1 Семафоры

Семафоры — одно из старейших средств разделения доступа к критическим ресурсам параллельно работающих процессов. Дейкстра определил две операции над семафорами: $P(s)$ и $V(s)$. Операция P блокирует семафор, операция V — деблокирует семафор. Более точно, операции P и V определены следующим образом:

Операция $P(s)$ проверяет, что семафор s открыт. Если семафор открыт, операция закрывает семафор и завершается. Операция проверки и открытия является атомарной, то есть ни один другой процесс не может наблюдать ситуации, когда семафор проверен, но ещё не закрыт. Если на момент проверки семафор уже закрыт, операция P приостановит процесс до тех пор, пока некоторый другой процесс не откроет семафор. После того, как некоторый процесс открыл семафор, система выбирает из множества процессов, ожидающих открытия семафора, какой-то один и разблокирует его.

Операция $V(s)$ разблокирует семафор. Эта операция никогда не приостанавливает выполнение процесса.

Обычно сам по себе семафор s — это переменная некоторого целого типа. Значению 0 может соответствовать открытый семафор, а значению 1 — закрытый семафор. Если семафор может принимать только два значения, такой семафор называется *бинарным*.

Фрагмент программы, использующей бинарный семафор, может выглядеть следующим образом:

```
semaphore s;

  /* программа */
  P(s);
  /* критическая секция */
  V(s);
```

Расширением бинарного семафора является *считающий семафор*. У считающего семафора значение 0 соответствует закрытому семафору, а значения, большие 0, открытому семафору. Операции P и V тогда принимают не один, а два параметра.

Операция $P(s,i)$ атомарно уменьшает значение семафора s на i. Если на момент проверки значение семафора уже было меньше i, процесс приостанавливается до тех пор, пока значение семафора не станет больше или равно i.

Операция $V(s,i)$ увеличивает значение семафора s на i. Операция всегда выполняется без блокировки процесса. У семафора может быть некоторое максимальное значение, которое определяется типом данных, отведённым для хранения значения семафора.

В случае считающих семафоров может потребоваться начальная инициализация семафора значением, отличным от нуля, так как в случае считающих семафоров 0 означает закрытый ресурс, а в начале программы ресурс, скорее всего, должен быть открыт. Считающий семафор можно использовать тогда, когда разделяемый ресурс имеет ограниченную ёмкость, и мы хотим блокировать процесс, когда ёмкость ресурса временно исчерпана.

Поскольку операция P должна быть атомарна, кроме того, эта операция включает приостановку работы процесса и пробуждение процесса, эта операция может быть корректно реализована только в ядре операционной системы. Поэтому ядро должно предоставлять некоторые системные вызовы для работы с семафорами.

1.1 Семафоры в System V

Развитые средства межпроцессной коммуникации, включающие именованные каналы, очереди сообщений, семафоры и разделяемую память появились в Unix System V. Примерно в то же время в Unix BSD появились сокеты и файлы, отображаемые в память. До этого единственными средствами межпроцессного взаимодействия были файлы, анонимные каналы и сигналы. Позднее все эти средства межпроцессного взаимодействия были реализованы в обеих разновидностях UNIX. Семафоры, разделяемая память и очереди сообщений часто объединяются в понятие «средства межпроцессного взаимодействия UNIX System V». Далее в этом документе мы будем иногда на них ссылаться как на SysV IPC.

Понимание семафоров в System V ещё более расширено по сравнению с оригинальным определением. Позволяется задавать массивы семафоров, в котором каждый отдельный семафор может рассматриваться или как бинарный, или как считающий. Более того, позволяется задавать несколько операций с семафорами одного массива семафоров за один вызов. В этом случае система гарантирует атомарность выполнения *всех* операций, то есть либо все операции выполнились, либо ни одна из указанных операций не выполнилась, а процесс переведён в состояние ожидания. Никакой другой процесс не может наблюдать ситуацию, когда часть операций выполнилась, а часть ожидает своего выполнения.

Семафоры хранятся в адресном пространстве ядра. Но, в отличие, например, от файловых дескрипторов, время жизни массивов семафоров не ограничено временем работы создавшего его процесса. То есть, массив семафоров не уничтожается, когда завершает работу последний процесс, использующий данный массив семафоров. Для уничтожения массива необходимо использовать явную команду уничтожения. Поэтому процесс, работающий с массивом семафоров, должен аккуратно обрабатывать завершение работы процесса. Впрочем, эта проблема не свойственна только средствам межпроцессного взаимодействия System V, но и временным файлам, файлам замков и именованным каналам.

Каждый массив семафоров должен иметь имя, уникальное в системе, чтобы разные никак не связанные друг с другом процессы могли использовать данный массив семафоров. Для объектов SysV IPC именем является некоторое целое положительное число, причём пространства имён для массивов семафоров, сегментов разделяемой памяти и очередей сообщений различны, то есть одно и то же число может обозначать и массив семафоров, и очередь сообщений, и сегмент разделяемой памяти. Это число имеет тип `key_t`, и мы будем ссылаться на него как на *ключ* массива семафоров.

