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]/exe

В Linux 2.2 и более поздних версиях этот файл представляет собой символическую ссылку, содержащую фактический путь к выполняемой команде. Эта символическая ссылка может быть разыменована обычным образом;

попытка открыть его откроет исполняемый файл. Можно даже ввести/proc/[pid]/exe, чтобы запустить другую копию того же исполняемого файла, который запускается процессом [pid]. Если путь не связан, символическая ссылка будет содержать строку '(удалено)', добавленную к исходному имени пути. В многопоточном процессе содержимое этой символической ссылки недоступно, если основной поток уже завершился (обычно вызовом pthread_exit(3)).

Разрешение на разыменование или чтение (readlink(2)) этой символической ссылки регулируется режимом доступа ptrace проверка PTRACE_MODE_READ_FSCREDS; см. ptrace

В Linux 2.0 и более ранних версиях /proc/[pid]/exe являлся указателем. к двоичному файлу, который был выполнен, и отображается как символическая ссылка. Вызов readlink(2) для этого файла в Linux 2.0 возвращает строку в формате:

[устройство]:inode

Например, [0301]:1502 будет inode 1502 на устройстве. старший 03 (диски IDE, MFM и т. д.) младший 01 (первый раздел на первом диске).
find(1) с опцией -inum можно использовать для поиска файла.

/proc/[pid]/cmdline

Этот файл, доступный только для чтения, содержит полную командную строку для процесс, если только этот процесс не является зомби. В последнем случае в этом файле ничего нет: то есть чтение этого файла вернет 0 символов. Команда-аргументы строки отображаются в этом файле в виде набора строк, разделенные нулевыми байтами ('\0') с дополнительным нулевым байтом после последней строки.

/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);
```

Замечание: в приведенном виде программа выведет информацию о самой себе (self).

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);
```

Данный файл содержит информацию о статусе процесса, которая используется при вызове ps. Его поля хранятся в виде одной строки и разделены пробелами.

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);
```

/proc/[pid]/io (since kernel 2.6.20)

This file contains I/O statistics for the process, for example:

```
# cat /proc/3828/io
rchar: 323934931
wchar: 323929600
syscr: 632687
syscw: 632675
read_bytes: 0
write_bytes: 323932160
cancelled_write_bytes: 0
```

Поля следующие:

rchar: прочитанные символы

Количество байтов, которые эта задача вызвала для чтения из хранилища. Это просто сумма байтов, которые этот процесс передал для чтения read(2) и аналогичных системных вызовов. Он включает в себя такие вещи, как терминальный ввод-вывод, и не

зависит от того, является он или нет фактическим вводом-выводом с физического диска (возможно считывание, было выполнено из кэша страниц (pagecache)).

wchar: записанные символы

Количество байтов, которые эта задача вызвала или должна вызвать для записи на диск. Аналогичные предостережения применяйте здесь, как и в случае с *rchar*.

syscr: системные вызовы (syscalls) read

Попытка подсчитать количество операций ввода-вывода чтения—то есть системные вызовы, такие как read(2) и pread(2).

syscw: системные вызовы записи (write syscalls)

Попытка подсчитать количество операций ввода-вывода при записи операции — то есть системные вызовы, такие как write(2) и pwrite(2).

read_bytes: прочитанные байты (bytes read)

Подсчет количества байтов, которые этот процесс действительно было извлечено из хранилища. Это верно для блоков с поддержкой файловых систем (чтение из блок ориентированных потоков.

write_bytes: записанные байты (bytes written)

Количество байтов, которые процесс отправил на устройство хранения.

cancelled_write_bytes:

The big inaccuracy here is truncate. If a process writes 1 MB to a file and then deletes the file, it will in fact perform no writeout. But it will have been accounted as having caused 1 MB of write. In other words: this field represents the number of bytes which this process caused to not happen, by truncating pagecache. A task can cause "negative" I/O too. If this task truncates some dirty pagecache, some I/O which another task has been accounted for (in its *write_bytes*) will not be happening.

Note: In the current implementation, things are a bit racy on 32-bit systems: if process A reads process B's */proc/[pid]/io* while process B is updating one of these 64-bit counters, process A could see an intermediate result.

Permission to access this file is governed by a ptrace access mode **PTRACE_MODE_READ_FSCREDS** check; see ptrace(2).

/proc/[pid]/maps

Файл, содержащий отображенные в данный момент области памяти и права доступа к ним. Смотрите mmap (2) для получения дополнительной информации о соответствующих регионах виртуальной памяти процесса. Размер регионов кратен размеру страницы.

Permission to access this file is governed by a ptrace access mode **PTRACE_MODE_READ_FSCREDS** check; see ptrace(2).

Формат файла:

```
address                  perms   offset   dev   inode      pathname
00400000-00452000 r-xp 00000000 08:02 173521     /usr/bin/dbus-daemon
00651000-00652000 r--p 00051000 08:02 173521     /usr/bin/dbus-daemon
00652000-00655000 rw-p 00052000 08:02 173521     /usr/bin/dbus-daemon
00e03000-00e24000 rw-p 00000000 00:00 0          [heap]
00e24000-011f7000 rw-p 00000000 00:00 0          [heap]
...
35b1800000-35b1820000 r-xp 00000000 08:02 135522  /usr/lib64/ld-2.15.so
35b1a1f000-35b1a20000 r--p 0001f000 08:02 135522  /usr/lib64/ld-2.15.so
35b1a20000-35b1a21000 rw-p 00020000 08:02 135522  /usr/lib64/ld-2.15.so
35b1a21000-35b1a22000 rw-p 00000000 00:00 0
35b1c00000-35b1dac000 r-xp 00000000 08:02 135870  /usr/lib64/libc-2.15.so
35b1dac000-35b1fac000 ---p 001ac000 08:02 135870  /usr/lib64/libc-2.15.so
35b1fac000-35b1fb0000 r--p 001ac000 08:02 135870  /usr/lib64/libc-2.15.so
35b1fb0000-35b1fb2000 rw-p 001b0000 08:02 135870  /usr/lib64/libc-2.15.so
...
f2c6ff8c000-7f2c7078c000 rw-p 00000000 00:00 0    [stack:986]
...
7fffb2c0d000-7fffb2c2e000 rw-p 00000000 00:00 0   [stack]
7fffb2d48000-7fffb2d49000 r-xp 00000000 00:00 0   [vdso]
```

Поле *address* – регион адресного пространства (address space) процесса сопоставленное с содержимым.

Права доступа (The *perms* field is a set of permissions):

r = read
w = write
x = execute
s = shared
p = private (copy on write)

The *offset* field is the offset into the file/whatever; *dev* is the device (major:minor); *inode* is the inode on that device. 0 indicates that no inode is associated with the memory region, as would be the case with BSS (uninitialized data).

The *pathname* field will usually be the file that is backing the mapping. For ELF files, you can easily coordinate with the *offset* field by looking at the Offset field in the ELF program headers (*readelf -l*).

Существуют дополнительные полезные псевдопути (There are additional helpful pseudo-paths) :

[stack]
The initial process's (also known as the main thread's) stack.

[stack:<tid>] (from Linux 3.4 to 4.4)
A thread's stack (where the *<tid>* is a thread ID). It corresponds to the ***/proc/[pid]/task/[tid]/*** path. This field was **removed** in Linux 4.5, since providing this information for a process with large numbers of threads is expensive.

[vdso] The virtual dynamically linked shared object. See vdso(7).

[heap] The process's heap.

If the *pathname* field is blank, this is an anonymous mapping as obtained via mmap(2).

