**МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

Факультет прикладной математики и информатики

Кафедра технологий программирования

Жуковский Павел Сергеевич

Программирование Мобильных и Встраиваемых Систем

Отчет по лабораторной работе №3

## «Управление процессами и потоками»

студента 3 курса 12 группы

**Преподаватель**

**Давидовская Мария Ивановна**

**Минск 2020**

**Цель работы**

* Изучить команды управления процессами;
* Изучить вызовы и функции управления каталогами;
* Изучить вызовы и функции управления процессами.

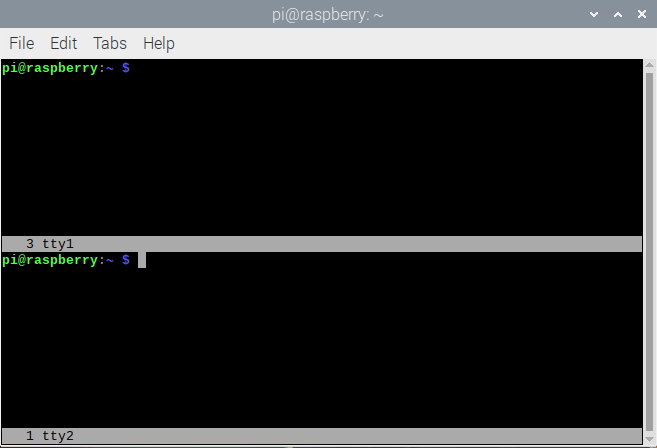
**Вариант 14**

**Ход выполнения**

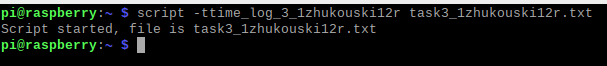
**Задание 3.1. Управление процессами**

*Изучите примеры задания 3.1 и выполните их в ОС Raspberry PI. Включите ведение протокола командой script с журналом меток времени. Протокол назвать по следующему шаблону — taskXФамилияNM, где X — номер выполняемого задания, Фамилия — заменить на вашу фамилию латиницей и строчными буквами, N — номер группы, например 12 или 13, M — r — для Raspberry PI. Журнал меток назвать по следующему шаблону — timelogXФамилияNM, где X — номер выполняемого задания, Фамилия — заменить на вашу фамилию латиницей и строчными буквами, N — номер группы, например 12 или 13, M — r — для Raspberry PI.*

*1. Используйте учетную запись, созданную в одной из предыдущих лабораторных работ. Войдите в систему на виртуальных терминалах 1 и 2 (tty1, tty2) под вашей учетной записью.*

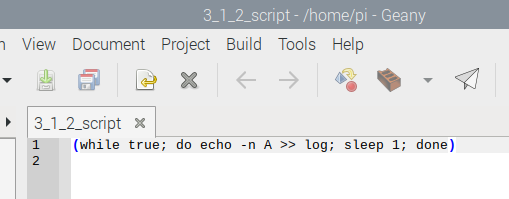


Для этого пришлось прописать sudo apt install screen, потом создать два терминала, разделить их командой split (или сочетанием ctrl+A+S) и переименовать каждый из них с помощью сочетаний клавиш ctrl+A, shift+A, добавить (имя пользователя@rapberry:~ $) во втором окне можно с помощью сочетания ctrl+A+C, переключение между ними можно сделать с помощью сочетаний ctrl+A+TAB. (тут я узнал, как работать с утилитой screen: [ссылочка](https://losst.ru/komanda-screen-linux))

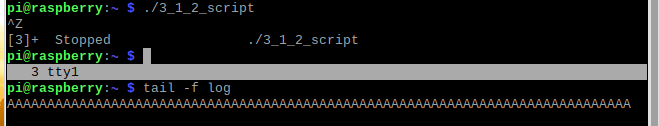


*2. Переключитесь в терминал tty1 и запустить процесс, выполняющий следующие команды:*

*$ (while true; do echo -n A >> log; sleep 1; done)*



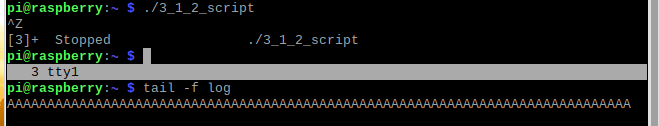




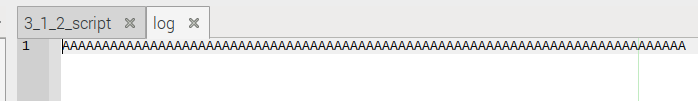
*3. Заметьте, что сейчас этот терминал занят исполнением запущенного процесса, который исполняется на переднем плане. Этот процесс присоединяет символ "А"к файлу ~/log через каждую секунду. Чтобы визуально проверить это, переключитесь в виртуальный терминал tty2 и выполните следующую команду:*

*$ tail -f log*

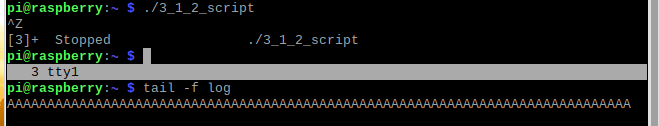
*Вы должны увидеть последовательность символов, длина которой возрастает.*

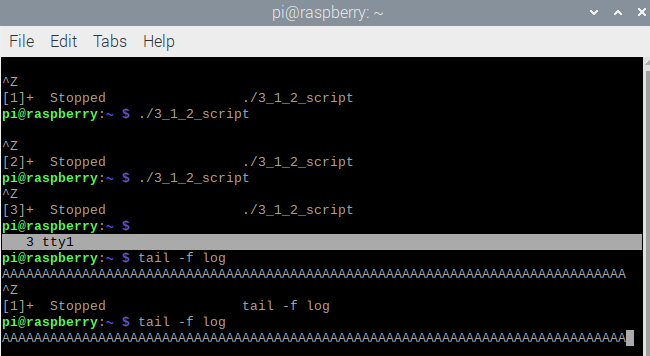


Файл log:



*4. Переключитесь в виртуальный терминал tty1 и приостановите работающий процесс, нажав клавиши <Ctrl+z>. Командная оболочка сообщит, что процесс остановлен и выдаст вам номер задания [1]. Переключитесь в виртуальный терминал tty2 и визуально проверьте, что файл ~/log больше не увеличивается.*



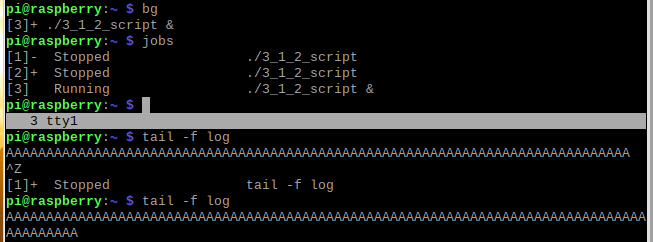


*5. Переключитесь в виртуальный терминал tty1 и возобновите работу процесса в фоновом режиме. Используйте команду jobs, чтобы проверить, что задание [1] снова работает.*

