ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ   
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

**КУРСОВОЙ ПРОЕКТ**

по дисциплине “Операционные системы”

на тему

**Проектирование и разработка гибридного сетевого приложения**

|  |  |
| --- | --- |
| Выполнил студент | Дьяченко Даниил Вадимович |
|  | Ф.И.О. |

|  |  |
| --- | --- |
| Группы | ИВ-621 |
|  |  |

|  |  |  |
| --- | --- | --- |
| Работу принял |  | преп. Берлизов Д.М. |
|  | подпись |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Защищена |  | Оценка |  |
|  |  |  |  |

Новосибирск – 2018

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc532755663)

[ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ 4](#_Toc532755664)

[РЕАЛИЗАЦИЯ ПРОГРАММНОГО ПРОЕКТА 8](#_Toc532755665)

[ЗАКЛЮЧЕНИЕ 15](#_Toc532755666)

[СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ 16](#_Toc532755667)

ВВЕДЕНИЕ

В ходе выполнения расчётно-графического задания необходимо спроектировать и разработать гибридное (параллельное и распределённое) программное обеспечение, реализующее сетевую многопользовательскую игру. Разрабатываемое программное обеспечение должно иметь возможность реализовать игровую ситуацию как на одной ЭВМ, так и с использованием распределённой среды. При реализации игры в распределённом режиме управлением игрой должен заниматься выделенный сервер (отдельный процесс).

В результате выполнения предложенного задания была спроектирована и реализована многопользовательское гибридное приложение, реализующее сетевую игру на базе самостоятельно разработанного алгоритма. Описание реализованного программного пакета, содержащееся в данной пояснительной записке, состоит из теоретических сведений об использованных технологиях и изученных в ходе проектирования стандартов, а также из детального разъяснения основных свойств и особенностей реализованного приложения: серверного и клиентского модулей, а также процесса взаимодействия между ними.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

В данной реализации был использован язык программирования C# и .NET Framework 4.7.2.

Рассмотрим базовые понятия и определения, составляющие основу реализованного сетевого приложения. Взаимодействие клиентской и серверной частей игры происходит на базе стека протоколов TCP/IPv4. Протокол IPv4 использует адреса размером в 4 байта (32 бита), пакет IP состоит из четырнадцати полей (рис. 1).

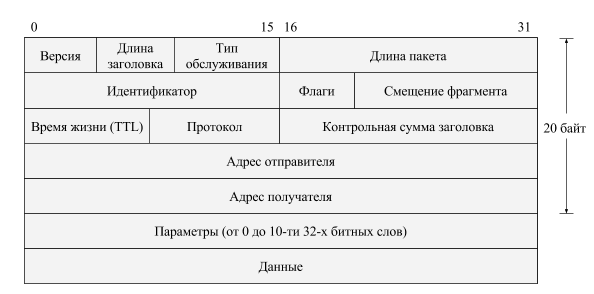
**

Рисунок 1 — Иллюстрация структуры пакета IPv4

**Сокеты**

Для обмена информацией между пользователями игры и серверным приложением используются сокеты. Сокетом (от англ. socket — «разьём») называется программный интерфейс, позволяющий реализовать взаимодействие между удалёнными компьютерами, или же между процессами в рамках одного компьютера (рис. 2). В UNIX-подобных системах сокет рассматривается как файловый дескриптор.

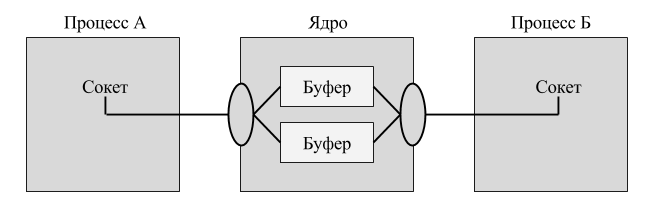
****

Рисунок 2 — Взаимодействие двух процессов посредством сокетов

В языке C# есть готовый класс Socket, который реализует интерфейс сокетов Berkeley. В описании функций присутствует класс EndPoint – определяет сетевой адрес, то есть ip и port.

