|  |  |  |
| --- | --- | --- |
| **ИВТиПТ**  **Кафедра**  **ПИиИ**  **Группа 3.6** | **Дисциплина: Операционные системы** | **Лабора­тор­ная работа №2** |
| **ФИО студента: Остапец Павел Игоревич** |

**Цели и задачи работы**

Основная задача данной лабораторной работы — работа с процессами в ОС. Также во время выполнения работы студент ознакомится с системными вызовами, необходимым для управления процессами. А именно:

• написать программу lab2\_1.c на языке C, выводящую информацию о пользователе и группе пользователя процесса, а также номер процесса и номер вышестоящего процесса,

• написать программу lab2\_2.c на языке C, создающей новый процесс из текущего и выводящую номер процесса и номер вышестоящего процесса из обоих процессов,

• написать программу lab2\_3.c на языке С, выполняющую другую программу в рамках текущего процесса,

• написать отчет о выполненной работе с фиксацией результата,

• ответить на контрольные вопросы.

**Лабораторное оборудование**

Лабораторным сервером для выполнения заданий представляет собой удаленный сервер под управлением ОС Linux. Для подключения к лабораторному оборудованию необходимо использовать любое ПО для подключения к серверу по SSH.

**Краткая теория**

Понятие процесса есть одна из ключевых абстракций современных операционных систем. Процесс — это совокупность программного задания, его оперативных данных и контекста исполнения. Операционная система управляет процессами. Для управления ОС ведет таблицу процессов. За каждым вновь созданным процессом закрепляется его номер, и заполняется небольшая структура данных PCB (process control block) в которой хранятся вся информация по процессу и его контекст (см лекцию). В частности, там хранится:

• PID и процесса и идентификатор PPID процесса родителя

• UID,GID идентификатор пользователя и группы пользователя, для которого был создан процесс.

Узнать номер текущего сеанса в консоли можно командой:

echo $$

Список процессов можно увидеть при помощи команд ps, pstree или top.

В этой работе будут исследованы следующие системные вызовы:

• getuid/getgid — получить ID пользователя и группы текущего процесса,

• getpid/getppid — получить ID текущего и родительского процесса,

• fork — создать новы процесс как копию текущего (за исключением PID и PPID),

• exec — заместить в PCB контекст текущего процесса новым исполняемым файлом,

• \_exit — завершить текущий процесс с кодом возврата.

Обратите внимание, что функции fork(), exec() и exit() работают несколько необычно.

API:

#include <unistd.h>

#include <sys/types.h>

uid\_t getuid(void);

gid\_t getgid(void);

pid\_t getpid(void);

pid\_t getppid(void);

pid\_t fork(void);

void \_exit(int status);

extern char \*\*environ;

int execl(const char \*path, const char \*arg, … /\* (char \*) NULL \*/);

int execlp(const char \*file, const char \*arg, … /\* (char \*) NULL \*/);

int execle(const char \*path, const char \*arg, … /\*, (char \*) NULL, char \* const envp[] \*/);

int execv(const char \*path, char \*const argv[]);

int execvp(const char \*file, char \*const argv[]);

int execve(const char \*file, char \*const argv[], char \*const envp[]);

**Ход работы**

1. Предварительно подключившись к удаленному серверу с помощью PuTTY с использованием своего логина и пароля, следует создать нужную директиву и файлы для выполнения заданий. В данном пункте используются знания, полученные в прошлой лабораторной работе.

