

# Липецкий государственный технический университет

Кафедра прикладной математики

## Отчет по лабораторной работе №2 «Процессы в операционной системе Linux.»

Студент

\_\_\_\_\_

подпись, дата

Комолых Т.О.  
фамилия, инициалы

Группа

ПМ-18

Руководитель

доц., к.п.н. кафедры АСУ  
ученая степень, ученое звание

\_\_\_\_\_

подпись, дата

Кургасов В. В.  
фамилия, инициалы

Липецк 2020 г.

# Содержание

<b>Цель работы</b>	<b>3</b>
<b>Задание кафедры</b>	<b>3</b>
<b>Выполнение работы</b>	<b>5</b>
Часть I: . . . . .	5
Загрузка пользователем. . . . .	5
Поиск файла с образом ядра. Определение по имени файла номера версии Linux. . . . .	5
Просмотр процессов <code>ps -f</code> . . . . .	5
Создание с помощью редактора <code>vi</code> двух сценариев <code>loop</code> и <code>loop2</code> . . . . .	6
Запустить <code>loop2</code> на переднем плане. . . . .	6
Остановка процесса через сигнал <code>STOP</code> . . . . .	6
Последовательный просмотр процессов через <code>ps -f</code> . . . . .	7
Использование сигнала <code>kill -9 PID</code> , чтобы Убить процесс <code>loop2</code> . . . . .	7
Запуск в фоне процесса <code>loop: sh loop</code> . . . . .	8
Завершение процесса <code>loop</code> командой <code>kill -15 PID</code> . . . . .	8
Третий запуск в фоне процесса <code>loop2</code> . Командой <code>kill -9 PID</code> без остановки убить данный процесс. . . . .	9
Запуск экземпляра оболочки: <code>bash</code> . . . . .	9
Запуск нескольких процессов в фоне. . . . .	9
Часть II: . . . . .	10
Запуск в консоли на выполнение трех задач(две в интерактивном режиме, одна - в фоновом). . . . .	10
Перевод одной из задач, выполняющихся в интерактивном режиме, в фоновый режим. . . . .	11
Перевод задач из фонового режима в интерактивный и наоборот. . . . .	11
Создание именованного канала для архивирования и осуществление передачи в канал. . . . .	12
Часть III (вариант 3): . . . . .	12
Генерирование информации о <code>m</code> ( $m > 2$ ) процессах системы, имеющих значение идентификатора больше заданного <code>n</code> . . . . .	12
Завершение выполнения двух процессов (первый процесс завершается с помощью сигнала <code>SIGKILL</code> , второй — с помощью сигнала <code>SIGINT</code> ). . . . .	13
Вывод идентификаторов процессов через символ « : » , для которых родителем является командный интерпретатор. . . . .	14
<b>Вывод</b>	<b>15</b>
<b>Список литературы</b>	<b>16</b>

## Цель работы

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

## Задание кафедры

### Часть I:

- 1) Загрузиться 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`.

### Часть II:

- 1) Запустить в консоли на выполнение три задачи, две в интерактивном режиме, одну - в фоновом.
- 2) Перевести одну из задач, выполняющихся в интерактивном режиме, в фоновый режим.
- 3) Провести эксперименты по переводу задач из фонового режима в интерактивный и наоборот.
- 4) 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.
- 5) 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.

### Часть III (Вариант 3):

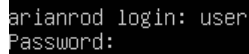
- 1) Сгенерировать следующую информацию о  $m$  ( $m > 2$ ) процессах системы, имеющих значение идентификатора больше заданного  $n$ : флаг — сведения о процессе, статус, PID, PPID, приоритет, использованное время и имя программы.
- 2) Завершить выполнение двух процессов, владельцем которых является текущий пользователь. Первый процесс завершить с помощью сигнала SIGKILL, задав его имя, второй — с помощью сигнала SIGINT, задав его номер.
- 3) Через символ « : » вывести идентификаторы процессов, для которых родителем является командный интерпретатор.
- 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.

# Выполнение работы

## Часть I:

### Загрузка пользователем.

Перед началом выполнения данной части лабораторной работы производится загрузка пользователем user (рисунок 1).

