БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Факультет прикладной математики и информатики

Веренич Владислав Николаевич

ОТЧЕТ ПО ПРОГРАММИРОВАНИЮ МОБИЛЬНЫХ И ВСТРАИВАЕМЫХ СИСТЕМ

студента 3 курса 12 группы

Лабораторная работа №3

Преподаватель

Давидовская М.И.

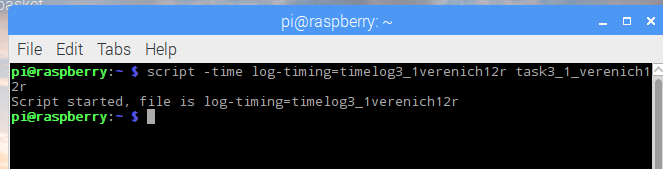
Минск 2019

**Цель работы** – изучение файловой системы ОС Unix/Linux и основных функций для работы с каталогами и файлами; исследование методов создания процессов, основных функций создания и управления процессами, обмена данными между процессами.

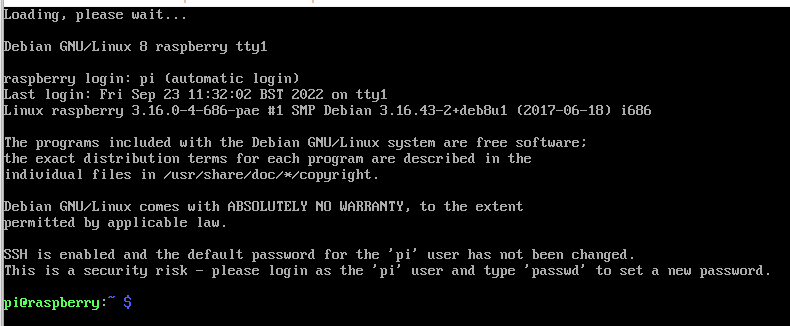
## Задание 3.1. Управление процессами

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 предыдущихлабораторных работ. Войдите в систему на виртуальных терминалах 1 и 2 (tty1, tty2) под вашей учетной записью.



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

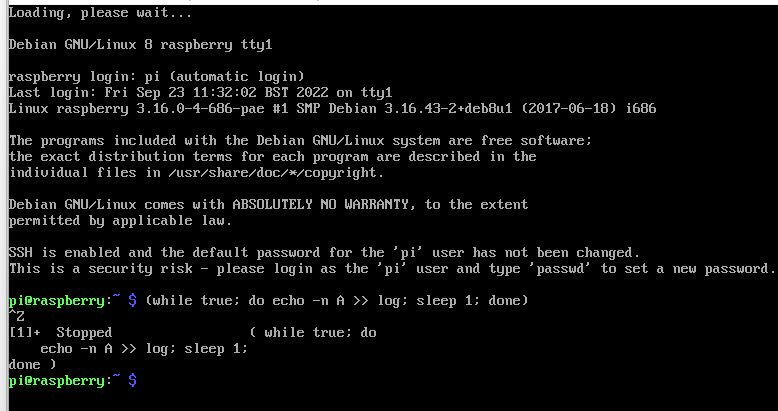
$ (while true; do echo -n A >> log; sleep 1; done)

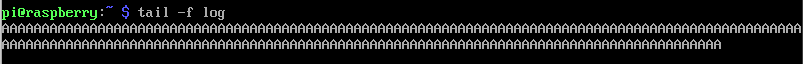
1. Заметьте, что сейчас этот терминал занят исполнением запущенногопроцесса, который исполняется на переднем плане. Этот процесс присоединяет символ "А"к файлу ~/log через каждую секунду. Чтобы визуально проверить это, переключитесь в виртуальный терминал tty2 и выполните следующую команду: $ tail -f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



Вы должны увидеть последовательность символов, длина которой возрастает.

1. 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.



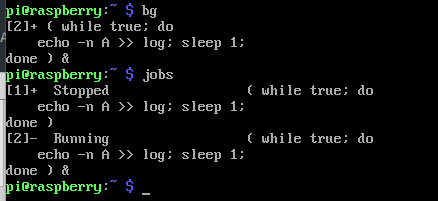


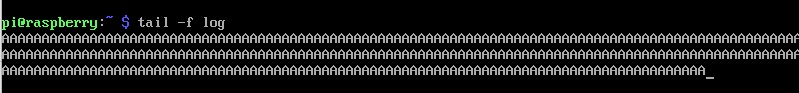
1. Переключитесь в виртуальный терминал tty1 и возобновите работу процесса в фоновом режиме. Используйте команду jobs, чтобы проверить, что задание [1] снова работает.

