

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Факультет прикладной математики и информатики
Кафедра технологий программирования

Кражевский Алексей Игоревич
ОТЧЕТ ПО ПРОГРАММИРОВАНИЮ МОБИЛЬНЫХ И
ВСТРАИВАЕМЫХ СИСТЕМ
студента 3 курса 13а группы
Лабораторная работа №3

Преподаватель
Давидовская М.И.

Минск, 2022

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

Цели работы:

- Изучить команды управления процессами.
- Изучить вызовы и функции управления каталогами.
- Изучить вызовы и функции управления процессами.

Вариант задания: 13

Задание 3.1. Управление процессами

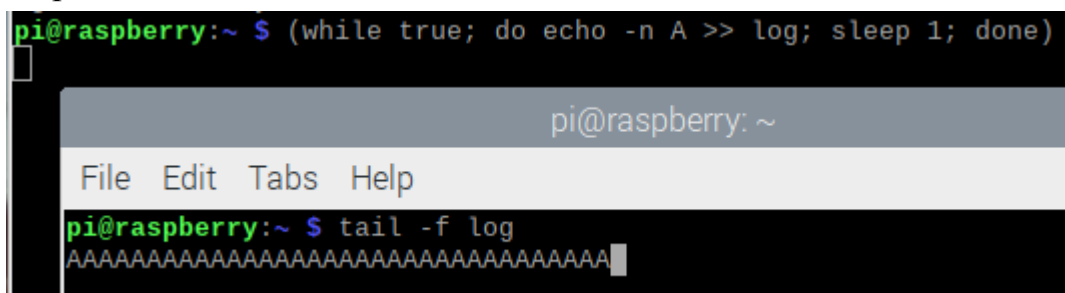
1. Используйте учетную запись, созданную в одной из предыдущих лабораторных работ. Войдите в систему на виртуальных терминалах 1 и 2 (tty1, tty2) под вашей учетной записью.

2. Переключитесь в терминал tty1 и запустите процесс, выполняющий следующие команды: `$ (while true; do echo -n A >> log; sleep 1; done)`

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

`$ tail -f log`

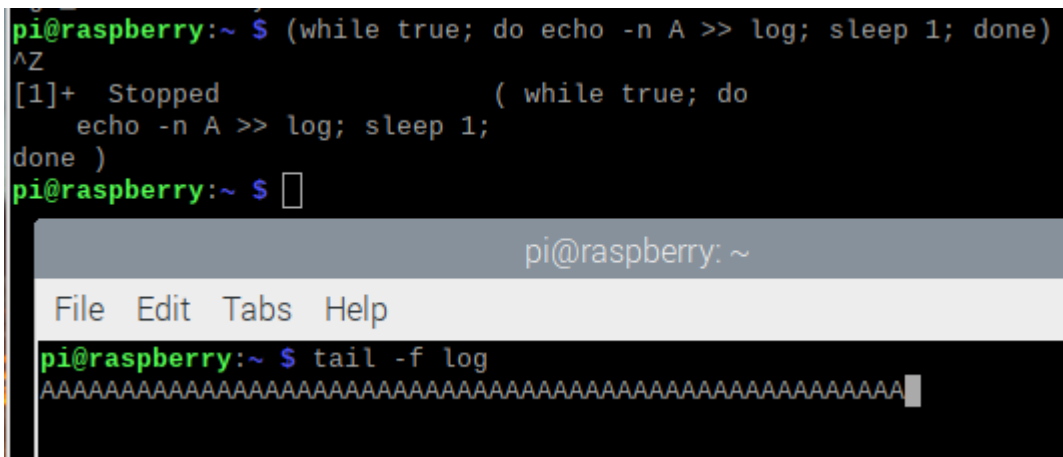
Вы должны увидеть последовательность символов, длина которой возрастает.



```
pi@raspberrypi:~ $ (while true; do echo -n A >> log; sleep 1; done)
pi@raspberrypi: ~
File Edit Tabs Help
pi@raspberrypi:~ $ tail -f log
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

4. Переключитесь в виртуальный терминал tty1 и приостановите работающий процесс, нажав клавиши `<Ctrl+z>`. Командная оболочка сообщит, что процесс остановлен и выдаст вам номер задания [1].

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



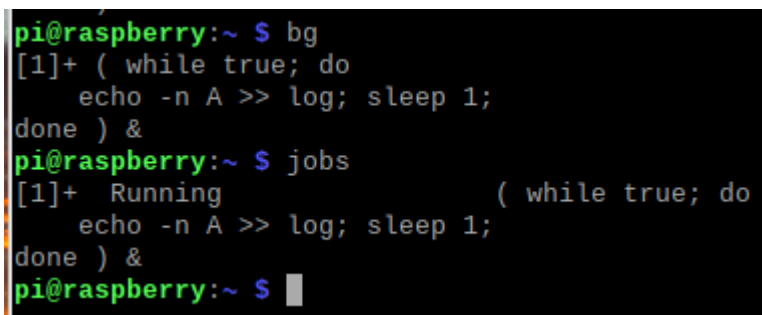
```
pi@raspberrypi:~$ (while true; do echo -n A >> log; sleep 1; done)
^Z
[1]+  Stopped                  ( while true; do
    echo -n A >> log; sleep 1;
done )
pi@raspberrypi:~$
```

The screenshot shows a terminal window with a title bar "pi@raspberrypi: ~". The menu bar includes "File", "Edit", "Tabs", and "Help". The terminal content shows a background process being stopped with `^Z`, resulting in `[1]+ Stopped (while true; do echo -n A >> log; sleep 1; done)`. The prompt returns to `pi@raspberrypi:~$`. A second terminal window is overlaid, showing the command `tail -f log` being executed, which outputs a long string of 'A' characters.

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

