

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ



**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБ-
РАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВА-
НИЯ «ЛИПЕЦКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ»**

Институт

компьютерных наук

Кафедра

автоматизированных систем управления

ЛАБОРАТОРНАЯ РАБОТА №3

По дисциплине "Операционные системы Linux"

На тему "Процессы и управление ими в операционной системе Linux"

Студент

ПИ-22-1

подпись, дата

Клименко Н.Д.

Руководитель

канд.техн.наук, доцент

ученая степень, ученое звание

подпись, дата

Кургасов В.В.

Липецк, 2024 г.

Оглавление

Цель работы	3
Ход работы	4
1. Часть I.....	4
2. Часть II	11
3. Часть III	15
4. Часть IV.....	18
Контрольные вопросы.....	24
Вывод	29

Цель работы

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

Ход работы

1. Часть I

1.1. Войти под пользовательской учетной записью (не root). Найти файл с образом ядра. Выяснить по имени файла номер версии Linux.

Файл с образом ядра хранится в директории boot, которая в свою очередь хранится в корневой директории. Следовательно, необходимо воспользоваться командой `ls -l /boot`. Содержимое директории boot представлено на рисунке 1.

```
nikita@deb-server:~$ ls -l /boot
итого 38500
-rw-r--r-- 1 root root 259508 авг 26 22:47 config-6.1.0-25-amd64
drwxr-xr-x 5 root root 4096 окт 3 20:46 grub
-rw-r--r-- 1 root root 30971957 окт 3 20:47 initrd.img-6.1.0-25-amd64
-rw-r--r-- 1 root root 83 авг 26 22:47 System.map-6.1.0-25-amd64
-rw-r--r-- 1 root root 8177600 авг 26 22:47 vmlinuz-6.1.0-25-amd64
nikita@deb-server:~$
```

Рисунок 1 – Директория /boot

Из рисунка 1 видно, что данная директория содержит файл с образом ядра "vmlinuz-6.1.0.25-amd64" (это сжатый бинарный файл ядра), где 6.1.0.25 обозначают версию ядра. Также в директории boot находятся следующие файлы:

- config: файл конфигурации ядра (содержит настройки, связанные с модулями, которые загружаются во время загрузки ядра);
- grub: директория, содержащая файлы, связанные с загрузчиком GRUB;
- initrd.img: файл образа initrd (временная корневая файловая система, которая монтируется в процессе загрузки системы в оперативную память для поддержки двухуровневой модели загрузки. В нём содержится минимальный набор директорий и исполняемых файлов для загрузки модулей);
- System.map: файл, содержащий символьную таблицу адресов функций и процедур, используемых ядром.

1.2. Посмотреть процессы `ps -f`. Прокомментировать, изучив предварительно справку командой `man ps`.

```
nikita@deb-server:~$ ps -f
UID          PID    PPID  C  STIME TTY          TIME CMD
nikita       762    481   0  19:12 tty1        00:00:00 -bash
nikita       765    762   0  19:12 tty1        00:00:00 ps -f
nikita@deb-server:~$
```

Рисунок 2 – Вывод активных процессов

Команда `ps` предоставляет отчет об активных процессах, запущенных в текущем сеансе терминала. Параметр `-f` предоставляет пользователю полноформатный список информации о процессах (без параметра `-f` команда `ps` выводит только PID, TTY, TIME, CMD):

- UID: пользователь, от имени которого запущен процесс;
- PID: идентификатор процесса;
- PPID: идентификатор родительского процесса;
- C: процент времени CPU, используемого процессом;
- STIME: время запуска процесса;
- TTY: терминал, из которого запущен процесс;
- TIME: общее время процессора, затраченное на выполнение процесса;
- CMD: команда запуска процесса.

Из рисунка 2 видно, что команда `ps -f` вывела два активных процесса, первый – оболочка `bash` (запущена при входе в систему), а второй – собственно сама команда `ps -f`.

1.3. Написать с помощью редактора `vi` два сценария `loop` и `loop2`.

С помощью редактора `vi` были созданы два файла `loop` и `loop2` со следующими сценариями:

1) `loop`: `while true; do true; done`. Данный сценарий создает бесконечный цикл, без какого-либо вывода в терминал;

2) `loop2`: `while true; do true; echo 'Hello'; done`. Данный сценарий также создает бесконечный цикл, но при этом каждый раз будет выводить текст в терминал.

Для дальнейшего запуска скриптов необходимо изменить права доступа и разрешить их исполнение. Все действия продемонстрированы на рисунке 3.

```

nikita@deb-server:~$ ls -l
итого 8
-rw-r--r-- 1 nikita nikita 38 окт 19 19:32 loop
-rw-r--r-- 1 nikita nikita 53 окт 19 19:31 loop2
nikita@deb-server:~$ cat loop loop2
#!/bin/bash
while true; do true; done
#!/bin/bash
while true; do true; echo 'Hello'; done
nikita@deb-server:~$ chmod u+x loop loop2
nikita@deb-server:~$ ls -l
итого 8
-rwxr--r-- 1 nikita nikita 38 окт 19 19:32 loop
-rwxr--r-- 1 nikita nikita 53 окт 19 19:31 loop2
nikita@deb-server:~$ _

```

Рисунок 3 – Создание сценариев

1.4. Запустить loop2 на переднем плане.

Для запуска процесса на переднем плане необходимо прописать команду `sh loop2` или `./loop2`, так как ранее в сценарии был прописан путь к интерпретатору (`#!/bin/bash`) и разрешено исполнение файла, то можем воспользоваться второй командой. Результат работы сценария показан на рисунке 3.

```

Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello
Hello

```

Рисунок 4 – Результат работы loop2

Как и следовало ожидать, бесконечный цикл выводит в терминал строку "Hello".

1.5. Остановить, послав сигнал STOP.

Остановки процесса, запущенного на переднем плане возможна двумя способами:

1) Сочетанием клавиш `CTRL + ALT + F2` создать новый терминал `TTY2`. Затем воспользоваться командой `ps -ef`, которая отобразит все процессы, запущенные в системе.

```

nikita      759      489    0 20:17 tty1      00:00:00 -bash
nikita      762      759    99 20:17 tty1      00:00:29 /bin/bash ./loop2
root        763        1    0 20:17 tty2      00:00:00 /bin/login -p --
nikita      768      763    0 20:17 tty2      00:00:00 -bash
nikita      771      768    0 20:18 tty2      00:00:00 ps -ef
nikita@deb-server:~$

```

Рисунок 5 – Поиск PID процесса

Найдя нужный процесс (/bin/bash ./loop2) и узнав его PID (762), можно воспользоваться командой `kill -STOP PID`, которая пошлет процессу сигнал `STOP` и процесс будет остановлен. После чего можно переключиться обратно на терминал `TTY1` (`CTRL + ALT + F1`) и увидеть, что процесс был остановлен.

```

Hello

[1]+  Остановлен    ./loop2
nikita@deb-server:~$

```

Рисунок 6 – Остановка процесса

2) Сочетанием клавиш `CTRL + Z`. `CTRL + Z` посылает процессу сигнал `SIGTSTP`, который приостанавливает его выполнение. Данный сигнал аналогичен сигналу `STOP`.