$ bg

$ jobs

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



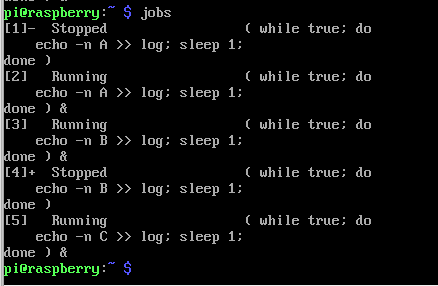


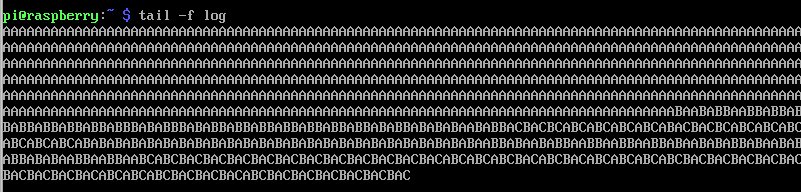
1. Переключитесь в виртуальный терминал tty1 и запустите еще два процесса, выполнив следующие команды:

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

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

1. 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.

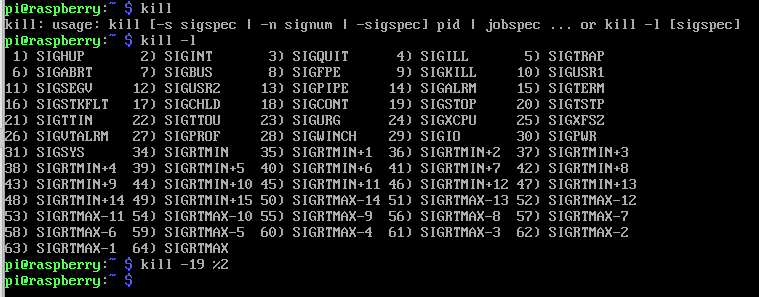




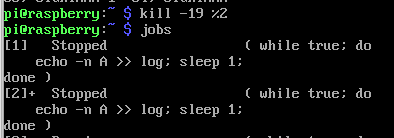
1. В пункте 4 вы приостановили исполнение процесса переднего плана путемнажатия клавиш <Ctrl+z>. В действительности эта комбинация нажатых клавиш посылает процессу сигнал. Используйте команду kill, чтобы получить список сигналов и соответствующие им имена и номера. Затем выполните команду kill, послав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

$ kill -l

$ kill -19 %1

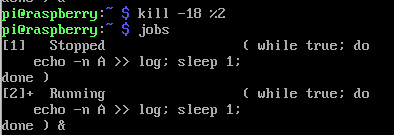


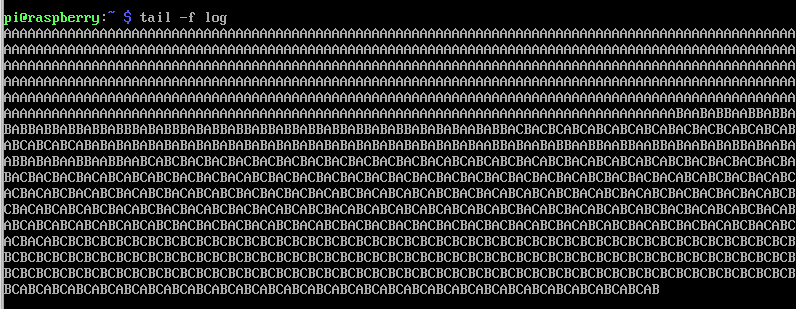
1. Выполните команду jobs и проверьте, что задание [1] остановлено. Переключитесь в виртуальный терминал tty2 и визуально проверьте, что задание [1] остановлено.





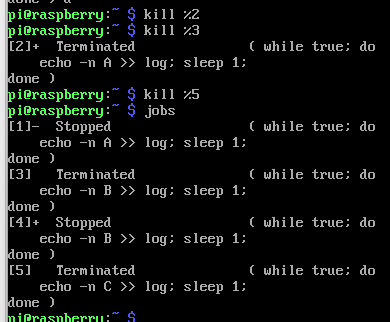
1. Возобновите выполнение задания [1], используя команду kill, которая посылает процессу сигнал SIGCONT (18). Используйте команду jobs и виртуальный терминал tty2 для проверки того, что все три задания опять работают.





1. 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:

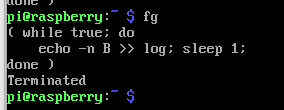
$ kill %2 %3 $ jobs



1. Чтобы завершить работу последнего процесса, выполните команды:

