Преимущества и недостатки потокового обмена данными

На предыдущем семинаре вы познакомились с механизмами, обеспечивающими потоковую передачу данных между процессами в операционной системе UNIX, а именно с pip'ами и FIFO. Потоковые механизмы достаточно просты в реализации и удобны для использования, но имеют ряд существенных недостатков:

- Операции чтения и записи не анализируют содержимое передаваемых данных. Процесс, прочитавший 20 байт из потока, не может сказать, были ли они записаны одним процессом или несколькими, записывались ли они за один раз или было, например, выполнено 4 операции записи по 5 байт. Данные в потоке никак не интерпретируются системой. Если требуется какая-либо интерпретация данных, то передающий и принимающий процессы должны заранее согласовать свои действия и уметь осуществлять её самостоятельно.
- Для передачи информации от одного процесса к другому требуется, как минимум, две операции копирования данных: первый раз – из адресного пространства передающего процесса в системный буфер, второй раз – из системного буфера в адресное пространство принимающего процесса.
- Процессы, обменивающиеся информацией, должны одновременно существовать в вычислительной системе. Нельзя записать информацию в поток с помощью одного процесса, завершить его, а затем, через некоторое время, запустить другой процесс и прочитать записанную информацию.

Понятие о System V IPC

Указанные выше недостатки потоков данных привели к разработке других механизмов передачи информации между процессами. Часть этих механизмов, впервые появившихся в UNIX System V и впоследствии перекочевавших оттуда практически во все современные версии операционной системы UNIX, получила общее название System V IPC (IPC – сокращение от InterProcess Communications). В группу System V IPC входят: очереди сообщений, разделяемая память и семафоры. Эти средства организации взаимодействия процессов связаны не только общностью происхождения, но и обладают схожим интерфейсом для выполнения подобных операций, например, для выделения и освобождения соответствующего ресурса в системе. Мы будем рассматривать их в порядке от менее семантически нагруженных с точки зрения операционной системы к более семантически нагруженным. Иными словами, чем позже мы начнем заниматься каким-либо механизмом из System V IPC, тем больше действий по интерпретации передаваемой информации придётся выполнять операционной системе при использовании этого механизма. Часть этого семинара мы посвятим изучению разделяемой памяти. Семафоры будут рассматриваться на семинарах 9-10, а очереди сообщений – на семинарах 11-12.

Пространство имён. Адресация в System V IPC. Функция ftok()

Все средства связи из System V IPC, как и уже рассмотренные нами pipe и FIFO, являются средствами связи с непрямой адресацией. Как мы установили на предыдущем семинаре, для организации взаимодействия неродственных процессов с помощью

средства связи с непрямой адресацией необходимо, чтобы это средство связи имело имя. Отсутствие имен у pip'ов позволяет процессам получать информацию о расположении pip'а в системе и его состоянии только через родственные связи. Наличие ассоциированного имени у FIFO – имени специализированного файла в файловой системе – позволяет неродственным процессам получать эту информацию через интерфейс файловой системы.

Множество всех возможных имён для объектов какого-либо вида принято называть пространством имён соответствующего вида объектов. Для FIFO пространством имён является множество всех допустимых имён файлов в файловой системе. Для всех объектов из System V IPC таким пространством имён является множество значений некоторого целочисленного типа данных – key_t – ключа. Причем программисту не позволено напрямую присваивать значение ключа, это значение задается опосредованно: через комбинацию имени какого-либо файла, уже существующего в файловой системе, и небольшого целого числа – например, номера экземпляра средства связи.

Такой хитрый способ получения значения ключа связан с двумя соображениями:

- Если разрешить программистам самим присваивать значение ключа для идентификации средств связи, то не исключено, что два программиста случайно воспользуются одним и тем же значением, не подозревая об этом. Тогда их процессы будут несанкционированно взаимодействовать через одно и то же средство коммуникации, что может привести к незапланированному поведению этих процессов. Поэтому основным компонентом значения ключа является числовое значение, сгенерированное на основе полного имени некоторого файла. Каждый программист имеет возможность использовать для этой цели свой специфический файл, например, исполняемый файл, связанный с одним из его взаимодействующих процессов. Следует отметить, что преобразование из текстового имени файла в число основывается на расположении указанного файла на жестком диске или ином физическом носителе. Поэтому для образования ключа следует применять файлы, для которых процессам разрешен доступ к информации об их расположении на диске (есть право «x» для директории, в которой расположен файл), и которые не меняют своего положения на диске в течение времени организации взаимодействия процессов.
- Второй компонент значения ключа используется для того, чтобы позволить программисту связать с одним и тем же именем файла более одного экземпляра каждого средства связи. В качестве такого компонента можно задавать порядковый номер соответствующего экземпляра.

