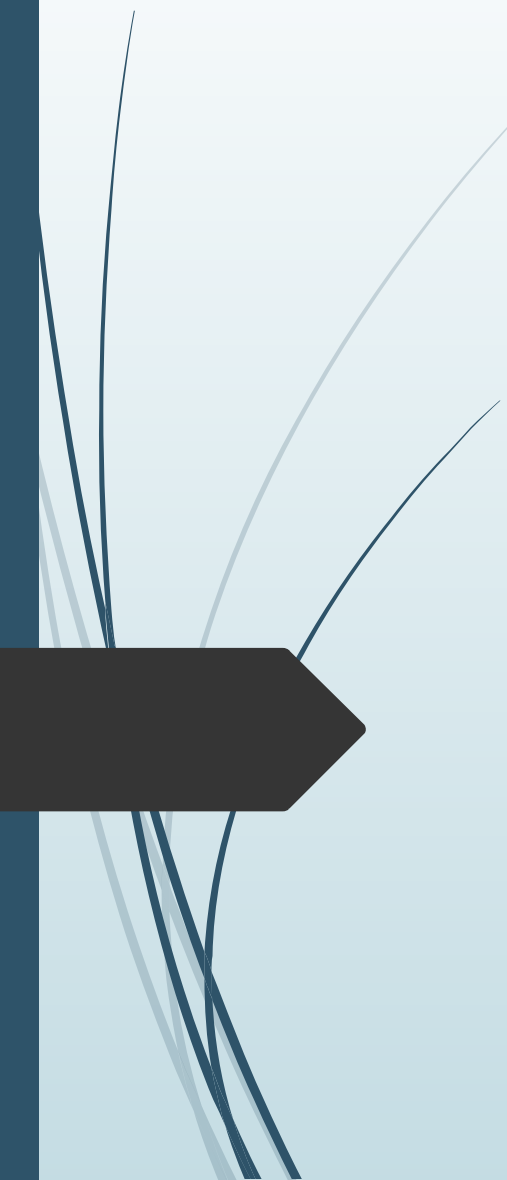


Inter-Process Communication (IPC)

Inter-Process Communication

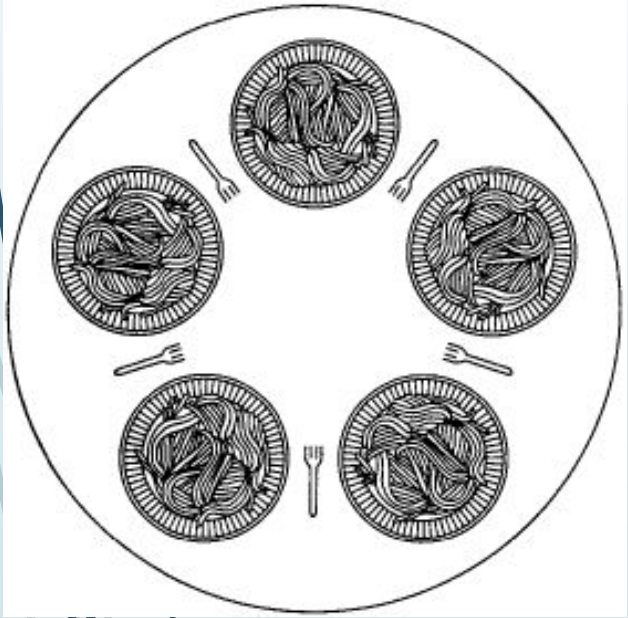
- Межпроцессное взаимодействие — обмен данными между потоками одного или разных процессов
- Реализуется посредством механизмов, предоставляемых ядром ОС или процессом, использующим механизмы ОС и реализующим новые возможности IPC

История

- В историческом плане сначала появилась необходимость в общении процессов, выполняющихся на одном компьютере
- С бурным развитием сетевых технологий появилась потребность в средствах для взаимодействия процессов, выполняющихся на разных компьютерах в сети
- Затем – для компьютеров на базе разных платформ и/или с разными операционными системами

Проблема обедающих философов

В 1965 году Дейкстра сформулировал и решил следующую проблему синхронизации:



- Пять философов сидят за круглым столом, и у каждого есть тарелка со спагетти
- Спагетти настолько скользкие, что каждому философу нужно две вилки, чтобы с ними управиться
- Между каждыми двумя тарелками лежит одна вилка
- Жизнь философа состоит из чередующихся периодов поглощения пищи и размышлений
- Когда философ голоден, он пытается получить две вилки, левую и правую, в любом порядке
- Если ему удалось получить две вилки, он некоторое время ест, затем кладет вилки обратно и продолжает размышления

Вопрос : можно ли написать алгоритм, который моделирует эти действия для каждого философа и никогда не застревает?