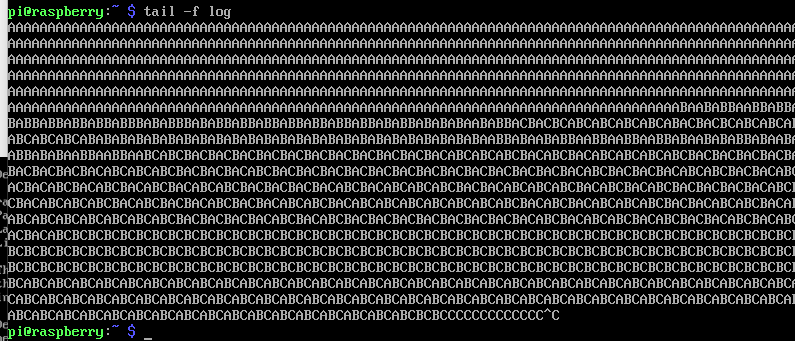
$ fg

$ <Ctrl+c>

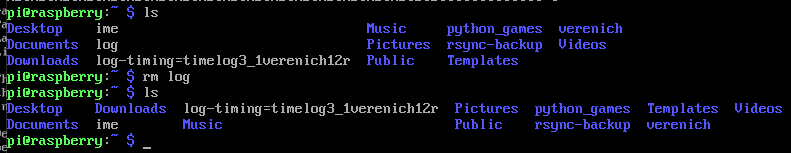


1. Выполните команду jobs и проверьте, что больше заданий не выполняется. Переключитесь в виртуальный терминал tty2 и визуально проверьте, что файл ~/log не увеличивается. Остановите исполнение команды tail, нажав клавиши <Ctrl+c>, и завершите сеанс на виртуальном терминале tty2.





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



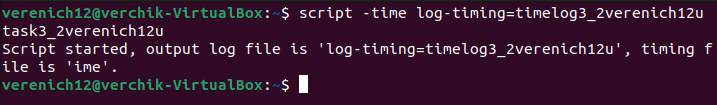
Завершаю ведение протокола



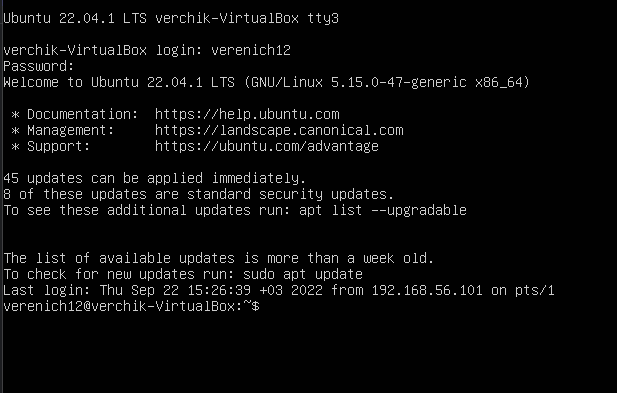
## Задание 3.2. Управление процессами

Изучите примеры задания 3.2 и выполните их в ОС Ubuntu.

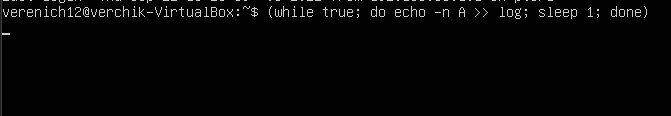
Включите ведение протокола командой script с журналом меток времени. **Протокол** назвать по следующему шаблону — taskXФамилияNM, где X — номер выполняемого задания, Фамилия — заменить на вашу фамилию латиницей и строчными буквами, N — номер группы, например 12 или 13, M — **u** — для Ubuntu. **Журнал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



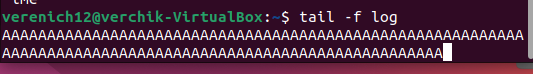
1. Войдите в систему на виртуальных терминалах 1 и 2 (tty1, tty2) под учетной своей записью.



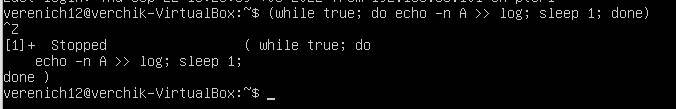
1. Переключитесь в терминал tty1 и запустите на переднем плане бесконечный процесс.



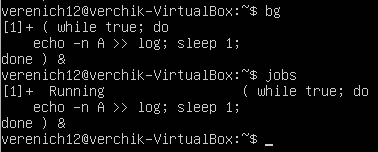
1. Переключитесь в виртуальный терминал tty2 и проверьте работу процесса, запущенного на пункте 2.



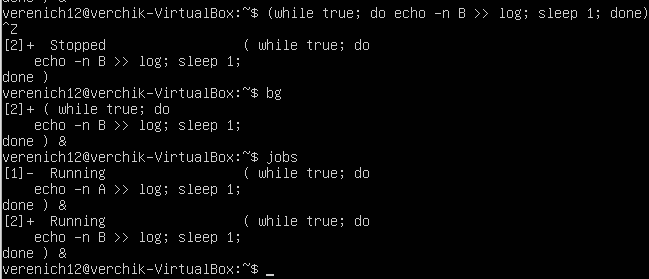
1. Переключитесь в виртуальный терминал tty1 и приостановите работающий процесс.



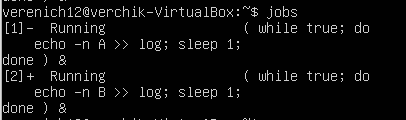
1. Возобновите работу процесса в виртуальном терминале tty1 в фоновом режиме.



