**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «ЛИПЕЦКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Институт компьютерных наук

Кафедра автоматизированных систем управления

Лабораторная работа №2

по операционным системам

Студент АС-21-1 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Станиславчук С. М.

(подпись, дата)

Руководитель

Доцент, к.п.н. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Кургасов В. В.

(подпись, дата)

Липецк 2024

Задание кафедры

Напишите программу для мониторинга запущенных процессов с применением модульной структуры. В отдельные модули вывести решение нижеследующих следующих задач.

1. Посчитать количество процессов, запущенных пользователем user, и вывести в файл пары PID:команда для таких процессов.
2. Вывести на экран PID процесса, запущенного последним (с последним временем запуска).
3. Вывести в файл список PID всех процессов, которые были запущены командами, расположенными в /sbin/
4. Для каждого процесса посчитать разность резидентной и разделяемой части памяти процесса (в страницах). Вывести в файл строки вида PID:разность, отсортированные по убыванию этой разности.
5. Для всех зарегистрированных в данный момент в системе процессов выведите в один файл строки ProcessID=PID : Parent\_ProcessID=PPID : Average\_Time=avg\_atom. Значения PPid и Pid возьмите из файлов status, значение avg\_atom из файлов sched, которые находятся в директориях с названиями, соответствующими PID процессов в /proc. Отсортируйте эти строки по идентификаторам родительских процессов.
6. В полученном на предыдущем шаге файле после каждой группы записей с одинаковым идентификатором родительского процесса вставить строку вида Average\_Sleeping\_Children\_of\_ParentID=N is M, где N = PPID, а M – среднее, посчитанное из SleepAVG для данного процесса.

Цель работы

Изучение процессов в ОС GNU/Linux.

Ход работы

Для выполнения данной лабораторной работы создадим скрипты, которые выполняют предложенные задания.

Для выполнения первого задания достаточно выполнить следующие команды:

ps -u stanik -o pid,cmd > 1.txt

ps -u stanik --no-headers | wc -l >> 1.txt

Первая команда выведет пары PID:команда в файл 1.txt, а вторая – выведет общее число данных пар и запишет в конце файла 1.txt.

Для создания скрипта создадим файл 1.sh командой touch 1.sh. Результат выполнения представлен на рисунке 1.

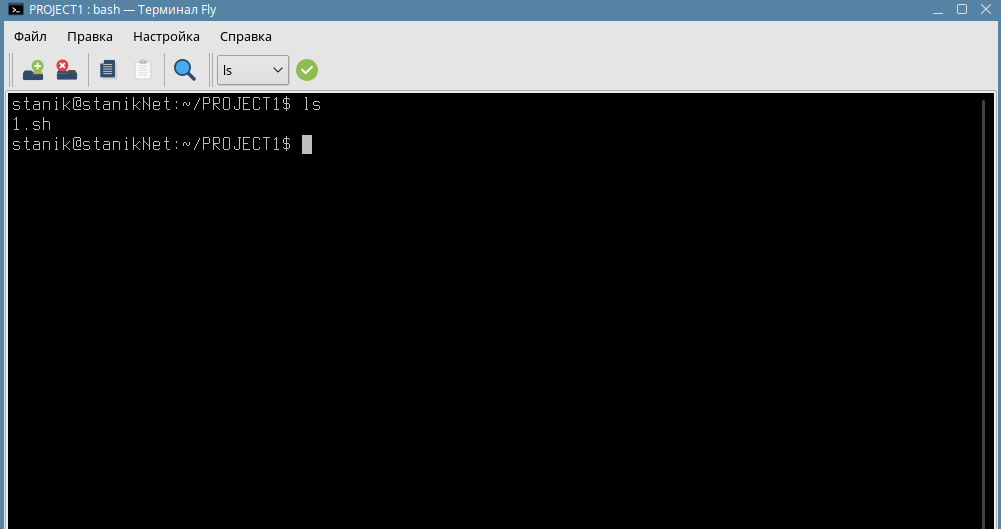


Рисунок - Создание файла 1.sh

C помощью редактора vim напишем скрипт для выполнения первого задания, который представлен на рисунке 2.

