Исходный код

1. Критическая секция

2. Асинхронные процедуры

3. Как работает акцепт в не блокирующем режиме

4. Механизм событий

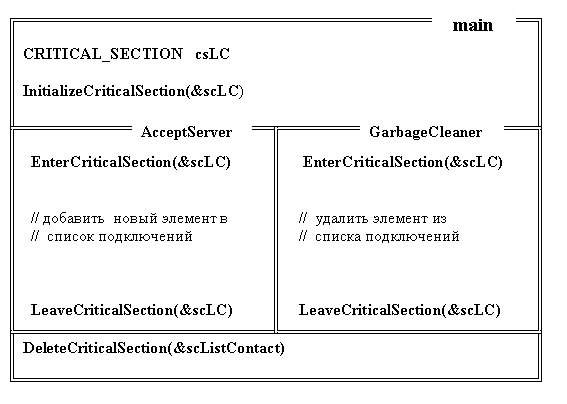
5. Таймер

6. Атомарные операции

7. Загрузка библиотеки

Синхронизация будет осуществляться между двумя потоками: AcceptServer и GarbageCleaner.

Оператор volatile указывает компилятору на необходимость размещения переменной cmd в памяти и не осуществлять относительно этого размещения никакой оптимизации. Дело в том, что область памяти отведенная переменной cmd, используется двумя параллельно работающими потоками AcceptServer и ConsolePipe. Поэтому оптимизация может привести к тому, что потоки будут использовать различные области памяти.



Критическая секция (critical section) — это небольшой участок кода, требующий монопольного доступа к каким-то общим данным. Она позволяет сделать так, чтобы единовременно только один поток получал доступ к определенному ресурсу.

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.

При подключении очередного клиента, функция AcceptServer запускает обслуживающий поток EchoServer. С этого момента AcceptServer теряет связь с потоком EchoServer и, в принципе, даже не знает о моменте окончания его работы. Будем предполагать, что в начале и при окончании работы потока EchoServer, поток AcceptServer должен выдавать на свою консоль сообщение о завершении работы обслуженного клиента. В этом случае поток EchoServer может поставить функцию (асинхронную процедуру) в специальную очередь к потоку AcceptServer. Момент выполнения асинхронной процедуры определяется внутри функции потока AcceptServer.

QueueUserAPC Поставить асинхронную процедуру в очередь

SleepEx Приостановит поток для выполнения асинхронных процедур

Изображение выглядит как текст, снимок экрана, Шрифт, число

Контент, сгенерированный ИИ, может содержать ошибки.

Одной из задач управления сервером является выявление слишком продолжительных подключений. Очевидным решением проблемы является введение ограничения на продолжительность соединения. Для реализации такого решения может быть использован механизм ожидающего таймера. Ожидающим таймером в Windows, называется объект синхронизации, который 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.

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. 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.

Изображение выглядит как текст, снимок экрана, число, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.

CreateWaitableTimer в диспатч и SetWaitableTimer

CancelWaitableTimer в диспатч и ехосервер ранд и тайм

WaitForSingleObject в конкарент и диспатч

Иногда параллельным потокам необходимо выполнять некоторые несложные действия над общими переменными, исключая совместный доступ к этим переменным. Блокирующие функции выполняют несколько элементарных операций, которые объединяются в одну неделимую операцию, называемую атомарной операцией.

InterlockedCompareExchange Сравнить и заменить значение

InterlockedDecrement Уменьшить значение на единицу

InterlockedExchange Заменить значение

InterlockedExchangeAdd Изменить значение

InterlockedIncrement Увеличить значение на единицу

Все перечисленные функции требуют, чтобы адреса переменных были выровнены на границу слова, т.е. были кратны четырем. Для такого выравнивания достаточно, чтобы переменная была объявлена в программе со спецификаторами типов long, unsigned long или LONG, ULONG, DWORD.

InterlockedDecrement в диспатч и garbage

InterlockedIncrement в акцепт и диспатч и garbage

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

int ioctlsocket(

SOCKET sock, // [in] дескриптор сокета

long scmd, // [in] команда, применяемая к сокету

u\_long\* pprm // [in,out] указатель на параметр команды

);

Примечание: параметр scmd может принимать следующие

// значения: FIONBIO, FIONREAD, SIOCATMARK; команда

// FIONBIO применяется для переключения режимов:

// blocking/nonblocking; для установки режима

// nonblocking значение параметра \*pprm должно быть

// отличным от нуля.

