# Файловая система /proc

## Лабораторная работа /proc (1 часть)

Файловая система /proc - виртуальная файловая система, предоставляющая интерфейс для доступа к структурам ядра. Папки (каталоги) и файлы виртуальной файловой системы /proc не хранятся на диске. Они создаются динамически при обращении к ним. Большинство файлов в ней доступны только для чтения, однако некоторые из них доступны для записи, что позволяет изменять переменные ядра.

Для каждого активного процесса в /proc существует своя поддиректория /proc/[pid] (для доступа к поддиректории текущего процесса можно использовать ссылку /proc/self). Каждая поддиректория /proc/[pid] содержит файлы и директории, владельцем которых является эффективный пользователь и эффективная группа процесса.

| Элемент | Тип | Содержание |
|---|---|---|
| cmdline | файл | Указывает на директорию процесса |
| cwd | символическая ссылка | Указывает на директорию процесса |
| environ | файл | Список окружения процесса |
| exe | символическая ссылка | Указывает на образ процесса (на его файл) |
| fd | директория | Ссылки на файлы, которые «открыл» процесс |
| root | символическая ссылка | Указывает на корень файловой системы процесса |
| stat | файл | Информация о процессе |

Таблица 1. Файлы и поддиректории /proc/<PID>.

Некоторые из этих файлов и директорий (актуально для версии ядра 4.15.0-88-generic):

### /proc/[pid]/cmdline

Данный файл содержит полную командную строку процесса, если процесс не находится в состоянии зомби, иначе файл пуст. Аргументы командной строки представлены в виде набора строк, разделенных символом конца строки ('\0').

### /proc/[pid]/environ

Листинг 1. Код вывода на экран окружения процесса.

```
FILE *f = fopen("/proc/self/environ", "r");
while ((len = fread(buf, 1, BUF_SIZE, f)) > 0)
{
      for (int i = 0; i < len; i++)
            if (buf[i] == 0)
                  buf[i] = 10;
      buf[len - 1] = 0;
      printf("%s", buf);
}
fclose(f);
```

Данный файл содержит исходное окружение, которое было установлено при запуске текущего процесса (вызове execve()). Переменные окружения разделены символами конца строки ('\0'). Если после вызова execve() окружение процесса будет модифицировано (например, вызовом функции putenv() или модификацией переменной окружения напрямую), этот файл не отразит внесенных изменений.

Некоторые переменные окружения:

- LS_COLORS - используется для определения цветов, с которыми будут выведены имена файлов при вызове ls.
- LESSCLOSE, LESSOPEN – определяют пре- и пост- обработчики файла, который открывается при вызове less.
- XDG_MENU_PREFIX, XDG_VTNR, XDG_SESSION_ID, XDG_SESSION_TYPE, XDG_DATA_DIRS, XDG_SESSION_DESKTOP, XDG_CURRENT_DESKTOP, XDG_RUNTIME_DIR, XDG_CONFIG_DIRS, DESKTOP_SESSION – переменные, необходимые для вызова xdg-open, использующейся для открытия файла или URL в пользовательском приложении.
- LANG – язык и кодировка пользователя.
- DISPLAY – указывает приложениям, куда отобразить графический пользовательский интерфейс.
- GNOME_SHELL_SESSION_MODE, GNOME_TERMINAL_SCREEN, GNOME_DESKTOP_SESSION_ID, GNOME_TERMINAL_SERVICE, GJS_DEBUG_OUTPUT, GJS_DEBUG_TOPICS, GTK_MODULES, GTK_IM_MODULE, VTE_VERSION – переменные среды рабочего стола GNOME.
- COLORTERM – определяет поддержку 24-битного цвета.
- USER – имя пользователя, от чьего имени запущен процесс,
- USERNAME – имя пользователя, кто инициировал запуск процесса.
- SSH_AUTH_SOCK - путь к сокету, который агент использует для коммуникации с другими процессами.
- TEXTDOMAINDIR, TEXTDOMAIN – директория и имя объекта сообщения, получаемого при вызове gettext.
- PWD – путь к рабочей директории.
- HOME – путь к домашнему каталогу текущего пользователя.
- SSH_AGENT_PID - идентификатор процесса ssh-agent.
- TERM – тип запущенного терминала.
- SHELL – путь к предпочтительной оболочке командной строки.
- SHLVL – уровень текущей командной оболочки.
- LOGNAME – имя текущего пользователя.
- PATH - список каталогов, в которых система ищет исполняемые файлы.
- _ - полная командная строка процесса
- OLDPWD - путь к предыдущему рабочему каталогу.