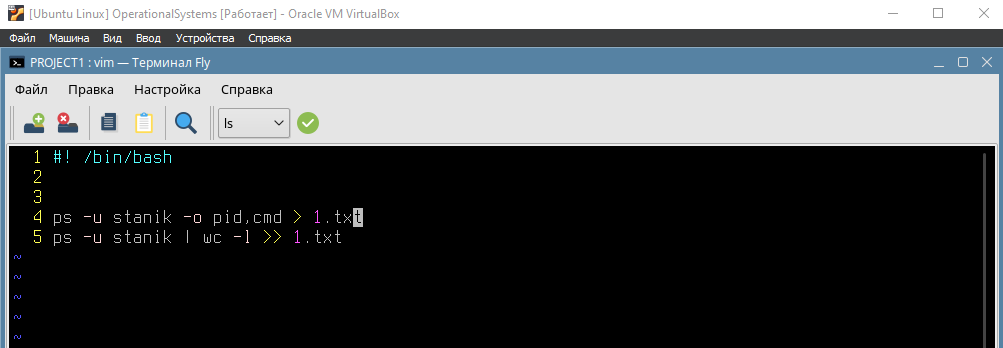


Рисунок - Скрипт для 1 задания

Сохраним файл и выйдем из vim с помощью команды :wq. Для запуска данного скрипта, его нужно сделать исполняемым. Для этого выполним команду chmod u+x 1.sh. Результат выполнения команды представлен на рисунке 3.

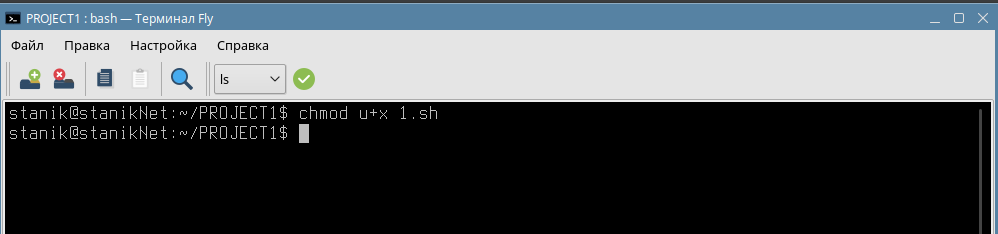


Рисунок - Выполнение команды

Протестируем скрипт запустив его командой ./1.sh. Результат представлен на рисунке 4.

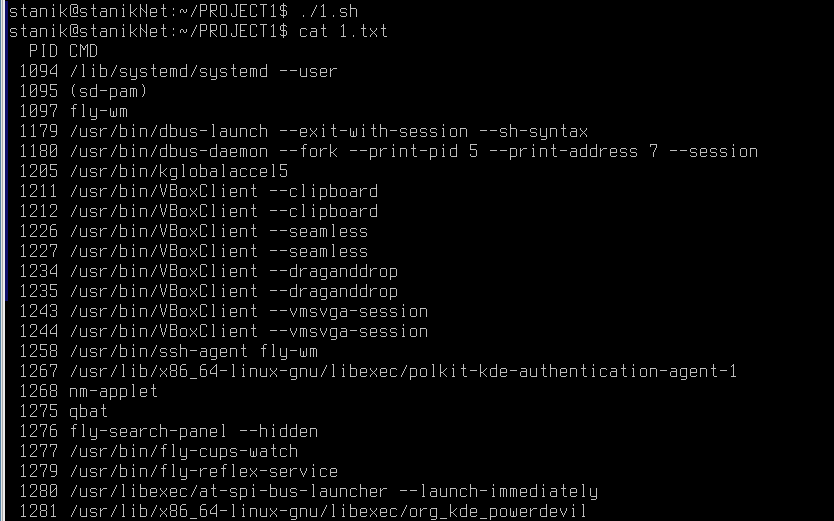


Рисунок - Выполнение скрипта

Было выведено количество запущенных процессов. Проверим файл 1.txt. Он представлен на рисунке 5.

