Правила программирования демонов

- Вызвать функцию umask, чтобы сбросить маску режима создания файлов в значение 0. Маска, наследуемая от запускающего процесса, может маскировать некоторые биты прав доступа. Если предполагается, что процесс-демон будет создавать файлы, может потребоваться установка определенных битов прав доступа.

- Вызвать функцию fork и завершить родительский процесс. Во-первых, если демон запущен как обычная команда оболочки, завершив родительский процесс, мы заставим командную оболочку думать, что команда выполнилась. Во-вторых, дочерний процесс наследует идентификатор группы процессов от родителя, но получает свой идентификатор процесса; тем самым гарантируется, что дочерний процесс не будет являться лидером группы, а это необходимое условие для вызова функции setsid, который будет произведен далее.

- Создать новый сеанс, обратившись к функции setsid. При этом процесс становится лидером нового сеанса, лидером новой группы процессов и лишается управляющего терминала.

- Сделать корневой каталог текущим рабочим каталогом. Текущий рабочий каталог, унаследованный от родительского процесса, может находиться на смонтированной файловой системе. Поскольку демон, как правило, существует все время, пока система не перезагрузится, в подобной ситуации, когда рабочий каталог демона находится в смонтированной файловой системе, ее невозможно будет отмонтировать.

- Закрыть все ненужные файловые дескрипторы. Это предотвращает удержание в открытом состоянии некоторых дескрипторов, унаследованных от родительского процесса (командной оболочки или другого процесса).

- Некоторые демоны открывают файловые дескрипторы с номерами 0, 1 и 2 на устройстве /dev/null, — таким образом, любые библиотечные функции, которые пытаются читать со стандартного устройства ввода или писать в стандартное устройство вывода или сообщений об ошибках, не будут оказывать никакого влияния. Поскольку демон не связан ни с одним терминальным устройством, он не сможет взаимодействовать с пользователем в интерактивном режиме. Даже если демон запущен в интерактивном сеансе, он все равно переходит в фоновый режим, и начальный сеанс может завершиться без воздействия на процесс-демон. С этого же терминала в систему могут входить другие пользователи, и демон не должен выводить какую-либо информацию на терминал, да и пользователи не ждут, что их ввод с терминала будет прочитан демоном.

Реализация

Демоны — это долгоживущие процессы. Зачастую они запускаются во время загрузки системы и завершают работу вместе с ней. Так как они не имеют управляющего терминала, говорят, что они работают в фоновом режиме. В системе UNIX демоны решают множество повседневных задач.

Демон не должен иметь управляющего терминала, потому что демон выполняет важные функции, работает длительное время, следовательно, воздействовать на демонов с терминала нельзя.

