

Липецкий государственный технический университет

Факультет автоматизации и информатики

Кафедра Автоматизированных систем управления

Отчет по лабораторной работе № 2

«Процессы в операционной системе Linux»

по курсу «ОС Linux»

Студент
Группа ПМ-18

Полухина Е.Д.

Руководитель

Кургасов В.В.

Липецк 2020 г.

СОДЕРЖАНИЕ

Цель работы	4
Задание.....	5
Ход работы	7
1. Загрузка пользователем user.	7
2. Поиск файла с образом ядра.	8
3. Перечень процессов.	9
4. Два сценария loop и loop2.	10
5. Запуск loop2 на переднем плане.	11
6. Остановка процесса loop2.	12
7. Просмотр запущенных процессов несколько раз с помощью ps -f.....	13
8. Kill loop2.	14
9. Запуск в фоне процесса loop.	15
10. Завершение процесса loop.....	16
11. Запуск процесса в фоне и его завершение.	17
12. Запуск экземпляра оболочки.....	18
13. Запуск нескольких процессов в фоне.	19
14. Запуск трех задач.....	21
15. Перевод задачи в фоновый режим.....	22
16. Эксперименты по переводу задач из режимов.	23
17. Создание именованного канала для архивирования и осуществление передачи в канал.....	24
18. Отображение информации о процессах, начиная с указанного PID, с выделением цветом текущего процесса и его предков.	26

19. Завершение выполнения процесса с помощью сигнала SIGINT двумя способами.	27
20. Просмотр приоритета редактора nano и его увеличение на 2.....	28
Вывод	30

Цель работы

Ознакомиться на практике с понятием процесса в операционной системе. Приобрести опыт и навыки управления процессами в операционной системе Linux.

Задание

1. Загрузиться не root, а пользователем.
2. Найти файл с образом ядра. Выяснить по имени файла номер версии Linux.
3. Посмотреть процессы `ps -f`. Прокомментировать. Для этого почитать `man ps`.
4. Написать с помощью редактора `vi` два сценария `loop` и `loop2`. Текст сценариев:

Loop:

```
while true; do true; done
```


Loop2:

```
while true; do true; echo 'Hello'; done
```
5. Запустить `loop2` на переднем плане: `sh loop2`.
6. Остановить, послав сигнал `STOP`.
7. Посмотреть последовательно несколько раз `ps -f`. Записать сообщение, объяснить.
8. Убить процесс `loop2`, послав сигнал `kill -9 PID`. Записать сообщение. Прокомментировать.
9. Запустить в фоне процесс `loop`: `sh loop&`. Не останавливая, посмотреть несколько раз: `ps -f`. Записать значение, объяснить.
10. Завершить процесс `loop` командой `kill -15 PID`. Записать сообщение, прокомментировать.
11. Третий раз запустить в фоне. Не останавливая убить командой `kill -9 PID`.
12. Запустить еще один экземпляр оболочки: `bash`.
13. Запустить несколько процессов в фоне. Останавливать их и снова запускать. Записать результаты просмотра командой `ps -f`.
14. Запустить в консоли на выполнение три задачи, две в интерактивном режиме, одну - в фоновом.

15. Перевести одну из задач, выполняющихся в интерактивном режиме, в фоновый режим.

16. Провести эксперименты по переводу задач из фонового режима в интерактивный и наоборот.

17. Создать именованный канал для архивирования и осуществить передачу в канал

- списка файлов домашнего каталога вместе с подкаталогами (ключ -R),
- одного каталога вместе с файлами и подкаталогами.

18. В отчете предоставьте все шаги ваших действий. То есть следует привести следующее: текст задания, а следом за ним снимок экрана консоли с результатами выполнения задания. Кроме того, перед скриншотом следует привести текстовую запись использованных команд.

Вариант 6.

1. Отобразить информацию о процессах, начиная с указанного идентификатора, с выделением цветом текущего процесса и его предков.

2. Завершить выполнение процесса, владельцем которого является текущий пользователь, с помощью сигнала SIGINT двумя способами: задав имя сигнала и используя комбинацию клавиш.

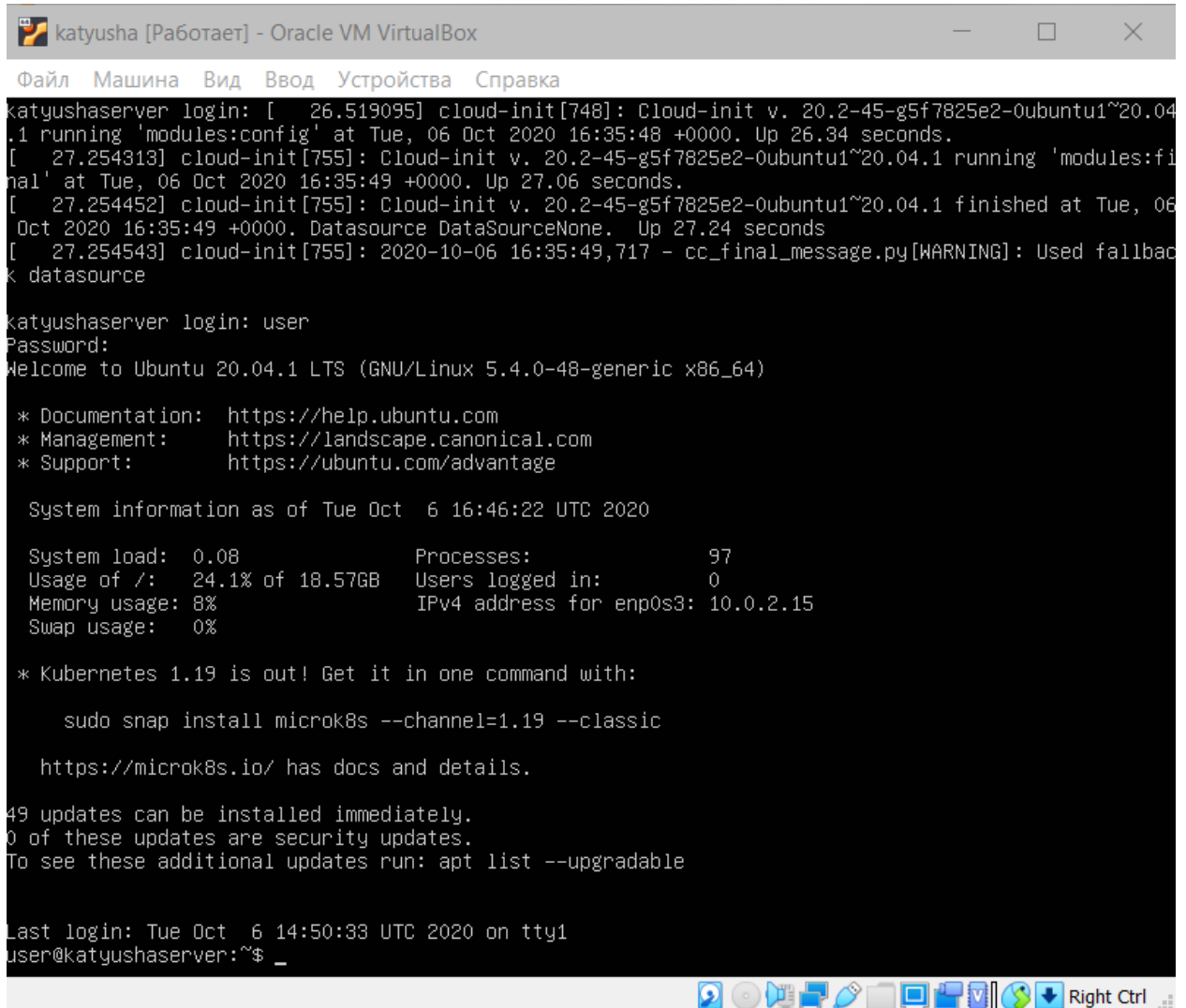
3. Запустите редактор nano, определите приоритет редактора. Запустите новый процесс данного редактора с увеличенным на 2 значением приоритета.