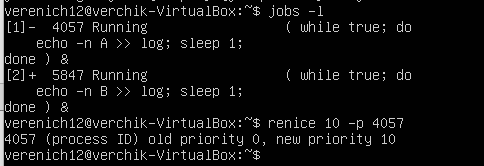
1. Запустите в виртуальном терминале tty1 второй бесконечный процессов запущенных в фоновом режиме.



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



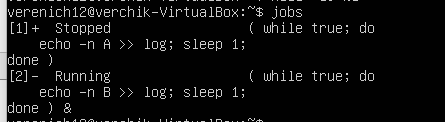
1. Установите приоритет процесса, запущенного в пункте 2, равным 10.



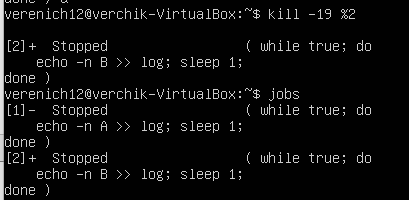
1. Остановите процесс, запущенный в пункте 6, командой kill.



1. Проверьте, что работает только один процесс.



1. Прервите исполнение работающего процесса и проверьте, что не осталосьработающих процессов.



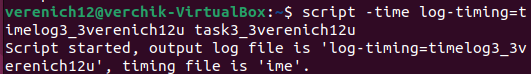
Завершаю ведение протокола



## Задание 3.3. Управление процессами и планирование заданий

Изучите примеры задания 3.3 и выполните их в ОС Ubuntu.

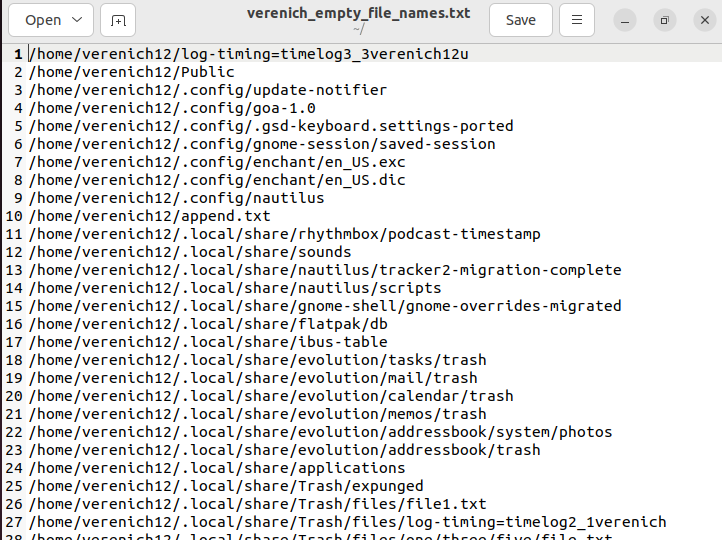
Включите ведение протокола командой script с журналом меток времени. **Протокол** назвать по следующему шаблону — taskXФамилияNM, где X — номер выполняемого задания, Фамилия — заменить на вашу фамилию латиницей и строчными буквами, N — номер группы, например 12 или 13, M — **u** — для Ubuntu. **Журнал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

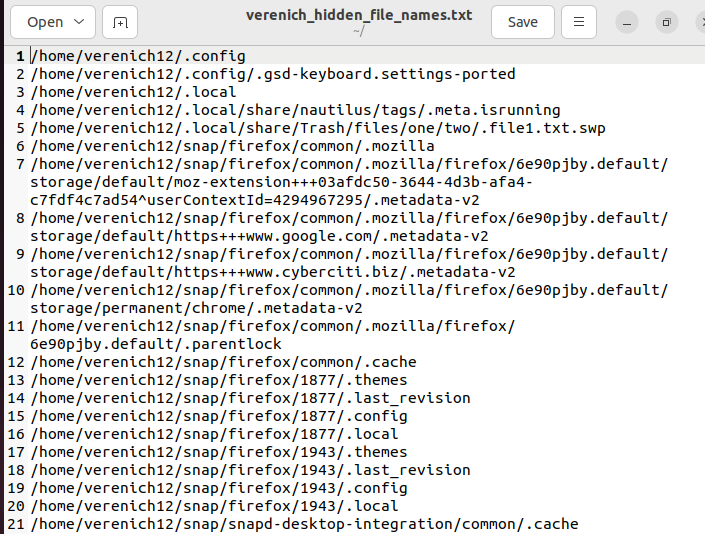
а) пустые



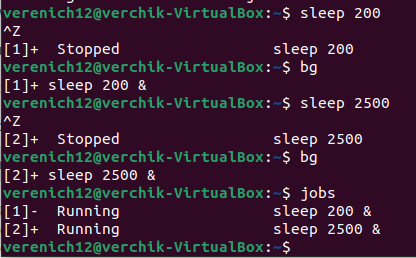


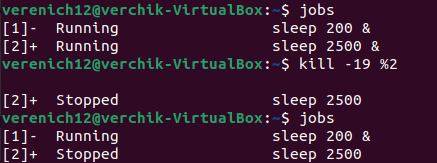
б) скрытые в домашнем каталоге в фоновом режиме и результат сохраните в файл со своей фамилией.





