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 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 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.

1. **ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

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 [1].

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.

Утилита make автоматизирует процесс — разработчику достаточно набрать одну команду в консоли. Она преобразует исходный код в модули, а их – в исполняемые файлы, которые можно запустить. Так создаются библиотеки и разнообразные программные продукты. Makefile показывает программе правила, по которым нужно выполнять преобразования.

Make работает на Linux и системах на базе Unix — для этих ОС утилита считается основным средством сборки программ. В Windows тоже есть концепция Makefile, но управляет этими файлами утилита nmake. У make две версии: для платформы BSD и для GNU. Первая используется в операционных системах FreeBSD, OpenBSD и NetBSD, вторая — в MacOS и [Linux](https://blog.skillfactory.ru/glossary/linux/). Утилита обычно есть в ОС по умолчанию [2].

1. **РЕЗУЛЬТАТ ВЫПОЛНЕНИЯ**

В результате работы был разработан упрощенный аналог программы ps на Linux. Аналог позволяет просматривать список всех процессов и показывать информацию о конкретном процессе. Программа выполнена в виде двух модулей на языке C, один модуль – основной, в котором программа обрабатывает свое использование и входные параметры, второй – модуль логики самой программы, содержащий нужные для получения списка процессов и информации о них функции.

Прежде, чем использовать скрипт, его необходимо собрать в исполняемый файл, для этого используется команда make (рисунок 1).

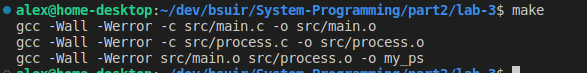


Рисунок 1 – Сборка исполняемого файла

Так же в Makefile присутствует возможность тестирования получившегося исполняемого файла. Для использования скрипта и получения списка процессов достаточно вызвать его с параметром -l (рисунок 2).

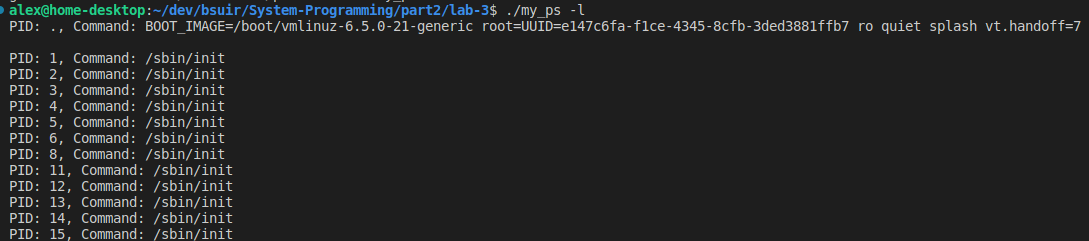


Рисунок 2 – Вывод списка процессов

Для получения информации о конкретном процессе, нужно вызвать скрипт с параметром -p <pid> (рисунок 3).

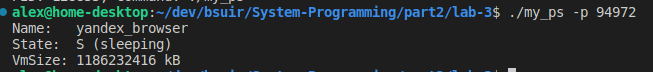


Рисунок 3 – Вывод информации о процессе

# ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы был разработан аналог программы ps, позволяющий получать список всех запущенных процессов системы, а также получать о них информацию такую как статус процесса и количество используемой памяти.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Что такое Makefile [Электронный ресурс]. – Режим доступа: https://guides.hexlet.io/ru/makefile-as-task-runner/. – Дата доступа: 28.02.2024.
2. Makefile [Электронный ресурс]. – Режим доступа: https://blog.skillfactory.ru/glossary/makefile/. – Дата доступа: 28.02.2024.

# ПРИЛОЖЕНИЕ А (обязательное) Листинг кода

Листинг 1 – process.c

#include <stdio.h>

#include <dirent.h>

#include <stdlib.h>

#include <string.h>

#include "process.h"

void list\_processes() {

DIR \*dir = opendir("/proc");

if (dir == NULL) {

perror("opendir");

exit(1);

}

struct dirent \*entry;

while ((entry = readdir(dir)) != NULL) {

if (entry->d\_type == DT\_DIR) {

char \*path = malloc(strlen("/proc/") + strlen(entry->d\_name) + strlen("/cmdline") + 1);

if (path == NULL) {

perror("malloc");

exit(1);

}

snprintf(path, strlen("/proc/") + strlen(entry->d\_name) + strlen("/cmdline") + 1, "/proc/%s/cmdline", entry->d\_name);

FILE \*file = fopen(path, "r");

if (file != NULL) {

char cmdline[256];

fgets(cmdline, sizeof(cmdline), file);

fclose(file);

if (strlen(cmdline) > 0) {

printf("PID: %s, Command: %s\n", entry->d\_name, cmdline);

}

}

}

}

closedir(dir);

}

void print\_process\_info(int pid) {

char path[256];

snprintf(path, sizeof(path), "/proc/%d/status", pid);

FILE \*file = fopen(path, "r");

if (file != NULL) {

char line[256];

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

if (strncmp(line, "Name:", 5) == 0 || strncmp(line, "State:", 6) == 0 || strncmp(line, "VmSize:", 7) == 0) {

printf("%s", line);

}

}

fclose(file);

} else {

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

}

}

Листинг 2 – main.c

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include "process.h"

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

if (argc < 2) {

printf("Usage: %s [-l | -p <pid>]\n", argv[0]);

return 1;

}

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

list\_processes();

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

if (argc < 3) {

printf("Usage: %s -p <pid>\n", argv[0]);

return 1;

}

int pid = atoi(argv[2]);

print\_process\_info(pid);

} else {

printf("Invalid argument: %s\n", argv[1]);

return 1;

}

return 0;

}

Листинг 3 – Makefile

CC = gcc

CFLAGS = -Wall -Werror

SRCS = src/main.c src/process.c

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

EXECUTABLE = my\_ps

.PHONY: all clean test

all: $(EXECUTABLE)

$(EXECUTABLE): $(OBJS)

$(CC) $(CFLAGS) $(OBJS) -o $(EXECUTABLE)

%.o: %.c

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

test: $(EXECUTABLE)

./$(EXECUTABLE) -l

clean:

rm -f $(OBJS) $(EXECUTABLE)