

# Тема: Загрузка ОС и процессы.



# План занятия:

1. Загрузка операционной системы.
2. Процесс. Мониторинг процессов.
3. Атрибуты процессов.
4. Управление процессами.
5. Стандартные потоки.
6. Конвейер (pipeline).

# 1. Загрузка операционной системы.

Процесс загрузки операционной системы Linux может быть разделен на несколько основных этапов. Вот общий план загрузки ОС Linux:

## 1. Предварительный этап:

Включение компьютера и активация BIOS (Basic Input/Output System) или (UEFI) Unified Extensible Firmware Interface.

Проведение самотестирования (POST - Power-On Self Test), чтобы проверить работоспособность аппаратной части компьютера.

Загрузка BIOS (UEFI) и чтение информации из основной платы.

## 2. Загрузка загрузчика:

BIOS (UEFI) ищет устройство загрузки (например, жесткий диск, SSD или сетевой интерфейс).

Загрузчик (как GRUB, LILO или SYSLINUX) загружается в оперативную память компьютера с выбранного устройства загрузки.

Загрузчик ищет и загружает ядро операционной системы Linux.

## 3. Загрузка ядра:

Ядро операционной системы Linux загружается в оперативную память.

Инициализация ядра, включая настройку аппаратного обеспечения, драйверов и подсистем ядра.

#### **4. Инициализация пользовательского пространства.**

После загрузки ядра запускается пользовательское пространство.

Инициализация системных служб и демонов, таких как `systemd` или `init`.

Загрузка и инициализация драйверов устройств.

Монтирование файловых систем и проверка целостности файловой системы.

Запуск системных служб и пользовательского интерфейса:

#### **5. Запуск системных служб и демонов, необходимых для нормальной работы операционной системы.**

Запуск графической среды рабочего стола (если используется) или командной строки.

Вход пользователя:

#### **6. Отображение экрана входа пользователя.**

Проверка и аутентификация пользователя.

Запуск пользовательской сессии с соответствующими настройками и предоставление доступа к ресурсам.

# Загрузка операционной системы

BIOS/UEFI

GRUB

Linux kernel & initrd

systemd

terminal

## 2. Процесс. Мониторинг процессов.

### **Процесс в Операционной Системе Linux:**

**Процесс** в операционной системе Linux представляет собой экземпляр программы, запущенной в памяти компьютера. Каждый процесс имеет свой уникальный идентификатор (PID), который используется для управления и идентификации.

### **Элементы Процесса:**

Процесс включает в себя код программы, данные, регистры процессора, указатели на память, атрибуты и стек вызовов. Каждый процесс работает в своем собственном пространстве памяти, обеспечивая изоляцию от других процессов.

Процесс — одно из основополагающих понятий в ОС Linux. По сути, это совокупность какого-то кода, выполняющегося в памяти компьютера. Но есть приложения, которые могут создавать в результате своей работы не один, а несколько процессов. Каждая команда, которую мы выполняем в терминале, или приложение, которое мы запускаем в графической оболочке, также порождает процессы.

Для просмотра списка процессов в Linux вы можете использовать команду ps:

```
srv1@srv1:~$ ps
  PID TTY          TIME CMD
 1006 pts/0    00:00:00 bash
 1015 pts/0    00:00:00 ps
```

Вывести список всех процессов:

```
srv1@srv1:~$ ps -e
  PID TTY          TIME CMD
    1 ?           00:00:02 systemd
    2 ?           00:00:00 kthreadd
    3 ?           00:00:00 rcu_gp
    4 ?           00:00:00 rcu_par_gp
    5 ?           00:00:00 slub_flushwq
```

Вывести список всех процессов в полном формате:

```
srv1@srv1:~$ ps -ef
UID          PID    PPID  C STIME TTY          TIME CMD
root           1        0  0  02:12 ?           00:00:02 /sbin/init
root           2        0  0  02:12 ?           00:00:00 [kthreadd]
root           3        2  0  02:12 ?           00:00:00 [rcu_gp]
```