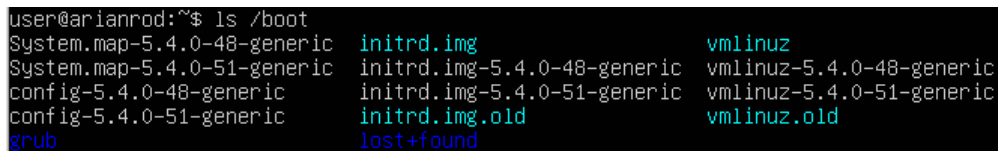


```
arianrod login: user
Password:
```

Рисунок 1.

### Поиск файла с образом ядра. Определение по имени файла номера версии Linux.

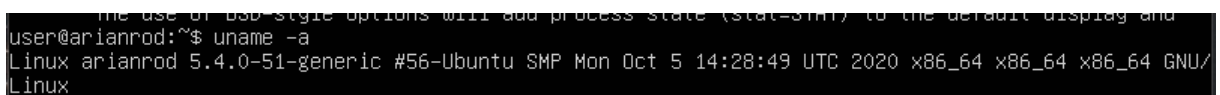
Ядро выступает посредником между программами и аппаратным обеспечением, контролирует процессы, управляет памятью. Файл ядра лежит в папке /boot и называется vmlinuz-<версия> (рисунок 2).



```
user@arianrod:~$ ls /boot
System.map-5.4.0-48-generic  initrd.img          vmlinuz
System.map-5.4.0-51-generic  initrd.img-5.4.0-48-generic  vmlinuz-5.4.0-48-generic
config-5.4.0-48-generic      initrd.img-5.4.0-51-generic  vmlinuz-5.4.0-51-generic
config-5.4.0-51-generic      initrd.img.old       vmlinuz.old
grub                          lost+found
```

Рисунок 2.

Самый распространенный способ посмотреть ядро linux - это команда uname. Она выводит информацию о системе в целом, и в том числе о ядре. Здесь (рисунок 3) сообщается вся доступная информация о ядре Linux, имя компьютера, дата сборки ядра, имя дистрибутива, архитектура и версия ядра - 5.4.0-51.



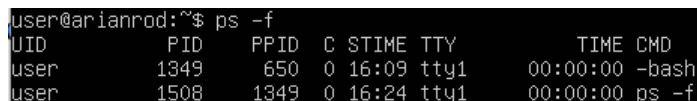
```
user@arianrod:~$ uname -a
Linux arianrod 5.4.0-51-generic #56-Ubuntu SMP Mon Oct 5 14:28:49 UTC 2020 x86_64 x86_64 x86_64 GNU/Linux
```

Рисунок 3.

### Просмотр процессов ps -f.

Если команда ps запускается без каких-либо аргументов, она отображает процессы для текущей оболочки. При добавлении к ps -f осуществляется полноформатный вывод листинга (рисунок 4).

Утилита ps выводит снимок процессов. Позволяет найти процессы по имени, пользователю или терминалу.



```
user@arianrod:~$ ps -f
UID          PID    PPID  C STIME TTY          TIME CMD
user         1349     650  0 16:09 tty1        00:00:00 -bash
user         1508    1349  0 16:24 tty1        00:00:00 ps -f
```

Рисунок 4.

## Создание с помощью редактора vi двух сценариев loop и loop2.

С помощью редактора vi создаются два сценария loop и loop2, где сценарий loop содержит (рисунок 5), а loop2 - (рисунок 7). Для создания сценариев используется команда vi <название>(6).



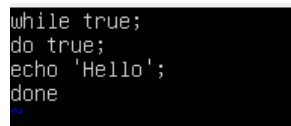
```
while true;
do true;
done
~
```

Рисунок 5.



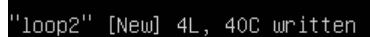
```
user@arianrod:~$ vi loop2_
```

Рисунок 6.



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

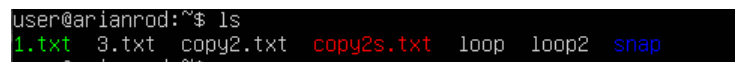
Рисунок 7.



```
"loop2" [New] 4L, 40C written
```

Рисунок 8.