```
$ bg
```

```
$ jobs
```



```
pi@raspberrypi:~$ bg
[1]+ ( while true; do
    echo -n A >> log; sleep 1;
done ) &
pi@raspberrypi:~$ jobs
[1]+  Running                  ( while true; do
    echo -n A >> log; sleep 1;
done ) &
pi@raspberrypi:~$
```

The screenshot shows a terminal window where the background process is resumed with `bg`, resulting in `[1]+ (while true; do echo -n A >> log; sleep 1; done) &`. The status is then checked with `jobs`, showing `[1]+ Running (while true; do echo -n A >> log; sleep 1; done) &`. The prompt returns to `pi@raspberrypi:~$`.

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

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

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

```
$ ^B^C
```

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

```

pi@raspberrypi:~$ (while true; do echo -n B >> log; sleep 1; done) &
[2] 2073
pi@raspberrypi:~$ ^B^C
(while true; do echo -n C >> log; sleep 1; done) &
[3] 2189
pi@raspberrypi:~$ jobs
[1]  Running                  ( while true; do
    echo -n A >> log; sleep 1;
done ) &
[2]-  Running                  ( while true; do
    echo -n B >> log; sleep 1;
done ) &
[3]+  Running                  ( while true; do
    echo -n C >> log; sleep 1;
done ) &
pi@raspberrypi:~$

```

```

pi@raspberrypi:~$ tail -f log
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAABABABABABABABABABABABABABABABABABABABABABABABABABABA
BABABCAVCABVCABVCABVCABVCABVCABVCABVCABVCABVCABVCABVCABVCABVCABVC
BCABVCABVCABVCABVC

```

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