|  |  |
| --- | --- |
| **Функция** | **Описание** |
| Socket(AddressFamily, SocketType, ProtocolType) | Инициализирует новый экземпляр класса Socket, используя заданные семейство адресов, тип сокета и протокол. |
| Accept() | Создает новый объект Socket для заново созданного подключения. |
| Bind(EndPoint) | Связывает объект Socket с локальной конечной точкой. |
| Close() | Закрывает подключение Socket и освобождает все связанные ресурсы. |
| Connect(EndPoint) | Создает подключение к удаленному узлу. |
| Listen(Int32) | Устанавливает объект Socket в состояние прослушивания. |
| Poll(Int32, SelectMode) | Определяет состояние объекта Socket. |
| Receive(Byte[]) | Возвращает данные из связанного объекта Socket в приемный буфер. |
| Send(Byte[]) | Передает данные в подключенный объект Socket. |

Таблица 1 – Основные функции для взаимодействия с сокетом

**Потоки**

Многопоточность является естественным продолжением многозадачности, точно также как виртуальные машины, позволяющие запускать несколько ОС на одном компьютере, представляют собой логическое развитие концепции разделения ресурсов. В рамках неформального, но простого, определения, поток – это выполнение последовательности машинных инструкций. В многопоточном приложении одновременно работает несколько потоков. Для реализации свойства многопоточности в разработанном приложении использовалось два вида потоков: потоки из стандарта POSIX, а также класс Thread, предоставляющий возможность создания потоков при написании кода на языке программирования C#.

POSIX Threads (Pthreads) — стандарт POSIX реализации потоков (нитей) выполнения. Стандарт POSIX.1c, Threads extensions (IEEE Std 1003.1c-1995) определяет API для управления потоками, их синхронизации и планирования. Реализации данного API существуют для большого числа UNIX-подобных ОС (GNU/Linux, Solaris, FreeBSD, OpenBSD, NetBSD, OS X), а также для Microsoft Windows и других ОС.

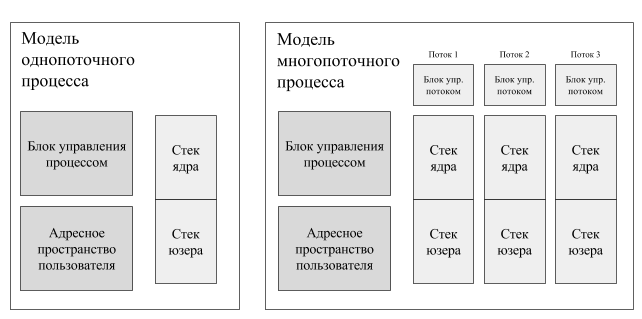
****

Рисунок 3 - Модели однопоточного и многопоточного приложений

Используемые классы в проекте:

* Thread - создание, контролирование, задание приоритета и возвращение статуса потока.
* Mutex – синхронизация потоков.

Ниже приведены таблицы, в которых указаны используемые методы, доступных классов.

|  |  |
| --- | --- |
| **Функция** | **Описание** |
| Thread(ThreadStart) | Инициализирует новый экземпляр класса Thread. |
| Join() | Блокирует вызывающий поток до завершения потока, представленного экземпляром, продолжая отправлять стандартные сообщения COM и SendMessage. |
| Start() | Вынуждает операционную систему изменить состояние текущего экземпляра на Running. |

Таблица 2 – используемые методы класса Thread.

|  |  |
| --- | --- |
| **Функция** | **Описание** |
| WaitOne() | Блокирует текущий поток до получения сигнала объектом WaitHandle. |
| ReleaseMutex() | Освобождает объект Mutex один раз |

Таблица 3 – используемые методы класса Mutex.

**Взаимодействие с терминалом**

Терминалы – устройства ввода и вывода информации. Часто для взаимодействия с ЭВМ, в составе которых не предусмотрены собственные средства для взаимодействия с оператором, используются специальные устройства, называемые терминалами. Терминалы различаются возможностями устройств, входящих в их состав (т.е. сколько клавиш на клавиатуре, может ли монитор выводить графическую информацию или только текст, используется ли цвет для вывода информации на монитор и т.п.).