1.6. Посмотреть последовательно несколько раз `ps -f`. Записать сообщение, объяснить.

```

nikita@deb-server:~$ ps -f
UID          PID    PPID  C  STIME TTY          TIME CMD
nikita       759      489    0  20:17 tty1      00:00:00 -bash
nikita       762      759    99  20:17 tty1      00:01:12 /bin/bash ./loop2
nikita       774      759    0  20:19 tty1      00:00:00 ps -f
nikita@deb-server:~$ ps -f
UID          PID    PPID  C  STIME TTY          TIME CMD
nikita       759      489    0  20:17 tty1      00:00:00 -bash
nikita       762      759    46  20:17 tty1      00:01:12 /bin/bash ./loop2
nikita       775      759    0  20:20 tty1      00:00:00 ps -f
nikita@deb-server:~$

```

Рисунок 7 – Анализ остановленного процесса

Из рисунка 7 можно заметить, что столбец `TIME` для процесса `./loop2` идентичен в двух запусках команды `ps -f`. Столбец `TIME` свидетельствует о затраченном времени процессора на выполнение процесса, и раз оно не изменилось, при двух запусках команды `ps -f`, то можно сделать вывод, что процесс `./loop2` находится в остановленном состоянии.

1.7. Убить процесс `loop2`, пошлав сигнал `kill -9 PID`. Записать сообщение. Прокомментировать.

Так как уже был известен PID процесса ./loop2 (762), то воспользуемся командой kill -9 762 для прекращения существования процесса.

```
nikita@deb-server:~$ kill -9 762
nikita@deb-server:~$ ps -f
UID          PID    PPID  C STIME TTY          TIME CMD
nikita        759      489  0 20:17 tty1        00:00:00 -bash
nikita        829      759  0 20:30 tty1        00:00:00 ps -f
[1]+  Убито                  ./loop2
nikita@deb-server:~$
```

Рисунок 8 – Прекращения существования процесса

Из рисунка 8, что процесс ./loop2, находившейся в остановке был убит и полностью утратил свое существование, что подтверждает команда ps -f.

Команда kill -9 PID: команда kill отправляет сигнал указанному процессу или группе процессов, если сигнал не указан, то команда отправляет сигнал SIGTERM (отправляется процессу с запросом на его завершение, то есть будет выполнена корректная процедура остановки процесса). Если указан сигнал -9 (SIGKILL), то процесс немедленно останавливается и закрывается (что в некоторых случаях может привести к потере данных или отправке сообщения о предполагаемом сбое).

1.8. Запустить в фоне процесс loop: sh loop &. Не останавливая, посмотреть несколько раз ps -f. Записать значение, объяснить.

```
nikita@deb-server:~$ sh loop &
[1] 783
nikita@deb-server:~$ ps -f
UID          PID    PPID  C STIME TTY          TIME CMD
nikita        780      511  0 20:43 tty1        00:00:00 -bash
nikita        783      780  9 20:43 tty1        00:00:09 sh loop
nikita        784      780  0 20:43 tty1        00:00:00 ps -f
nikita@deb-server:~$ ps -f
UID          PID    PPID  C STIME TTY          TIME CMD
nikita        780      511  0 20:43 tty1        00:00:00 -bash
nikita        783      780  9 20:43 tty1        00:01:16 sh loop
nikita        785      780  0 20:44 tty1        00:00:00 ps -f
nikita@deb-server:~$
```

Рисунок 9 – Запуск процесса в фоне

Из рисунка 9 видно, что после запуска процесса loop в фоновом режиме (sh loop &) ему был присвоен PID 783 и номер задачи [1] (job number относится к фоновым процессам и указывает на то, что они управляются оболочкой). Также, помимо присвоения job number вывод о том, что процесс действительно работает в

фоне можно сделать по столбцу TIME. При первом запуске команды `ps -f` процесс `loop` занял 9 секунд процессорного времени, а при втором запуске `ps -f` уже 1 минуту и 16 секунд, это свидетельствует о том, что все это время процесс работает в фоновом режиме.

1.9. Завершить процесс `loop` командой `kill -15 PID`. Записать сообщение, прокомментировать.

Команда `kill -15 PID`: команда `kill` отправляет процессу сигнал `SIGTERM` с запросом на его завершение. Процесс выполнит необходимые действия перед своим завершением, после чего будет "корректно" завершен.

```
nikita@deb-server:~$ ps -f
UID          PID    PPID  C STIME TTY          TIME CMD
nikita        780      511  0 20:43 tty1        00:00:00 -bash
nikita        789      780  0 20:53 tty1        00:00:00 ps -f
[1]+  Завершено      sh loop
nikita@deb-server:~$
```

Рисунок 10 – Завершение процесса

1.10. Третий раз запустить в фоне. Не останавливая, убить командой `kill -9 PID`.

```
nikita@deb-server:~$ sh loop &
[1] 764
nikita@deb-server:~$ ps -f
UID          PID    PPID  C STIME TTY          TIME CMD
nikita        761      486  0 22:43 tty1        00:00:00 -bash
nikita        764      761  99 22:43 tty1        00:00:04 sh loop
nikita        765      761  0 22:44 tty1        00:00:00 ps -f
nikita@deb-server:~$ kill -9 764
nikita@deb-server:~$ ps -f
UID          PID    PPID  C STIME TTY          TIME CMD
nikita        761      486  0 22:43 tty1        00:00:00 -bash
nikita        766      761  0 22:44 tty1        00:00:00 ps -f
[1]+  Убито          sh loop
nikita@deb-server:~$ _
```

Рисунок 11 – Принудительное завершение фонового процесса

На рисунке 11 фоновый процесс `loop` был принудительно завершен командой `kill` с сигналом `SIGKILL`. Выполнение данной команды рекомендуется только в случае зависания процесса, так как она приводит к немедленному завершению процесса без возможностей его корректного завершения.