Чтобы процесс мог работать с массивом семафоров, он должен «получить» массив семафоров, то есть преобразовать ключ массива семафоров в идентификатор массива семафоров, действительный для данного процесса (аналог файлового дескриптора). При этом могут указываться дополнительные флаги, например, флаг создания массива семафоров и права доступа к нему. Затем процесс может выполнять операции с массивом семафоров, используя идентификатор массива семафоров, полученный ранее.

1.2 Функции работы с массивами семафоров

Как было сказано ранее, ключ семафора — это некоторое положительное целое число. Чтобы «облегчить» программисту задачу выбора этого числа, предусмотрена специальная функция `ftok`.

```
#include <sys/types.h>
#include <sys/ipc.h>
key_t ftok(char *pathname, char proj);
```

Функция применима к любым средствам межпроцессного взаимодействия SysV IPC: массивам семафоров, очередям сообщений, сегментам разделяемой памяти. Эта функция генерирует ключ объекта SysV IPC по имени *существующего* файла `pathname`. Аргумент `proj` — это некоторый идентификатор проекта. В описании функции не специфицируется, что это должно быть за число, соответственно, это значение может быть произвольным. При генерации ключа используется номер устройства и номер индексного дескриптора файла, поэтому файл не должен удаляться и создаваться заново в промежутке между обращениями к `ftok`, иначе будут получены разные ключи. Используется только часть бит номера индексного дескриптора и номера устройства, поэтому вполне возможна (хотя и маловероятна) коллизия генерируемых ключей, когда двум разным файлам соответствует один и тот же ключ объекта.

1.2.1 Получение массива семафоров

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semget(key_t key, int nsems, int semflg);
```

Функция `semget` позволяет связать с массивом семафоров с ключом `key` идентификатор семафора. Этот идентификатор должен использоваться при всех обращения к функциям работы с этим массивом семафоров. Параметр `nsems` задаёт количество семафоров в массиве семафоров. Он может быть равен 0, если процесс не пытается создать новый массив семафоров. Параметр `semflg` определяет флаги создания массива семафоров и права доступа к создаваемому массиву семафоров.

`IPC_CREAT` Если этот флаг установлен, массив семафоров будет создан.

`IPC_EXCL` Если установлен этот флаг, и установлен флаг `IPC_EXCL`, выполнение завершится с ошибкой, если массив семафоров с таким ключом уже существует.

9 младших бит параметра `semflg` определяют права доступа к создаваемому массиву семафоров. Назначение бит прав доступа точно такое же, как у бит прав доступа к файлу, но бит `x`, естественно, игнорируется. Например, права `0600` обозначают доступ на чтение и запись только для создателя семафора.

Функция инициализирует значение семафоров нулями. Если необходима инициализация массива семафоров какими-либо другими значениями, нужно использовать функцию `semctl`. Чтобы избежать при этом race condition, необходимо в массиве семафоров завести ещё один семафор, который примет значение 1, когда все семафоры будут корректно проинициализированы. Один процесс создаёт и инициализирует семафоры, а другие процессы должны пытаться создавать массив с флагом `IPC_EXCL`, и если создание не удалось из-за того, что массив семафоров был уже создан, приостановить своё выполнение до тех пор, пока упомянутый выше семафор завершения инициализации не примет значение 1.

Если функция завершилась успешно, возвращается идентификатор массива семафоров. При ошибке возвращается -1, а переменная `errno` устанавливается в код ошибки. Возможные коды ошибок приведены ниже.

EACCES Для заданного ключа массив семафоров уже существует, но процесс не имеет прав доступа к нему.

EEXIST Массив семафоров существует, а параметр semflg содержит флаги IPC_CREAT и IPC_EXCL.

EIDRM Заданный массив семафоров помечен на удаление.

ENOENT Массив семафоров с заданным ключом не существует, а флаг IPC_CREAT не был указан.

ENOMEM Недостаточно памяти ядра для создания семафора.

ENOSPC Превышено максимальное количество семафоров или массивов семафоров в системе.

1.2.2 Управление массивом семафоров

Под управлением массивом семафоров понимается выполнение разных дополнительных операций, например, инициализация массива семафоров, или удаление массива семафоров. Все эти операции выполняются с помощью функции semctl. Прототип этой функции выглядит следующим образом:

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>

int  semctl(int semid, int semnum, int cmd, ...);
```

Аргумент semid определяет идентификатор массива семафоров, над которым следует произвести операцию. Аргумент semnum задаёт номер семафора в массиве, аргумент cmd задаёт выполняемую команду. Здесь функция semctl описана как принимающая переменное количество параметров, хотя она всегда принимает 4 параметра. Последний параметр, назовём его arg, имеет разные типы в зависимости от того, какая команда над массивом семафоров выполняется. При описании каждой команды будет точно определено, какой именно тип данных требуется.

