Обзор методов повышения производительности программного обеспечения диспетчерскогоцентра

Д.НАкользин

Введение

В первую очередь необходимо объяснить, что подразумевается под диспетчерским центром. Общая структура диспетчерского центра, которая раскрывает внутренние детали взаимодействия и взаимосвязь с внешними объектами, показана на рисунке 1.

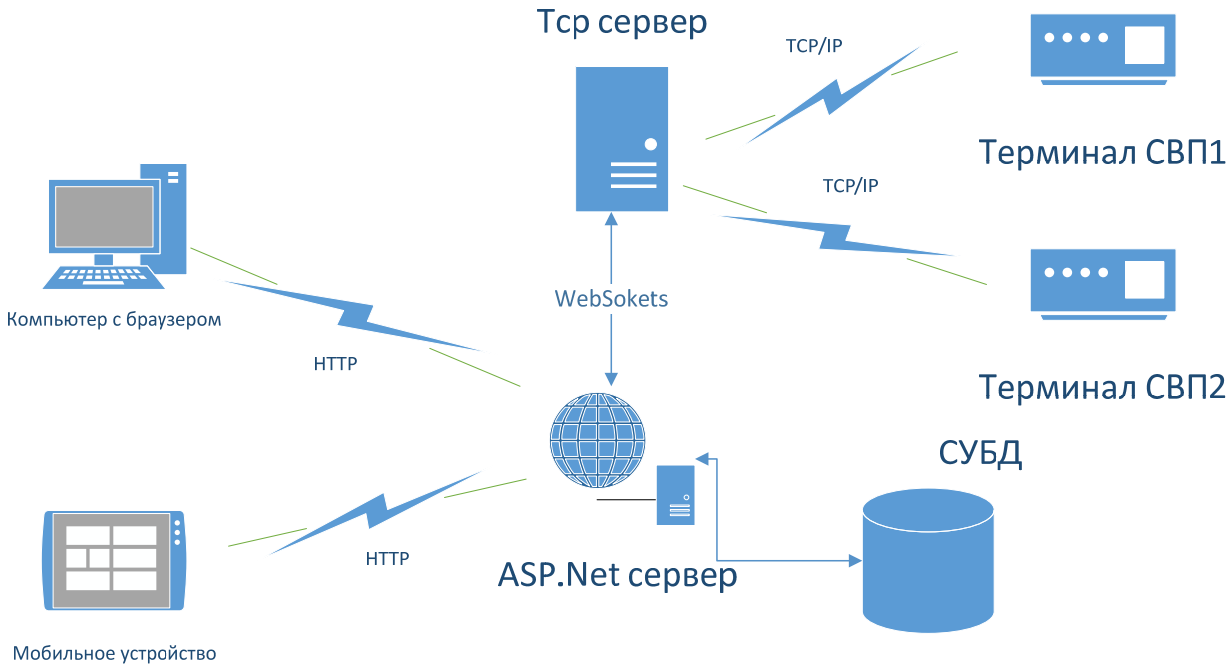

Рисунок 1- Общая структура серверного диспетчерского программного обеспечения

На рисунке видно, что TCP-сервер получает данные от терминалов СВП по протоколу TCP/IP, после этого отправляет их HTTP-серверу. HTTP-сервер осуществляет сохранение этих данных в СУБД [1].

Пользователи могут получить актуальную информацию об местоположении терминалов системы высокоточного позиционирования (СВП), выполнив запрос к HTTP-серверу, по протоколу HTTP. Терминал СВП представляет собой по сути GPS-трекер, показания которого корректируются при помощи дополнительных подсистем, таких как подсистема инерциальной навигации. При необходимости

пользователи могут получить историю сообщений от терминалов СВП, а так же список активных на данный момент терминалов СВП [2, 3].

Рассмотрим более подробно структуру серверного программного обеспечения на рисунке 2.