1.11. Запустить еще один экземпляр оболочки: `bash`.

Чтобы запустить еще один TTY необходимо воспользоваться сочетанием клавиш CTRL + ALT + F*, где вместо * указать цифру терминала. На рисунке 12 представлен переход в TTY2 из TTY1.

```
Debian GNU/Linux 12 deb-server tty2

deb-server login: nikita
Password:
Linux deb-server 6.1.0-25-amd64 #1 SMP PREEMPT_DYNAMIC Debian 6.1.106-3 (2024-08-26) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sat Oct 19 22:43:50 MSK 2024 on tty1
You have new mail.
nikita@deb-server:~$
```

Рисунок 12 – Запуск TTY2

1.12. Запустить несколько процессов в фоне. Останавливать их и снова запускать. Записать результаты просмотра командой ps -f.

Запустим два процесса loop в фоновом режиме. За остановку процессов отвечает команда kill -STOP PID, а за запуск остановленного процесса kill -CONT PID.

```
nikita@deb-server:~$ sh loop &
[1] 764
nikita@deb-server:~$ sh loop &
[2] 765
nikita@deb-server:~$ ps -f
UID          PID    PPID  C STIME TTY          TIME CMD
nikita       761      492  0 23:26 tty1          00:00:00 -bash
nikita       764      761  99 23:26 tty1          00:00:08 sh loop
nikita       765      761  99 23:26 tty1          00:00:06 sh loop
nikita       766      761  33 23:26 tty1          00:00:00 ps -f
nikita@deb-server:~$ kill -STOP 765
nikita@deb-server:~$ ps -f
UID          PID    PPID  C STIME TTY          TIME CMD
nikita       761      492  0 23:26 tty1          00:00:00 -bash
nikita       764      761  99 23:26 tty1          00:00:28 sh loop
nikita       765      761  82 23:26 tty1          00:00:22 sh loop
nikita       767      761  0 23:26 tty1          00:00:00 ps -f

[2]+  Остановлен    sh loop
nikita@deb-server:~$ ps -f
UID          PID    PPID  C STIME TTY          TIME CMD
nikita       761      492  0 23:26 tty1          00:00:00 -bash
nikita       764      761  99 23:26 tty1          00:00:37 sh loop
nikita       765      761  62 23:26 tty1          00:00:22 sh loop
nikita       768      761  0 23:27 tty1          00:00:00 ps -f
nikita@deb-server:~$ kill -CONT 765
nikita@deb-server:~$ kill -STOP 764
nikita@deb-server:~$ ps -f
UID          PID    PPID  C STIME TTY          TIME CMD
nikita       761      492  0 23:26 tty1          00:00:00 -bash
nikita       764      761  95 23:26 tty1          00:01:14 sh loop
nikita       765      761  47 23:26 tty1          00:00:35 sh loop
nikita       769      761  0 23:27 tty1          00:00:00 ps -f

[1]+  Остановлен    sh loop
nikita@deb-server:~$ kill -STOP 765
nikita@deb-server:~$ ps -f
UID          PID    PPID  C STIME TTY          TIME CMD
nikita       761      492  0 23:26 tty1          00:00:00 -bash
nikita       764      761  73 23:26 tty1          00:01:14 sh loop
nikita       765      761  56 23:26 tty1          00:00:55 sh loop
nikita       770      761  0 23:28 tty1          00:00:00 ps -f

[2]+  Остановлен    sh loop
nikita@deb-server:~$ _
```

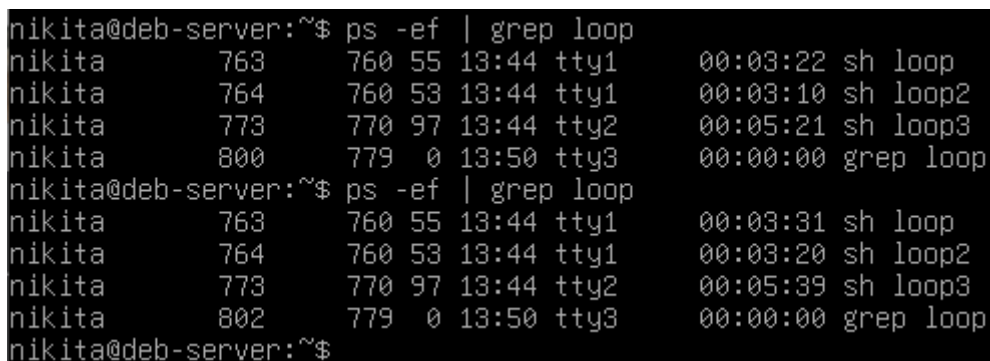
Рисунок 13 – Исследование фоновых процессов

2. Часть II

2.1. Запустить в консоли на выполнение три задачи: две в интерактивном режиме, одну – в фоновом.

Для выполнения данного задания будет создан еще один сценарий loop3 со следующим кодом: while true; do true; done. Данный сценарий просто запустит бесконечный цикл.

Так как при запуске задачи в интерактивном режиме она занимает терминал и не позволяет взаимодействовать с ним до ее остановки, то в TTY1 будут запущены задачи loop в фоновом режиме (sh loop &) и задача loop2 в интерактивном (sh loop2). После этого перейдем в терминал TTY2 (CTRL + ALT + F2) и запустим задачу loop3 в интерактивном режиме (sh loop3). Отслеживать данные процессы будем в терминале TTY3.



```
nikita@deb-server:~$ ps -ef | grep loop
nikita      763      760  55 13:44 tty1      00:03:22 sh loop
nikita      764      760  53 13:44 tty1      00:03:10 sh loop2
nikita      773      770  97 13:44 tty2      00:05:21 sh loop3
nikita      800      779    0 13:50 tty3      00:00:00 grep loop
nikita@deb-server:~$ ps -ef | grep loop
nikita      763      760  55 13:44 tty1      00:03:31 sh loop
nikita      764      760  53 13:44 tty1      00:03:20 sh loop2
nikita      773      770  97 13:44 tty2      00:05:39 sh loop3
nikita      802      779    0 13:50 tty3      00:00:00 grep loop
nikita@deb-server:~$
```

Рисунок 14 – Отслеживание процессов

Из рисунка 14 видно, что была использована команда ps -ef | grep loop (оператор "|" перенаправляет вывод первой команды во входные данные команды grep, которая фильтрует процессы по ключевому слову loop, то есть выводя только те строки, где встречается это слово) два раза и в столбце TIME можно заметить разницу во времени потраченным процессором на выполнение процессов, что свидетельствует о том, что процессы активны.