Команда **IPC_RMID** удаляет массив семафоров. Все другие процессы, которые заблокированы на этом массиве семафоров, разблокируются, а функция semop вернёт этим процессам ошибку с кодом EIDRM. Аргументы semnum и arg игнорируются. Прототип функции semctl может условно выглядеть следующим образом:

```
int semctl(int semid, int semnum, int cmd);
```

Команда **GETALL** возвращает значение всех семафоров в массив arg типа unsigned short. Аргумент semnum игнорируется. Прототип функции semctl в этом случае может выглядеть следующим образом:

```
int semctl(int semid, int semnum, int cmd,
           unsigned short *arg);
```

Команда **GETVAL** возвращает значение семафора с номером semnum в массиве семафоров. Аргумент arg игнорируется. Прототип функции semctl может условно выглядеть следующим образом:

```
int semctl(int semid, int semnum, int cmd);
```

Команда **SETALL** устанавливает значение всех семафоров. Массив новых значений семафоров передаётся в параметре arg. Прототип функции semctl может быть записан следующим образом:

```
int semctl(int semid, int semnum, int cmd,
           unsigned short *arg);
```

Команда **SETVAL** устанавливает значение семафора с номером semnum. Новое значение семафора передаётся в аргументе arg. Если новое значение меньше нуля или больше константы SEMVMX, определяющей максимальное значение семафора в данной системе (например, для Linux значение SEMVMX равно 32767), функция завершится с ошибкой ERANGE.

При успешном завершении функция semctl возвращает значение, большее 0, при ошибке возвращается -1, а переменная errno устанавливается в код ошибки. Возможные коды ошибок приведены ниже.

EACCES	Процесс не имеет достаточно полномочий, чтобы выполнить заданную команду.
EFAULT	Неверный адрес передан в параметре arg (если требуется параметр-адрес).
EIDRM	Массив семафоров был удалён.
EINVAL	Неверная команда или идентификатор массива семафоров.
EPERM	Недостаточно полномочий выполнить команду.
ERANGE	Запрошено выполнение команды SETALL или SETVAL, а новое значение семафора меньше 0 или больше, чем максимальное значение SEMVMX, зависящее от реализации.

1.2.3 Операции с массивом семафоров

Все операции с массивом семафоров: блокировка, разблокировка, ожидание выполняются с помощью функции semop.

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
int semop(int semid, struct sembuf *sops, unsigned nsops);
```

Функция позволяет задавать сразу несколько операций над массивом семафоров semid. Операции должны находится в массиве, адрес которого передаётся в аргументе sops. В аргументе nsops передаётся количество операций в массиве операций. Каждая операция над семафором задаётся структурой struct sembuf, определённой следующим образом:

```
struct sembuf
{
  short sem_num;   /* номер семафора (0 - первый) */
  short sem_op;    /* операция над семафором */
  short sem_flg;   /* флаги операции */
};
```

Поле sem_flg может содержать флаг IPC_NOWAIT, указывающий, что эта операция не должна приводить к блокированию выполнения процесса, а вместо этого функция semop должна завершиться с ошибкой с кодом EAGAIN.

Операции над семафорами будут выполнены в том и только том случае, когда все операции завершатся успешно. Операции над семафорами выполняются атомарно, то есть никакой другой процесс не может наблюдать ситуацию, когда часть операций над семафорами уже завершилась, а часть — нет.

Поле sem_num задаёт номер семафора в массиве семафоров (номер отсчитывается, как обычно, от 0), над которым нужно выполнить операцию. Поле sem_op задаёт операцию:

одну из трёх в зависимости от знака значения sem_op.

- Если значение sem_op **положительно**, операция добавляет это значение к значению семафора. Операция не может привести к блокированию процесса. Процесс должен иметь права на запись в массив семафоров.

- Если значение sem_op **равно 0**, операция проверяет значение семафора. Если значение семафора равно 0, операция завершается успешно, и проверяется следующая операция в массиве операций. Если значение семафора не равно 0, и в поле sem_flg указан флаг IPC_NOWAIT, вызов semop завершается с ошибкой EAGAIN, и ни одна из запрошенных операций не выполняется. Если значение семафора не равно 0, и в поле sem_flg не указан флаг IPC_NOWAIT, процесс будет приостановлен до наступления одного из следующих событий:

  - Значение семафора станет равно 0.
  - Массив семафоров будет удалён. В этом случае semop завершится с ошибкой EIDRM.
  - Процесс получит сигнал, обработчик которого вернёт управление процессу. В этом случае semop завершится с ошибкой EINTR.

- Если значение sem_op **отрицательно**, значение семафора уменьшается на заданную величину. Если текущее значение семафора больше, чем величина -sem_op, значение семафора уменьшается на -sem_op. Операция завершается успешно, и проверяется следующая операция в массиве операций. Если текущее значение семафора меньше -sem_op, и указан флаг IPC_NOWAIT, вызов semop завершается с ошибкой EAGAIN, и ни одна операция из запрошенных не выполнится. Если текущее значение семафора меньше -sem_op, а флаг IPC_NOWAIT не указан, процесс будет заблокирован до тех пор, пока либо значение семафора не станет большим -sem_op, либо массив семафоров не будет удалён (в этом случае semop завершится с ошибкой EIDRM), либо процесс получит сигнал, который вызовет возврат управления в процесс (в этом случае semop завершится с ошибкой EINTR).

В случае успешного завершения функция возвращает 0, а при ошибке возвращает -1, и переменная errno устанавливается в одно из следующих значений.

E2BIG	Аргумент nsops больше максимального значения SEMOPM, зависящего от реализации.
EACCES	Недостаточно прав выполнить какую-либо из заданных операций.
EAGAIN	Процесс мог бы быть заблокирован, но указан флаг IPC_NOWAIT.
EFAULT	Недопустимый адрес sops.
EFBIG	Значение sem_num для некоторой операции выходит за диапазон допустимых значений.
EIDRM	Массив семафоров был удалён.
EINTR	Сон процесса был прерван приходом сигнала.
EINVAL	Аргументы semid или nsops имеют недопустимое значение.
ERANGE	Для некоторой операции величина sem_op, добавленная к текущему значению семафора, вызовет выход за границу SEMVMX допустимых значений семафора.

2 Разделяемая память

Разделяемая память — это самый быстрый механизм межпроцессного взаимодействия. При использовании разделяемой памяти два или более процессов имеют доступ к одному и тому же фрагменту физической памяти, поэтому все изменения, которые один из процессов делает в разделяемой памяти, немедленно становятся доступны другим процессам, и не нужны никакие операции пересылки изменённого значения с помощью файлов, каналов и пр.

Практически всегда при использовании разделяемой памяти необходимо использование какого-либо другого механизма, который бы обеспечил сериализацию критических секций, то есть строго последовательное выполнение разными процессами критических секций, модифицирующих один и тот же разделяемый ресурс. Наиболее естественно для этой цели использовать рассмотренные выше семафоры.

Сегментом разделяемой памяти мы назовём блок разделяемой памяти, полученный за одну операцию `shmget`.

Данные, хранимые в некотором сегменте разделяемой памяти, не должны содержать указателей на объекты, находящиеся вне этого сегмента разделяемой памяти, поскольку такой указатель имеет смысл только в адресном пространстве одного процесса. Более того, данные, хранимые в некотором сегменте разделяемой памяти, не должны содержать указателей на объекты, хранящиеся даже в этом же сегменте разделяемой памяти. Один и тот же сегмент разделяемой памяти может отображаться у разных процессов на разные виртуальные адреса, и не всегда возможно сделать, чтобы разные процессы отображали сегмент разделяемой памяти на одни и те же виртуальные адреса.

Сегмент разделяемой памяти идентифицируется, как и другие объекты SysV IPC, с помощью *ключа* — положительного целого числа. Функция `ftok` позволяет получить ключ по имени некоторого существующего файла. Каждый процесс должен сначала получить сегмент разделяемой памяти с помощью вызова `shmget`. При этом сегмент может быть создан. Результатом работы этой функции является идентификатор сегмента разделяемой памяти, который должен использоваться во всех остальных операциях с этим сегментом разделяемой памяти. После того, как сегмент получен, его нужно подключить к адресному пространству процесса с помощью вызова `shmat`. После этого процесс может работать с сегментом разделяемой памяти как с обычной памятью процесса. Процесс может отсоединить сегмент разделяемой памяти с помощью вызова `shmdt`. Отсоединение сегмента разделяемой памяти, и даже завершение работы всех процессов, использовавших данный сегмент разделяемой памяти, не означает его уничтожения системой. Ненужный сегмент должен быть явно уничтожен с помощью вызова функции `shmctl`.

2.1 Функции работы с разделяемой памятью

2.1.1 Получение разделяемой памяти

```
#include <sys/ipc.h>
#include <sys/shm.h>
int shmget(key_t key, int size, int shmflg);
```

Функция `shmget` преобразовывает ключ сегмента разделяемой памяти `key` в идентификатор сегмента разделяемой памяти. Если в параметре `shmflg` указан флаг `IPC_CREAT`, создаётся новый сегмент размера `size` с правами доступа, определёнными в 9 младших битах параметра `shmflg`. Если дополнительно был указан флаг `IPC_EXCL`, и сегмент с задан-

ным ключом уже существует, функция shmget завершается с ошибкой EEXIST. Если флаг IPC_CREAT не указан, значение параметра size может быть равно 0.

Системные вызовы создания/уничтожения процессов влияют на сегмент разделяемой памяти следующим образом:

- При вызове fork сыновний процесс наследует подключённые сегменты разделяемой памяти отцовского процесса.
- При вызове exec все подключённые сегменты разделяемой памяти отключаются, но не уничтожаются.
- При вызове exit все подключённые сегменты разделяемой памяти отключаются, но не уничтожаются.

В случае успешного завершения функция возвращает идентификатор сегмента разделяемой памяти (положительное число), а в случае неудачи возвращается -1, и переменная errno устанавливается в код ошибки. Возможные коды ошибок приведены ниже.

EINVAL	Недопустимый параметр size.
EEXIST	Комбинация флагов IPC_CREAT и IPC_EXCL указана в аргументе shmflg, но сегмент с заданным ключом уже существует.
EIDRM	Сегмент помечен на удаление.
ENOSPC	Превышен лимит системы на количество сегментов разделяемой памяти либо на их суммарный размер.
ENOENT	Сегмент с заданным ключом не существует, а флаг IPC_CREAT не был указан.
EACCES	Процесс не имеет достаточно прав доступа к сегменту.
ENOMEM	Недостаточно памяти для выделения сегмента.

2.1.2 Удаление сегмента разделяемой памяти

Удалить сегмент разделяемой памяти можно с помощью общей функции управления сегментом разделяемой памяти.