4. В отчете предоставьте все шаги ваших действий. То есть следует привести следующее: текст задания, а следом за ним снимок экрана консоли с результатами выполнения задания. Кроме того, перед скриншотом следует привести текстовую запись использованных команд. Кратко поясните результаты выполнения всех команд.

Ход работы

1. Загрузка пользователем user.

На рисунке 1 представлен терминал после запуска виртуальной машины с Linux Ubuntu и загрузки пользователем user.



```
katyusha [Работает] - Oracle VM VirtualBox
Файл  Машина  Вид  Ввод  Устройства  Справка
katyushaserver login: [ 26.519095] cloud-init[748]: Cloud-init v. 20.2-45-g5f7825e2-0ubuntu1~20.04
.1 running 'modules:config' at Tue, 06 Oct 2020 16:35:48 +0000. Up 26.34 seconds.
[ 27.254313] cloud-init[755]: Cloud-init v. 20.2-45-g5f7825e2-0ubuntu1~20.04.1 running 'modules:fi
nal' at Tue, 06 Oct 2020 16:35:49 +0000. Up 27.06 seconds.
[ 27.254452] cloud-init[755]: Cloud-init v. 20.2-45-g5f7825e2-0ubuntu1~20.04.1 finished at Tue, 06
Oct 2020 16:35:49 +0000. Datasource DataSourceNone. Up 27.24 seconds
[ 27.254543] cloud-init[755]: 2020-10-06 16:35:49,717 - cc_final_message.py[WARNING]: Used fallback
k datasource

katyushaserver login: user
Password:
Welcome to Ubuntu 20.04.1 LTS (GNU/Linux 5.4.0-48-generic x86_64)

* Documentation:  https://help.ubuntu.com
* Management:    https://landscape.canonical.com
* Support:       https://ubuntu.com/advantage

System information as of Tue Oct  6 16:46:22 UTC 2020

System load:  0.08           Processes:           97
Usage of /:   24.1% of 18.57GB Users logged in:          0
Memory usage: 8%           IPv4 address for enp0s3: 10.0.2.15
Swap usage:   0%

* Kubernetes 1.19 is out! Get it in one command with:

    sudo snap install microk8s --channel=1.19 --classic

https://microk8s.io/ has docs and details.

49 updates can be installed immediately.
0 of these updates are security updates.
To see these additional updates run: apt list --upgradable

Last login: Tue Oct  6 14:50:33 UTC 2020 on tty1
user@katyushaserver:~$ _
```

Рисунок 1 – Загрузка пользователем user

2. Поиск файла с образом ядра.

На рисунке 2 представлен поиск файла с образом ядра с помощью команды `find` и просмотр версии, которая используется в данный момент с помощью команды `uname -r`.

```
user@katyushaserver:/boot$ find / -name "vmlinuz*" 2>/dev/null
/boot/vmlinuz-5.4.0-51-generic
/boot/vmlinuz.old
/boot/vmlinuz
/boot/vmlinuz-5.4.0-48-generic
user@katyushaserver:/boot$ uname -r
5.4.0-51-generic
user@katyushaserver:/boot$
```

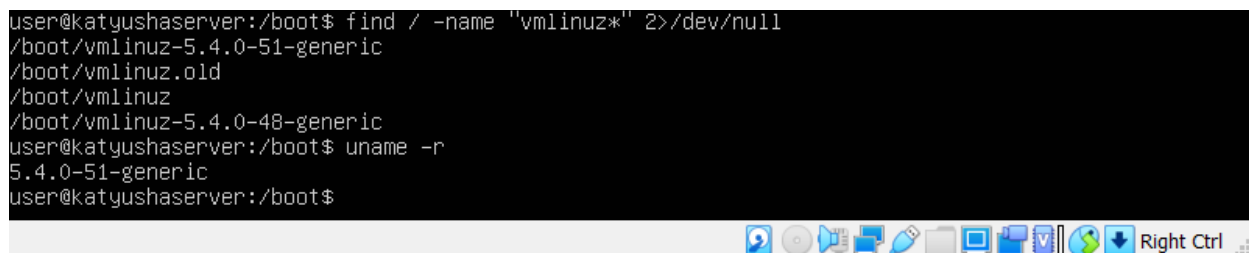


Рисунок 2 – Поиск файла с образом ядра

Название файла с образом ядра – `vmlinuz-5.4.0-51-generic`.
Следовательно, версия ядра – 5.4.0-51.

Первая цифра – это мажорный номер версии, у нас – это 5, 4 – минорная версия, цифра 0 – это номер ревизии, а 51 – это уже относится к номеру сборки от разработчиков дистрибутива.

3. Перечень процессов.

На рисунке 3 показан список процессов, запущенных в текущей командой оболочке.

```
user@katyushaserver:/$ ps -f
UID          PID    PPID  C STIME TTY          TIME CMD
user         898      626  0 07:58 tty1        00:00:00 -bash
user         913      898  0 08:00 tty1        00:00:00 ps -f
user@katyushaserver:/$
```



Рисунок 3 – Перечень запущенных процессов

UID – пользователь, от имени которого процесс запущен.

PID (Process Identifier) – идентификатор процесса, который дает система для обозначения и различия процессов. Каждый процесс имеет свой уникальный идентификатор.

PPID – идентификатор родительского процесса.

C – расходование ресурсов процессора в процентах.

STIME – время запуска процесса.

TTY – отображает название терминала, к которому подключен указанный процесс.

TIME – в этом поле указывается процессорное время, затраченное на выполнение процесса.

CMD – выводится название команды или же сервиса, которые запустили процесс.

Представлены оба принадлежащих пользователю user процесса: стартовый командный интерпретатор, bash, и выполняющийся ps. Оба процесса запущены с терминала tty1 и имеют идентификаторы 626 и 898 соответственно. В поле PPID указан идентификатор родительского процесса, то есть процесса, породившего данный. Для ps это – bash, а для bash, очевидно, login, так как именно он запускает стартовый shell. В выдаче не оказалось строки для этого login, равно как и для большинства других процессов системы, так как они не принадлежат пользователю user.


4. Два сценария loop и loop2.

На рисунке 4 показано создание сценария loop с помощью редактора vi. На рисунке 5 показано сообщение о том, что loop создан успешно. На рисунке 6 представлено написание текста сценария для loop2.