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

В интерактивном режиме выполняется задача loop2 и loop3. Переведем задачу loop3 в фоновый режим. Для необходимо вернуться в TTY2 и остановить процесс с помощью CTRL + Z (процессу будет послан сигнал SIGTSTP) и с

помощью команды `bg %job number` возобновить выполнение процесса в фоновом режиме. Данные действия продемонстрированы на рисунке 15.

```
nikita@deb-server:~$ sh loop3
^Z
[1]+  Остановлен    sh loop3
nikita@deb-server:~$ bg %1
[1]+  sh loop3 &
nikita@deb-server:~$ ps -ef | grep loop
nikita      763      760 55 13:44 tty1      00:09:18 sh loop
nikita      764      760 54 13:44 tty1      00:09:03 sh loop2
nikita      773      770 92 13:44 tty2      00:15:08 sh loop3
nikita      810       70 14:01 tty2      00:00:00 grep loop
nikita@deb-server:~$ ps -ef | grep loop
nikita      763      760 55 13:44 tty1      00:09:25 sh loop
nikita      764      760 54 13:44 tty1      00:09:11 sh loop2
nikita      773      770 92 13:44 tty2      00:15:22 sh loop3
nikita      812       70 14:01 tty2      00:00:00 grep loop
nikita@deb-server:~$ _
```

Рисунок 15 – Перевод процесса в фоновый режим

2.3. Провести эксперименты по переводы задач из фонового режима в интерактивный и наоборот.

Для возврата процесса в фоновый режим применяется команда `fg %job number`. Применим данную команда для того же процесса `loop3`, предварительно остановив его командой `kill -STOP PID`.

```
nikita@deb-server:~$ kill -STOP 773
nikita@deb-server:~$ fg %1
sh loop3
_
```

Рисунок 16 – Перевод процесса в интерактивный режим

Из рисунка 16 видно, что при переводе процесса `loop3` в интерактивный режим он снова взял управление терминалом на себя, то есть пользователь больше не сможет с ним взаимодействовать, не остановив процесс.

2.4. Создать именованный канал для архивирования и осуществить передачу в канал списка файлов домашнего каталога вместе с подкаталогами и одного каталога вместе с файлами и подкаталогами.

Именованный канал – это один из методов межпроцессорного взаимодействия. Он позволяет различным процессам обмениваться данными, даже если программы, выполняющиеся в этих процессах, изначально не были написаны

для взаимодействия с другими программами. Именованный канал создается с помощью команды `mkfifo <name>`.

```
nikita@deb-server:~$ mkfifo kanal
nikita@deb-server:~$ ls -R ~ > kanal
_
```

Рисунок 17 – Создание именованного канала

На рисунке 17 показано создание именованного канала. Командой `ls -R` был вызван вывод списка файлов домашней директории и ее подкаталогов, но с помощью оператора `>` данный вывод был перенаправлен в именованный канал, который ожидает, пока с этими данными будет что-то сделано, например, можно перейти в ТТУ2 и при помощи команды `cat` и оператора перенаправления вывода `<` вывести содержимое этого канала в терминал. Данный результат показан на рисунке 18.

```
nikita@deb-server:~$ cat < kanal
/home/nikita:
kanal
loop
loop2
loop3
test-dir

/home/nikita/test-dir:
file1
file2
nikita@deb-server:~$
```

Рисунок 18 – Вывод содержимого канала

Вывод будет содержать полный список файлов и подкаталогов в домашней директории. Это демонстрирует, что именованный канал не хранит данные, а предоставляет механизм для передачи их от одного процесса к другому, позволяя процессам взаимодействовать друг с другом.

Теперь попробуем передать в именованный канал содержимое одного каталога с его файлами и подкаталогами, архивируя его. Для этого воспользуемся командой `tar -cvf kanal /home/nikita/test-dir`. Команда архивирует содержимое каталога `test-dir` отправляет архивные данные в именованный канал.

```
nikita@deb-server:~$ tar -cvf kanal /home/nikita/test-dir
_
```

Рисунок 19 – Передача архивных данных

После чего перейдем в ТТУ2 и пропишем: `cat < kanal > arh.tar`, таким образом считаем данные из канала и запишем их в архивный файл.

```
nikita@deb-server:~$ cat < kanal > arh.tar
nikita@deb-server:~$
nikita@deb-server:~$ ls -l
итого 28
-rw-r--r-- 1 nikita nikita 10240 окт 20 15:52 arh.tar
prw-r--r-- 1 nikita nikita      0 окт 20 15:52 kanal
-rwxr--r-- 1 nikita nikita   38 окт 19 19:32 loop
-rwxr--r-- 1 nikita nikita   53 окт 19 19:31 loop2
-rwxr--r-- 1 nikita nikita   38 окт 20 13:43 loop3
drwxr-xr-x 2 nikita nikita  4096 окт 20 15:47 test-dir
nikita@deb-server:~$ tar -tf arh.tar
home/nikita/test-dir/
home/nikita/test-dir/file2
home/nikita/test-dir/file1
nikita@deb-server:~$ _
```

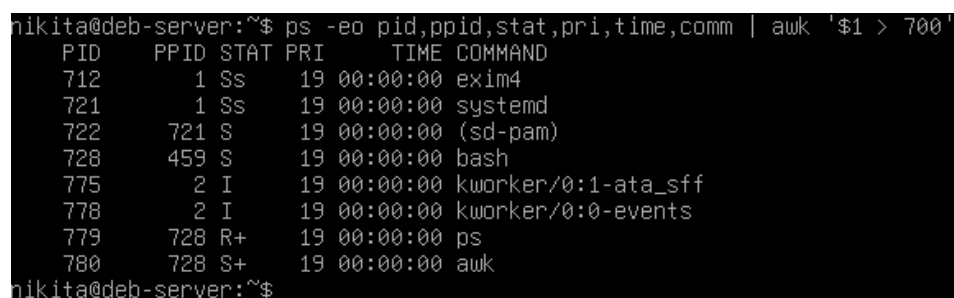
Рисунок 20 – Запись в архив

3. Часть III

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.