*$ bg*

*$ jobs*

*Переключитесь в виртуальный терминал tty2 и визуально проверьте, что файл ~/log снова увеличивается.*

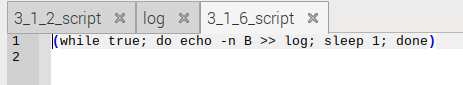


*6. Переключитесь в виртуальный терминал tty1 и запустите еще два процесса, выполнив следующие команды:*

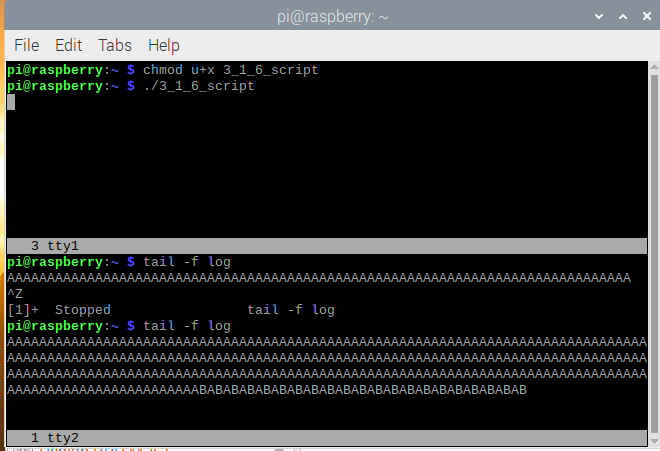
*$ (while true; do echo -n B >> log; sleep 1; done) &*

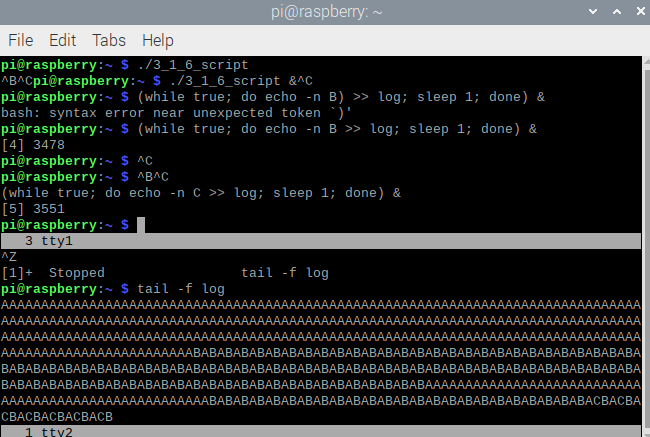
*$ ^B^C*

*Вторая команда просто запускает предыдущую команду, заменяя символ "B"символом "C".*

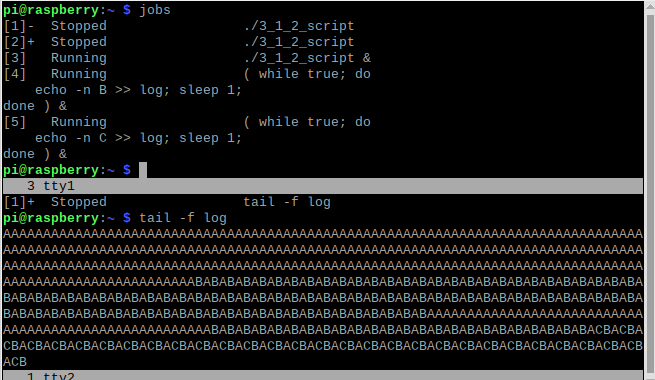








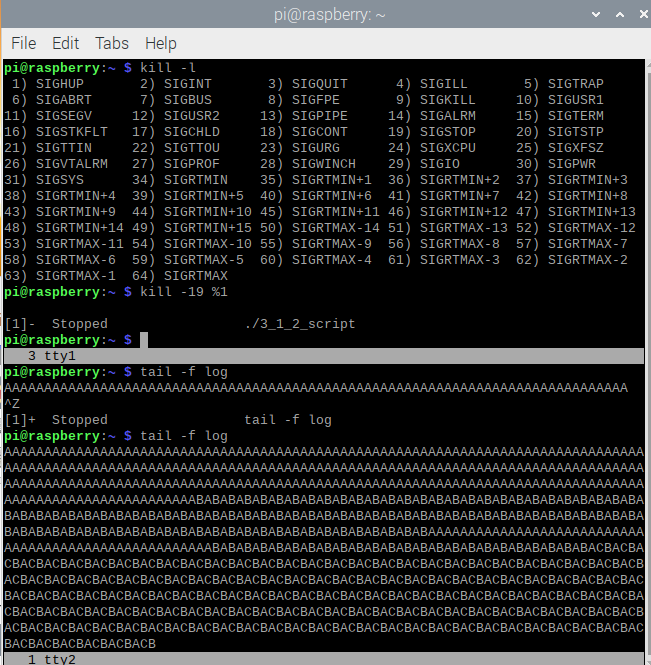
*7. Выполните команду jobs и проверьте, что все три процесса работают. Переключитесь в виртуальный терминал tty2 и визуально проверьте, что файл ~/log снова увеличивается путем добавления символов "A" "B"и "C"через каждую секунду.*



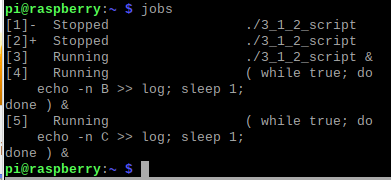
*8. В пункте 4 вы приостановили исполнение процесса переднего плана путем нажатия клавиш <Ctrl+z>. В действительности эта комбинация нажатых клавиш посылает процессу сигнал. Используйте команду kill, чтобы получить список сигналов и соответствующие им имена и номера. Затем выполните команду kill, послав сигнал SIGSTOP заданию [1], чтобы приостановить его работу. Переключитесь в виртуальный терминал tty1 и выполните следующие команды:*

*$ kill -l*

*$ kill -19 %1*



*9. Выполните команду jobs и проверьте, что задание [1] остановлено. Переключитесь в виртуальный терминал tty2 и визуально проверьте, что задание [1] остановлено.*