Проблематика

IPC призваны решать проблемы, возникающие при организации параллельных вычислений:

- Передача данных между процессами
  Адресные пространства процессов в операционных системах изолированы друг от друга, но иногда коммуникации нужны
- Синхронный доступ
  Если множество процессов считывают общие данные, то при попытке одним из процессов изменить эти данные, может возникнуть ситуация гонки (race conditions)
- Дисциплина доступа
  Если нужно, чтобы большое количество процессов могли записывать данные, организуется очередь по правилу «один пишет, несколько читают».
  Практически организуется две очереди: одна - для чтения, другая - для записи.
  Такую дисциплину доступа можно организовать с помощью барьеров (блокировок)
- Тупик (deadlock)
  Классический тупик возникает, если процесс А получает доступ к ресурсу А и ждет освобождения ресурса В. Одновременно процесс В, получив доступ к ресурсу В, ждет освобождения ресурса А. Оба процесса бесконечно ждут освобождения ресурсов

Общие принципы реализации

- Все IPC (кроме общей памяти) ведутся через посредника – операционную систему(-ы)
- В основе локальных IPC обычно находится разделяемый *ресурс* (например, канал или сегмент разделяемой памяти), и, следовательно, ОС должна предоставить средства для генерации, именования, установки режима доступа и атрибутов защиты таких ресурсов. Обычно такой ресурс может быть доступен всем процессам, которые знают его имя и имеют необходимые привилегии

Общие принципы реализации

Организация связи между процессами предполагает установления таких ее характеристик, как:

- направление связи
  Связь бывает однонаправленная (симплексная) и двунаправленная (полудуплексная для поочередной передачи информации и дуплексная с возможностью одновременной передачи данных в разных направлениях);
- тип адресации
  В случае прямой адресации информация посылается непосредственно получателю, например, процессу P-Send (P, message). В случае непрямой или *косвенной адресации* информация помещается в некоторый промежуточный объект, например, в почтовый ящик;
- используемая модель передачи данных - потоковая или модель сообщений;
- объем передаваемой информации и сведения о том, обладает ли канал буфером необходимого размера;
- синхронность обмена данными
  Если отправитель сообщения блокируется до получения этого сообщения адресатом, то обмен считается синхронным, в противном случае - асинхронным.

Кроме перечисленных у каждого вида связи есть еще ряд особенностей

Виды

По семантике назначения и использования механизмы, предоставляемых ОС и используемые для IPC, можно разделить на:

- механизмы разделения памяти
- механизмы синхронизации
- механизмы обмена сообщениями
- механизмы удалённых вызовов (RPC)

Разделяемая память

- Техника разделяемой памяти позволяет осуществлять обмен информацией через общий для процессов сегмент памяти без использования системных вызовов ядра.
- Сегмент разделяемой памяти подключается в свободную часть виртуального адресного пространства процесса.
- Таким образом, два разных процесса могут иметь разные адреса одной и той же ячейки подключенной разделяемой памяти.

Механизмы разделения памяти

- Буфер обмена (clipboard)

Это одна из самых примитивных и хорошо известных форм IPC. Он появился в ранних версиях Windows. Основная его задача - обеспечивать обмен данными между программами по желанию и под контролем пользователя. Не рекомендуется использовать его для внутренних нужд приложения и помещать туда то, что не предназначено для прямого просмотра пользователем

- Отображение файлов на оперативную память (File Mapping)

Этот механизм позволяет осуществлять доступ к файлу таким образом, как будто это обыкновенный массив, хранящийся в памяти (не загружая файл в память явно). "Побочным эффектом" этой технологии является возможность работать с таким отображенным файлом сразу нескольким процессам. Таким образом, можно создать объект file mapping, но не ассоциировать его с каким-то конкретным файлом. Получаемая область памяти как раз и будет общей между процессами. Работая с этой памятью, потоки обязательно должны согласовывать свои действия с помощью объектов синхронизации.