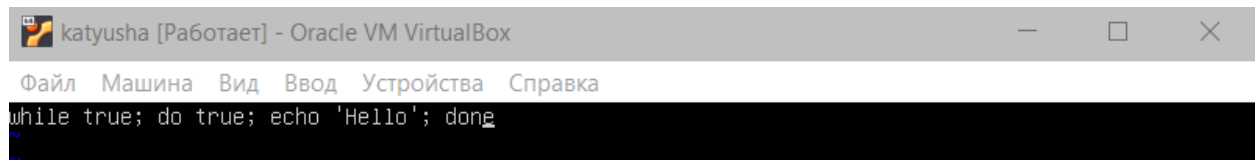
```
user@katyushaserver:~$ vi loop
```

Рисунок 4 – Создание сценария loop



```
"loop" [New] 1L, 26C written
user@katyushaserver:~$ _
```

Рисунок 5 – Редактирование сценария loop и его сохранение



katyusha [Работает] - Oracle VM VirtualBox

Файл Машина Вид Ввод Устройства Справка

```
while true; do true; echo 'Hello'; done
```

Рисунок 6 – Редактирование сценария loop2

5. Запуск loop2 на переднем плане.

Произведен запуск loop2 с помощью команды `sh loop2` (рис. 7).

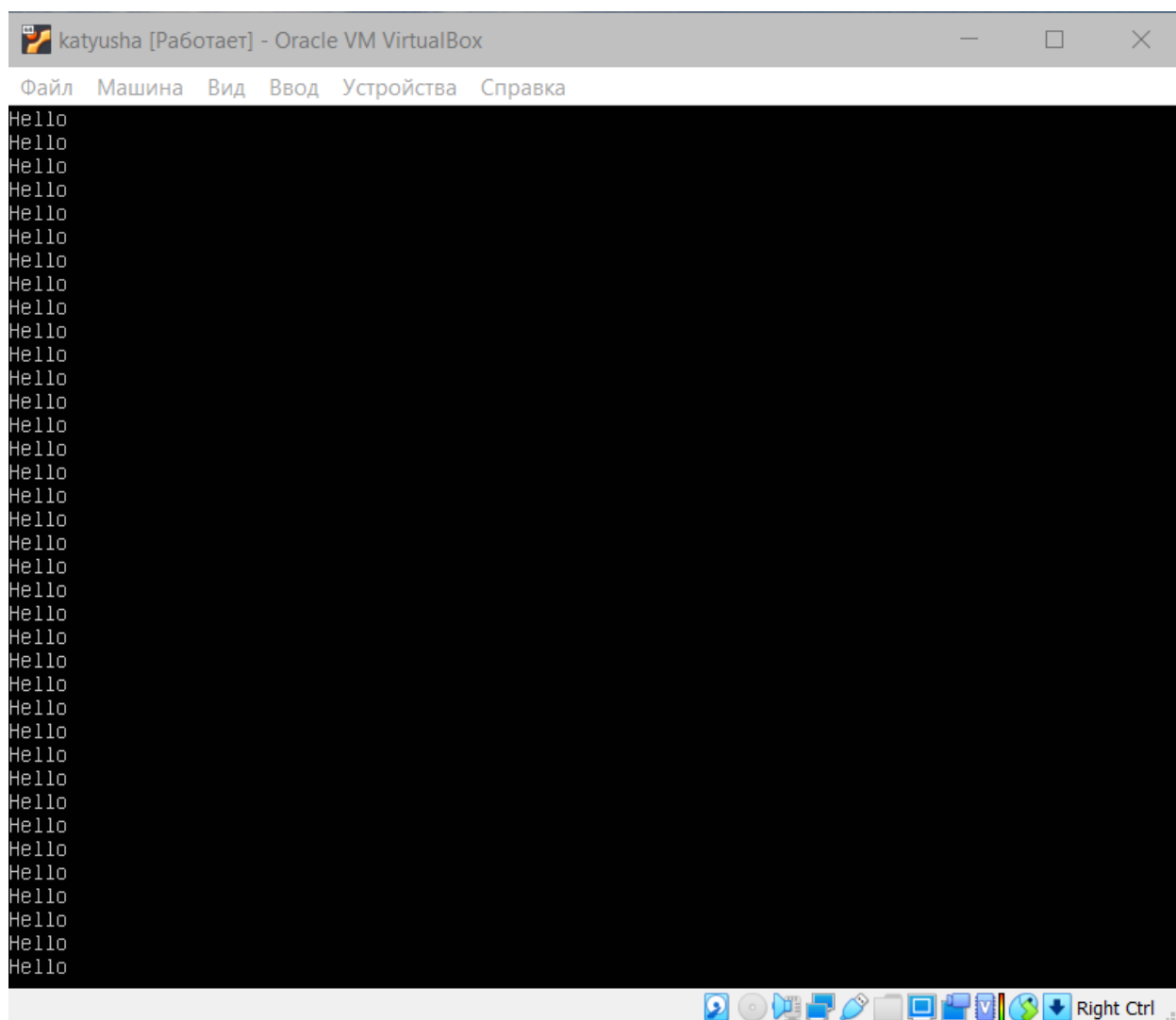


Рисунок 7 – Запуск loop2

6. Остановка процесса loop2.

На рисунке 8 показана остановка процесса loop2 с помощью сочетания клавиш ^Z (Ctrl + Z). Данный процесс ушел в фоновый режим.



```
Hello
Hello
Hello
^Z
[1]+  Stopped                  sh loop2
user@katyushaserver:~$
```

The screenshot shows a terminal window with a black background and white text. The text displays the output of a program printing 'Hello' three times, followed by the user pressing Ctrl+Z (^Z). The terminal then shows '[1]+ Stopped sh loop2', indicating the process has been suspended. The prompt 'user@katyushaserver:~\$' is visible at the bottom. The window's title bar and system tray are also visible.

Рисунок 8 – Остановка loop2

7. Просмотр запущенных процессов несколько раз с помощью `ps -f`.

На рисунке 9 показано два последовательных запуска команды `ps -f`. Делаем это, чтобы посмотреть запущенные процессы. Видно, что сценарий `loop` был запущен одну секунду. Также можно заметить, что со временем расходование ресурсов процессора в процентах, используемое процессом `loop2`, уменьшается.

```
user@katyushaserver:~$ ps -f
UID          PID    PPID  C STIME TTY          TIME CMD
user         907      624  0 08:32 tty1          00:00:00 -bash
user        934      907  2 08:37 tty1          00:00:01 sh loop2
user        936      907  0 08:38 tty1          00:00:00 ps -f
user@katyushaserver:~$ ps -f
UID          PID    PPID  C STIME TTY          TIME CMD
user         907      624  0 08:32 tty1          00:00:00 -bash
user        934      907  1 08:37 tty1          00:00:01 sh loop2
user        937      907  0 08:38 tty1          00:00:00 ps -f
user@katyushaserver:~$ _
```