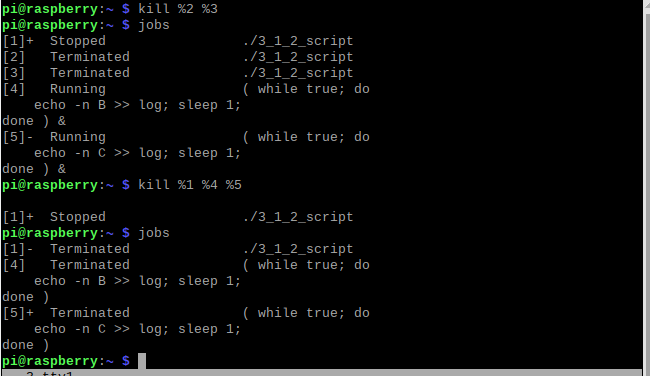
*10. Возобновите выполнение задания [1], используя команду kill, которая посылает процессу сигнал SIGCONT (18). Используйте команду jobs и виртуальный терминал tty2 для проверки того, что все три задания опять работают.*



*11. Завершите работу всех трех процессов. Если вы не задаете сигнал, который нужно послать процессу, то команда kill посылает по умолчанию сигнал SIGTERM (15), который вызывает завершение процесса. После посылки сигналов заданиям [2] и [3], используйте команду jobs, чтобы проверить завершение работы этих заданий:*

*$ kill %2 %3*

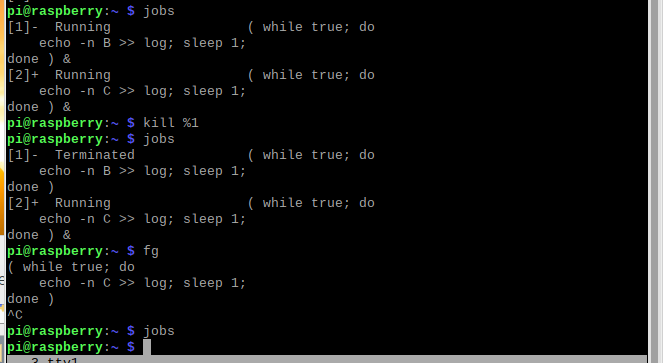
*$ jobs*



*12. Чтобы завершить работу последнего процесса, выполните команды:*

*$ fg*

*$ <Ctrl+c>*

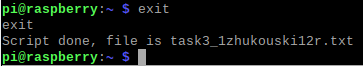


*13. Выполните команду jobs и проверьте, что больше заданий не выполняется. Переключитесь в виртуальный терминал tty2 и визуально проверьте, что файл ~/log не увеличивается. Остановите исполнение команды tail, нажав клавиши <Ctrl+c>, и завершите сеанс на виртуальном терминале tty2.*



*14. Переключитесь в виртуальный терминал tty1 и удалите файл ~/log.*



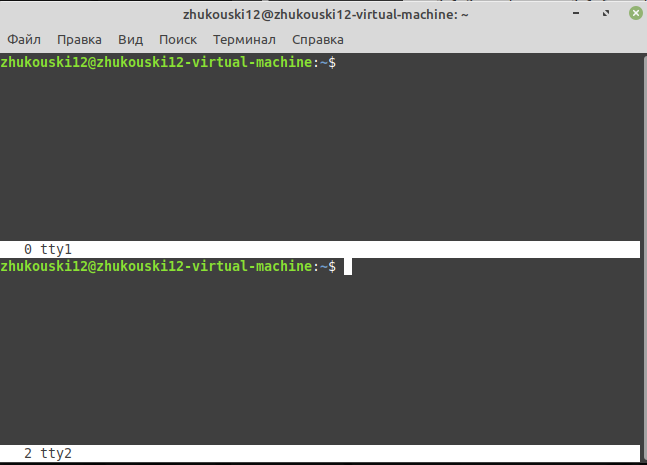


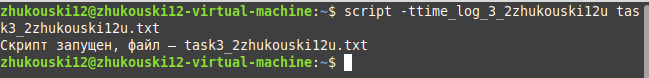
**Задание 3.2. Управление процессами**

*Изучите примеры задания 3.2 и выполните их в ОС Ubuntu. Включите ведение протокола командой script с журналом меток времени. Протокол назвать по следующему шаблону — taskXФамилияNM, где X — номер выполняемого задания, Фамилия — заменить на вашу фамилию латиницей и строчными буквами, N — номер группы, например 12 или 13, M — u — для Ubuntu. Журнал меток назвать по следующему шаблону — timelogXФамилияNM, где X — номер выполняемого задания, Фамилия — заменить на вашу фамилию латиницей и строчными буквами, N — номер группы, например 12 или 13, M — u — для Ubuntu.*

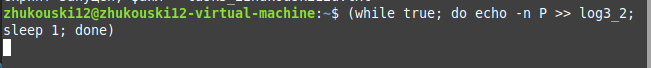
*1. Войдите в систему на виртуальных терминалах 1 и 2 (tty1, tty2) под учетной своей записью.*

По аналогии с прошлым заданием

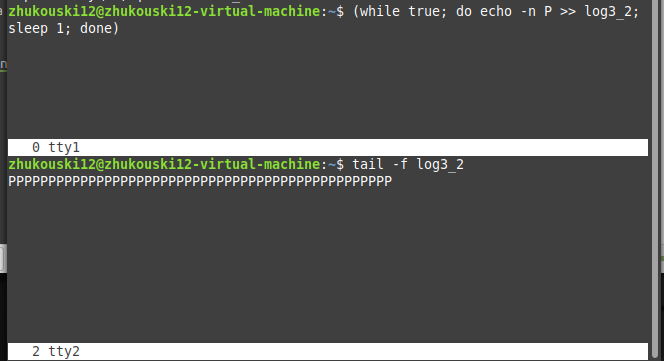




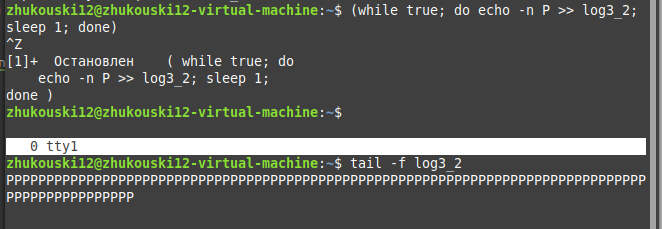
*2. Переключитесь в терминал tty1 и запустите на переднем плане бесконечный процесс.*



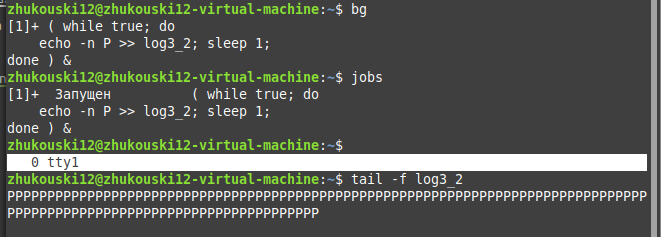
*3. Переключитесь в виртуальный терминал tty2 и проверьте работу процесса, запущенного на пункте 2.*



