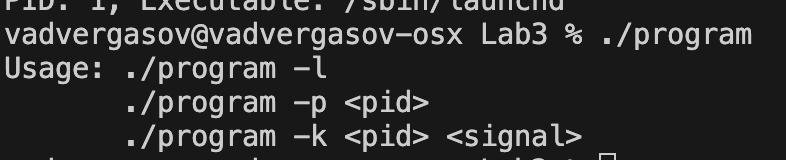


Рисунок 1 – Вывод программы при запуске без аргументов

При указании флага *k* и указания номера процесса и сиггнала, программа попробует отправить сигнал указанному процессу и напишет результат (рисунок 2):

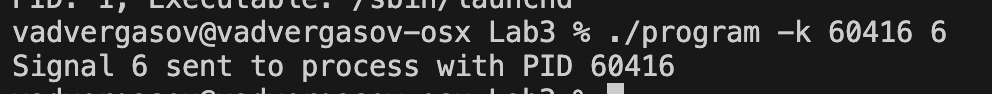


Рисунок 2 – Результат отправки сигнала процессу

# ЗАКЛЮЧЕНИЕ

В ходе выполнения данной лабораторной работы были изучены основные особенности подсистемы управления процессами и средства взаимодействия процессов в операционной системе Unix. Были изучены методы проектирования, реализации и отладки программных комплексов, состоящих из нескольких взаимодействующих процессов.

Было проведено практическое применение полученных знаний и навыков путем разработки аналога команды «*kill*». Реализованный скрипт позволяет завершать процессы, указывая их идентификаторы (*PID*). При запуске скрипта с указанием соответствующего *PID* процесса, данный процесс будет завершен.

Разработанная программа представляет собой полезный инструмент для управления процессами в операционной системе. Он позволяет прерывать выполнение нежелательных или зависших процессов, обеспечивая более эффективное использование системных ресурсов.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Processes [Электронный ресурс]. – https://linux-kernel-labs.github.io/refs/heads/master/lectures/processes.html.
2. Overview of process resources [Электронный ресурс]. –Режим доступа: https://linux-kernel-labs.github.io/refs/heads/master/lectures/processes.html#overview-of-process-resources.

# ПРИЛОЖЕНИЕ А (обязательное) Листинг кода

Листинг 1 – main.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "process\_utils.h"

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

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

list\_processes();

} else if (argc == 3 && strcmp(argv[1], "-p") == 0) {

int pid = atoi(argv[2]);

process\_info(pid);

} else if (argc == 4 && strcmp(argv[1], "-k") == 0) {

int pid = atoi(argv[2]);

int signal = atoi(argv[3]);

send\_signal(pid, signal);

} else {

printf("Usage: ./program -l\n");

printf(" ./program -p <pid>\n");

printf(" ./program -k <pid> <signal>\n");

}

return 0;

}

Листинг 2 – process\_utils.h

#ifndef PROCESS\_UTILS\_H

#define PROCESS\_UTILS\_H

void list\_processes();

void process\_info(int pid);

int send\_signal(int pid, int signal);

#endif

Листинг 3 – process\_utils.c

#include "process\_utils.h"

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <unistd.h>

#include <signal.h>

#include <libproc.h>

void list\_processes() {

pid\_t pids[10240];

int count = proc\_listallpids(pids, sizeof(pids));

if (count <= 0) {

printf("Failed to get process list\n");

return;

}

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

char path[PROC\_PIDPATHINFO\_MAXSIZE];

int ret = proc\_pidpath(pids[i], path, sizeof(path));

if (ret > 0) {

printf("PID: %d, Executable: %s\n", pids[i], path);

}

}

}

void process\_info(int pid) {

char path[PROC\_PIDPATHINFO\_MAXSIZE];

int ret = proc\_pidpath(pid, path, sizeof(path));

if (ret > 0) {

printf("PID: %d, Executable: %s\n", pid, path);

} else {

printf("Process with PID %d not found\n", pid);

}

}

int send\_signal(int pid, int signal) {

int result = kill(pid, signal);

if (result == 0) {

printf("Signal %d sent to process with PID %d\n", signal, pid);

} else {

printf("Failed to send signal %d to process with PID %d\n", signal,

pid);

}

return result;

}

Листинг 4 – Makefile

CC = gcc

CFLAGS = -Wall -Wextra

SRCS = main.c process\_utils.c

OBJS = $(SRCS:.c=.o)

TARGET = program

.PHONY: all clean

all: $(TARGET)

$(TARGET): $(OBJS)

$(CC) $(CFLAGS) $(LDFLAGS) -o $@ $^

%.o: %.c

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

clean:

rm -f $(OBJS) $(TARGET)