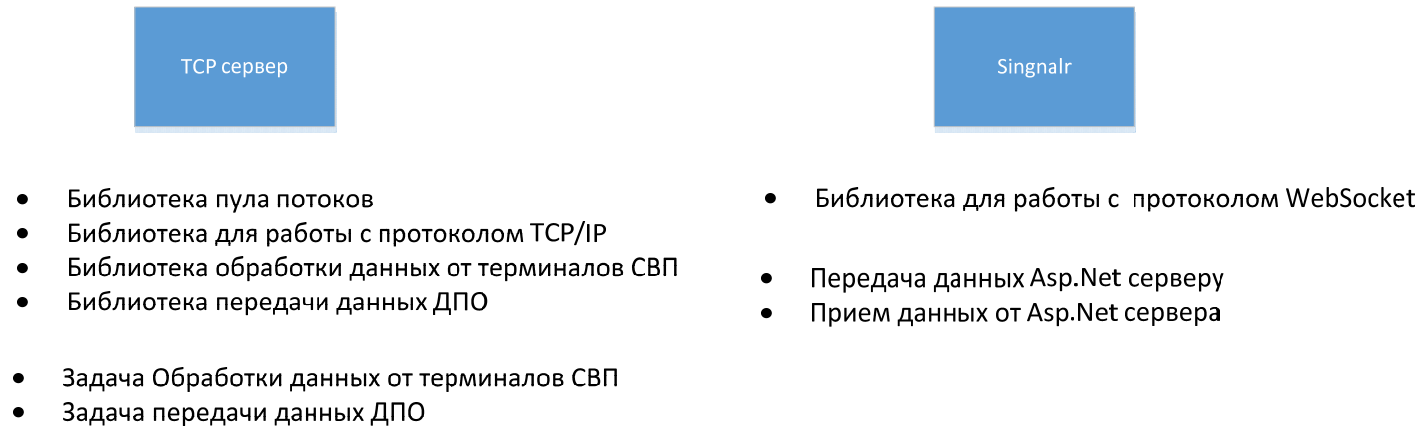

Рисунок 2 - Общая структура серверного диспетчерского программного обеспечения

Программное обеспечение сервера состоит из двух компонентов:

1) TCP-сервер;

2) библиотека Signalr.

В состав TCP-сервера входят следующие библиотеки:

- библиотека пула потоков;
- библиотека для работы с протоколом TCP/IP;
- библиотека обработки данных от терминалов СВП;
- библиотека передачи данных ДПО.

Библиотека пула потоков обеспечивает создание отдельного потока каждому терминалу СВП и выполнение уже созданных потоков, для обработки данных от терминалов СВП.

Библиотека для работы с протоколом TCP/IP обеспечивает создание TCP-канала, поддержания его, а также передачу данных с помощью этого канала.

Библиотека обработки данных от терминалов СВП предоставляет функции, для работы с терминалов СВП, которые будут исполняться в пуле потоков.

Библиотека передачи данных диспетчерского программного обеспечения (ДПО) осуществляет формирование и передачу данных посредством использования библиотеки Signalr.

В состав TCP-сервера входят следующие задачи:

- задача обработки данных от терминалов СВП;
- задача передачи данных ДПО.

Задача обработки данных обеспечивает прием, декодирование данных. Задача передачи данных осуществляет передачу полученных данных от терминала СВП с помощью библиотеки Signalr для дальнейшей обработки данных ДПО.

Методы повышения производительности

Мной были выделены три метода повышения производительности такого сервера:

1) повышение производительности с помощью использования более производительных комплектующих [4];
2) выбор более производительной платформы программирования, среды выполнения или операционной системы. Например, вместо операционной системы Windows использовать Unix систему или выбрать для сервера среду исполнения JRE (J) вместо .Net;
3) изменение архитектуры сервера в пределах одной среды выполнения.

Повышение производительности с помощью аппаратного изменения является. самым простым способом, но есть предел улучшения аппаратной части сервера. Необходимо отметить, что если диспетчерский центр используется на нескольких рабочих станциях, то необходимо провести улучшение каждой рабочей станции, в отличии от следующих двух методов, при использовании которых, необходимо лишь обновить программное обеспечение. В рамках данной статьи этот метод не рассматривается.

При изменении текущей платформы на более производительную стоить учесть, что придется заново разработать сервер на новой платформе, с учетом ее

особенностей. Этот метод должен выбираться в последнюю очередь, когда невозможны другие, либо они уже использованы. Стоить заметить, что сложность перехода на новую платформу зависит от текущего этапа разработки сервера, на более раннем этапе можно фактически безболезненно перейти на новую платформу.