Созданные два сценария поместились в содержимое корневой папки текущего пользователя. Для просмотра содержимого используется команда ls (рисунок 9).



```
user@arianrod:~$ ls
1.txt 3.txt copy2.txt copy2s.txt loop loop2 snap
```

Рисунок 9.

## Запустить loop2 на переднем плане.

Запустить сценарий loop2 на исполнение можно командой sh <имя>(10).

Активный процесс (процесс, исполняемый на переднем плане) - процесс, имеющий возможность вводить данные с терминала. В каждый момент у каждого терминала может быть не более одного активного процесса.



```
user@arianrod:~$ sh loop2
```

Рисунок 10.

## Остановка процесса через сигнал STOP.

Для управления выполнением процессов в Linux предусмотрен механизм передачи сигналов. Сигнал — это способность процессов обмениваться стандартными короткими сообщениями непосредственно с помощью системы. Сообщение - сигнал не содержит никакой информации, кроме номера сигнала.

Сигнал STOP приостанавливает процесс: в таком состоянии процесс не удаляется из таблицы процессов, но и не выполняется. Приостановить процесс можно комбинацией клавиш "Ctrl + Z"(рисунок 11).

```

Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
^Z
[1]+  Stopped                  sh loop2

```

Рисунок 11.

### Последовательный просмотр процессов через `ps -f`.

После остановки процесса `loop2` с помощью последовательного неоднократного вызова команды `ps -f` осуществляется проверка поведения текущего процесса. Исходя из результатов, полученных на (рисунок 12) можно увидеть, что время выполнения процесса `loop2` не меняется.

```

Hello
Hello
^Z
[1]+  Stopped                  sh loop2
user@arianrod:~$ ps -f
UID          PID    PPID  C  STIME TTY          TIME CMD
user          908      656  0  17:34 tty1        00:00:00 -bash
user          927      908  30  17:38 tty1        00:00:03 sh loop2
user          928      908  0  17:38 tty1        00:00:00 ps -f
user@arianrod:~$ ps -f
UID          PID    PPID  C  STIME TTY          TIME CMD
user          908      656  0  17:34 tty1        00:00:00 -bash
user          927      908  21  17:38 tty1        00:00:03 sh loop2
user          929      908  0  17:38 tty1        00:00:00 ps -f
user@arianrod:~$ ps -f
UID          PID    PPID  C  STIME TTY          TIME CMD
user          908      656  0  17:34 tty1        00:00:00 -bash
user          927      908  17  17:38 tty1        00:00:03 sh loop2
user          930      908  0  17:38 tty1        00:00:00 ps -f
user@arianrod:~$ _

```

Рисунок 12.

### Использование сигнала `kill -9 PID`, чтобы Убить процесс `loop2`.

На (рисунок 13) можно заметить, что в таблице процессов содержится процесс `loop2` с PID 1384. Команда `KILL` нужна для того, чтобы убить процесс наверняка, перехватить или игнорировать который невозможно. Пользователь может послать сигнал процессу с идентификатором PID командой `kill -9 <PID>`, где `<сигнал>` — это номер или символическое имя

```

 893 user      20    0 100M 3518  20 S  0.0  0.3  0:00.00 (sd-pam)
 898 user      20    0 7072 5044 3344 S  0.0  0.5  0:00.02 -bash
1384 user      20    0 2608 608  540 T  0.0  0.1  0:01.45 sh loop2
1402 user      20    0 5248 3848 3128 R  0.0  0.4  0:00.03 htop
user@arianrod:~$ kill -9 1384

```

Рисунок 13.

На (рисунок 14) можно заметить, что процесс `loop2` в таблице процессов отсутствует, следовательно, он был удалён.

