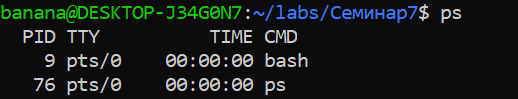
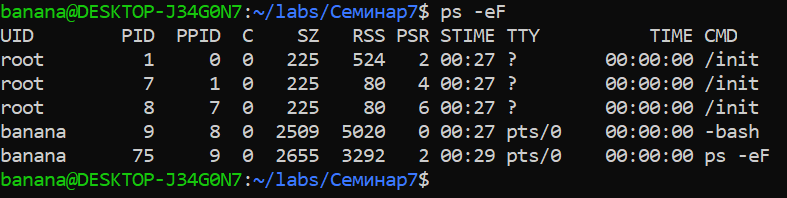
**1.Вопрос** посмотрите, какие у вас запущены процессы. Выберите один (например, оболочку, которой вы пользуетесь) и посмотрите на содержимое директории /proc/PID/, где PID — его идентификатор.

**УТИЛИТА PS**

$ps- для просмотра списка процессов, запущенных в текущей командной оболочке, используемой команду ps без параметров.

$ps -eF- для просмотра всех процессов нужно добавить опцию -e, а для максимально подробной информации -F.

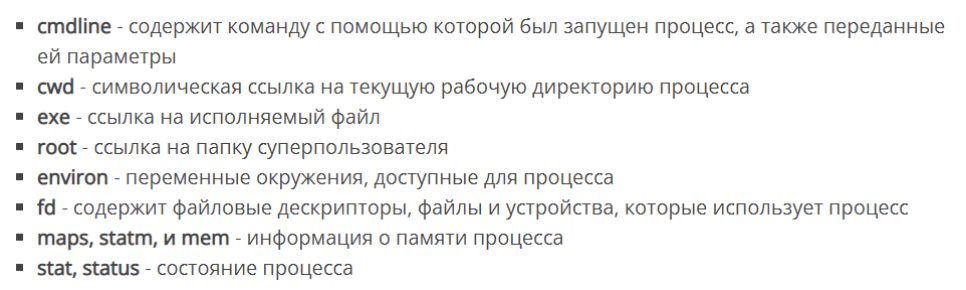
Для начала посмотрим, какие у меня есть процессы:



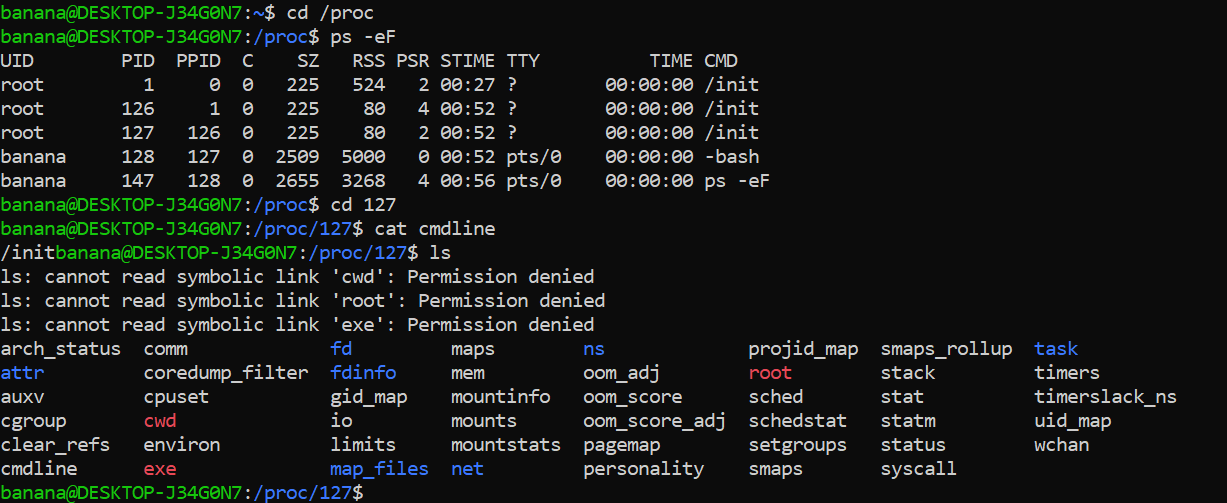
Значение основных колонок в выводе утилиты:

* **UID** - имя пользователя, от имени которого работает процесс;
* **PID** - идентификатор пользователя;
* **PPID** - идентификатор родительского процесса пользователя;
* **C** - расходование ресурсов процессора, в процентах;
* **SZ** - размер процесса;
* **RSS** - реальный размер процесса в памяти;
* **PSR** - ядро процессора, на котором выполняется процесс;
* **STIME** - время, когда процесс был запущен;
* **TTY** - если процесс привязан к терминалу, то здесь будет выведен его номер;
* **TIME** - общее время выполнения процесса (user + system);
* **CMD** - команда, которой был запущен процесс, если программа не может прочитать аргументы процесса, он будет выведен в квадратных скобках;

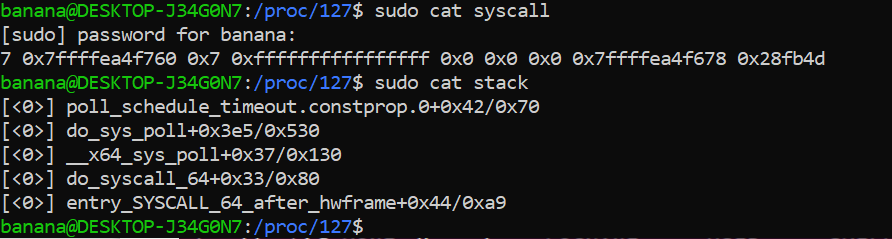
**А также обратим внимание на то, что у всех процессов есть конкретные данные:**



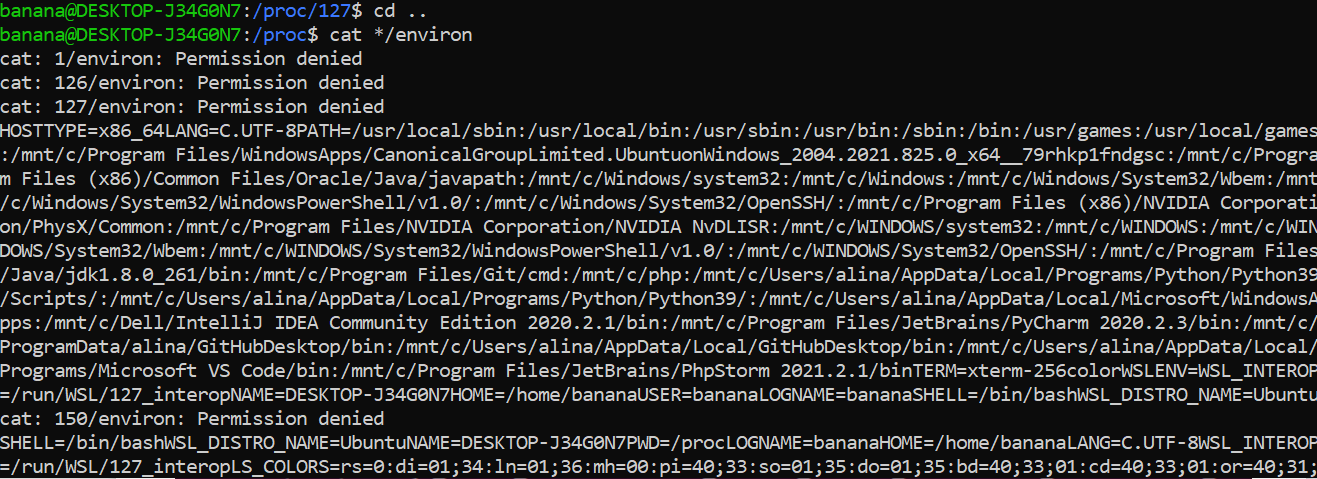
**Теперь зайду в любой процесс(возьму, например, процесс 127):**