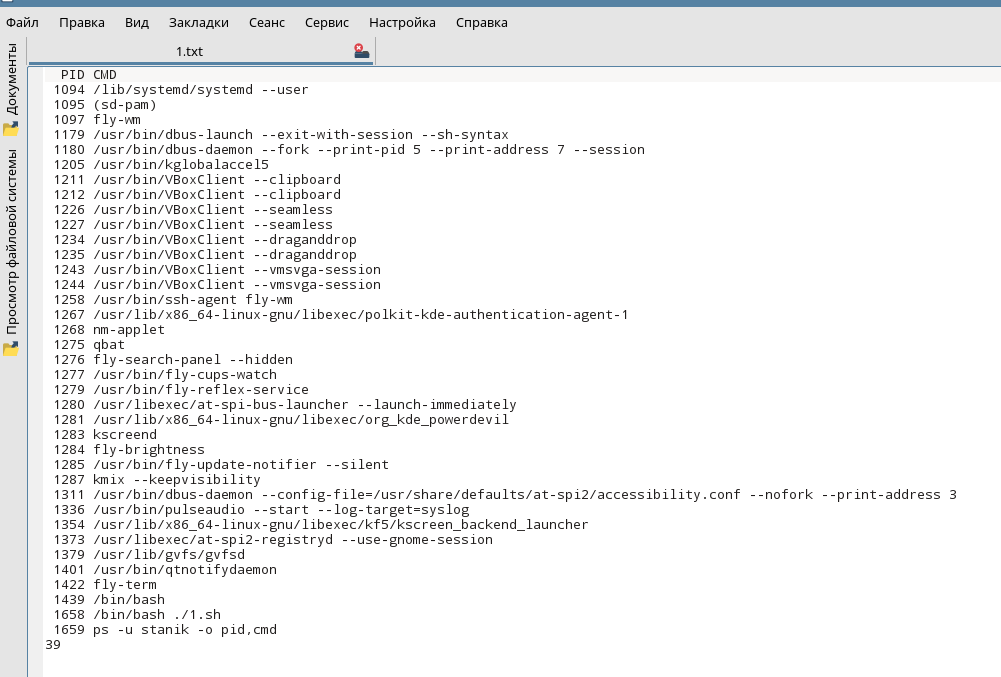


Рисунок - Результат выполнения скрипта

Скрипт успешно работает. Аналогично создадим скрипты для последующих заданий.

Скрипт для выполнения второго задания представлен на рисунке 6.

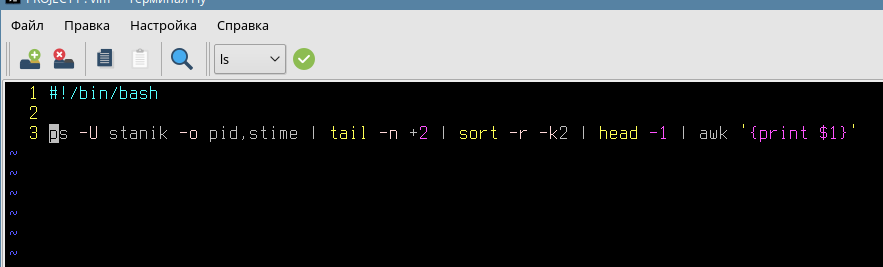


Рисунок - Скрипт для второго задания

Результат выполнения данного скрипта представлен на рисунке 7.

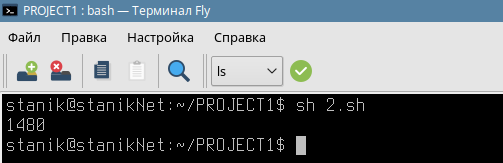


Рисунок - Результат выполнения скрипта

Скрипт для выполнения третьего задания представлен на рисунке 8.

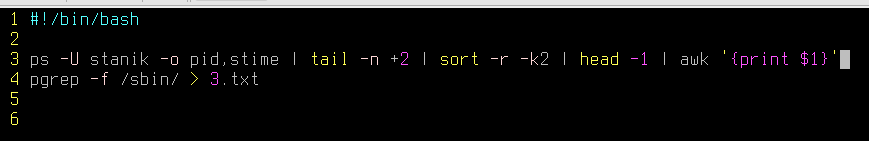


Рисунок - Скрипт для выполнения третьего задания

Результат выполнения данного скрипта представлен на рисунке 9.

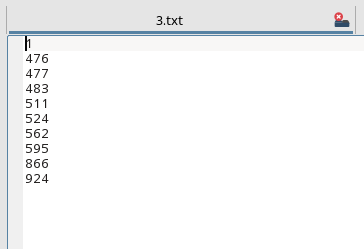


Рисунок - Результат выполнения скрипта

Скрипт для выполнения четвертого задания представлен на рисунке 10.

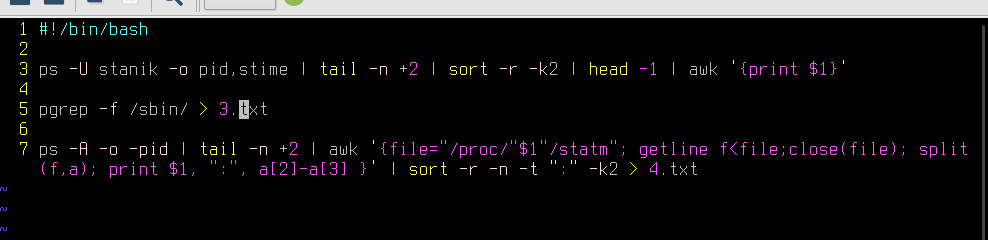


Рисунок - Скрипт для выполнения четвертого задания

Результат выполнения скрипта представлен на рисунке 11.

