Разделяемая память Posix IPC

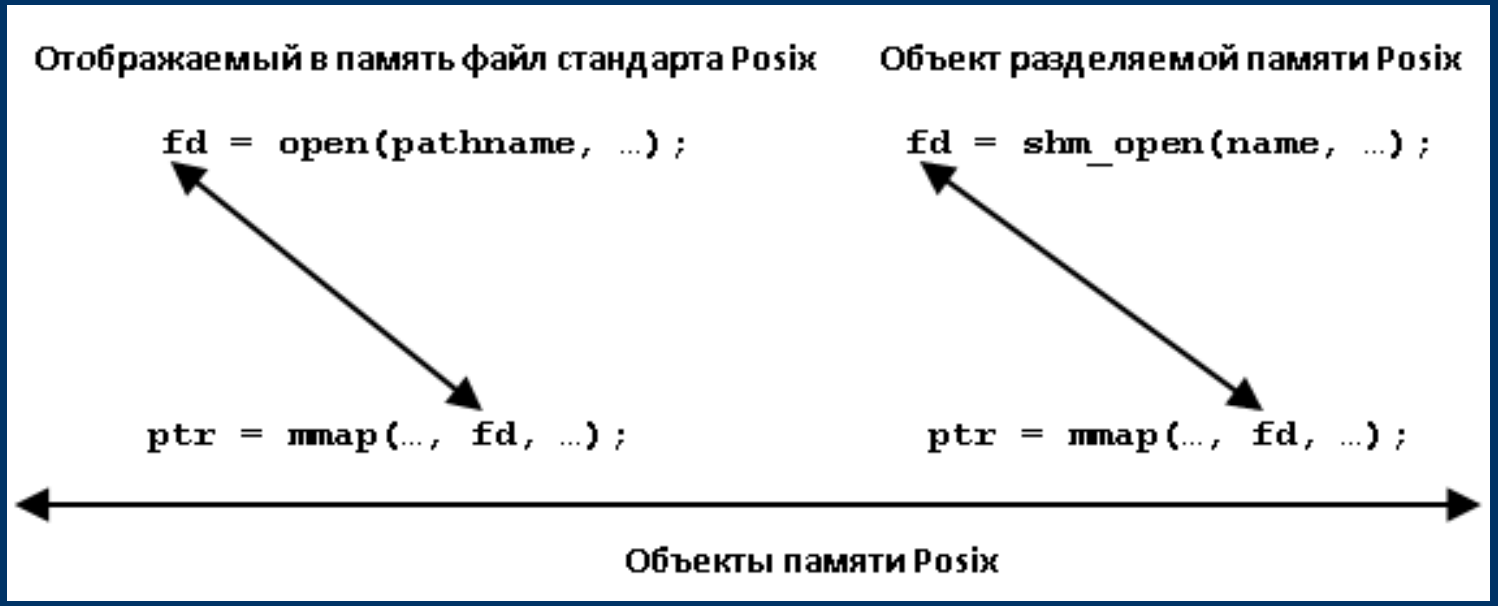
- Введение
- Функции mmap, munmap и msync
- Функции shm_open и shm_unlink
- Функции ftruncate и fstat
- Простые программы
- Ограничения на разделяемую память

Введение

- Стандарт Posix.1 предоставляет два механизма совместного использования областей памяти для неродственных процессов:
- 1. Отображение файлов в память: файл открывается вызовом **open**, а его дескриптор используется при вызове функции **mmap** для отображения содержимого файла в адресное пространство процесса. Этот метод позволяет реализовать совместное использование памяти как для родственных, так и для неродственных процессов.
- 2. Объекты разделяемой памяти: функция **shm_open** открывает объект IPC с именем стандарта Posix (например, полным именем объекта файловой системы), возвращая дескриптор, который может быть использован для отображения в адресное пространство процесса вызовом **mmap**.

Введение

Оба метода требуют вызова **mmap**. Отличие состоит в методе получения дескриптора, являющегося аргументом **mmap**: в первом случае он возвращается функцией **open**, а во втором — **shm_open**. Это иллюстрирует рисунок ниже. Стандарт Posix называет объектами памяти (memory objects) и отображенные в память файлы, и объекты разделяемой памяти стандарта Posix.



