CQ 21.

10 апреля 2023 г.

1 Какие средства используются для организации межпроцессного взаимодействия?

Межпроцессное взаимодействие (IPC) – это механизм, который позволяет процессам взаимодействовать друг с другом и синхронизировать свои действия. Реализуется посредством механизмов, предоставляемых ядром ОС или процессом, использующим механизмы ОС и реализующим новые возможности IPC. Для организации IPC используются (рассматриваем IPC в рамках одной ОС):

- Shared memory (разделяемая память) (см. подробности в следующем вопросе).
- Memory-mapped files (см. подробности в следующем вопросе).
- Messaging passing (обмен сообщениями). Два взаимодействующих процесса устанавливают канал связи и обмениваются сообщениями с помощью базовых примитивов: send() и receive(). Как правило, организация отправления сообщений основана на FIFO-принципе.

P.s. В конспекте есть также "запись на диск пакетами или после flush"(дополнительный материал).

2 Чем файлы, отображаемые в память, отличаются от разделяемой памяти?

Shared memory (разделяемая память) – это самый быстрый способ IPC, т.к. обмен данными осуществляется через общую для процессов часть памяти без использования системных вызовов ядра. Состоит в том, что сегмент разделяемой памяти создаётся и подключается ОС в свободную часть АП указанного процесса. Однако, shared memory требует реализации синхронизации для различных процессов.

Memory-mapped files (файлы, отображаемые в память) подразумевают, что часть АП процесса ассоциируется с некоторым существующим файлом, а затем данный файл может быть использован для IPC и работы с самим файлом. Однако поскольку управление файлом происходит на уровне filesystem, то данное средство организации IPC является более медленным.

3 Что необходимо учитывать при создании контейнеров в разделяемой памяти?

При создании контейнеров разделяемой памяти (или отображаемых в память файлов) при использовании средств Boost.Interprocess необходимо учитывать, что они, не могут предполагать ни одного из следующих соглашений/ограничений Стандарта (для обычных контейнеров):

- допустимости, что typedef указателя на аллокатор для каждого аллокатора является синонимом T*.

- допустимости, что все объекты-аллокаторы одного и того же типа эквивалентны и всегда являются равными.

Первое недопустимо, т.к. иначе нельзя будет использовать умные указатели как allocator::pointer. Второй недопустим, т.к. аллокаторы одного типа, выделяющие память в разных адресных пространствах, будут считаться одинаковыми.

Поэтому контейнеры, которые мы хотим поместить в разделяемую память или отображаемые в память файлы (при использовании Boost.Interprocess) таковы, что:

- НЕ могут предполагать, что память, выделенная с помощью аллокатора, может быть освобождена другими аллокаторами того же типа. Все объекты-аллокаторы должны совпадать только в том случае, если память, выделенная для одного объекта, может быть освобождена другим, что следует проверять лишь использованием operator==() во время выполнения.

- Их внутренние указатели должны иметь тип allocator::pointer, и контейнеры не могут предполагать, что allocator::pointer является необработанным указателем (не является умным указателем).

- Все контейнерные объекты должны быть сконструированы/уничтожены с помощью функций allocator::construct/allocator::destroy.

Итак, контейнеры Boost.Interprocess размещаются в разделяемой памяти или отображаемых в память файлых и т.д. с использованием двух механизмов:

- Boost.Interprocess construct<>, find_or_construct<>... функций. Эти функции размещают объекты C++ в разделяемой памяти или отображаемых в память файлых. Но они размещают лишь объекты, но не другие данные, которые могут быть динамически аллоцированы уже самими объектами.

- Shared memory allocators (аллокаторы разделяемой памяти). Они позволяют выделять разделяемую память или части отображаемых в память файлов так, что созданные контейнеры могут динамически аллоцировать память для хранения новых и новых элементов.

Это означает что для размещения какого-либо Boost.Interprocess контейнера в разделяемой памяти или в отображаемых в память файлых, контейнер должен:

- Вывести тип шаблонного параметра аллокатора как некоторого Boost.Interprocess аллокатора.

- Принимать в качестве параметра Boost.Interprocess аллокатор.

- Использовать construct<>/find_or_construct<>... функции для размещения контейнера в управляемой памяти.

4 Чем отличаются анонимные и именованные примитивы синхронизации?