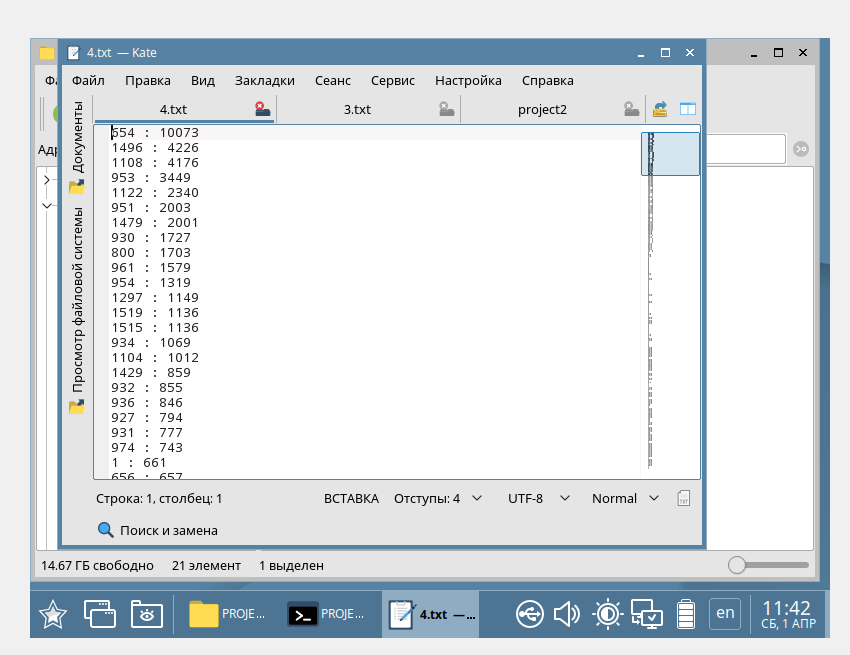


Рисунок - Результат выполнения скрипта

Скрипт для выполнения пятого задания представлен на рисунке 12.

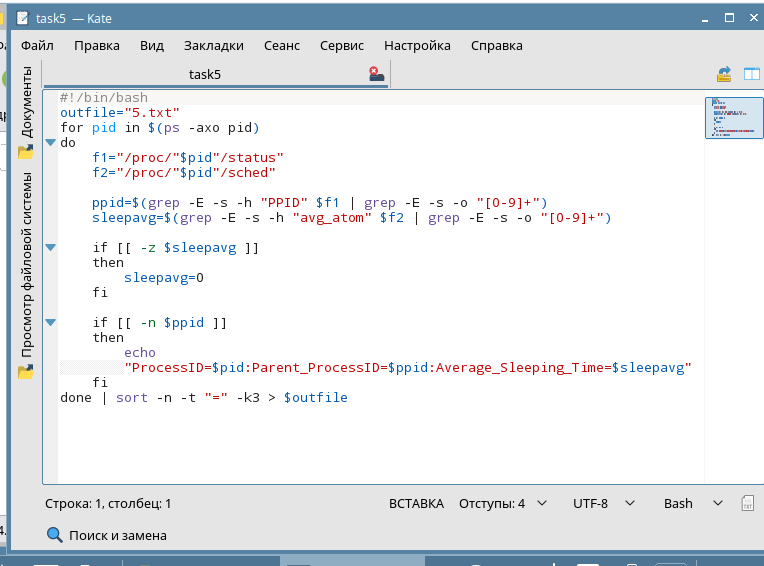


Рисунок - Скрипт для выполнения пятого задания

Результат выполнения скрипта представлен на рисунке 13.

