# Семинар 1 ОС (09.02.2022)

## Демоны ведение

У демона нет группы, и он является лидером сессии.

Смысл демона - он начинает выполнять какую-то работу, когда в этой работе возникает необходимость. (как правило находится в состоянии прерываемого сна)

Большинство демонов, которые мы рассматривали имеет права суперпользователя и идентификатор user0.

TPGID = -1 (это идентификатор группы переднего плана)

Daemon is a background process. (Но он либо блокирован, либо выполняется с наивысшим приоритетом)

## Демоны

Существует 6 правил, которые необходимо выполнять.

ps -ajx – показать демона  
(идентификаторы: собственный (PID) = группы = сессии)

Демон будет усыновлен и это будет видно в ParentPID (обычно = 1)

1. Сброс маски режима создания файлов функцией unmask(0);  
   Демон создаётся как потомок. Поэтому он наследует маски предка. Их надо сбросить. Это делается для того, чтобы демон беспрепятственно мог создавать файлы с нужными правами доступа.
2. Вызов fork() и завершение процесса предка.  
   Когда мы вызываем системный вызов fork, создаётся новый процесс который наследует … и идентификатор группы процессов от родителя. Завершив предка, мы делаем так, что процесс потомок не будет являться лидером группы.
3. Вызвать функцию setsid();  
   Эта функция делает процесс лидером новой сессии, лидером новой группы (и в этой группе он один) и лишает процесс управляющего терминала.
4. chdir(“/”);  
   Меняем директорию на корневой каталог. ?Делается это для того что если демон запущен с подмонтированой …, то поскольку демон может выполнять служебную функцию, может возникнуть проблема при отмонтировании? (Пример с флешкой)
5. Необходимо закрыть все файловые дескрипторы.  
   Файлы открытые предком не нужны – это предотвратит держание в открытом состоянии файлов, которые были унаследованы.  
   Для этого используется функция getrlimit(), которая вернёт нам, максимально возможный номер дескриптора, который программа может открыть, а нам надо закрыть все дескрипторы до этого номера.  
   if(getrlimit(RLIMIT\_NOFILE, &r1) < 0)  
    erl\_quit(“%s:невозм. Получ. Макс. Номер дескр.”, cmd);  
   ----------------------  
   if(r1.rlim\_max==RLIMIT\_INFINITY)  
    r1.rlim\_max = 1024  
   for (i=0; i<r1.rlim\_max; i++)  
    close(i);
6. Открываются файловые дескрипторы с номерами 0, 1, 2 (stdin, stdout, stder) на устройстве /dev/null (так называемое нулевое устройство – вся информация будет утрачена) для того, чтобы избежать стандартных ошибок при работе с потоками ввода и вывода. То есть в результате библиотечные функции, которые будут пытаться писать, читать писать в стандартный поток ошибок, никакого влияния оказывать на работу демона не будут. (Защита от дурака)

Почему у демона не должно быть собственного терминала (то есть почему пользователь не должен взаимодействовать с ним как с обычной программой). Это объясняется задачами, которые ставятся перед демоном. Они выполняют свои, строго определённые функции. Выполняют определённую работу и влиять на него через терминал не надо.

Делаем по примеру (Листинг 13.3)

Функция daemonize (Листинг 13.1)

В функции daemonize, кроме перечисленных действий используется функция sigaction, в результате вызова которой будет игнорироваться сигнал sighub. Это сигнал, который процесс получает, если он утратил управляющий терминал. А затем восстанавливается реакция на этот сигнал.

Функция Already running (Листинг 13.2 – демон в единственном экземпляре). Нельзя чтобы работало сразу несколько демонов, иначе возникнет неоднозначность.

Файл, в который демон записывает свой идентификатор должен быть блокирован (lockfile). Следующий демон попытается записать свой getpid(), не получится и демон завершится. Но блокировать обычный файл не надёжно. Поэтому используется файл режима ядра.

Поэтому запускать демона нужно с правами superuser’а.

## По файлам пошли

\*В Листинге 4.7 используются глобальные переменные только в одной функции (иначе слишком длинный заголовок у функции).

7 типов файлов в Linux

1. Регулярные обозначается как прочерк “–“  
   Обычные (regular) – которые мы используем для долговременного хранения.
2. Директории “d”
3. Символьная ссылка “l”
4. Труба pipe “p”
5. Сокет “s”
6. Специальный файл символьного устройства “c”
7. Специальный файл блочного устройства “b”

6 и 7 можем увидеть только в dev

В системе определена структура  
struct stat  
{  
 dev\_t st\_dev; //dev – device // inode’s dev  
 ino\_t st\_ino; // индекс (номер) inode’а  
 mode\_t st\_mode; // битовая маска, которая определяет права доступа  
 nlint\_t st\_nlink; // кол-во жёстких ссылок   
 uid\_t st\_uid; //user id  
 gid\_t st\_gid; //group id  
 dev\_t st\_rdev; // rdev - real device // dev type  
 off\_t st\_size; //размер файла в байтах  
 blksize\_t st\_blksize; // размер блока  
 blkcnt\_t st\_blocks; // кол-во блоков  
 //информация о времени  
 time\_t st\_atime; // a – access время доступа  
 time\_t st\_mtime; // m – modification время модификации  
 time\_t st\_ctime; // c – change время изменения  
 int64\_t st\_blocks; //новая версия  
};

Это всё integer. Разные имена типов, чтобы сделать дополнительный контроль типов.