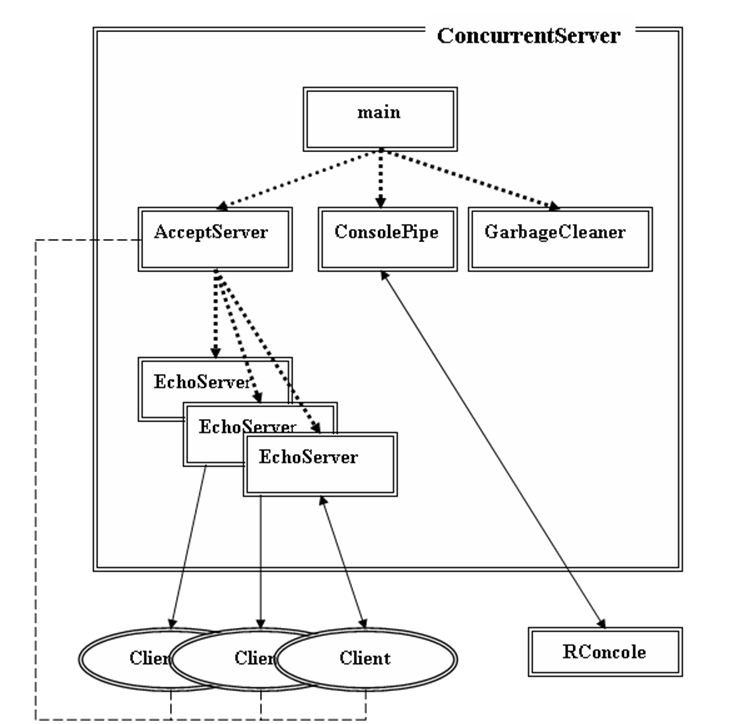
1. **Итеративный и параллельный сервер**

Итеративные серверы, используют для связи с клиентами протокол UDP и используются в тех случаях, когда предполагается, что запросы клиентов являются редкими, а исполнении их не требует много времени. В каждый момент времени итеративным сервером всегда обслуживается только один клиент.

В параллельных серверах с каждым клиентом устанавливается TCP-соединение. Одновременно к одному параллельному серверу может быть подключено несколько клиентов.

1. **Структура параллельного сервера**



**Процесс main**. Основным назначением процесса main, является запуск, инициализация и завершение работы сервера. Как уже отмечалось, именно этот процесс первым получает управление от операционной системы. Процесс main запускает основные процессы: AcceptServer, ConsolePipe и RConsole.

**Процесс AcceptServer**. AcceptServer создается процессом main и предназначен для выполнения процедуры подключения клиентов к серверу, для исполнения команд консоли управления, а также для запуска процессов EchoServer, обслуживающих запросы клиентских программ по созданным соединениям. Кроме того, AcceptServer создает список подключений, который далее будем называть ***ListContanct***. При подключении очередного клиента, процесс AcceptServer добавляет в ListContact элемент, предназначенный для хранения информации о состоянии данного подключения.

**Процесс ConsolePipe**.ConsolePipe создается процессом main и является сервером именованного канала, по которому осуществляется связь между программой RConsole (консоль управления сервером) и параллельным сервером.

**Процесс** **GarbageCleaner**. Основным назначением процесса GarbageCleaner является удаление элемента списка подключений ListContact, после отключения программы клиента. Следует сразу отметить, что ListContact является ресурсом, требующим последовательного использование. Одновременная запись и (или) удаление элементов списка может привести к разрушению списка ListContact.

**Процесс EchoServer**. Процессы EchoServer создаются процессом AcceptServer по одному для каждого успешного подключения программы клиента. Основным назначением процесса EchoServer является прием данных по созданному процессом AcceptServer подключению и отправка этих же данных без изменения обратно программе клиента. Условием окончания работы сервера является получение от клиента пустого сегмента данных (имеющего нулевую длину).

