Семинар 7

Что такое виртуальная память

Читаем данный раздел, вопросов в нём нет, он вводный.

Изучение карты памяти процесса

Нас просят начать работу с запущенными процессами в каталоге /proc. Давайте сначала перейдём в него и посмотрим:

```
user@llp-ubuntu:~$ cd /proc
user@llp-ubuntu:/proc$ ls
1     1275  1479  1785  26    399  748        driver       pagetypeinfo
10    1281  1484  18    2630  4    750        execdomains  partitions
100   1284  15    19    2631  40   8          fb           sched_debug
101   1285  1501  192   2633  41   817        filesystems  schedstat
1013  1297  1506  193   268   42   826        fs           scsi
102   13    1517  194   27    43   852        interrupts   self
1035  1311  1519  195   2744  44   887        iomem        slabinfo
105   1323  1541  196   2751  45   888        ioports      softirqs
1067  1333  1547  198   276   46   9          irq          stat
109   1335  1580  199   2770  47   909        kallsyms     swaps
11    1347  1589  2     2775  48   97         kcore        sys
115   1356  1591  20    2776  49   98         keys         sysrq-trigger
1182  136   1592  2075  28    5    982        key-users    sysvipc
1184  137   1593  2082  2807  52   99         kmsg         thread-self
119   1382  16    2086  2815  53   acpi       kpagecgroup  timer_list
1196  1388  1624  209   30    54   asound     kpagecount   tty
12    1389  1626  2095  31    57   buddyinfo  kpageflags   uptime
1220  1390  1629  21    32    6    bus        loadavg      version
1222  1391  1662  22    33    690  cgroups    locks        version_signature
1224  1392  1671  2211  34    693  cmdline    mdstat       vmallocinfo
1226  1393  1673  222   35    695  consoles   meminfo      vmstat
1238  1394  1678  223   36    698  cpuinfo    misc         zoneinfo
1245  1395  1702  24    37    7    crypto     modules
1251  1396  1713  25    374   725  devices    mounts
1264  14    1729  2525  38    728  diskstats  mtrr
1273  1464  1730  258   39    734  dma        net
user@llp-ubuntu:/proc$
```

Видно, что просто запущено множество разных процессов(каждый процесс в своем каталоге)

Вопрос: посмотрите, какие у вас запущены процессы. Выберите один (например, оболочку, которой вы пользуетесь) и посмотрите на содержимое директории /proc/PID/, где PID — его идентификатор.

Ответ:

Но не будем спешить и **обязательно** прочитаем данную страничку. Там много полезного.
Из статьи сразу становится очевидно, что у всех процессов есть конкретные данные, например

- **cmdline** - содержит команду с помощью которой был запущен процесс, а также переданные ей параметры
- **cwd** - символическая ссылка на текущую рабочую директорию процесса
- **exe** - ссылка на исполняемый файл
- **root** - ссылка на папку суперпользователя
- **environ** - переменные окружения, доступные для процесса
- **fd** - содержит файловые дескрипторы, файлы и устройства, которые использует процесс
- **maps, statm, и mem** - информация о памяти процесса
- **stat, status** - состояние процесса

ну зайдём в любой процесс и посмотрим на его содержимое(я выбрал процесс **222**, вы можете выбрать любой из своих):

```
user@llp-ubuntu:/proc/222$ cat cmdline
user@llp-ubuntu:/proc/222$ ls
ls: cannot read symbolic link 'cwd': Permission denied
ls: cannot read symbolic link 'root': Permission denied
ls: cannot read symbolic link 'exe': Permission denied
attr             exe         mounts         projid_map    status
autogroup        fd          mountstats     root          syscall
auxv             fdinfo      net            sched         task
cgroup           gid_map     ns             schedstat     timers
clear_refs       io          numa_maps      sessionid     timerslack_ns
cmdline          limits      oom_adj        setgroups     uid_map
comm             loginuid    oom_score      smaps         wchan
coredump_filter  map_files   oom_score_adj  smaps_rollup
cpuset           maps        pagemap        stack
cwd              mem         patch_state    stat
environ          mountinfo   personality    statm
user@llp-ubuntu:/proc/222$
```