Изменение архитектуры является первым методом, который должен рассматриваться. В настоящее время при построении сетевых серверов используют многопоточную архитектуру, в которой каждому клиенту выделяется один отдельный поток, но у этой архитектуры существует недостаток - часть процессорного времени расходуется на переключение между потоками и при увеличении клиентов эта часть увеличивается до критических значений [5]. Так же стоит учесть, что операции сервера с TCP-сокетом занимают определенное время и все это время поток не выполняет полезной работы [6]. Возможных выходом из этой проблемы является использование асинхронных операций ввода-вывода.

Асинхронные операции

Асинхронность на текущий момент является достаточно популярной темой. На волне этой популярности все большее распространение получает платформа Node.js. Создатели этой платформы преподносят асинхронность как главное преимущество над остальными. Стоит заметить, что все достаточно зрелые языки в том или ином виде поддерживают асинхронные операции. В новой версии языка C# разработчикам рекомендуют использовать асинхронный шаблон, основанный на задачах. Так же все функции на платформе .Net: чтения/записи данных на диск, операции работы с сокетами имеют асинхронную реализацию, которая использует порты завершения в операционной системе Windows [7, 8]. Пример выполнения асинхронной операции в операционной системе Windows изображен на рисунке 3.

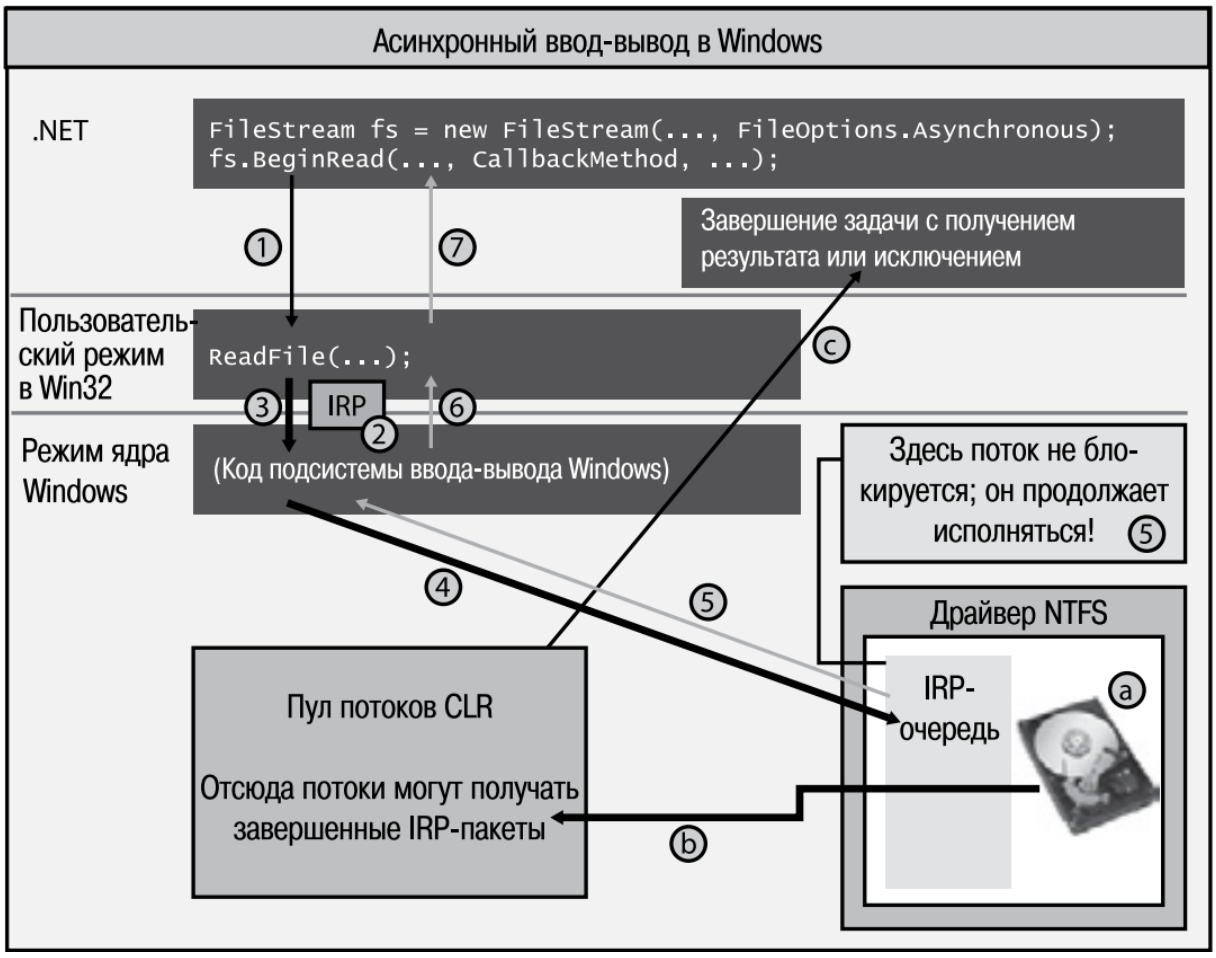