Получение значения ключа из двух компонентов осуществляется функцией `ftok()`.

Необходимо ещё раз подчеркнуть три важных момента, связанных с использованием имени файла для получения ключа. Во-первых, необходимо указывать имя файла, который **уже существует** в файловой системе и для которого процесс **имеет право доступа на чтение его расположения на диске** (не путайте с заданием имени файла при создании FIFO, где указывалось имя **для вновь создаваемого** специального файла). Во-вторых, указанный файл должен **сохранять свое положение на диске** до тех пор, пока все процессы, участвующие во взаимодействии, не получат ключ System V IPC. В-третьих, задание имени файла, как одного из компонентов для получения ключа, ни в коем случае не означает, что информация, передаваемая с помощью

ассоциированного средства связи, будет располагаться в этом файле. Информация будет храниться **внутри адресного пространства операционной системы**, а заданное имя файла лишь позволяет различным процессам сгенерировать идентичные ключи.

Дескрипторы System V IPC

Мы говорили (Семинар 5, раздел «Файловый дескриптор»), что информацию о потоках ввода-вывода, с которыми имеет дело текущий процесс, в частности о pip'ах и FIFO, операционная система хранит в таблице открытых файлов процесса. Системные вызовы, осуществляющие операции над потоком, используют в качестве параметра индекс элемента таблицы открытых файлов, соответствующий потоку, – файловый дескриптор. Использование файловых дескрипторов для идентификации потоков внутри процесса позволяет применять к ним уже существующий интерфейс для работы с файлами, но в то же время приводит к автоматическому закрытию потоков при завершении процесса. Этим, в частности, объясняется один из перечисленных выше недостатков потоковой передачи информации.

При реализации компонентов System V IPC была принята другая концепция. Ядро операционной системы хранит информацию обо всех средствах System V IPC, используемых в системе, вне контекста пользовательских процессов. При создании нового средства связи или получении доступа к уже существующему процесс получает неотрицательное целое число – *дескриптор (идентификатор) этого средства связи*, который однозначно идентифицирует его во всей вычислительной системе. Этот дескриптор должен передаваться в качестве параметра всем системным вызовам, осуществляющим дальнейшие операции над соответствующим средством System V IPC.

Подобная концепция позволяет устранить один из самых существенных недостатков, присущих потоковым средствам связи – требование одновременного существования взаимодействующих процессов, но в то же время требует повышенной осторожности для того, чтобы процесс, получающий информацию, не принял взамен новых старые данные, случайно оставленные в механизме коммуникации.

Разделяемая память в UNIX. Системные вызовы shmget(), shmat(), shmdt()

С точки зрения операционной системы, наименее семантически нагруженным средством System V IPC является разделяемая память (shared memory). Для текущего семинара вам достаточно знать, что операционная система может позволить нескольким процессам совместно использовать некоторую область адресного пространства. Внутренние механизмы, позволяющие реализовать такое использование, будут подробно рассмотрены на лекции, посвящённой сегментной, страничной и сегментно-страничной организации памяти.

Все средства связи System V IPC требуют предварительных инициализирующих действий (создания) для организации взаимодействия процессов.

