федеральное государственное бюджетное образовательное учреждение высшего образования

**«Вологодский государственный университет»**

Институт математики, естественных и компьютерных наук

(наименование института)

Кафедра «Автоматики и вычислительной техники»

(наименование кафедры)

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №7**

|  |  |
| --- | --- |
| Дисциплина | Операционные системы |

|  |  |  |
| --- | --- | --- |
| Преподаватель | доц.Дианов С.В. | |
|  | *(уч. степень, звание, должность. Ф.И.О)* | |
| Выполнил студент | Попов Кирилл Михайлович | |
|  | *(Ф.И.О)* | |
| Группа, курс | 4Б09 РПС-31 | |
| Оценка по защите |  | |
|  | | *(подпись преподавателя)* |

Вологда

2022 г.

**Работа с процессами в операционной системе Linux с использованием С-программ**

**Цель работы:** получение навыков создания С-программ и их использования для работы с процессами в операционной системе Linux

**Задание 1**

**Компиляция и запуск C-программ**

Для компиляции программ в Linux мы будем применять компилятор gcc. Для того чтобы он нормально работал, необходимо, чтобы исходные файлы, содержащие текст программы, имели имена, заканчивающиеся на .c.

В простейшем случае откомпилировать программу можно, запуская компилятор командой gcc имя\_исходного\_файла.

Если программа была написана без ошибок, то компилятор создаст исполняемый файл с именем имя\_исходного\_файла.out. Изменить имя создаваемого исполняемого файла можно, задав его с помощью опции -o:

gcc имя\_исходного\_файла -o имя\_исполняемого\_файла

Компилятор gcc имеет несколько сотен возможных опций. Получить информацию о них вы можете в UNIXManual.

Обычно во всех версиях Linux имеется компилятор с именем cc, поддерживающий опцию –о.

Запустить программу на исполнение можно, набрав имя исполняемого файла и нажав клавишу <Enter>.

**Самостоятельная работа**

1. Перейти во вторую текстовую консоль.

<alt>+F2

2. Запустить редактор Vim

3. Создать файл, содержащий следующие строки:

#include <stdio.h> int main()

{

printf(“Добро пожаловать!\n”); Return 0; }

4. Сохраняем файл под именем quine.c

:w quine.c



5. Переключаемся на другую консоль.

<alt>+F3

6. Компилируем программу.

gcc quine.c -o binfile



7. Запускаем скомпилированный файл (не забываем прописать путь целиком, даже несмотря на то, что файл в текущей папке).

./binfile

Должна появиться строка «Добро пожаловать!»



**Задание 2**

**Процессы в операционной системе Linux**

Процесс (process) — блок адресного пространства, в котором выполняются одна или более нитей, экземпляр выполняемой программы. Любой процесс может запускать другие процессы. Таким образом, процессы в среде UNIX образуют иерархическую структуру. На вершине этой структуры находится процесс init, являющийся предком всех остальных процессов.

**Команды для работы с процессами**

**ps [-axewjlu] [-o формат] [-U пользователь] [-ppid]**

Выводит список и статус процессов, работающих в системе. Без аргументов выводит список процессов текущего пользователя, подключенных к терминалу.

Значения параметров следующие:

- a - вывести информацию о процессах всех пользователей.

- x - вывести информацию о процессах, не подключенных к терминалу.

- e - вывести значения переменных окружения процесса

-w - использовать строки длиной 132 символа. Если указан несколько раз, то строки не обрезаются совсем.

- j, -l, -u - меняют формат вывода информации.

- o формат - вывести информацию в указанном формате.

- U пользователь - вывести информацию о процессах указанного пользователя.

- ppid - вывести информацию о процессе с указанным идентификатором.

Значение формата для параметра -o является списком из следующих ключевых слов, разделенных запятыми (без пробелов):

- Command - командная строка и аргументы.

- nice - уровень nice (приоритет).

- pgid - идентификатор группы процессов.

- pid - идентификатор процесса.

- ppid - идентификатор родительского процесса.