Для изменения параметров терминала, а также для ввода/вывода был использован класс Console, предоставляющий стандартыне потоки для консольных приложений: входной, выходной, поток сообщений об ошибках. Для определения нажатой клавиши использовался класс ConsoleKey, представляющий перечисление (enum) стандартных клавиш консоли.

|  |  |
| --- | --- |
| **Функция** | **Описание** |
| ReadKey(Boolean) | Получает следующий нажатый пользователем символ или функциональную клавишу. Нажатая клавиша может быть отображена в окне консоли. |
| SetBufferSize(Int32, Int32) | Устанавливает заданные значения высоты и ширины буферной области экрана. |
| SetCursorPosition(Int32, Int32) | Устанавливает положение курсора. |
| SetWindowSize(Int32, Int32) | Устанавливает заданные значения высоты и ширины окна консоли. |
| Write(String) | Записывает заданное строковое значение в стандартный выходной поток. |
| BackgroundColor | Возвращает или задает цвет фона консоли. |
| ForegroundColor | Возвращает или задает цвет фона консоли. |

РЕАЛИЗАЦИЯ ПРОГРАММНОГО ПРОЕКТА

Кодовое название приложения «Ping Pong with Chatting».

В данном разделе описывается процесс проектирования и разработки приложения.

В текущей версии реализованы следующие функции:

* Мульти сессия – клиенты после подключения к серверу могут либо создать сессию игры, либо зайти в существующую.
* Чат – игроки могут общаться текстовыми сообщения в рамках сессии.
* Ping Pong – классическая игра с двумя ракетками и шариком.

В исходном коде зашиты параметры консоли, ip и порт сервера игры и чата.

Запуск клиентского приложения встречает пользователя вводом никнейма.

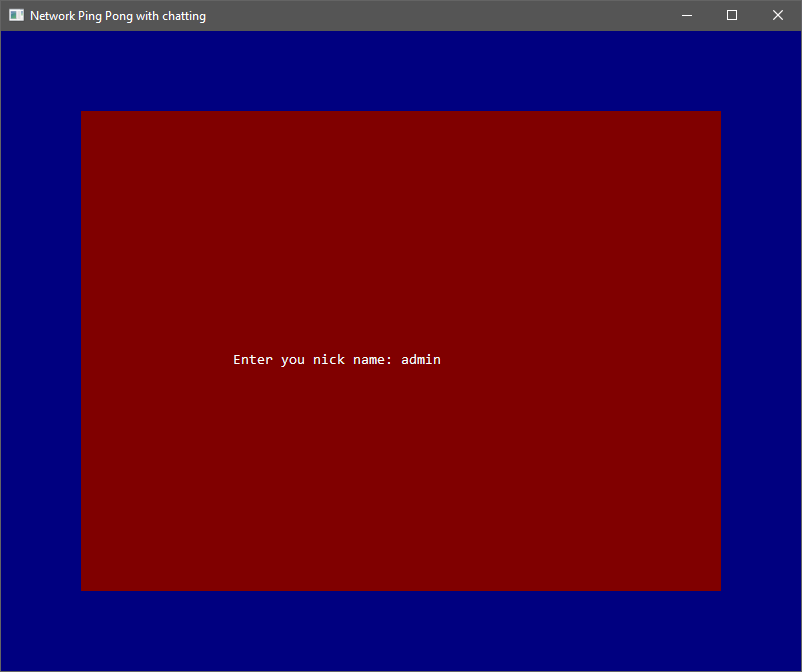


Рисунок 4 – Интерфейс приветствие игрока и ввод никнейма

В случае, если подключившийся клиент оказывается первым, сервер уведомляет соответствующим сообщением и создает сессию.