Используем команду `ps` с параметрами `-e` (отобразит все процессы) и `-o` (позволяет определить свой формат вывода), полученный результат перенаправим в формат ввода команды `awk`, которая отобразит в терминале только те процессы, у которых PID будет, например, больше 700. Для этого команда `awk` сравнивает значение первого столбца, которые содержит PID, используя '\$1' для обращения к этому столбцу. Итоговая команда: `ps -eo pid,ppid,stat,pri,time,comm | awk '$1 > 700'`, где:

- pid: идентификатор процесса;
- ppid: идентификатор родительского процесса;
- stat: статус процесса;
- pri: приоритет процесса;
- time: общее время процессора, затраченное на выполнение процесса;
- comm: команда, запустившая процесс.



```
nikita@deb-server:~$ ps -eo pid,ppid,stat,pri,time,comm | awk '$1 > 700'
  PID  PPID  STAT PRI   TIME COMMAND
    712     1  Ss   19  00:00:00 exim4
    721     1  Ss   19  00:00:00 systemd
    722    721  S    19  00:00:00 (sd-pam)
    728   459  S    19  00:00:00 bash
    775     2  I    19  00:00:00 kworker/0:1-ata_sff
    778     2  I    19  00:00:00 kworker/0:0-events
    779    728  R+   19  00:00:00 ps
    780    728  S+   19  00:00:00 awk
nikita@deb-server:~$
```

Рисунок 21 – Вывод специализированных процессов

На рисунке 21 продемонстрированная ранее описанная команда. Можно убедиться, что действительно, вывелись только те процессы, у которых идентификатор больше 700.

3.2. 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.

Сперва необходимо узнать, какие имеются процессы, запущенные необходимым пользователем. Для этого можно воспользоваться командой `ps` с параметром `-u <user>`, тогда она отобразит процессы определенного пользователя. Перед применением этой команды и дальнейшей остановкой процессов, запустим два ранее написанных сценария `loop` и `loop3` в фоновом режиме. Результат данных действий продемонстрирован на рисунке 22.

```
nikita@deb-server:~$ sh loop &
[1] 764
nikita@deb-server:~$ sh loop3 &
[2] 765
nikita@deb-server:~$ ps -fu nikita
  UID      PID  PPID  C  STIME TTY          TIME CMD
nikita      754      1   0  19:53 ?        00:00:00 /lib/systemd/systemd --user
nikita      755     754   0  19:53 ?        00:00:00 (sd-pam)
nikita      761     455   0  19:53 tty1    00:00:00 -bash
nikita      764     761  99  19:53 tty1    00:00:15 sh loop
nikita      765     761  99  19:53 tty1    00:00:11 sh loop3
nikita      766     761   0  19:54 tty1    00:00:00 ps -fu nikita
nikita@deb-server:~$
```

Рисунок 22 – Процессы конкретного пользователя

Чтобы завершить процесс по его имени нужно воспользоваться командой `kill`, послав соответствующий сигнал. Если требуется завершить процесс по его идентификатору, то подойдет команда `kill`. Завершение процессов показано на рисунке 22.

```
nikita@deb-server:~$ sh proc &
[1] 766
nikita@deb-server:~$ sh loop &
[2] 767
nikita@deb-server:~$ ps -fu nikita
  UID      PID  PPID  C  STIME TTY          TIME CMD
nikita      754      1   0  19:58 ?        00:00:00 /lib/systemd/systemd --user
nikita      755     754   0  19:58 ?        00:00:00 (sd-pam)
nikita      761     492   0  19:58 tty1    00:00:00 -bash
nikita      766     761  99  19:58 tty1    00:00:19 sh proc
nikita      767     761  99  19:59 tty1    00:00:09 sh loop
nikita      768     761   0  19:59 tty1    00:00:00 ps -fu nikita
nikita@deb-server:~$ kill -SIGKILL -f "sh proc"
[1]-  Убито                  sh proc
nikita@deb-server:~$ kill -SIGINT 767
nikita@deb-server:~$ ps -fu nikita
  UID      PID  PPID  C  STIME TTY          TIME CMD
nikita      754      1   0  19:58 ?        00:00:00 /lib/systemd/systemd --user
nikita      755     754   0  19:58 ?        00:00:00 (sd-pam)
nikita      761     492   0  19:58 tty1    00:00:00 -bash
nikita      770     761   0  20:00 tty1    00:00:00 ps -fu nikita
[2]+  Прерывание            sh loop
nikita@deb-server:~$ _
```

Рисунок 23 – Завершение процессов

Чтобы завершить процесс по его имени необходим параметр `-f`, он будет искать не только по процессу `sh`, но учитывая еще и аргумент процесса, что предотвратит завершение одинаковых процессов.

Сигнал SIGINT отправляется процессу для запроса его остановки, идентичен нажатию CTRL + C терминале, то есть происходит корректное завершение процесса.

3.3. Через символ ":" вывести идентификаторы процессов, для которых родителем является командный интерпретатор.

Воспользуемся следующей командой: `ps -eo | grep bash | awk '{print $1 ":" $2}'`. Эта команда сначала выводит список всех процессов с указанием идентификатора процесса, идентификатора родительского процесса и команды, запустившей процесс. Затем результат перенаправляется на вход команды `grep`, которая фильтрует только те процессы, в строке которых содержится слово `bash`. После чего вывод передается команде `awk`, которая отображает в терминал только PID и PPID через двоеточие. Выполнение данной команды показано на рисунке 24.

```
nikita@deb-server:~$ ps -eo pid,ppid,comm | grep bash | awk '{print $1 ":" $2}'
761:492
nikita@deb-server:~$ _
```

Рисунок 24 – Вывод PID и PPID

4. Часть IV

4.1. Вывести общую информацию о системе

4.1.1. Вывести информацию о текущем интерпретаторе команд

Переменная SHELL хранит путь к текущему интерпретатору команд, чтобы получить значение этой переменной необходимо воспользоваться командой `echo $SHELL`. Результат продемонстрирован на рисунке 25.

```
nikita@deb-server:~$ echo $SHELL
/bin/bash
nikita@deb-server:~$
```

Рисунок 25 – Путь к интерпретатору команд

4.1.2. Вывести информацию о текущем пользователе

Команда `whoami` выводит текущее имя пользователя, связанного с активным сеансом. Данная команда представлена на рисунке 26.

```
nikita@deb-server:~$ whoami
nikita
nikita@deb-server:~$
```

Рисунок 26 – Информация о пользователе

4.1.3. Вывести информацию о текущем каталоге

Команда `pwd` выводит полный от корневого каталога к текущему рабочему каталогу. Данная команда представлена на рисунке 27.

```
nikita@deb-server:~$ pwd
/home/nikita
nikita@deb-server:~$
```

