Семафоры Posix IPC

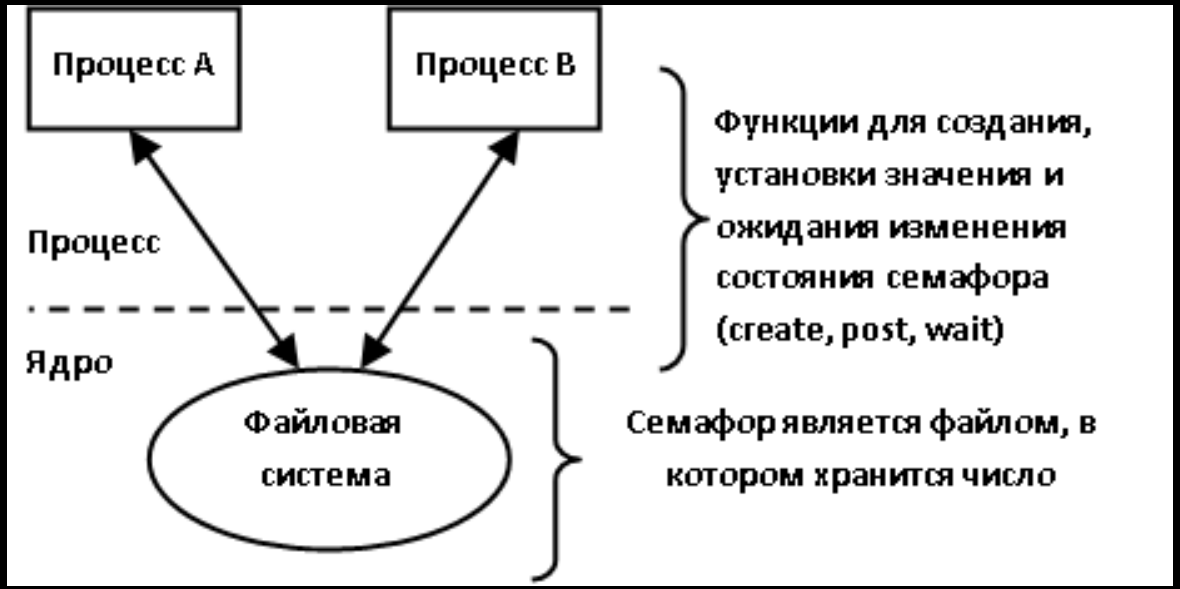
- **Введение**
- **Функции sem_open, sem_close и sem_unlink**
- **Функции sem_wait и sem_trywait**
- **Функции sem_post и sem_getvalue**
- **Простые примеры**
- **Функции sem_init и sem_destroy**
- **Ограничения на семафоры**

Введение

- Семафоры Posix (в отличие от семафоров System V IPC) не обязательно должны храниться в ядре. Их особенностью является наличие имен, которые могут соответствовать именам реальных файлов в файловой системе. Далее изображена схема, иллюстрирующая именованный семафор Posix.
- Заметим, что хотя именованные семафоры Posix обладают именами в файловой системе, они не обязательно должны храниться в файлах. Во встроенной операционной системе реального времени значение семафора, скорее всего, будет размещаться в ядре, а имя файла будет использоваться исключительно для идентификации семафора.

Введение

- При реализации с помощью отображения файлов в память значение семафора будет действительно храниться в файле, который будет отображаться в адресное пространство всех процессов, использующих семафор.



Введение

- На рис. выше указаны три операции, которые могут быть применены к семафорам:
- 1. Создание семафора (***create***). При этом вызвавший процесс должен указать начальное значение (положительное число, часто 1, но может быть и 0).
- 2. Ожидание изменения значения семафора (***wait***). При этом производится проверка его значения и процесс блокируется, если значение оказывается меньшим либо равным 0, а при превышении 0 значение уменьшается на 1.