```
#include <sys/ipc.h>
#include <sys/shm.h>
int shmctl(int shmid, int cmd, struct shmid_ds *buf);
```

Аргумент shmid — идентификатор сегмента разделяемой памяти, полученный с помощью shmget. Если указать в аргументе cmd значение IPC_RMID, а аргументом buf передать NULL, сегмент разделяемой памяти будет удалён. Точнее, он будет помечен на удаление, и ни один процесс не сможет его получить или подключить к своему адресному пространству. Окончательно такой сегмент разделяемой памяти будет удалён, когда его отключит последний процесс, который его использовал.

2.1.3 Подключение сегмента разделяемой памяти

```
#include <sys/types.h>
#include <sys/shm.h>
void *shmat(int shmid, const void *shmaddr, int shmflg);
```

Функция shmat подключает сегмент разделяемой памяти с идентификатором shmid к адресному пространству процесса. Адрес подключения задаётся аргументом shmaddr следующим образом:

- Если shmaddr равен 0, система пытается подобрать адрес сегмента, начиная от старших адресов к младшим.
- Если shmaddr не равен 0, а в аргументе shmflg указан флаг SHM_RND, указанный адрес выравнивается вниз по границе страницы памяти процессора, и сегмент памяти подключается по этому адресу. Если флаг SHM_RND не установлен, адрес подключения shmaddr должен быть выровнен по границе страницы.

Если в параметре shmflg указан флаг SHM_RDONLY, сегмент подключается в режиме «только на чтение», в противном случае сегмент подключается и для чтения, и для записи. Естественно, у процесса должны быть соответствующие права на данный сегмент разделяемой памяти.

При успешном завершении функция возвращает адрес начала подключённого сегмента разделяемой памяти. При ошибке функция возвращает значение (void*) -1, и переменная errno устанавливается в код ошибки.

EACCES Процесс не имеет достаточно прав на указанный тип подключения сегмента.

EINVAL Недопустимое значение shmid, неверное (невыровненное) значение shmaddr, или подключение по адресу shmaddr невозможно.

ENOMEM Недостаточно памяти ядра.

2.1.4 Отключение сегмента разделяемой памяти

```
#include <sys/types.h>
#include <sys/shm.h>
int shmdt(const void *shmaddr);
```

Функция shmdt отсоединяет сегмент разделяемой памяти с начальным адресом shmaddr от адресного пространства процесса. В случае успеха функция возвращает 0. При неудаче (неправильный адрес shmaddr) возвращается -1, а переменная errno устанавливается в значение EINVAL.

2.2 Пример программы

Рассмотрим программу, которая проиллюстрирует работу с массивами семафоров и с сегментом разделяемой памяти.

Программа порождает два процесса, которые обмениваются друг с другом целым числом. Первый процесс получает от второго некоторое число, печатает, увеличивает его на 1 и отправляет обратно второму процессу и наоборот. Для пересылки значения числа используется сегмент разделяемой памяти, а для синхронизации процессов — массив бинарных семафоров. Программа корректно освобождает занятые ресурсы.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/sem.h>
#include <sys/shm.h>
```

```
#include <sys/wait.h>
#include <signal.h>

#define FTOK_FILE "sem14_2"
#define FTOK_PROJ 0

key_t  ipc_key;
int    shm_id = -1;
int    sem_id = -1;
int   *mem_ptr = (void *) -1;
int    pid[2] = { -1, -1 };

/* функция освобождает все занятые ресурсы */
void cleanup(void)
{
  if (pid[0] > 0) kill(pid[0], SIGTERM);
  if (pid[1] > 0) kill(pid[1], SIGTERM);
  if (mem_ptr != (void*) -1) shmdt(mem_ptr);
  if (sem_id >= 0) semctl(sem_id, 0, IPC_RMID);
  if (shm_id >= 0) shmctl(shm_id, IPC_RMID, 0);
}

/* обработчик сигналов завершения программы */
void handler(int signo)
{
  exit(0);
}

void pexit(char const *str)
{
  perror(str);
  exit(1);
}

void do_work(int read_ind, int write_ind)
{
  // операции, выполняемые при ожидании: ждем появления 0 (открыто)
  // на семафоре данного процесса, сразу же устанавливаем его
  // в 1 (закрыто)
  struct sembuf wait_ops[] = {{ -1, 0, 0 }, {-1, 1, 0}};
  // операция при разблокировании: семафор должен быть
  // установлен в 1, тогда переустанавливаем его в 0
  struct sembuf unlock_ops[] = {{ -1, -1, 0 }};
  sigset_t s, os;

  sigfillset(&s);
  wait_ops[0].sem_num = read_ind;
  wait_ops[1].sem_num = read_ind;
  unlock_ops[0].sem_num = write_ind;

  while (1) {
```

```
    // блокируем сигналы завершения на время ожидания и печати
    sigprocmask(SIG_BLOCK, &s, &os);
    // ожидаем
    if (semop(sem_id, wait_ops, 2) < 0) break;
    printf("%d: %d\n", getpid(), *mem_ptr);
    fflush(stdout);
    (*mem_ptr)++;
    // разблокируем другой процесс
    if (semop(sem_id, unlock_ops, 1) < 0) break;
    // разблокируем сигналы
    sigprocmask(SIG_SETMASK, &os, NULL);
  }
}

int main(void)
{
  int i;

  // регистрируем функцию, которая будет автоматически вызываться
  // при завершении программы с помощью exit() или
  // return из функции main
  atexit(cleanup);

  // устанавливаем обработчики сигналов
  signal(SIGINT, handler);
  signal(SIGTERM, handler);
  signal(SIGALRM, handler);
  signal(SIGHUP, handler);

  // получаем и инициализируем разделяемую память
  ipc_key = ftok(FTOK_FILE, FTOK_PROJ);
  printf("Key = %d\n", ipc_key);
  if ((shm_id=shmget(ipc_key,sizeof(int),IPC_CREAT|IPC_EXCL|0600))<0)
    pexit("shmget");
  mem_ptr = shmat(shm_id, NULL, 0);
  if (mem_ptr == (void*) -1)
    pexit("shmat");
  *mem_ptr = 0;

  // получаем семафоры
  if ((sem_id = semget(ipc_key, 2, IPC_CREAT | IPC_EXCL | 0600)) < 0)
    pexit("semget");
  // значение семафора [0] - 0 (открыт), [1] - 1 (закрыт)
  if (semctl(sem_id, 1, SETVAL, 1) < 0)
    pexit("semctl");

  for (i = 0; i < 2; i++) {
    pid[i] = fork();
    if (pid[i] < 0) pexit("fork");
    if (!pid[i]) {
      // сыновние процессы не выполняют никаких особенных действий
```

```
        // при завершении работы, поэтому обработчики сбрасываются
        signal(SIGINT, SIG_DFL);
        signal(SIGTERM, SIG_DFL);
        signal(SIGALRM, SIG_DFL);
        signal(SIGHUP, SIG_DFL);
        do_work(i, 1 - i);
        _exit(1);
      }
    }

    alarm(5);
    wait(0); wait(0);

    return 0;
}
```