- Библиотеки динамической компоновки (DLL)

Библиотеки динамической компоновки также имеют способность обеспечивать обмен данными между процессами. Когда в рамках DLL объявляется переменная, ее можно сделать разделяемой (shared). Все процессы, обращающиеся к библиотеке, для таких переменных будут использовать одно и то же место в физической памяти (Здесь также важно не забыть о синхронизации)

Механизмы синхронизации

- мьютексы
- семафоры
- критические секции

Критическая область

- Критическая область (КО) — это примитив для синхронизации нескольких потоков одного процесса. Критической областью защищают участок кода, который не должен одновременно выполняться несколькими потоками
- Объект-критическую область необходимо инициализировать перед использованием один раз из произвольного потока вызовом функции InitializeCriticalSection()
- Перед обращением к защищенному участку все потоки должны попытаться захватить критическую область (говорят: войти в неё) вызовом EnterCriticalSection(). Если данную КО уже захватил другой поток, текущий войдет в состояние ожидания, пока КО освободится вызовом LeaveCriticalSection() из владеющего КО потока. Обеим функциям передается переменная типа CRITICAL_SECTION; при доступе к критической области все потоки должны использовать одну и ту же переменную

Мьютекс (mutex – mutual execution)

- Мьютекс — аналог критической области, позволяющий синхронизировать потоки в разных процессах
- Мьютексам присваиваются имена при создании. Сначала поток одного из процессов создает мьютекс функцией CreateMutex(); потоки остальных процессов получают доступ к этому мьютексу функций OpenMutex() по известному имени мьютекса
- Захват мьютекса может выполняться функциями ожидания. Завершив работу с захваченным мьютексом, поток может освободить его функцией ReleaseMutex().
- Прекращение доступа к мьютексу выполняется функцией CloseHandle(). На каждый вызов CreateMutex() или OpenMutex() должно приходиться по вызову CloseHandle() с тем же дескриптором. Мьютекс уничтожается, когда закрывается последний дескриптор, связанный с ним.

Семафоры

- Семафор позволяет ограничить число потоков-пользователей общего ресурса. Определение введено Э. Дейкстрой в 1963 году.
- Основные операции - «DOWN» и «UP»
- Семафоры Windows API создаются функцией CreateSemaphore(), которая позволяет указать начальное и максимальное значение счетчика семафора (число одновременных пользователей общего ресурса). Если необходим доступ к семафору из разных процессов, можно задать имя семафора, а затем получать его дескриптор в других процессах функцией OpenSemaphore().
- Захват семафора (действие «DOWN») выполняется функциями ожидания. Освобождение семафора (действие «UP») производится функцией ReleaseSemaphore(), с помощью которой можно также определить значение счетчика.
- Прекращение доступа к семафорам и их уничтожение выполняется аналогично мьютексам

Механизмы обмена сообщениями

- **Каналы (pipes)**

Каналы - это очень мощная технология обмена данными. В общем случае канал можно представить в виде трубы, соединяющей два процесса. Что попадает в трубу на одном конце, мгновенно появляется на другом. Чаще всего каналы используются для передачи непрерывного потока данных.

Каналы делятся на неименованные (anonymous pipes) и именованные (named pipes). Анонимные каналы используются достаточно редко, они просто передают поток вывода одного процесса на поток ввода другого. Именованные каналы передают произвольные данные и могут работать через сеть.

- **Сокеты (sockets)**

Это очень важная технология, т.к. именно она отвечает за обмен данными в Интернет. Сокеты также часто используются в крупных ЛВС. Взаимодействие происходит через разъемы-"сокеты", которые представляют собой абстракцию конечных точек коммуникационной линии, соединяющей два приложения. С этими объектами программа и должна работать, например, ждать соединения, посылать данные и т.д.