```
$ kill -l
```

```
$ kill -19 %1
```

```

pi@raspberrypi:~$ kill -l
 1) SIGHUP          2) SIGINT           3) SIGQUIT          4) SIGILL           5) SIGTRAP
 6) SIGABRT         7) SIGBUS           8) SIGFPE           9) SIGKILL          10) SIGUSR1
11) SIGSEGV        12) SIGUSR2         13) SIGPIPE         14) SIGALRM          15) SIGTERM
16) SIGSTKFLT      17) SIGCHLD         18) SIGCONT          19) SIGSTOP          20) SIGTSTP
21) SIGTTIN        22) SIGTTOU         23) SIGURG          24) SIGXCPU          25) SIGXFSZ
26) SIGVTALRM      27) SIGPROF         28) SIGWINCH         29) SIGIO            30) SIGPWR
31) SIGSYS         34) SIGRTMIN        35) SIGRTMIN+1      36) SIGRTMIN+2      37) SIGRTMIN+3
38) SIGRTMIN+4     39) SIGRTMIN+5      40) SIGRTMIN+6      41) SIGRTMIN+7      42) SIGRTMIN+8
43) SIGRTMIN+9     44) SIGRTMIN+10     45) SIGRTMIN+11     46) SIGRTMIN+12     47) SIGRTMIN+13
48) SIGRTMIN+14    49) SIGRTMIN+15     50) SIGRTMAX-14     51) SIGRTMAX-13     52) SIGRTMAX-12
53) SIGRTMAX-11    54) SIGRTMAX-10     55) SIGRTMAX-9      56) SIGRTMAX-8      57) SIGRTMAX-7
58) SIGRTMAX-6     59) SIGRTMAX-5      60) SIGRTMAX-4      61) SIGRTMAX-3      62) SIGRTMAX-2
63) SIGRTMAX-1     64) SIGRTMAX

```

```

pi@raspberrypi:~$ kill -19 %1
pi@raspberrypi:~$ jobs
[1]+  Stopped                  ( while true; do
    echo -n A >> log; sleep 1;
done )
[2]   Running                  ( while true; do
    echo -n B >> log; sleep 1;
done ) &
[3]-  Running                  ( while true; do
    echo -n C >> log; sleep 1;
done ) &
pi@raspberrypi:~$

```

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

```

pi@raspberrypi:~$ kill -19 %1
pi@raspberrypi:~$ jobs
[1]+  Stopped                  ( while true; do
    echo -n A >> log; sleep 1;
done )
[2]   Running                  ( while true; do
    echo -n B >> log; sleep 1;
done ) &
[3]-  Running                  ( while true; do
    echo -n C >> log; sleep 1;
done ) &
pi@raspberrypi:~$

```

```

pi@raspberrypi:~$ tail -f log
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAABABABABABABABABABABABABABABABABABABABABABABABABAB
BABABABABABABABABABABABABABABABABABABABABABABABABABABABABABABABAB
BCABABABABABABABABABABABABABABABABABABABABABABABABABABABABABABAB
CABABABABABABABABABABABABABABABABABABABABABABABABABABABABABABAB
ABABABABABABABABABABABABABABABABABABABABABABABABABABABABABABABAB
BCBVBVBVBVBVBVBVBVBVBVBVBVBVBVBVBVBVBVBVBVBVBVBVBVBVBVBVBVBVBVB
VBVBVBVBVBVBVBVBVBVBVBVBVBVBVBVBVBVBVBVBVBVBVBVBVBVBVBVBVBVBVB
VBVBVBVBVBVBVBVBVBVBVBVBVBVBVBVBVBVBVBVBVBVBVBVBVBVBVBVBVBVBVB

```

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

```

pi@raspberrypi:~$ kill -18 %1
pi@raspberrypi:~$

```

```
pi@raspberrypi:~$ tail -f log
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAABABABABABABABABABABABABABABABABABABABABABABABABABABABA
BABABCABCABCABCABCABCABCABCABCABCABCABCABCABCABCABCABCABCABCABCABC
BCABCABCABCABCABCABCABCABCABCABCABCABCABCABCABCABCABCABCABCABCABCAB
CABCABCABCABCABCABCABCABCABCABCABCABCABCABCABCABCABCABCABCABCABCABC
ABCABCABCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCB
BCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCB
CBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCBCB
BCBCBCBCBCBCBCBACBACBACBACBACBACBACBACBACBACBACBACBACBACBACBACBACBACB
```

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

