Архитектура ОС UNIX

Санкт-Петербургский государственный политехнический университет

13 октября 2011

Что есть демон

Демон(служба) — фоновый процесс, который разработан специально для автономной работы.

Необходимые компоненты для разработки

- Компилятор (gcc).
- Библиотеки и заголовочные файлы для разработки в Linux.

Unix-way при написании демонов

Демон должен делать только одну вещь и делать ее хорошо.

Интерактивность

Демоны не должны общаться с пользователем напрямую через терминал.

Базовая структура демона

- Отделение (ответвление, fork) от родительского процесса
- Изменение файловой маски (umask)
- Открытие любых журналов на запись
- Создание уникального ID сессии (SID)
- Изменение текущего рабочего каталога на безопасное место
- Закрытие стандартных файловых дескрипторов
- Цикл выполнения кода демона

Отделение от родительского процесса

Демон запускается либо самой системой, либо пользователем в терминале или скрипте. Во время запуска его процесс ничем не отличается от любого другого процесса в системе. Чтобы сделать его по-настоящему автономным, нужно создать дочерний процесс, в котором будет выполняться код демона.

```
#include <stdlib.h>
#include <unistd.h>
pid_t pid;
/* отделяемся от родительского процесса */
pid = fork();
if (pid < 0) {
  exit(EXIT_FAILURE);
}
/* Если с PID'ом все получилось, то родительский
   процесс можно завершить. */
if (pid > 0) {
  exit(EXIT_SUCCESS);
}
```

Изменение файловой маски (Umask)

Чтобы иметь возможномть писать в любые файлы (включая журналы), созданные демоном, файловая маска (umask) должна быть изменена так, чтобы они могли быть записаны или прочитаны правильным образом.

```
pid_t pid, sid;
/* Ответвляемся от родительского процесса */
pid = fork();
if (pid < 0) {
  /* Фиксируем ошибку (через syslog при возможности
     ) */
  exit(EXIT_FAILURE);
}
/* Если с PID'ом все получилось, то родительский
   процесс можно завершить. */
if (pid > 0) {
  exit(EXIT_SUCCESS);
}
/* Изменяем файловую маску */
umask(0);
```

Открытие журналов на запись

Необходимо для того чтобы иметь возможность посмотреть на отладочную информацию от демона.

Создание уникального ID сессии (SID)

Для нормальной работы дочерний процесс должен получить уникальный SID от ядра. Иначе дочерний процесс станет сиротой.

```
/* Здесь можно открывать любые журналы */

/* Создание нового SID для дочернего процесса */
sid = setsid();
if (sid < 0) {
   /* Журналируем любой сбой */
   exit(EXIT_FAILURE);
}
```

Открытие журналов на запись

Необходимо для того чтобы иметь возможность посмотреть на отладочную информацию от демона.

Изменение рабочего каталога

Текущий рабочий каталог нужно сменить на некоторое место, гарантированно присутствующее в системе.

```
/* Изменяем текущий рабочий каталог */
if ((chdir("/")) < 0) {
   /* Журналируем любой сбой */
   exit(EXIT_FAILURE);
}
```

Закрытие стандартных файловых дескрипторов

Поскольку демон не может использовать терминал, эти файловые дескрипторы излишни и создают угрозу безопасности.

```
/* Закрываем стандартные файловые дескрипторы */
close(STDIN_FILENO);
close(STDOUT_FILENO);
close(STDERR_FILENO);
```

Инициализация демона

Задание стартовых значений переменных, включение/выключение опций в зависимости от переданных параметров командной строки.

Большой Цикл

Основной код демона обычно находится внутри бесконечного цикла.

```
/* Большой Цикл */
while (1) {
  /* Делаем тут чего-нибудь ... */
  sleep(30); /* ждем 30 секунд */
}
```

Журналирование событий через syslog

```
#include <syslog.h>
void openlog(const char *ident, int option, int
    facility);
void syslog(int priority, const char *format, ...);
void closelog(void);
```

openlog() — устанавливает связь с программой, ведущей системный журнал. Строка ident добавляется к каждому сообщению и обычно представляет собой название программы.
syslog() — создает сообщение для журнала, которое передается syslogd.