3 Очереди сообщений

Последний механизм, входящий в состав SysV IPC — это очереди сообщений. Сама очередь сообщений находится в адресном пространстве ядра и имеет ограниченный размер. В отличие от каналов, которые обладают теми же самыми свойствами, очереди сообщений сохраняют границы сообщений. Это значит, что ядро ОС гарантирует, что сообщение, поставленное в очередь, не сольётся при чтении из очереди сообщений с предыдущим или следующим сообщением. Кроме того, с каждым сообщением связывается его тип. Процесс, читающий очередь сообщений, может отбирать только сообщения заданного типа или все сообщения кроме сообщений заданного типа.

Следует заметить, что, к сожалению, не определены системные вызовы, которые позволяют читать сразу из нескольких очередей сообщений, или из очередей сообщений и файловых дескрипторов. Видимо, отчасти и по-этому очереди сообщений широко не используются.

3.1 Создание очереди сообщений

Очередь сообщений создаётся с помощью системного вызова `msgget`.

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int msgget(key_t key, int msgflg);
```

Эта функция создаёт новую очередь сообщений и возвращает её идентификатор. Идентификатор очереди сообщений — неотрицательное целое число. Он используется в дальнейшем во всех операциях с данной очередью сообщений и играет ту же роль, что дескриптор открытого файла при операциях с файлами. При ошибке функция возвращает значение -1, а в переменную `errno` записывается код ошибки.

Параметр `key` — ключ объекта. Это - положительное целое число, уникальное в системе. Пространство ключей очередей сообщений не связано с пространством ключей семафоров и областей разделяемой памяти, то есть в системе могут существовать область раз-

деляемой памяти, массив семафоров и очередь сообщений с одним и тем же ключом. Ключ выбирается программистом или генерируется с помощью функции ftok.

Допускается специальное значение ключа IPC_PRIVATE, которое обозначает анонимную очередь сообщений. В этом случае никакой другой процесс не сможет подключиться к этой очереди сообщений с помощью вызова msgget. При указании ключа IPC_PRIVATE очередь создаётся всегда, не зависимо от флага IPC_CREAT.

Параметр msgflg задаёт права доступа к создаваемой очереди и флаги создания очереди. Если установлен флаг IPC_CREAT, очередь будет создана, если до этого она ещё не существовала. Указание флага IPC_EXCL совместно с флагом IPC_CREAT приведёт к тому, что очередь сообщений будет создана, если она ещё не существовала, но вызов msgget завершится ошибкой, если очередь сообщений уже существовала.

В случае, когда создаётся новая очередь сообщений, младшие 9 битов определяют права доступа к создаваемой очереди. Биты интерпретируются точно так же, как и в случае задания прав доступа к файлам (за исключением того, что бит x не используется).

Если очередь сообщений уже существует, и флаг IPC_EXCL не задан, проверяется, что очередь сообщений не помечена на удаление, и проверяются права доступа процесса к данной очереди сообщений.

Если вызов msgget завершился с результатом -1, переменная errno содержит один из следующих кодов ошибки.

EACCES	Очередь сообщений уже существует, но процесс не имеет прав для доступа к ней.
EEXIST	Очередь сообщений уже существует, а параметр msgflg содержит установленные флаги IPC_CREAT и IPC_EXCL.
EIDRM	Очередь сообщений помечена на удаление.
ENOENT	Очередь сообщений не существует, и флаг IPC_CREAT не указан.
ENOMEM	Очередь сообщений должна быть создана, но система не располагает достаточным количеством памяти.
ENOSPC	Очередь сообщений должна быть создана, но достигнуто максимальное количество очередей сообщений в системе (параметр MSGMNI ядра ОС).

3.2 Удаление очереди сообщений

Удаление очереди сообщений выполняется с помощью системного вызова msgctl указанием команды IPC_RMID в параметр cmd.

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int msgctl(int msqid, int cmd, struct msqid_ds *buf);
```