```
$ kill %2 %3
```

\$ jobs

```
pi@raspberrypi:~$ kill %2 %3
bash: kill: (2073) - No such process
bash: kill: (2189) - No such process
[2] Terminated ( while true; do
    echo -n B >> log; sleep 1;
done )
[3]- Terminated ( while true; do
    echo -n C >> log; sleep 1;
done )
pi@raspberrypi:~$ kill %1
pi@raspberrypi:~$ jobs
[1]+ Terminated ( while true; do
    echo -n A >> log; sleep 1;
done )
pi@raspberrypi:~$
```

12. Чтобы завершить работу последнего процесса, выполните команды:

\$ fg

\$ <Ctrl+c>

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

[illegible]

```
pi@raspberrypi:~$ ls
Bookshelf  Downloads  Pictures  Templates
Desktop    log         Public    timelog3_1Krazhevskiy13r.txt
Documents  Music      task3_1Krazhevskiy13r  Videos
pi@raspberrypi:~$ rm log
pi@raspberrypi:~$ ls
Bookshelf  Downloads  Public    timelog3_1Krazhevskiy13r.txt
Desktop    Music      task3_1Krazhevskiy13r  Videos
Documents  Pictures   Templates
pi@raspberrypi:~$
```

1. Войдите в систему на виртуальных терминалах 1 и 2 (tty1, tty2) под учетной своей записью.
2. Переключитесь в терминал tty1 и запустите на переднем плане бесконечный процесс.

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

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

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

```
(base) alex@avocadobook:~$ bg
[1]+ ( while true; do
    echo -n A >> log; sleep 1;
done ) &
(base) alex@avocadobook:~$
```

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

```
(base) alex@avocadobook:~$ (while true; do echo -n B >> log; sleep 1; done) &
[2] 71300
(base) alex@avocadobook:~$
```

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

```
(base) alex@avocadobook:~$ jobs
[1]-  Running                  ( while true; do
    echo -n A >> log; sleep 1;
done ) &
[2]+  Running                  ( while true; do
    echo -n B >> log; sleep 1;
done ) &
(base) alex@avocadobook:~$
```

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

```
(base) alex@avocadobook:~$ sudo renice 10 -p 4287
[sudo] password for alex:
4287 (process ID) old priority 0, new priority 10
(base) alex@avocadobook:~$
```

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

```
(base) alex@avocadobook:~$ kill -19 %1

[1]+  Stopped                  ( while true; do
    echo -n A >> log; sleep 1;
done )
(base) alex@avocadobook:~$
```

10. Проверьте, что работает только один процесс.


```
(base) alex@avocadobook:~$ jobs
[1]+  Stopped                  ( while true; do
    echo -n A >> log; sleep 1;
done )
[2]-  Running                  ( while true; do
    echo -n B >> log; sleep 1;
done ) &
(base) alex@avocadobook:~$
```

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

```
(base) alex@avocadobook:~$ kill -19 %2

[2]+  Stopped                  ( while true; do
    echo -n B >> log; sleep 1;
done )
(base) alex@avocadobook:~$ jobs
[1]-  Stopped                  ( while true; do
    echo -n A >> log; sleep 1;
done )
[2]+  Stopped                  ( while true; do
    echo -n B >> log; sleep 1;
done )
(base) alex@avocadobook:~$
```

```
(base) alex@avocadobook:~$ tail -f log
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
BABABABABABABABABABABABABABABABABABABABABABABABABABABABAB
ABABABABABABABABABABABABABABABABABABABABABABABABABABABAB
[1]+  Stopped                  tail -f log
(base) alex@avocadobook:~$
```

Задание 3.3. Управление процессами и планирование заданий

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

a)

```
(base) alex@avocadobook:~$ find ~ -empty > krazhevskiy_empty.txt
(base) alex@avocadobook:~$
```

```
/home/alex/Documents/icpc-2022-23/tasks/venv/lib/python3.10/site-packages/numpy/
/home/alex/Documents/icpc-2022-23/tasks/venv/lib/python3.10/site-packages/numpy/
.py
/home/alex/Documents/icpc-2022-23/tasks/venv/lib/python3.10/site-packages/pkg_re
py
```

b)

```
(base) alex@avocadobook:~$ (for i in ~/.*; do echo $i > krazhevskiy_hid.txt; done) &
```