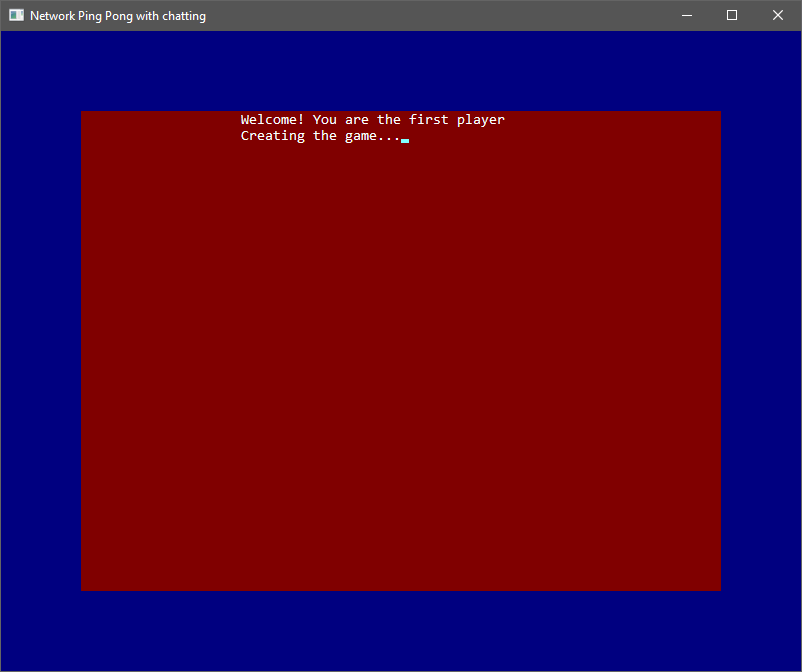


Рисунок 5 – Интерфейс ожидания подключения второго игрока к сессии

В обратном случае, клиенту отправляется список существующих сессий. Данный список содержит следующие поля:

* номер сессии
* никнейм игрока, создавшего сессию
* никнейм игрока, подключившегося к сессии
* GID (game id) – уникальный номер сессии
* статус сессии (либо Free, либо Busy)

В Главном меню поддерживаются следующие команды:

* -cr – создание сессии
* -r – обновление списка
* <номер сессии> - зарос на подключение к сессии

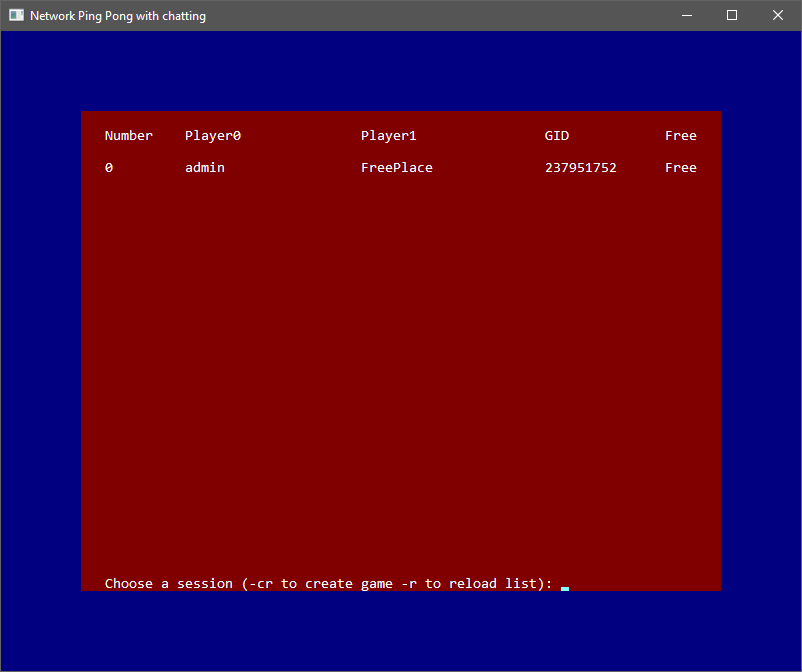


Рисунок 6 - Главное меню (выбор и создание сессии).

После ввода номера сессии серверу отправляется запрос на подключение, если сессия свободна, то обоим игрокам отправляется положительный ответ и вызывается метод отрисовки игрового поля (RenderGame) и начала игры (Start). Затем начинается игровой процесс.