Обработка сигналов

Обработка сигналов переданных процессу осуществляется в специально написанном обработчике.

```
void signal_handler(int sig)
{
switch(sig) {
case SIGHUP:
syslog(LOG_FILE,"hangup signal catched");
break;
case SIGTERM:
syslog(LOG_FILE,"terminate signal catched");
exit(0);
break;
}
}
```

Обработка сигналов

После чего вызов обработчика настраивается на определенные типы сигналов

```
#include <signal.h>

signal(SIGHUP, signal_handler); /* catch hangup
    signal */
signal(SIGTERM, signal_handler); /* catch kill
    signal */
```

Работа с файловой системой

```c
#include <dirent.h>

DIR *opendir(const char *name);
int closedir(DIR *dir);
int readdir(unsigned int fd, struct dirent *dirp,
    unsigned int count);
```

opendir() — открывает директорию для чтения с именем name и возвращает указатель на directory stream.
closedir() — закрывает directory stream.
readdir() — возвращает следующую структуру dirent считанную из файла-директории.

Структура dirent

```
struct dirent
{
long d_ino;                    /* порядковый номер
    файла в файловой системе */
off_t d_off;                   /* смещение данного
    элемента в реальном каталоге */
unsigned short d_reclen;       /* длина имени файла */
char d_name [NAME_MAX+1];      /* имя файла */
}
```

Работа с файловой системой

```
#include <stdio.h>
#include <dirent.h>
int main() {
    DIR *dir;
    struct dirent *entry;
    dir = opendir("/");
    while ( (entry = readdir(dir)) != NULL) {
        printf("%d - %s [%d] %d\n",
            entry->d_ino, entry->d_name, entry->
                d_type, entry->d_reclen);
    }
    closedir(dir);
}
```

Работа с файловой системой

```
#include <stdio.h>
#include <dirent.h>
int main() {
    DIR *dir;
    struct dirent *entry;
    dir = opendir("/");
    while ( (entry = readdir(dir)) != NULL) {
        printf("%d - %s [%d] %d\n",
            entry->d_ino, entry->d_name, entry->
                d_type, entry->d_reclen);
    }
    closedir(dir);
}
```

stat

```
int stat(const char *file_name, struct stat *buf);
```

stat() — получение статусной информации файла.

Структура содержит следующие поля:

```
ushort st_mode;   /* Режим файла */
ino_t  st_ino;    /* Номер описателя файла */
dev_t  st_dev;    /* Идент. устройства, содержащего
    каталог с входом в этот файл */
dev_t  st_rdev;   /* Идент. устройства. Поле
    определено только для специальных символьных и
    блочных файлов */
short  st_nlink; /* Количество ссылок */
ushort st_uid;    /* Идент. владельца файла */
ushort st_gid;    /* Идент. группы владельца файла
    */
off_t  st_size;   /* Размер файла в байтах */
```

Структура содержит следующие поля:

```
time_t st_atime; /* Время последнего доступа к
    файлу */
time_t st_mtime; /* Время последней записи в файл
    */
time_t st_ctime; /* Время  последнего изменения
    статуса файла */
/* Время измеряется в секундах от 00:00:00 1 января
    1970г. по Гринвичу */
```

Задание

Нужно написать демон под линукс который бы читал конфигурационный файл, брал оттуда параметры: каталог, время проверки. После чего проверял заданий каталог(рекурсивно), через заданные промежутки времени, на предмет наличия/отсутствия модификации файлов и записывал результаты в файл журнала, предусмотреть обработку сигналов: SIGHUP - для перечитывания конфигурационного файла, и SIGTERM - для контролируемого завершения демона (запись о выходе в файл журнала).