Видим, что обязательные файлы присутствуют. Некоторые из них нельзя посмотреть без админки(их права в таком случае 400). Это можно сделать с командой sudo перед вашей командой. Ну я посмотрел случайные файлы у процесса, но я даже не знаю что делает данный процесс, так что информация не очень полезна:

```
user@llp-ubuntu:/proc/222$ ls
ls: cannot read symbolic link 'cwd': Permission denied
ls: cannot read symbolic link 'root': Permission denied
ls: cannot read symbolic link 'exe': Permission denied
attr             exe        mounts         projid_map    status
autogroup        fd         mountstats     root          syscall
auxv             fdinfo     net            sched         task
cgroup           gid_map    ns             schedstat     timers
clear_refs       io         numa_maps      sessionid     timerslack_ns
cmdline          limits     oom_adj        setgroups     uid_map
comm             loginuid   oom_score      smaps         wchan
coredump_filter  map_files  oom_score_adj  smaps_rollup
cpuset           maps       pagemap        stack
cwd              mem        patch_state    stat
environ          mountinfo  personality    statm
user@llp-ubuntu:/proc/222$ sudo cat syscall
0 0x0 0x0 0x0 0x0 0x0 0x0 0x0 0x0
user@llp-ubuntu:/proc/222$ sudo cat stack
[<0>] __wait_on_buffer+0x32/0x40
[<0>] jbd2_journal_commit_transaction+0x104e/0x1870
[<0>] kjournald2+0xc8/0x250
[<0>] kthread+0x105/0x140
[<0>] ret_from_fork+0x35/0x40
[<0>] 0xffffffffffffffff
user@llp-ubuntu:/proc/222$
```

Вопрос: что внутри файла */proc/PID/environ*?

Ответ: **данный файл содержит переменные окружения, доступные для процесса**

В моём случае у процесса **222** при выводе этого файла ничего не показывается, а значит процесс не использует переменные окружения. Но я не расстроился и **вывел переменные всех процессов:**

```
user@llp-ubuntu:/proc$ cat */environ
cat: 100/environ: Permission denied
LANG=en_US.UTF-8LC_ADDRESS=fr_FR.UTF-8LC_IDENTIFICATION=fr_FR.UTF-8LC_MEASUREME
NT=fr_FR.UTF-8LC_MONETARY=fr_FR.UTF-8LC_NAME=fr_FR.UTF-8LC_NUMERIC=fr_FR.UTF-8L
C_PAPER=fr_FR.UTF-8LC_TELEPHONE=fr_FR.UTF-8LC_TIME=fr_FR.UTF-8PATH=/usr/local/s
bin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/snap/binNOTIFY_SOCKET=/run/sy
stemd/notifyHOME=/home/userLOGNAME=userUSER=userSHELL=/bin/bashXDG_RUNTIME_DIR=
/run/user/1000cat: 101/environ: Permission denied
cat: 102/environ: Permission denied
cat: 1035/environ: Permission denied
cat: 105/environ: Permission denied
XDG_VTNR=7LC_PAPER=fr_FR.UTF-8XDG_SESSION_ID=c1LC_ADDRESS=fr_FR.UTF-8CLUTTER_IM
_MODULE=ximXDG_GREETER_DATA_DIR=/var/lib/lightdm-data/userLC_MONETARY=fr_FR.UTF
-8SHELL=/bin/bashQT_LINUX_ACCESSIBILITY_ALWAYS_ON=1LC_NUMERIC=fr_FR.UTF-8GTK_MO
DULES=gail:atk-bridgeUSER=userQT_ACCESSIBILITY=1LC_TELEPHONE=fr_FR.UTF-8XDG_SES
SION_PATH=/org/freedesktop/DisplayManager/Session0XDG_SEAT_PATH=/org/freedeskto
p/DisplayManager/Seat0DEFAULTS_PATH=/usr/share/gconf/ubuntu.default.pathXDG_CON
FIG_DIRS=/etc/xdg/xdg-ubuntu:/usr/share/upstart/xdg:/etc/xdgDESKTOP_SESSION=ubu
ntuPATH=/home/user/bin:/home/user/.local/bin:/usr/local/sbin:/usr/local/bin:/us
r/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/snap/binQT_IM_MODULE=ib
usQT_QPA_PLATFORMTHEME=appmenu-qt5LC_IDENTIFICATION=fr_FR.UTF-8PWD=/home/userXD
G_SESSION_TYPE=x11XMODIFIERS=@im=ibusLANG=en_US.UTF-8MANDATORY_PATH=/usr/share/
gconf/ubuntu.mandatory.pathGDM_LANG=en_USLC_MEASUREMENT=fr_FR.UTF-8IM_CONFIG_PH
ASE=1COMPIZ_CONFIG_PROFILE=ubuntuGDMSESSION=ubuntuSESSIONTYPE=gnome-sessionGTK2
```