Данная функция выполняет операцию над очередью сообщений. Идентификатор очереди сообщений, полученный в результате вызова msgget, указывается в параметре msqid. Выполняемая операция задаётся параметром cmd, который может принимать одно из следующих значений:

- IPC_STAT. Получить информацию о данной очереди сообщений. Заполняется переменная, адрес которой указан в параметре buf. Мы не будем здесь подробно рассматривать эту команду.

- IPC_SET. Изменить некоторые служебные данные очереди сообщений. Для этого в переменную, адрес которой передаётся в параметре buf, записываются новые значения. Эту команду мы также не будем подробно рассматривать.

- IPC_RMID. Немедленно удалить данную очередь сообщений и все связанные с ней структуры данных. Все процессы, заблокированные на чтении из этой очереди сообщений или записи в эту очередь сообщений, разблокируются, и соответствующие системные вызовы (msgrcv или msgsnd) завершаются с ошибкой EIDRM. Процесс, выполняющий операцию удаления, должен иметь достаточный уровень привилегий, либо его эффективный идентификатор пользователя должен совпадать с идентификатором создателя или владельца очереди сообщений.

При успешном завершении возвращается 0, а при неуспешном — -1, и в переменную errno записывается код возникшей ошибки.

EACCES	Аргумент cmd равен IPC_STAT, но процесс не имеет прав на чтение из очереди сообщений.
EFAULT	Аргумент cmd равен IPC_SET или IPC_STAT, но в параметре buf передан недопустимый адрес.
EIDRM	Очередь сообщений уже помечена на удаление.
EINVAL	Аргументы cmd или msqid содержат недопустимое значение.
EPERM	Аргумент cmd равен IPC_SET или IPC_RMID, но процесс не имеет достаточно привилегий для выполнения этой команды.

3.3 Запись в очередь сообщений

Добавить сообщение в очередь сообщений можно с помощью системного вызова msgsnd.

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

int msgsnd(int msqid, struct msgbuf *msgp, size_t msgsz, int msgflg);
```

Данный системный вызов добавляет копию сообщения, адрес которого передан в аргумента msgp, в очередь сообщений с идентификатором, переданным в аргумент msqid.

Чтобы добавить сообщение в очередь процесс должен создать структуру следующего общего вида:

```
struct msgbuf {
        long mtype;      /* тип сообщения, должен быть > 0 */
        char mtext[0];   /* данные */
};
```

Поле mtext — это массив или другая структура данных, размер которой задаётся в параметре msgsz. Допускаются сообщения нулевой длины, то есть без поля mtext. В поле mtype записывается тип сообщения — положительное целое число, которое может использоваться при чтении из очереди сообщений для фильтрации сообщений.

Размер сообщения не должен превышать константы MSGMAX, конкретное значение которой зависит от реализации. Например, в ядре Linux она равна 8192 байт.

Для успешной записи процесс должен иметь права на запись в очередь сообщений.

Если в буфере очереди сообщений достаточно места для нового элемента, msgsnd успешно завершается без ожидания. Ёмкость очереди сообщений задаётся полем msg_bytes служебной структуры данных очереди. Исходное значение этого поля при создании очереди сообщений равно MSGMNB, но оно может быть изменено с помощью системного вызова msgctl, если процесс имеет суперпользовательские полномочия. Значение константы MSGMNB зависит от реализации. В ядре Linux эта константа равна 16384 байтам.

Если места для нового элемента в очереди сообщений недостаточно, msgsnd блокирует процесс до тех пор, пока не освободиться достаточно места. Однако, если в аргументе msgflg установлен флаг IPC_NOWAIT, то при отсутствии места в очереди сообщений системный вызов завершится немедленно с ошибкой EAGAIN.

Если вызов msgsnd заблокировал процесс, он может также завершиться в следующих случаях.

- Очередь сообщений удалена с помощью вызова msgctl. В этом случае возвращается ошибка EIDRM.
- Процессу был доставлен сигнал. В этом случае возвращается ошибка EINTR. Системный вызов msgsnd никогда не перезапускается, если был прерван поступлением сигнала, даже если флаг SA_RESTART был задан при установке обработчика сигнала.

При успешном завершении msgsnd возвращает 0, а при ошибке — -1, и в переменную errno записывается код ошибки. Возможные коды ошибок приведены ниже.

EAGAIN	Сообщение не может быть добавлено в очередь, так как в данный момент в очереди нет достаточно места, и флаг IPC_NOWAIT указан в аргументе msgflg.
EACCES	Процесс не имеет прав записи в данную очередь сообщений.
EFAULT	В аргументе msgp передан недопустимый адрес.
EIDRM	Очередь сообщений была удалена.
EINTR	При ожидании освобождения места в очереди процесс получил и обработал сигнал.
EINVAL	Недопустимое значение аргумента msqid или неположительное значение поля mtype, или недопустимое значение аргумента msgsz (меньшее чем 0 или большее чем MSGMAX).
ENOMEM	У ядра недостаточно памяти, чтобы сделать копию сообщения msgbuf.

3.4 Чтение из очереди сообщений

Для чтения из очереди сообщений используется функция msgrcv.

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/msg.h>

ssize_t msgrcv(int msqid, struct msgbuf *msgp, size_t msgsz,
               long msgtyp, int msgflg);
```

