Статья по теме “mmap – менеджмент памяти и отображение диска в раму”.

Выполнена Андреевым Артёмом ИУ5, PREP-11 осень 2017

Что такое mmap?

mmap() – системный вызов UNIX-систем, который отображает файловый дескриптор процесса или устройства в виртуальную память. Функция расположена в заголовочном файле менеджмента памяти sys/mman.h.

Аргументы функции, флаги.

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

Прототип функции munmap():
int munmap(void *addr, size_t length);

mmap() возвращает указатель на адрес начала созданного отображения. При ошибке возвращается MAP_FAILURE ((void *)-1) и в глобальную переменную errno записывается код ошибки.

void *addr – начальный адрес для отображения (на Linux-системах будет выбрана ближайшая свободная страница памяти). Если адрес NULL, то система сама выберет адрес для создания отображения (это самый переносимый способ создания отображения под разные системы).

size_t length – количество байт, которые нужно вычитать из файла, на который ссылается файловый дескриптор int fd, начиная от смещения off_t offset. Если length выходит за границы файла, то вышедшая память будет отображена нулями. length должен быть не равен 0.

int prot – флаги доступа к отображаемой памяти (не должны конфликтовать с правами доступа процесса к файлу), выставляется PROT_NONE или побитовое ИЛИ одного или нескольких флагов:

- PROT_EXEC – страницы памяти могут быть запущены как исполняемые;
- PROT_READ – страницы памяти доступны для чтения;
- PROT_WRITE – страницы доступны для записи;
- PROT_NONE – доступ к страницам запрещен.

int flags – флаги, определяющие тип, атрибуты области отображения, то есть как будет выполняться отображение файла:

- MAP_FIXED – отображение будет выполнено точно в address длиной length. Если область недоступна, также возвращается MAP_FAILURE. Если область памяти, заданная с помощью address и length, перекрывает уже существующее отображение, то перекрытая часть будет удалена и отдана под новое отображение. Использование не рекомендуется из-за низкой переносимости на другие системы, так как реализация системы не требует обязательного включения этого флага;
- MAP_SHARED – разделяет область отображения между процессами. Все изменения в отображении применяются сразу к файлу и видны другим процессам. Файл может быть обновлен не сразу, рекомендуется синхронизация при помощи системного вызова msync(2), munmap() также запишет все изменения в файл;
- MAP_PRIVATE – отображение не видно другим процессам, изменения в нем тоже, изменения не могут быть записаны в файл.

Эти флаги описаны в стандарте POSIX.1-2001 и POSIX.1-2008, обязательна реализация в системах только MAP_SHARED и MAP_PRIVATE. В дополнение к одному из этих флагов, который должен быть выбран обязательно, можно использовать побитовое ИЛИ одного или нескольких следующих флагов:

- MAP_32BIT (начиная с версии ядра Linux 2.4.20, 2.6) – располагает отображение в первых 2 ГБ адресного пространства вызывающего процесса. Этот флаг поддерживается только x86-64 архитектурами (64-разрядными) и 64-битными программами. Был необходим для повышения производительности первых 64-разрядных процессоров, сейчас необходимость в нем отпала. Игнорируется, если установлен MAP_FIXED;
- MAP_ANONYMOUS – созданное отображение не связано с каким-либо файлом, все его содержимое проинициализировано нулями, аргументы fd и offset игнорируются. Некоторые реализации требуют fd, равный -1, и/или offset, равный 0, необходимо учитывать это для переносимости на другие системы. Использование вместе с MAP_SHARED поддерживается Linux-ядром, начиная с версии 2.4;
- MAP_ANON – синоним MAP_ANONYMOUS, устарел;
- MAP_DENYWRITE – флаг игнорируется (раньше служил для того, чтобы попытка записи в файл выдавала ошибку ETXTBUSY, но это использовалось для совершения DoS-атак);
- MAP_EXECUTABLE – этот флаг игнорируется;
- MAP_FILE – флаг для совместимости, игнорируется, отображение по умолчанию из файла;
- MAP_GROWSDOWN – используется для стеков, указывает системе виртуальной памяти, что отображение должно продолжаться вниз по

памяти. Возвращаемое значение при этом указывает на страницу виртуальной памяти, которая находится ниже, чем созданное фактически отображение в виртуальном пространстве. Каждое обращение по адресу приведет к наращению виртуального пространство на еще одну страницу до тех пор, пока не будет достигнуто следующее отображение, при этом испускается сигнал SIGSEGV;