Функции mmap, munmap и msync

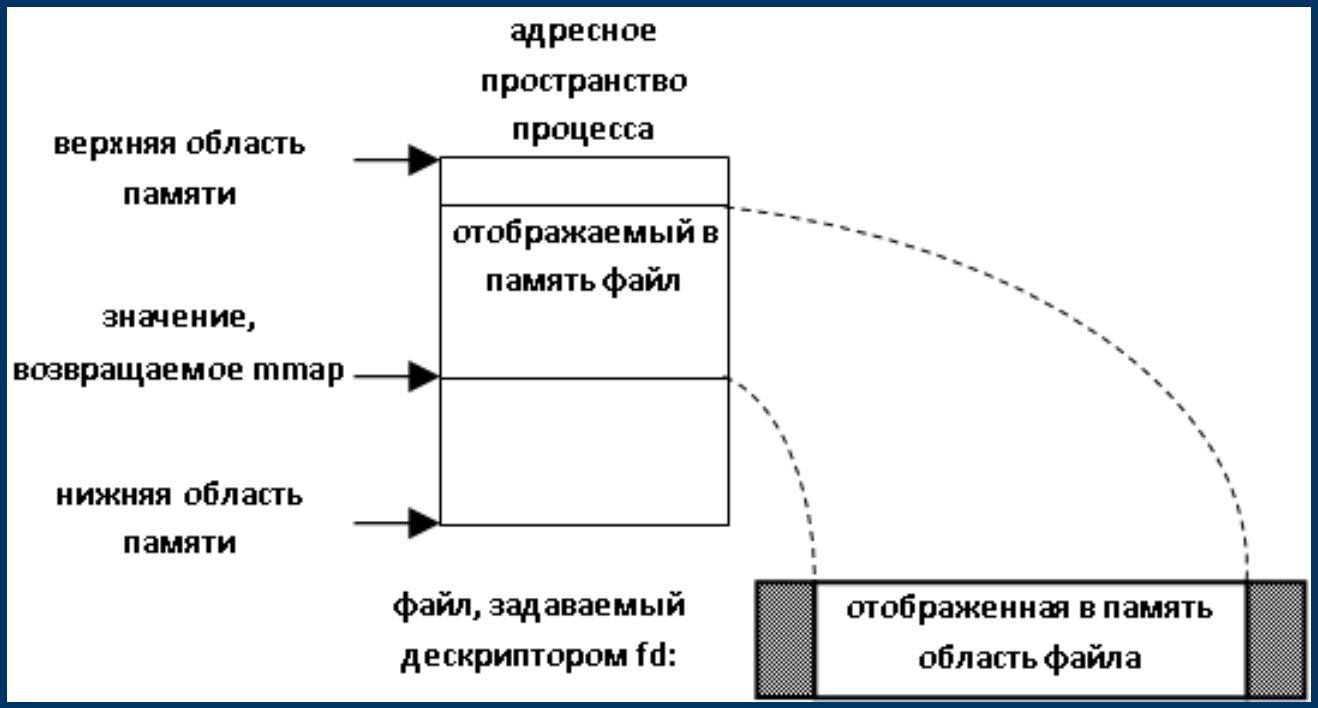
- Поскольку оба подхода требуют использования функций для работы с отображаемыми на память файлами, вначале рассмотрим их.
- Функция **mmap** отображает в адресное пространство процесса файл или объект разделяемой памяти Posix. Мы используем эту функцию в следующих ситуациях:
- 1. С обычными файлами для обеспечения ввода-вывода через отображение в память.
- 2. Со специальными файлами для обеспечения неименованного отображения памяти (несуществующие файлы, как в Windows API и файл **/dev/zero**) , но такие файлы поддерживаются только в ОС семейства BSD.
- 3. С функцией **shm_open** для создания участка разделяемой неродственными процессами памяти Posix.

Функции mmap, munmap и msync

- **#include <sys/mman.h>**
- **void *mmap(void *addr, size_t len, int prot, int flags, int fd, off_t offset);**
- Функция возвращает начальный адрес участка памяти в случае успешного завершения или **MAP_FAILED** в случае ошибки.
- Аргумент **addr** может указывать начальный адрес участка памяти процесса, в который следует отобразить содержимое дескриптора **fd**. Обычно ему присваивается значение нулевого указателя, что говорит ядру о необходимости выбрать начальный адрес самостоятельно. В любом случае функция возвращает начальный адрес сегмента памяти, выделенной для отображения.