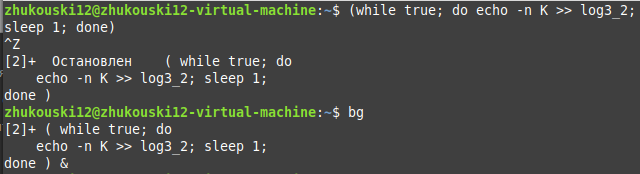
*4. Переключитесь в виртуальный терминал tty1 и приостановите работающий процесс.*



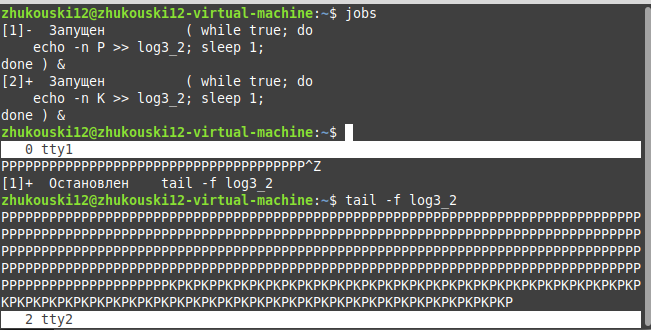
*5. Возобновите работу процесса в виртуальном терминале tty1 в фоновом режиме.*



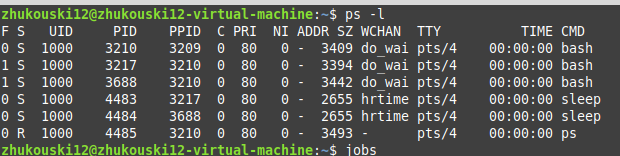
*6. Запустите в виртуальном терминале tty1 второй бесконечный процессов запущенных в фоновом режиме.*



*7. Выполните команду jobs и проверьте, что два запущенных процесса работают.*

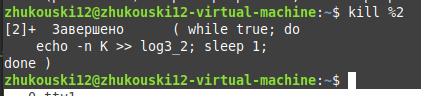


*8. Установите приоритет процесса, запущенного в пункте 2, равным 10.*



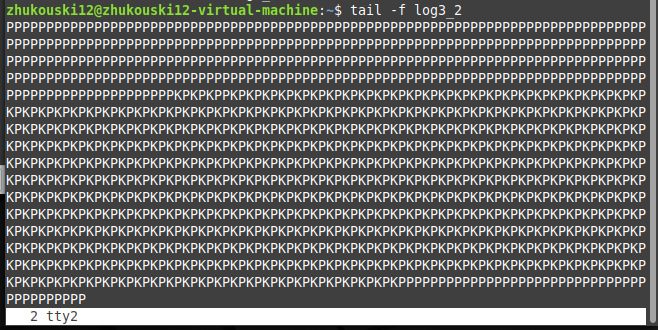


*9. Остановите процесс, запущенный в пункте 6, командой kill.*

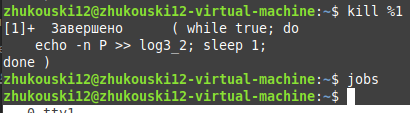


*10. Проверьте, что работает только один процесс.*

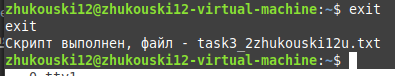
Теперь выводятся только буквы P:



*11. Прервите исполнение работающего процесса и проверьте, что не осталось работающих процессов.*

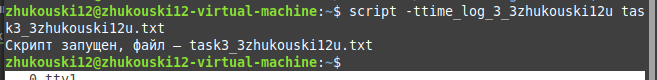


Вывод во втором терминале также остановился.



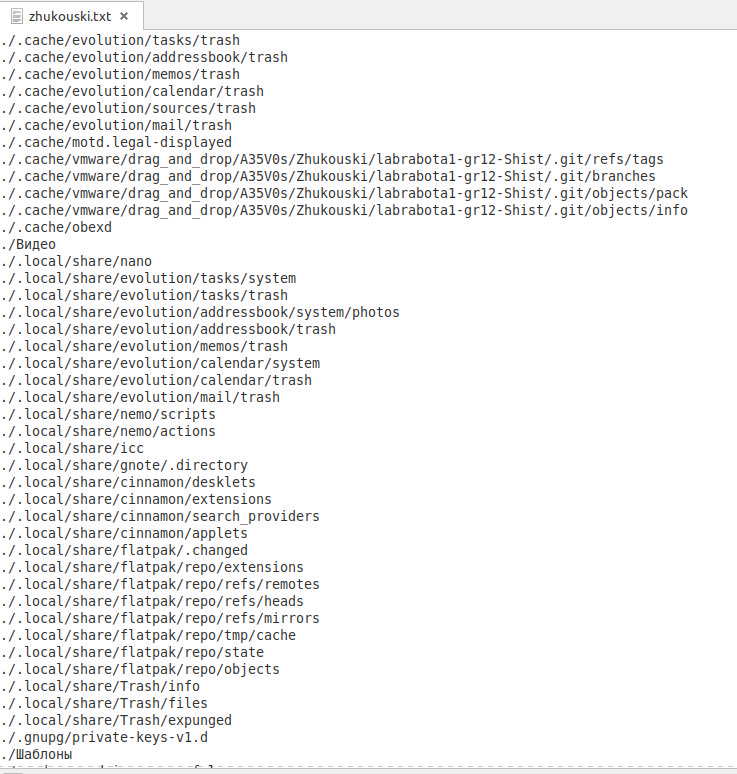
**Задание 3.3. Управление процессами и планирование заданий**

*Изучите примеры задания 3.3 и выполните их в ОС Ubuntu. Включите ведение протокола командой script с журналом меток времени. Протокол назвать по следующему шаблону — taskXФамилияNM, где X — номер выполняемого задания, Фамилия — заменить на вашу фамилию латиницей и строчными буквами, N — номер группы, например 12 или 13, M — u — для Ubuntu. Журнал меток назвать по следующему шаблону — timelogXФамилияNM, где X — номер выполняемого задания, Фамилия — заменить на вашу фамилию латиницей и строчными буквами, N — номер группы, например 12 или 13, M — u — для Ubuntu.*

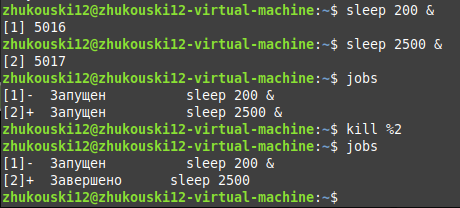


*1. Найдите файлы а) пустые б) скрытые в домашнем каталоге в фоновом режиме и результат сохраните в файл со своей фамилией.*