Окружение (environment) или среда — это набор пар ПЕРЕМЕННАЯ=ЗНАЧЕНИЕ, доступный каждому пользовательскому процессу. Иными словами, окружение — это набор переменных окружения.

## /proc/[pid]/stat

Листинг 3. Код вывода на экран содержимого файла stat.

```
FILE *f = fopen("/proc/self/stat", "r");
fread(buf, 1, BUF_SIZE, f);
char *pch = strtok(buf, " ");

printf("stat: \n");
while(pch != NULL)
{
      printf("%s\n", pch);
      pch = strtok(NULL, " ");
}
fclose(f);
```

Содержимое файла /proc/[pid]/stat:

1) pid - уникальный идентификатор процесса.
2) comm - имя исполняемого файла в круглых скобках.
3) state - состояние процесса.
4) ppid - уникальный идентификатор процесса-предка.
5) pgrp - уникальный идентификатор группы.
6) session - уникальный идентификатор сессии.
7) tty_nr – управляющий терминал.
8) tpgid – уникальный идентификатор группы управляющего терминала.
9) flags – флаги.
10) minflt - Количество незначительных сбоев, которые возникли при выполнении процесса, и которые не требуют загрузки страницы памяти с диска.
11) cminflt - количество незначительных сбоев, которые возникли при ожидании окончания работы процессов-потомков.
12) majflt - количество значительных сбоев, которые возникли при работе процесса, и которые потребовали загрузки страницы памяти с диска.
13) cmajflt - количество значительных сбоев, которые возникли при ожидании окончания работы процессов-потомков.
14) utime - количество тиков, которые данный процесс провел в режиме пользователя.
15) stime - количество тиков, которые данный процесс провел в режиме ядра.
16) cutime - количество тиков, которые процесс, ожидающий завершения процессов-потомков, провёл в режиме пользователя.
17) cstime - количество тиков, которые процесс, ожидающий завершения процессов-потомков, провёл в режиме ядра.
18) priority – для процессов реального времени это отрицательный приоритет планирования минус один, то есть число в диапазоне от -2 до -100, соответствующее приоритетам в реальном времени от 1 до 99. Для остальных процессов это необработанное значение nice, представленное в ядре. Ядро хранит значения nice в виде чисел в диапазоне от 0 (высокий) до 39 (низкий), соответствующих видимому пользователю диапазону от -20 до 19.