Рисунок 3 - Схема выполнения асинхронного запроса к базе данных.

Вызывая асинхронную операцию поток входит в пользовательский режим работы Windows. На каждую операцию ввода/вывода система Windows создает IRP-пакет, который включает себя параметры операции и идентификатор вызывающего потока. После этого IRP-пакет ставиться в очередь к устройству. После выполнения устройством задачи в IRP-пакет записывается результат операции и передается в пул потоков CLR. Планировщик передает IRP-пакет потоку согласно дескриптору процесса [7, 8].

Асинхронные операции являются ключом к созданию высокопроизводительных масштабируемых приложений, выполняющих множество операций при помощи небольшого количества потоков, используя модель проактора. Для всех операций вызывается функция и передается callback, который автоматически выполнится по окончании операции. Т.е. поток продолжит работу, как только операция завершится. Это отличается от модели реактора, когда мы должны сами вызывать нужные обработчики, наблюдая за

состоянием операций. Наибольшую пользу асинхронные операции приносят при использовании их для долговременных операций ввода-вывода, в течение которых рабочий поток ожидает их завершения. Это позволяет экономить ресурсы сервера.

Для максимального повышения производительности асинхронные операции используются с паттерном параллельного программирования «Пул потоков» [9].

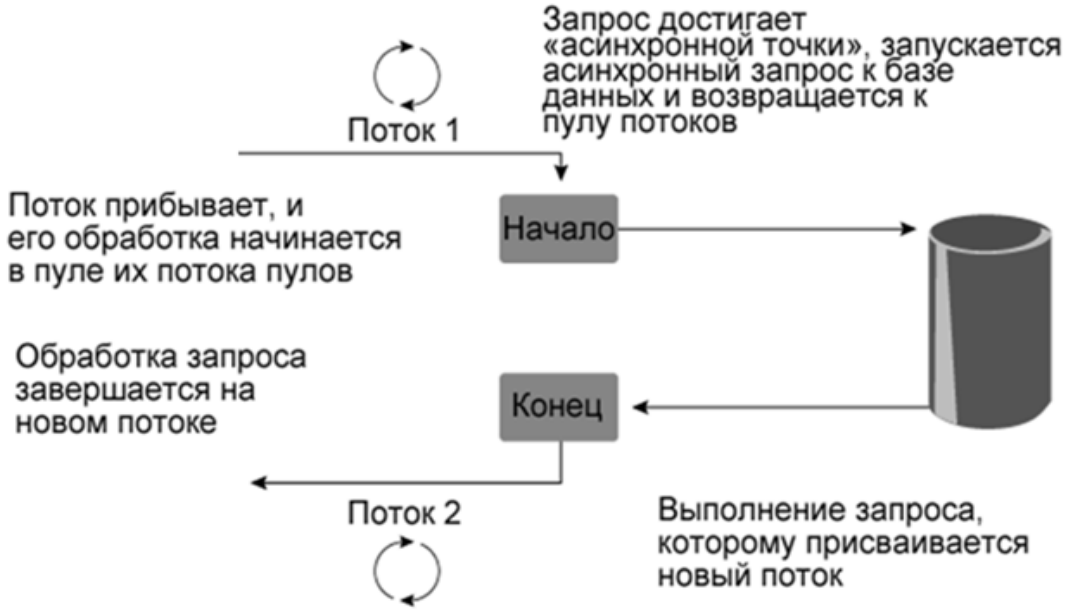

Рисунок 4 - Схема выполнения асинхронного запроса к базе данных.