Рисунок 27 – Информация о директории

4.1.4. Вывести информацию об оперативной памяти и области подкачки

Команда `free` отобразит информацию об оперативной памяти и области подкачки, чтобы информация была с размером данных необходим параметр `-h`. Данная команда представлена на рисунке 28.

```
nikita@deb-server:~$ free -h
              total        used         free       shared    buff/cache   available
Mem:           1,9Gi        228Mi        1,7Gi         548Ki         137Mi        1,7Gi
Swap:          974Mi           0B          974Mi
```

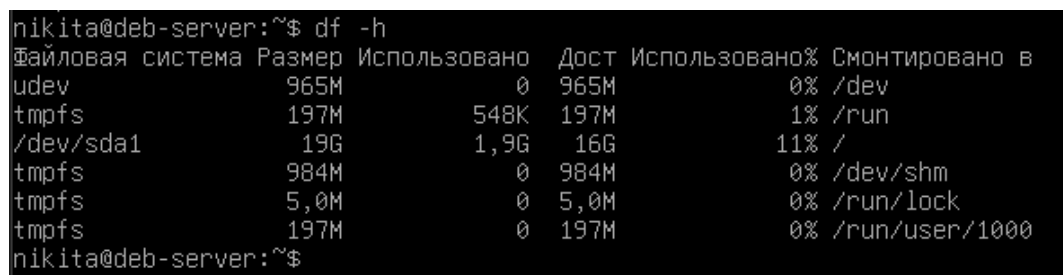
Рисунок 28 – Информация об оперативной памяти и области подкачки

- Mem: информация об оперативной памяти;

- Swap: информация об области подкачки;
- total: общий объем памяти;
- used: используемая в данный момент память;
- free: свободная на данный момент память;
- shared: показывает объем разделяемой памяти;
- buff/cache: память, используемая для кеширования и буферов, которую можно освободить;
- available: объем памяти, доступный для выделения новым процессам или уже существующим.

4.1.5. Вывести информацию о дисковой памяти

Команда `df` показывает информацию о файловых системах и использовании дискового пространства. Параметр `-h` также выводит информацию с размером данных. Данная команда представлена на рисунке 29.



```

nikita@deb-server:~$ df -h
Файловая система  Размер  Использовано  Дост  Использовано%  Смонтировано в
udev              965M      0           965M      0% /dev
tmpfs             197M      548K         197M      1% /run
/dev/sda1         19G      1,9G         16G      11% /
tmpfs             984M      0           984M      0% /dev/shm
tmpfs             5,0M      0            5,0M      0% /run/lock
tmpfs            197M      0           197M      0% /run/user/1000
nikita@deb-server:~$

```

Рисунок 29 – Информация о дисковой памяти

- udev: виртуальный каталог для каталога `/dev`;
- tmpfs: временное файловое хранилище, предназначено для монтирования файловой системы, но размещается в ОЗУ, вместо физического диска.
- `/dev/sda1`: основной раздел жесткого диска, смонтированный как корневая файловая система.

4.2. Выполнить команды получения информации о процессах

4.2.1. Получить идентификатор текущего процесса

Переменная `$$` хранит идентификатор процесса оболочки, в котором выполняется команда. Выведем данный PID с помощью команды `echo $$`. Данная команда представлена на рисунке 30.

```

nikita@deb-server:~$ echo $$
859
nikita@deb-server:~$ ps -f 859
UID          PID    PPID  C  STIME TTY          STAT       TIME CMD
nikita        859      853  0  22:54 tty1      S           0:00 -bash
nikita@deb-server:~$

```

Рисунок 30 – Идентификатор текущего процесса

4.2.2. Получить идентификатор родительского процесса

Идентификатор родительского процесса хранится в переменной PPID. Родительским процессом является процесс /bin/login, который запускает сессию пользователю и оболочку bash. Данная команда представлена на рисунке 31.

```

nikita@deb-server:~$ echo $PPID
853
nikita@deb-server:~$ ps -f 853
UID          PID    PPID  C  STIME TTY          STAT       TIME CMD
root          853        1  0  22:54 tty1      Ss          0:00 /bin/login -p --
nikita@deb-server:~$

```

Рисунок 31 – Идентификатор родительского процесса

4.2.3. Получить информацию о выполняющихся процессах текущего пользователя в текущем интерпретаторе команд

Команда ps с параметром -u выведет процессы пользователя, чье имя будет указано после данного параметра. Пример данной команды представлен на рисунке 32.

```

nikita@deb-server:~$ ps -fu $(whoami)
UID          PID    PPID  C  STIME TTY          STAT       TIME CMD
nikita        753        1  0  23:27 ?        00:00:00 /lib/systemd/systemd --user
nikita        754        753  0  23:27 ?        00:00:00 (sd-pam)
nikita        760        491  0  23:27 tty1      00:00:00 -bash
nikita        764        760  50  23:27 tty1      00:00:00 ps -fu nikita
nikita@deb-server:~$ _

```

Рисунок 32 - Процессы текущего пользователя

Команда whoami возвращает имя пользователя, а с помощью \$() происходит подстановка этого имени в программу, таким образом получаем выполняющиеся процессы текущего пользователя, без явного указания его имени.

4.2.4. Отобразить все процессы

Команда ps с параметром -e выводит все процессы, выполняющиеся в системе, не зависимо от пользователей. Пример некоторых таких процессов представлен на рисунке 33.

```

root      180      2  0 23:26 ?        00:00:00 [jbd2/sda1-8]
root      181      2  0 23:26 ?        00:00:00 [ext4-rsv-conver]
root      221      1  0 23:26 ?        00:00:00 /lib/systemd/systemd-journald
root      250      1  0 23:26 ?        00:00:00 /lib/systemd/systemd-udev
root      297      2  0 23:26 ?        00:00:00 [cryptd]
systemd+  299      1  0 23:26 ?        00:00:00 /lib/systemd/systemd-timesyncd
root      334      2  0 23:26 ?        00:00:00 [irq/18-vmwgfx]
root      337      1  0 23:26 ?        00:00:00 dhclient -4 -v -i -pf /run/dhclient.enp0s3.pid -lf /var/lib/dhcp/dhclient.enp0s3.leases -I -df /var/lib/dhcp
root      481      1  0 23:26 ?        00:00:00 /usr/sbin/cron -f
message+  482      1  0 23:26 ?        00:00:00 /usr/bin/dbus-daemon --system --address=systemd: --nofork --nopidfile --systemd-activation --syslog-only
root      487      1  0 23:26 ?        00:00:00 /lib/systemd/systemd-logind
daemon    490      1  0 23:26 ?        00:00:00 /usr/sbin/atd -f
root      491      1  0 23:26 tty1    00:00:00 /bin/login -p --
root      529      1  0 23:26 ?        00:00:00 sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups
Debian-+  744      1  0 23:26 ?        00:00:00 /usr/sbin/exim4 -bd -g30m
nikita    753      1  0 23:27 ?        00:00:00 /lib/systemd/systemd --user
nikita    754      753  0 23:27 ?        00:00:00 (sd-pam)
nikita    760      491  0 23:27 tty1    00:00:00 -bash
nikita    765      760  0 23:31 tty1    00:00:00 ps -fe
nikita@deb-server:~$ _