****

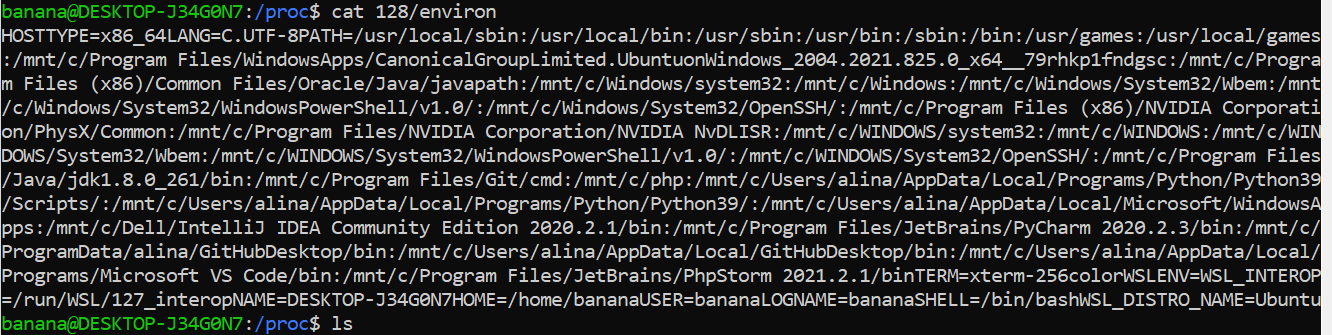
Видим, что обязательные файлы присутствуют. Некоторые из них нельзя посмотреть без админки (их права в таком случае 400). Это можно сделать с командой sudo перед самой командой.

**2.Вопрос** что внутри файла /proc/PID/environ?

Этот файл содержит переменные окружения, доступные для процесса. В моём случае у процесса 127 при выводе этого файла ничего не показывается, а значит процесс не использует переменные окружения.

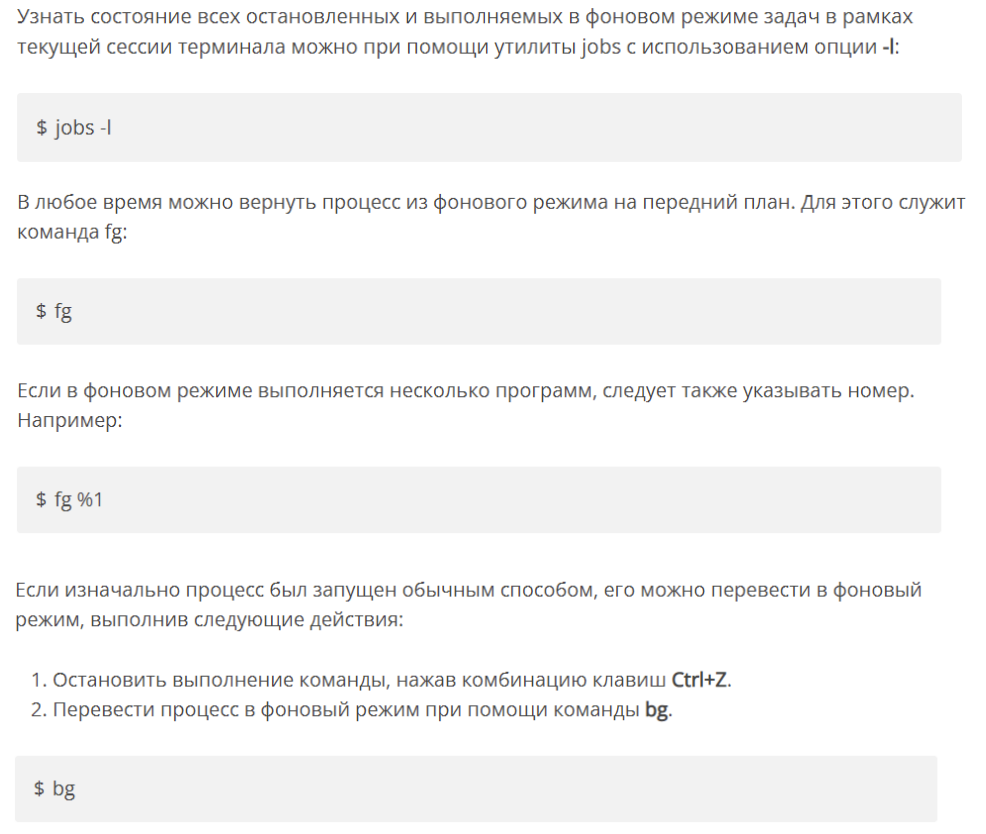
Для наглядности выведу переменные всех процессов:

Мы видим большой список, 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, выведем environ у конкретного одного файла, где он не пустой. Этим файлом оказался процесс с ID = 128:

****

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

**3.Вопрос** Прочитайте про запуск программ в фоне. Что делают команды bg, fg, jobs?



Напишем простую программу, которая входит в бесконечный цикл:

section .data

correct: dq -1

section .text

global \_start

\_start:

jmp \_start

Что она делает? Скомпилируйте и запустите её в фоне.

Запустим бесконечный процесс:

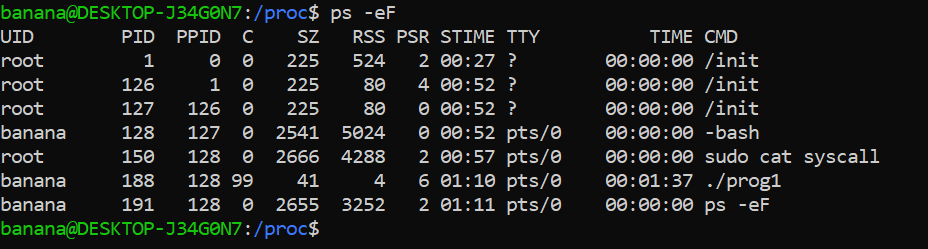
Тут мы видим, что данная программа просто прыгает сама на себя и выполняет переход на саму себя снова и снова.

А теперь запустим её в фоновом режиме:



Она работает с номером 188 в системе и с номером [2] запущенным от пользователя.

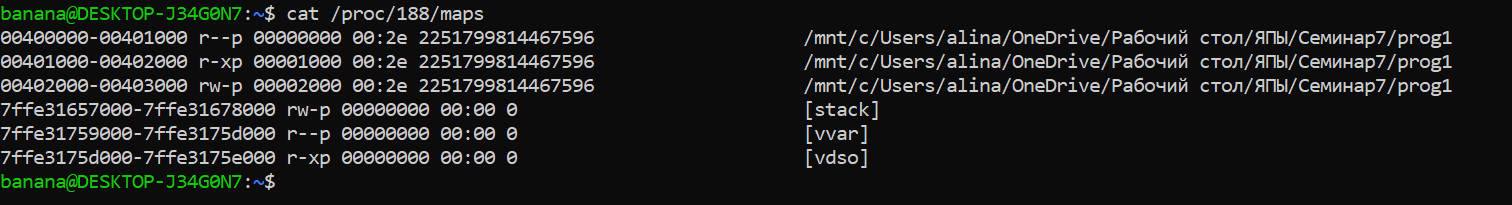
Ну и сразу же хочется зайти в папку /proc и проверить, не появился ли там наш процесс(появился)?



Мы убедились, что он тута.

**4.Вопрос** выведите с помощью cat содержимое файла /proc/PID/maps, где PID — идентификатор процесса, который вы запустили в фоне.