Этот системный вызов считывает одно сообщение из очереди сообщений, идентификатор которой задаётся в аргументе msqid. Аргумент msgp задаёт адрес буфера для чтения сообщения. После успешного чтения сообщение удаляется из очереди.

Аргумент msgsz задаёт максимальный размер в байтах поля mtext переменной, адрес которой указан аргументом msgp. Если длина сообщения больше, чем значение аргумента msgsz, и у аргумента msgflg установлен флаг MSG_NOERROR, сообщение будет обрезано по заданной длине, и обрезанная часть будет потеряна. Если флаг MSG_NOERROR не

установлен, сообщение не удаляется из очереди, а системный вызов завершается с ошибкой E2BIG.

Чтобы считать сообщение из очереди процесс должен создать структуру следующего общего вида:

```
struct msgbuf {
        long mtype;      /* тип сообщения, должен быть > 0 */
        char mtext[0];   /* данные */
};
```

Поле mtext — это массив или другая структура данных, размер которой задаётся в параметре msgsz. Допускаются сообщения нулевой длины, то есть без поля mtext. Если размер первого подходящего сообщения превысит значение аргумента msgsz, системный вызов msgrcv завершится с ошибкой, или сообщение будет обрезано.

Если функция msgrcv завершилась успешно, в поле mtype записывается тип сообщения — положительное целое число, а в поле mtext — содержимое сообщения.

Чтобы читать из очереди сообщения, процесс должен иметь права на чтение для своего эффективного идентификатора пользователя.

Аргумент msgtyp задаёт ожидаемый тип сообщения. Аргумент может принимать следующие значения.

- Если msgtyp равен 0, считывается первое сообщение из очереди.
- Если msgtyp больше 0, считывается первое сообщение в очереди, тип которого совпадает с msgtyp. Но если в аргументе msgflg установлен флаг MSG_EXCEPT, считывается первое сообщение в очереди, тип которого не совпадает с msgtyp.
- Если msgtyp меньше 0, считывается первое сообщение из очереди, тип которого меньше либо равен абсолютному значению аргумента msgtyp.

В аргументе msgflg могут быть заданы 0 или более флагов, изменяющих работу системного вызова msgrcv.

- IPC_NOWAIT. Если флаг установлен, msgrcv возвращается немедленно с кодом ошибки ENOMSG, если сообщение требуемого типа в очереди отсутствует.
- MSG_EXCEPT. Используется вместе с положительным значением аргумента msgtyp для чтения первого сообщения в очереди, тип которого не равен msgtyp.
- MSG_NOERROR. Если флаг установлен, сообщение обрезается, если его длина превышает msgsz байт.

Если в очереди отсутствует сообщение требуемого типа, и флаг IPC_NOWAIT не был установлен в аргументе msgflg, процесс блокируется до наступления одного из следующих условий.

- Сообщение требуемого типа помещено в очередь.
- Очередь сообщений удалена. В этом случае системный вызов завершается с ошибкой EIDRM.

- Процесс получил и обработал сигнал. В этом случае системный вызов завершается с ошибкой EINTR. Системный вызов msgrcv никогда не перезапускается, если был прерван поступлением сигнала, даже если флаг SA_RESTART был задан при установке обработчика сигнала.

При успешном завершении системный вызов возвращает действительную длину сообщения, скопированного в поле mtext. При ошибке возвращается -1, а в переменную errno записывается код ошибки.

E2BIG	Длина сообщения больше, чем значение аргумента msgsz, а флаг MSG_NOERROR в аргументе msgflg не установлен.
EACCES	Процесс не имеет прав на чтение из данной очереди сообщений.
EFAULT	Недопустимый адрес передан в аргументе msgp.
EIDRM	Процесс был заблокирован на ожидании поступления сообщения, но очередь сообщений в это время была удалена.
EINTR	Процесс был заблокирован на ожидании поступления сообщений, но поступил и был обработан сигнал.
EINVAL	Недопустимое значение аргумента msgqid, или значение аргумента msgsz меньше 0.
ENOMSG	В аргументе msgflg установлен флаг IPC_NOWAIT, и в очереди сообщений нет сообщения требуемого типа.