1. 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.



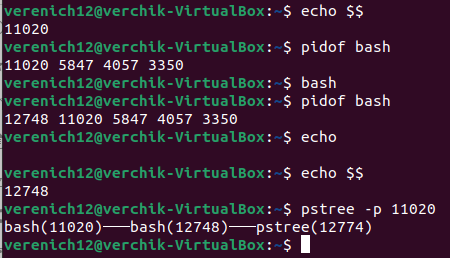


1. Выполните команду exec ls -R /etc. Изучите её поведение.

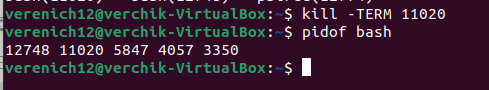
Команда ls -R /etc рекурсивно выводит содержимое каталога etc. Причем создается дочерний процесс, который и выполняет эту команду.

При добавлении exec shell-процесс заменяется на ls-процесс. Поэтому после выполнения данной команды окно терминала пропадает.

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



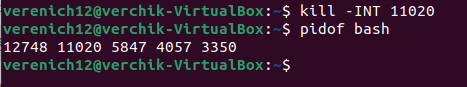
**Сигнал TERM** является перехватываемым и блокируемым и предназначен для корректного (предпочтительного) завершения работы процесса.



Как мы можем увидеть, посылание сигнала TERM родительскому процессу ничего не сделало.

**Сигнал INT** в отличии от **TERM** является блокируемым сигналом и перехватываемым. Самый безобидный сигнал.

Его выполнение тоже ничего не поменяло:

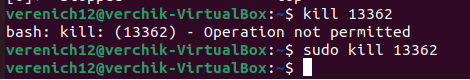


**Сигнал QUIT** - похож на **TERM**, но позволяет сохранить дамп памяти. Программа может выполнить корректное завершение или проигнорировать сигнал.

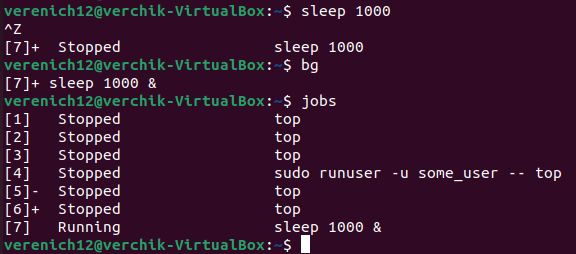


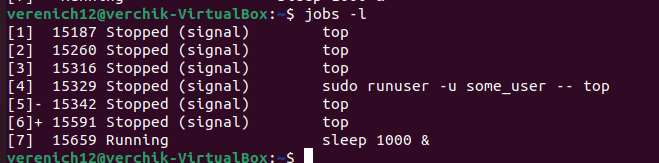
**Сигнал HUP** - сообщает процессу, что соединение с управляющим терминалом разорвано, единственный сработавший.

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

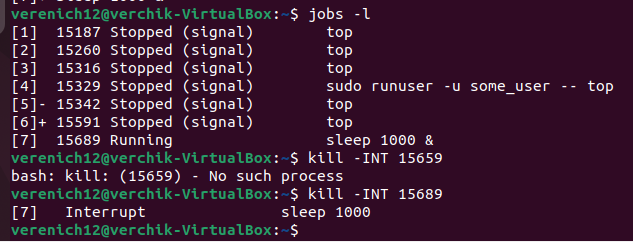


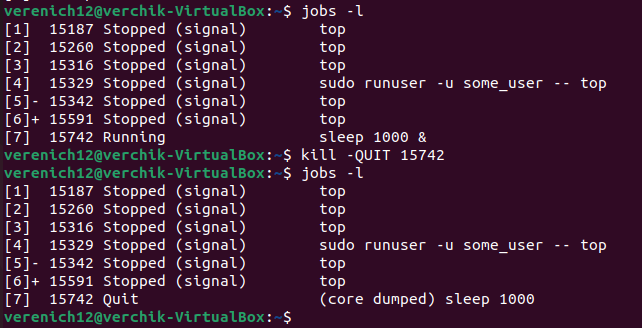
1. Запустите в фоновом режиме команду sleep 1000. Проверьте, на какие сигналы из следующих: TERM, INT, QUIT и HUP, реагирует эта команда.