**Программа Client**. Программа Client предназначена для пересылки данных серверу и получения ответа от сервера. Программа может работать, как на одном компьютере с сервером (будет использоваться интерфейс внутренней петли), так и на другом компьютере, соединенным с компьютером сервера сетью TCP/IP. Для окончания работы с сервером программа формирует и отправляет сегмент данных нулевой длины.

**Программа RConsole**. Программа RConsole предназначена для ввода команд управления сервером и для вывода диагностических сообщений полученных от сервера . RConsole является клиентом именованного канала.

**Список подключений ListContact**. Список ListContact (не изображен на рисунке) создается основе стандартного класса list и предназначен для хранения информации о каждом подключении. Список создается пустым при инициализации процесса AcceptServer. В рамках этого же процесса осуществляется добавление элементов списка, по одному для каждого подключения. При отключении программы клиента от сервера, соответствующий элемент списка помечается, как неиспользуемый. Удаление неиспользуемого элемента осуществляется процессом GarbageCleaner, который работает в фоновом режиме.

Описанная выше модель распределенного приложения, по мнению автора, является достаточно полной для того, чтобы изложить основные принципы создания параллельного сервера.

1. **Что такое поток управления**

Поток управления в Windows является объектом ядра операционной системы, которому выделяется процессорное время для выполнения приложенияd

1. **Что такое главный/первичный поток. основные функции для работы с потоками(5 шт)**

это основной поток выполнения программы. Когда программа запускается, создается главный поток, который начинает выполнение с первой строки кода и последовательно выполняет инструкции, пока не завершит свою работу или не создаст дополнительные потоки.

|  |  |
| --- | --- |
| **Наименование**  **функции** | **Назначение** |
| **CreateThread** | Создать поток |
| **ResumeThread** | Возобновить поток |
| **SuspendThread** | Приостановить поток |
| **Sleep** | Задержать исполнение |
| **TermitateThread** | Завершить поток |

1. **какие два механизма предназначены для организации параллельной работы программ в ОС**

*механизм* ***потоков*** и *механизм* ***процессов*** .

1. **параллельные потоки,многопоточная программа,однопоточная программа**

программа является ***многопоточной***, если в ней существуют одновременно несколько потоков управления. Сами потоки в этом случае называются ***параллельными***. Если в программе может существовать только один поток, то такая программа называется ***однопоточной***.

1. **безопасность для потока, какие два свойства есть (блокировка и реентерабельность)**

Безопасная для потоков функции в обладает двумя свойствами: 1) свойством ***реентерабельности***; 2) функция обеспечивает блокировку доступа к критическим ресурсам, которые она использует.   
 В общем случае функция называется **реентерабельной**, если она не изменяет собственный код или собственные статические данные. Другими словами программный код реентерабельной функции должен допускать корректное его использование несколькими потоками одновременно.

**Блокировка** требуется в том случае, если функцией используется ресурс, доступ к которому может быть только упорядоченным (критический ресурс). Примером критического ресурса может служить изменяемые функцией статические и глобальные переменные.

1. **какие ресурсы принадлежат потоку(образуют контекст потока)**

* код исполняемой функции;
* набор регистров процессора;
* область оперативной памяти;
* стек для работы приложения;
* стек для работы операционной системы;
* маркер доступа, содержащий информацию для системы безопасности.

1. **Что такое процесс**

Под ***процессом*** операционной системы Window понимается объект ядра, которому принадлежат системные ресурсы, используемые исполняемым приложением операционной системы

1. **синхронизация сервера**

С некоторыми механизмами синхронизации используемыми в Windows мы уже познакомились, когда говорили о функции WaitSingleObject, которая использовалась для ожидания окончания работы потока. Еще один рассмотренный способ синхронизации – приостановка потока с помощью функции SuspendThread.

критические секции, мьютексы, события, семафоры, ожидающий таймер и т.д.