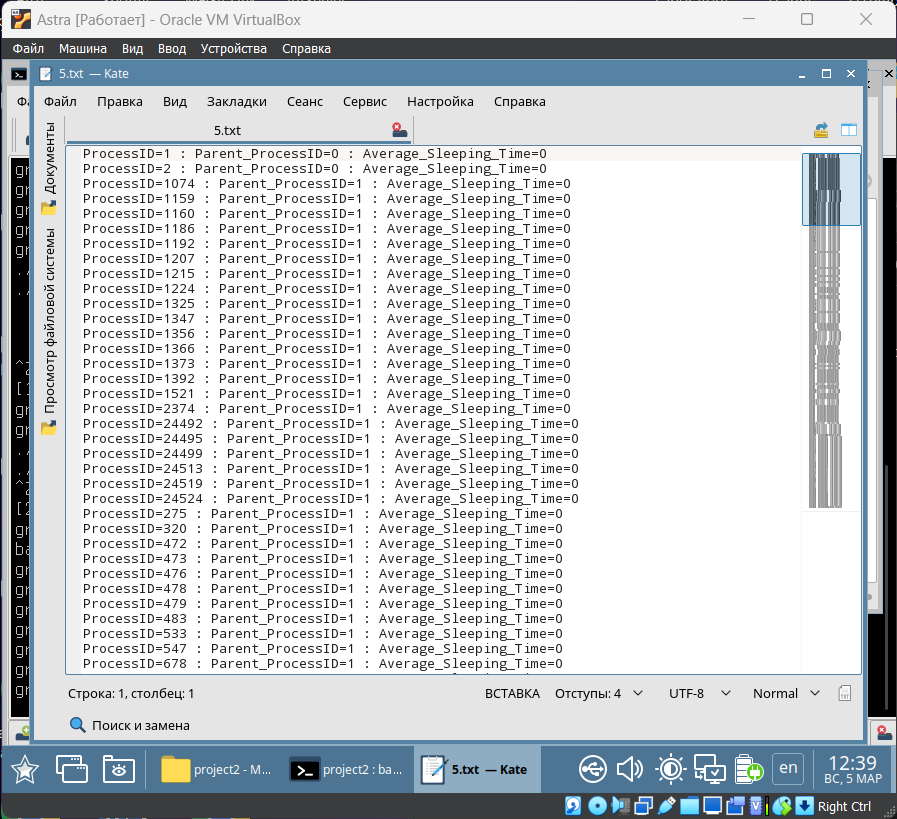


Рисунок - Результат выполнения скрипта

Скрипт для выполнения шестого задания представлен на рисунке 14.

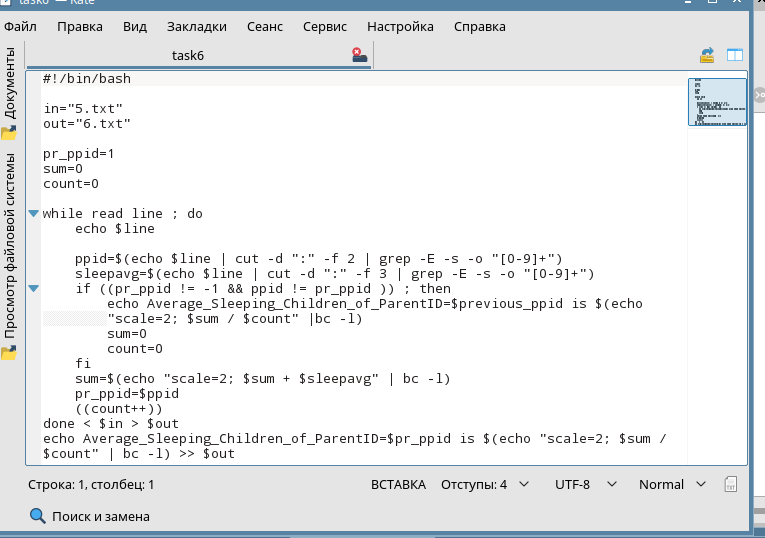


Рисунок - Скрипт для выполнения шестого задания

Результат выполнения данного скрипта представлен на рисунке 15.