19) nice - значение для nice в диапазоне от 19 (наиболее низкий приоритет) до -20 (наивысший приоритет).
20) num_threads – число потоков в данном процессе.
21) itrealvalue – количество мигов до того, как следующий SIGALARM будет послан процессу интервальным таймером. С ядра версии 2.6.17 больше не поддерживается и установлено в 0.
22) starttime - время в тиках запуска процесса после начальной загрузки системы.
23) vsize - размер виртуальной памяти в байтах.
24) rss - резидентный размер: количество страниц, которые занимает процесс в памяти. Это те страницы, которые заняты кодом, данными и пространством стека. Сюда не включаются страницы, которые не были загружены по требованию или которые находятся в своппинге.
25) rsslim - текущий лимит в байтах на резидентный размер процесса.
26) startcode - адрес, выше которого может выполняться код программы.
27) endcode - адрес, ниже которого может выполняться код программ.
28) startstack - адрес начала стека.
29) kstkesp - текущее значение ESP (указателя стека).
30) kstkeip - текущее значение EIP (указатель команд).
31) signal - битовая карта ожидающих сигналов. Устарела, потому что не предоставляет информацию о сигналах реального времени, необходимо использовать /proc/[pid]/status.
32) blocked - битовая карта блокируемых сигналов. Устарела, потому что не предоставляет информацию о сигналах реального времени, необходимо использовать /proc/[pid]/status.
33) sigignore - битовая карта игнорируемых сигналов. Устарела, потому что не предоставляет информацию о сигналах реального времени, необходимо использовать /proc/[pid]/status.
34) sigcatch - битовая карта перехватываемых сигналов. Устарела, потому что не предоставляет информацию о сигналах реального времени, необходимо использовать /proc/[pid]/status.
35) wchan - "канал", в котором ожидает процесс.
36) nswap - количество страниц на своппинге (не обслуживается).
37) cnswap - суммарное nswap для процессов-потомков (не обслуживается).
38) exit_signal - сигнал, который будет послан предку, когда процесс завершится.
39) processor - номер процессора, на котором последний раз выполнялся процесс.
40) rt_priority - приоритет планирования реального времени, число в диапазоне от 1 до 99 для процессов реального времени, 0 для остальных.
41) policy - политика планирования.
42) delayacct_blkio_ticks - суммарные задержки ввода/вывода в тиках.
43) guest_time – гостевое время процесса (время, потраченное на выполнение виртуального процессора на гостевой операционной системе) в тиках.
44) cguest_time - гостевое время для потомков процесса в тиках.
45) start_data - адрес, выше которого размещаются инициализированные и неинициализированные (BSS) данные программы.
46) end_data - адрес, ниже которого размещаются инициализированные и неинициализированные (BSS) данные программы.
47) start_brk - адрес, выше которого куча программы может быть расширена с использованием brk().
48) arg_start - адрес, выше которого размещаются аргументы командной строки (argv).

49) arg_end - адрес, ниже которого размещаются аргументы командной строки (argv).
50) env_start - адрес, выше которого размещается окружение программы.
51) env_end - адрес, ниже которого размещается окружение программы.
52) exit_code – статус завершения потока в форме, возвращаемой waitpid().

## /proc/[pid]/fd/

Данная поддиректория содержит одну запись для каждого файла, который открыт процессом. Имя каждой такой записи соответствует номеру файлового дескриптора и является символьной ссылкой на реальный файл. Так, 0 - это стандартный ввод, 1 - стандартный вывод, 2 - стандартный вывод сообщений об ошибках и т. д.

Для файловых дескрипторов сокетов и программных каналов записи будут являться символьными ссылками, содержащими тип файла с inode. Вызов readlink() для них вернет строку следующего формата: type:[inode].

Для файловых дескрипторов, не имеющих соответствующего inode, символьная ссылка будет иметь следующий вид: anon_inode:<file-type>.

Для мультипоточных процессов содержимое данной поддиректории может быть недоступно, если главный поток завершил свое выполнение.

Программы, которые принимают имя файла в качестве аргумента командной строки, но не используют стандартный ввод, и программы, которые пишут в файл, но не используют стандартный вывод, могут использовать стандартный ввод и вывод в качестве аргументов командной строки с помощью файлов из /proc/[pid]/fd. Например:
$ foobar -i /proc/self/fd/0 -o /proc/self/fd/1 ...

```
void read_fd()
{
    printf("\nfd:\n");
    struct dirent *dirp;
    DIR *dp;
    char str[BUFF_SIZE];
    char path[BUFF_SIZE];
    dp = opendir("/proc/self/fd");//открыть директорию
    while ((dirp = readdir(dp)) != NULL)//читаем директорию
    {
        if ((strcmp(dirp->d_name, ".") != 0) &&
            (strcmp(dirp->d_name, "..") != 0))
        {
            sprintf(path, "%s%s", "/proc/self/fd/", dirp->d_name);
            readlink(path, str, BUFF_SIZE);
            printf("%s -> %s\n", dirp->d_name, str);
        }
    }
    closedir(dp);
}
```

Можно использовать команду ls, которую надо передать системному вызову exec().

```
    execl("/bin/ls", "ls", "/proc/self/fd", NULL);
```

**Задание: написать программу, которая в пользовательском режиме выводит на экран:**

- информацию об окружении процесса (environ) с комментариями;
- информацию о состоянии (state) процесса с комментариями;
- вывести информацию из файла cmdline и директории fd на экран.