1. **Критическая секция?5 функций**

Критические секции, является одним из самых простых механизмов синхронизации и в нашей модели могут быть использованы для исключения совместного использования списка ListContact потоками AcceptServer и GarbageCleaner. Критическая секция является объектом операционной системы типа CRITICAL\_SECTION

|  |  |
| --- | --- |
| **Наименование**  **функции** | **Назначение** |
| **DeleteCriticalSection** | Разрушить критическую секцию |
| **EnterCriticalSection** | Войти в критическую секцию |
| **InitializeCriticalSection** | Инициализировать критическую секцию |
| **LeaveCriticalSection** | Покинуть критическую секцию |
| **TryEnterCriticalSection** | Пытаться войти в критическую секцию |

1. **асинхронная процедура**

***Асинхронной процедурой*** называется функция, которая выполняется асинхронно в контексте какого-нибудь потока. Для исполнения асинхронной процедуры необходимо определить асинхронную процедуру, указать поток, в контексте которого она будет выполняться, и дать разрешение на выполнение асинхронной процедуры.

|  |  |
| --- | --- |
| **Наименование**  **функции** | **Назначение** |
| **QueueUserAPC** | Поставить асинхронную процедуру в очередь |
| **SleepEx** | Приостановит поток для выполнения асинхронных процедур |

1. **ожидающий таймер**

***Ожидающим таймером*** в Windows, называется объект синхронизации, который переходит в ***сигнальное состояние*** при наступлении заданного момента времени.

1. **какие три состояния у него есть**

**сигнальное, активное, пассивное**

1. **сигнальный, активный, пассивный что значит**

переходит в ***сигнальное состояние*** при наступлении заданного момента времени. Если ожидающий таймер ждет момента перехода в сигнальное состояние, то говорят, что он находится в ***активном состоянии***. Другое состояние ожидающего таймера ***пассивное*** – из этого состояния он не может перейти в сигнальное состояние.

1. **таймер с ручным и автоматическим сбросом что значит?5 функций для таймера**

По способу перехода из сигнального стояния в несигнальное, ожидающие таймеры разделяются на ***таймеры с ручным сбросом*** и таймеры с ***автоматическим сбросом***, иначе называемые ***таймерами синхронизации***.

|  |  |
| --- | --- |
| **Наименование**  **функции** | **Назначение** |
| **CancelWaitableTimer** | Отменить ожидающий таймер |
| **CreateWaitableTimer** | Создать ожидающий таймер |
| **OpenWaitableTimer** | Открыть существующий ожидающий таймер |
| **SetWaitableTimer** | Установить ожидающий таймер |
| **WaitForSingleObject** | Ждать сигнального состояния ожидающего таймера |

1. **Переодические и непереодические**

По способу перехода из несигнального состояния в сигнальное, ожидающие таймеры бывают ***периодические*** и ***непериодические***. Периодические таймеры работают по циклу: активное состояние – сигнальное состояние – активное состояние. Непериодические таймеры могут только один раз перейти из активного состояния в сигнальное.

1. **атомарные операции**

Иногда параллельным потокам необходимо выполнять некоторые несложные действия над общими переменными, исключая совместный доступ к этим переменным. Если в этом случаqе использовать критические секции или другие механизмы синхронизации, то может оказаться, что затраты на синхронизацию потоков значительно превысят затраты на выполнение самих операций. В таких случаях применяют блокирующие функции. Блокирующие функции выполняют несколько элементарных операций, которые объединяются в одну неделимую операцию, называемую ***атомарной операцией***.