Вывести список процессов в виде дерева:

```
srvl@srvl:~$ ps -ef --forest
UID          PID     PPID  C  STIME TTY          TIME CMD
root          2         0  0  02:12 ?        00:00:00 [kthreadd]
root          3         2  0  02:12 ?        00:00:00 \_ [rcu_gp]
root          4         2  0  02:12 ?        00:00:00 \_ [rcu_par_gp]
```

Вывести список процессов, запущенных текущим пользователем:

```
srvl@srvl:~$ ps -u srvl
  PID TTY          TIME CMD
  929 ?        00:00:00 systemd
  930 ?        00:00:00 (sd-pam)
  936 tty1      00:00:00 bash
```

Вывести список процессов, сортируя их по использованию центрального процессора (CPU):

```
srvl@srvl:~$ ps -e --sort=-pcpu
  PID TTY          TIME CMD
 1159 ?        00:00:12 kworker/0:0-events
   635 ?        00:00:03 snapd
     1 ?        00:00:02 systemd
```



# Команды для мониторинга процессов

1. **ps**: используется для отображения текущих процессов.

```
srv1@srv1:~$ ps
  PID TTY          TIME CMD
 1006 pts/0        00:00:00 bash
 1195 pts/0        00:00:00 ps
```

2. **top**: предоставляет динамическое отображение текущих процессов и их использование ресурсов.

```
top - 03:02:30 up 50 min,  2 users,  load average: 0.01, 0.01, 0.00
Tasks: 106 total,   1 running, 105 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.7 us,  0.7 sy,  0.0 ni, 98.2 id,  0.0 wa,  0.0 hi,  0.4 si,  0.0 st
MiB Mem :   957.6 total,   255.9 free,   244.9 used,   456.8 buff/cache
MiB Swap:  1915.0 total,  1915.0 free,    0.0 used.   585.5 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
1159	root	20	0	0	0	0	I	0.7	0.0	0:25.09	kworker+
25	root	20	0	0	0	0	S	0.3	0.0	0:01.30	kcompac+
635	root	20	0	736468	53872	21072	S	0.3	5.5	0:03.91	snaped
1193	root	20	0	0	0	0	I	0.3	0.0	0:00.21	kworker+

# Команды для управления процессами

3. **kill**: используется для завершения процесса по его идентификатору (PID).

```
srv1@srv1:~$ kill 1200
```

Опция -9 может быть добавлена для отправки сигнала SIGKILL (принудительное завершение процесса).

```
srv1@srv1:~$ kill -9 1200
```

4. **pkill**: позволяет завершить процесс по его имени.

```
srv1@srv1:~$ pkill nginx
```

# Команды для управления процессами

5. **Htop** - это улучшенная версия команды `top` с графическим интерфейсом пользователя.

```
CPU[ | 2.8%] Tasks: 30, 34 thr; 1 running
Mem[||||||| 270M/958M] Load average: 0.06 0.02 0.00
Swp[ 0K/1.87G] Uptime: 00:59:27
```

PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	VMEM%	TIME+	Command
1	root	20	0	99M	13096	8424	S	0.0	1.3	0:02.39	/sbin/init
362	root	19	-1	47860	16032	14940	S	0.0	1.6	0:00.19	/lib/systemd/sy
401	root	RT	0	282M	27240	9072	S	0.0	2.8	0:00.62	/sbin/multipath
406	root	20	0	11896	6716	4448	S	0.0	0.7	0:00.13	/lib/systemd/sy
409	root	20	0	282M	27240	9072	S	0.0	2.8	0:00.00	/sbin/multipath
410	root	RT	0	282M	27240	9072	S	0.0	2.8	0:00.00	/sbin/multipath

6. **nohup**: позволяет запустить процесс, который будет продолжать выполняться даже после завершения текущей сессии пользователя.