- MAP_HUGETLB (начиная с версии ядра Linux 2.6.32) – выделить память под отображение на страницах кучи;
- MAP_HUGE_2MB, MAP_HUGE_1GB (начиная с версии ядра Linux 3.8) – используется вместе с MAP_HUGETLB для выбора альтернативного размера страницы кучи (2 Мб или 1 Гб соответственно) на системах, которые поддерживают кратные размеры страниц кучи;
- MAP_LOCKED (начиная с версии ядра Linux 2.5.37) – флаг, указывающий на то, что страницы отображения, находящиеся в виртуальной памяти, должны быть заблокированы в оперативной памяти, препятствуя перемещению страниц в файл подкачки (swap) (в стиле mlock(2)). Игнорируется старыми ядрами;
- MAP_NONBLOCK (начиная с версии ядра Linux 2.5.46) – флаг используется вместе с MAP_POPULATE, указывает на то, что система не должна выполнять чтение наперед, то есть должна создавать таблицы страниц виртуальной памяти только для тех страниц, которые фактически находятся в оперативной памяти. Начиная с версии Linux 2.6.23 флаг приводит к тому, что MAP_POPULATE ничего не делает.
- MAP_NORESERVE – не резервировать память в подкачке (swap) для отображения. Когда место зарезервировано, гарантируется возможность изменять отображение. Если нет, то испускается сигнал SIGSEGV во время записи, когда место в физической оперативной памяти заканчивается. В версиях ядра Linux ниже 2.6 работает только для приватных (MAP_PRIVATE) допускающих запись отображений;
- MAP_POPULATE (начиная с версии ядра Linux 2.5.46) – заполнить страницы виртуальной памяти для отображения, приводит к чтению наперед. Это помогает в дальнейшем избежать блокировок памяти при ошибках страниц. MAP_POPULATE работает для приватных (MAP_PRIVATE) отображений только начиная с версии ядра Linux 2.6.23;
- MAP_STACK (начиная с версии ядра Linux 2.6.27) – выделить память под отображение по адресу, соответствующему процессу или стеку потоков. Этот флаг в настоящее время не работает, но используется в потоковой реализации в glibc, так что если некоторым архитектурным решениям понадобится особый способ выделения памяти под стек, поддержка может быть добавлена в glibc.
- MAP_UNINITIALIZED (начиная с версии ядра Linux 2.6.33) – не очищать анонимные (MAP_ANONYMOUS) страницы отображения. Служит для

повышения производительности встраиваемых систем, работает только для ядер, настроенных с параметром CONFIG_MMAP_ALLOW_UNINITIALIZED. По соображениям безопасности, этот параметр включен только на встраиваемых системах (то есть на системах, которые имеют полный контроль над памятью пользователей).

Зачастую в системах реализованы помимо обязательных MAP_SHARED и MAP_PRIVATE MAP_FIXED и MAP_ANONYMOUS (MAP_ANON).

int fd – валидный файловый дескриптор процесса, содержимое которого будет отображаться в виртуальную память (при условии, что отображение не анонимное (MAP_ANONYMOUS) – описано выше). mmap() добавляет дополнительную ссылку на файл, которая не удаляется при закрытии (системный вызов close()) соответствующего файлового дескриптора. Ссылка снимается, когда больше нет ни одного отображения этого файла.

off_t offset – смещение – определяет, откуда начнется отображение длиной length. offset должно быть кратно размеру страницы памяти, определяемого возвращаемым значением вызова sysconf(_SC_PAGE_SIZE).

munmap() используется для удаления созданных при помощи mmap() отображений. Дальнейшие обращения к адресу отображения не валидны. Память также автоматически освобождается, когда процесс завершается. Отображение не удаляется, когда закрывается (системный вызов close()) файловый дескриптор (как было сказано выше). При успешном выполнении возвращается 0, при ошибке – -1, в errno записывается код ошибки.

void *addr – адрес созданного отображения, должен быть кратен размеру страницы памяти (возвращаемое значение mmap() кратно). Последующие обращения по освобожденному адресу приводят к испусканию сигнала SIGSEGV. munmap() не генерирует ошибку, если адрес не содержит страниц отображения.

size_t length – длина удаляемого отображения.