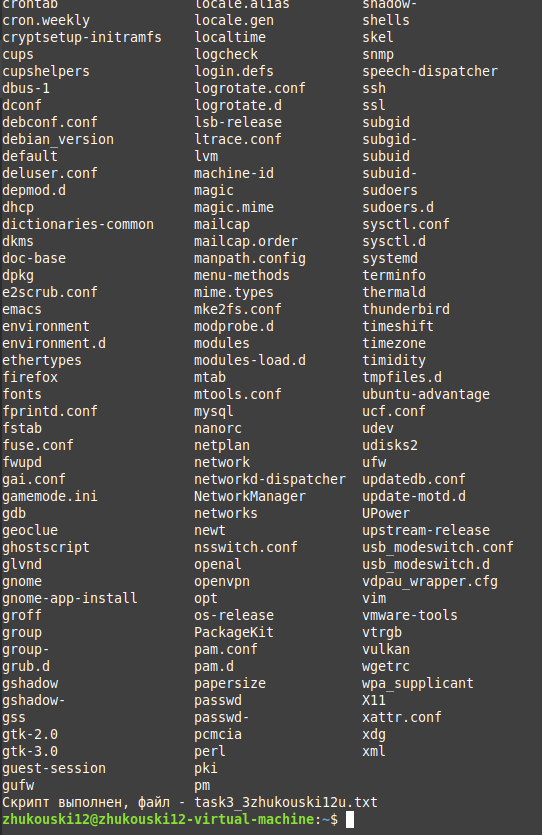


*2. Запустите в фоновом режиме два задания: sleep 200 и sleep 2500, выведите информацию о состоянии заданий. Снимите с выполнения второе задание, выведите информацию о заданиях.*

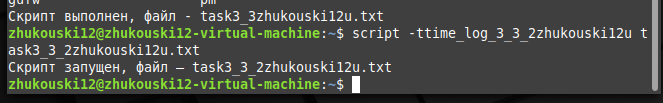


*3. Выполните команду exec ls R /etc. Изучите её поведение.*

Фрагмент её вывода:



Данная команда выполняет рекурсивное отображение содержимого поддиректорий /etc. К тому же эта команда почему-то завершила мой скрипт, потому я создал второй:



*4. Запустите порожденную оболочку bash. Исследуйте, посылая родительской оболочке сигналы TERM, INT, QUIT и HUP, что при этом происходит?*

TERM – завершает процесс

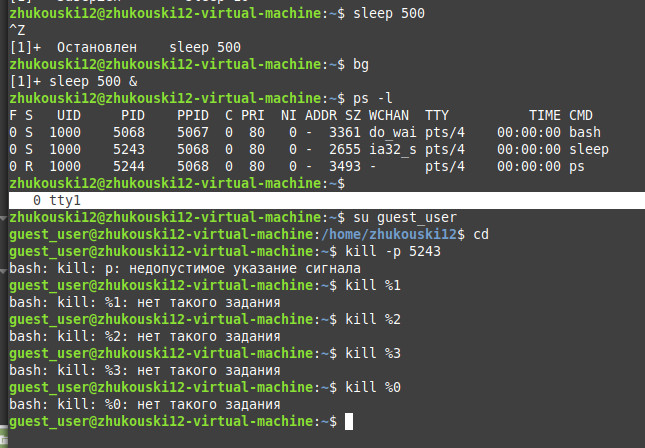
INT – прерывает процесс

QUIT – выходит из процесса

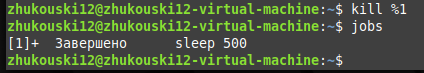
HUP – обрыв терминальной линии

*5. От имени обычного пользователя пошлите сигнал KILL любому процессу, запущенному от имени другого пользователя. Что произойдет?*

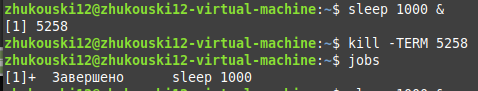
Я провёл эксперименты и выяснил, что второму пользователю попросту не видны процессы первого пользователя:

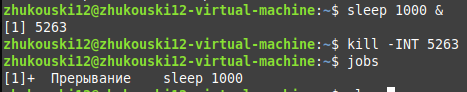


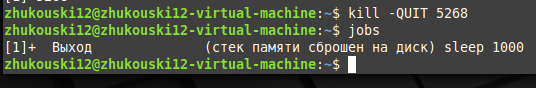
Хотя первый пользователь этот же процесс видит и может завершить:

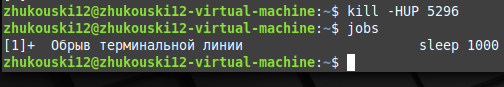


*6. Запустите в фоновом режиме команду sleep 1000. Проверьте, на какие сигналы из следующих: TERM, INT, QUIT и HUP, реагирует эта команда.*

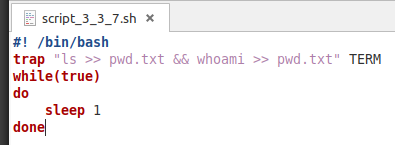




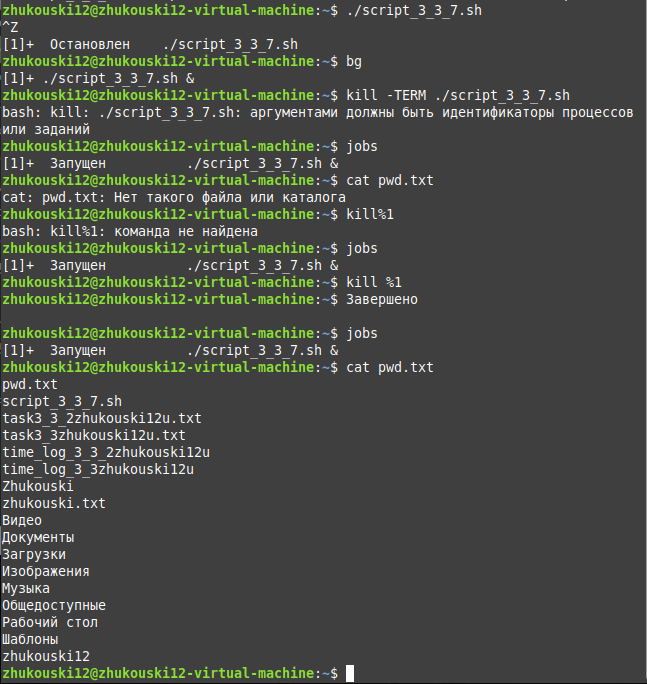




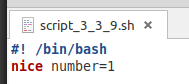
*7. Запрограммируйте оболочку так, чтобы при получении ей сигнала TERM создавался файл pwd.txt, содержащий информацию о текущем каталоге и текущем пользователе.*