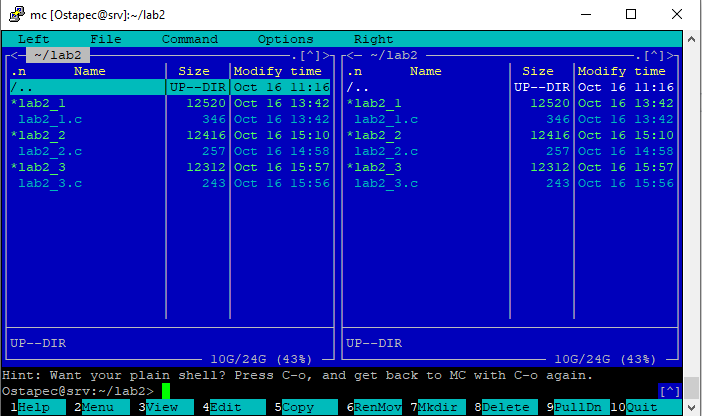


Рисунок 1. Директива и файлы для Лаб. 2

1. Создав файл lab2\_1 для выполнения второго задания по заданному шаблону,

добавляем код, который обеспечивает вывод системных вызовов getuid(), getgid(), getpid(), getppid(), которые предоставляют информацию о процессе. Далее программа компилируется:

**gcc lab2\_1.c -o lab2\_1**

**chmod a+x lab2\_1**

**./lab2\_1**

Далее выполняются команды **echo $?**, **echo $$.** Данные команды возвращают соответственно код возврата процесса и номер текущего процесса.Чтобы код возврата соответствовал программе, добавим системный вызов **exit()**, дописав в начало программы **#include <stdlib.h>** для его корректной работы.

Из вывода видно, что благодаря **exit(5);** код возврата процесса соответствует указанному в программе, а номер текущего процесса совпадает с номером родительского процесса только что выполненной программы.

Это говорит о том, что выполнив программу мы возвращаемся в тот же процесс, что и перед выполнением.

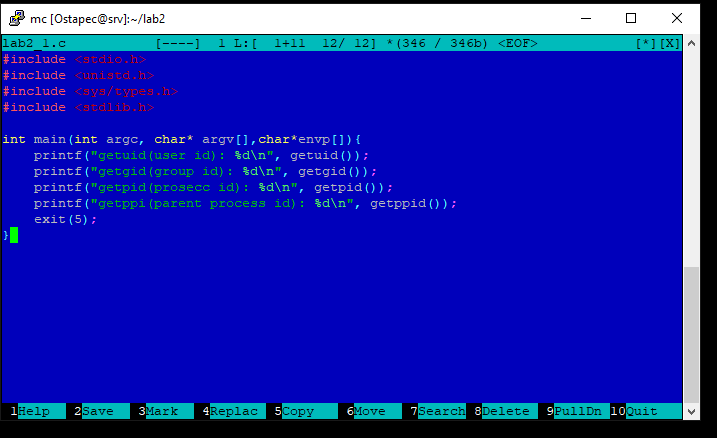


Рисунок 2. Код программы lab2\_1.c

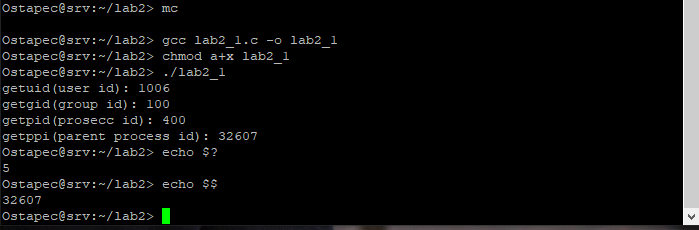


Рисунок 3. Вывод программы lab2\_1

1. В следующей программе **lab2\_2** с использованием системного вызова **fork()** буден порожден новый процесс.

Далее программа компилируется:

**gcc lab2\_2.c -o lab2\_2**

**chmod a+x lab2\_2**

**./lab2\_2**

В этом задании порождается новый процесс с помощью **fork().** В строках вывода мы видим **pid** текущего процесса и его **ppid**. Далее выводятся **pid** и **ppid** порожденного с помощью **fork()** процесса. Можно заметить, что **pid** нового процесса отличается от родительского на единицу, а значение его **ppid** совпадает со значением **pid** предыдущего процесса, что подтверждает наследственность. Такой «корректный» вывод данных обеспечивается благодаря методу **wait(),** который не дает системе успеть подменить родительский процесс процессом **init**, а без вызова **wait() ppid** рожденного с помощью **fork()** процесса был бы равен единице.