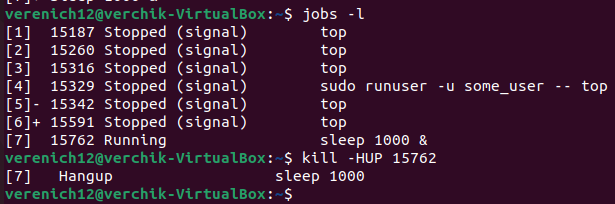






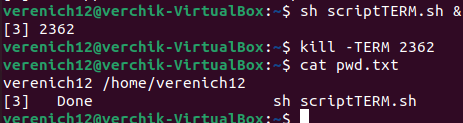


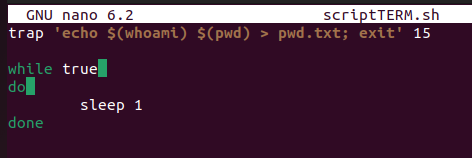




Реагирует на все сигналы

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





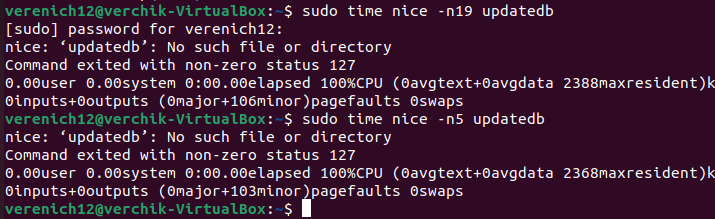
1. Запустите порожденную оболочку. Работает ли в ней созданный обработчик?

Так точно

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

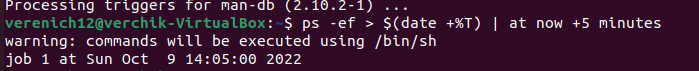


10.От имени суперпользователя запустите команду индексирования базы данных поиска вследующем виде: time nice -n19 updatedb. Затем выполните такую же команду, в которой значение nice number для updatedb будет 5. Сравните полученные результаты.

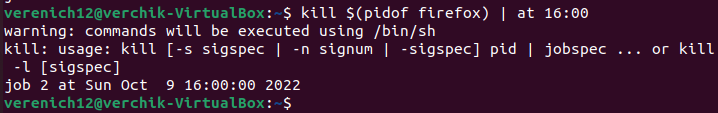


Сравнил)

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.

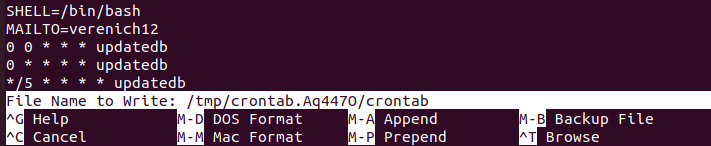


12.При помощи команды at организовать обычное завершение работы браузера firefox или chrome в 16:00.



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

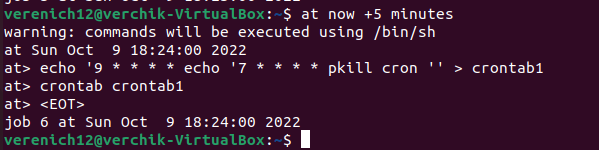
$crontab -e



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.

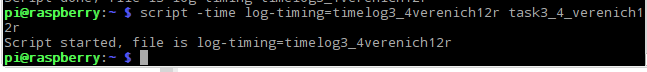


## Задание 3.4. Управление процессами и планирование заданий

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

# there was #‘:’ instead # of ‘;’

# Thanks a #lot for good # example

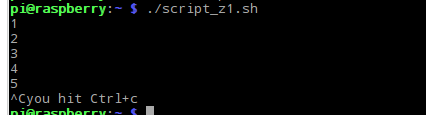
while;

do sleep 1

count=$(expr $count + 1)

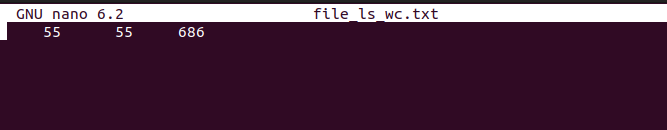
echo $count

done

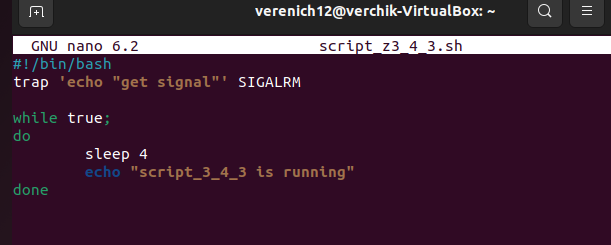


**Задача 2.** Написать пример программы, которая запускает и связывает каналом два процесса: вывод содержимого каталога и подсчет количества строк (ls и wc).

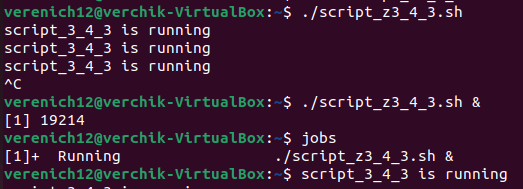




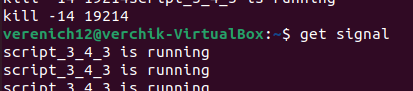
**Задача 3.** Изучить параграф **10.10. «Функции alarm и pause»** из книги [У. Ричард Стивенс, Стивен А. Раго. UNIX. Профессиональное программирование. 3-е изд](https://edufpmi.bsu.by/mod/url/view.php?id=902). Написать пользовательскую функцию обработки сигнала. Установка обработки сигнала происходит одноразово (обрабатывается только одно событие, связанное с появлением данного сигнала SIG\_ALRM). Возврат из функции-обработчика происходит в точку прерывания процесса.