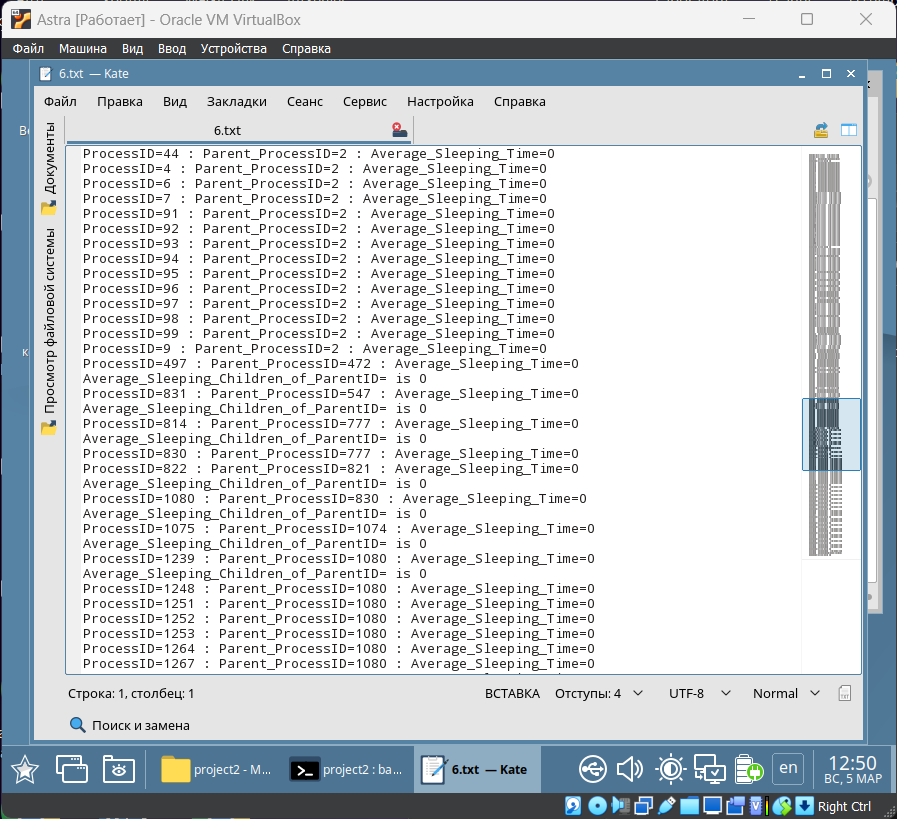


Рисунок - Результат выполнения

Совместим все эти скрипты в программу на C++, которая будет служить для запуска данных скриптов. Код программы представлен на рисунке 16.

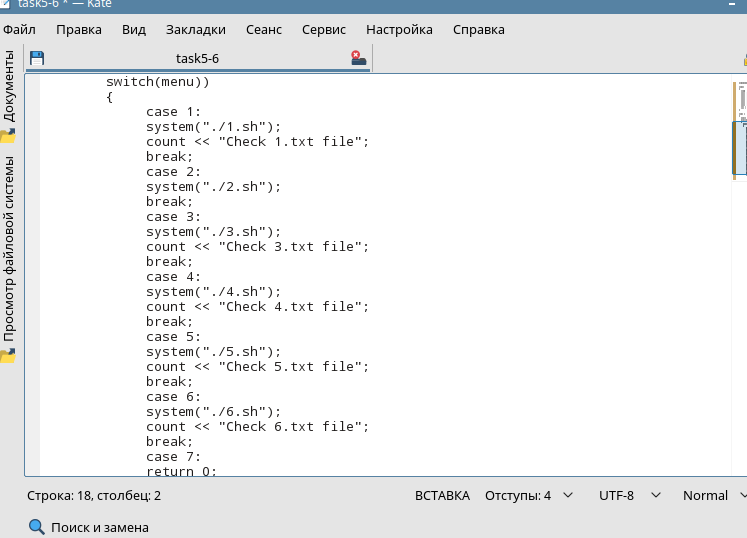
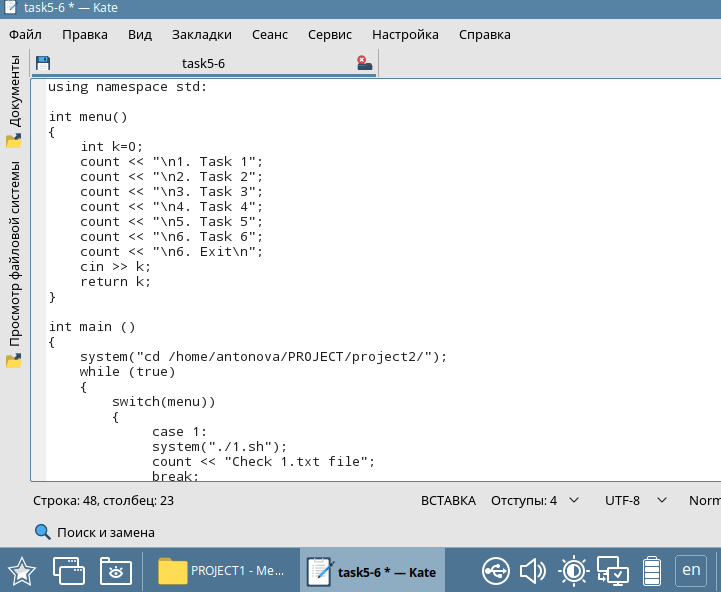


Рисунок - Код программы

Скомпилируем и запустим программу. Результат работы представлен на рисунке 17.

