Лабораторная работа № 1

Организация распределённых вычислений с использованием сокет TCP/IP средствами WinAPI.

1. Теоретические сведения

Общие сведения о сокетах

Сокетом (от англ. socket - гнездо, розетка) называется специальный объект, создаваемый для отправки и получения данных через сеть. Отметим, что под термином "объект" в данном случае подразумевается не объект в терминах объектно-ориентированного программирования, а некоторая сущность, внутренняя структура которой скрыта от нас, поэтому с этой сущностью мы можем оперировать только как с единым и неделимым (атомарным) объектом. Этот объект создаётся внутри библиотеки сокетов, а программист, использующий эту библиотеку, получает уникальный номер (дескриптор) этого сокета. Конкретное значение этого дескриптора не несёт для программиста никакой полезной информации и может быть использовано только для того, чтобы при вызове функции из библиотеки сокетов указать, с каким сокетом требуется выполнить операцию.

Чтобы две программы могли общаться друг с другом через сеть, каждая из них должна создать сокет. Каждый сокет обладает двумя основными характеристиками: протоколом и адресом, к которым он привязан. Протокол задаётся при создании сокета и не может быть изменён впоследствии. Адрес сокета задаётся позже, но обязательно до того, как через сокет пойдут данные. В некоторых случаях привязка сокета к адресу может быть неявной.

Формат адреса сокета определяется конкретным протоколом. В частности, для протоколов TCP и UDP адрес состоит из IP-адреса сетевого интерфейса и номера порта.

Каждый сокет имеет два буфера: для входящих и для исходящих данных. При отправке данных они сначала кладутся в буфер исходящих, и лишь затем отправляются в фоновом режиме. Программа в это время продолжает свою работу. При получении данных сокет кладёт их в буфер для входящих, откуда они затем могут извлекаться программой.

Классификация сокетов

Для программирования сокетов воспользуемся библиотекой Winsock. Библиотека Winsock поддерживает два вида сокетов – **синхронные** (блокируемые) и **асинхронные** (неблокируемые).

Синхронные сокеты задерживают управление на время выполнения операции, а асинхронные возвращают его немедленно, продолжая выполнение в фоновом режиме, и, закончив работу, уведомляют об этом вызывающий код.

Независимо от вида, сокеты делятся на два типа – **потоковые** (опираются на протокол TCP) и **дейтаграммные** (опираются на протокол UDP).

Потоковые сокеты работают с установкой соединения, обеспечивая надежную идентификацию обоих сторон и гарантируют целостность и успешность доставки данных.

Дейтаграмные сокеты работают без установки соединения и не обеспечивают ни идентификации отправителя, ни контроля успешности доставки данных, зато они заметно быстрее потоковых.

Создание и работа с сокетами в WinAPI

Функции, позволяющие осуществлять те или иные операции с сокетами экспортируются системной библиотекой wsock32.dll, а также библиотекой ws2_32.dll.

Для их использования в Delphi в раздел uses нужно добавить стандартный модуль WinSock.

Для работы с библиотекой Winsock 2.x с помощью Microsoft Visual C++ в исходный тест программы необходимо включить директиву #include , а в командной строке линкера указать "ws2_32.lib". В Microsoft Visual Studio для этого достаточно нажать , перейти к закладке "Link" и к списку библиотек, перечисленных в строке "Object/Library modules", добавить "ws2_32.lib", отделив ее от остальных символом пробела.

Перед началом использования функций библиотеки Winsock ее необходимо подготовить к работе вызовом функции:

int WSAStartup (WORD wVersionRequested, LPWSADATA lpWSAData)

Передаем в старшем байта слова wVersionRequested номер требуемой версии, а в младшем – номер подверсии. Аргумент lpWSAData должен указывать на структуру WSADATA, в которую при успешной инициализации будет занесена информация о производителе библиотеки. Никакого особенного интереса она не представляет и прикладное приложение может ее игнорировать. Если инициализация проваливается, функция возвращает ненулевое значение. Создание объекта "сокет". Это осуществляется функцией:

SOCKET socket (int af, int type, int protocol)

Первый аргумент указывает на семейство используемых протоколов. Следующий аргумент задает тип создаваемого сокета – потоковый (SOCK_STREAM) или дейтаграммный (SOCK_DGRAM). Последний аргумент уточняет какой транспортный протокол следует использовать. Нулевое значение соответствует выбору по умолчанию: TCP – для потоковых сокетов и UDP для дейтаграммных. Если функция завершилась успешно она возвращает дескриптор сокета, в противном случае INVALID_SOCKET.