Функция CommandCycle предназначена для обработки команд управления сервером и подключения клиентов. Подключение клиентов осуществляется функцией AcceptCycle.

сначала программа клиента осуществляет процедуру подключения (для этого используется поток AcceptServer), потом поток DispatchServer принимает от клиента запрос (команду) на обслуживание и после этого уже запускается соответствующий поток ServiceServer, который исполняет команду и в случае необходимости обменивается данными с клиентом.

На поток DispatchServer возлагается прием первой команды от клиента после подключения, содержащей идентификатор, запрашиваемого сервиса. Соответствие идентификатора и адреса потоковой функции содержится в специальной таблице TableService, который формируется при инициализации сервера (например, на основе конфигурационного файла).

Поток DispatchServer получает информацию (сокет и его параметры) о новом подключении через список ListContact (элементы списка создаются и заполняются в потоке AcceptServer). Поиск в списке ListContact нового подключения DispatchServer осуществляет после того, как поток AcceptServer сигнализирует потоку DispatchServer (на рисунке сигнал обозначен штриховой линией). Сигнал о наличии нового подключения можно выполнить с помощью, уже рассмотренного выше, механизма асинхронных процедур или с помощью механизма событий.

В любом случае поток DispatchServer должен отследить интервал времени (с помощью ожидающего таймера) от момента подключения до получения первой команды, проанализировать команду на правильность, а также запустить поток соответствующий запрошенному сервису (и) или оправить диагностирующее сообщение.

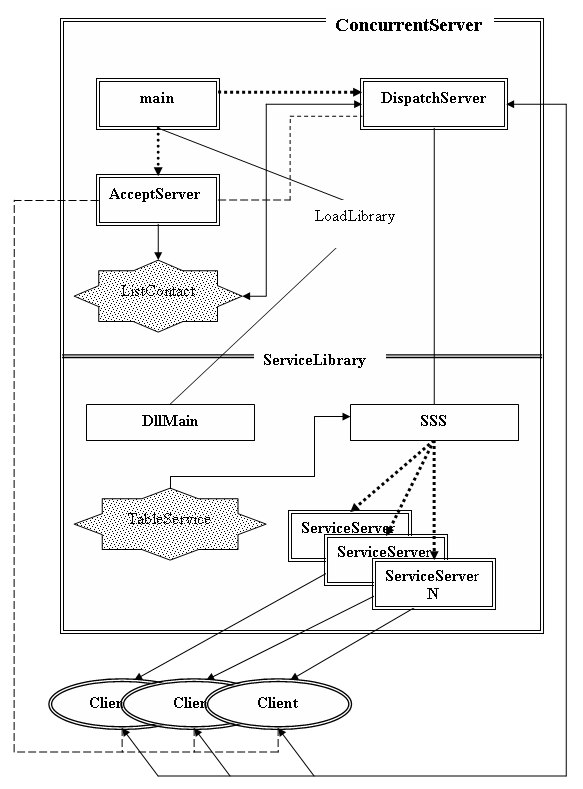
Выше уже упоминался механизм событий, позволяющий оповестить поток о некотором выполненном действии, произошедшем за пределами потока. Саму задачу оповещения часто называют задачей условной синхронизации.

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

CreateEvent в глобал

SetEvent в акцепт

При разработке параллельного сервера часто бывает полезным разделить процедуры управления сервером и обслуживания клиентов. Имеется в виду, что поддерживаемый сервером сервис может меняться при неизменной логике управления сервером. Одним из способов такого разделения является размещения функций обслуживающих потоков в динамически подключаемой библиотеке (dll-библиотеке).



В состав библиотеки входит стандартная функция DllMain, экспортируемая функция SSS для запуска потока по заданному клиентом коду команды, а также функции обслуживающих потоков.

FreeLibrary Отключить dll-библиотеку от процесса

GetProcAddress Импортировать функцию

LoadLibrary Загрузить dll-библиотеку

в конкарент

Модификатор extern “C” используется для указания компилятору на то, что эта функция имеет имя в стиле языка С. Квалификатор \_\_declspec(dllexport) применяется для обозначения экспортируемых dll-библиотекой функций.