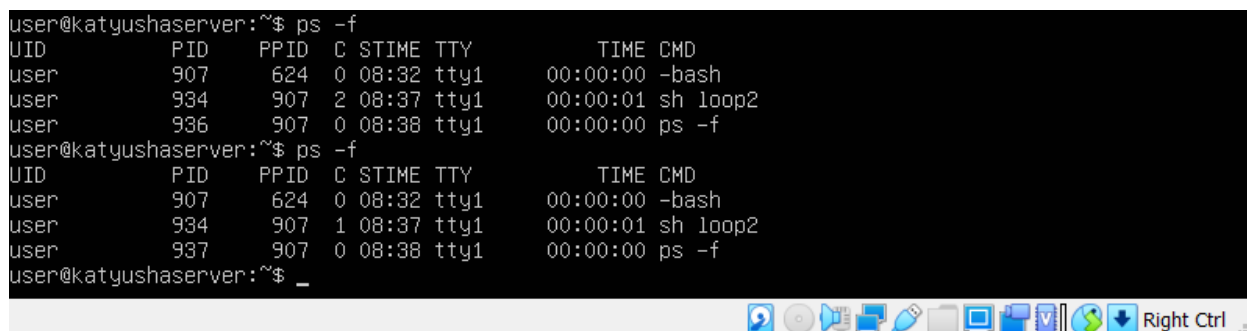


Рисунок 9 – Список запущенных процессов

8. Kill loop2.

Чтобы завершить процесс, используем команду `kill -9 PID`. Видим сообщение об успешном выполнении команды. Используя `ps -f` убеждаемся, что сценарий `loop2` завершен (рис. 10).

```
user@katyushaserver:~$ kill -9 934
[1]+  Killed                  sh loop2
user@katyushaserver:~$ ps -f
UID      PID     PPID  C  STIME TTY          TIME CMD
user      907      624  0  08:32 tty1        00:00:00 -bash
user      942      907  0  08:44 tty1        00:00:00 ps -f
user@katyushaserver:~$
```

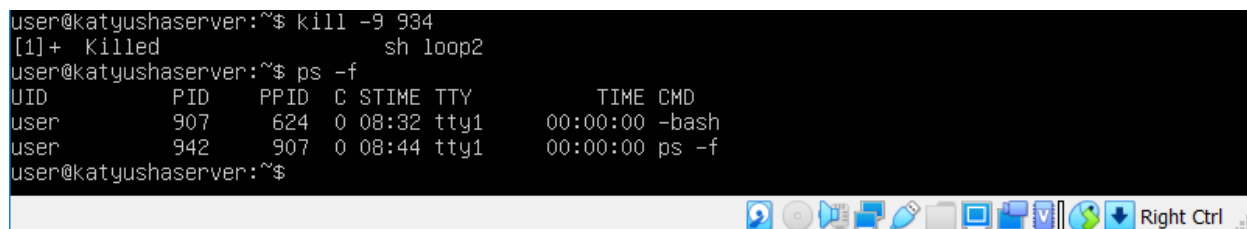


Рисунок 10 – Завершение процесса `loop2`

9. Запуск в фоне процесса loop.

На рисунке 11 показан запуск в фоне процесса loop с помощью команды `sh loop&` и просмотр запущенных процессов. Как можно увидеть, со временем сценарий loop расходует все больше ресурсов процессора, практически все.

```
user@katyushaserver:~$ sh loop&
[1] 922
user@katyushaserver:~$ ps -f
UID          PID    PPID  C STIME TTY          TIME CMD
user          908      629  0 12:45 tty1        00:00:00 -bash
user          922      908  97 12:47 tty1        00:00:12 sh loop
user          923      908  0 12:47 tty1        00:00:00 ps -f
user@katyushaserver:~$ ps -f
UID          PID    PPID  C STIME TTY          TIME CMD
user          908      629  0 12:45 tty1        00:00:00 -bash
user          922      908  99 12:47 tty1        00:01:02 sh loop
user          925      908  0 12:48 tty1        00:00:00 ps -f
user@katyushaserver:~$ ps -f
UID          PID    PPID  C STIME TTY          TIME CMD
user          908      629  0 12:45 tty1        00:00:00 -bash
user          922      908  99 12:47 tty1        00:02:15 sh loop
user          928      908  0 12:49 tty1        00:00:00 ps -f
user@katyushaserver:~$ ps -f
UID          PID    PPID  C STIME TTY          TIME CMD
user          908      629  0 12:45 tty1        00:00:00 -bash
user          922      908  99 12:47 tty1        00:03:57 sh loop
user          930      908  0 12:51 tty1        00:00:00 ps -f
user@katyushaserver:~$
```

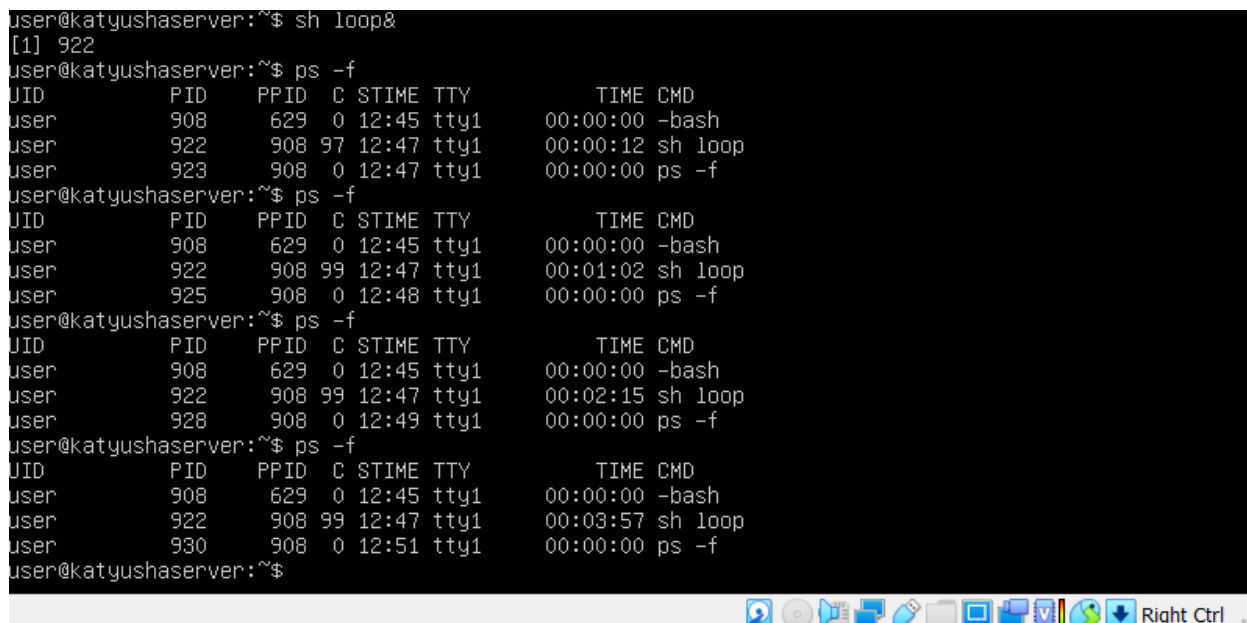


Рисунок 11 – Запуск сценария loop и просмотр запущенных процессов

10. Завершение процесса loop.