Для установки соединения с удаленным узлом потоковый сокет должен вызвать функцию:

int connect (SOCKET s, const struct sockaddr FAR* name, int namelen)

Первый аргумент – дескриптор сокета, возращенный функцией socket; второй – указатель на структуру "sockaddr", содержащую в себе адрес и порт удаленного узла с которым устанавливается соединение. Последний аргумент сообщает функции размер структуры sockaddr. После вызова connect система предпринимает попытку установить соединение с указанным узлом. Если по каким-то причинам это сделать не удастся (адрес задан неправильно, узел не существует, компьютер находится не в сети), функция возвратит ненулевое значение.

Прежде, чем сервер сможет использовать сокет, он должен связать его с локальным адресом. Локальный адрес состоит из IP-адреса узла и номера порта. Если сервер имеет несколько IP-адресов, то сокет может быть связан как со вмести ними сразу (для этого вместо IP-адреса следует указать константу INADDR_ANY равную нулю), так и с каким- то конкретным одним. Связывание осуществляется вызовом функции:

int bind (SOCKET s, const struct sockaddr FAR* name, int namelen)

Первым аргументом передается дескриптор сокета, возращенный функций socket, за ним следуют указатель на структуру sockaddr и ее длина. При успешном выполнении функция возвращает нулевое значение и ненулевое в противном случае.

Выполнив связывание, потоковый сервер переходит в режим ожидания подключений, вызывая функцию:

int listen (SOCKET s, int backlog)

Где s – дескриптор сокета, a backlog – максимально допустимый размер очереди сообщений. Извлечение запросов на соединение из очереди осуществляется функцией: SOCKET accept (SOCKET s, struct

sockaddr FAR* addr, int FAR* addrlen)

Функция автоматически создает новый сокет, выполняет связывание и возвращает его дескриптор, а в структуру sockaddr заносит сведения о подключившемся клиенте (IP- адрес и порт). Если в момент вызова accept очередь пуста, функция не возвращает управление до тех пор, пока с сервером не будет установлено хотя бы одно соединение. В случае возникновения ошибки функция возвращает отрицательное значение.

После того как соединение установлено, потоковые сокеты могут обмениваться с удаленным узлом данными, вызывая функции посылки и приема данных:

int send (SOCKET s, const char FAR * buf, int len,int flags)

int recv (SOCKET s, char FAR* buf, int len, int flags)

Функция send возвращает управление сразу же после ее выполнения независимо от того, получила ли принимающая сторона наши данные или нет. При успешном завершении функция возвращает количество передаваемых данных. Если связь прервется до окончания пересылки, данные останутся не переданными, но вызывающий код не получит об этом никакого уведомления. А ошибка возвращается лишь в том случае, если соединение разорвано до вызова функции send.

Функция же recv возвращает управление только после того, как получит хотя бы один байт. Точнее говоря, она ожидает прихода целой дейтаграммы. Дейтаграмма – это совокупность одного или нескольких IP пакетов, посланных вызовом send. Упрощенно говоря, каждый вызов recv за один раз получает столько байтов, сколько их было послано функцией send. При этом подразумевается, что функции recv предоставлен буфер достаточных размеров, в противном случае ее придется вызвать несколько раз. Однако, при всех последующих обращениях данные будет браться из локального буфера, а не приниматься из сети, т. к. TCP-провайдер не может получить часть дейтаграммы, а только ею всю целиком.

При реализации UDP сокетов функции listen и accept не нужны, т.к. UDP протокол не создает соединений, следовательно прослушивать порт и ожидать подключений не нужно. Для передачи данных используются функции sendto и recvfrom (См. пример ниже).