```

Рисунок 33 – Все процессы в системе

4.3. Выполнить команды управления процессами

4.3.1. Определить текущее значение nice по умолчанию

Чтобы узнать значение nice по умолчанию достаточно запустить команду nice без параметров.

```

nikita@deb-server:~$ nice
0
nikita@deb-server:~$ _

```

Рисунок 34 – Приоритет по умолчанию

На рисунке 34 видно, что по умолчанию значение nice указано как 0.

4.3.2. Запустить интерпретатор bash с понижением приоритета nice -n 10 bash

Выполним команду nice -n 10 bash, чтобы понизить приоритет интерпретатора. Результат команды представлен на рисунке 35.

```

nikita@deb-server:~$ nice -n 10 bash
nikita@deb-server:~$ ps -o pid,ni,cmd
PID  NI  CMD
760   0  -bash
763  10  bash
765  10  ps -o pid,ni,cmd
nikita@deb-server:~$

```

Рисунок 35 – Понижение приоритета

Интерпретатор bash будет запущен с приоритетом ниже обычного, то есть он будет получать меньше процессорного времени, чем процессы с более высоким приоритетом. Также, все процессы, запущенные в данном интерпретаторе, будут наследовать его приоритет.

4.3.3. Определить PID запущенного интерпретатора

```

nikita@deb-server:~$ ps -u $(whoami) -o pid,ppid,ni,cmd | grep bash
  763      482    0 -bash
  770      763   10 bash
  777      770   10 grep bash
nikita@deb-server:~$ echo $$
770
nikita@deb-server:~$

```

Рисунок 36 – PID запущенного интерпретатора

На рисунке 36 команда `ps -u $(whoami) -o pid,ppid,ni,cmd | grep bash` выводит процессы, где встречается слово `bash`. Таким образом, PID запущенного интерпретатора будет 770. Или вывести значение переменной `$$`, которая хранит идентификатор процесса оболочки, в котором выполняется команда.

4.3.4 Установить приоритет запущенного интерпретатора равным 5

Воспользуемся командой `sudo renice -n 5 <PID>`, чтобы переназначить приоритет запущенного интерпретатора (PID узнали в 4.3.4). Также с помощью команды `ps -o pid,ni,cmd` убедимся в смене приоритета. Результат представлен на рисунке 37.

```

nikita@deb-server:~$ sudo renice -n 5 770
770 (process ID) old priority 10, new priority 5
nikita@deb-server:~$
nikita@deb-server:~$ ps -o pid,ni,cmd
  PID  NI  CMD
   763    0 -bash
   770    5 bash
   788    5 ps -o pid,ni,cmd
nikita@deb-server:~$ _

```

Рисунок 37 – Переназначение приоритета

4.3.5. Получить информацию о процессах `bash`: `ps lax | grep bash`

Используем команду `ps lax | grep bash` и изучим ее вывод. Результат применения данной команды представлен на рисунке 38.

```

nikita@deb-server:~$ ps lax | grep bash
4  1000    763    482  20    0  7972  4752 do_wai S   tty1      0:00 -bash
0  1000    770    763  25    5  8004  4692 do_wai SN  tty1      0:00 bash
0  1000    800    770  25    5  6356  2116 pipe_r SN+  tty1      0:00 grep bash
nikita@deb-server:~$

```

Рисунок 38 – Информация о процессах

Использованные параметры в команде `ps`:

- l: расширенный формат вывода, включающий более количество столбцов;
- a: отображает процессы всех пользователей;
- x: отображает процессы, не привязанные к конкретному терминалу.

Отображаемые столбцы:

- F: флаги процесса (4 – использование прав root, 0 – отсутствие специальных флагов);
- UID: идентификатор пользователя (от какого пользователя запущен процесс);
- PID: идентификатор процесса;
- PPID: идентификатор родительского процесса;
- PRI: динамический приоритет процесса, который изменяется операционной системой в зависимости от его текущей активности и состояния;
- NI: статическое значение, устанавливаемое пользователем или системой, которое влияет на начальный приоритет процесса;
- VSZ: объем виртуальной памяти, используемое процессом;
- RSS: объем физической памяти, который использует процесс;
- WCHAN: показывает, в каком системном вызове процесс ожидает выполнения;
- STAT: состояние процесса.

Контрольные вопросы

1. Перечислите состояния задачи в ОС Linux.

Задачи могут находиться в следующих состояниях:

- R (Running): процесс работает или может начать работу;
- S (Sleeping): процесс не работает, ожидает появления события;
- D (Uninterruptible sleep): процесс ожидает окончания операций ввода/вы-

вода;

- T (Stopped): процессу дана команда на остановку;
- Z (Zombie): процесс был завершен, но информация о нем была обработана процессом-родителем.

2. Как создаются задачи в ОС Linux?

Процессы в ОС Linux создаются путем дублирования текущего процесса с помощью системного вызова `fork()`. Существующий процесс называется родительским, а созданный заново – дочерним. После выполнения `fork()` получаются два практически идентичных процесса, за исключением:

- `fork()` возвращает родителю PID ребенка, ребенку возвращается 0;
- У ребенка меняется PPID на PID родителя.

После выполнения `fork()` все ресурсы дочернего процесса – это копия ресурсов родителя.

3. Назовите классы потоков ОС Linux.

В ОС Linux выделяются три класса потоков для процессов:

- Потоки реального времени, обслуживаемые по алгоритмы FIFO (имеют наивысшие приоритеты и не могут вытесняться другими потоками, за исключением того же потока реального времени с более высоким приоритетом, перешедшего в состояние готовности);

- Потоки реального времени, обслуживаемые в порядке циклической очереди (имеют квант времени и могут вытесняться по таймеру. Находящийся в состоянии готовности поток выполняется в течение кванта времени, после чего поток помещается в конец своей очереди);

- Потоки разделения времени (представлены приоритетами от 100 до 139, то есть в системе Linux реализовано 140 приоритетов).