На рисунке 12 показано завершение выполнения сценария loop с помощью команды `kill -15 PID`. С помощью `ps -f` проверяем, что процесс действительно завершён. В отличие от `kill -9 PID`, сигнал `kill -15 PID` завершает выполнение процесса корректно, а не принудительно.

```
user@katyushaserver:~$ kill -15 922
user@katyushaserver:~$ ps -f
UID          PID     PPID  C  STIME TTY          TIME CMD
user          908       629  0  12:45 tty1        00:00:00 -bash
user          931       908  0  12:52 tty1        00:00:00 ps -f
[1]+  Terminated                sh loop
user@katyushaserver:~$ _
```

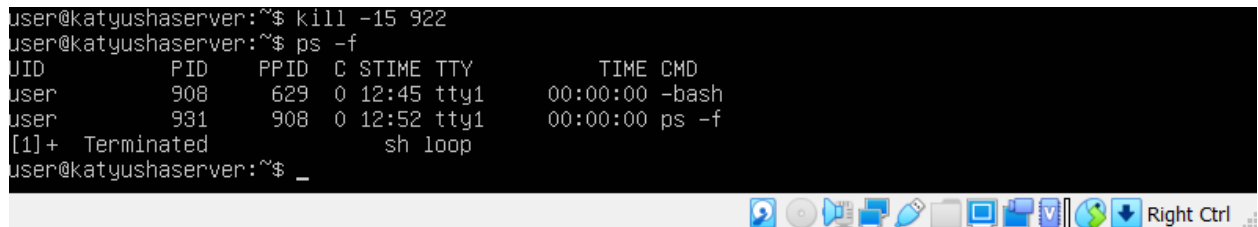


Рисунок 12 – Завершение процесса loop

11. Запуск процесса в фоне и его завершение.

Запускаем процесс `loop` в фоне, используя команду `sh loop&`, и, не останавливая, завершаем процесс, используя команду `kill -9 PID`. Сигнал `kill -9` всегда обрабатывает система. Смотрим запущенные процессы, чтобы убедиться, что сценарий больше не выполняется, и видим сообщение об успешном завершении процесса (рис. 13).

```
user@katyushaserver:~$ sh loop&
[1] 966
user@katyushaserver:~$ ps -f
UID          PID    PPID  C STIME TTY          TIME CMD
user          908      629  0 12:45 tty1          00:00:00 -bash
user          966      908  99 13:23 tty1          00:00:03 sh loop
user          967      908  0 13:23 tty1          00:00:00 ps -f
user@katyushaserver:~$ kill -9 966
user@katyushaserver:~$ ps -f
UID          PID    PPID  C STIME TTY          TIME CMD
user          908      629  0 12:45 tty1          00:00:00 -bash
user          968      908  0 13:23 tty1          00:00:00 ps -f
[1]+  Killed                  sh loop
user@katyushaserver:~$
```

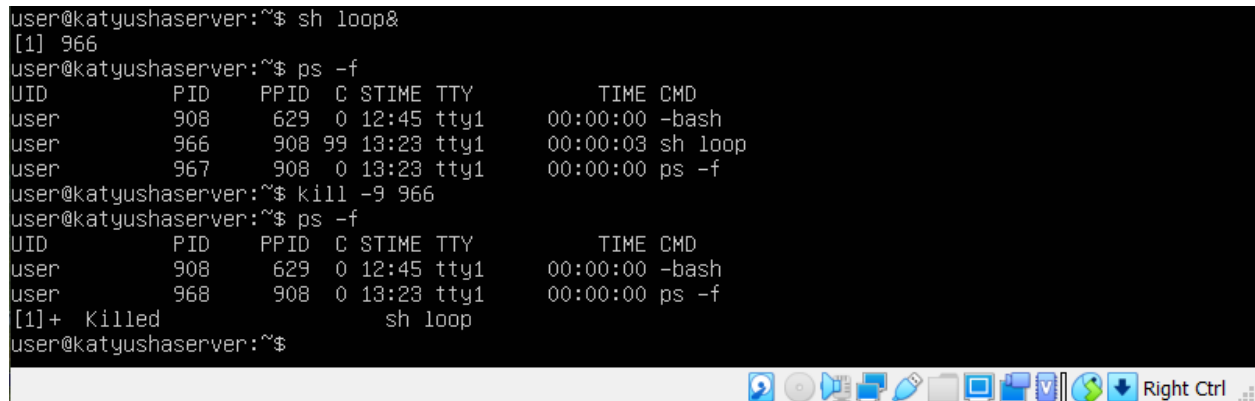
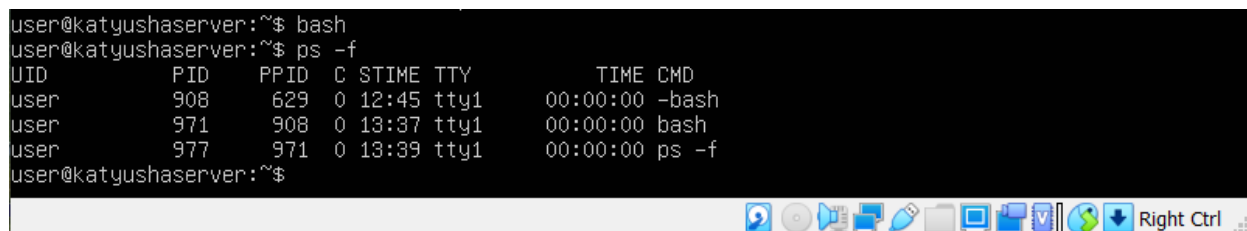


Рисунок 13 – Завершение выполнения `loop`

12. Запуск экземпляра оболочки.

На рисунке 14 показан запуск еще одного экземпляра оболочки с помощью команды `bash` и просмотр процессов с помощью команды `ps -f`. Как мы видим, идентификатор родительского процесса `PPID` у команды `ps -f` равен `PID` процесса `bash`.



```
user@katyushaserver:~$ bash
user@katyushaserver:~$ ps -f
UID          PID    PPID  C STIME TTY          TIME CMD
user         908     629  0 12:45 tty1        00:00:00 -bash
user         971     908  0 13:37 tty1        00:00:00 bash
user         977     971  0 13:39 tty1        00:00:00 ps -f
user@katyushaserver:~$
```

Рисунок 14 – Запуск экземпляра оболочки

13. Запуск нескольких процессов в фоне.