- **Очереди сообщений (message queue) и почтовые слоты (mailslots)**

Это механизм однонаправленного IPC. Если приложению известно имя очереди, оно может помещать туда сообщения, а приложение-хозяин этой очереди (приемник) может их оттуда извлекать и соответствующим образом обрабатывать. Основное преимущество этого способа - возможность передавать сообщения по локальной сети сразу нескольким компьютерам за одну операцию. Для этого приложения-приемники создают почтовые слоты с одним и тем же именем. Когда в дальнейшем какое-либо приложение помещает сообщение в этот слот, приложения-приемники получают его одновременно.

Именованные каналы (named pipes)

- Одна программа, называемая, создает именованный канал. Другие программы, называемые клиентами, подключаются к каналу, указывая его имя
- В любой момент времени сервер может быть соединен только с одним клиентом (по одному каналу). Далее и сервер, и клиент могут работать с каналом как с файлом
- Любую порцию данных может вычитать только один клиент; сервер считывает все записанные данные от всех клиентов
- Каналы могут быть ориентированы на передачу байт или сообщений
- В случае сообщений передача выполняется неделимыми порциями байт, и не нужно обрабатывать случаи, когда вычитана лишь часть сообщения; размер сообщений ограничен 64 КБ. Кроме того, доступны специальные функции, упрощающие и ускоряющие передачу. Сообщения могут вычитываться и побайтово.

Неименованные каналы (anonymous pipes)

- Анонимные каналы (anonymous pipes или просто pipes) работают идентично именованным, однако не имеют связанного с ними имени и пути в файловой системе
- Как следствие, невозможно открыть анонимный канал, созданный другим процессом, так как без имени канала нельзя его идентифицировать. Используются для передачи данных между родительскими и дочерними процессами
- При создании анонимного канала функцией CreatePipe() процесс получает описатель конца канала — анонимный канал всегда однонаправленный
- При создании нового процесса функцией CreateProcess() можно указать в полях hStdIn, hStdOut и hStdErr структуры STARTUPINFO, какие дескрипторы, полученные создающим (родительским) процессом, использовать в качестве стандартных потоков порождаемого (дочернего) процесса. Так, можно заменить стандартный поток дочернего процесса концом анонимного канала (или даже два потока — одним концом)

Сокеты (sockets)

- Сокеты обеспечивают двухстороннюю связь типа «точка-точка» между двумя процессами
- Являются основными компонентами межсистемной и межпроцессной связи
- Каждый сокет представляет собой конечную точку связи, с которой может быть совмещено некоторое имя. Он имеет определенный тип, и один процесс или несколько, связанных с ним процессов

Очереди сообщений (message queue)

- В ОС Windows механизм очередей сообщений реализуется почтовыми ящиками (mailslots)
- Процесс, создавший почтовый ящик функцией CreateMailslot(), считается сервером. Только он может получать сообщения из почтового ящика функцией ReadFile()
- Отправители, или клиенты, — любые другие процессы — записывают сообщения в почтовый ящик функцией WriteFile(); сервер же отправлять сообщения в созданный ящик не может
- Количество сообщений в ящике позволяет определить функция GetMailslotInfo()
- Сообщения не имеют адресатов и адресантов, то есть нельзя установить процесс-отправитель при получении (если этого не указать в самом сообщении)
- Почтовые ящики доступны по локальной сети, то есть через почтовый ящик можно передавать сообщения между процессами на разных компьютерах. Доставка сообщений при этом, однако, не гарантируется, а их размер ограничен

Реализации

- В разных ОС реализации описанных механизмов могут отличаться
- Существует стандарт POSIX (**p**ortable **o**perating **s**ystem **i**nterface) для UNIX-подобных систем, описывающий в том числе интерфейс ОС для работы с потоками, однако его поддержка – добровольна)
  - POSIX-сертифицированные: Mac OS X, Solaris, LynxOS, UnixWare, …
  - POSIX-совместимые: FreeBSD, OpenSolaris, OpenVMS , …
  - По большей части POSIX-совместимые: Linux, NetBSD, OpenBSD, Symbian OS, …
  - Использующие основные принципы: Windows, …