```
srv1@srv1:~$ nohup ./hello.sh
```

# 3. Атрибуты процессов.

Вывод содержит различные столбцы с информацией о каждом процессе:

**USER:** Имя пользователя, от имени которого запущен процесс.

**PID:** Идентификатор процесса (Process ID) - уникальный числовой идентификатор для каждого процесса.

**PPID:** Идентификатор родительского процесса (Parent Process ID) - идентификатор процесса, который породил данный процесс.

**%CPU:** Процент использования центрального процессора (CPU) процессом.

**%MEM:** Процент использования оперативной памяти (RAM) процессом.

**VSZ:** Размер виртуальной памяти (Virtual Memory Size) процесса в килобайтах.

**RSS:** Размер собственно используемой физической памяти (Resident Set Size) процесса в килобайтах.

**TTY:** Терминал, связанный с процессом (если есть).

**STAT:** Статус процесса, например, R (работает), S (ожидание), Z (зомби) и т. д.

**START:** Время запуска процесса.

**COMMAND:** Команда или имя исполняемого файла, связанного с процессом.

# Некоторые состояния процесса

Процесс работает



Процесс спит



Процесс-зомби



**Родительский процесс** (Parent process) - это процесс, который породил другой процесс. В иерархии процессов в операционной системе Linux каждый процесс, кроме процесса с идентификатором 1 (init), имеет родительский процесс.

Когда новый процесс создается с помощью системного вызова, такого как `fork()`, родительский процесс порождает дочерний процесс. Родительский процесс передает дочернему процессу свое окружение, файловые дескрипторы и другие связанные ресурсы. Дочерний процесс наследует многие характеристики своего родительского процесса.

Родительский процесс обычно отвечает за управление своими дочерними процессами. Он может ожидать завершения дочернего процесса с помощью системных вызовов, таких как `wait()` или `waitpid()`, чтобы получить статус завершения и освободить ресурсы, связанные с дочерним процессом. Родительский процесс также может отправлять сигналы дочерним процессам для управления их выполнением.

Иерархия процессов в Linux представляет собой древовидную структуру, где каждый процесс имеет родительский процесс, за исключением процесса с идентификатором 1 (init), который является корневым процессом всех процессов в системе.

**Процесс-зомби** (Zombie process) - это особое состояние процесса в операционной системе Linux. Зомби-процесс возникает, когда процесс завершил свою работу, но его родительский процесс еще не получил статус завершения этого процесса.

Когда процесс завершается, он остается в системе в состоянии зомби до тех пор, пока его родительский процесс не запросит статус завершения этого процесса. Зомби-процесс не выполняет никаких действий и не использует системные ресурсы, за исключением небольшого объема информации о своем завершении.

Зомби-процессы обычно не представляют опасности для системы, и они автоматически удаляются из списка процессов, когда их родительский процесс получает статус завершения.

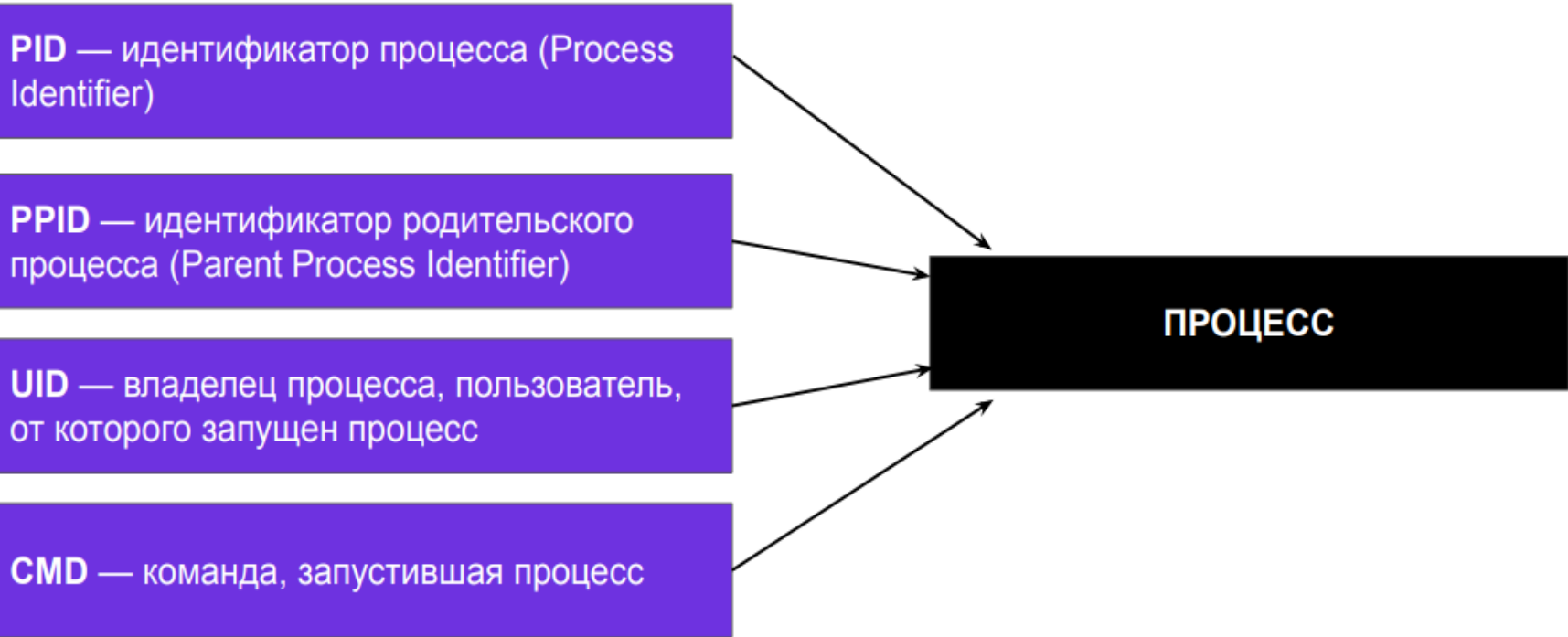
# Атрибуты процессов

**PID** — идентификатор процесса (Process Identifier)

**PPID** — идентификатор родительского процесса (Parent Process Identifier)

**UID** — владелец процесса, пользователь, от которого запущен процесс

**CMD** — команда, запустившая процесс