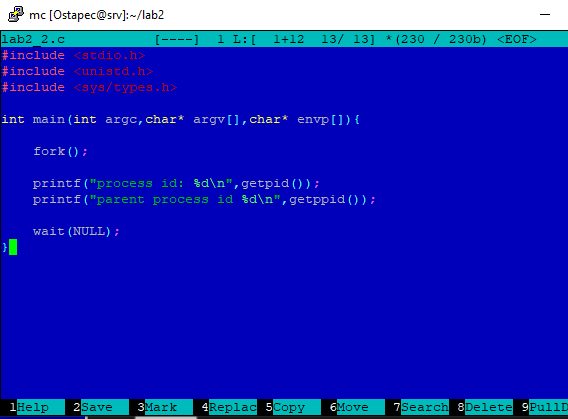


Рисунок 4. Код программы lab2\_2

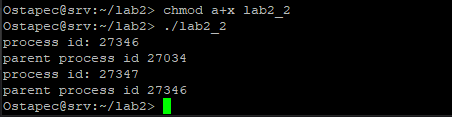


Рисунок 5. Вывод программы lab2\_2

1. В последнем задании требуется вызвать команду os-release в программе lab2\_3, которая выводит на экран информацию о релизе дистрибутива с помощью одного из системных вызовов: **execl(), execlp(), execle(), execv(), execvp(), execve()** и запустить программу на выполнение:

**gcc lab2\_3.c -o lab2\_3**

**chmod a+x lab2\_3**

**./lab2\_3**

Вызов execl происходит таким образом, что программа загружается в память вместо старой, которая вызывала execl. Старой программе с этого момента больше не будут доступны сегменты памяти, которые перезаписаны новой программой, поэтому и не срабатывает вывод **“done”.** Чтобы “**done**” все-таки было выведено, нужно чтобы **execl** передал управление основной программе обратно, а это происходит только в случае возникновении ошибки в самом **execl** (неправильно задан путь и тд.)

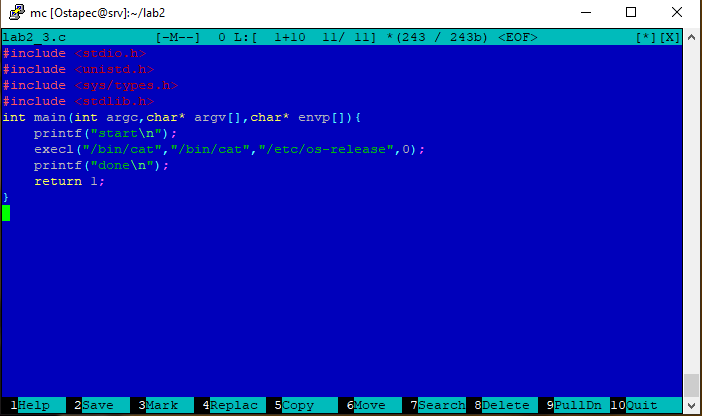


Рисунок 6. Код программы lab2\_3.

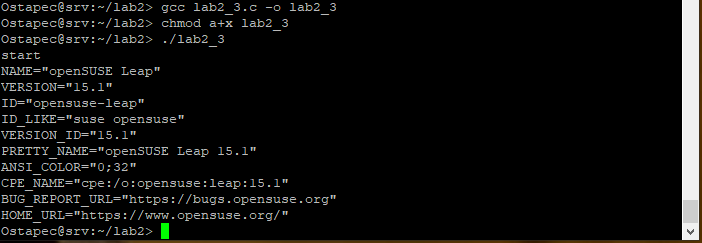


Рисунок 7. Вывод программы lab2\_3.

**Результаты работы.**

В результате работы было выполнено 3 задания, связанных с работой процессов в ОС, системными вызовами, необходимыми для управления процессами, которые проиллюстрированы выше.

**Литература**

1. В.Е. Карпов, К.А. Коньков «Основы операционных систем Курс лекций» издание 3-е, М.: Физматкнига, 2019

2. В.Е. Карпов, К.А. Коньков «Основы операционных систем Практикум» http://www.intuit.ru/studies/courses/2249/52/info