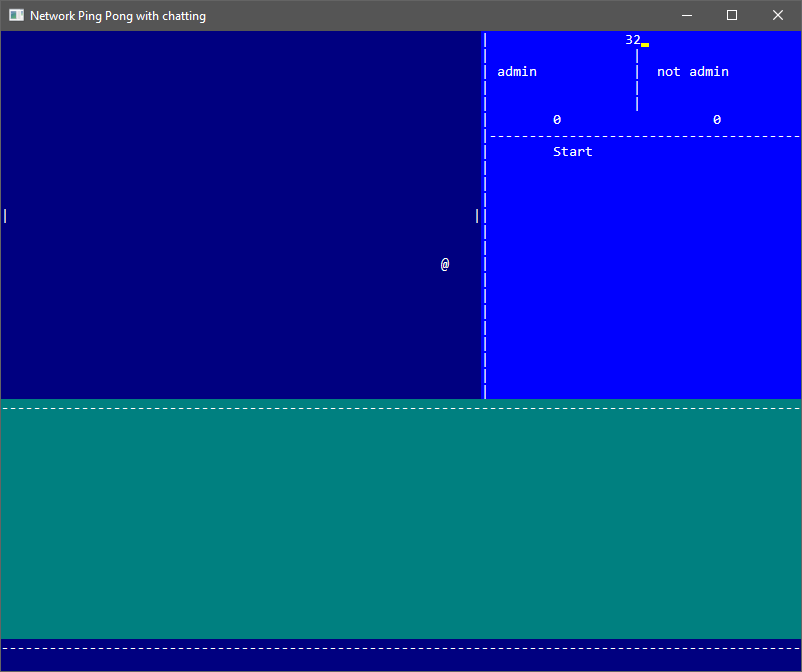


Рисунок 1 - Интерфейс игровой зоны. Событие – процесс игры.

В левой верхней части располагается игровой поле, справа блок статистики (время, никнеймы игроков, количество очков и action события), нижняя половина игровой зоны отдана под чат (самая нижняя строчка выделена для ввода, а остальная часть под вывод).

Управление происходит по нажатию кнопок «стрелочка вверх» и «стрелочка вниз» для движения ракеткой вверх и вниз, соответственно.

Ввод в чат происходит в любой момент игры. Любая нажатая клавиша с буквой или пробелом отобразится соответствующим символом в самой нижней строке. «Backspace» работает по-умолчанию, то есть стирает один символ. По нажатию «Enter» накопившееся сообщение отправляется на сервер и далее отсылается второму игроку.

Игра ведется до 5 очков.

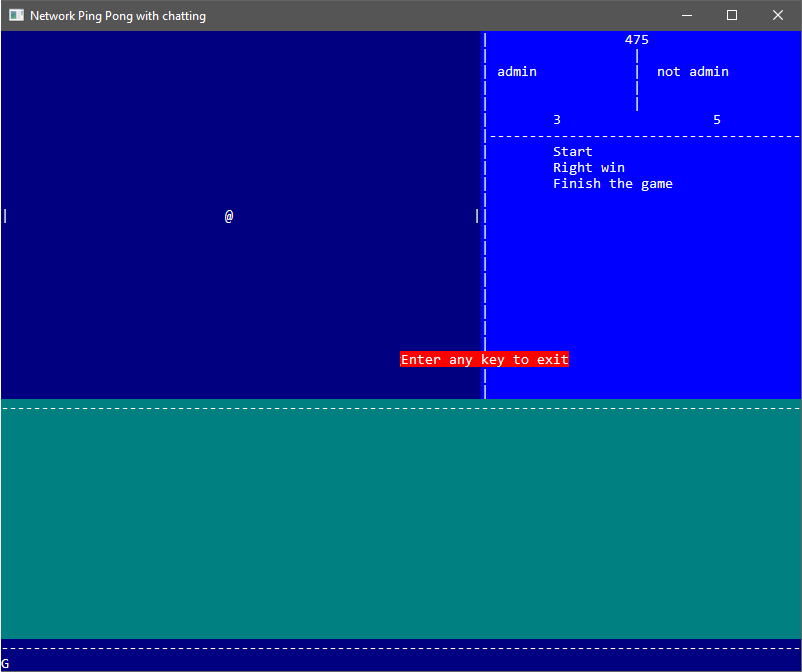


Рисунок 2 - Интерфейс игровой зоны. Событие - завершение игры.

По завершении игры выводится информация в область статистике о том, кто победил и о том, что игра завершена, а также подсказка для закрытия приложения.