```
graph LR; PID[PID — идентификатор процесса (Process Identifier)] --> Process[ПРОЦЕСС]; PPID[PPID — идентификатор родительского процесса (Parent Process Identifier)] --> Process; UID[UID — владелец процесса, пользователь, от которого запущен процесс] --> Process; CMD[CMD — команда, запустившая процесс] --> Process;
```

**ПРОЦЕСС**



## 4. Управление процессами.

**systemd** - это система инициализации и управления службами в современных дистрибутивах Linux. Ее основной целью является управление процессами и службами, начиная с загрузки операционной системы и заканчивая её выключением.

**systemd** предоставляет множество утилит для управления службами, настройкой системы и отслеживания её состояния.

# Компоненты systemd

## systemd Utilities

systemctl journalctl notify analyze cglc cgtop loginctl nspawn

## systemd Daemons

systemd  
journald networkd  
logind user session

## systemd Targets

bootmode basic multi-user graphical user-session  
dbus telephony display service  
shutdown reboot dlog logind session tizen service

## systemd Core

manager unit login namespace log  
systemd service timer mount target multiseat inhibit  
snapshot path socket swap session pam cgroup dbus

## systemd Libraries

dbus-1 libpam libcap libcryptsetup tcpwrapper libaudit libnotify

## Linux Kernel

cgroups autofs kdbus

**systemctl**: Это основная утилита для управления службами в systemd. Она позволяет запускать, останавливать, перезапускать и проверять статус служб.

**journalctl**: Эта утилита используется для просмотра журналов системы, включая журналы служб и другие системные сообщения.

**analyze**: Эта утилита предоставляет информацию о времени загрузки системы и её компонентов.

**cgtop**: Эта утилита используется для отслеживания использования ресурсов в cgroups, которые являются механизмом управления ресурсами в systemd.

**resolve**: Утилита для настройки и управления системой разрешения DNS в systemd.

**hostnamectl**: Управление и просмотр информации о хостнейме системы.

**localectl**: Управление и просмотр информации о локали системы.

**systemctl** - это утилита управления службами в системах, использующих systemd. systemd - это система инициализации и управления службами в большинстве современных дистрибутивов Linux.

Основные параметры systemctl:

1. **systemctl status** выведет на экран состояние системы.
2. **systemctl** выведет список запущенных юнитов. С точки зрения systemctl, юнитом может быть служба, точка монтирования дискового устройства.
3. **systemctl [start|stop|status|restart|reload] service\_name** позволит запустить службу (start), остановить (stop), получить информацию о службе (status), перезапустить службу (restart), перечитать конфигурационный файл службы (reload).
4. **systemctl [enable|disable] service\_name** позволит добавить (enable) или убрать (disable) службу из автозагрузки.