```
#include <sys/mman.h>

       void *mmap(void addr[.length], size_t length, int prot,
int flags,
                  int fd, off_t offset);
       int munmap(void addr[.length], size_t length);
```

Функция mmap() создает новое сопоставление в виртуальном адресном пространстве вызывающего процесса. Начальный адрес для нового сопоставления указан в addr. Аргумент length указывает длину сопоставления (которая должна быть больше 0).

Если addr равно NULL, то ядро выбирает (выровненный по странице) адрес, по которому будет создано сопоставление; это наиболее переносимый метод создания нового сопоставления. Если addr не равно NULL, то ядро воспринимает это как подсказку о том, где разместить отображение; на Linux, ядро выберет ближайшую границу страницы (но всегда выше или равно значению, указанному в /proc/sys/vm/mmap_min_addr) и попытается создать сопоставление там. Если там уже существует другое сопоставление, ядро выбирает новый адрес, который может зависеть, а может и не зависеть от подсказки. Адрес нового сопоставления возвращается в результате вызова.

Поле pathname показывает имя файла и вспомогательных псевдопутей.

// *pathname* is shown unescaped except for newline characters, which are replaced with an octal escape sequence. As a result, it is not possible to determine whether the original pathname contained a newline character or the literal *\012* character sequence.

If the mapping is file-backed and the file has been deleted, the string " (deleted)" is appended to the pathname. Note that this is ambiguous too.
Under Linux 2.0, there is no field giving pathname.

/proc/[pid]/comm (since Linux 2.6.33)
Этот файл предоставляет значение **comm** процесса, т. е. имя команды, связанное с процессом. Разные потоки в одном и том же процессе могут иметь разные **comm** значения, доступные через **/proc/[pid]/task/[tid]/comm**. А поток может изменить свое значение **comm** или значение любого другого потока в той же группе потоков (см. обсуждение CLONE_THREAD в clone(2)), записав в файл **/proc/self/task/[tid]/comm.** Строки длиннее, чем TASK_COMM_LEN (16) символов (включая завершающий нулевой байт) автоматически усекаются.

Этот файл предоставляет надмножество prctl(2) PR_SET_NAME и PR_GET_NAME, и используется pthread_setname_np(3) при использовании для переименования потоков других чем вызывающий. Значение в этом файле используется для %e спецификатора в /proc/sys/kernel/core_pattern; смотрите ядро core (5).

/proc/[pid]/pagemap **(since Linux 2.6.25)**

This file shows the mapping of each of the process's virtual pages into physical page frames or swap area. It contains one 64-bit value for each virtual page, with the bits set as follows:

63 If set, the page is present in RAM.

62 If set, the page is in swap space

61 (since Linux 3.5)
The page is a file-mapped page or a shared anonymous page.

60–57 (since Linux 3.11)
Zero

56 (since Linux 4.2)
The page is exclusively mapped.

55 (since Linux 3.11)
PTE is soft-dirty (see the kernel source file *Documentation/admin-guide/mm/soft-dirty.rst*).

54–0 If the page is present in RAM (bit 63), then these bits provide the page frame number, which can be used to index */proc/kpageflags* and */proc/kpagecount*. If the page is present in swap (bit 62), then bits 4–0 give the swap type, and bits 54–5 encode the swap offset.

Before Linux 3.11, bits 60–55 were used to encode the base-2 log of the page size.

To employ */proc/[pid]/pagemap* efficiently, use */proc/[pid]/maps* to determine which areas of memory are actually mapped and seek to skip over unmapped regions.

The */proc/[pid]/pagemap* file is present only if the **CONFIG_PROC_PAGE_MONITOR** kernel configuration option is enabled.

Permission to access this file is governed by a ptrace access mode **PTRACE_MODE_READ_FSCREDS** check; see ptrace(2).

/proc/[pid]/root

UNIX and Linux support the idea of a per-process root of the filesystem, set by the chroot(2) system call. This file is a symbolic link that points to the process's root directory, and behaves in the same way as *exe*, and *fd/**.

Note however that this file is not merely a symbolic link. It provides the same view of the filesystem (including namespaces and the set of per-process mounts) as the process itself. An example illustrates this point. In one terminal, we start a shell in new user and mount namespaces, and in that shell we create some new mounts:

```
$ PS1='sh1# ' unshare -Urnm
sh1# mount -t tmpfs tmpfs /etc  # Mount empty tmpfs at /etc
sh1# mount --bind /usr /dev     # Mount /usr at /dev
sh1# echo $$
27123
```

In a second terminal window, in the initial mount namespace, we look at the contents of the corresponding mounts in the initial and new namespaces:

```
$ PS1='sh2# ' sudo sh
sh2# ls /etc | wc -l               # In initial NS
309
sh2# ls /proc/27123/root/etc | wc -l  # /etc in other NS
0                                  # The empty tmpfs dir
sh2# ls /dev | wc -l               # In initial NS
205
sh2# ls /proc/27123/root/dev | wc -l  # /dev in other NS
11                                 # Actually bind
                                   # mounted to /usr
sh2# ls /usr | wc -l               # /usr in initial NS
11
```

In a multithreaded process, the contents of the */proc/[pid]/root* symbolic link are not available if the main thread has already terminated (typically by calling pthread_exit(3)).

Permission to dereference or read (readlink(2)) this symbolic link is governed by a ptrace access mode **PTRACE_MODE_READ_FSCREDS** check; see ptrace(2).

/proc/[pid]/smaps **(since Linux 2.6.14)**

This file shows memory consumption for each of the process's mappings. (The pmap(1) command displays similar information, in a form that may be easier for parsing.) For each mapping there is a series of lines such as the following:

```
00400000-0048a000 r-xp 00000000 fd:03 960637       /bin/bash
Size:                552 kB
Rss:                 460 kB
Pss:                 100 kB
Shared_Clean:        452 kB
Shared_Dirty:          0 kB
Private_Clean:         8 kB
Private_Dirty:         0 kB
Referenced:          460 kB
Anonymous:             0 kB
AnonHugePages:         0 kB
ShmemHugePages:        0 kB
ShmemPmdMapped:        0 kB
Swap:                  0 kB
KernelPageSize:        4 kB
MMUPageSize:           4 kB
KernelPageSize:        4 kB
MMUPageSize:           4 kB
Locked:                0 kB
ProtectionKey:         0
VmFlags: rd ex mr mw me dw
```

The first of these lines shows the same information as is displayed for the mapping in */proc/[pid]/maps*. The following lines show the size of the mapping, the amount of the mapping that is currently resident in RAM ("Rss"), the process's proportional share of this mapping ("Pss"), the number of clean and dirty shared pages in the mapping, and the number of clean and dirty private pages in the mapping. "Referenced" indicates the amount of memory currently marked as referenced or accessed. "Anonymous" shows the amount of memory that does not belong to any file. "Swap" shows how much would-be-anonymous memory is also used, but out on swap.

The "KernelPageSize" line (available since Linux 2.6.29) is the page size used by the kernel to back the virtual memory area. This matches the size used by the MMU in the majority of cases. However, one counter-example occurs on PPC64 kernels whereby a kernel using 64 kB as a base page size may still use 4 kB pages for the MMU on older processors. To distinguish the two attributes, the "MMUPageSize" line (also available since Linux 2.6.29) reports the page size used by the MMU.

The "Locked" indicates whether the mapping is locked in memory or not.

The "ProtectionKey" line (available since Linux 4.9, on x86 only) contains the memory protection key (see pkeys(7)) associated with the virtual memory area. This entry is present only if the kernel was built with the

CONFIG_X86_INTEL_MEMORY_PROTECTION_KEYS configuration option (since Linux 4.6).

The "VmFlags" line (available since Linux 3.8) represents the kernel flags associated with the virtual memory area, encoded using the following two-letter codes:

rd - readable
wr - writable
ex - executable
sh - shared
mr - may read
mw - may write
me - may execute
ms - may share
gd - stack segment grows down
pf - pure PFN range
dw - disabled write to the mapped file
lo - pages are locked in memory
io - memory mapped I/O area
sr - sequential read advise provided
rr - random read advise provided
dc - do not copy area on fork
de - do not expand area on remapping
ac - area is accountable
nr - swap space is not reserved for the area
ht - area uses huge tlb pages
sf - perform synchronous page faults (since Linux 4.15)
nl - non-linear mapping (removed in Linux 4.0)
ar - architecture specific flag
wf - wipe on fork (since Linux 4.14)
dd - do not include area into core dump
sd - soft-dirty flag (since Linux 3.13)
mm - mixed map area
hg - huge page advise flag
nh - no-huge page advise flag
mg - mergeable advise flag
um - userfaultfd missing pages tracking (since Linux 4.3)