Параллелизм клиентской части в одновременной работе нескольких потоков для обеспечения правильной логики и комфортной игры. В методе «Start» клиент запускает несколько потоков, выполняющие следующие функции:

* TimerHandler – таймер, по истечении которого клиент понимает, что пакеты от сервера больше не приходят.
* StartSend – метод упаковки и отправки пакета сообщения на сервер с заданной задержкой, которая определяет скорость игры. В сообщение содержится положение ракетки по оси Y.
* StartReceive – метод, принимающий и распаковывающий пакет сообщения от сервера, а также ререндер игровой зоны по событию. Форматы принимаемого сообщения:
  + ACTION;YY;YY;XX,YY;P,P – первое слово определяет поведение клиента, второе положение левой ракетки, третье положение правой ракетки, четвертое положение шарика по X и Y, пятое очки левого и правого игрока
  + MOTION;LOSE(WIN,END) – первое слово определяет поведение клиента: засчитать очко поражения или победы или завершить игру.
* ChatHandler – метод, отвечающий за прием сообщений чата и его вывод.
* PlayHandler – главный метод, в котором происходит обработка нажатия клавиш, а также рендер чата по событию.

Следующая часть документации посвящена реализации серверной части приложения. Реализованная серверная часть игры также является многопоточной и поддерживает проведение множества игр одновременно, так же, как и получение и обработку информации от множества пользователей в рамках одного сеанса.

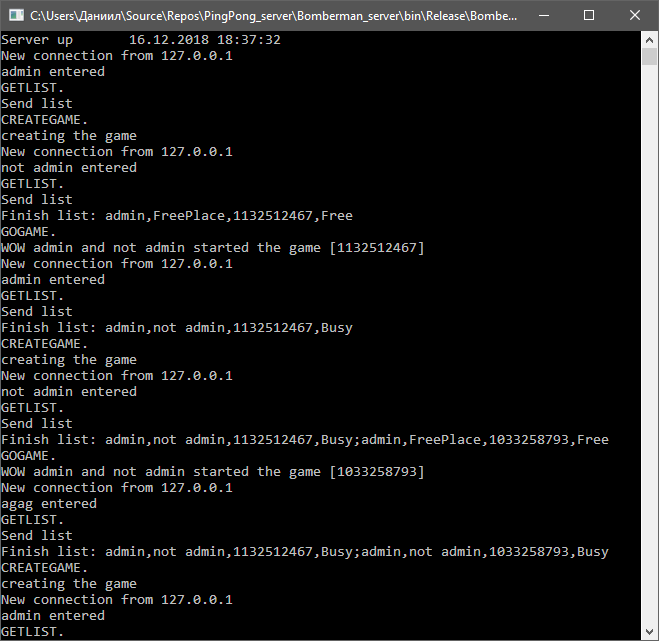


Рисунок 3 - Пример вывода логов сервера

Главный поток сервера ожидает подключения клиента, с помощью метода Accept. Подсоединившись, сервер создает поток и отдает ему функцию для обработки запросов клиента. Таким образом, получается ситуация при которой создается два потока для обработки запросов от двух разных клиентов, но так как для их взаимодействия в процессе игры требуется лишь один, то поток, отвечающий за подключившегося к сессии игрока закрывается.

Сервер хранит список сессий. Сессия представляет собой класс, инкапсулирующий следующие поля:

* Player Left;
* Player Right;
* int GID;
* SessionStatus status;

Игрок также представляет собой класс, инкапсулирующий следующие поля:

* string nickName;
* Socket socket;
* Socket chatSocket;
* PlayerStatus status;

SessionStatus (статус сессии) является перечислением (enum), содержащим поля Free (свободно) и Busy (занято).

ЗАКЛЮЧЕНИЕ

В результате выполнения курсовой работы было спроектировано и разработано гибридное сетевое приложение, реализующее многопользовательскую игру на основе самостоятельно разработанного алгоритма.

Приложение состоит из клиентской и серверной части. В процессе проектирования были изучены различные технологии и стандарты, приобретены навыки проектирования программного обеспечения в парадигме многопоточности и имеющего возможность выполнения в рамках распределённых вычислительных систем и коммуникационных сетей.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Мамойленко С.Н., Молдованова О.В. ЭВМ и периферийные устройства: Учебное пособие. – Новосибирск: СибГУТИ, 2012. – 106 с.
2. D. P. Bovet, M. Cesati, Understanding the Linux Kernel, 3rd Edition, O'Reilly, 2005