```
srv1@srv1:~$ sudo systemctl status ssh
● ssh.service - OpenBSD Secure Shell server
   Loaded: loaded (/lib/systemd/system/ssh.service; enabled; vendor preset: en>
   Active: active (running) since Wed 2024-01-31 02:12:28 UTC; 1h 19min ago
     Docs: man:sshd(8)
           man:sshd_config(5)
  Main PID: 678 (sshd)
    Tasks: 1 (limit: 1013)
   Memory: 6.7M
      CPU: 105ms
   CGroup: /system.slice/ssh.service
           └─678 "sshd: /usr/sbin/sshd -D [listener] 0 of 10-100 startups"
```

## Запуск процессов независимо от терминала.

Если программа запускалась на терминале, при закрытии терминала процесс этой программы также закрывается.

Для того чтобы процесс, запущенный из терминала, продолжал работать после закрытия терминала, вы можете использовать несколько подходов:

1. Использование **nohup**: Вы можете использовать команду **nohup** (от "no hang up") для запуска процесса, который будет продолжать работать в фоновом режиме, даже после закрытия терминала. Пример использования:

```
nohup command &
```

2. Другие способы (disown, tmux, screen...)

## 5. Стандартные потоки

Стандартные потоки (standard streams) - это основные каналы ввода-вывода, которые связаны с каждым процессом в операционной системе.

Они предоставляют способ взаимодействия процесса с его окружением и пользователями.

Стандартные потоки в Linux используются для реализации концепции "все является файлом".

Вместо того, чтобы иметь специальные API для обработки ввода-вывода с разными устройствами (клавиатура, дисплей и т. д.), каждое устройство представлено как файл и может быть обработано с помощью файловых дескрипторов и стандартных потоков.

Файловый дескриптор	Открытые файлы
0	Стандартный поток ввода ( <b>STDIN</b> ). Файл, из которого осуществляется чтение данных.
1	Стандартный поток вывода ( <b>STDOUT</b> ). Файл, в который осуществляется запись данных.
2	Стандартный поток ошибок ( <b>STDERR</b> ). Файл, в который осуществляется запись об ошибках или сообщения, которые не могут быть записаны в стандартный поток вывода.
...	....
N	file_name



ПРОЦЕСС



**Файловый дескриптор** (file descriptor) - это целочисленное значение, которое операционная система использует для идентификации открытых файлов или других входно-выходных устройств.

Файловые дескрипторы представляют собой абстракцию, через которую программы могут взаимодействовать с файлами, сетевыми соединениями и другими ресурсами системы.

В большинстве операционных систем, включая Linux, файловые дескрипторы представлены целыми числами.

Они являются неотрицательными значениями, где определенные диапазоны дескрипторов могут быть зарезервированы для стандартных потоков ввода-вывода:

- 0: стандартный ввод (stdin)
- 1: стандартный вывод (stdout)
- 2: стандартный поток ошибок (stderr)

Когда процесс открывает файл или устанавливает соединение, операционная система возвращает файловый дескриптор, который ассоциируется с этим ресурсом.

Затем программы могут использовать этот файловый дескриптор для чтения, записи или других операций над ресурсом.

**Стандартный ввод (stdin):** Обозначается как stdin или файловый дескриптор 0. Он представляет собой поток ввода, через который процесс может принимать данные.

Обычно это клавиатура или другой процесс, который перенаправляет свой вывод.

В примере команда `cat > file.txt` открывает файл `file.txt` для записи и ждет ввода текста от пользователя.

Пользователь может вводить текст в командную строку, а затем, нажав `Ctrl + D` (в Unix-подобных системах), завершить ввод и сохранить введенный текст в файле `file.txt`.

