Терентьев Алексей Борисович

Программы-демоны в Linux

Материал
для лабораторных работ по курсу
ОПЕРАЦИОННЫЕ СИСТЕМЫ

Санкт-Петербургский государственный политехнический университет

Компиляция и сборка

gcc

GCC

GCC - GNU compiler collection - набор компиляторов для различных языков программирования, разработанный в рамках проекта GNU.

- Разное:

```
gcc file.c -o file
g++ file.c -o file
```

- Компиляция и линковка нескольких файлов:

```
gcc -c file1.c file2.c file3.c
gcc -o total file1.o file2.o file3.o
```

Демоны в Linux

Определение

Демон (daemon) — компьютерная программа в системах класса UNIX, запускаемая самой системой и работающая в фоновом режиме без прямого взаимодействия с пользователем.

Как породить демона

- Создание обычного процесса(родительского)
- Создание дочернего процесса родительским (fork)
- Завершение родительского процесса
- Регистрация обработчика сигналов (signal)
- Создание новой сессии для процесса (setsid) + второй fork и убийство родителя
- Сброс маски режима создания пользовательских файлов (umask)
- Смена текущей директории на “/”(chdir) и закрытие потоков stdin, stdout, stderr (close)
- Запуск рабочего цикла демона

fork

fork() — системный вызов, создающий новый процесс, который является практически полной копией родителя.

Различия:

- PID(потомка) $\neq$ PID(родителя);
- PPID(потомка) = PID(родителя);
- Потомок получает копию таблицы файловых дескрипторов родителя.
- Для потомка очищаются все ожидающие доставки сигналы;
- Временная статистика выполнения процесса-потомка в таблицах ОС обнуляется;

fork(2)

```
pid_t pid = fork();
if (pid == -1) {
    perror("fork failed");
    exit(EXIT_FAILURE);
}
else if (pid == 0) { \\child process
    printf("Hello from the child process!\n");
    _exit(EXIT_SUCCESS);
}
else { \\parent process
    int status;
    (void)waitpid(pid, &status, 0);
}
return EXIT_SUCCESS;
```

Кроме fork

- Стать лидером группы процессов и лидером сессии, после этого происходит отделение от терминала: `setsid();`
- Сброс маски режима создания пользовательских файлов (umask): `umask(0);`
- Смена текущей директории на “/” и закрытие потоков stdin, stdout, stderr:

```
chdir(“/”);
close(STDIN_FILENO);
close(STDOUT_FILENO);
close(STDERR_FILENO);
```

Для чего <function_name>?

setsid

- Создает новую сессию (сеанс, session), в которой наш процесс становится лидером.
- Вместе с сессией создается и новая группа процессов (process group)
- Создание новой сессии позволяет отделиться от управляющего терминала, который мог бы слать ненужные сигналы демону (например, SIGHUP при закрытии терминала)
- Сигналы также могут посылаться группе процессов, так что необходимо создавать собственную группу процессов

fork

- Первый fork нужен, чтобы процесс-потомок запустил функцию демона, когда родитель продолжит работу или завершится + потомок точно не является лидером сессии для успеха вызова setsid.
- Иногда fork вызывается второй раз, после setsid, чтобы процесс перестал быть лидером сессии (сеанса), тк в некоторых системах лидеру сессии может быть предоставлен управляющий терминал.

umask

- umask(0) (иногда - другое значение) вызывается для того, чтобы демон имел определенное поведение про создании файлов и не наследовал umask от родителя, которому можно указать любой

chdir

- chdir('/') если оставить демона работать в некоторой произвольно директории, это может помешать размонтированию файловой системы, содержащей эту директорию

close

- Несмотря на отделение от терминала, демон еще может иметь открытые файлы, файлы в терминале, что может привести к проблемам

Обработка сигналов

Сигналы

Сигналы — сообщения, высылаемые программе ОС или другой программой.

