Министерство цифрового развития

Сибирский Государственный Университет Телекоммуникаций и

Информатики

СибГУТИ

Кафедра прикладной математики и кибернетики

Лабораторная работа №1

по дисциплине: Операционные системы

Выполнил: Гринченко А. В.

Группа: ЗП-022

Номер зачетки: 73210117

Проверил: проф. Малков Е. А.

Новосибирск, 2023

# Оглавление

[Оглавление 2](#_Toc1)

[Постановка задачи 3](#_Toc2)

[Инструментарий 4](#_Toc3)

[Разработка 5](#_Toc4)

[Заключение 9](#_Toc5)

# **Постановка задачи**

**Цель работы:** получение навыков использования функций API создания процессов на платформе Linux.

**Задание:** Разработать приложение, запускающее несколько программ. Определить идентификаторы соответствующих процессов. Установить родственные связи между ними.

# Инструментарий

В качестве инструментов для разработки приложения использованы текстовый редактор, компилятор gcc, терминал для команд сборки и запуска исполняемого файла.

# Разработка

Листинг программы:

#include <stdio.h>

#include <stdlib.h>

#include <ctype.h>

#include <string.h>

#include <sys/types.h>

#include <unistd.h>

#include <dirent.h>

#define READ\_BUF\_SIZE 512

#define READ\_PPID 6

int spawn(char\* program, char\*\* arg\_list)

{

pid\_t child\_pid;

child\_pid = fork();

if (child\_pid != 0) { return child\_pid; }

else

{

execvp(program, arg\_list);

fprintf(stderr, "an error occured in execvp\n");

abort();

}

}

void get\_pid\_and\_ppid(const char\* pname)

{

char filename\_status[READ\_BUF\_SIZE];

char buffer[READ\_BUF\_SIZE];

char name[READ\_BUF\_SIZE];

char ppid[READ\_PPID];

FILE\* status;

DIR \*dir;

struct dirent \*next;

dir = opendir("/proc");

if (!dir)

{

perror("Cannot open /proc");

exit(-1);

}

printf("Name: %s\t", pname);

while ((next = readdir(dir)) != NULL)

{

if (strcmp(next -> d\_name, "..") == 0) { continue; }

if (!isdigit(\*next -> d\_name)) { continue; }

sprintf(filename\_status, "/proc/%s/status", next -> d\_name);

status = fopen(filename\_status, "r");

fgets(buffer, READ\_BUF\_SIZE - 1, status);

sscanf(buffer, "%\*s %s", name);

if (strcmp(pname, name) != 0) { continue; }

else

{

printf("PID: %s\t", next -> d\_name);

while (fgets(buffer, READ\_BUF\_SIZE - 1, status) != NULL)

{

if (strstr(buffer, "PPid") != NULL)

{

sscanf(buffer, "%\*s %s", name);

sscanf(buffer, "%\*s %s", ppid);

printf("PPID: %s\t", name);

break;

}

}

}

}

fclose(status);

sprintf(filename\_status, "/proc/%s/status", ppid);

status = fopen(filename\_status, "r");

fgets(buffer, READ\_BUF\_SIZE - 1, status);

sscanf(buffer, "%\*s %s", name);

printf("Name of parent process: %s", name);

printf("\n");

}

int main()

{

char\* arg\_list1[] = {"atom", NULL};

spawn("atom", arg\_list2);

get\_pid\_and\_ppid("atom");

char\* arg\_list2[] = {"emacs", NULL};

spawn("emacs", arg\_list3);

get\_pid\_and\_ppid("emacs");

printf("\ndone with main program\n");

return 0;

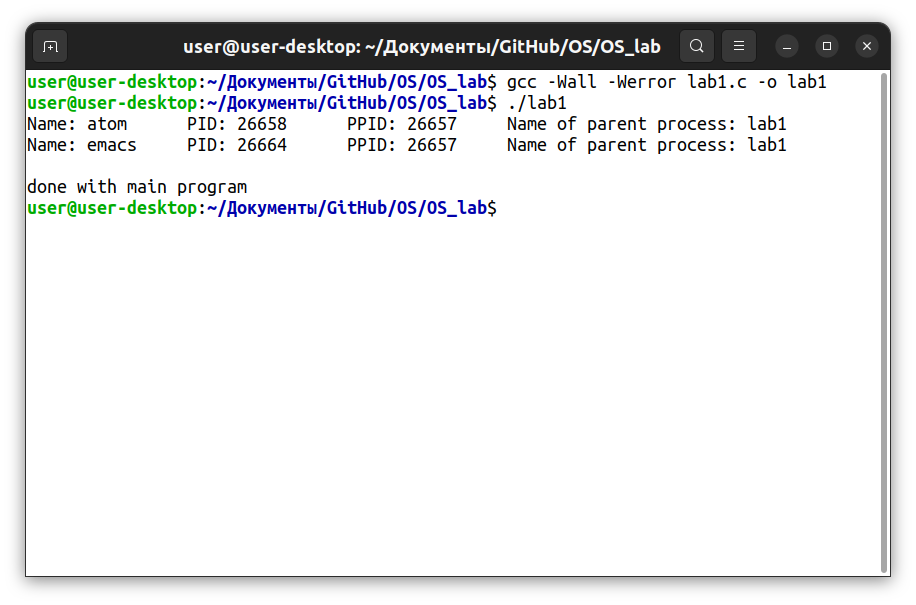
}

Функция **main()** программы определяет два массива аргументов для функции int spawn(char\* program, char\*\* arg\_list), которая инициирует запуск одной программы. В качестве примеров запускаемых программ взяты текстовые редакторы Atom и Emacs. После завершения работы функции spawn вызывается функция void get\_pid\_and\_ppid(const char\* pname), которая по имени процесса находит его PID и PPID и позволяет установить родственные связи между запущенными процессами.

Функция **spawn(char\* program, char\*\* arg\_list)** принимает два аргумента: имя запускаемой программы и аргументы, необходимые для работы функции execvp. В начале работы функции spawn объявляется переменная child\_pid типа pid\_t для выполнения функции fork(). Вызывая fork(), программа создает дочерний процесс. fork() возвращает разные значения в родительском и дочернем процессах: родительский процесс получает идентификатор своего потомка, дочерний - 0. Функция execvp заменяет программу, выполняющуюся в текущем процессе, другой программой, поэтому, для продолжения выполнения главной программы в родительском процессе функция execvp выполняется в дочернем процессе, child\_pid которого равен 0. При нормальной работе программы execvp никогда не возвращает значение, в случае, если ошибка произошла, в поток stderr выводится сообщение об ошибке и программа аварийно завершает работу.

Функция **get\_pid\_and\_ppid(const char\* pname)** выводит сообщения о PID и PPID запущенного процесса. Для этого считывается информация из директории /proc, а именно из файла /proc/<имя процесса>/status. Перебирая все процессы, функция находит совпадение с запущенным, после чего выводит сначала его PID, затем – PPID и имя процесса с идентификатором, равным PPID (родительского процесса). get\_pid\_and\_ppid запускается отдельно для каждого процесса и позволяет вывести данные для всех запущенных в программе процессов.

Результат работы программы:



Как видно из скриншота, оба процесса имеют разные PID и одинаковые PPID, имя родительского процесса – lab1. Таким образом, видно, что обе программы запущены из одной и той же программы – лабораторной работы.

# Заключение

В процессе выполнения лабораторной работы были получены навыки использования функций создания процессов на платформе Linux.

В результате выполнения лабораторной работы было разработано приложение, запускающее несколько программ, определяющее идентификаторы соответствующих процессов и демонстрирующее родственные связи между ними.