Смотрим результат работы вывода maps для процесса 188:

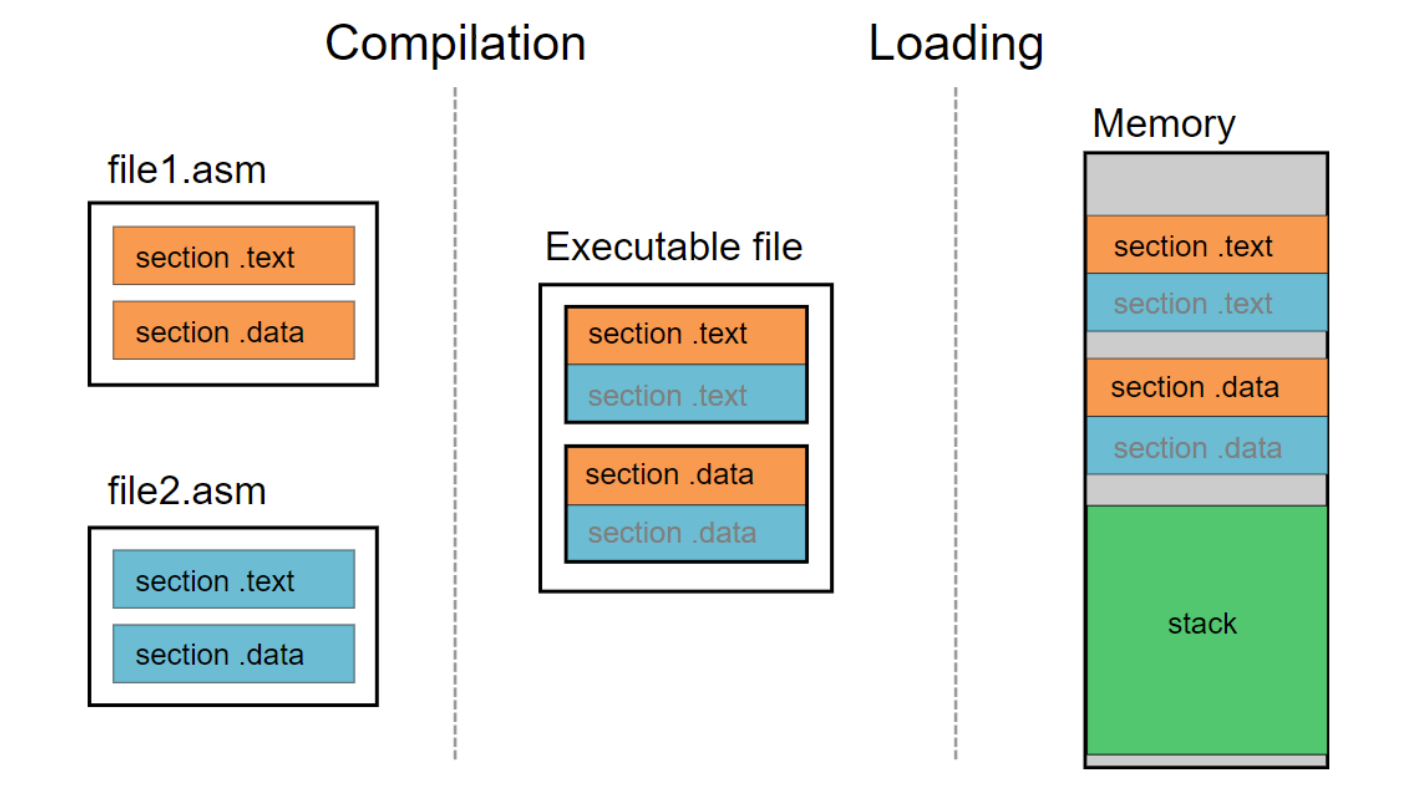


с примерным выводом совпадает, читаем из семинара дальше:

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.

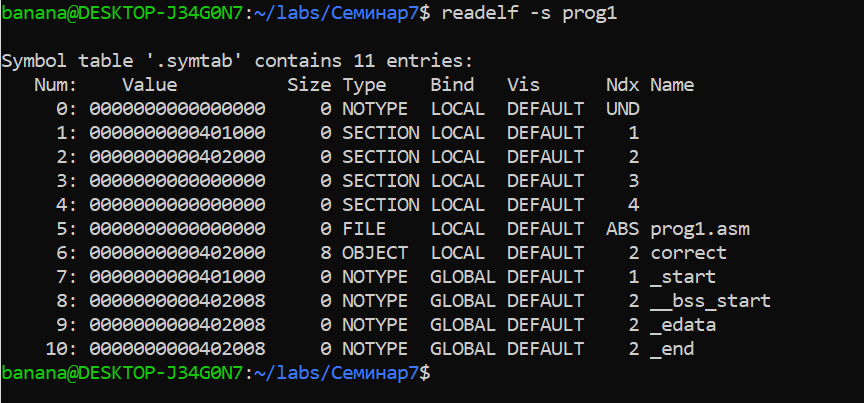
**5.Вопрос** почему в регионах начальный и конечный адрес в 16-ричном формате заканчиваются всегда на три нуля?

