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 №3

на тему

**ОСНОВЫ ПРОГРАММИРОВАНИЯ НА C ПОД UNIX. ИНСТРУМЕНТАРИЙ ПРОГРАММИСТА В UNIX**

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 среды программирования и основных инструментов: компилятор/сборщик («коллекция компиляторов») *gcc*, управление обработкой проекта *make* (и язык *makefile*), библиотеки и т.д.

Практическое использование основных библиотек и системных вызовов: ввод-вывод и работа с файлами, обработка текста, распределение памяти, управление выполнением и т.п.

Упрощённый аналог *ps*. Запуск программы с аргументом *-l* выводит список процессов (*pid* и имя исполняемого файла).

Запуск с аргументом *-p <pid>* выводит дополнительную информацию о заданном процессе.

1. **ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

Философия системы UNIX основана на многократном использовании имеющихся программ и идеях: данные, программы, каталоги и физические устройства рассматриваются как файлы разных типов, а файл понимается как одномерный массив байт без другой структуры [1].

Инструменты ОС UNIX включают:

1. Файловую систему без древовидной структуры, которая позже стала системой каталогов и файлов;
2. Команды для различных действий с файлами и ассемблер для генерации исполняемых программ;
3. Межпроцессные каналы для связи между процессами;

Три вида программистов в UNIX:

1. Программисты-одиночки, которые работают над своими проектами и не зависят от других.
2. Прикладные программисты, которые разрабатывают программное обеспечение для конкретных задач и пользователей.
3. Системные программисты, которые занимаются разработкой и поддержкой ядра операционной системы и системных компонентов.

В лабораторной работе вам предстоит изучить основы программирования на языке C в UNIX, познакомиться с инструментарием программиста и освоить базовые навыки работы с файловой системой, командами и процессами.

Формат Makefile состоит из трёх основных компонентов [2]:

1. Правила: описывают, как создавать другие файлы (например, .o из .c) и как собирать программу из отдельных файлов.
2. Зависимости: указывают, из каких файлов состоит программа и как эти файлы собираются.
3. Определения переменных: позволяют Makefile быть более гибкими и адаптивными.
4. **РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ**

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

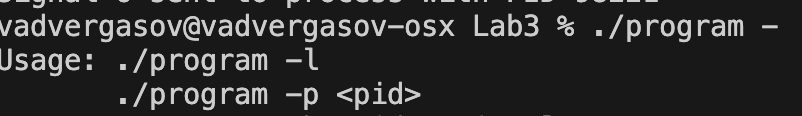


Рисунок 1 – Вывод программы при запуске без аргументов

При указании флага *l*, программа выводит список процессов (рисунок 2):



Рисунок 2 – Список процессов

# ЗАКЛЮЧЕНИЕ

В ходе выполнения данной лабораторной работы были изучены основные методы и инструменты среды программирования, включая компилятор/сборщик gcc, систему управления проектами make и язык makefile, а также различные библиотеки и системные вызовы.

В результате лабораторной работы был разработан упрощенный аналог команды «*ps*». Реализованный скрипт позволяет выводить список запущенных процессов, включая их идентификаторы (*PID*) и имена исполняемых файлов. При запуске скрипта с аргументом «*-l*» выводится основная информация о процессах, а при использовании аргумента «*-p <PID>»* также выводится дополнительная информация о конкретном процессе.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. GNU/Linux Command−Line Tools Summary [Электронный ресурс]. – Режим доступа: http://ploug.eu.org/doc/GNU-Linux-Tools-Summary.pdf.
2. GNU make [Электронный ресурс]. – Режим доступа: https://www.gnu.org/software/make/manual/make.html.

# 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)