Введение

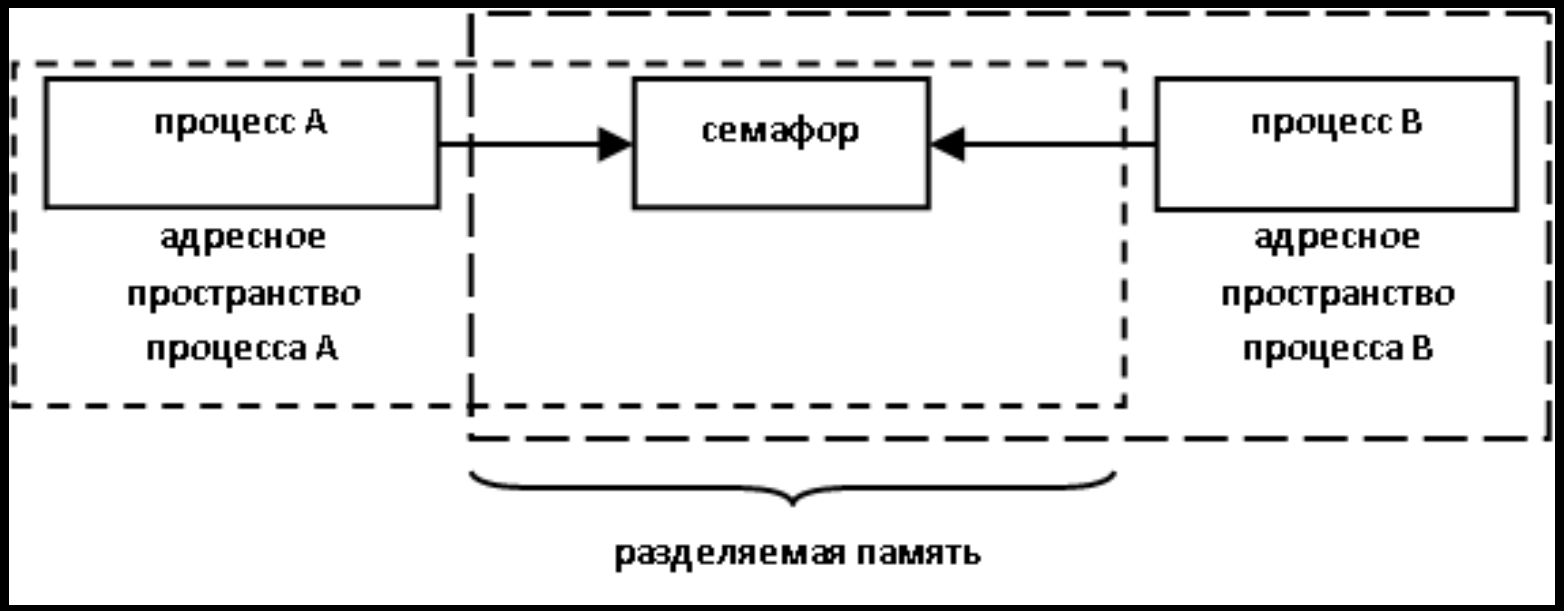
- Основным требованием является атомарность выполнения операций проверки значения и последующего уменьшения значения семафора (то есть как одной операции) по отношению к другим потокам. Это одна из причин, по которой семафоры System V были реализованы в середине 80-х как часть ядра. Операции с ними выполняются с помощью системных вызовов, что легко гарантирует их атомарность по отношению к другим процессам.
- 3. Установка значения семафора (*post*). Значение семафора увеличивается одной командой, и если в системе имеются процессы, ожидающие изменения значения семафора до величины, превосходящей 0, один из них может быть пробужден.

Введение

- Как и операция ожидания, операция установки значения семафора также должна быть атомарной по отношению к другим процессам, работающим с этим семафором.
- Стандартом Posix описано два типа семафоров: именованные (*named*) и размещаемые в памяти (*memory-based* или *unnamed*). Именованный семафор Posix был изображен на рис. выше. Неименованный, или размещаемый в памяти (в том числе разделяемой), семафор, который можно использовать и для синхронизации потоков одного процесса (см. главу 7), изображен на рис. ниже. В случае использования разделяемой памяти общий ее сегмент принадлежит адресному пространству нескольких процессов.

Введение

- Далее сначала рассматриваются именованные семафоры Posix, а затем — размещаемые в памяти.



Функции sem_open, sem_close и sem_unlink

- Функция **sem_open** создает новый именованный семафор или открывает существующий. Именованный семафор может использоваться для синхронизации выполнения потоков и процессов:
- **#include <semaphore.h>**
- **sem_t *sem_open(char *name, int oflag, … /* mode_t mode, unsigned int value */);**
- Функция возвращает указатель на семафор в случае успешного завершения или **SEM_FAILED** — в случае ошибки.