Не бойтесь, этот список огромный, на 5+ экранов, так как выводит все переменные окружения(и это с учетом того, что environ некоторых процессов закрыт для чтения), тут скорее мы хотим убедиться что они не всегда пустые. Чуть вспомнив как работает звездочка при выборке у команды **cat** я вывел environ у конкретного одного файла, где он не пустой. Этим файлом оказался процесс с **ID = 1013:**

```
user@llp-ubuntu:/proc$ cat 1013/environ
LANG=en_US.UTF-8LC_ADDRESS=fr_FR.UTF-8LC_IDENTIFICATION=fr_FR.UTF-8LC_MEASUREME
NT=fr_FR.UTF-8LC_MONETARY=fr_FR.UTF-8LC_NAME=fr_FR.UTF-8LC_NUMERIC=fr_FR.UTF-8L
C_PAPER=fr_FR.UTF-8LC_TELEPHONE=fr_FR.UTF-8LC_TIME=fr_FR.UTF-8PATH=/usr/local/s
bin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/snap/binNOTIFY_SOCKET=/run/sy
stemd/notifyHOME=/home/userLOGNAME=userUSER=userSHELL=/bin/bashXDG_RUNTIME_DIR=
/run/user/1000user@llp-ubuntu:/proc$
```

Убедились, что у каких-то процессов всё же есть такие переменные окружения.

> **Вопрос:** Прочитайте про запуск программ в фоне. Что делают команды bg, fg, jobs?
> **Ответ:** Прочитаем по ссылочке.

Узнать состояние всех остановленных и выполняемых в фоновом режиме задач в рамках текущей сессии терминала можно при помощи утилиты jobs с использованием опции **-l**:

```
$ jobs -l
```

В любое время можно вернуть процесс из фонового режима на передний план. Для этого служит команда fg:

```
$ fg
```

Если в фоновом режиме выполняется несколько программ, следует также указывать номер. Например:

```
$ fg %1
```

Если изначально процесс был запущен обычным способом, его можно перевести в фоновый режим, выполнив следующие действия:

1. Остановить выполнение команды, нажав комбинацию клавиш **Ctrl+Z**.
2. Перевести процесс в фоновый режим при помощи команды **bg**.

```
$ bg
```

Двигаемся далее по семинару и запускаем свой бесконечный(и бесполезный) процесс по коду, который дан в семинаре:

```
section .data
```

```
correct: dq -1
section .text
global _start
_start:
jmp _start
```

Ну видно, что данная обосанная программа просто прыгает сама на себя и выполняет переход на саму себя снова и снова.
Ну давайте запустим её в фоновом режиме:(перед этим создали и скомпоновали в запускаемый файл, как мы это делали раньше с прочими ассемблерными файликами)

```
user@llp-ubuntu:~/seminars/sem7$ cat inf-loop-0.asm
section .data
correct: dq -1
section .text
global _start
_start:
jmp _start
user@llp-ubuntu:~/seminars/sem7$ nasm -g inf-loop-0.asm -felf64 -o loop0.o
user@llp-ubuntu:~/seminars/sem7$ ld -o loop0 loop0.o
user@llp-ubuntu:~/seminars/sem7$ ls
inf-loop-0.asm  loop0  loop0.o
user@llp-ubuntu:~/seminars/sem7$
```

Ну и **запускаем процесс в фоновом режиме:**

```
user@llp-ubuntu:~/seminars/sem7$ ./loop0 &
[2] 3309
user@llp-ubuntu:~/seminars/sem7$
```

ну и пусть работает с номером 3309 в системе и с номером [2] запущенным от пользователя(то есть меня, вас). Ну и сразу же хочется зайти в папку **/proc** и проверить, не появился ли там наш процесс(появился)?

Вопрос: выведите с помощью **cat** содержимое файла /proc/PID/maps, где PID — идентификатор процесса, который вы запустили в фоне. У меня **PID=3309**, думаю не надо пояснять, что у вас он вполне может оказаться другим…

Ответ: идём и смотрим результат работы вывода maps для нашего процесса:

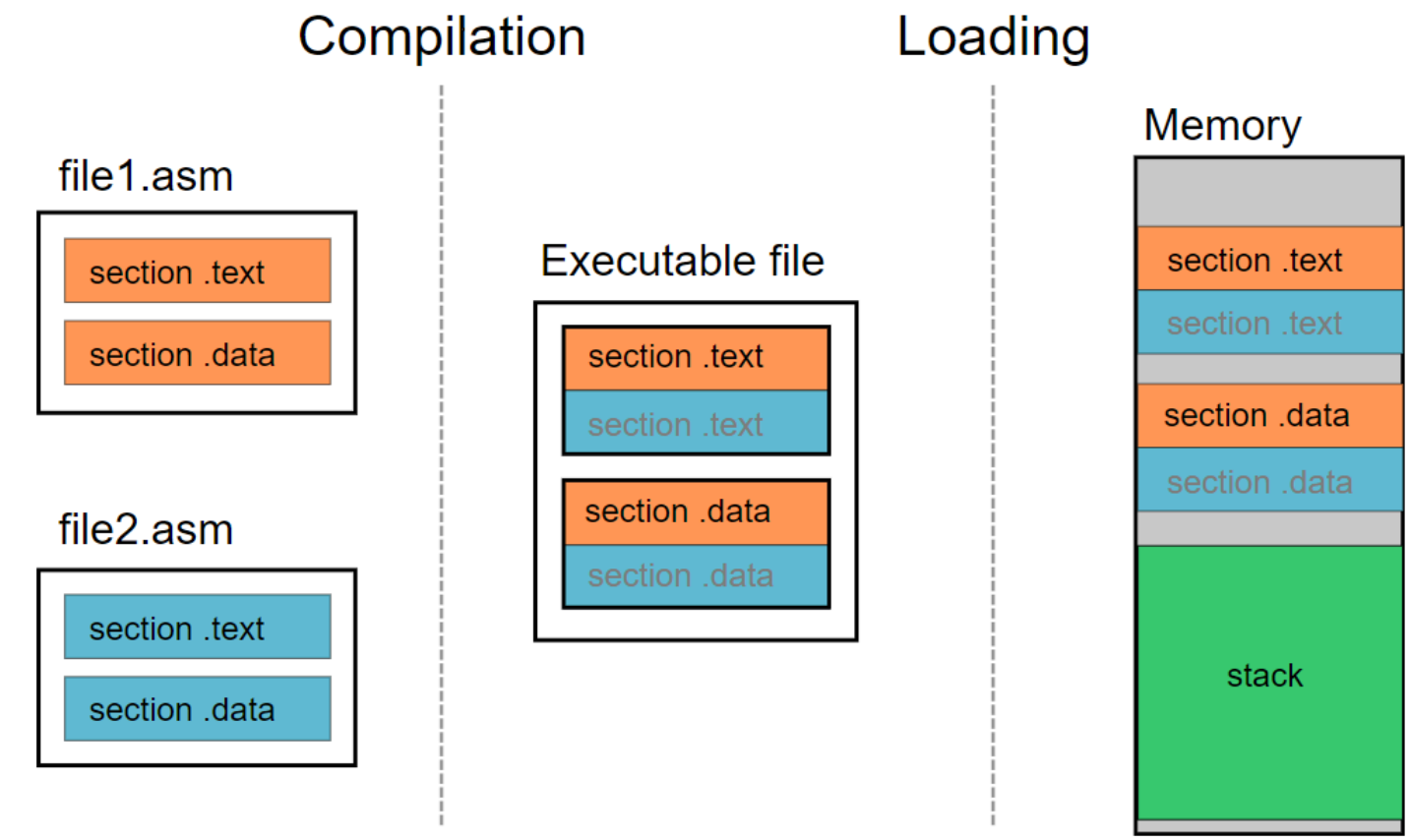
```
user@llp-ubuntu:~/seminars/sem7$ cat /proc/3309/maps
00400000-00401000 r-xp 00000000 08:01 275521                             /home/u
ser/seminars/sem7/loop0
00600000-00601000 rwxp 00000000 08:01 275521                             /home/u
ser/seminars/sem7/loop0
7ffdffc4b000-7ffdffc6c000 rwxp 00000000 00:00 0                          [stack]
7ffdffdba000-7ffdffdbd000 r--p 00000000 00:00 0                          [vvar]
7ffdffdbd000-7ffdffdbf000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsysca
ll]
user@llp-ubuntu:~/seminars/sem7$
```

с примерным выводом совпадает, читаем из семинара дальше:
Первая колонка это *регион памяти*, который описывается далее в других колонках.
Непрерывная область памяти называется *регионом*, если она состоит из целого числа страниц с одинаковыми флагами.

Вопрос: почему в регионах начальный и конечный адрес в 16-ричном формате заканчиваются всегда на три нуля?