614	root	20	0	1000	8044	7000	S	0.0	0.0	0:00.04	/lib/systemd/systemd-logind
620	daemon	20	0	3792	2184	2008	S	0.0	0.2	0:00.00	/usr/sbin/atd -f
649	root	20	0	6064	3920	3100	S	0.0	0.4	0:00.01	/bin/login -p --
661	root	20	0	102M	20716	13080	S	0.0	2.1	0:00.00	/usr/bin/python3 /usr/share/unatt
653	root	20	0	102M	20716	13080	S	0.0	2.1	0:00.05	/usr/bin/python3 /usr/share/unatt
657	root	20	0	230M	9112	8176	S	0.0	0.9	0:00.00	/usr/lib/policykit-1/polkitd --no-d
659	root	20	0	230M	9112	8176	S	0.0	0.9	0:00.00	/usr/lib/policykit-1/polkitd --no-d
656	root	20	0	230M	9112	8176	S	0.0	0.9	0:00.00	/usr/lib/policykit-1/polkitd --no-d
892	user	20	0	18652	9768	8076	S	0.0	1.0	0:00.04	/lib/systemd/systemd --user
893	user	20	0	100M	3516	20	S	0.0	0.3	0:00.00	(sd-pam)
898	user	20	0	7072	5044	3344	S	0.0	0.5	0:00.02	-bash
1403	user	20	0	5248	3672	2952	R	0.0	0.4	0:00.02	htop

F1Help F2Setup F3Search F4Filter F5Tree F6SortBy F7Nice F8Nice + F9Kill F10Quit

Рисунок 14.

### Запуск в фоне процесса loop: sh loop.

Для того, чтобы запустить процесс сценария параллельно, достаточно добавить в конец командной строки символ "&": sh имя<sub>(15)</sub>.

Фоновые задания не получают ввода с терминала; как правило, такие задания не нуждаются во взаимодействии с пользователем. Некоторые задания исполняются очень долго. Пример таких заданий — компилирование программ, а также сжатие больших файлов. Такие задания следует запускать в фоновом режиме.

```

user@arianrod:~$ sh loop&
[1] 1404
user@arianrod:~$ ps -f

```

UID	PID	PPID	C	STIME	TTY	TIME	CMD
user	898	649	0	18:03	tty1	00:00:00	-bash
user	1404	898	98	18:45	tty1	00:00:09	sh loop
user	1405	898	0	18:46	tty1	00:00:00	ps -f

```

user@arianrod:~$ ps -f

```

UID	PID	PPID	C	STIME	TTY	TIME	CMD
user	898	649	0	18:03	tty1	00:00:00	-bash
user	1404	898	99	18:45	tty1	00:00:40	sh loop
user	1406	898	0	18:46	tty1	00:00:00	ps -f

```

user@arianrod:~$ ps -f

```

UID	PID	PPID	C	STIME	TTY	TIME	CMD
user	898	649	0	18:03	tty1	00:00:00	-bash
user	1404	898	99	18:45	tty1	00:00:56	sh loop
user	1407	898	0	18:46	tty1	00:00:00	ps -f

Рисунок 15.

### Завершение процесса loop командой kill -15 PID.

На (рисунке 16) приведена таблица процессов, где присутствует исполняемый в фоновом режиме процесс loop.

657	root	20	0	230M	9112	8176	S	0.0	0.9	0:00.00	/usr/lib/policykit-1/polkitd --no-d
659	root	20	0	230M	9112	8176	S	0.0	0.9	0:00.00	/usr/lib/policykit-1/polkitd --no-d
656	root	20	0	230M	9112	8176	S	0.0	0.9	0:00.00	/usr/lib/policykit-1/polkitd --no-d
892	user	20	0	18652	9768	8076	S	0.0	1.0	0:00.04	/lib/systemd/systemd --user
893	user	20	0	100M	3516	20	S	0.0	0.3	0:00.00	(sd-pam)
898	user	20	0	7072	5044	3344	S	0.0	0.5	0:00.03	-bash
1404	user	20	0	2608	608	540	R	99.3	0.1	1:28.57	sh loop
1408	user	20	0	5248	3792	3072	R	0.0	0.4	0:00.02	htop

```

user@arianrod:~$ kill -15 1404

```

Рисунок 16.

Сигнал номер 15 (TERM) служит для прерывания работы задания. При прерывании (interrupt) задания процесс погибает. Для этого используется команда kill -15 <PID> (рисунок 16). На (рисунок 17) видно, что процесс loop, запущенный в фоновом режиме, был удалён.