uw - userfaultfd wprotect pages tracking (since Linux 4.3)

The */proc/[pid]/smaps* file is present only if the **CONFIG_PROC_PAGE_MONITOR** kernel configuration option is enabled.

/proc/[pid]/statm

Provides information about memory usage, measured in pages. Предоставляет информацию об использовании памяти, измеренную в страницах.

The columns are:

size (1) total program size
(same as VmSize in */proc/[pid]/status*)
resident (2) resident set size
(inaccurate; same as VmRSS in */proc/[pid]/status*)
shared (3) number of resident shared pages
(i.e., backed by a file)
(inaccurate; same as RssFile+RssShmem in
/proc/[pid]/status)
text (4) text (code)
lib (5) library (unused since Linux 2.6; always 0)
data (6) data + stack
dt (7) dirty pages (unused since Linux 2.6; always 0)

Some of these values are inaccurate because of a kernel-internal scalability optimization. If accurate values are required, use */proc/[pid]/smaps* or */proc/[pid]/smaps_rollup* instead, which are much slower but provide accurate, detailed information.

/proc/[pid]/wchan **(since Linux 2.6.0)**
The symbolic name corresponding to the location in the kernel where the process is sleeping.

Permission to access this file is governed by a ptrace access mode **PTRACE_MODE_READ_FSCREDS** check; see ptrace(2).

Символическое имя, соответствующее местоположению в ядре, в котором процесс спит.
Разрешение на доступ к этому файлу регулируется ptrace проверка режима доступа PTRACE_MODE_READ_FSCREDS; см. ptrace(2).

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;
- вывести содержание символической ссылки cwd;
- вывести содержание символической ссылки exe;
- вывести содержание символической ссылки root;
- вывести содержимое файла maps;
- вывести pagemap:
- вывести содержимое файла io;
- вывести содержание файла comm;
- вывести root;
- вывести содержимое поддиректории task.

Файловая система proc предоставляет общую информацию о системе
Например:

/proc/filesystems
A text listing of the filesystems which are supported by

the kernel, namely filesystems which were compiled into the kernel or whose kernel modules are currently loaded. (See also filesystems(5).) If a filesystem is marked with "nodev", this means that it does not require a block device to be mounted (e.g., virtual filesystem, network filesystem).

Incidentally, this file may be used by mount(8) when no filesystem is specified and it didn't manage to determine the filesystem type. Then filesystems contained in this file are tried (excepted those that are marked with "nodev").

/proc/fs

Contains subdirectories that in turn contain files with information about (certain) mounted filesystems.

/proc/interrupts

This is used to record the number of interrupts per CPU per IO device. Since Linux 2.6.24, for the i386 and x86-64 architectures, at least, this also includes interrupts internal to the system (that is, not associated with a device as such), such as NMI (nonmaskable interrupt), LOC (local timer interrupt), and for SMP systems, TLB (TLB flush interrupt), RES (rescheduling interrupt), CAL (remote function call interrupt), and possibly others. Very easy to read formatting, done in ASCII.

/proc/ioports

This is a list of currently registered Input-Output port regions that are in use.

/proc/kcore

This file represents the physical memory of the system and is stored in the ELF core file format. With this pseudo-file, and an unstripped kernel (*/usr/src/linux/vmlinux*) binary, GDB can be used to examine the current state of any kernel data structures.

The total length of the file is the size of physical memory (RAM) plus 4 KiB.

/proc/kpagecount **(since Linux 2.6.25)**

This file contains a 64-bit count of the number of times each physical page frame is mapped, indexed by page frame number (see the discussion of */proc/[pid]/pagemap*).

The */proc/kpagecount* file is present only if the **CONFIG_PROC_PAGE_MONITOR** kernel configuration option is enabled.

/proc/meminfo

This file reports statistics about memory usage on the system. It is used by free(1) to report the amount of free and used memory (both physical and swap) on the