Листинг 1: Файл daemon.c

```
1 #include <syslog.h>
2 #include <stdlib.h>
```

```c
#include <fcntl.h>
#include <sys/resource.h>
#include <sys/stat.h> //umask
#include <unistd.h> //setsid
#include <stdio.h> //perror
#include <signal.h> //sidaction
#include <string.h>
#include <errno.h>
#include <sys/file.h>

#define LOCKFILE "/var/run/daemon.pid"
#define LOCKMODE (S_IRUSR | S_IWUSR | S_IRGRP | S_IROTH)

int lockfile(int fd)
{
    struct flock fl; // структура для управления блокировкой
    fl.l_type = F_WRLCK; // блокировка записи
    fl.l_start = 0; // начальное смещение блокировки (начало файла при
       l_whence = SEEK_SET)
    fl.l_whence = SEEK_SET;
    fl.l_len = 0; //кол-во байт, которые буду заблокированы (0 означает блокир
       овку всех байтов, начиная от позиции, заданной l_whence и l_start до ко
       нца файла)
    return(fcntl(fd, F_SETLK, &fl)); // выполняет операции над файловым дескри
       птором
}

int already_running(void)
{
    syslog(LOG_ERR, "Проверка на многократный запуск");

    int fd;
    char buf[16];

    fd = open(LOCKFILE, O_RDWR | O_CREAT, LOCKMODE);

    if (fd < 0)
    {
        syslog(LOG_ERR, "невозможно открыть %s: %s", LOCKFILE, strerror(errno)
           );
        exit(1);
    }

    syslog(LOG_WARNING, "Lock-файл открыт");
    flock(fd, LOCK_EX | LOCK_UN);
    if (errno == EWOULDBLOCK) {
        syslog(LOG_ERR, "невозможно установить блокировку на %s: %s!",
           LOCKFILE, strerror(errno));
        close(fd);
        exit(1);
    }

    syslog(LOG_WARNING, "Записываем PID");

    ftruncate(fd, 0); // укорачивает файл до указанной длины
```

```
52      sprintf(buf, "%ld", (long)getpid());
53      write(fd, buf, strlen(buf) + 1);
54
55      syslog(LOG_WARNING, "Записали PID");
56
57      return 0;
58  }
59
60  void daemonize(const char *cmd)
61  {
62      int fd0, fd1, fd2;
63      pid_t pid;
64      struct rlimit rl;
65      struct sigaction sa;
66
67      // 1 правило. Сбрасывание маски режима создания файла
68      // umask можно вызвать в процессе предке, потому что потомок наследует мас
            ку режима создания файлов, а сам предок вскоре завершается
69
70      umask(0);
71
72      // Получение максимального возможного номера дискриптора
73      if (getrlimit(RLIMIT_NOFILE, &rl) < 0)
74          perror("Невозможно получить максимальный номер дискриптора\n");
75
76      // 2 правило. Стать лидером новой сессии, чтобы утратить управляющий терми
            нал
77      if ((pid = fork()) < 0)
78          perror("Ошибка функции fork\n");
79      else if (pid != 0) //родительский процесс
80          exit(0);
81
82      // 3 правило. Создать новую сессию
83      setsid();
84
85      // Игнорируем сигнал SIGHUP о потере терминала, который приведет к заверше
            нию процесса
86      sa.sa_handler = SIG_IGN;
87      sigemptyset(&sa.sa_mask);
88      sa.sa_flags = 0;
89      if (sigaction(SIGHUP, &sa, NULL) < 0)
90          perror("Невозможно игнорировать сигнал SIGHUP\n");
91
92
93      // 4 правило. Назначить корневой каталог текущим рабочим каталогом,
94      // чтобы впоследствии можно было отмонтировать файловую систему
95      if (chdir("/") < 0)
96          perror("Невозможно назначить корневой каталог текущим рабочим каталого
                м\n");
97
98      // 5 правило. Закрыть все файловые дескрипторы
99      if (rl.rlim_max == RLIM_INFINITY)
100         rl.rlim_max = 1024;
101     for (int i = 0; i < rl.rlim_max; i++)
102         close(i);
```

```
103
104     // 6 правило. Присоеденить файловые дескрипторы 0, 1, 2 к /dev/null, чтобы
            можно было использовать функции стандартных библиотек ввода–вывода и о
        ни не выдавали ошибки
105
106     fd0 = open("/dev/null", O_RDWR);
107     fd1 = dup(0); //копируем файловый дискриптор
108     fd2 = dup(0);
109
110     // Инициализировать файл журнала
111     openlog(cmd, LOG_CONS, LOG_DAEMON);
112     if (fd0 != 0 || fd1 != 1 || fd2 != 2)
113     {
114         syslog(LOG_ERR, "ошибочные файловые дескрипторы %d %d %d\n", fd0, fd1,
                fd2);
115         exit(1);
116     }
117
118     syslog(LOG_WARNING, "Daemon is running");
119
120 }
121
122 int main()
123 {
124     daemonize("daemon"); // процесс становится демоном
125     // Блокировка файла для одной существующей копии демона
126     if (already_running() != 0)
127     {
128         syslog(LOG_ERR, "Daemon is running\n");
129         exit(1);
130     }
131
132     syslog(LOG_WARNING, "Daemon is running");
133     while(1)
134     {
135         syslog(LOG_INFO, "Daemon is running");
136         sleep(5);
137     }
138 }
```

Результаты работы программы

На рисунках 1-4 продемонстрирована работа программы.

ps -axj
a - для вывода процессов, которыми владеют другие пользователи.
x - для вывода процессов, не имеющих управляющего терминала.
j - для вывода дополнительных сведений, имеющих отношение к заданиям: идентификатора сеанса, идентификатора группы процессов, управляющего терминала и идентификатора группы процессов терминала.