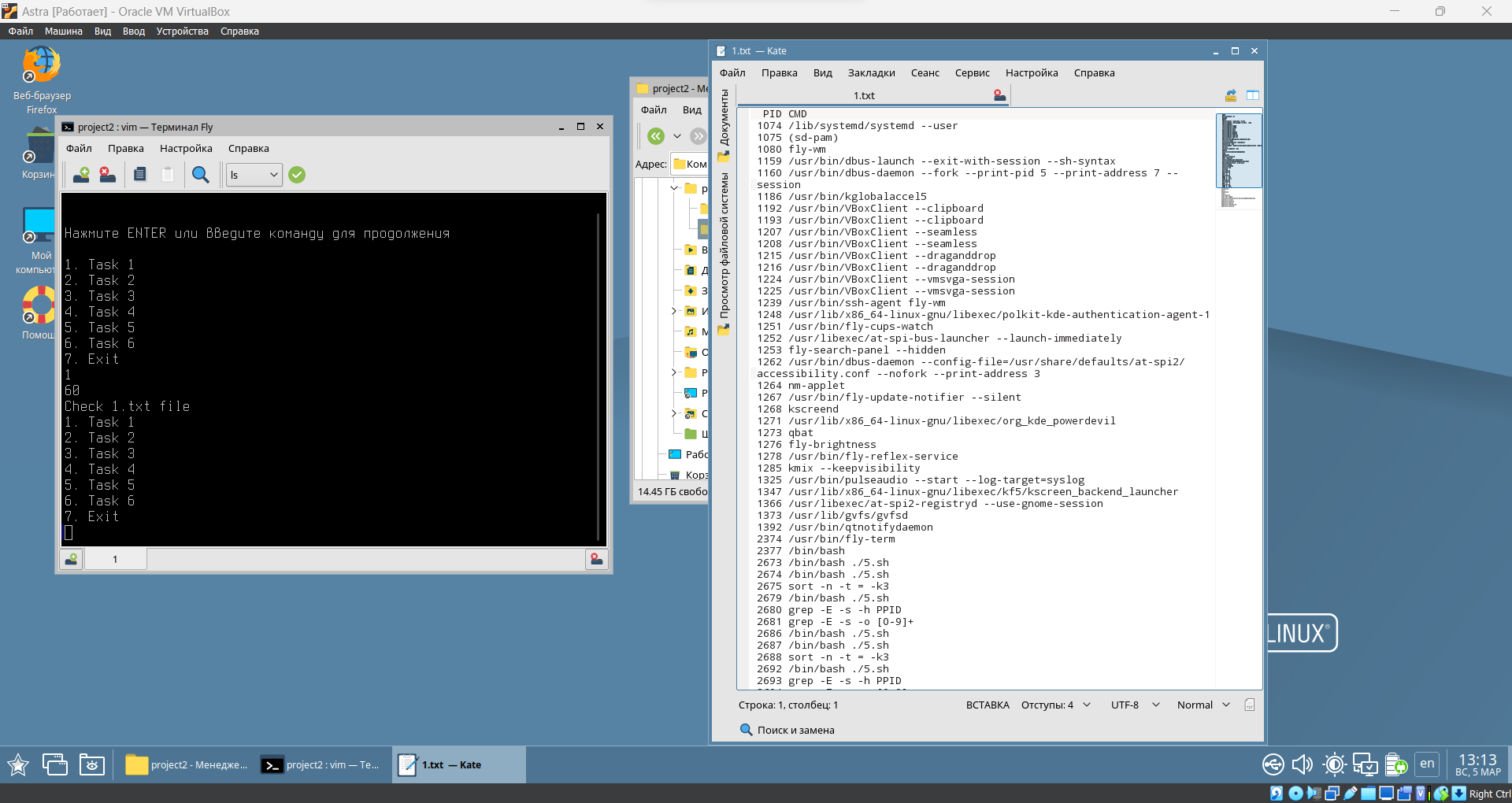


Рисунок - Результат работы программы

Контрольные вопросы

1. Что понимается под процессом в ОС Linux?

Процесс – это совокупность набора исполняемых команд, ассоциированных с ним ресурсов и контекста выполнения управляемая операционной системы. Процесс может содержать несколько потоков исполнения.

1. Что такое пользовательский контекст? Контекст ядра?

Под пользовательским контекстом процесса понимают код и данные, расположенные в адресном пространстве процесса. Все данные подразделяются на:

- инициализируемые неизменяемые данные (например, константы);

- инициализируемые изменяемые данные (все переменные, начальные значения которых присваиваются на этапе компиляции);

- неинициализируемые изменяемые данные (все статические переменные, которым не присвоены начальные значения на этапе компиляции);

- стек пользователя;

- данные, расположенные в динамически выделяемой памяти (например, с помощью стандартных библиотечных C функций malloc(), calloc(), realloc().