system as well as the shared memory and buffers used by the kernel. Each line of the file consists of a parameter name, followed by a colon, the value of the parameter, and an option unit of measurement (e.g., "kB"). The list below describes the parameter names and the format specifier required to read the field value. Except as noted below, all of the fields have been present since at least Linux 2.6.0. Some fields are displayed only if the kernel was configured with various options; those dependencies are noted in the list.

MemTotal %lu
: Total usable RAM (i.e., physical RAM minus a few reserved bits and the kernel binary code).

MemFree %lu
: The sum of *LowFree*+*HighFree*.

MemAvailable %lu (since Linux 3.14)
: An estimate of how much memory is available for starting new applications, without swapping.

Buffers %lu
: Relatively temporary storage for raw disk blocks that shouldn't get tremendously large (20 MB or so).

Cached %lu
: In-memory cache for files read from the disk (the page cache). Doesn't include *SwapCached*.

SwapCached %lu
: Memory that once was swapped out, is swapped back in but still also is in the swap file. (If memory pressure is high, these pages don't need to be swapped out again because they are already in the swap file. This saves I/O.)

Active %lu
: Memory that has been used more recently and usually not reclaimed unless absolutely necessary.

Inactive %lu
: Memory which has been less recently used. It is more eligible to be reclaimed for other purposes.

Active(anon) %lu (since Linux 2.6.28)
: [To be documented.]

Inactive(anon) %lu (since Linux 2.6.28)
: [To be documented.]

Active(file) %lu (since Linux 2.6.28)
[To be documented.]

Inactive(file) %lu (since Linux 2.6.28)
[To be documented.]

Unevictable %lu (since Linux 2.6.28)
(From Linux 2.6.28 to 2.6.30, **CONFIG_UNEVICTABLE_LRU** was required.) [To be documented.]

Mlocked %lu (since Linux 2.6.28)
(From Linux 2.6.28 to 2.6.30, **CONFIG_UNEVICTABLE_LRU** was required.) [To be documented.]

HighTotal %lu
(Starting with Linux 2.6.19, **CONFIG_HIGHMEM** is required.) Total amount of highmem. Highmem is all memory above ~860 MB of physical memory. Highmem areas are for use by user-space programs, or for the page cache. The kernel must use tricks to access this memory, making it slower to access than lowmem.

HighFree %lu
(Starting with Linux 2.6.19, **CONFIG_HIGHMEM** is required.) Amount of free highmem.

LowTotal %lu
(Starting with Linux 2.6.19, **CONFIG_HIGHMEM** is required.) Total amount of lowmem. Lowmem is memory which can be used for everything that highmem can be used for, but it is also available for the kernel's use for its own data structures. Among many other things, it is where everything from *Slab* is allocated. Bad things happen when you're out of lowmem.

LowFree %lu
(Starting with Linux 2.6.19, **CONFIG_HIGHMEM** is required.) Amount of free lowmem.

MmapCopy %lu (since Linux 2.6.29)
(**CONFIG_MMU** is required.) [To be documented.]

SwapTotal %lu
Total amount of swap space available.

SwapFree %lu
Amount of swap space that is currently unused.

Dirty %lu
: Memory which is waiting to get written back to the disk.

Writeback %lu
: Memory which is actively being written back to the disk.

AnonPages %lu (since Linux 2.6.18)
: Non-file backed pages mapped into user-space page tables.

Mapped %lu
: Files which have been mapped into memory (with mmap(2)), such as libraries.

Shmem %lu (since Linux 2.6.32)
: Amount of memory consumed in tmpfs(5) filesystems.

KReclaimable %lu (since Linux 4.20)
: Kernel allocations that the kernel will attempt to reclaim under memory pressure. Includes *SReclaimable* (below), and other direct allocations with a shrinker.

Slab %lu
: In-kernel data structures cache. (See slabinfo(5).)

SReclaimable %lu (since Linux 2.6.19)
: Part of *Slab*, that might be reclaimed, such as caches.

SUnreclaim %lu (since Linux 2.6.19)
: Part of *Slab*, that cannot be reclaimed on memory pressure.

KernelStack %lu (since Linux 2.6.32)
: Amount of memory allocated to kernel stacks.

PageTables %lu (since Linux 2.6.18)
: Amount of memory dedicated to the lowest level of page tables.

Quicklists %lu (since Linux 2.6.27)
: (**CONFIG_QUICKLIST** is required.) [To be documented.]

NFS_Unstable %lu (since Linux 2.6.18)
: NFS pages sent to the server, but not yet committed to stable storage.

Bounce %lu (since Linux 2.6.18)