Ответ: ~~создатели памяти кончи~~ вспомним самое начало семинара, было сказано, что все разбивается на виртуальную память по 4КБ каждая, и адреса должны быть кратны 4КБ. Если посчитать это в 16ричной системе, то выйдет, что $??? 000_{16} = ?? ?_{10} * 2^{12}$ну а это и есть 4096 адресов, а значит это просто гарант делимости адреса на 4КБ, чтобы страницу можно было размещать.

Идём дальше, картиночка, которая особо не сложная:



Ну и вопросы к ней:

Вопрос: определите, по каким адресам загружаются секции *.text* и *.data* из примера. Вам может помочь *readelf и таблица символов*.

Ответ: выведем таблицу символов для собранного файла(но не для запущенного процесса, такой опции я не нашел)

```
user@llp-ubuntu:~/seminars/sem7$ readelf -s loop0

Symbol table '.symtab' contains 11 entries:
   Num:    Value          Size Type    Bind   Vis      Ndx Name
     0: 0000000000000000     0 NOTYPE  LOCAL  DEFAULT  UND
     1: 00000000004000b0     0 SECTION LOCAL  DEFAULT    1
     2: 00000000006000b4     0 SECTION LOCAL  DEFAULT    2
     3: 0000000000000000     0 SECTION LOCAL  DEFAULT    3
     4: 0000000000000000     0 SECTION LOCAL  DEFAULT    4
     5: 0000000000000000     0 FILE    LOCAL  DEFAULT  ABS inf-loop-0.asm
     6: 00000000006000b4     8 OBJECT  LOCAL  DEFAULT    2 correct
     7: 00000000004000b0     0 NOTYPE  GLOBAL DEFAULT    1 _start
     8: 00000000006000bc     0 NOTYPE  GLOBAL DEFAULT    2 __bss_start
     9: 00000000006000bc     0 NOTYPE  GLOBAL DEFAULT    2 _edata
    10: 00000000006000c0     0 NOTYPE  GLOBAL DEFAULT    2 _end
user@llp-ubuntu:~/seminars/sem7$
```

Помня, что Value хранит информацию о 2 значениях, обратим внимание лишь на вторую половину(она отвечает за адрес). Видно, что секции могут начинаться с 400, а могут с 600. Видимо это и есть то, что от нас просят.

Можно еще написать readelf -l, тогда выводятся секции

```
Section to Segment mapping:
 Segment Sections...
  00
  01     .text
  02     .data
```

Вопрос: определите хотя бы один запрещённый диапазон адресов.
Ответ: можно найти тут.

Список всех зарезервированных областей памяти в системе можно просмотреть командой `lspci -m`:

```
 lsipc -m
KEY        ID          PERMS OWNER  SIZE NATTCH STATUS CTIME  CPID  LPID COMMAND
0xbe130fa1 3112960 rw-------  root 1000B     11        May03  7217  9422 /usr/sbin/httpd -DFOREGROUND
0x00000000 557057  rw------- usr74  384K      2 dest   Apr28 17752  7476 kdeinit4: konsole [kdeinit]
0x00000000 5898243 rw------- usr92  512K      2 dest   12:05  5265  9678 /usr/bin/geany /home/s0392/1_1.s
0x00000000 4521988 rw------- usr75  384K      2 dest   May06 22351 16323 sview
0x00000000 3276805 rw------- usr15  384K      1 dest   May05 24835 15236
0x00000000 4587530 rw------- usr75    2M      2 dest   May06 19404 16323 metacity
```

Введём команду lsipc -m и получим сразу множество зарезервированных областей. Очевидно, что под них память выделиться уже не может.

```
user@llp-ubuntu:~/seminars/sem7$ lsipc -m
KEY        ID          PERMS OWNER SIZE NATTCH STATUS CTIME CPID LPID COMMAND
0x00000000 65536   rw-------  user 512K      2 dest   21:14 1285 3505 /usr/lib/i
0x00000000 1376257 rw-------  user 512K      2 dest   21:15 2211 3505 update-not
0x00000000 196610  rw-------  user 512K      2 dest   21:14 1356 3505 /usr/lib/x
0x00000000 393219  rw-------  user  16M      2        21:14 1501 3505 compiz
0x00000000 491524  rw-------  user 512K      2 dest   21:14 1593 3505 nm-applet
0x00000000 655365  rw-------  user 512K      2 dest   21:14 1580 3505 nautilus -
0x00000000 753670  rw-------  user 512K      2 dest   21:14 1501 3505 compiz
0x00000000 1048583 rw-------  user 512K      2 dest   21:14 1251 3505 /usr/lib/x
0x00000000 917512  rw-------  user 512K      2 dest   21:14 1501 3505 compiz
0x00000000 950281  rw-------  user  16M      2 dest   21:14 1580 3505 nautilus -
0x00000000 1605642 rw-------  user 512K      2 dest   21:25 2744 3505 /usr/lib/g
user@llp-ubuntu:~/seminars/sem7$
```