Функции sem_open, sem_close и sem_unlink

- Аргумент **oflag** может принимать значения 0, **O_CREAT**, **O_CREAT | O_EXCL**, как описано ранее. Если указано значение **O_CREAT**, третий и четвертый аргументы функции являются обязательными. Аргумент **mode** указывает биты разрешений доступа (см. табл. 6.11), а **value** указывает начальное значение семафора. Это значение не может превышать константу **SEM_VALUE_MAX**, которая, согласно Posix, должна быть не менее 32767. Бинарные семафоры обычно устанавливаются в 1, тогда как семафоры-счетчики чаще инициализируются бóльшими величинами.

Функции sem_open, sem_close и sem_unlink

- При указании флага **O_CREAT** (без **O_EXCL**) семафор инициализируется только в том случае, если он еще не существует. Если семафор существует, ошибки не возникнет. Ошибка будет возвращена только в том случае, если указаны флаги **O_CREAT | O_EXCL**.
- Возвращаемое значение представляет собой указатель на тип **sem_t**. Этот указатель впоследствии передается в качестве аргумента функциям **sem_close**, **sem_wait**, **sem_trywait**, **sem_post** и **sem_getvalue**.
- Открыв семафор с помощью **sem_open**, можно потом закрыть его, вызвав **sem_close**:

Функции sem_open, sem_close и sem_unlink

- **#include <semaphore.h>**
- **int sem_close(sem_t *sem);**
- Функция возвращает 0 в случае успешного завершения или –1 в случае ошибки.
- Операция закрытия выполняется автоматически при завершении процесса для всех семафоров, которые были им открыты. Автоматическое закрытие осуществляется как при добровольном завершении работы (вызове **exit**), так и при принудительном (с помощью сигнала).

Функции sem_open, sem_close и sem_unlink

- Закрытие семафора не удаляет его из системы. Именованные семафоры Posix обладают по меньшей мере живучестью ядра. Значение семафора сохраняется, даже если ни один процесс не держит его открытым.
- Именованный семафор удаляется из системы вызовом **sem_unlink**:
- **#include <semaphore.h>**
- **int sem_unlink(const char *name);**
- Функция возвращает 0 в случае успешного завершения или –1 в случае ошибки.

Функции sem_open, sem_close и sem_unlink

- Для каждого семафора ведется подсчет процессов, в которых он является открытым, и функция **sem_unlink** действует аналогично **unlink** для файлов. Таким образом, объект **name** может быть удален из файловой системы, даже если он открыт какими-либо процессами, но реальное удаление семафора не будет осуществлено до тех пор, пока он не будет окончательно закрыт.

Функции sem_wait и sem_trywait

- Функция **sem_wait** проверяет значение заданного семафора на положительность, уменьшает его на единицу и немедленно возвращает управление процессу. Если значение семафора при вызове функции равно нулю, процесс приостанавливается, до тех пор, пока оно снова не станет больше нуля, после чего значение семафора будет уменьшено на единицу и произойдет возврат из функции. Операция «проверка и уменьшение» должна быть атомарной по отношению к другим потокам, работающим с этим семафором:
- **#include <semaphore.h>**
- **int sem_wait(sem_t *sem);**
- **int sem_trywait(sem_t *sem);**

Функции sem_wait и sem_trywait

- Обе функции возвращают 0 в случае успешного завершения или –1 в случае ошибки.
- Разница между **sem_wait** и **sem_trywait** заключается в том, что последняя не приостанавливает выполнение процесса, если значение семафора равно нулю, а просто немедленно возвращает ошибку **EAGAIN**.
- Возврат из функции **sem_wait** может произойти преждевременно, если будет получен сигнал. При этом возвращается ошибка с кодом **EINTR**.

Функции sem_post и sem_getvalue

- После завершения работы с семафором процесс (поток) вызывает **sem_post**. Этот вызов увеличивает значение семафора на единицу и возобновляет выполнение любых потоков, ожидающих изменения значения семафора:
- **#include <semaphore.h>**
- **int sem_post(sem_t *sem);**
- **int sem_getvalue(sem_t *sem , int *valp);**
- Обе функции возвращают 0 в случае успешного завершения или –1 в случае ошибки.