Memory used for block device "bounce buffers".

WritebackTmp %lu (since Linux 2.6.26)
Memory used by FUSE for temporary writeback buffers.

CommitLimit %lu (since Linux 2.6.10)
This is the total amount of memory currently available to be allocated on the system, expressed in kilobytes. This limit is adhered to only if strict overcommit accounting is enabled (mode 2 in */proc/sys/vm/overcommit_memory*). The limit is calculated according to the formula described under */proc/sys/vm/overcommit_memory*. For further details, see the kernel source file *Documentation/vm/overcommit-accounting.rst*.

Committed_AS %lu
The amount of memory presently allocated on the system. The committed memory is a sum of all of the memory which has been allocated by processes, even if it has not been "used" by them as of yet. A process which allocates 1 GB of memory (using malloc(3) or similar), but touches only 300 MB of that memory will show up as using only 300 MB of memory even if it has the address space allocated for the entire 1 GB.

This 1 GB is memory which has been "committed" to by the VM and can be used at any time by the allocating application. With strict overcommit enabled on the system (mode 2 in */proc/sys/vm/overcommit_memory*), allocations which would exceed the *CommitLimit* will not be permitted. This is useful if one needs to guarantee that processes will not fail due to lack of memory once that memory has been successfully allocated.

VmallocTotal %lu
Total size of vmalloc memory area.

VmallocUsed %lu
Amount of vmalloc area which is used. Since Linux 4.4, this field is no longer calculated, and is hard coded as 0. See */proc/vmallocinfo*.

VmallocChunk %lu
Largest contiguous block of vmalloc area which is free. Since Linux 4.4, this field is no longer calculated and is hard coded as 0. See */proc/vmallocinfo*.

HardwareCorrupted %lu (since Linux 2.6.32)
(**CONFIG_MEMORY_FAILURE** is required.) [To be documented.]

LazyFree %lu (since Linux 4.12)
Shows the amount of memory marked by madvise(2) **MADV_FREE**.

AnonHugePages %lu (since Linux 2.6.38)
(**CONFIG_TRANSPARENT_HUGEPAGE** is required.) Non-file backed huge pages mapped into user-space page tables.

ShmemHugePages %lu (since Linux 4.8)
(**CONFIG_TRANSPARENT_HUGEPAGE** is required.) Memory used by shared memory (shmem) and tmpfs(5) allocated with huge pages.

ShmemPmdMapped %lu (since Linux 4.8)
(**CONFIG_TRANSPARENT_HUGEPAGE** is required.) Shared memory mapped into user space with huge pages.

CmaTotal %lu (since Linux 3.1)
Total CMA (Contiguous Memory Allocator) pages. (**CONFIG_CMA** is required.)

CmaFree %lu (since Linux 3.1)
Free CMA (Contiguous Memory Allocator) pages. (**CONFIG_CMA** is required.)

HugePages_Total %lu
(**CONFIG_HUGETLB_PAGE** is required.) The size of the pool of huge pages.

HugePages_Free %lu
(**CONFIG_HUGETLB_PAGE** is required.) The number of huge pages in the pool that are not yet allocated.

HugePages_Rsvd %lu (since Linux 2.6.17)
(**CONFIG_HUGETLB_PAGE** is required.) This is the number of huge pages for which a commitment to allocate from the pool has been made, but no allocation has yet been made. These reserved huge pages guarantee that an application will be able to allocate a huge page from the pool of huge pages at fault time.

HugePages_Surp %lu (since Linux 2.6.24)
(**CONFIG_HUGETLB_PAGE** is required.) This is the number of huge pages in the pool above the value in */proc/sys/vm/nr_hugepages*. The maximum number of

surplus huge pages is controlled by */proc/sys/vm/nr_overcommit_hugepages*.

Hugepagesize %lu
(**CONFIG_HUGETLB_PAGE** is required.) The size of huge pages.

DirectMap4k %lu (since Linux 2.6.27)
Number of bytes of RAM linearly mapped by kernel in 4 kB pages. (x86.)

DirectMap4M %lu (since Linux 2.6.27)
Number of bytes of RAM linearly mapped by kernel in 4 MB pages. (x86 with **CONFIG_X86_64** or **CONFIG_X86_PAE** enabled.)

DirectMap2M %lu (since Linux 2.6.27)
Number of bytes of RAM linearly mapped by kernel in 2 MB pages. (x86 with neither **CONFIG_X86_64** nor **CONFIG_X86_PAE** enabled.)

DirectMap1G %lu (since Linux 2.6.27)
(x86 with **CONFIG_X86_64** and **CONFIG_X86_DIRECT_GBPAGES** enabled.)

/proc/modules
A text list of the modules that have been loaded by the system. See also lsmod(8).