На рисунке 15 мы запускаем сразу три процесса `loop` и смотрим запущенные процессы, используя `ps -f`. Затем мы останавливаем процесс с `PID = 913`, используя команду `kill -19 PID`, и видим, что данный сценарий после остановки начал занимать меньше ресурсов процессора, чем занимал раньше. Затем мы отправляем сигнал `kill -18 PID`, чтобы продолжить процесс, и при этом останавливаем процесс с `PID = 914`. Видим, что продолженный процесс начал увеличивать расход ресурсов процессора, а остановленный уменьшать.

```
[1] 913
user@katyushaserver:~$ sh loop&
[2] 914
user@katyushaserver:~$ sh loop&
[3] 915
user@katyushaserver:~$ ps -f
UID          PID    PPID  C STIME TTY          TIME CMD
user          893      630  0 14:19 tty1        00:00:00 -bash
user          902      893  0 14:19 tty1        00:00:00 bash
user          913      902  67 14:19 tty1        00:00:06 sh loop
user          914      902  45 14:19 tty1        00:00:02 sh loop
user          915      902  33 14:19 tty1        00:00:00 sh loop
user          916      902  0 14:19 tty1        00:00:00 ps -f
user@katyushaserver:~$ kill -19 913
user@katyushaserver:~$ ps -f
UID          PID    PPID  C STIME TTY          TIME CMD
user          893      630  0 14:19 tty1        00:00:00 -bash
user          902      893  0 14:19 tty1        00:00:00 bash
user          913      902  21 14:19 tty1        00:00:15 sh loop
user          914      902  43 14:19 tty1        00:00:28 sh loop
user          915      902  42 14:19 tty1        00:00:26 sh loop
user          917      902  0 14:20 tty1        00:00:00 ps -f

[1]+  Stopped                  sh loop
user@katyushaserver:~$ kill -18 913
user@katyushaserver:~$ kill -19 914
user@katyushaserver:~$ ps -f
UID          PID    PPID  C STIME TTY          TIME CMD
user          893      630  0 14:19 tty1        00:00:00 -bash
user          902      893  0 14:19 tty1        00:00:00 bash
user          913      902  22 14:19 tty1        00:00:25 sh loop
user          914      902  39 14:19 tty1        00:00:42 sh loop
user          915      902  41 14:19 tty1        00:00:42 sh loop
user          918      902  0 14:21 tty1        00:00:00 ps -f

[2]+  Stopped                  sh loop
user@katyushaserver:~$
```

Рисунок 15 – Просмотр трех запущенных процессов

После продолжения выполнения всех остановленных процессов мы видим, что каждый из сценариев `loop` занимает практически одинаковое количество ресурсов процессора (рис. 16).

```
user@katyushaserver:~$ ps -f
UID          PID    PPID  C  STIME TTY          TIME CMD
user         893      630  0  14:19 tty1        00:00:00 -bash
user         902      893  0  14:19 tty1        00:00:00 bash
user         913      902  33  14:19 tty1        00:03:44 sh loop
user         914      902  31  14:19 tty1        00:03:31 sh loop
user         915      902  36  14:19 tty1        00:04:01 sh loop
user         925      902  0  14:31 tty1        00:00:00 ps -f
user@katyushaserver:~$ _
```

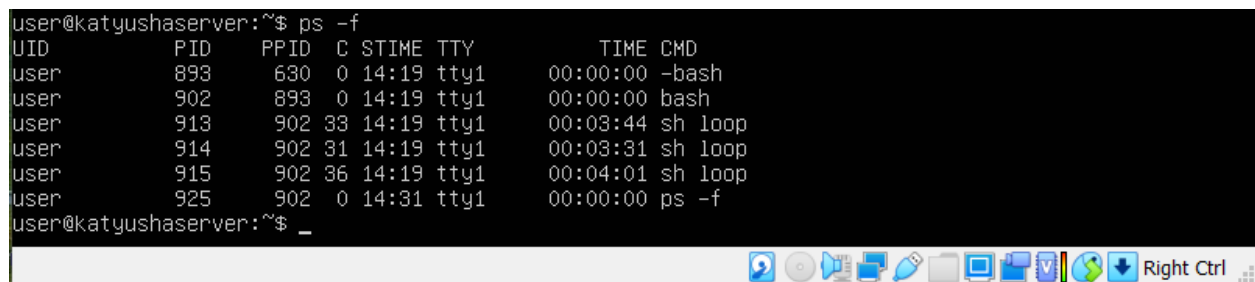
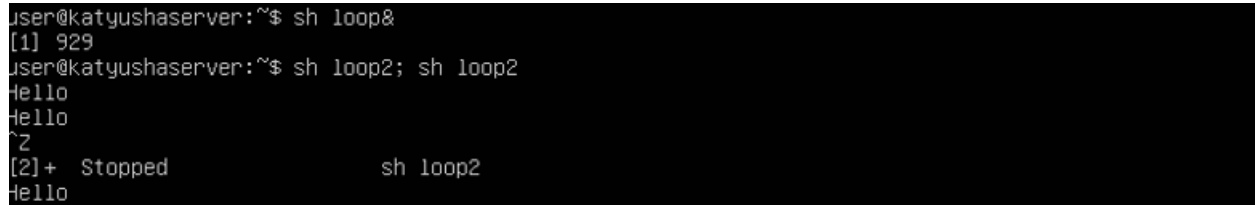


Рисунок 16 – Просмотр трех запущенных процессов

14. Запуск трех задач.

На рисунке 17 показан запуск в консоли на выполнение трех задач. Сначала запускаем в фоновом режиме loop, используя команду `sh loop&`. Затем с помощью команды `sh loop2; sh loop2` запускаем две задачи в интерактивном режиме одновременно.

Сначала запускается `loop2`, с помощью `Ctrl+Z` мы останавливаем процесс, и автоматически запускается второй процесс `loop2`.

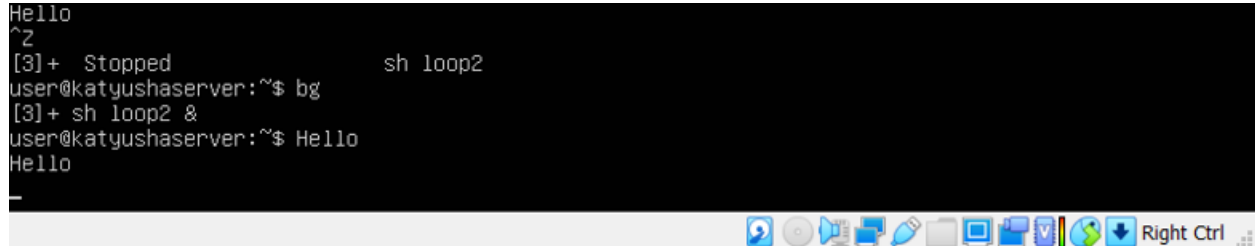


```
user@katyushaserver:~$ sh loop&
[1] 929
user@katyushaserver:~$ sh loop2; sh loop2
Hello
Hello
^Z
[2]+  Stopped                  sh loop2
Hello
```

Рисунок 17 – Запуск трех задач

15. Перевод задачи в фоновый режим.

Останавливаем запущенный процесс `loop2`, используя комбинацию клавиш `Ctrl+Z`. С помощью команды `bg` переводим задачу, выполняющуюся в интерактивном режиме, в фоновый режим (рис. 18).



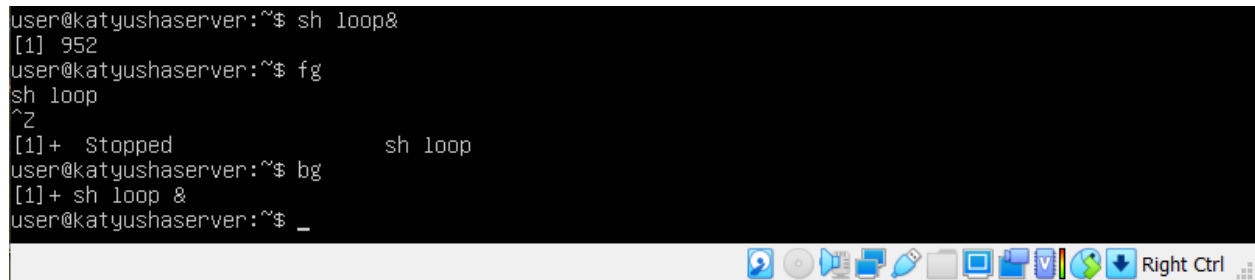
```
Hello
^Z
[3]+  Stopped                  sh loop2
user@katyushaserver:~$ bg
[3]+  sh loop2 &
user@katyushaserver:~$ Hello
Hello
```

Рисунок 18 – Перевод `loop2` в фоновый режим

16. Эксперименты по переводу задач из режимов.