```
srv1@srv1:~$ cat > file.txt
hello world
srv1@srv1:~$ cat file.txt
hello world
```

**Стандартный вывод (stdout):** Обозначается как stdout или файловый дескриптор 1.

Он представляет собой поток вывода, через который процесс отправляет свои данные.

Обычно это консоль или другой процесс, который может перенаправить.

```
srvl@srvl:~$ cat > file.txt
hello world
srvl@srvl:~$ cat file.txt
hello world
srvl@srvl:~$ echo Hello!
Hello!
srvl@srvl:~$ ls -l > file.txt
srvl@srvl:~$ cat file.txt
total 4
-rw-rw-r-- 1 srvl srvl    0 Jan 29 07:17 document
-rw-rw-r-- 1 srvl srvl    0 Jan 31 03:48 file.txt
drwxrwxr-x 2 srvl srvl 4096 Jan 29 07:24 newfolder
-rw----- 1 srvl srvl    0 Jan 31 03:15 nohup.out
srvl@srvl:~$
```

В примере команда `echo "Hello!"` выводит строку "Hello!" в стандартный вывод.

Когда эта команда выполняется в командной строке, она просто отображает указанную строку на экране.

Команда `ls -l` используется для отображения содержимого текущего каталога с дополнительной информацией о файлах и каталогах.

Оператор перенаправления `>` перенаправляет вывод команды `ls -l` в файл `file.txt`.

Результат выполнения команды сохраняется в файле `file.txt`, а не отображается на экране.

**Стандартный поток ошибок**, обозначаемый как `stderr` файловый дескриптор 2 (стандартный вывод ошибок), предназначен для вывода сообщений об ошибках и диагностических данных.

В UNIX-подобных системах, таких как Linux, `stderr` обычно направлен на тот же устройство вывода, что и `stdout` (стандартный вывод), по умолчанию это экран терминала.

Основные особенности `stderr`:

Можно перенаправить `stderr` в файл, чтобы сохранить сообщения об ошибках:

```
srv1@srv1:~$ ./hello.sh
-bash: ./hello.sh: No such file or directory
srv1@srv1:~$ ./hello.sh 2>error.txt
srv1@srv1:~$ cat error.txt
-bash: ./hello.sh: No such file or directory
srv1@srv1:~$
```

Можно игнорировать stderr, используя `/dev/null` (специальное устройство в UNIX, представляющее ничего):

```
srv1@srv1:~$ ./hello.sh 2>/dev/null
srv1@srv1:~$
```

Можно комбинировать stdout и stderr и направить их в один и тот же файл:

```
srv1@srv1:~$ ./hello.sh > out_error.txt 2>&1
srv1@srv1:~$
```

## 6. Конвейер (pipeline)

Конвейер (pipeline) в Unix-подобных операционных системах представляет собой механизм для передачи вывода одной программы в качестве ввода другой программе.

Конвейер обеспечивает эффективный и гибкий способ объединения различных команд для выполнения сложных задач.

Символ **|** используется для создания конвейера в командной строке Linux. Результат выполнения первой команды передается вводом во вторую команду, и так далее, по цепочке.

# Конвейер (pipeline)

В этом примере команда `ls -l` отображает содержимое текущего каталога с дополнительной информацией о файлах и каталогах.

Результат выполнения этой команды передается вводом в команду `grep ".txt"`, которая фильтрует строки, содержащие ".txt".

Таким образом, вывод команды `ls -l` фильтруется и отображается только строки, относящиеся к файлам с расширением ".txt".

```
$ ls -l | grep ".txt"
```



# Конвейер (pipeline)

Отображение содержимого файла с использованием cat и фильтрация строк с использованием grep:

```
cat file.txt | grep "pattern"
```

Фильтрация и сортировка результатов:

```
cat data.txt | grep "keyword" | sort
```

Подсчет уникальных строк в файле:

```
cat file.txt | sort | uniq
```

# Домашнее задание

1. Повторить материалы лекции.
2. Изучить методичку доп. материалы.