```
(base) alex@avocadobook:~$ cat krazhevskiy_hid.txt  
/home/alex/.wget-hsts
```

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

```
(base) alex@avocadobook:~$ sleep 200 &  
[1] 74860  
(base) alex@avocadobook:~$ sleep 2500 &  
[2] 74871  
(base) alex@avocadobook:~$ jobs  
[1]-  Running                  sleep 200 &  
[2]+  Running                  sleep 2500 &  
(base) alex@avocadobook:~$
```

```
(base) alex@avocadobook:~$ kill -19 %1  
[1]+  Stopped                  sleep 200  
(base) alex@avocadobook:~$ jobs  
[1]+  Stopped                  sleep 200  
[2]-  Running                  sleep 2500 &  
(base) alex@avocadobook:~$
```

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

```
(base) alex@avocadobook:~$ exec ls -R /etc
```

```
/etc/xdg/menus:  
gnome-applications.menu  xfce-applications.menu  
  
/etc/xdg/systemd:  
user  
  
/etc/xdg/Thunar:  
uca.xml  
  
/etc/xdg/tumbler:  
tumbler.rc  
  
/etc/xdg/xfce4:  
xfconf
```

Команда `exec ls -R /etc` запускает команду `ls` рекурсивно, т.е. просмотр рекурсивно всего содержимого каталога `/etc`.

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

Создаю во втором терминале через `vim` файл, в первом проверяю его `pid`.

```
(base) alex@avocadobook:~$ pidof bash
74956 74634
(base) alex@avocadobook:~$ pstree -p 74956
bash(74956)---vim(74972)
(base) alex@avocadobook:~$ pstree -p 74634
bash(74634)---pstree(74978)
(base) alex@avocadobook:~$
```

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

```
(base) alex@avocadobook:~$ kill -TERM 74956
(base) alex@avocadobook:~$ pidof bash
74956 74634
(base) alex@avocadobook:~$
```

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

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

Его выполнение тоже ничего не поменяло:

```
(base) alex@avocadobook:~$ kill -INT 74956
(base) alex@avocadobook:~$ pidof bash
74956 74634
(base) alex@avocadobook:~$
```

Сигнал QUIT - похож на **TERM**, но позволяет сохранить дамп памяти.

Программа может выполнить корректное завершение или проигнорировать сигнал.

```
(base) alex@avocadobook:~$ kill -QUIT 74956
(base) alex@avocadobook:~$ pidof bash
74956 74634
(base) alex@avocadobook:~$
```

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

```
(base) alex@avocadobook:~$ kill -HUP 74956
(base) alex@avocadobook:~$ pidof bash
74634
(base) alex@avocadobook:~$
```

5. От имени обычного пользователя пошлите сигнал KILL любому процессу, запущенному от имени другого пользователя. Что произойдет? Ничего не произойдет, тк для убийства процесса другого пользователя нужны root-права:

```
(base) alex@avocadobook:~$ kill 74634
(base) alex@avocadobook:~$ pidof bash
74634
(base) alex@avocadobook:~$
```

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

```
(base) alex@avocadobook:~$ jobs
(base) alex@avocadobook:~$ sleep 1000 &
[1] 75136
(base) alex@avocadobook:~$
```

При сигнале `TERM` процесс останавливается, а затем умирает:

```
(base) alex@avocadobook:~$ sleep 1000 &
[1] 75136
(base) alex@avocadobook:~$ jobs
[1]+  Running                  sleep 1000 &
(base) alex@avocadobook:~$ kill -TERM 75136
(base) alex@avocadobook:~$ jobs
[1]+  Terminated              sleep 1000
(base) alex@avocadobook:~$
```

При `INT` - процесс прерывается, а через некоторое время также погибает:

```
(base) alex@avocadobook:~$ sleep 1000 &
[1] 75192
(base) alex@avocadobook:~$ jobs
[1]+  Running                  sleep 1000 &
(base) alex@avocadobook:~$ kill -INT 75192
(base) alex@avocadobook:~$ jobs
[1]+  Interrupt                sleep 1000
(base) alex@avocadobook:~$ jobs
(base) alex@avocadobook:~$
```

При `QUIT` - процесс закрывается, в результате также через некоторое время умирает, также появляется дамп:

```
(base) alex@avocadobook:~$ sleep 1000 &
[1] 75209
(base) alex@avocadobook:~$ jobs
[1]+  Running                  sleep 1000 &
(base) alex@avocadobook:~$ kill -QUIT 75209
(base) alex@avocadobook:~$ jobs
[1]+  Quit                    (core dumped) sleep 1000
(base) alex@avocadobook:~$ jobs
(base) alex@avocadobook:~$
```

При `HUP` - получает статус подвешенный и тоже в скором времени умирает:

```
(base) alex@avocadobook:~$ sleep 1000 &
[1] 75234
(base) alex@avocadobook:~$ jobs
[1]+  Running                  sleep 1000 &
(base) alex@avocadobook:~$ kill -HUP 75234
(base) alex@avocadobook:~$ jobs
[1]+  Hangup                  sleep 1000
(base) alex@avocadobook:~$ jobs
(base) alex@avocadobook:~$
```

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

```
(base) alex@avocadobook:~$ vim task7
(base) alex@avocadobook:~$ cat task7
trap 'echo $(whoami) $(pwd) > pwd.txt; exit' 15
while true; do
a=0
done
(base) alex@avocadobook:~$ sh task7 &
[1] 75287
```

```
(base) alex@avocadobook:~$ kill -TERM 75287
(base) alex@avocadobook:~$ cat pwd.txt
alex /home/alex
[1]+  Done                    sh task7
(base) alex@avocadobook:~$
```

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

Да, работает.

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

```
(base) alex@avocadobook:~$ nice -n1 bash
(base) alex@avocadobook:~$
```

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

```
(base) alex@avocadobook:~$ sudo time nice -n19 updatedb
[sudo] password for alex:
nice: 'updatedb': No such file or directory
Command exited with non-zero status 127
0.00user 0.00system 0:00.00elapsed 0%CPU (0avgtext+0avgdata 2200maxresident)k
0inputs+0outputs (0major+98minor)pagefaults 0swaps
(base) alex@avocadobook:~$ sudo time nice -n5 updatedb
nice: 'updatedb': No such file or directory
Command exited with non-zero status 127
0.00user 0.00system 0:00.00elapsed 90%CPU (0avgtext+0avgdata 1984maxresident)k
0inputs+0outputs (0major+97minor)pagefaults 0swaps
(base) alex@avocadobook:~$
```

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

```
(base) alex@avocadobook:~$ ps -ef > $(date +%T) | at now +5 minutes
warning: commands will be executed using /bin/sh
job 1 at Mon Oct 17 13:35:00 2022
(base) alex@avocadobook:~$
```

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

```
(base) alex@avocadobook:~$ kill $(pidof chromium) | at 16:00
warning: commands will be executed using /bin/sh
kill: usage: kill [-s sigspec | -n signum | -sigspec] pid | jobspec ... or kill -l [sigspec]
job 2 at Mon Oct 17 16:00:00 2022
(base) alex@avocadobook:~$
```

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

```
SHELL=/bin/bash
MAILTO=alex
0 0 * * * updatedb
0 * * * * updatedb
*/5 * * * * updatedb
```

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

```
(base) alex@avocadobook:~$ echo '*/*10 * * * * pkill chromium' > cron 2
(base) alex@avocadobook:~$ crontab cron2
cron2: No such file or directory
(base) alex@avocadobook:~$ crontab cron 2
(base) alex@avocadobook:~$
```

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


```
(base) alex@avocadobook:~$ at now +5 minutes
warning: commands will be executed using /bin/sh
at Mon Oct 17 13:41:00 2022
at> echo '9 * * * * echo '7 * * * * pkill cron ' > crontab1
at> crontab crontab1
```

Задание 3.4. Управление процессами и планирование заданий

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