Как происходит отображение, чтение или запись в память?

mmap() является методом ввода-вывода через отображение файла на виртуальную память.

Из Википедии:

Виртуальная память (англ. virtual memory) — метод управления памятью компьютера, позволяющий выполнять программы, требующие больше оперативной памяти, чем имеется в компьютере, путём автоматического перемещения частей программы между основной памятью и вторичным хранилищем (например, жёстким диском). В системе с виртуальной памятью используемые программами адреса, называемые виртуальными адресами, транслируются в физические адреса в памяти компьютера. Управление виртуальными адресными пространствами, соотнесение физической и виртуальной памяти, а также перемещение фрагментов памяти между основным и вторичным хранилищами выполняет операционная система (см. подкачка страниц).
Более подробно принцип работы виртуальной памяти, страниц можно узнать из других источников.

mmap() реализует фактически подкачку по обращению, то есть содержимое файла не вычитывается сразу полностью и не использует оперативную память. Чтение с диска происходит в “ленивом” режиме (“ленивое” чтение), то есть при обращении программой (процессом) к определенной части файла вычитывается часть файла блоком, равным размеру страницы виртуальной памяти, в основную память (оперативную или в подкачку (swap)) процесса. При этом чтение из адресов памяти приводит к фактическому чтению из файла, а запись в них – запись в файл.

Файл отображается в размере, кратном размеру страницы виртуальной памяти. Для файла, размер которого не кратен размеру страницы, оставшаяся память будет заполнена нулями, и запись в этот участок не будет приводить к изменению файла.

Преимущества и недостатки использования.

Преимущества:

1. Выигрыш в производительности и быстродействие (основная причина):
   Для обычного чтения/записи из/в файл:
   a. нужно помнить текущую позицию в файле, передвигать ее;
   b. каждый вызов смены/чтения позиции – системный, приводит к трате времени;
   c. нужно выделять буферы определенного размера, то есть работа состоит из трех этапов: чтение в буфер, модификация данных в буфере, запись в файл. При отображении модифицируются данные файла сразу в определенной области.

   “Ленивое” чтение – меньшая нагрузка на систему благодаря загрузке в память лишь частей файла и связанное с этим быстродействие.

Имея меньшее количество памяти, можно легко отобразить больший по размеру файл, при этом нагрузка на систему не приведет к большим накладным расходам. При обновлении большого массива данных они могут быть одновременно записаны на диск (за один проход головки). “Ленивое” чтение также обеспечивает более быструю загрузку процесса.

2. возможность легкого изменения размера файла: можно достаточно легко менять его размер и при этом (после переотображения) получать в своё распоряжение непрерывный кусок памяти нужного размера (в отличие от динамической памяти).
3. возможность разделения одной и той же памяти между процессами, отображающими данный файл (при этом у каждого процесса выделенная память отдельная);
4. забывание отображенных страниц без выгрузки в область подкачки, обязательной для выделенной памяти;
5. не использование дискового кэша;
6. снижение стоимости систем (например, смартфонов) из-за отпада необходимости в использовании большого количества основной (оперативной) памяти, которая дороже вторичной.

Недостатки:

1. Размер отображения всегда кратен размеру страницы памяти, это может привести к лишним затратам;
2. некоторые программы сталкиваются с дополнительной задержкой при первом доступе к странице;
3. использование отображений чревато замедлениями из-за страничных ошибок доступа (в частности, чистка мусора (trashing) при повторяющихся ошибках страницы);
4. управление памятью с алгоритмами замещения страниц становится несколько более сложным;
5. размер отображения зависит от архитектуры (32-битные не могут создавать отображения длиной более 4 Гб (1 страница = 4096 байт);
6. не поддерживается всеми системами, особенно некоторыми дешевыми, энергоэффективными встраиваемыми решениями, которые не имеют блок управления памятью;
7. возможные уязвимости, в частности, к атакам по времени (timing attacks).

Применение:

1. загрузка процесса в память (и для Microsoft Windows, и для Unix-подобных систем);
2. создание разделяемых несколькими процессами фрагментов памяти. В современных ОС (использующих защищенный режим) процесс не

позволяет другим процессам обращаться к своей памяти, поэтому отображение является наиболее популярным и безопасным методом сделать память доступным нескольким приложениям.

3. загрузка исполняемых модулей и динамических библиотек;
4. используется утилитой GNU grep;
5. загрузка шрифтов в графических подсистемах.