На рисунке 19 показан запуск сценария loop в фоновом режиме, затем перевод его в интерактивный режим с помощью команды fg.

После этого мы останавливаем процесс, используя комбинацию клавиш Ctrl+Z, и переводим его обратно в фоновый с помощью команды bg. Как можно заметить, после смены режима появляются сообщения об успешном выполнении команды.



```
user@katyushaserver:~$ sh loop&
[1] 952
user@katyushaserver:~$ fg
sh loop
^Z
[1]+  Stopped                  sh loop
user@katyushaserver:~$ bg
[1]+ sh loop &
user@katyushaserver:~$ _
```

Рисунок 19 – Перевод задачи loop в разные режимы

17. Создание именованного канала для архивирования и осуществление передачи в канал.

Чтобы создать именованный канал, используем команду «mkfifo /tmp/pipe» (рис. 20).

```
user@katyushaserver:~$ mkfifo /tmp/pipe
```

Рисунок 20 – Создание именованного канала

Для осуществления передачи в канал списка файлов домашнего каталога вместе с подкаталогами используем команду «ls -R > /tmp/pipe &» (рис. 21).

```
user@katyushaserver:~$ ls -R > /tmp/pipe &  
[3] 957
```

Рисунок 21 – Передача списка файлов в канал

После передачи нужно принять данные. Для принятия данных пишем команду «cat /tmp/pipe». Перед нами появляется список файлов домашнего каталога вместе с подкаталогами (рис. 22).

```
user@katyushaserver:~$ cat /tmp/pipe  
.:  
docket  
hello.txt  
loop  
loop2  
./docket:  
d1.txt  
d2.txt  
docket2  
./docket/docket2:  
[3]- Done  
ls --color=auto -R > /tmp/pipe  
user@katyushaserver:~$ _
```

Рисунок 22 – Прием списка файлов домашнего каталога в именованный канал

Чтобы передать в именованный канал один каталог вместе с файлами и подкаталогами, будем использовать архиватор tar. Передадим данные в канал с помощью команды «tar -cvf docket.tar docket > /tmp/pipe &» (рис. 23).

```
user@katyushaserver:~$ tar -cvf docket.tar docket > /tmp/pipe &  
[3] 959
```

Рисунок 23 – Передача каталога docket в канал pipe

Принимаем данные с помощью команды «cat /tmp/pipe». Мы видим каталог docket вместе с файлами и подкаталогами (рис. 24).


```
user@katyushaserver:~$ cat /tmp/pipe
docket/
docket/docket2/
docket/d1.txt
docket/d2.txt
[3]- Done          tar -cvf docket.tar docket > /tmp/pipe
user@katyushaserver:~$
```



Рисунок 24 – Прием данных каталога docket и его подкаталогов

18. Отображение информации о процессах, начиная с указанного PID, с выделением цветом текущего процесса и его предков.

На рисунке 25 показана информация о процессах, начиная с PID = 920. При этом выделяем цветом предков процесса. Делаем это с помощью команды «pstree -pash PID», где параметр -p инструктирует pstree показать PID процессов, -a показывает процессы в виде дерева, -s нужен, чтобы показать родительские процессы данного процесса с PID, -h инструктирует pstree выделить текущий процесс и всех его предков.

```
user@katyushaserver:~$ pstree -pash 920
systemd,1
├─login,629 -p --
│   └─bash,908
│       └─sh,920 loop
user@katyushaserver:~$ _
```

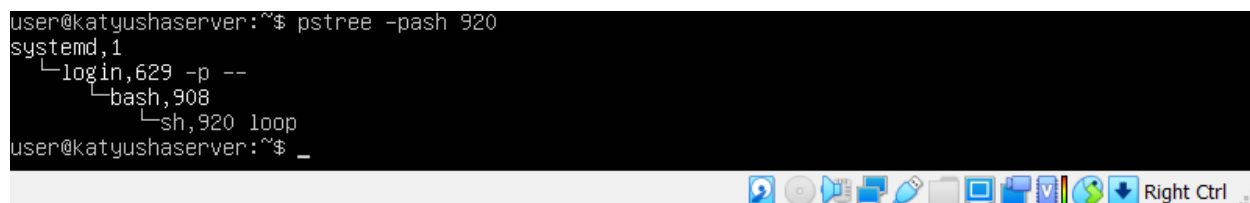


Рисунок 25 – Процессы в виде дерева

19. Завершение выполнения процесса с помощью сигнала SIGINT двумя способами.

Запускаем процесс loop2 в интерактивном режиме с помощью команды sh loop2.

Затем завершаем процесс через сигнал SIGINT (Interrupt, завершает процесс в щадящем режиме), используя комбинацию клавиш ^C (Ctrl+C). Смотрим запущенные процессы и видим, что процесс loop2 действительно завершился (рис. 26).

```
user@katyushaserver:~$ sh loop2
Hello
Hello
^C
user@katyushaserver:~$ ps -f
  UID      PID     PPID  C  STIME TTY          TIME CMD
user        913       639  0  16:07 tty1        0:00:00 -bash
user       1139       913  0  16:58 tty1        0:00:00 ps -f
user@katyushaserver:~$ _
```

Рисунок 26 – Завершение процесса комбинацией клавиш

Теперь опять запускаем процесс loop2 в интерактивном режиме с помощью команды sh loop2. Останавливаем процесс с помощью Ctrl+Z. Для завершения процесса в этот раз задаем имя сигнала kill -2 PID, но, как видим, процесс не завершился. Это связано с тем, сигнал SIGINT посылается активному процессу, а наш на данный момент остановлен. Переводим loop2 в фоновый режим, он автоматически запускается, и, как можно увидеть, срабатывает сигнал kill -2 PID и процесс завешается (рис. 27).