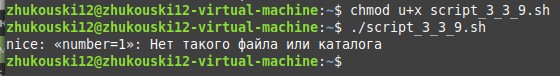


*8. Запустите порожденную оболочку. Работает ли в ней созданный обработчик?*

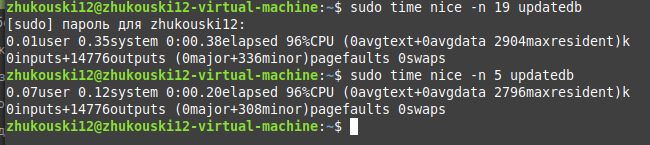


*9. От имени обычного пользователя попытайтесь запустить оболочку bash со значением nice number, равным 1. Какое сообщение выводится?*

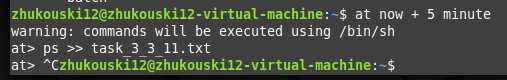




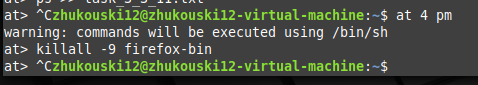
*10.От имени суперпользователя запустите команду индексирования базы данных поиска вследующем виде: time nice n 19 updatedb. Затем выполните такую же команду, в которой значение nice number для updatedb будет 5. Сравните полученные результаты.*



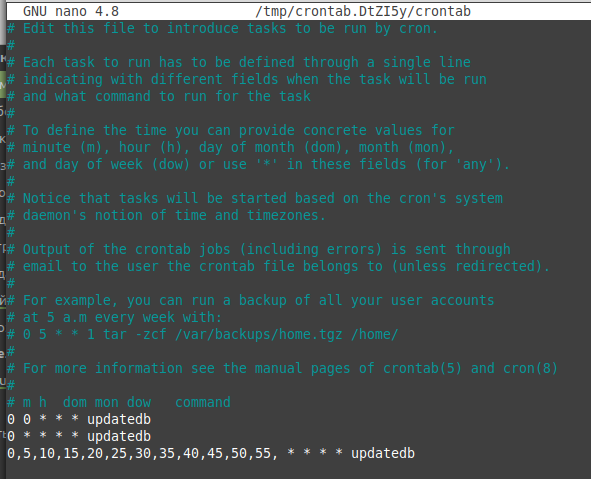
*11. При помощи команды at сделать так, чтобы ровно через 5 минут от текущего времени произошла запись списка всех процессов в файл с именем, содержащим в своём названии системное время на момент записи.*



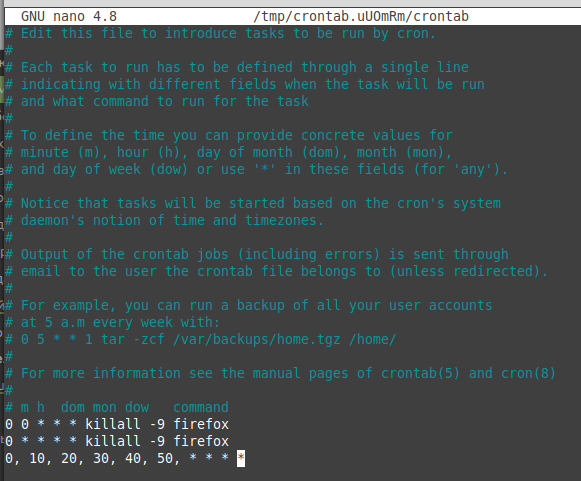
*12.При помощи команды at организовать обычное завершение работы браузера firefox или chrome в 16:00.*



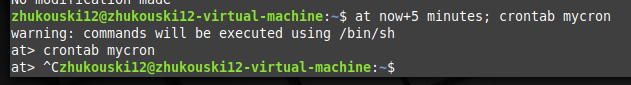
*13.Сделать при помощи cron так, чтобы команда updatedb запускалась раз в сутки, каждый час, каждые 5 минут.*



*14.При помощи cron организовать убийство браузера firefox и chrome каждые 10 минут.*

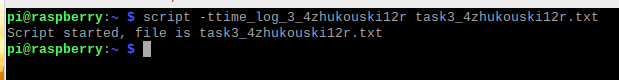


*15.При помощи команды at сделать так, чтобы через 5 минут от текущего времени создалось задание для cron, которое создавало бы каждые 9 минут ещё одно задание для cron, заключающееся в том, чтобы каждые 7 минут уничтожать все задания пользователя для cron.*



**Задание 3.4. Управление процессами и планирование заданий**

*Изучите задачи задания 3.4 и выполните их в ОС Raspberry PI. Выполните задачи ниже и результат запуска скрипта из каждой задачи залогируйте с помощью команды script с журналом меток времени. Протокол назвать по следующему шаблону — taskXФамилияNM, где X — номер выполняемого задания, Фамилия — заменить на вашу фамилию латиницей и строчными буквами, N — номер группы, например 12 или 13, M — r — для Raspberry PI. Журнал меток назвать по следующему шаблону — timelogXФамилияNM, где X — номер выполняемого задания, Фамилия — заменить на вашу фамилию латиницей и строчными буквами, N — номер группы, например 12 или 13, M — r — для Raspberry PI.*



***Задача 1.*** *Создайте и запустите следующий сценарий, который представляет бесконечный процесс, выводящий значение счетчика каждую секунду и завершающий свою работу при нажатии клавиш <Ctrl+c>.*

*#!/bin/bash*

*# trap test*

*trap ’echo you hit Ctrl+c; exit’ SIGINT*

*count=0*

*while:*

*do*

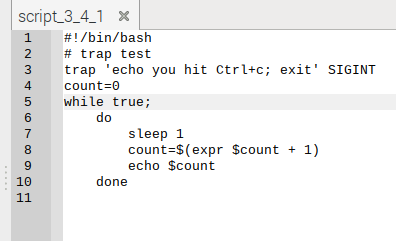
*sleep 1*

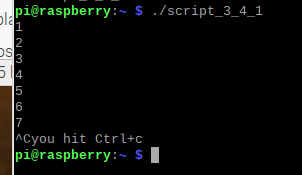
*count=$(expr $count + 1)*

*echo $count*

*done*

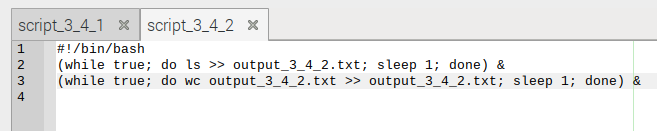




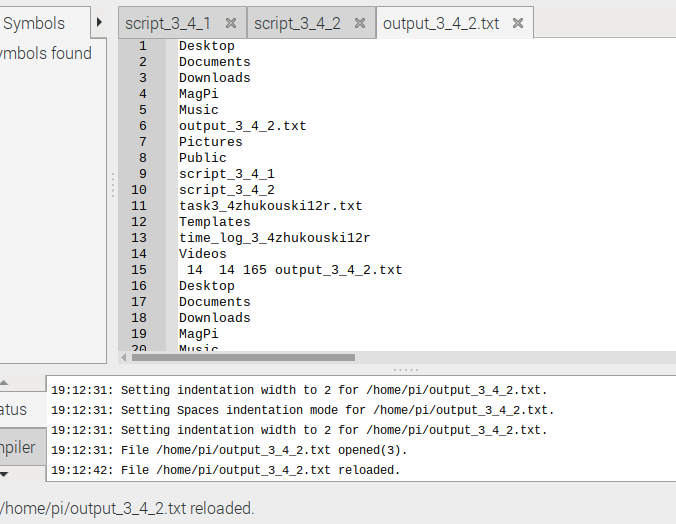


***Задача 2.*** *Написать пример программы, которая запускает и связывает каналом два процесса: вывод содержимого каталога и подсчет количества строк (ls и wc).*