Функции mmap, munmap и msync

- Аргумент **len** задает длину отображаемого участка в байтах; участок может начинаться не с начала файла, а с некоторого места, задаваемого аргументом **offset**. Обычно **offset** = 0. На рисунке изображена схема отображения:



Функции mmap, munmap и msync

- Защита участка памяти с отображенным объектом обеспечивается с помощью аргумента **prot** и констант. Обычное значение этого аргумента — **PROT_READ | PROT_WRITE**, что обеспечивает доступ на чтение и запись.
- Аргумент **flags** может принимать три значения. Можно указать только один из флагов — **MAP_SHARED** или **MAP_PRIVATE**, прибавив к нему при необходимости **MAP_FIXED**. Если указан флаг **MAP_PRIVATE**, все изменения будут производиться только с образом объекта в адресном пространстве процесса; другим процессам они доступны не будут. Если же указан флаг **MAP_SHARED**, изменения отображаемых данных видны всем процессам, совместно использующим объект.

Функции mmap, munmap и msync

- Для обеспечения переносимости программ флаг **MAP_FIXED** указывать не следует. Если он не указан, но аргумент **addr** представляет собой ненулевой указатель, интерпретация этого аргумента зависит от реализации. В переносимой программе значение **addr** должно быть нулевым и флаг **MAP_FIXED** не должен быть указан.
- Одним из способов добиться совместного использования памяти родительским и дочерним процессами является вызов **mmap** с флагом **MAP_SHARED** перед вызовом **fork**. Стандарт Posix.1 гарантирует в этом случае, что все отображения памяти, установленные родительским процессом, будут унаследованы дочерним. Изменения в содержимом объекта, вносимые родительским процессом, будут видны дочернему процессу, и наоборот.

Функции mmap, munmap и msync

- Для отключения отображения объекта в адресное пространство процесса используется вызов **munmap**:
- **#include <sys/mman.h>**
- **int munmap(void *addr, size_t len);**
- Функция возвращает 0 в случае успешного завершения или –1 в случае ошибки.
- Аргумент **addr** должен содержать адрес, возвращенный **mmap**, а **len** — длину области отображения. После вызова **munmap** любые попытки обратиться к этой области памяти приведут к отправке процессу сигнала **SIGSEGV** (предполагается, что эта область памяти не будет снова отображена вызовом **mmap**).

Функции mmap, munmap и msync

- Если область была отображена с флагом **MAP_PRIVATE**, все внесенные за время работы процесса изменения сбрасываются.
- В изображенной на рисунке выше схеме ядро обеспечивает синхронизацию содержимого файла, отображенного в память, с самой памятью при помощи алгоритма работы с виртуальной памятью (если сегмент был отображен с флагом **MAP_SHARED**).
- Если происходит изменение содержимого ячейки памяти, в которую отображен файл, через некоторое время содержимое файла будет соответствующим образом изменено ядром. Однако в некоторых случаях необходимо, чтобы содержимое файла всегда было в соответствии с содержимым памяти. Тогда для осуществления моментальной синхронизации вызывают функцию **msync**:

Функции mmap, munmap и msync

- **#include <sys/mman.h>**
- **int msync(void *addr, size_t len , int flags);**
- Функция возвращает 0 в случае успешного завершения или –1 в случае ошибки.
- В аргументе **flags** Из двух констант **MS_ASYNC** и **MS_SYNC** указать нужно одну и только одну. Отличие между ними в том, что возврат из функции при указании флага **MS_ASYNC** происходит сразу же, как только данные для записи будут помещены в очередь ядром, а при указании флага **MS_SYNC** возврат происходит только после завершения операций записи. Если указан и флаг **MS_INVALIDATE**, все копии файла, содержимое которых не совпадает с его текущим содержимым, считаются устаревшими. Последующие обращения к этим копиям приведут к считыванию данных из файла.

Функции mmap, munmap и msync

- Удобство работы с отображением в память содержимого файла состоит в том, что все операции ввода-вывода осуществляются ядром и скрыты от программиста, а он просто пишет код, считывающий и записывающий данные в некоторую область памяти. Ему не приходится вызывать **read**, **write** или **lseek**. Часто это заметно упрощает код.
- Следует, однако, иметь в виду, что не все файлы могут быть отображены в память. Попытка отобразить дескриптор, указывающий на терминал или сокет, приведет к возвращению ошибки при вызове **mmap**. К дескрипторам этих типов доступ осуществляется только с помощью **read** и **write** (и аналогичных вызовов).