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 ну а это и есть 16 =?? ? 10 \* 2 12 4096 адресов, а значит это просто гарант делимости адреса на 4КБ, чтобы страницу можно было размещать. Идём дальше, картиночка, которая особо не сложная:



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

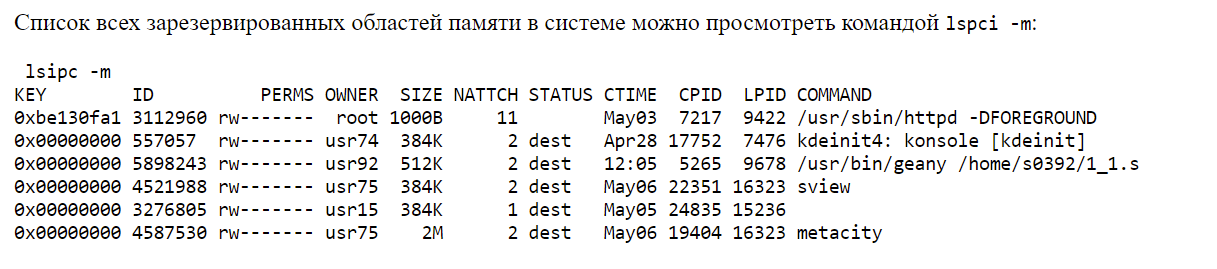
Ответ: выведем таблицу символов для собранного файла(но не для запущенного процесса, такой опции я не нашел



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.

**7.Вопрос** определите хотя бы один запрещённый диапазон адресов.

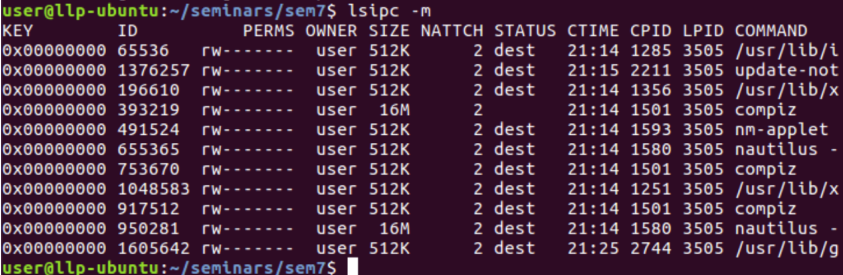
Ответ я нашла тут: [Память процесса (uran.ru)](https://parallel.uran.ru/book/export/html/536)



Введём команду lsipc -m и должны получить сразу множество зарезервированных областей, под которые память выделиться уже не может.

Однако, у меня таких областей нет, но они есть у Жеки😉)

****

****

**8.Вопрос** Что такое inode в файловой системе?

inode - это файловый дескриптор или метаданные, то есть данные о данных, отвечают за много всего, реализуют организацию файловой системы.

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)...

**9.Вопрос** Что находится в остальных столбцах? Прочитайте про файл /proc/PID/maps в man procfs.

Функция **mmap** отражает *length* байтов, начиная со смещения *offset*файла (или другого объекта), определенного файловым описателем *fd*, в память, начиная с адреса *start*. Последний параметр (адрес) необязателен, и обычно бывает равен 0. Настоящее местоположение отраженных данных возвращается самой функцией **mmap**, и никогда не бывает равным 0.