649	root	20	0	6064	3920	3100	S	0.0	0.4	0:00.01	/bin/login -p --
661	root	20	0	102M	20716	13080	S	0.0	2.1	0:00.00	/usr/bin/python3 /usr/share/unatten
653	root	20	0	102M	20716	13080	S	0.0	2.1	0:00.05	/usr/bin/python3 /usr/share/unatten
657	root	20	0	230M	9112	8176	S	0.0	0.9	0:00.00	/usr/lib/policykit-1/polkitd --no-d
659	root	20	0	230M	9112	8176	S	0.0	0.9	0:00.00	/usr/lib/policykit-1/polkitd --no-d
656	root	20	0	230M	9112	8176	S	0.0	0.9	0:00.00	/usr/lib/policykit-1/polkitd --no-d
892	user	20	0	18652	9768	8076	S	0.0	1.0	0:00.04	/lib/systemd/systemd --user
893	user	20	0	100M	3516	20	S	0.0	0.3	0:00.00	(sd-pam)
898	user	20	0	7072	5044	3344	S	0.0	0.5	0:00.03	-bash
1409	user	20	0	5248	3796	3072	R	0.0	0.4	0:00.02	htop

F1Help F2Setup F3Search F4Filter F5Tree F6SortBy F7Nice - F8Nice + F9Kill F10Quit

Рисунок 17.

**Третий запуск в фоне процесса loop2. Командой kill -9 PID без остановки убить данный процесс.**

Через команду sh loop2 запускается в фоновом режиме процесс loop2. Для удаления текущего процесса используется команда kill -9 <PID> (рисунок 18).

1418	user	20	0	2608	540	458	T	0.0	0.1	1:21.95	sh loop
1421	user	20	0	2608	604	536	T	0.0	0.1	0:01.74	sh loop2
1422	user	20	0	5248	3672	2952	R	0.0	0.4	0:00.01	htop

user@arianrod:~\$ kill -9 1421  
[2]+ Killed sh loop2

Рисунок 18.

**Запуск экземпляра оболочки: bash.**

Чтобы запустить новый экземпляр оболочки bash, необходимо задать команду bash или sh (рисунок 19).

user@arianrod:~\$ bash						
user@arianrod:~\$ ps -f						
UID	PID	PPID	C	STIME	TTY	TIME CMD
user	880	646	0	18:14	tty1	00:00:00 -bash
user	890	880	0	18:14	tty1	00:00:00 bash
user	897	890	0	18:14	tty1	00:00:00 ps -f

Рисунок 19.

**Запуск нескольких процессов в фоне.**

На (рисунок 20) осуществляется запуск в фоновом режиме трёх процессов. Через команду kill -19 <PID> процесс приостанавливается, что видно в полноформатном листинге ps -f.

```

user@arianrod:~$ sh loop&
[1] 958
user@arianrod:~$ sh loop&
[2] 959
user@arianrod:~$ sh loop&
[3] 960
user@arianrod:~$ kill -19 958 959 960
user@arianrod:~$ ps -f

```

UID	PID	PPID	C	STIME	TTY	TIME	CMD
user	880	646	0	18:14	tty1	00:00:00	-bash
user	958	880	32	18:54	tty1	00:00:09	sh loop
user	959	880	29	18:54	tty1	00:00:08	sh loop
user	960	880	29	18:54	tty1	00:00:07	sh loop
user	961	880	0	18:54	tty1	00:00:00	ps -f

```

[1]+  Stopped                  sh loop
[2]   Stopped                  sh loop
[3]-  Stopped                  sh loop
user@arianrod:~$ ps -f

```

UID	PID	PPID	C	STIME	TTY	TIME	CMD
user	880	646	0	18:14	tty1	00:00:00	-bash
user	958	880	17	18:54	tty1	00:00:09	sh loop
user	959	880	16	18:54	tty1	00:00:08	sh loop
user	960	880	15	18:54	tty1	00:00:07	sh loop
user	962	880	0	18:55	tty1	00:00:00	ps -f

Рисунок 20.