Пример реализации сокетов в Microsoft Visual C++

Пример реализации TCP-сервера

```
// Пример простого TCP-сервера
#include <stdio.h>
#include <winsock2.h> // Wincosk2.h должен быть раньше windows!
#include <windows.h>
#define MY_PORT 123 // Порт, который слушает сервер
// макрос для печати количества активных пользователей
#define PRINTNUSERS if (nclients) printf("%d user
onnline\n",nclients);else printf("No User on line\n");
// прототип функции, обслуживающий подключившихся пользователей
DWORD WINAPI NewClient(LPVOID client_socket);
// глобальная переменная – количество активных пользователей
int nclients = 0;
int main(int argc, char* argv[])
{
        char buff[1024]; // Буфер для различных нужд
        printf("TCP SERVER DEMO\n");
        //Инициализация библиотеки сокетов.
        //Т. к. возвращенная функцией информация не
        //используется, ей передается указатель на рабочий буфер, преобразуемый
        к //указателю на структуру WSADATA. Такой прием позволяет сэкономить
                одну //переменную, однако, буфер должен быть не менее полкилобайта
```

```
        размером //(структура WSADATA занимает 400 байт)
if (WSAStartup(0x0202, (WSADATA *)&buff[0]))
{
        // Ошибка!
        printf("Error WSAStartup %d\n", WSAGetLastError());
        return 1;
}
// Создание сокета
SOCKET mysocket;
// AF_INET - сокет Интернета
// SOCK_STREAM - потоковый сокет (с установкой соединения)
// 0 - по умолчанию выбирается TCP протокол
if ((mysocket = socket(AF_INET, SOCK_STREAM, 0))<0)
{
        // Ошибка!
        printf("Error socket %d\n", WSAGetLastError());
        WSACleanup(); // Деиницилизация библиотеки Winsock
        return 1;
}
// Связывание сокета с локальным адресом
sockaddr_in local_addr;
local_addr.sin_family = AF_INET;
local_addr.sin_port = htons(MY_PORT);
local_addr.sin_addr.s_addr = 0; // сервер принимаем подключения
// на все свои IPPадреса
// вызываем bind для связывания
if (bind(mysocket, (sockaddr *)&local_addr, sizeof(local_addr)))
{
        // Ошибка
        printf("Error bind %d\n", WSAGetLastError());
        closesocket(mysocket); // закрываем сокет!
        WSACleanup();
        return 1;
}
// Ожидание подключений
// размер очереди – 0x100
if (listen(mysocket, 0x100))
{
        // Ошибка
        printf("Error listen %d\n", WSAGetLastError());
        closesocket(mysocket);
        WSACleanup();
        return 1;
}
printf("Ожидание подключений…\n");
// Извлекаем сообщение из очереди
SOCKET client_socket; // сокет для клиента
sockaddr_in client_addr; // адрес клиента (заполняется системой)
// функции accept необходимо передать размер структуры
int client_addr_size = sizeof(client_addr);
```

```
	// цикл извлечения запросов на подключение из очереди
	while ((client_socket = accept(mysocket, (sockaddr *)&client_addr,
		&client_addr_size)))
	{
		nclients++; // увеличиваем счетчик подключившихся клиентов
		// пытаемся получить имя хоста
		HOSTENT *hst;
		hst = gethostbyaddr((char *)&client_addr.sin_addr.s_addr, 4, AF_INET);
		// вывод сведений о клиенте
		printf("+%s [%s] new connect!\n",
			(hst) ? hstt>h_name:"", inet_ntoa(client_addr.sin_addr));
		PRINTNUSERS
			// Вызов нового потока для обслуживания клиента
			// Да, для этого рекомендуется использовать _beginthreadex
			// но, поскольку никаких вызов функций стандартной Си библиотеки
			// поток не делает, можно обойтись и CreateThread
			DWORD thID;
		CreateThread(NULL, NULL, NewClient, &client_socket, NULL, &thID);
	}
	return 0;
}
// Эта функция создается в отдельном потоке
// и обсуживает очередного подключившегося клиента независимо от
остальных
DWORD WINAPI NewClient(LPVOID client_socket)
{
	SOCKET my_sock;
	my_sock = ((SOCKET *)client_socket)[0];
	char buff[20 * 1024];
#define sHELLO "Hello, Sailor\r\n"
	// отправляем клиенту приветствие
	send(my_sock, sHELLO, sizeof(sHELLO), 0);
	// цикл эхоосервера: прием строки от клиента и возвращение ее клиенту
	while ((int bytes_recv = recv(my_sock, &buff[0], sizeof(buff), 0)) &&
		bytes_recv != SOCKET_ERROR)
		send(my_sock, &buff[0], bytes_recv, 0);
	// если мы здесь, то произошел выход из цикла по причине
	// возращения функцией recv ошибки – соединение с клиентом разорвано
	nclientsss; // уменьшаем счетчик активных клиентов
	printf(""disconnect\n"); PRINTNUSERS
		// закрываем сокет
		closesocket(my_sock);
	return 0;
}
```