Вопрос: Что такое inode в файловой системе?
Ответ: ~~ура ОПД~~ inode - это файловый дескриптор или метаданные, то есть данные о данных, отвечают за много всего, реализуют организацию файловой системы. Про это можно много всякого всюду найти. Основное: inode есть у любого файла(каталога), в ней есть глобальный номер файла(по сути указатель для поиска по памяти), указаны права, тип файла и прочее(некоторые времена)...

Вопрос: Что находится в остальных столбцах? Прочитайте про файл */proc/PID/maps* в *man procfs*.
Ответ: Можете запустить этот *man*, в целом в нём есть очень много полезного про процессы и их переменные, такие как тот же *environ*, который уже рассматривался, но я вставлю сюда секцию об остальных столбцах.
/proc/[pid]/maps

A file containing the currently mapped memory regions and their access permissions. See mmap(2) for some further information about memory mappings.

The format of the file is:

```
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
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]
```

The address field is the address space in the process that the mapping occupies. 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).

There are additional helpful pseudo-paths:

[stack]
The initial process's (also known as the

main thread's) stack.

[stack:<tid>] (since Linux 3.4)
A thread's stack (where the <tid> is a thread ID). It corresponds to the /proc/[pid]/task/[tid]/ path.

[vdso] The virtual dynamically linked shared object.

[heap] The process's heap.
If the pathname field is blank, this is an anonymous mapping as obtained via the mmap(2) function. There is no easy way to coordinate this back to a process's source, short of running it through gdb(1), strace(1), or similar.

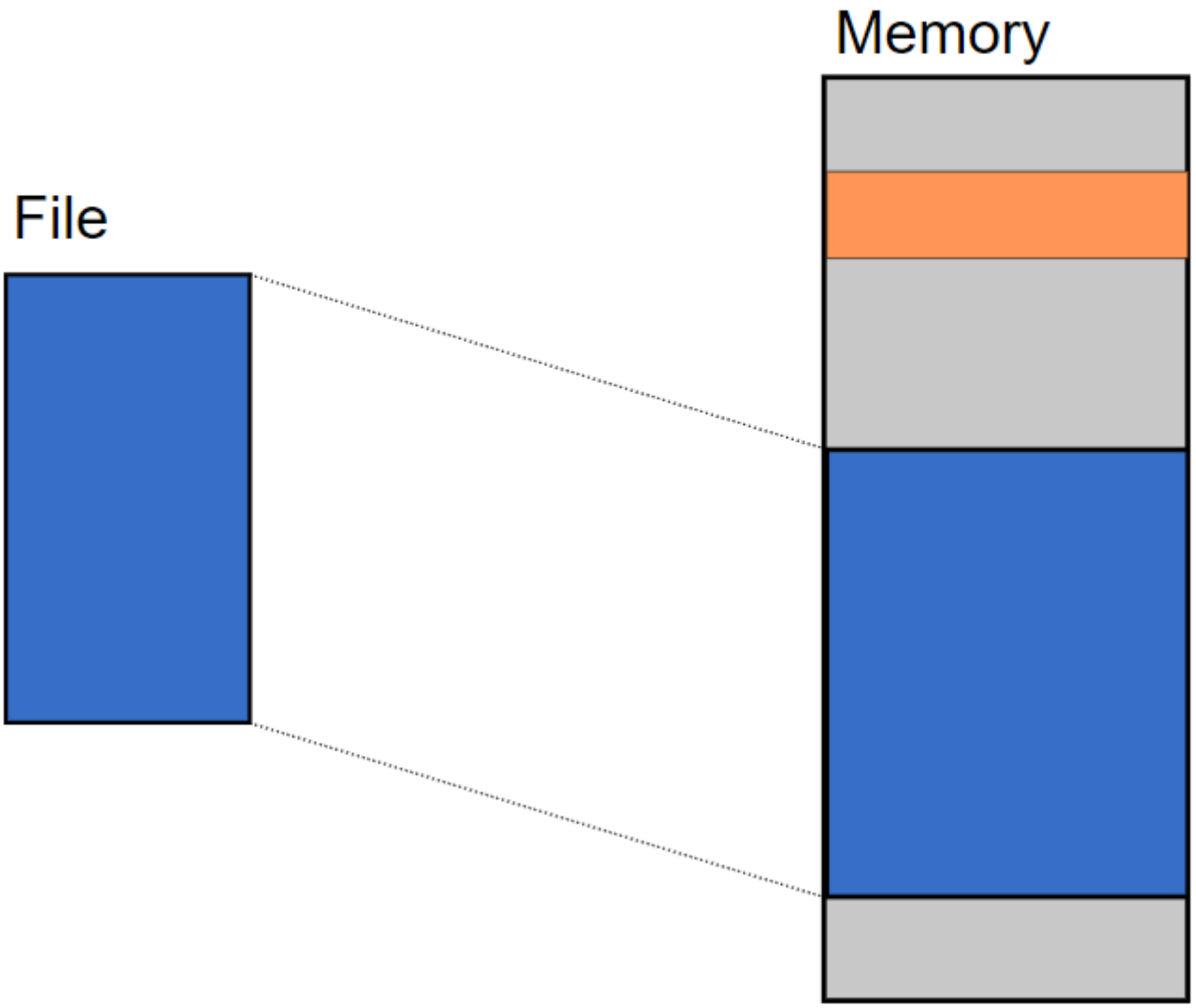
Under Linux 2.0, there is no field giving pathname.