- Идентифицируются числами от 1 до 31
- Не содержат аргументов
- Прерывают выполнение программы (один из потоков, который и обрабатывает сигнал). Прерван может быть и сам обработчик сигналов
- Сигналы могут:
  - Обрабатываться по умолчанию
  - Обрабатываться пользовательскими обработчиками
  - Игнорироваться

Виды сигналов

- **SIGINT** - сигнал, получаемый при нажатии CTRL-C при работе (foreground) в терминале;
- **SIGHUP** - сигнализирует о том, что контролирующий терминал был уничтожен;
- **...**
- **SIGUSR1, SIGUSR2** - пользовательские сигналы, нет стд. обработчиков;
- **SIGKILL, SIGSTOP** - не переопределяются(программа завершилась аномально/остановка программы)

Обработка сигналов

```
void sig_handler(int signum)
{
   printf("Received signal %d\n", signum);
}
int main()
{
   signal(SIGINT, sig_handler);
   sleep(10); // This is your chance to press CTRL-C
   return 0;
}
```

sigaction

int sigaction(int signum, const struct sigaction *act, struct sigaction *oldact);

```
struct sigaction
{
   void (*sa_handler)(int signum);//обработчик(как в signal)
   void (*sa_sigaction)(int signum, siginfo_t *siginfo,
   void *uctx);//более продвинутый обработчик(SA_SIGINFO )
   sigset_t sa_mask;//маска блокируемых сигналов
   int sa_flags;//флаги
   void (*sa_restorer)(void);//устаревший элемент
};
```

sigaction(2)

```
void sighandler(int signum, siginfo_t *info, void *ptr)
{
   printf("Received signal %d\n", signum);
   printf("Sig from %lu\n",    (unsigned long)info->si_pid);
}
int main()
{
   memset(&act, 0, sizeof(act));
   act.sa_sigaction = sighandler;
   act.sa_flags = SA_SIGINFO;
   sigaction(SIGTERM, &act, NULL);
   sleep(100);
   return 0;
}
```

http://www.alexonlinux.com/signal-handling-in-linux

Отправка сигналов

- Команда **kill**:

```
kill -s USR1 PID
```

- Функция **kill**:

```
int ret;
ret = kill(pid,SIGHUP);
printf("ret : %d",ret);
```

- С клавиатуры(CTRL-C,..):

```
stty -a | grep intr
intr = ^C; quit = ^\; erase = ^?; kill = ^U; eof = ^D;
eol = M-^?; eol2 = M-^?;;
```

Дополнительно

Выполнение команды командной строки

- **system**:

```
sprintf(cmd, "/bin/ls -l");
system(cmd);
```

- **popen**:

```
FILE* file = popen("ls", "r");
char buffer[100];
fscanf(file, "%100s", buffer);
pclose(file);
```

- **exec,..**: Заменяет текущий процесс. Не требуется для задания.

Дополнительно изучить

- syslog:

```
setlogmask (LOG_UPTO (LOG_NOTICE));
openlog (“exampleprog”, LOG_CONS | LOG_PID |
LOG_NDELAY, LOG_LOCAL1);
syslog (LOG_NOTICE, “Program started by User %d”, getuid ());
syslog (LOG_INFO, “A tree falls in a forest”);
closelog ();
```

- Команда ps: `ps -xj | grep myfile`

---

http://www.gnu.org/software/libc/manual/html_node/Syslog-Example.html

Добавление модуля ядра

- Узнать версию исходников:

```
uname -a dpkg -s linux-headers-$(uname -r)
```

- Проверить их наличие(установить в случае отсутствия)
- Добавить нужный модуль, например, создать в подпапке *drivers/misc* свою папку. Полный путь в референсной системе - */usr/src/linux-headers-3.13.0-24-generic/drivers/misc/mymodule*

```
make -C /usr/src/linux-headers-3.13.0-24-generic/
SUBDIRS=$PWD modules
```

Полезные ссылки

- М. К. Джонсон, Э.В. Троан. *Разработка приложений в среде Linux*. Вильямс. 2007
- Р. Лав. *Linux. Системное программирование*. Питер. 2008.
- Соловьев А. *Разработка модулей ядра ОС Linux. Kernel newbie's manual*
- *Advanced Linux Programming*
- `http://www.tldp.org/LDP/lkmpg/2.6/html/index.html`
- `http://people.ee.ethz.ch/~arkeller/linux/kernel_user_space_howto.html`
- `http://www.makelinux.net/ldd3/`
- `http://www.xml.com/ldd/chapter/book/ch09.html`
- `https://www.centos.org/docs/5/html/Deployment_Guide-en-US/ch-proc.html`
- `http://www.tldp.org/LDP/Linux-Filesystem-Hierarchy/html/proc.html`