Для создания области разделяемой памяти с определённым ключом или доступа по ключу к уже существующей области применяется системный вызов `shmget()`. Существует два варианта его использования для создания новой области разделяемой памяти:

- Стандартный способ. В качестве значения ключа системному вызову поставляется значение, сформированное функцией `ftok()` для некоторого имени файла и номера экземпляра области разделяемой памяти. В качестве флагов поставляется комбинация прав доступа к создаваемому сегменту и флага `IPC_CREAT`. Если сегмент для данного ключа ещё не существует, то система будет пытаться создать его с указанными правами доступа. Если же вдруг он уже существовал, то мы просто получим его дескриптор. Возможно добавление к этой комбинации флагов флага `IPC_EXCL`. Этот флаг гарантирует нормальное завершение системного вызова только в том случае, если сегмент действительно был создан (т. е. ранее он не существовал), если же сегмент существовал, то системный вызов завершится с ошибкой, и значение системной переменной `errno` , описанной в файле errno.h, будет установлено в `EEXIST`.
- Нестандартный способ. В качестве значения ключа указывается специальное значение `IPC_PRIVATE`. Использование значения `IPC_PRIVATE` **всегда** приводит к попытке создания нового сегмента разделяемой памяти с заданными правами доступа и с ключом, который не совпадает со значением ключа ни одного из уже существующих сегментов и который не может быть получен с помощью функции `ftok()` ни при одной комбинации её параметров. Наличие флагов `IPC_CREAT` и `IPC_EXCL` в этом случае игнорируется.

Доступ к созданной области разделяемой памяти в дальнейшем обеспечивается её дескриптором, который вернет системный вызов `shmget()`. Доступ к уже существующей области также может осуществляться двумя способами:

- Если мы знаем её ключ, то, используя вызов `shmget()`, можем получить её дескриптор. В этом случае нельзя указывать в качестве составной части флагов флаг `IPC_EXCL`, а значение ключа, естественно, не может быть `IPC_PRIVATE`. Права доступа игнорируются, а размер области должен совпадать с размером, указанным при её создании.
- Либо мы можем воспользоваться тем, что дескриптор System V IPC действителен в рамках всей операционной системы, и передать его значение от процесса, создавшего разделяемую память, текущему процессу. Отметим, что при создании разделяемой памяти с помощью значения `IPC_PRIVATE` – это единственно возможный способ.

Для старых версий UNIXа характерно следующее поведение вызова shmget(): если сегмент уже существует, но размер не совпадает с параметром size, то констатируется ошибка. Для подавляющего большинства современных систем характерно другое поведение: ошибка констатируется только если размер существующей памяти меньше параметра size.

После получения дескриптора необходимо включить область разделяемой памяти в адресное пространство текущего процесса. Это осуществляется с помощью системного вызова `shmat()`. При нормальном завершении он вернёт адрес разделяемой памяти в адресном пространстве текущего процесса. Дальнейший доступ к этой памяти осуществляется с помощью обычных средств языка программирования.

После окончания использования разделяемой памяти процесс может уменьшить размер своего адресного пространства, исключив из него эту область с помощью системного вызова `shmdt()`. Отметим, что в качестве параметра системный вызов `shmdt()` требует адрес начала области разделяемой памяти в адресном пространстве процесса, т.

е. значение, которое вернул системный вызов `shmat()`, поэтому данное значение следует сохранять на протяжении всего времени использования разделяемой памяти.

Для иллюстрации использования разделяемой памяти рассмотрим две взаимодействующие программы: **07-1a.c** и **07-1b.c**.

Эти программы очень похожи друг на друга и используют разделяемую память для хранения числа запусков каждой из программ и их суммы. В разделяемой памяти размещается массив из трех целых чисел. Первый элемент массива используется как счетчик для программы 1, второй элемент – для программы 2, третий элемент – для обеих программ суммарно. Дополнительный нюанс в программах возникает из-за необходимости инициализации элементов массива при создании разделяемой памяти. Для этого нам нужно, чтобы программы могли различать случай, когда они создали её, и случай, когда она уже существовала. Мы добиваемся различия, используя вначале системный вызов `shmget()` с флагами `IPC_CREAT` и `IPC_EXCL`. Если вызов завершается нормально, то мы создали разделяемую память. Если вызов завершается с констатацией ошибки и значение переменной errno равняется EEXIST, то, значит, разделяемая память уже существует, и мы можем получить её IPC-дескриптор, применяя тот же самый вызов с нулевым значением флагов. *Откомпилируйте эти программы и запустите несколько раз. Проанализируйте полученные результаты.*

Дополнительное замечание. Третий счётчик необходим будет дальше для иллюстрации наличия race condition, чтобы доказать, что программы **07-1a** и **07-1b** на самом деле написаны некорректно.