Ознакомьтесь с этой информацией сами, я английский не шарю, может Егор еще распишет.

Использование *mmap*.

Отображение файла в память (memory mapped files) — это способ работы с файлами, при котором файлу ставится в соответствие диапазон адресов виртуальной памяти. Операционная система прозрачно для программиста организует этот процесс так, чтобы чтение данных из этих адресов приводило к чтению данных из отображенного файла, а запись данных по этим адресам приводила к записи в файл.
В *nix подобных системах файлы также используются как абстракции для устройств, то есть взаимодействие с файлом означает взаимодействие с устройством. Такие файлы обычно находятся в директории /dev. Файлы устройств тоже можно отображать в память.

Memory mapped file



Вопрос: прочитайте в *man mmap* ответы на следующие вопросы:

- Какие аргументы принимает mmap? **Ответ:**
- В чём их смысл? **Ответ:**
- Какой аргумент в каком регистре? **Ответ:**

Пожалуй я просто скину вам *man*, а дальше распишу, если найду время и силы хотя бы понять самому:

```
MMAP(2)          Linux Programmer's Manual          MMAP(2)

NAME
       mmap, munmap - map or unmap files or devices into memory

SYNOPSIS
       #include <sys/mman.h>

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

See NOTES for information on feature test macro requirements.

DESCRIPTION

mmap() creates a new mapping in the virtual address space of the calling process. The starting address for the new mapping is specified in addr. The length argument specifies the length of the mapping.

If addr is NULL, then the kernel chooses the address at which to create the mapping; this is the most portable method of creating a new mapping. If addr is not NULL, then the kernel takes it as a hint about where to place the mapping; on Linux, the mapping will be created at a nearby page boundary. The address of the new mapping is returned as the result of the call.

The contents of a file mapping (as opposed to an anonymous mapping; see MAP_ANONYMOUS below), are initialized using length bytes starting at offset offset in the file (or other object) referred to by the file descriptor fd. offset must be a multiple of the page size as returned by sysconf(_SC_PAGE_SIZE).

The prot argument describes the desired memory protection of the mapping (and must not conflict with the open mode of the file). It is either PROT_NONE or the bitwise OR of one or more of the following flags:

PROT_EXEC Pages may be executed.

PROT_READ Pages may be read.

PROT_WRITE Pages may be written.

PROT_NONE Pages may not be accessed.

The flags argument determines whether updates to the mapping are visible to other processes mapping the same region, and whether updates are carried through to the underlying file. This behavior is determined by including exactly one of the following values in flags:

MAP_SHARED
Share this mapping. Updates to the mapping are visible to other processes that map this file, and are carried through to the underlying file. (To precisely control when updates are carried through to the underlying file requires the use of msync(2).)

MAP_PRIVATE
Create a private copy-on-write mapping. Updates to the mapping are not visible to other processes mapping the same file, and are not carried through to the underlying file. It is unspecified whether changes made to the file after the mmap() call are visible in the mapped region.

Both of these flags are described in POSIX.1-2001 and POSIX.1-2008.

In addition, zero or more of the following values can be ORed in flags:

MAP_32BIT (since Linux 2.4.20, 2.6)
Put the mapping into the first 2 Gigabytes of the process address space. This flag is supported only on x86-64, for 64-bit programs. It was added to allow thread stacks to be allocated somewhere in the first 2GB of memory, so as to improve context-switch performance on some early 64-bit processors. Modern x86-64 processors no longer have this performance problem, so use of this flag is not required on those systems. The MAP_32BIT flag is ignored when MAP_FIXED is set.

MAP_ANON
Synonym for MAP_ANONYMOUS. Deprecated.

MAP_ANONYMOUS
The mapping is not backed by any file; its contents are initialized to zero. The fd and offset arguments are ignored; however, some implementations require fd to be -1

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

Функция возвращает адрес, по которому был замаплен файл
*addr - предпочтительный адрес виртуальной памяти по которому будет начат мапинг (предпочтительный потому что если у системы чето не получится то она сама выберет адрес)
length - длина размера замапленой области в байтах
prot - режим защиты:
PROT_EXECUTE: pages can be executed
PROT_READ: pages can be read
PROT_WRITE: pages can be written into
PROT_NONE: pages cannot be accessed
flags - режим видимости изменений другим процессам на том же регионе (там их дохуя)
fd - дескриптор файла который мапится в память

offset - отступ с начала файла, с которого будем мапить

Программа: чуть вводной — код ниже представлен для какого-то вашего файла, который будет работать с файлом ***fname***, в качестве примера указан hello.txt, в него помещаем любую строчку, которую хотим потом увидеть(я например поместил строчку из известной песни)