Запускаем процесс в фоновом режиме:



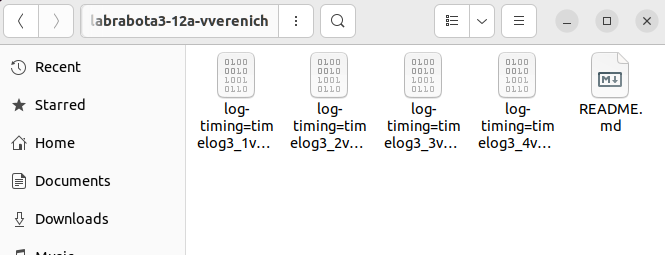
Проверяем, как он реагирует на сигнал:

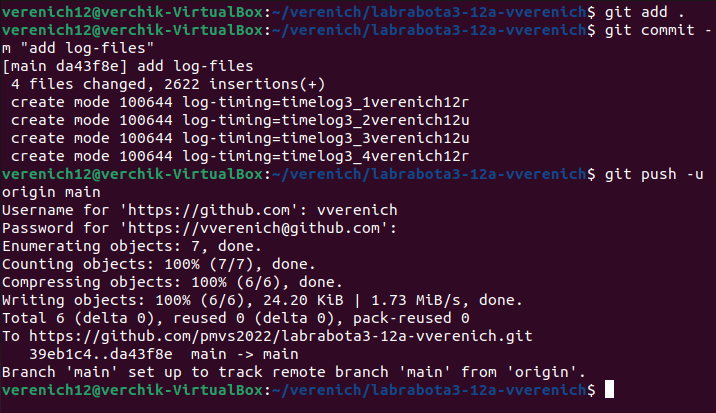


Завершаем протоколирование:



Заливаем лог-файлы на гитхаб





## 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](https://edufpmi.bsu.by/mod/url/view.php?id=902).

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

1. Написать программу поиска одинаковых по их содержимому файлов в двухкаталогов, например, Dir1 и Dir2. Пользователь задаёт имена Dir1 и Dir2. В результате работы программы файлы, имеющиеся в Dir1, сравниваются с файлами в Dir2 по их содержимому. Процедуры сравнения должны запускаться с использованием функции fork() в отдельном процессе для каждой пары сравниваемых файлов. Каждый процесс выводит на экран свой pid, имя файла, число просмотренных байт и результаты сравнения. Число запущенных процессов в любой момент времени не должно превышать N (вводится пользователем). Проверить работу программы для каталога /usr/include/ и любого другого каталога в /home N=6.

ВМЕСТО ПРОЦЕССОВ ПОТОКИ

# Контрольные вопросы

1. Объясните понятия процесса и ресурса. Какое их значение в организациивычислительного процесса в ОС Linux?

**Процесс** – набор выполняемых заданий, инструкций. **Ресурс** – это то, что выделяется процессу для выполнения данных задач. Процесс представлен в памяти в виде дескриптора процесса, который включает: исполняемый текст программы и управляющая информация. **Концепция процессов** обеспечивает свойство многозадачности ОС, т.е. когда в активном состоянии могут находиться несколько задач, между которыми разделяется процессорное время.

1. Какая информация содержится в описателях процессов?

**Описатель процесса** – это дескриптор. Дескриптор включает в себя:

исполняемый текст программы задачи;

управляющую информацию (контекст процесса), необходимую для выделения и использования выделенных процессу ресурсов: приоритет, необходимое адресное пространство, вектор прерывания (слово состояния программы), величину кванта процессорного времени, признаки состояния процесса и пр.

Посмотреть можно так: ps -l, ps.

1. Какими способами можно организовать выполнение программ в фоновом режиме?

Для запуска фонового (параллельного с другими потомками) процесса в командную строку необходимо и достаточно последним символом добавить знак ***& (амперсанд):***

*$ cc prog.c &*

*[1]2388*

*$*

Shell выводит номер этого процесса (PID) и разрешает ввод следующей команды.

Фоновые процессы обладают некоторыми недостатками:

- не допускают ввода с клавиатуры;

- обеспечивают вывод на экран, но при этом нарушают целостность вывода диалогового процесса.

Либо так: bg %номер\_задания

1. 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?

Предыдущий вопрос + это:

Общепринятый прием **исключения влияния фонового вывода на интерактивную работу:**

***$<командная\_строка> > имя\_файла.out &***

- <командная\_строка> планирует задание для фонового режима;

- перенаправляет вывод вместо экрана в указанный файл головного каталога пользователя.