Далее после использования команды `kill -18 <PID>` процессы продолжают свою работу, что видно в полноформатном листинге `ps -f`.

```

user@arianrod:~$ kill -18 958 959 960
user@arianrod:~$ ps -f

```

UID	PID	PPID	C	STIME	TTY	TIME	CMD
user	880	646	0	18:14	tty1	00:00:00	-bash
user	958	880	11	18:54	tty1	00:00:10	sh loop
user	959	880	10	18:54	tty1	00:00:09	sh loop
user	960	880	10	18:54	tty1	00:00:08	sh loop
user	963	880	0	18:55	tty1	00:00:00	ps -f

```

user@arianrod:~$ ps -f

```

UID	PID	PPID	C	STIME	TTY	TIME	CMD
user	880	646	0	18:14	tty1	00:00:00	-bash
user	958	880	13	18:54	tty1	00:00:11	sh loop
user	959	880	12	18:54	tty1	00:00:10	sh loop
user	960	880	11	18:54	tty1	00:00:10	sh loop
user	964	880	0	18:55	tty1	00:00:00	ps -f

Рисунок 21.

## Часть II:

**Запуск в консоли на выполнение трех задач**(две в интерактивном режиме, одна - в фоновом).

На (рисунок 22) создаётся два процесса `loop` в интерактивном режиме и один процесс `loop` в фоновом, предварительно поставленные на паузу. Просмотрев процессы в текущей оболочке, можно заметить, что процессы в интерактивном режиме приостановлены, а процесс `loop` в фоновом режиме продолжает свою работу.

```

user@arianrod:~$ sh loop
^Z
[1]+  Stopped                  sh loop
user@arianrod:~$ sh loop
^Z
[2]+  Stopped                  sh loop
user@arianrod:~$ sh loop&
[3] 912
user@arianrod:~$ ps -f
UID          PID    PPID  C STIME TTY          TIME CMD
user          891      648  0 21:16 tty1        00:00:00 -bash
user          910      891 10 21:19 tty1        00:00:07 sh loop
user          911      891  3 21:19 tty1        00:00:02 sh loop
user          912      891 98 21:19 tty1        00:00:52 sh loop
user          913      891  0 21:20 tty1        00:00:00 ps -f
user@arianrod:~$ ps -f
UID          PID    PPID  C STIME TTY          TIME CMD
user          891      648  0 21:16 tty1        00:00:00 -bash
user          910      891  9 21:19 tty1        00:00:07 sh loop
user          911      891  3 21:19 tty1        00:00:02 sh loop
user          912      891 99 21:19 tty1        00:00:59 sh loop
user          915      891  0 21:20 tty1        00:00:00 ps -f

```

Рисунок 22.

### Перевод одной из задач, выполняющихся в интерактивном режиме, в фоновый режим.

Для перевода первого процесса (рисунок 23) в фоновый режим используется команда `bg`. Просмотрев процессы в текущей оболочке, можно заметить, что данный процесс продолжил своё исполнение.

```

user@arianrod:~$ bg 1
[1]- sh loop &
user@arianrod:~$ ps -f
UID          PID    PPID  C STIME TTY          TIME CMD
user          891      648  0 21:16 tty1        00:00:00 -bash
user          910      891 14 21:19 tty1        00:00:19 sh loop
user          911      891  1 21:19 tty1        00:00:02 sh loop
user          912      891 89 21:19 tty1        00:01:42 sh loop
user          918      891  0 21:21 tty1        00:00:00 ps -f
user@arianrod:~$ ps -f
UID          PID    PPID  C STIME TTY          TIME CMD
user          891      648  0 21:16 tty1        00:00:00 -bash
user          910      891 15 21:19 tty1        00:00:20 sh loop
user          911      891  1 21:19 tty1        00:00:02 sh loop
user          912      891 88 21:19 tty1        00:01:43 sh loop
user          919      891  0 21:21 tty1        00:00:00 ps -f

```

Рисунок 23.

### Перевод задач из фонового режима в интерактивный и наоборот.

На (рисунок 24) происходит перевод первого процесса из фонового режима в интерактивный посредством команды `fg`, остановив процесс командой `STOP`. Просмотрев процессы в текущей оболочке, можно заметить, что данный процесс остановлен.