```
%define O_RDONLY 0
%define PROT_READ 0x1
%define MAP_PRIVATE 0x2
%define SYS_OPEN 2
%define SYS_MMAP 9
%define STR_LEN 12

section .data
; This is the file name. You are free to change it. **fname here**
fname: db 'hello.txt', 0

section .text
global _start
exit:
  mov rax, 60          ; use exit system call to shut down correctly
  xor rdi, rdi
  syscall
; These functions are used to print a null terminated string
print_string:
    push rdi
    call string_length
    pop rsi
    mov rdx, rax
    mov rax, 1
    mov rdi, 1
    syscall
    ret
string_length:
    xor rax, rax
.loop:
    cmp byte [rdi+rax], 0
    je .end
    inc rax
    jmp .loop
.end:
    ret

_start:

; Вызовите open и откройте fname в режиме read only.
mov rax, SYS_OPEN
mov rdi, fname
mov rsi, O_RDONLY    ; Open file read only
```

```
mov rdx, 0          ; We are not creating a file
                    ; so this argument has no meaning
syscall


; Вызовите mmap с правильными аргументами
; Дайте операционной системе самой выбрать, куда отобразить файл
; Размер области возьмите в размер страницы
; Область не должна быть общей для нескольких процессов
;  и должна выделяться только для чтения.

mov rdi, 0
mov rsi, STR_LEN
mov rdx, PROT_READ
mov r10, MAP_PRIVATE
mov r8, rax
mov r9, 0
mov rax, SYS_MMAP
syscall


; с помощью print_string теперь можно вывести его содержимое

mov rdi, rax
call print_string

call exit
```

Теперь займёмся вызовом всего этого: собираем данный файл и запускаем

```
user@llp-ubuntu:~/seminars/sem7$ ./opencheck
Never gonna give you up
Never gonna let you down
Never gonna run around and desert you
Never gonna make you cry
Never gonna say goodbye
Never gonna tell a lie and hurt you
user@llp-ubuntu:~/seminars/sem7$
```

Что вообще тут происходит: команда **open** открывает файл для чтения, далее она передаёт его **дескриптор** в команду **mmap**, которая уже подгружает его в память программы. После этого команда **mmap** вернёт нам адрес в памяти у файла. Мы его используем для вывода содержимого, выводим как 0-терминированную строку.
На самом деле пока есть пару вопросов про длину и смещение в атрибутах команд…
Вроде **mmap** должен выделять в памяти более 1 страницы, но если вы так сделает, то у вас будет ошибка **Segmentation Fault**, по **моим**(что проверено вызовами с комментированием разных участков программ) предположениям это вызвано тем, что **mmap** подгрузит эти разные

страницы в разные участки памяти(а не последовательно) и вызов вывода данных упадет, так как будет обращение к запрещенным участкам памяти.

Благодарность выражается:
Иванову Евгению
Полошкову Борису