Пример:

*$grep ааа\* > grep.out &*

*[2]194*

*$ps*

*PID TTY TIME CMD*

*194 tty5 0:02 grep*

*200 tty5 0:01 ps*

Особенности работы с фоновым режимом:

- выполняемая в фоновом режиме программа (команда), требующая стандартного ввода, должна читать его из файла с использованием перенаправленного ввода;

- программа, выполняемая в фоновом режиме, не может быть прервана <Ctrl\*C>, так как она отсоединяется от клавиатуры и может быть прекращена только с помощью команды kill или выходом из системы;

- выход из системы exit надо выполнять два раза: для завершения фонового процесса и завершения основного процесса shell.

Выполнение фоновых заданий прекращается с выходом пользователя из системы. НО! Если фоновая программа должна быть продолжена и после прекращения текущего сеанса работы, то необходимо использовать команду:

***$nohup имя\_фоновой\_программы &*— команда, во-первых, запускает и защищает фоновую программу от прерываний, вырабатываемых при выходе пользователя из системы, и, во-вторых, перенаправляет фоновый протокол в системный файл nohup.out.**

Вместе с тем, вывод протокола работы фоновой программы можно **перенаправить в специальный файл, который можно просмотреть позже без нарушения протокола работы с оперативной задачей:**

***$nohup имя\_фон\_программы>имя\_файла&***

1. Как пользователь может повлиять на распределение ресурсов между активными процессами?

С помощью команды nice.

Если при выполнении задания образуются несколько порожденных процессов, то все они имеют одинаковый приоритет равный родительскому. В этом случае все процессы получают ресурсы равными долями (простой режим разделения времени). При необходимости выделения наиболее важных родительских процессов порожденным второстепенным можно понизить приоритет с помощью команды:

**nice [-k] имя\_программы выполнение программы, указанной в строке, с пониженным приоритетом.**

-k - к-т понижения приоритета (k = 1;...10; по умолчанию k = 10).

**Пример 1.**

$ prog1 &

$ ps -al

........ PID ...... PRI NI......CMD..

................... 20 sh

.................. . 20 prog1........

т.е. родительский и порожденный процессы имеют приоритет=20

**Пример 2.**

$ nice -5 prog2 &

[2] 3752

$ps -flu lev

........ PID ...... PRI .....NI.. CMD

.................. . 20 ........ sh

........ 3752 ..... 25 .......5 prog2.

................... 20 .. ......

Приоритет процесса задачи prog2 понижен на 5 единиц, тем самым другие процессы этой группы больше возможностей в использовании ресурсов.

1. Как можно прервать выполнение активных процессов? Какая информациядля этого необходима и откуда она извлекается?

Завершение процессов – одна из функций управления процессами. Прекратить выполнение любого процесса можно с помощью команды:

***kill [-опции] PID1 [PID2......]*— передает сигнал процессу PID.**

**Сигнал —**ключевое слово, при получении которого процесс выполняет некоторые действия. Сигналы с использованием команды kill могут передаваться другими прикладными процессами или системными программами (например, при появлении некоторых событий, как то сбой в канале, сигнала с таймера, завершения фонового процесса и пр.). Существуют двадцать пять видов сигналов, предназначенных для выполнения различных действий процессами при наступлении определенных событий в системе. С получением большинства сигналов процесс завершается самостоятельно. Тот, кто посылает сигнал, должен быть владельцем процесса или администратором. Для безусловного и немедленного завершения указанного процесса kill должен послать сигнал с именем TERM (по умолчанию). Другие сигналы передаются с помощью опции **-S**.

Значения опций:

*-S<имя\_сигнала>*или

*-№*— системный номер сигнала;

-*l —*вывод на экран справочника имен сигналов.

Сигнал определяет дальнейшее действие процесса. Это еще один способ управления процессами .

Примеры опций:

*-S KILL*— немедленное завершение процесса по сигналу KILL;

-9 — то же, но уже по номеру сигнала KILL.

Пример:

*$kill [-9] 3752*

*3752: killed*

*$*

Пример завершения фонового процесса:

*$kill %1*— указывается номер завершаемого задания

*[1] +Terminated inf*

*$*

Номер процесса получаем по команде ps либо номер задания по команде jobs.

1. Перечислите базовые средства взаимодействия процессов в Linux.

Именованные каналы, общая память, семафоры.

1. Поясните особенности работы с каналами в Linux.

Каналы бывают именованные и неименованные (анонимные).

Анонимные каналы мы использовали для перенаправления ввода-вывода.

В отличие от анонимного программного канала, автоматически создаваемого шеллом, именованный канал обладает именем, и создается явно при помощи команд mknod или mkfifo.

1. Почему отложенные вызовы не обрабатываются непосредственно обработчиком прерывания таймера?

Функции отложенных вызовов выполняются в системном контексте, а не в контексте прерывания. Вызов этих функций выполняется не обработчиком прерывания таймера, а отдельным обработчиком отложенных вызовов, который запускается после завершения обработки прерывания таймера. При обработке прерывания таймера система проверяет необходимость запуска тех или иных функций отложенного вызова и устанавливает соответствующий флаг для них. В свою очередь обработчик отложенных вызовов проверяет флаги и запускает необходимые в системном контексте.