- rgid, - ruid - реальные идентификаторы группы и пользователя.

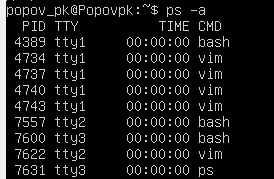
- uid - реальный идентификатор пользователя.

- tty - управляющий терминал

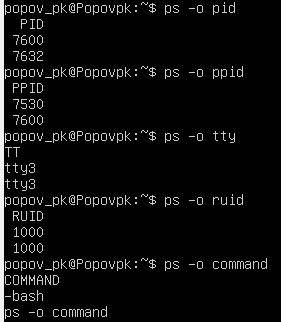
- top - выводит сведения о запущенных процессах в динамике. Прекратить работукоманды можно нажав клавишу q

**Самостоятельная работа**

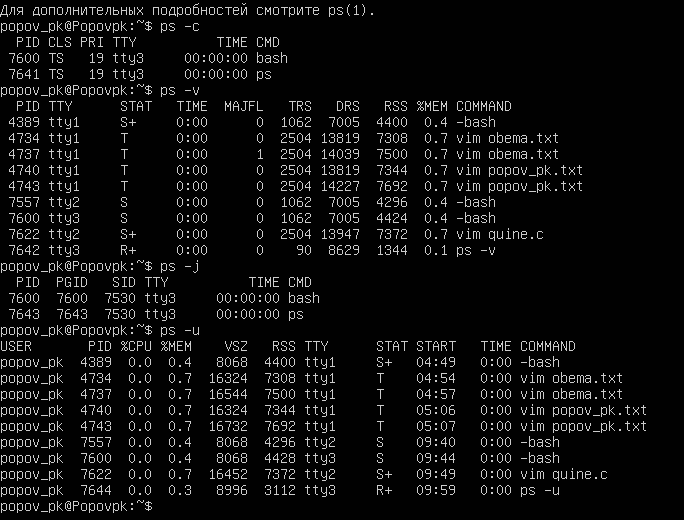
1. Освойте работу с командой ps. Попробуйте запускать ее с различными аргументами. Если вывод команды не помещается на экране, используйте команду less.



1. Запишите в отчет следующую информацию о запущенных Вами процессах: pid, ppid, tty, ruid, command. Вывод должен быть отсортирован по номеру процесса.



1. Ключи -c, -v, -j, -u изменяют формат вывода команды. Попробуйте выполнить команду ps с каждым из этих ключей. Результат запишите в отчет.



**Задание 3**

**Создание процесса в Linux. Системный вызов fork()**

В операционной системе UNIX новый процесс может быть порожден единственным способом – с помощью системного вызова fork(). При этом вновь созданный процесс будет являться практически полной копией родительского процесса. У порожденного процесса по сравнению с родительским процессом (на уровне уже полученных знаний) изменяются значения следующих параметров:

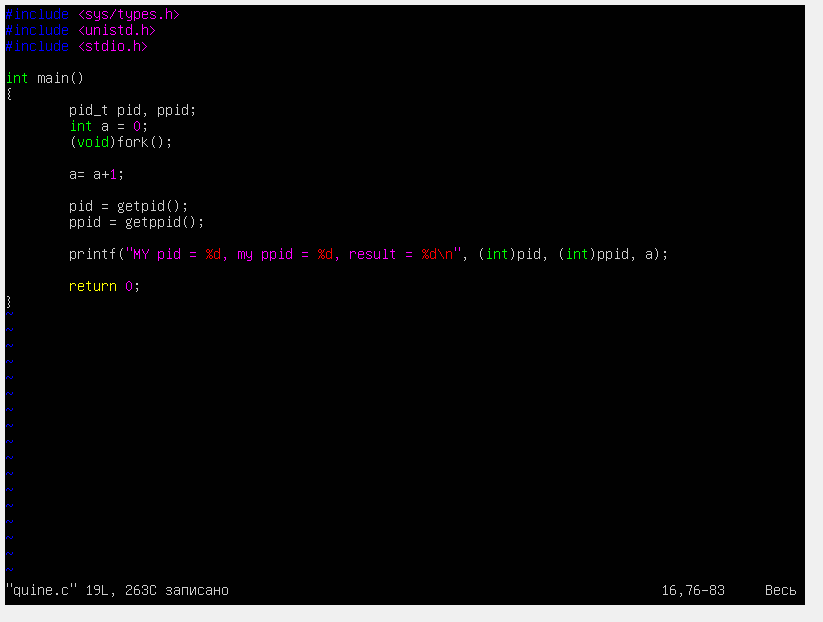
· идентификатор процесса – PID;