Далее осуществляется обратный перевод данного процесса в фоновый режим командой `bg`. Просмотрев процессы в текущей оболочке, можно заметить, что данный процесс продолжает свою работу.

```

user@arianrod:~$ fg 1
sh loop
^Z
[1]+  Stopped                  sh loop
user@arianrod:~$ ps -f
  UID      PID     PPID  C  STIME TTY          TIME CMD
  user      891       648  0  21:16 tty1        00:00:00 -bash
  user      910       891 23  21:19 tty1        00:00:47 sh loop
  user      911       891  1  21:19 tty1        00:00:02 sh loop
  user      912       891 77  21:19 tty1        00:02:18 sh loop
  user      921       891  0  21:22 tty1        00:00:00 ps -f
user@arianrod:~$ bg 1
[1]+ sh loop &
user@arianrod:~$ ps -f
  UID      PID     PPID  C  STIME TTY          TIME CMD
  user      891       648  0  21:16 tty1        00:00:00 -bash
  user      910       891 23  21:19 tty1        00:00:49 sh loop
  user      911       891  1  21:19 tty1        00:00:02 sh loop
  user      912       891 78  21:19 tty1        00:02:29 sh loop
  user      922       891  0  21:23 tty1        00:00:00 ps -f

```

Рисунок 24.

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

Создание именованного канала для архивирования производится с помощью команды `mkfifo <название_канала>` (рисунок 25). Просмотрев содержимое текущего каталога, можно заметить, что данный канал был успешно создан.

Для осуществления передачи в канал списка файлов домашнего каталога вместе с подкаталогами используется ключ `-R`.

Для осуществления передачи в канал одного каталога вместе с файлами и подкаталогами используется директива `-l`.

```

user@arianrod:~$ ls
1.txt 3.txt 5 copy2.txt copy2s.txt loop loop2 loop3.sh snap
user@arianrod:~$ mkfifo pipe
user@arianrod:~$ ls
1.txt 3.txt 5 copy2.txt copy2s.txt loop loop2 loop3.sh pipe snap
user@arianrod:~$ ls -R > pipe
user@arianrod:~$ ls -l > pipe

```

Рисунок 25.

## Часть III (вариант 3):

### Генерирование информации о $m$ ( $m > 2$ ) процессах системы, имеющих значение идентификатора больше заданного $n$ .

Задание: Сгенерировать следующую информацию о  $m$  ( $m > 2$ ) процессах системы, имеющих значение идентификатора больше заданного  $n$ : флаг — сведения о процессе, статус, PID, PPID, приоритет, использованное время и имя программы.

`read n`

`(ps -A -o stat=,pid=,ppid=,priority=,time=,cmd= | awk '$2 > '$ n' print')`

Первоначально создаётся сценарий `loop4.sh` с расширением `sh`. Команда - `vi loop4.sh`

Текст данного сценария приведён на рисунке 26.

Изначально считывается число  $n$  (указываемый PID). Для просмотра всех запущенных процессов используется команда `ps -A`. Команда `-o` задаёт столбцы, которые будут отображаться (`stat`, `pid`, `ppid`, `priority`, `time`, `cmd`).

Команда `awk` читает документ по одной строке за раз, выполняет указанные действия, то есть сравнивает PID процесса с заданным числом  $n$  и, если PID процесса превышает значение  $n$ , выводит результат на стандартный вывод.

```
read n
(ps -A -o stat=,pid=,ppid=,priority=,time=,cmd= | awk '$2 > '$n' {print}')
```

Рисунок 26.

Прежде чем запустить созданный сценарий нужно изменить его права доступа.

Команда - `chmod a+rx loop4.sh`

На рисунке 27 видно, что теперь сценарий loop4 имеет следующие права `-r-xr-xr-x`.

```
user@arianrod:~$ ls -l loop4.sh
-r-xr-xr-x 1 user user 82 Oct 31 06:27 loop4.sh
```

Рисунок 27.