Под понятием «контекст ядра» объединяются системный контекст и регистровый контекст, рассмотренные на лекции. Мы будем выделять в контексте ядра стек ядра, который используется при работе процесса в режиме ядра (kernelmode), и данные ядра, хранящиеся в структурах, являющихся аналогом блока управления процессом – PCB.

1. В чем отличие классической (пятикомпонентной) модели состояния процесса от модели состояний процессов в ОС Linux?

Для описания состояний процессов используется несколько моделей. Самая простая модель - это модель трех состояний. Модель состоит из:

- состояния выполнения

- состояния ожидания

- состояния готовности

Выполнение — это активное состояние, во время которого процесс обладает всеми необходимыми ему ресурсами. В этом состоянии процесс непосредственно выполняется процессором.

Ожидание — это пассивное состояние, во время которого процесс заблокирован, он не может быть выполнен, потому что ожидает какое-то событие, например, ввода данных или освобождения нужного ему устройства.

Готовность — это тоже пассивное состояние, процесс тоже заблокирован, но в отличие от состояния ожидания, он заблокирован не по внутренним причинам (ведь ожидание ввода данных - это внутренняя, "личная" проблема процесса - он может ведь и не ожидать ввода данных и свободно выполняться - никто ему не мешает), а по внешним, независящим от процесса, причинам. Когда процесс может перейти в состояние готовности? Предположим, что наш процесс выполнялся до ввода данных. До этого момента он был в состоянии выполнения, потом перешел в состояние ожидания - ему нужно подождать, пока мы введем нужную для работы процесса информацию. Затем процесс хотел уже перейти в состояние выполнения, так как все необходимые ему данные уже введены, но не тут-то было: так как он не единственный процесс в системе, пока он был в состоянии ожидания, его "место под солнцем" занято - процессор выполняет другой процесс. Тогда нашему процессу ничего не остается как перейти в состояние готовности: ждать ему нечего, а выполняться он тоже не может.

Из состояния готовности процесс может перейти только в состояние выполнения. В состоянии выполнения может находится только один процесс на один процессор. Если у вас n-процессорная машина, у вас одновременно в состоянии выполнения могут быть n процессов.

Из состояния выполнения процесс может перейти либо в состояние ожидания или состояние готовности. Почему процесс может оказаться в состоянии ожидания, мы уже знаем - ему просто нужны дополнительные данные или он ожидает освобождения какого-нибудь ресурса, например, устройства или файла. В состояние готовности процесс может перейти, если во время его выполнения, квант времени выполнения "вышел". Другими словами, в операционной системе есть специальная программа - планировщик, которая следит за тем, чтобы все процессы выполнялись отведенное им время. Например, у нас есть три процесса. Один из них находится в состоянии выполнения. Два других - в состоянии готовности. Планировщик следит за временем выполнения первого процесса, если "время вышло", планировщик переводит процесс 1 в состояние готовности, а процесс 2 - в состояние выполнения. Затем, когда, время отведенное, на выполнение процесса 2, закончится, процесс 2 перейдет в состояние готовности, а процесс 3 - в состояние выполнения. Более сложная модель - это модель, состоящая из пяти состояний. В этой модели появилось два дополнительных состояния: рождение процесса и смерть процесса. Рождение процесса — это пассивное состояние, когда самого процесса еще нет, но уже готова структура для появления процесса. Как говорится в афоризме: "Мало найти хорошее место, надо его еще застолбить", так вот во время рождения как раз и происходит "застолбление" этого места. Смерть процесса - самого процесса уже нет, но может случиться, что его "место", то есть структура, осталась в списке процессов.

1. С помощью какого системного вызова может быть получено значение идентификатора текущего процесса, значение идентификатора родительского процесса для текущего процесса?

Значение идентификатора текущего процесса может быть получено с помощью системного вызова getpid() , а значение идентификатора родительского процесса для текущего процесса – с помощью системного вызова getppid() . Прототипы этих системных вызовов и соответствующие типы данных описаны в системных файлах <sys/types.h> и <unistd.h>. Системные вызовы не имеют параметров и возвращают идентификатор текущего процесса и идентификатор родительского процесса соответственно.

Вывод

В ходе выполнения данной лабораторной работы я изучила процессы в ОС GNU/Linux, написала программу, которая выполняет различные действия с ними.