Функции mmap, munmap и msync

- Другой целью использования **mmap** может являться разделение памяти между неродственными процессами. В этом случае содержимое файла становится начальным содержимым разделяемой памяти и любые изменения, вносимые в нее процессами, копируются обратно в файл (что дает этому виду IPC живучесть файловой системы). Предполагается, что при вызове **mmap** указывается флаг **MAP_SHARED**, необходимый для разделения памяти между процессами.

Функции shm_open и shm_unlink

- Процесс получения доступа к объекту разделяемой памяти Posix выполняется в два этапа:
- 1. Вызов **shm_open** с именем IPC в качестве аргумента позволяет либо создать новый объект разделяемой памяти, либо открыть существующий.
- 2. Вызов **mmap** позволяет отобразить разделяемую память в адресное пространство вызвавшего процесса.
- Аргумент **name**, указанный при первом вызове **shm_open**, должен впоследствии использоваться всеми прочими процессами, желающими получить доступ к данной области памяти.

Функции shm_open и shm_unlink

- Причина, по которой этот процесс выполняется в два этапа вместо одного, на котором в ответ на имя объекта возвращался бы адрес соответствующей области памяти, заключается в том, что функция **mmap** уже существовала, когда эта форма разделяемой памяти была включена в стандарт Posix.
- **#include <sys/mman.h>**
- **int shm_open(const char *name, int oflag, mode_t mode);**
- Функция возвращает неотрицательный дескриптор в случае успешного завершения или -1 в случае ошибки.

Функции shm_open и shm_unlink

- **int shm_unlink(const char *name);**
- Функция возвращает 0 в случае успешного завершения или -1 в случае ошибки.
- Аргумент **oflag** должен содержать флаг **O_RDONLY** либо **O_RDWR** и один из следующих: **O_CREAT**, **O_EXCL**, **O_TRUNC**. Флаги **O_CREAT** и **O_EXCL** были описаны ранее. Если вместе с флагом **O_RDWR** указан флаг **O_TRUNC**, существующий объект разделяемой памяти будет укорочен до нулевой длины.
- Аргумент **mode** задает биты разрешений доступа (указаны выше) и используется только при указании флага **O_CREAT**. Обратите внимание, что в отличие от функций **mq_open** и **sem_open** для **shm_open** аргумент **mode** указывается всегда.

Функции shm_open и shm_unlink

- Если флаг **O_CREAT** не указан, значение аргумента **mode** может быть нулевым.
- Возвращаемое значение **shm_open** представляет собой целочисленный дескриптор, который может использоваться при вызове **mmap** в качестве пятого аргумента.
- Функция **shm_unlink** удаляет имя объекта разделяемой памяти. Как и другие подобные функции (удаление файла из файловой системы, удаление очереди сообщений и именованного семафора Posix), она не выполняет никаких действий до тех пор, пока объект не будет закрыт всеми открывшими его процессами. Однако после вызова **shm_unlink** последующие вызовы **open**, **mq_open** и **sem_open** выполняться не будут.

Функции ftruncate и fstat

- Размер файла или объекта разделяемой памяти можно изменить вызовом **ftruncate**:
- **#include <unistd.h>**
- **int ftruncate(int fd, off_t length);**
- Функция возвращает 0 в случае успешного завершения или –1 в случае ошибки.
- Стандарт Posix делает некоторые различия в определении действия этой функции для обычных файлов и для объектов разделяемой памяти.

Функции ftruncate и fstat

- 1. Для обычного файла: если размер файла превышает значение **length**, избыточные данные отбрасываются. Если размер файла оказывается меньше значения **length**, действие функции не определено. Поэтому для переносимости следует использовать следующий способ увеличения длины обычного файла: вызов **seek** со сдвигом **length-1** и запись 1 байта в файл. К счастью, почти все реализации Unix поддерживают увеличение размера файла вызовом **ftruncate**.
- 2. Для объекта разделяемой памяти **ftruncate** устанавливает размер объекта равным значению аргумента **length**.

Функции ftruncate и fstat

- Итак, **ftruncate** вызывается для установки размера только что созданного объекта разделяемой памяти или изменения размера существующего объекта. При открытии существующего объекта разделяемой памяти следует воспользоваться **fstat** для получения информации о нем:
- **#include <sys/types.h>**
- **#include <sys/stat.h>**
- **int fstat(int fd, struct stat *buf);**
- Функция возвращает 0 в случае успешного завершения или –1 в случае ошибки.
- В структуре **stat** содержится больше десятка полей, но только четыре из них содержат актуальную информацию, если **fd** представляет собой дескриптор области разделяемой памяти:

Функции ftruncate и fstat

- **struct stat {**
- **...**
- **mode_t st_mode; /* mode: S_I{RW}{USR,GRP,OTH} */**
- **uid_t st_uid; /* UID владельца */**
- **gid_t st_gid; /* GID владельца */**
- **off_t st_size; /* размер в байтах */**
- **...**
- **};**
- Пример использования этих двух функций приведен ниже.