4. Как используется приоритет планирования при запуске задачи?

Приоритет планирования используется при запуске задачи для определения ее приоритета относительно других задач, находящихся в очереди на выполнение. Задачи с более высоким приоритетом будут выполняться раньше, чем задачи с более низким приоритетом.

5. Объясните, что произойдет, если запустить программу в фоновом режиме без подавления потока вывода.

Если запустить программу в фоновом режиме без подавления потока вывода, она продолжит выводить сообщения в терминал, из которого была запущена. Для этого она использует потоки `stdout` и `stderr`.

6. Объясните разницу между действием сочетаний клавиш `Ctrl^Z` и `Ctrl^C`.

- `Ctrl^Z`: приостанавливает выполнение текущей задачи, переводя ее в состояние `Stopped (T)` и возвращает контроль управления терминалом. Задача может быть продолжена с помощью команды `fg` (на переднем плане) или `bg` (в фоновом режиме);

- `Ctrl^C`: посылает сигнал `SIGINT`, что приводит к завершению задачи.

7. Опишите, что значит каждое поле вывода команды `jobs`.

Команда `jobs` выводит информацию о запущенных задачах. Поля данной команды:

- `Job number`: уникальный идентификатор задания;

- `"+" / "-"`: указывают на текущее задание, на которое могут влиять команды переднего или заднего фона;

- `Status`: текущее состояние задачи;

- `Command`: команда, запустившая задачу.

8. Назовите главное отличие утилиты `top` от `jobs`.

Утилита `top` отображает динамическую информацию о всех процессах системы, включая их потребление ресурсов, а команда `jobs` выводит информацию только о задачах, запущенных в текущей оболочке.

9. В чем отличие результата выполнения команд `top` и `htop`?

`Top` предустановлена во всех дистрибутивах Linux и не требует отдельной установки, `htop` же требует предварительной установки. `Top` не поддерживает прокрутку, `htop` позволяет прокручивать процессы по горизонтали и вертикали. `Top` использует простой текст и выделяет информацию жирным шрифтом, `htop` имеет цветной интерфейс и более наглядное отображение потребления ресурсов.

10. Какую комбинацию клавиш нужно использовать для принудительного завершения задания, запущенного в интерактивном режиме?

Для принудительного завершения задания, запущенного в интерактивном режиме, используется сочетание клавиш `CTRL + C`, которая посылает процессу сигнал `SIGINT`.

11. Какую комбинацию клавиш нужно использовать для приостановки задания, запущенного в интерактивном режиме?

Для приостановки задания в интерактивном режиме используется сочетание клавиш `CTRL + Z`, которое переводит процесс в состояние остановки, с помощью сигнала `SIGTSTP`.

12. Какая команда позволяет послать сигнал конкретному процессу?

Для отправки сигнала конкретному процессу используется команда `kill`, которая принимает PID процесса и тип сигнала в качестве аргументов. Например, `kill -SIGINT PID` завершит указанный процесс.

13. Какая команда позволяет поменять поправку к приоритету уже запущенного процесса?

Команда `renice` используется для изменения значения `nice` уже запущенного процесса. Она позволяет изменить приоритет процесса, чтобы сделать его более или менее приоритетным для планировщика. Например, `renice -n <значение> <PID>`.

14. Какая команда позволяет запустить задание с пониженным приоритетом?

Команда `nice` позволяет запустить задачу с измененным приоритетом. Для запуска задания с пониженным приоритетом используется положительное значение `nice`. Например, `nice -n 10 <команда>`.

15. Какая команда позволяет запустить задание с защитой от прерывания при выходе из системы пользователя?

Команда `nohup` позволяет запустить задание с защитой от прерывания при выходе пользователя из системы. Она предотвращает посылку сигнала `SIGHUP` процессу при завершении сеанса. Например, `nohup <команда> &`.

16. Какой процесс всегда присутствует в системе и является предком всех процессов?

Процесс с PID 1 – `init`, всегда присутствует в системе и является предком всех процессов. Он запускается первым при загрузке системы и порождает другие процессы.

17. Каким образом можно запустить задание в фоновом режиме?

Задание можно запустить в фоновом режиме, добавив символ "&" в конце команды.

18. Каким образом задание, запущенное в фоновом режиме, можно перевести в интерактивный режим?

Чтобы перевести задание, запущенное в фоновом режиме, в интерактивный режим, используется команда `fg <номер задания>`. Например, `fg %1` вернет в интерактивный режим задание с номером 1.

19. Каким образом приостановленное задание можно перевести в интерактивный режим?

Приостановленное задание можно перевести в интерактивный режим командой `fg <номер задания>`. Если не указать номер задания, команда `fg` возобновит последнее приостановленное задание.

20. Что произойдет с заданием, выполняющимся в фоновом режиме, если оно попытается обратиться к терминалу?

Если задание в фоновом режиме попытается обратиться к терминалу, например, через стандартный ввод/вывод, оно будет приостановлено системой с

выводом сообщения типа "Stopped (tty output) ", до тех пор, пока не будет возвращено в интерактивный режим (с помощью команды fg).

21. Сколько терминалов может быть открыто в одной системе? Как перемещаться между терминалами (какие комбинации клавиш необходимо использовать)?

Количество виртуальных терминалов в системе обычно по умолчанию составляет 6, но может быть увеличено в зависимости от конфигурации. Перемещаться между ними можно с помощью комбинаций клавиш CTRL + Alt + Fn, где n – номер виртуального терминала.

22. В чем отличие идентификаторов PID и PPID? При каких условиях возможна ситуация, когда PPID равен нулю или отсутствует?

- PID: уникальный идентификатор процесса;
- PPID: идентификатор родительского процесса.

Если PPID равен 0, это означает, что процесс не имеет родительского процесса, что характерно для процессов, порожденных системой на уровне ядра, например, init.

23. Поясните, от чего зависит максимальное значение PID?

Максимальное значение PID в системе зависит от настроек ядра и может быть изменено с помощью файла /proc/sys/kernel/pid_max.

24. В каком случае, при создании нового процесса, его идентификатор (PID) будет меньше, чем у процесса, запущенного ранее?

В случае, если все доступные PID-идентификаторы уже использованы, система начинает перераспределять PID с начала (при достижении kernel.pid_max). Также это возможно, если предыдущий процесс с меньшим PID завершился, освободив его для нового процесса.

Вывод

В ходе выполнения данной лабораторной работы ознакомился на практике с понятием процесса в операционной системе. Приобрел опыт и навыки управления процессами в операционной системе Linux.