При использовании именованных примитивов синхронизации для создания какого-либо объекта и последующей работы с ним процессы используют его имя. Это похоже на создание или

открытие файлов: например, процесс создает файл, используя класс fstream с именем файла filename, а другой процесс открывает этот файл, используя другой процесс fstream с тем же аргументом имени файла. Т.о., каждый процесс использует свой объект для доступа к ресурсу, но оба процесса используют один и тот же базовый ресурс. Работой же именованных принципов синхронизации управляет ОС. Преимущество именованных утилит состоит в том, что их легче обрабатывать для простых задач синхронизации, так как обоим процессам не нужно создавать разделяемую область памяти и создавать там механизм синхронизации.

При использовании анонимных примитивов синхронизации процессы совместно используют один и тот же объект через общую память или файлы отображения памяти (т.е. объекты хранятся в разделяемой памяти или в файлых отображения памяти). Это похоже на традиционные объекты синхронизации потоков (например, мьютексы): оба процесса совместно используют один и тот же объект. Но в отличие от синхронизации потоков, где, например, глобальные переменные и память кучи могут совместно использоваться потоками одного и того же процесса, совместное использование объектов между двумя потоками из разных процессов возможно только через сопоставленные области, которые отображают один и тот же сопоставляемый ресурс (например, общую память или отображаемые файлы памяти). Анонимные утилиты могут быть сериализованы на диск при использовании отображаемых в память объектов, обеспечивающих автоматическое сохранение утилит синхронизации. Можно создать утилиту синхронизации в файле отображения памяти, перезагрузить систему, снова отобразить файл и снова использовать утилиту синхронизации без каких-либо проблем. Этого нельзя добиться с помощью именованных утилит синхронизации.

Основное различие интерфейса между именованными и анонимными примитивами синхронизации заключается в конструкторах. Обычно анонимные утилиты имеют только один конструктор, тогда как именованные утилиты имеют несколько конструкторов, первый аргумент которых представляет собой специальный тип, который запрашивает создание, открытие или открытие или создание базового ресурса.

5 Как могут быть организованы библиотеки динамической компоновки DLL?

Dynamic Link Library (DLL) – библиотека динамической компоновки / динамически подключаемая библиотека – динамическая библиотека, позволяющая многократное использование кода или данных различными программными приложениями. Практически невозможно создать приложение Windows, в котором не использовались бы библиотеки DLL. В DLL содержатся все функции Win32 API и несчётное количество других функций операционных систем Win32.

Библиотеки DLL выполняются в пространстве памяти вызывающего процесса и с теми же правами доступа, что означает, что при их использовании возникают небольшие накладные расходы, но также и то, что вызывающая программа не защищена, если в библиотеке DLL есть какая-либо ошибка. Иными словами, DLL – это просто наборы функций, собранные в библиотеки. Однако, в отличие от своих статических родственников (файлов .lib), библиотеки DLL не присоединены непосредственно к выполняемым файлам с помощью редактора связей. В выполняемый файл занесена только информация об их местонахождении.

Чаще всего проект подключается к DLL статически, или неявно, на этапе компоновки. Загрузкой DLL при выполнении программы управляет ОС. Однако, DLL можно загрузить и явно, или динамически, в ходе работы приложения.

Некоторые преимущества DLL:

- Использование меньшего количества ресурсов.
- Модульность архитектуры.
- Упрощение развёртывания и установки.

Эти преимущества позволяют создавать и использовать DLL патчи, которые способствуют оперативному или динамическому обновлению программного обеспечения, представляющему собой применение исправлений без выключения и перезапуска системы или соответствующей программы.

Тем не менее библиотеки DLL являются ядром архитектуры Windows, у них есть несколько недостатков, которые в совокупности составляют т.н. DLL Hell.

Список литературы

[1] Конспект. И.С. Макаров.

[2] https://www.boost.org/doc/libs/1_81_0/doc/html/interprocess.html#interprocess.intro

[3] https://learn.microsoft.com/en-us/troubleshoot/windows-client/deployment/dynamic-link-library

[4] https://en.wikipedia.org/wiki/Dynamic-link_library

[5] http://www.cyberguru.ru/programming/visual-cpp/visual-cpp-beginners.html?showall=&start=29