1. **режим блокировки и без нее что значит?4 блокирующим функции**

Алгоритм работы функции accept, который рассматривался выше, был обусловлен ***режимом блокировки (blocking mode)***, установленным (по умолчанию) для сокета. Переключение сокета в ***режим без блокировки ( nonblocking mode)***, позволяет избежать приостановки программы. В режиме без блокировки выполнение accept, не приостанавливает выполнение потока, как это было прежде, а возвращает значение нового сокета, если обнаружен запрос на создание канала (функция connect, выполненная клиентом), или значение INVALID\_SOCKET, если запроса на создание канала нет в очереди запросов или возникла ошибка

|  |  |
| --- | --- |
| **Наименование**  **Функции** | **Назначение** |
| **InterlockedCompareExchange** | Сравнить и заменить значение |
| **InterlockedDecrement** | Уменьшить значение на единицу |
| **InterlockedExchange** | Заменить значение |
| **InterlockedExchangeAdd** | Изменить значение |
| **InterlockedIncrement** | Увеличить значение на единицу |

1. **какие приоритеты потоков и процессов есть?4 функции приоритетов и 4 функции потоков**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Приоритет**  **потока** | **Приоритет процесса** | | | | | |
| **реального**  **времени** | **высокий** | **выше**  **норм.** | **нормальный.** | **ниже**  **норм.** | **фоновый.** |
| **реального**  **времени** | 31 | 15 | 15 | 15 | 15 | 15 |
| **высший** | 26 | 15 | 12 | 10 | 8 | 6 |
| **выше норм.** | 25 | 14 | 11 | 9 | 7 | 5 |
| **нормальный** | 24 | 13 | 10 | 8 | 6 | 4 |
| **ниже норм.** | 23 | 12 | 9 | 7 | 5 | 3 |
| **низший** | 22 | 11 | 8 | 6 | 4 | 2 |
| **фоновый** | 16 | 1 | 1 | 1 | 1 | 1 |

|  |  |
| --- | --- |
| **GetPriorityClass** | Получить приоритет процесса |
| **GetThreadPriority** | Получить приоритет потока |
| **GetProcessPriorityBoost** | Определить состояние режима процесса |
| **GetThreadPriorityBoost** | Определить состояние режима потока |
| **SetPriorityClass** | Изменить приоритет процесса |
| **SetThreadPriority** | Изменить приоритет потока |
| **SetProcessPriorityBoost** | Установить или отменить динамический режим потоков процесса |
| **SetThreadPriorityBoost** | Установить или отменить динамический режим потока |

1. **задача оповещения и условных процедур что такое?**

механизм событий, позволяющий оповестить поток о некотором выполненном действии, произошедшем за пределами потока. Саму ***задачу оповещения*** часто называют ***задачей условной синхронизации***.

1. **что такое динамические библиотеки?**

Динамические библиотеки представляют собой программный модуль, который может быть загружен в виртуальную память процесса как статически, во время создания исполняемого модуля процесса, так и динамически во время исполнения процесса операционной системой

Динамические библиотеки позволяют программистам создавать функциональность, которую можно использовать в различных программах, без необходимости повторной компиляции и создания статической библиотеки для каждой программы. Кроме того, использование динамических библиотек позволяет экономить память, поскольку одна копия библиотеки может быть загружена в память и использована несколькими программами одновременно

1. **как загрузить/импортировать/отключить библиотеку**

|  |  |
| --- | --- |
| **FreeLibrary** | Отключить dll-библиотеку от процесса |
| **GetProcAddress** | Импортировать функцию |
| **LoadLibrary** | Загрузить dll-библиотеку |

1. **что такое events и 5 функций**

В операционных системах семейства Windows события описываются объектами ядра ***Events***.

|  |  |
| --- | --- |
| **Наименование**  **функции** | **Назначение** |
| **CreateEvent** | Создать событие |
| **OpenEvent** | Открыть событие |
| **PulseEvent** | Освободить ожидающие потоки |
| **ResetEvent** | Перевести событие в несигнальное состояние |
| **SetEvent** | Перевести событие в сигнальное состояние |

1. **как освободить ожидающие потоки?**

**PulseEvent**

1. **в рамках ос какие две группы можно создать**

Различают два типа событий: с ручным сбросом и с автоматическим сбросом. Различие между этими типами заключается в том, что событие с ручным сбросом можно перевести в несигнальное состояние только с помощью функции ResetEvent, а событие с автоматическим сбросом переходит в несигнальное состояние как с помощью функции ResetEvent, так и при помощи функции ожидания