Функции sem_post и sem_getvalue

- Функция **sem_getvalue** возвращает текущее значение семафора, помещая его в целочисленную переменную, на которую указывает **valp**. Если семафор заблокирован, возвращается либо 0, либо отрицательное число, модуль которого соответствует количеству потоков (процессов), ожидающих разблокирования семафора.

Простые примеры

- В файле semcreate.c, доступном на сайте в разделе «PosixSem», приведен текст программы, создающей именованный семафор. При вызове программы можно указать параметр **–e**, обеспечивающий исключающее создание (если семафор уже существует, будет выведено сообщение об ошибке), а параметр **–i** с числовым аргументом позволяет задать начальное значение семафора, отличное от 1. В Linux, именованные семафоры создаются в виртуальной файловой системе, обычно монтируемой в **/dev/shm**, с именами вида **sem.имя** (по этой причине длина имени семафора ограничена **NAME_MAX-4**, а не **NAME_MAX** символами).

Простые примеры

- Программа в файле semunlink.c удаляет именованный семафор.
- В файле semgetvalue.c приведен текст простейшей программы, которая открывает указанный именованный семафор, получает его текущее значение и выводит его.
- Программа в файле semwait.c открывает именованный семафор, вызывает **sem_wait** (которая приостанавливает выполнение процесса, если значение семафора меньше либо равно 0, а при положительном значении семафора уменьшает его на 1), получает и выводит значение семафора, а затем останавливает свою работу при вызове **pause**.

Простые примеры

- В файле sempost.c приведена программа, которая выполняет операцию *post* для указанного семафора (то есть увеличивает его значение на 1), а затем получает значение этого семафора и выводит его.
- Далее рассмотрим работу этих примеров.
- Создадим именованный семафор в CentOS 6.9 и выведем его значение, устанавливаемое по умолчанию при инициализации:
- **[gun@CentOS]$ ./semcreate /test1**
- **[gun@CentOS}$ ls -l /dev/shm/sem.test1**
- **-rw-------. 1 gun gun 16 Фев 2 11:50 /dev/shm/sem.test1**

Простые примеры

- **[gun@CentOS]$ ./semgetvalue test1**
- **value = 1**
- Теперь подождем изменения семафора и прервем работу программы, установившей блокировку:
- **[gun@CentOS]$ ./semwait /test1**
- **pid 19830 has semaphore, value = 0**
- **^C** *//клавиша прерывания работы*
- **[gun@CentOS]$ ./semgetvalue test1**
- **value = 0** *//значение остается нулевым*

Простые примеры

- Приведенный пример иллюстрирует упомянутые ранее особенности. Во-первых, значение семафора обладает живучестью ядра. Значение 1, установленное при создании семафора, хранится в ядре даже тогда, когда ни одна программа не пользуется этим семафором. Во-вторых, при выходе из программы semwait, заблокировавшей семафор, значение его не изменяется, то есть ресурс остается заблокированным. Это отличает семафоры от блокировок **fcntl**, которые снимались автоматически при завершении работы процесса.
- Покажем теперь, что в этой реализации ОС нулевое значение семафора используется для блокирования семафора:

Простые примеры

- **[gun@CentOS]$ ./semgetvalue test1**
- **value = 0** *// это значение сохранилось с конца предыдущего примера*
- **[gun@CentOS]$ ./semwait /test1 &** *// запуск в фоновом режиме*
- **[1] 20298**
- **[gun@CentOS]$ ./semgetvalue test1**
- **value = 0**
- **[gun@CentOS]$ ./semwait /test1 &** *// запуск в фоновом режиме*
- **[2] 20335**
- **[gun@CentOS]$ ./semgetvalue test1**
- **value = 0**

Простые примеры

- **[gun@ CentOS]$ ./sempost /test1**
- **pid 20298 has semaphore, value = 0 //** *вывод программы* **semwait**
- **value = 0 //** *значение после возвращения из* **sem_post**
- **[gun@CentOS]$ ./sempost /test1**
- **value = 1 //** *значение после возвращения из* **sem_post**
- **pid 20335 has semaphore, value = 0 //** *вывод программы* **semwait**
- **[gun@CentOS]$ ./semgetvalue /test1**
- **value = 1**