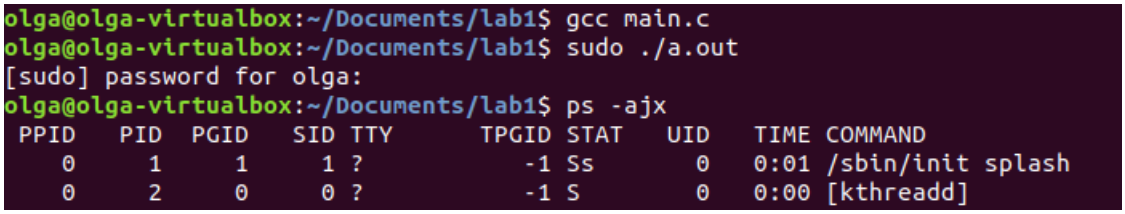

Рис. 1: Запуск демона и вызов ps -ajx

PPID: ID родительского процесса.
PID: ID процесса.
PGID: ID группы процесса.
SID: ID сессия.
В трех столбцах одинаковые цифры, т.к. демон является лидером группы и сессии (и он один в группе).
TTY: управляющий терминал (знак ? означает, что у демона нет терминала).
TPGID: ID группы, владеющей управляющим терминалом данного процесса (-1 означает, что у демона нет терминала).
STAT: S - прерываемый сон, s - лидер сессии.

Рис. 2: Процесс демон (./a.out)

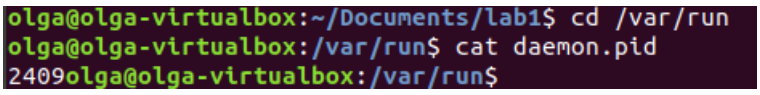

Рис. 3: PID процесса в lock-файле

```
olga@olga-virtualbox:~/Documents/lab1$ tail -f /var/log/syslog
Jul  9 00:38:35 olga-VirtualBox systemd[1]: Created slice system-clean\x2dmount\
x2dpoint.slice.
Jul  9 00:38:35 olga-VirtualBox systemd[1]: Started Clean the /media/olga/VBox_G
As_6.1.2 mount point.
Jul  9 00:38:36 olga-VirtualBox udisksd[464]: Mounted /dev/sr0 at /media/olga/VB
ox_GAs_6.1.2 on behalf of uid 1000
Jul  9 00:38:36 olga-VirtualBox gnome-shell[1456]: GNOME Shell started at Thu Ju
l 09 2020 00:38:32 GMT+0300 (MSK)
Jul  9 00:38:44 olga-VirtualBox pulseaudio[1478]: [pulseaudio] bluez5-util.c: Ge
tManagedObjects() failed: org.freedesktop.DBus.Error.TimedOut: Failed to activat
e service 'org.bluez': timed out (service_start_timeout=25000ms)
Jul  9 00:38:44 olga-VirtualBox dbus-daemon[1288]: [session uid=1000 pid=1288] A
ctivating via systemd: service name='org.gnome.Terminal' unit='gnome-terminal-se
rver.service' requested by ':1.59' (uid=1000 pid=1809 comm="/usr/bin/gnome-termi
nal.real " label="unconfined")
Jul  9 00:38:44 olga-VirtualBox systemd[1263]: Starting GNOME Terminal Server...
Jul  9 00:38:44 olga-VirtualBox dbus-daemon[1288]: [session uid=1000 pid=1288] S
uccessfully activated service 'org.gnome.Terminal'
Jul  9 00:38:44 olga-VirtualBox systemd[1263]: Started GNOME Terminal Server.
Jul  9 00:39:25 olga-VirtualBox daemon: демон работает
Jul  9 00:39:30 olga-VirtualBox daemon: демон работает
```

Рис. 4: Содержимое syslog