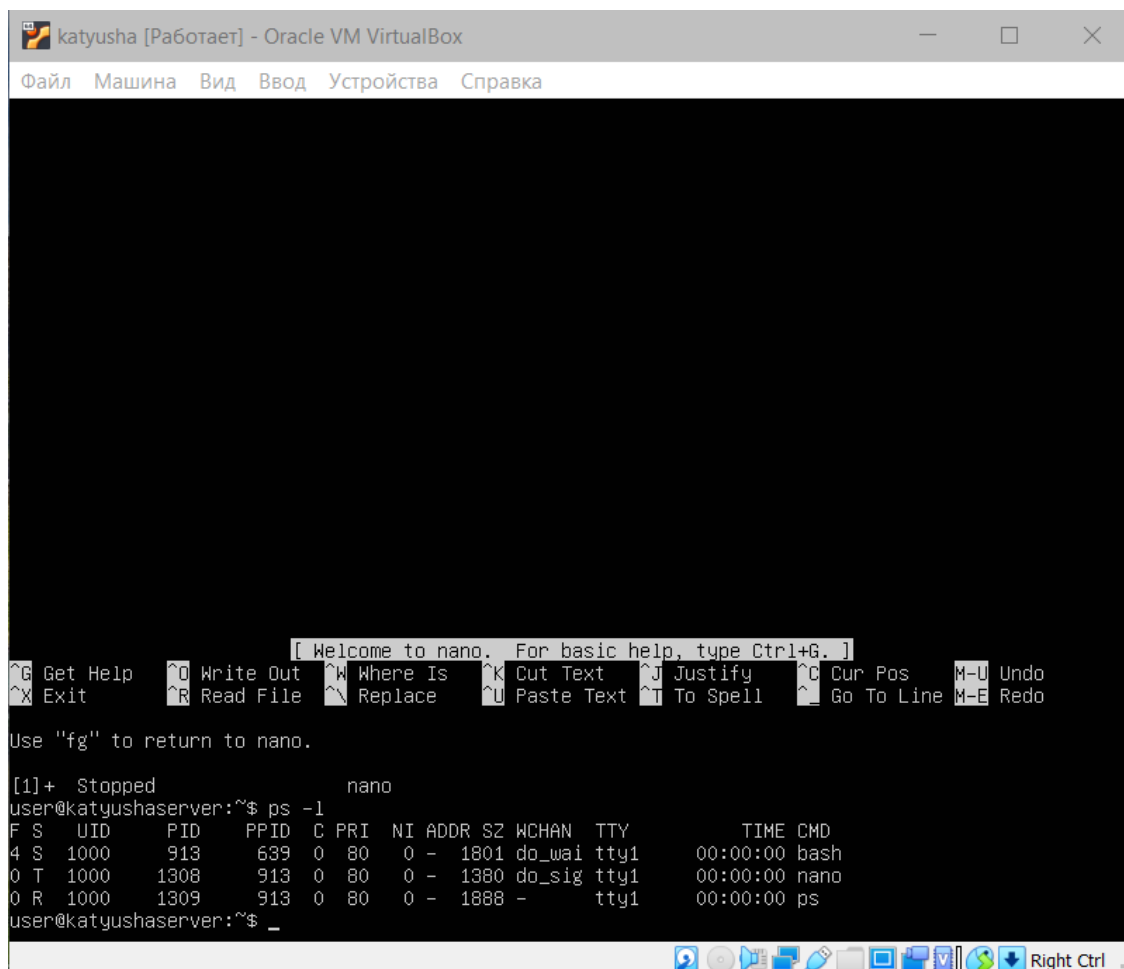
```
user@katyushaserver:~$ sh loop2
Hello
^Z
[1]+  Stopped                  sh loop2
user@katyushaserver:~$ ps
  PID TTY          TIME CMD
  913 tty1        0:00:00 bash
 1225 tty1        0:00:00 sh
 1226 tty1        0:00:00 sleep
 1227 tty1        0:00:00 ps
user@katyushaserver:~$ kill -2 1225
user@katyushaserver:~$ ps
  PID TTY          TIME CMD
  913 tty1        0:00:00 bash
 1225 tty1        0:00:00 sh
 1226 tty1        0:00:00 sleep
 1228 tty1        0:00:00 ps
user@katyushaserver:~$ bg
[1]+  sh loop2 &
[1]+  Interrupt                sh loop2
user@katyushaserver:~$
```

Рисунок 27 – Завершение процесса через сигнал

20. Просмотр приоритета редактора nano и его увеличение на 2.

Запускаем редактор nano, затем останавливаем его с помощью Ctrl+Z. Используя команду ps -l, смотрим информацию о запущенных процессах и видим, что значение приоритета процесса данного редактора равно нулю (рис. 28).

NI - значение nice, которое находится в диапазоне от -20 до 19. Большее значение означает меньший приоритет.



```
katyusha [Работает] - Oracle VM VirtualBox
Файл  Машина  Вид  Ввод  Устройства  Справка

[ Welcome to nano. For basic help, type Ctrl+G. ]
^G Get Help  ^O Write Out  ^W Where Is  ^K Cut Text   ^J Justify    ^C Cur Pos   M-U Undo
^X Exit      ^R Read File  ^_ Replace   ^U Paste Text ^T To Spell   ^_ Go To Line M-E Redo

Use "fg" to return to nano.

[1]+  Stopped                  nano
user@katyushaserver:~$ ps -l
F S  UID      PID     PPID  C  PRI  NI ADDR  SZ  WCHAN  TTY          TIME CMD
4 S   1000      913       639  0   80   0 -  1801 do_wai tty1      00:00:00 bash
0 T   1000     1308      913  0   80   0 -  1380 do_sig tty1      00:00:00 nano
0 R   1000     1309      913  0   80   0 -  1888 -      tty1      00:00:00 ps
user@katyushaserver:~$ _
```

Рисунок 28 – Просмотр приоритета редактора nano

Теперь нам нужно запустить процесс данного редактора с увеличенным на 2 значением приоритета. Чтобы установить значение nice ниже нуля, требуются права суперпользователя. В противном случае будет установлено значение 0. На рисунке 29 показано, как мы сначала переходим в режим суперпользователя командой sudo -s. Теперь повышаем приоритет с помощью команды «nice -n -2 nano».

```
user@katyushaserver:~$ sudo -s
root@katyushaserver:/home/user# nice -n -2 nano_
```

Рисунок 29 – Переход в режим суперпользователя и повышение приоритета

Останавливаем процесс и используя команду `ps -l` видим, что приоритет действительно поднялся, NI стал равен -2 (рис. 30).

```
katyusha [Работает] - Oracle VM VirtualBox
Файл  Машина  Вид  Ввод  Устройства  Справка

[ Welcome to nano. For basic help, type Ctrl+G. ]
^G Get Help  ^O Write Out  ^W Where Is  ^K Cut Text  ^J Justify  ^C Cur Pos  M-U Undo
^X Exit      ^R Read File  ^\ Replace  ^U Paste Text ^T To Spell  ^_ Go To Line M-E Redo

Use "fg" to return to nano.

[1]+  Stopped                  nice -n -2 nano
root@katyushaserver:/home/user# ps -l
F S  UID      PID     PPID  C PRI  NI ADDR S2 WCHAN  TTY          TIME CMD
4 S   0        624      1   0  80   0 -  1493 do_wai tty1      00:00:00 login
4 S   0        908     899   0  80   0 -  2051 poll_s tty1      00:00:00 sudo
4 S   0        914     908   0  80   0 -  1498 do_wai tty1      00:00:00 bash
4 T   0        921     914   0  78  -2 -  1400 do_sig tty1      00:00:00 nano
0 R   0        922     914   0  80   0 -  1888 -      tty1      00:00:00 ps
root@katyushaserver:/home/user# _
```

Рисунок 30 – Просмотр запущенных процессов

Вывод

В результате выполнения лабораторной работы я получила знания по работе с процессами в ОС Linux Ubuntu. Научилась смотреть запущенные процессы, создавать сценарии, изменять приоритет процесса. Поняла, как создавать именованный канал и осуществлять передачу в канал. Также я приобрела навыки по работе командами и сигналами для управления процессами: по их запуску, остановке, завершению, переводу на передний план.