После начала асинхронной операции рабочий поток не блокируется, а возвращается в пул, получая возможность заняться обработкой других клиентских запросов. Таким образом, получается, что для обработки всех входящих запросов достаточно всего одного потока. Полученный от базы данных ответ также окажется в очереди пула потоков, то есть наш поток сможет тут же его обработать и отправить данные клиенту. Таким образом, единственный поток обрабатывает не только клиентские запросы, но и все ответы базы данных. На рисунке 4 схематически изображено выполнение такого запроса. В итоге сервер практически не потребляет системных ресурсов, но работает с максимально возможной скоростью, так как переключения контекста не происходит! Если элементы появляются в пуле быстрее, чем поток может их обработать, пул может создать дополнительные потоки. Пул быстро создаст по одному потоку на

каждый процессор. Соответственно, на машине с четырьмя процессорами четыре клиентских запроса к базе данных и ответа базы данных будут обрабатываться в четырех потоках без какого-либо переключения контекста [8].

Главный недостаток асинхронного программирования в большей сложности, как написания исходного кода, так и понимания, чем в данный момент может заниматься приложение, вследствие этого обработка ошибочных ситуаций еще сложнее, чем при мультипоточном сервере. При небольшом усложнении схемы обработки запросов возрастает чуть ли не экспоненциально. Обработка ошибок теперь существенно отличается. В случае синхронного подхода у нас 2 варианта: возврат кода ошибки либо генерация исключения (в данный момент является рекомендуемым при разработке). В случае асинхронного вызова способ существует ровно один: передача ошибки через обработчик. Т.е. даже не через результат, а как входной параметр обработчика.

Одним из способов решения проблемы сложности разработки асинхронного сервера и обработки ошибок является использование сопрограмм.

Сопрограммы

Сопрограммы являются более обобщёнными, чем подпрограммы. Отличительной особенностью сопрограмм является наличие нескольких точек входа, как изображено на рисунке 5. Подпрограмма имеет всегда одну входную точку, сопрограмма имеет стартовую точку входа и размещённые внутри последовательность возвратов и следующих за ними точек входа. Подпрограмма может возвращаться только однажды, сопрограмма может возвращать управление несколько раз. После повторного вызова подпрограммы, выполнение начнется со стартовой точки входа, а в сопрограмме с места в котором было передано управление.

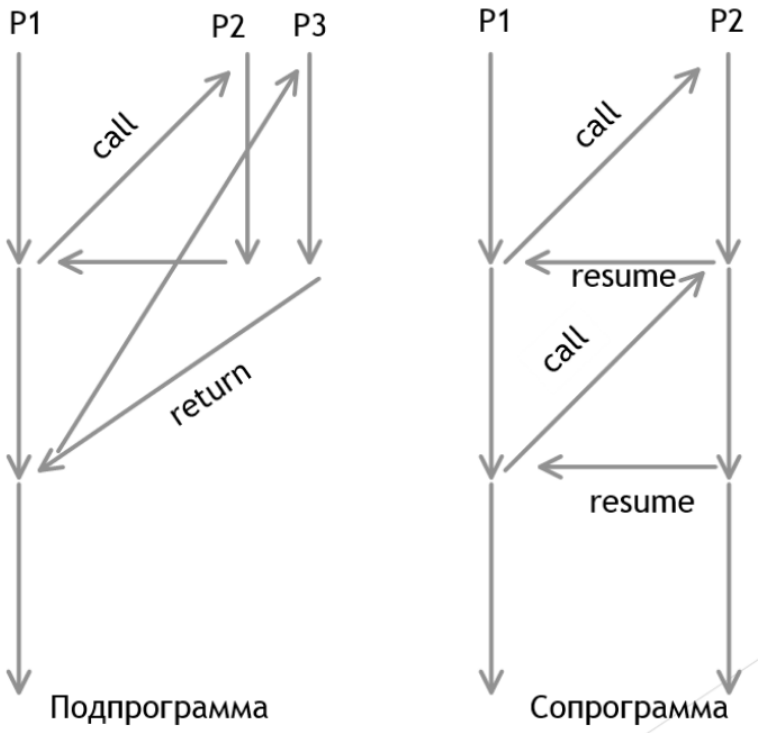

Рисунок 5 - Отличительные особенности сопрограммы от подпрограммы.

Благодаря сопрограммам код выглядит так, как будто выполняется в одном потоке выполнения. Вследствие этого повышается легкость написания программ, легкость поддержки и отлова ошибок. Стоит заметить, что на создание и на управление сопрограммам тратится процессорное время поэтому производительность на 10%-15% меньше, чем у чисто асинхронного подхода [9].

