**Демон**

**q: Запустить, показать строку процесса-демона после выполнения команды**

a: ps -ajx.

* сa : процессы, связанные с текущим терминалом, а также процессы других пользователей;
* x : процессы, отсоединенные от терминала.
* j: Print information associated with the following keywords: **user**, **pid**, **ppid**, **pgid**, **sess**, **jobc**, **state**, **tt**, **time**, and **command**.

//в некоторых системах: ps ajx без дефиса

Вопросы:

**q: Что значат состояния Ss Sl?**

a:  
 S - interruptible sleep (waiting for an event to complete)  
 s - is a session leader  
 l - is multi-threaded (using CLONE\_THREAD, like NPTL pthreads do)  
 L - has pages locked into memory (for real-time and custom IO)  
 (подробнее: >man ps)

**q: Рассказать правила и показать, где в коде они применены**

-----------------------------------------------------------------------------------------------

a: Открыть метод demonize в коде и рассказать 6 правил программирования процессов-демонов:

1) вызвать umask для сброса маски создания файлов

- маска наследуется и может маскировать биты прав доступа (запись, чтение)

2) вызвать fork() и завершить предка

- чтобы командная оболочка думала, что команда была выполнена

- чтобы новый процесс гарантированно не был лидером группы, что позволит вызвать setsid (у дочернего процесса id отличный от родителя, а pgid наследуется)

3) создать новую сессию, вызвав setsid, тогда процесс станет:

- лидером новой сессии

- лидером новой группы процессов

- лишится управляющего терминала (TTY = ?)

4) сделать корневой каталог текущим рабочим каталогом

- если рабочий каталог на смонтированной файловой системе, то её нельзя будет отмонтировать, так как процессы-демоны обычно живут, пока система не перезагрузится

5) закрыть все ненужные открытые файловые дескрипторы, которые процесс-демон может унаследовать и препятствовать их закрытию (для этого нужно сначала получить максимально возможный номер файлового дескриптора (см. код))

6) такой процесс не связан ни с каким терминальным устройством и не может взаимодействовать с пользователем в интерактивном режиме, даже если он был запущен в рамках интерактивной сессии, он все равно будет переведен в фоновый режим (некоторые процессы-демоны открывают файловые дескрипторы 0 1 и 2 на dev/null - "пустые" stdin, stdout, stderr, что позволяет вызывать в них функции стандартного ввода вывода, не получая при этом ошибок)

---------------------------------------------------------------------------------------------------

**q: Зачем закрывать файловые дескрипторы?**

a: Это ненужные процессу-демону дескрипторы, унаследованные от предка, закрытию которых он может препятствовать

**q: Зачем сигнал SIGHUP игнорируется в демоне?**

a: Чтобы мы не получали сигнал о том, что был потерян управляющий терминал, что вообще-то приведёт к завершению процесса

**q: Зачем мы подключили файловые дескрипторы 0, 1, 2?**

a: Чтобы можно было использовать функции стандартных библиотек ввода-вывода и они не выдавали ошибки

**q: Почему три цифры в PID одинаковые, а в TTY знак вопроса?**

a: Демон является лидером группы и сессии. Знак вопроса означает, что нет управляющего терминала

**q: Покажите PID процесса в lock-файле**

a: Переходим в папку var/run, показываем содержимое файла mydemon.pid (или см. в коде имя LOCKFILE)

>cd /var/run

>cat mydemon.pid

**q: Что такое процесс-демон?**

a: Это процесс, не имеющий управляющего терминала, который соответственно не взаимодействует с пользователем. Как правило демоны предоставляют системе различные сервисы.

a2: Это фоновый процесс, предназначенный для автономной работы, практически без вмешательства пользователя.

**q: Покажите место в коде, где процесс становится демоном**

a: Вызов функции demonize(cmd) в main.c

**q: Что мы тут делаем(там где sigaction, SIGHUP)**

a: Игнорируем сигнал о потере терминала. (SIGHUP говорит, что терминал был потерян и SIGHUP завершает процесс, тк нам это не нужно, просто игнорируем его)

**q: что значит TPGID == -1?**

a: процесс не имеет управляющего терминала (TPGID показывает id процесса, который в данный момент выполняется на этом терминале, тк терминала нет, то и процесса быть не может)

**q: почему мы имеем право использовать umask в процессе предке?**

a: потому что потомок наследует маску режима создания файлов, а сам предок вскоре завершается

**q: почему демон не должен иметь управляющего терминала?**

a: потому что демон предоставляет системы важные сервисы и у пользователя не должно быть возможности **влиять/управлять (другое слово не подходит)** на него **!ALERT!** на этот вопрос в ее дряхлой ебанутой башке заученный ответ, смысл которого совпадает с этим, но в правильности формулировки я не уверен

**q: зачем функция already\_running?**

a: чтобы в системе не было одинаковых работающих демонов

**q: почему демон не может быть запущен больше 1 раза?**

а: Причиной такого поведения может служить, например, требование монопольного владения каким-либо ресурсом. (из книжки 13.5)

(этот ответ её не совсем устроил, она что-то сказала типа “зачем одну и ту же задачу несколько раз выполнять??!” Но какой точный ответ на этот вопрос непонятно…)

**q: почему мы делаем проверку на if (rl.rlim\_max == RLIM\_INFINITY)?**

а: не уверен насчет правильности ответа, но я сказал что “ есть вероятность не получить значение максимального открытого дескриптора, следовательно установим его значение максимальному и закроем все к чертям собачьим” слово в слово (нет)

(The value RLIM\_INFINITY, defined in [*<sys/resource.h>*](https://pubs.opengroup.org/onlinepubs/7908799/xsh/sysresource.h.html), is considered to be larger than any other limit value. If a call to *getrlimit()* returns RLIM\_INFINITY for a resource, it means the implementation does not enforce limits on that resource. Specifying RLIM\_INFINITY as any resource limit value on a successful call to *setrlimit()* inhibits enforcement of that resource limit.  <https://pubs.opengroup.org/onlinepubs/7908799/xsh/getrlimit.html>)

**q: С терминала можем убить демона, имея права superuser?**

a: да

**q: А если попробовать убить демона из user mode?**

a: Недостаточно прав

**q: Какой сон можно и нельзя прервать?**

a: Блокировку на ввод вывод прерывать нельзя

**q: Объяснить, какие значения присваиваете полям структуры flock в функции lockfile**  
a: Блокируем файл на запись с самого начала файла (F\_WRLCK и SEEK\_SET). (Примечание: Также можно было не писать свою функцию lockfile, а использовать flock())

**q: почему мы запускаем демона от суперпользователя? Что нужно в коде поменять, чтобы не обязательно было так делать?**

а: это из-за локфайла: он создается в системной директории, к системным директориям доступ есть только у суперпользователя. Мы можем поменять путь к локфайлу, заменив его на пользовательский каталог.

чтобы пользователь не мог иметь доступа к файлу

**q: а если мы локфайл создадим в пользовательском каталоге? Блокировка будет работать?**

Блокировка не будет работать, так как раз мы создали файл в пользовательском каталоге, после запуска демона мы его можем удалить, и затем запустить второго демона.