Простые программы

- Приведем несколько примеров программ, работающих с разделяемой памятью Posix.
- Программа shmcreate.c, текст которой доступен на сайте в разделе «PosixShm» создает объект разделяемой памяти с указанным именем и длиной. Вызов **shm_open** создает объект разделяемой памяти. Если указан параметр **–e**, будет возвращена ошибка в том случае, если такой объект уже существует. Вызов **ftruncate** устанавливает длину (размер объекта), а **mmap** отображает его содержимое в адресное пространство процесса. Затем программа завершает работу. Поскольку разделяемая память Posix обладает живучестью ядра, объект разделяемой памяти при этом не исчезает.

Простые программы

- В файле shmunlink.c приведен текст тривиальной программы, удаляющей имя объекта разделяемой памяти из системы.
- В файле shmwrite.c приведен текст программы, записывающей последовательность 0, 1, 2, 254, 255, 0, 1 и т. д. в объект разделяемой памяти. Объект разделяемой памяти открывается вызовом **shm_open**. Его размер можно узнать с помощью **fstat**. Затем файл отображается в память вызовом **mmap**, после чего его дескриптор может быть закрыт.
- Программа в файле shmread.c проверяет значения, помещенные в разделяемую память программой shmwrite. Объект разделяемой памяти открывается только для чтения, его размер получается вызовом **fstat**, после чего он отображается в память с доступом только на чтение, а дескриптор закрывается.

Простые программы

- Далее рассмотрим работу этих примеров.
- Создадим объект разделяемой памяти с именем myshm объемом 123 456 байт в системе CentOS 6.9:
- **[gun@CentOS]$ ./shmcreate /temp 123456**
- **Result:Success**
- **[gun@CentOS]$ ls -l /dev/shm/temp**
- **-rw-------. 1 gun gun 123456 Фев 3 18:13 /dev/shm/temp**
- **[gun@CentOS]$ od /dev/shm/temp**
- **0000000 000000 000000 000000 000000 000000 000000 000000 000000**
- *****
- **0361100**

Простые программы

- Здесь видно, что файл с указываемым при создании объекта разделяемой памяти именем появляется в виртуальной файловой системе, там же, где и семафоры. Соответственно, объекты можно просматривать стандартными утилитами **ls** и **cat**. Используя программу **od**, можно выяснить, что после создания файл целиком заполнен нулями (восьмеричное число 0361100 — сдвиг, соответствующий байту, следующему за последним байтом файла, — эквивалентно десятичному 123 456).
- Запустим программу shmwrite и убедимся в правильности записываемых значений с помощью программы **od**:

Простые программы

- **[gun@CentOS]$ ./shmwrite /temp**
- **[gun@CentOS]$ od -x /dev/shm/temp |head -3**
- **0000000 0100 0302 0504 0706 0908 0b0a 0d0c 0f0e**
- **0000020 1110 1312 1514 1716 1918 1b1a 1d1c 1f1e**
- **0000040 2120 2322 2524 2726 2928 2b2a 2d2c 2f2e**
- Проверим содержимое разделяемой памяти с помощью программы shmread:
- **[gun@CentOS]$ ./shmread /temp**
- Ошибок не обнаружено. Удалим объект, запустив программу shmunlink:
- **[gun@CentOS]$ ./shmunlink /temp**
- **Result:Success**

Простые программы

- Убедимся визуально, что объект удален:
- **[gun@CentOS]$ ls -l /dev/shm/temp**
- **ls: невозможно получить доступ к /dev/shm/temp: Нет такого файла или каталога**

Ограничения на разделяемую память

- Ограничения на разделяемую память можно определить, просматривая виртуальную файловую систему:
- *максимальное количество объектов*
- **[gun@CentOS]# cat /proc/sys/kernel/shmmni**
- **4096**
- *максимальный размер сегмента*
- **[gun@CentOS]# cat /proc/sys/kernel/shmmax**
- **4294967295**
- **[gun@CentOS]# cat /proc/sys/kernel/shmall**
- **268435456**