Команды ipcs и ipcrm

Как вы видели из предыдущего примера, созданная область разделяемой памяти сохраняется в операционной системе даже тогда, когда нет ни одного процесса, включающего её в своё адресное пространство. С одной стороны, это имеет определённые преимущества, поскольку не требует одновременного существования взаимодействующих процессов, с другой стороны, может причинять существенные неудобства. Допустим, что предыдущие программы мы хотим использовать таким образом, чтобы подсчитывать количество запусков в течение одного, текущего, сеанса работы в системе. Однако в созданном сегменте разделяемой памяти остаётся информация от предыдущего сеанса, и программы будут выдавать общее количество запусков за всё время работы с момента загрузки операционной системы. Можно было бы создавать для нового сеанса новый сегмент разделяемой памяти, но количество ресурсов в системе не безгранично. Нас спасает то, что существуют способы удалять неиспользуемые ресурсы System V IPC как с помощью команд операционной системы, так и с помощью системных вызовов. Все средства System V IPC требуют определённых действий для освобождения занимаемых ресурсов после окончания взаимодействия процессов. Для того чтобы удалять ресурсы System V IPC из командной строки, нам понадобятся две команды, `ipcs` и `ipcrm`.

Команда `ipcs` выдает информацию обо всех средствах System V IPC, существующих в системе, для которых пользователь обладает правами на чтение: областях разделяемой памяти, семафорах и очередях сообщений.

Из всего многообразия выводимой информации вас будут интересовать только IPC идентификаторы для средств, созданных вами. Эти идентификаторы будут

использоваться в команде ipcrm, позволяющей удалить необходимый ресурс из системы. Для удаления сегмента разделяемой памяти эта команда имеет вид

```
$ ipcrm shm <IPC идентификатор>
```

Удалите созданный вами сегмент разделяемой памяти из операционной системы, используя эти команды.

Если поведение программ, использующих средства System V IPC, базируется на предположении, что эти средства были созданы при их работе, не забывайте перед их запуском удалять уже существующие ресурсы.

Использование системного вызова shmctl() для освобождения ресурса

Для той же цели – удалить область разделяемой памяти из системы – можно воспользоваться и системным вызовом `shmctl()`. Этот системный вызов позволяет полностью ликвидировать область разделяемой памяти в операционной системе по заданному дескриптору средства IPC, если, конечно, у вас хватает для этого полномочий. Системный вызов `shmctl()` позволяет выполнять и другие действия над сегментом разделяемой памяти, но их изучение лежит за пределами нашего курса.

Разделяемая память и системные вызовы fork(), exec() и функция exit()

Важным вопросом является поведение сегментов разделяемой памяти при выполнении процессом системных вызовов `fork()`, `exec()` и функции `exit()`.

При выполнении системного вызова `fork()` все области разделяемой памяти, размещённые в адресном пространстве процесса, наследуются порожденным процессом.

При выполнении системных вызовов `exec()` и функции `exit()` все области разделяемой памяти, размещенные в адресном пространстве процесса, исключаются из его адресного пространства, но продолжают существовать в операционной системе.

Задачи на семинар

***Задача 1 (10 баллов)**:*

Напишите две программы, осуществляющие взаимодействие через разделяемую память. Первая программа (писатель) должна создавать сегмент разделяемой памяти и копировать туда собственный исходный текст, вторая программа (читатель) должна брать оттуда этот текст, печатать его на экране и удалять сегмент разделяемой памяти из системы.

Для определения размера файла можно воспользоваться системным вызовом lseek (аккуратный вариант), также допустимо просто выделить память с запасом.

Использование библиотечных функций при работе с файлами запрещено, пользуйтесь системными вызовами open, read, write, close, изученными на предыдущих семинарах. Именно тот факт, что вам необходимо вспомнить работу с файлами и считать в буфер файл достаточно большого объема, делает эту задачу задачей на 10 баллов, а не на 5.

***Задача 1* (1 бонусный балл)**:

Его вы получите, если в вашей программе нет лишних операций копирования.