```
pi@raspberrypi:~$ cat task1.sh
#!/bin/bash
# trap test
trap 'echo you hit Ctrl+C; exit' SIGINT
count=0
while:
do
    sleep 1
    count=$(expr $count + 1)
    echo $count
done
pi@raspberrypi:~$
```

```
pi@raspberrypi:~$ ./task1.sh
1
2
3
4
^Cyou hit Ctrl+C
pi@raspberrypi:~$
```

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

```
pi@raspberrypi:~$ vi task2.sh
pi@raspberrypi:~$ chmod u+x task2.sh
pi@raspberrypi:~$ cat task2.sh
#!/bin/bash

(while true; do ls >> res.txt; sleep 5; done) &
(while true; do wc res.txt >> res.txt; sleep 5; done) &
pi@raspberrypi:~$
```



```
pi@raspberrypi:~$ cat res.txt
Bookshelf
Desktop
Documents
Downloads
Music
Pictures
Public
res.txt
task1.sh
task2.sh
```

```
45 54 510 res.txt
Bookshelf
Desktop
Documents
Downloads
Music
Pictures
Public
res.txt
task1.sh
task2.sh
task3_4Krazhevskiy13r
```

Задача 3. Написать пользовательскую функцию обработки сигнала. Установка обработки сигнала происходит однократно (обрабатывается только одно событие, связанное с появлением данного сигнала SIG_ALARM). Возврат из функции-обработчика происходит в точку прерывания процесса.

```
pi@raspberrypi:~$ vi task3.sh
pi@raspberrypi:~$ chmod u+x task3.sh
pi@raspberrypi:~$ cat t
task1.sh          task3_4Krazhevskiy13r      timelog3_4Krazhevskiy13r
task2.sh          task3.sh
pi@raspberrypi:~$ cat task3.sh
#!/bin/bash
trap 'echo hello' SIGALRM
(while true; do echo A; sleep 5; done)
pi@raspberrypi:~$ ./ task3.sh
bash: ./: Is a directory
pi@raspberrypi:~$ ./task3.sh
A
^C
pi@raspberrypi:~$
```

Задание 3.5. Управление процессами

13. То же, что и в п. 3, но вместо процессов использовать потоки.

3. Для заданного каталога (аргумент 1 командной строки) и всех его подкаталогов вывести в заданный файл (аргумент 2 командной строки) и на консоль имена файлов, их размер и дату создания, удовлетворяющих заданным условиям:

1 – размер файла находится в заданных пределах от N1 до N2 (N1,N2 задаются в аргументах командной строки),

2 – дата создания находится в заданных пределах от M1 до M2 (M1,M2 задаются в аргументах командной строки). Процедура поиска для каждого подкаталога должна запускаться в отдельном процессе. Каждый процесс выводит на экран свой pid, полный путь, имя и размер просмотренного файла, общее число просмотренных файлов в подкаталоге. Число запущенных процессов в любой момент времени не должно превышать N (вводится пользователем). Проверить работу программы для каталога /usr/ размер 31000 31500 дата с 01.01.1970 по текущую дату N=6.

```
(base) alex@avocadobook:~$ ./script.sh /usr out.txt 31000 31500 19700101 20221017
/usr/bin snap 15964536 2022-10-05+19:55:12.1514024060 total 1631
(base) alex@avocadobook:~$ cat out.txt
/usr/bin snap 15964536 2022-10-05+19:55:12.1514024060 total 1631
(base) alex@avocadobook:~$
```

```
(base) alex@avocadobook:~$ cat script.sh
#!/bin/bash
#USAGE: ./script dir file min_size max_size [create_date_min [create_date_max [max_procs]]]
#outputs to FILE <PID FULL_PATH FILE_NAME FILE_SIZE TOTAL_FILES> for each subdirectory
#+in DIR by separate processes with total amount of its do not exceed MAX_PROC

max_procs=${7:-1}
before_date=$(date +%s -d ${6:-today})
from_date=$(date +%s -d ${5:-19700101})