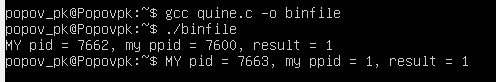
· идентификатор родительского процесса – PPID.

В процессе выполнения системного вызова fork() порождается копия родительского процесса и возвращение из системного вызова будет происходить уже как в родительском, так и в порожденном процессах. Этот системный вызов является единственным, который вызывается один раз, а при успешной работе возвращается два раза (один раз в процессе-родителе и один раз в процессе-ребенке)! После выхода из системного вызова оба процесса продолжают выполнение регулярного пользовательского кода, следующего за системным вызовом.

**Прогон программы с fork() с одинаковой работой родителя и ребенка**

Наберите эту программу, откомпилируйте ее и запустите на исполнение. Проанализируйте полученный результат.





Изменение пользовательского контекста процесса. Семейство функций для системного вызова exec(). Для изменения пользовательского контекста процесса применяется системный вызов exec(), который пользователь не может вызвать непосредственно. Вызов exec() заменяет пользовательский контекст текущего процесса на содержимое некоторого исполняемого файла и

устанавливает начальные значения регистров процессора (в том числе устанавливает программный счетчик на начало загружаемой программы). Этот вызов требует для своей работы задания имени исполняемого файла, аргументов командной строки и параметров окружающей среды. Для осуществления вызова программист может воспользоваться одной из шести функций:

execlp(), execvp(), execl() и execv(), execle(), execve(), отличающихся друг от друга представлением параметров, необходимых для работы системного вызова exec().

**Прогон программы с использованием системного вызова exec().**

Для иллюстрации использования системного вызова exec() давайте рассмотрим следующую программу:



Откомпилируйте ее и запустите на исполнение. Поскольку при нормальной работе будет распечатываться содержимое файла с именем 03-2.c, такой файл при запуске должен присутствовать в текущей директории (проще всего записать исходный текст программы под этим именем). Проанализируйте результат.

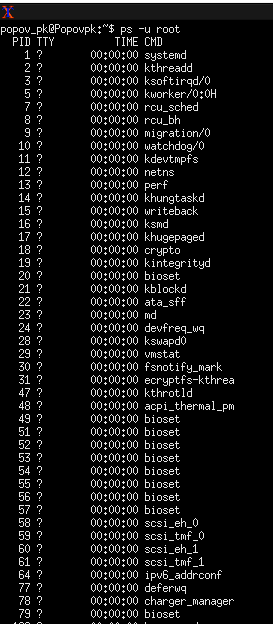
**Задание 4**

**Команды для управления процессами.**

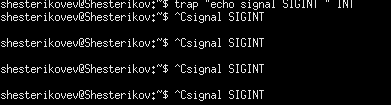
Linux имеет большое число механизмов межпроцессного взаимодействия. Наиболее популярными средствами являются сигналы, программные каналы (pipes) и именованные каналы (FIFO).

**Самостоятельная работа**

Запишите в отчет информацию о процессах, запущенных пользователем root. Вывод должен быть отсортирован по номеру процесса.



Установите обработчик сигнала SIGINT. Для этого выполните команду: trap "echo получен сигнал SIGINT " INT. Пошлите несколько раз оболочке сигнал SIGINT. Для этого следует нажать клавиши CTRL-C



**Вывод:** Мы получили навыки создания С-программ и их использования для работы с процессами в операционной системе Linux.