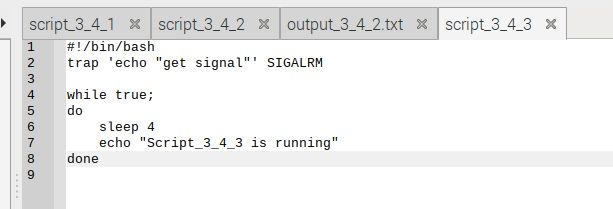


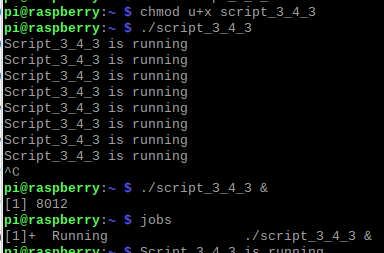


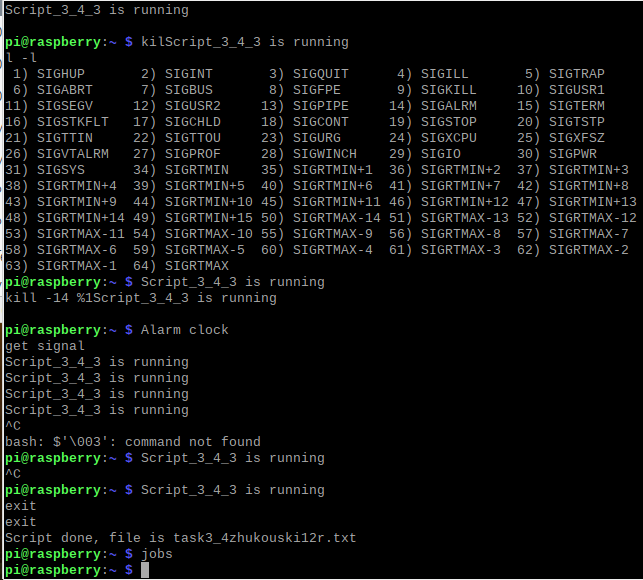


***Задача 3.*** *Изучить параграф 10.10. «Функции alarm и pause» из книги У. Ричард Стивенс, Стивен А. Раго. UNIX. Профессиональное программирование. 3-е изд. Написать пользовательскую функцию обработки сигнала. Установка обработки сигнала происходит одноразово (обрабатывается только одно событие, связанное с появлением данного сигнала SIG\_ALRM). Возврат из функции-обработчика происходит в точку прерывания процесса.*





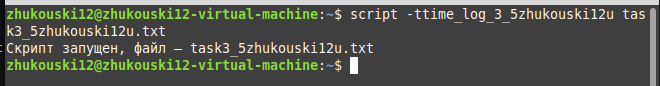




**Задание 3.5. Управление процессами**

*Перед началом выполнения задания изучите материалы глав 8-12 из книги У. Ричард Стивенс, Стивен А. Раго. UNIX. Профессиональное программирование. 3-е изд.*

*Результаты тестирования работы скриптов залогируйте, т. е. включите ведение протокола командой script с журналом меток времени. Протокол назвать по следующему шаблону — taskXФамилияNM, где X — номер выполняемого задания, Фамилия — заменить на вашу фамилию латиницей и строчными буквами, N — номер группы, например 12 или 13, M — u — для Ubuntu. Журнал меток назвать по следующему шаблону — timelogXФамилияNM, где X — номер выполняемого задания, Фамилия — заменить на вашу фамилию латиницей и строчными буквами, N — номер группы, например 12 или 13, M — u — для Ubuntu.*



**Варианты индивидуальных заданий**

*Номер индивидуального варианта K равен числу букв вашей фамилии (N1), умноженному на число букв по паспорту вашего имени (N2), умноженному на число букв Вашего отчества (N3) по модулю 23:*

*K= (N1\*N2\*N3) mod 23*

*В каждой программе должен быть контроль ошибок для всех операций с файлами и каталогами.*

**Жуковский – 9 букв**

**Павел – 5 букв**

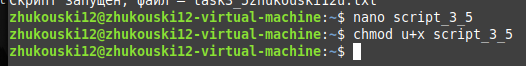
**Сергеевич – 9 букв**

**k = (9\*5\*9) mod 23 = 405 mod 23 = 14 => 14 Вариант**

**Вариант 14.**

*14. То же, что и в п. 4, но вместо процессов использовать потоки.*

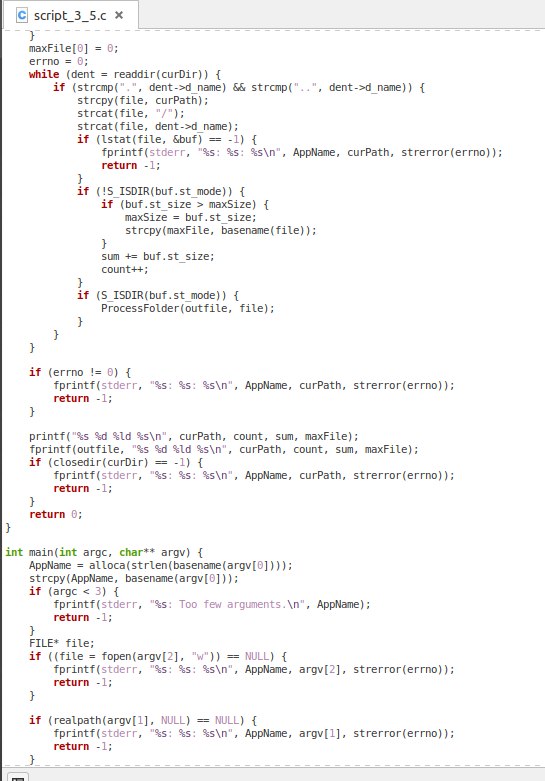
*4. Подсчитать суммарный размер файлов в заданном каталоге (аргумент 1 командной строки) и для каждого его подкаталога отдельно. Вывести на консоль и в файл (аргумент 2 командной строки) название подкаталога, количество файлов в нём, суммарный размер файлов, имя файла с наибольшим размером. Процедура просмотра для каждого подкаталога должна запускаться в отдельном процессе (****потоке****). Каждый процесс (****поток****) выводит на экран свой pid, полный путь, имя и размер просмотренного файла, общее число просмотренных файлов в подкаталоге. Число запущенных процессов (****потоков****) в любой момент времени не должно превышать N (вводится пользователем). Проверить работу программы для каталога /usr N=6.*