Простые примеры

- **[gun@CentOS]$ ./sempost /test1**
- **value = 2** // *значение после возвращения из* **sem_post**
- **[gun@CentOS]$ ./semgetvalue /test1**
- **value = 2**
- Именованные семафоры Posix могут быть размещены не только в виртуальной файловой системе, но и в отображаемом на память файле. Отображаемые в память файлы описаны в следующем разделе.
- По окончании работы с примерами, не забудем удалить семафор:

Простые примеры

- **[gun@CentOS]$ ./semunlink /test1**
- **Result:Success**
- Убедимся визуально, что семафор удален:
- **[gun@CentOS]$ ls -l /dev/shm/sem.test1**
- **ls: невозможно получить доступ к /dev/shm/sem.test1: Нет такого файла или каталога**

Функции sem_init и sem_destroy

- До сих пор обсуждались только именованные семафоры Posix. Они идентифицируются аргументом **name**, обычно представляющим собой имя файла в файловой системе. Стандарт Posix описывает также семафоры, размещаемые в памяти, память под которые (типа **sem_t**) выделяет приложение, а инициализируются они системным вызовом:
- **#include <semaphore.h>**
- **int sem_init(sem_t *sem, int shared, unsigned int value);**
- Функция возвращает 0 в случае успешного завершения (проверить!) или –1 в случае ошибки.

Функции sem_init и sem_destroy

- Размещаемый в памяти семафор инициализируется вызовом **sem_init**. Аргумент **sem** указывает на переменную типа **sem_t**, место под которую должно быть выделено приложением. Если аргумент **shared** равен 0, то семафор используется потоками одного процесса (см. главу 7), в противном случае доступ к нему могут иметь несколько процессов. Если аргумент **shared** ненулевой, семафор должен быть размещен в одном из видов разделяемой памяти и должен быть доступен всем процессам, использующим его. Как и в вызове **sem_open**, аргумент **value** задает начальное значение семафора.

Функции sem_init и sem_destroy

- Обратите внимание, что для размещаемого в памяти семафора нет ничего аналогичного флагу **O_CREAT**: функция **sem_init** всегда инициализирует значение семафора. Следовательно, нужно быть внимательным, чтобы вызывать **sem_init** только один раз для каждого семафора. При вызове **sem_init** для уже инициализированного семафора результат непредсказуем.
- После завершения работы с размещаемым в памяти семафором его можно уничтожить, вызвав **sem_destroy**:
- **int sem_destroy(sem_t *sem);**
- Функция возвращает 0 в случае успешного завершения или –1 в случае ошибки.

Функции sem_init и sem_destroy

- Размещаемый в памяти семафор может быть использован в тех случаях, когда нет необходимости использовать имя, связываемое с именованным семафором. Именованные семафоры обычно используются для синхронизации работы неродственных процессов. Имя в этом случае используется для идентификации семафора.
- Размещаемый в памяти семафор не утрачивает функциональности до тех пор, пока память, в которой он размещен, еще доступна какому-либо процессу.

Функции sem_init и sem_destroy

- Если размещаемый в памяти семафор совместно используется потоками одного процесса (аргумент **shared** при вызове **sem_init** равен 0), семафор обладает живучестью процесса и удаляется при завершении последнего.
- Если размещаемый в памяти семафор совместно используется несколькими процессами (аргумент **shared** при вызове **sem_init** равен 1), он должен располагаться в разделяемой памяти, и в этом случае семафор существует столько, сколько существует эта область памяти, поскольку и разделяемая память Posix, и разделяемая память System V обладают живучестью ядра.

Ограничения на семафоры

- Стандартом Posix определены два ограничения на семафоры:
- **SEM_NSEMS_MAX** — максимальное количество одновременно открытых семафоров для одного процесса (Posix требует, чтобы это значение было не менее 256);
- **SEM_VALUE_MAX** — максимальное значение семафора (Posix требует, чтобы оно было не меньше 32767).
- Две эти константы обычно определены в заголовочном файле **<unistd.h>** и могут быть получены во время выполнения вызовом **sysconf**, как показано в файле semsysconf.c, доступной на сайте.

Ограничения на семафоры

- При запуске этой программы в системе CentOS 6.9 получим следующий результат:
- **[gun@CentOS]$ ./semsysconf**
- **SEM_NSEMS_MAX = 256, SEM_VALUE_MAX = 32767**