find "$1" -type f -size +$3 -size -$4 -newermt @$from_date ! -newermt @$before_date \
    -execdir readlink -ne . \; \
    -printf " %f %s %C+ total " \
    -execdir bash -c 'ls -AF . | grep -cv "/" ' \; | \
tee "$2"
(base) alex@avocadobook:~$
```

Контрольные вопросы

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

Процесс - это каждая программа. Для каждой запускаемой программы создается отдельный процесс. В рамках процесса программе выделяется процессорное время, оперативная память и другие **системные ресурсы**. Получается, это основополагающая часть вычислительного процесса.

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

Дескриптор процесса - специальная структура со следующей информацией:

1. Идентификатор процесса (ProcessIdentifier(ID))
2. Тип или класс процесса, который определяет для ОС некоторые правила предоставления ресурсов.
3. Приоритет процесса, соответствии с которым ОС предоставляет ресурсы. В рамках одного класса процессов в первую очередь обслуживается более приоритетный процесс.
4. Переменная состояния, которая определяет в каком состоянии находится процесс (готовность к работе, состояние выполнения, ожидание устройства ввода/вывода и т. д.).
5. Защищенная область памяти, в которой хранится текущее значение регистров процессора, если процесс прерван, не закончив работу. Эта информация называется контекстом процесса(задачи).
6. Информация о ресурсах, которыми процесс владеет и имеет право пользоваться (указатели на открытые файлы, информация о независимых операциях вв/выв и т. д.)
7. Место памяти или адрес этого места для организации общения с другими процессами.
8. Параметры времени запуска (момент времени, когда процессор должен активизироваться и периодичность этой операции).

В Си можно использовать следующие функции:

```
extern struct task_struct *find_task_by_vpid(pid_t nr);  
extern struct task_struct *find_task_by_pid_ns(pid_t nr, struct pid_namespace *ns);
```

3) Какими способами можно организовать выполнение программ в фоновом режиме?

Дописать & в конце bash-команды.

Или командой `bg` перевести программу в фоновый режим.

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

Фоновое выполнение позволяет из одного терминала выполнять следующую команду не дожидаясь завершения предыдущей.

5) Как пользователь может повлиять на распределение ресурсов между активными процессами? Установить приоритет (nice, renice)

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

Послать сигналы.

- **SIGINT** - самый безобидный сигнал завершения, означает Interrupt. Он отправляется процессу, запущенному из терминала с помощью сочетания клавиш `Ctrl+C`. Процесс правильно завершает все свои действия и возвращает управление;
- **SIGQUIT** - это еще один сигнал, который отправляется с помощью сочетания клавиш, программе, запущенной в терминале. Он сообщает ей что нужно завершиться и программа может выполнить корректное завершение или проигнорировать сигнал. В отличие от предыдущего, она генерирует дампы памяти. Сочетание клавиш `Ctrl+/\`;
- **SIGHUP** - сообщает процессу, что соединение с управляющим терминалом разорвано, отправляется, в основном, системой при разрыве соединения с интернетом;
- **SIGTERM** - немедленно завершает процесс, но обрабатывается программой, поэтому позволяет ей завершить дочерние процессы и освободить все ресурсы;
- **SIGKILL** - тоже немедленно завершает процесс, но, в отличие от предыдущего варианта, он не передается самому процессу, а

обрабатывается ядром. Поэтому ресурсы и дочерние процессы остаются запущенными.

Необходимо знать айди процесса.

7) Перечислите базовые средства взаимодействия процессов в Linux.

Именованные каналы, общая память, семафоры.

8) Поясните особенности работы с каналами в Linux.

Каналы бывают именованные и неименованные (анонимные).

Анонимные каналы мы использовали для перенаправления ввода-вывода.

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

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

Функции отложенных вызовов выполняются в системном контексте, а не в контексте прерывания. Вызов этих функций выполняется не обработчиком прерывания таймера, а отдельным обработчиком отложенных вызовов, который запускается после завершения обработки прерывания таймера.

При обработке прерывания таймера система проверяет необходимость запуска тех или иных функций отложенного вызова и устанавливает соответствующий флаг для них. В свою очередь обработчик отложенных вызовов проверяет флаги и запускает необходимые в системном контексте.