Некоторые моменты для скрипта были взяты из следующего источника:

[ссылочка](https://github.com/Emaxan/OSiSP_Lab2/blob/master/laba2.c)



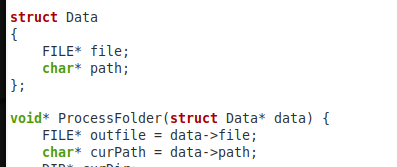


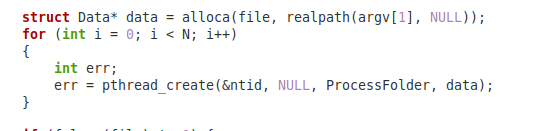


Ключевые фрагменты:





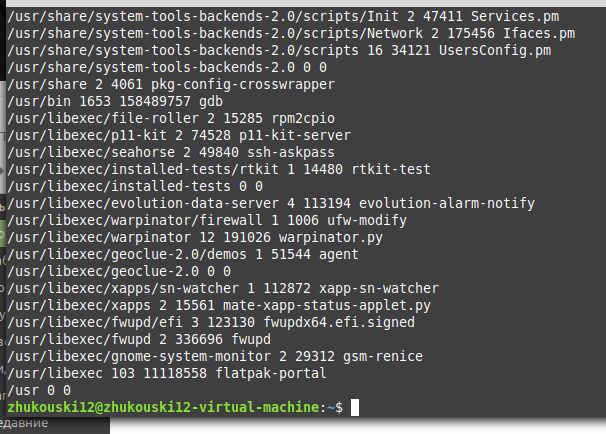


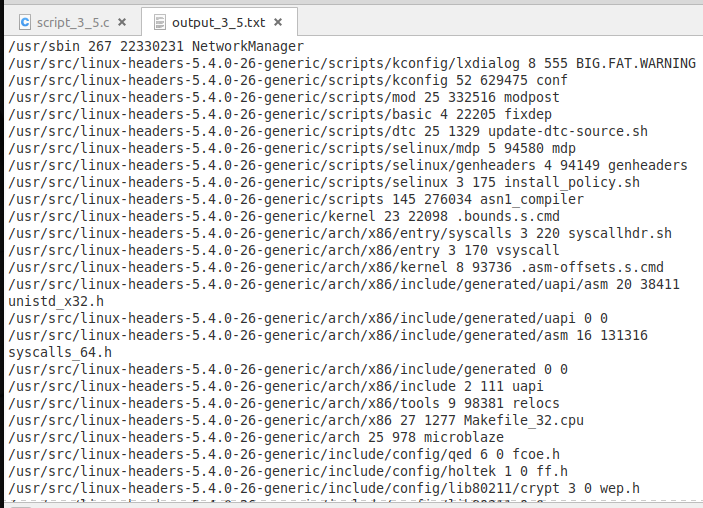


Изменил расширение файла на \*.c и создал исполняемый файл, затем запустил:



Часть вывода для каталога /usr:





**Контрольные вопросы**

*1) Объясните понятия процесса и ресурса. Какое их значение в организации вычислительного процесса в ОС Linux?*

Вычислительный процесс представляет собой процесс выполнения некоторой программы вместе с данными этой программы на процессоре (это может быть что угодно, от трансляции до редактирования того или иного текста). Ресурс – это некоторый объект, обладающий свойствами неоднократного и/или повторного использования процессами, которые эти ресурсы запрашивают, используют и затем освобождают.

*2) Какая информация содержится в описателях процессов? Как просмотреть их содержание в процессе работы с системой?*

По сути, описатель процесса – это его дескриптор. А в дескрипторе мы можем найти как исполняемый текст программы задачи, так и управляющую информацию (которая используется для выделения и использования каких-то ресурсов для процесса, приоритет этого процесса, адресное пространство и т.д.).

Просмотреть содержание можно командами: **ps –l** или **ps**.

*3) Какими способами можно организовать выполнение программ в фоновом*

*режиме?*

Для того, чтобы запустить выполнение программы в фоновом режиме, достаточно прописать амперсанд на конце команды. В этом также могут помочь команды: **bg** или **fg**.

*4) Какие особенности выполнения программ в фоновом режиме? Как избежать вывода фоновых сообщений на экран и прерывания выполнения фоновых программ при прекращении сеанса работы с системой?*

В отличие от обычного режима, фоновой режим выполнения программ позволяет одной программе не дожидаться своего ввода во время завершения другой программы, что очень удобно. При фоновых процессах не происходит блокировка доступа к оболочке операционной системы. Прерывание выполнения фоновых программ при прекращении сеанса можно несколькими способами:

> С помощью команды **disown**

> С помощью команды **nohup** command &

*5) Как пользователь может повлиять на распределение ресурсов между активными процессами?*

Для влияния на распределение ресурсов между активными процессорами пользователь может изменять их приоритеты. Сделать это можно с помощью команд:

> **nice** –n 10 apt-get upgrade

> **renice** –n 10 –p 1343

При чём 1-ая задаёт приоритет, а 2-ая – изменяет приоритет.

*6) Как можно прервать выполнение активных процессов? Какая информация для этого необходима и откуда она извлекается?*

Если нам нужно прервать какой-то процесс, мы можем прописать утилиту **kill**, у которой в свою очередь есть различные параметры (QUIT, INT и т.д.). После неё нужно прописать данные о процессе, что мы хотим прервать, которые можно получить с помощью команды **ps –l**.

*7) Перечислите базовые средства взаимодействия процессов в Linux.*

Среди базовых средств взаимодействия процессов в Linux можно выделить такие средства, как:

> семафоры

> разделяемая память и ресурсы

> очереди сообщений

*8) Поясните особенности работы с каналами в Linux.*

Каналы в Linux (pipe) могут отправлять вывод одних команд в другие. Она также используются для перенаправления стандартного вывода, ввода или ошибки одного процесса другому процессу для дальнейшей её обработки.

*9) Почему отложенные вызовы не обрабатываются непосредственно обработчиком прерывания таймера?*

Отметим, что отложенные вызовы определяют те функции, вызовы которых будут произведены ядром системы лишь через некоторое время. Ядро производит вызов функции в системном контексте, так что функция отложенного вызова не должна обращаться к адресному пространству текущего процесса (она не имеет к нему никакого отношения), и в то же время не должна переходить в состояние сна. Так что, отложенные вызовы не выполняются в контексте процесса, ибо их выполнение осуществляется на уровне операционной системы.