Результат запуска сценария loop4 приведён на рисунке 28.

```
user@arianrod:~$ ./loop4.sh
800
Ss      888      1  20 00:00:00 /lib/systemd/systemd --user
S       889      888 20 00:00:00 (sd-pam)
S       894      655 20 00:00:00 -bash
I       1043      2  20 00:00:00 [kworker/u2:1-events_unbound]
I<      11636      2  0 00:00:00 [xfsalloc]
I<      11637      2  0 00:00:00 [xfs_mru_cache]
S       11642      2  20 00:00:00 [jfsIO]
S       11643      2  20 00:00:00 [jfsCommit]
S       11644      2  20 00:00:00 [jfsSync]
I       11988      2  20 00:00:00 [kworker/u2:2-events_power_efficient]
I       12010      2  20 00:00:00 [kworker/0:0]
I       12020      2  20 00:00:00 [kworker/u2:0-events_power_efficient]
S+      12083      894 20 00:00:00 -bash
S+      12084     12083 20 00:00:00 -bash
R+      12085     12084 20 00:00:00 ps -A -o stat=,pid=,ppid=,priority=,time=,cmd=
S+      12086     12084 20 00:00:00 awk $2 > 800 {print}
```

Рисунок 28.

**Завершение выполнения двух процессов (первый процесс завершается с помощью сигнала SIGKILL, второй — с помощью сигнала SIGINT).**

Задание: Завершить выполнение двух процессов, владельцем которых является текущий пользователь. Первый процесс завершить с помощью сигнала SIGKILL, задав его имя, второй — с помощью сигнала SIGINT, задав его номер.

`kill -s SIGKILL 1212 1213`

Для завершения выполнения процесса с помощью сигнала SIGKILL используется команда `kill -s SIGKILL <PID>` (рисунок 27).

```
user@arianrod:~$ kill -s SIGKILL 1212 1213
user@arianrod:~$ ps -f
UID      PID     PPID  C  STIME TTY          TIME CMD
user      1200      1125  0  22:08 tty1        00:00:00 -bash
user      1214      1200  99  22:09 tty1        00:02:31 sh loop
user      1236      1200  0  22:11 tty1        00:00:00 ps -f
[1]-  Killed                  sh loop
[2]+  Killed                  sh loop
```

Рисунок 29.

`kill -2 1214`

Для завершения выполнения процесса с помощью сигнала SIGINT используется команда `kill -2 <PID>` (рисунок 28).

```

user@arianrod:~$ kill -2 1214
user@arianrod:~$ ps -f
UID      PID    PPID    C  STIME TTY          TIME CMD
user      1200    1125    0  22:08 tty1        00:00:00 -bash
user      1242    1200    0  22:14 tty1        00:00:00 ps -f
[3]+  Interrupt                  sh loop

```

Рисунок 30.

**Вывод идентификаторов процессов через символ « : » , для которых родителем является командный интерпретатор.**

Задание: Через символ « : » вывести идентификаторы процессов, для которых родителем является командный интерпретатор.

```
(ps -ef | awk '{if ($3 == 1) printf ":%d " , $2}')
```

Чтобы получить полный список процессов, используется команда ps -ef. Опция -e показывает все процессы, а -f – полную информацию: UID - идентификатор пользователя, выполняющего команду, PID - идентификатор процесса команды, PPID - идентификатор родительского процесса, который отпустил команду, C - количество дочерних процессов, STIME - время начала процесса, TTY, TIME, CMD.

Текст данного сценария приведён на рисунке 31.

```

printf 'pid '
(ps -ef | awk '{if ($3 == 1) printf ":%d ", $2}')
printf "\n"

```

Рисунок 31.

Результат запуска сценария loop.sh приведён на рисунке 32.

```

tatyana@arianrod:~$ sudo sh loop.sh
[sudo] password for tatyana:
pid :381 :435 :636 :673 :717 :732 :737 :738 :744 :746 :754 :758 :760 :763 :764 :765 :778 :814 :
847 :861 :915 :989 :1130

```

Рисунок 32.

## Вывод

В ходе лабораторной работы были усвоены понятия процесса в операционной системе и приобретены навыки управления процессами в операционной системе Linux.

## Список литературы

- [1] Львовский, С.М. Набор и верстка в системе  $\text{\LaTeX}$  [Текст] / С.М. Львовский. М.: МЦНМО, 2006. — 448 с.
- [2] SEDICOMM. 30 полезных команд «ps» для мониторинга процессов Linux: <https://blog.sedicom.com/2018/05/28/30-poleznyh-komand-ps-dlya-monitoringa-protsessov-linux/> (дата обращения: 29.10.2020). - Текст: электронный.