Void \*mmap(void \*addr, size\_t length, int prot, int flags, int fd, off\_t offset);

Данная функция возвращает адрес, по которому был замаплен файл

\*addr- предпочтительный адрес виртуальной памяти, по которому будет начат мапинг(предпочтительный потому что если у системы что-то не получитс, то она сам выберет адрес)

Length-длина размера замапленной области в байтах

Аргумент Prot- желаемый режим защиты памяти (он не должен конфликтовать с режимом открытия файла). //Оно является либо **PROT\_NONE** либо побитовым ИЛИ одного или нескольких флагов PROT\_\*.

**PROT\_EXEC**

(данные в страницах могут исполняться);

**PROT\_READ**

(данные можно читать);

**PROT\_WRITE**

(в эту область можно записывать информацию);

**PROT\_NONE**

(доступ к этой области памяти запрещен).

Параметр *flags* задает тип отражаемого объекта, опции отражения и указывает, принадлежат ли отраженные данные только этому процессу или их могут читать другие.

(Режим видимости изменений другим процессам на том же регионе).

Он состоит из комбинации следующих битов:

**MAP\_FIXED**

Не использовать другой адрес, если адрес задан в параметрах функции. Если заданный адрес не может быть использован, то функция **mmap** вернет сообщение об ошибке. Если используется MAP\_FIXED, то *start* должен быть пропорционален размеру страницы. Использование этой опции не рекомендуется.

**MAP\_SHARED**

Разделить использование этого отражения с другими процессами, отражающими тот же объект. Запись информации в эту область памяти будет эквивалентна записи в файл. Файл может не обновляться до вызова функций **[msync](https://www.opennet.ru/cgi-bin/opennet/man.cgi?topic=msync&category=2)**(2) или **[munmap](https://www.opennet.ru/cgi-bin/opennet/man.cgi?topic=munmap&category=2)**(2)**.**

**MAP\_PRIVATE**

Создать неразделяемое отражение с механизмом copy-on-write. Запись в эту область памяти не влияет на файл. Не определено, являются или нет изменения в файле после вызова **mmap** видимыми в отраженном диапазоне.

Вы должны задать либо MAP\_SHARED, либо MAP\_PRIVATE.

Эти три флага описаны в POSIX.1b (бывшем POSIX.4) and SUSv2. В Linux также анализируются следующие нестандартные флаги:

**MAP\_DENYWRITE**

Этот флаг игнорируется. (Раньше он обозначал, что попытки записи в подчиненные файлы должны завершаться с кодом ошибки ETXTBUSY. Но это стало основой для атак типа 'отказ-в-доступе' - 'denial-of-service'.)

**MAP\_EXECUTABLE**

Этот флаг игнорируется.

**MAP\_NORESERVE**

(Используется вместе с MAP\_PRIVATE.) Не выделяет страницы пространства подкачки для этого отображения. Если пространство подкачки выделяется, то это частное пространство копирования-при-записи может быть изменено. Если оно не выделено, то можно получить SIGSEGV при записи и отсутствии доступной памяти.

**MAP\_LOCKED**

(Linux 2.5.37 и выше) Блокировать страницу или размеченную область в памяти так, как это делает **mlock()**. Этот флаг игнорируется в старых ядрах.

**MAP\_GROWSDOWN**

Используется для стеков. Для VM системы ядра обозначает, что отображение должно распространяться вниз по памяти.

**MAP\_ANONYMOUS**

Отображение не резервируется ни в каком файле; аргументы *fd* и *offset* игнорируются. Этот флаг вместе с MAP\_SHARED реализован с Linux 2.4.

**MAP\_ANON**

Псевдоним для MAP\_ANONYMOUS. Не используется.

**MAP\_FILE**

Флаг совместимости. Игнорируется.

**10.Вопрос** Создайте файл hello.txt с текстом Hello, mmap!. Используя заготовку, отобразите его в память и выведите текст из него.

%define O\_RDONLY 0

%define PROT\_READ 0x1

%define MAP\_PRIVATE 0x2

%define SYS\_OPEN 2

%define SYS\_MMAP 9

%define STR\_LEN 12