Существуют две различных реализации сопрограмм:

1) StackLess реализация, которая, по сути, представляет собой конечный автомат, хранящий в себе нужные локальные переменные и точки входа. Конструкция Async/await языка C# 5-ой версии реализована именно так [10].
2) StackFull реализация работает через сохранение/восстановление всех регистров процессора и стека соответственно — по сути, как у настоящих потоков, только это всё без обращения к ОС, так что практически мгновенно. Эта реализация нашла применение в библиотеке Boost.Coroutine, Boost.Context языка C++ .

StackFull является более производительной реализацией сопрограмм, но так же и самой сложной. Данная реализация существует лишь в библиотеках

Boost.Coroutine, Boost.Context языка C++. Остальные языки программирования используют StackLess реализацию.

Эффективное использование асинхронных операций

Асинхронные операции являются самым эффективным методом повышения производительности диспетчерского центра, который большую часть времени работает с операциями ввода-вывода. Этот метод опробован мной на проекте диспетчерского центра системы высокоточного позиционирования объектов. Количество клиентов, запросы которых обрабатывает сервер, возросло на 40%. Это очень существенный прирост производительности при минимальных трудозатратах. В это внес свой вклад правильный выбор языка программирования. В данном случае это C#, который в своей пятой версии получил поддержку асинхронных операций с помощью сопрограмм.

Результаты исследований, изложенные в данной статье, получены при финансовой поддержке Минобрнауки РФ в рамках реализации проекта «Создание высокотехнологичного производства для изготовления комплексных реконфигурируемых систем высокоточного позиционирования объектов на основе спутниковых систем навигации, локальных сетей лазерных и СВЧ маяков и МЭМС технологии » по постановлению правительства №218 от 09.04.2010 г. Исследования проводились в ФГАОУ ВПО ЮФУ.

Литература:

1. Вишневский В. М. Теоретические основы проектирования компьютерных сетей. [Текст] / В. М.Вишневский – М.: Техносфера, 2003. – 512 с.: ил.
2. Овчинников В.Н. Организация передачи информации в автоматизированных системах управления [Текст] / В.Н.Овчинников – М.: Энергия, 1974. - 128 с.: ил.
3. Мишин А. И., Леус В. А. Асинхронно-локальные вычислительные системы и среды. [Текст] / А.И. Мишин, В.А. Леус. - Новосибирск: ИМ, 1991. - 178 с. : ил.

4. Рудь Д.Е. Технологии топологической оптимизации трафика информационных потоков в телекоммуникационных сетях [Электронный ресурс] / Д.Е. Рудь // «Инженерный вестник Дона», 2010, №2. – Режим доступа: http://www.ivdon.ru/magazine/archive/n2y2010/193 (доступ свободный) – Загл. с экрана. – Яз. рус.
5. Пономарева Е.И. Совершенствование процесса обработки данных при помощи облачных вычислений [Электронный ресурс] / Е.И. Пономарева // «Инженерный вестник Дона», 2012, 81№1. – Режим доступа: http://www.ivdon.ru/magazine/archive/n1y2012/628 (доступ свободный) – Загл. с экрана. – Яз. рус.
6. Маккафри Д. Асинхронные TCP-сокеты как альтернатива WCF [Электронный ресурс] / WCF. MSDNMagazine. 2014, № 3. - Режим доступа: http://msdn.microsoft.com/ru-ru/magazine/dn605876.aspx (доступ свободный) – Загл. с экрана. – Яз. рус.
7. Рихтер Д. CLR via C#. Программирование на платформе Microsoft.NET Framework 4.5 на языке C# [Текст] / Рихтер Д. – Питер– 2012. – 896 с.
8. Petzold C. Asynchronous Processing in Windows 8. // New York, 2011 - Url: http://www.charlespetzold.com/blog/2011/11/Asynchronous-Processing-in-Windows-8.html.
9. Демченко Г. Асинхронность: назад в будущее [Электронный ресурс] / Режим доступа: http://habrahabr.ru/post/201826 (доступ свободный) – Загл. с экрана. – Яз. рус.
10. Shankar A. Implementing Coroutines for .NET by wrapping the Unmanaged Fiber API. // MSDN Magazine, 2005, № 11 - Url: http://msdn.microsoft.com/ru-ru/magazine/cc164086%28en-us%29.