Пример реализации TCP-клиента

```
// Пример простого TCP-клиента
```

```
#include <stdio.h>
#include <string.h>
#include <winsock2.h>
#include <windows.h>
#define PORT 123
#define SERVERADDR "127.0.0.1"
int main(int argc, char* argv[])
{
	char buff[1024];
	printf("TCP DEMO CLIENT\n");
	// Инициализация библиотеки Winsock
	if (WSAStartup(0x202, (WSADATA *)&buff[0]))
	{
		printf("WSAStart error %d\n", WSAGetLastError());
		return 1;
	}
	// Создание сокета
	SOCKET my_sock;
	my_sock = socket(AF_INET, SOCK_STREAM, 0);
	if (my_sock<0)
	{
		printf("Socket() error %d\n", WSAGetLastError());
		return 1;
	}
	// Установка соединения
	// заполнение структуры sockaddr_in – указание адреса и порта сервера
	sockaddr_in dest_addr;
	dest_addr.sin_family = AF_INET;
	dest_addr.sin_port = htons(PORT);
	HOSTENT *hst;
	// преобразование IP адреса из символьного в сетевой формат
	if (inet_addr(SERVERADDR) != INADDR_NONE)
		dest_addr.sin_addr.s_addr = inet_addr(SERVERADDR);
	else
		// попытка получить IP адрес по доменному имени сервера
	if (hst = gethostbyname(SERVERADDR))
		// hstt>h_addr_list содержит не массив адресов,
		// а массив указателей на адреса
		((unsigned long *)&dest_addr.sin_addr)[0] =
		((unsigned long **)hstt>h_addr_list)[0][0];
	else
	{
		printf("Invalid address %s\n", SERVERADDR);
		closesocket(my_sock);
		WSACleanup();
		return 1;
	}
	// Чтение и передача сообщений
	int nsize;
	while ((nsize = recv(my_sock, &buff[0], sizeof(buff))1, 0)) != SOCKET_ERROR)
```

```
        {
                // ставим завершающий ноль в конце строки
                buff[nsize] = 0;
                // выводим на экран
                printf("S=>C:%s", buff);
                // читаем пользовательский ввод с клавиатуры
                printf("S<=C:"); fgets(&buff[0], sizeof(buff))1, stdin);
                // проверка на "quit"
                if (!strcmp(&buff[0], "quit\n"))
                {
                        // Корректный выход
                        printf("Exit...");
                        closesocket(my_sock);
                        WSACleanup();
                        return 0;
                }
                // передаем строку клиента серверу
                send(my_sock, &buff[0], nsize, 0);
        }
        printf("Recv error %d\n", WSAGetLastError());
        closesocket(my_sock);
        WSACleanup();
        return 1;
}
```

Пример реализации UDP сервера

```
// Пример простого UDP_эхо сервера
#include <stdio.h>
#include <winsock2.h>
#define PORT 666 // порт сервера
#define sHELLO "Hello, %s [%s] Sailor\n"
int main(int argc, char* argv[])
{
        char buff[1024];
        printf("UDP DEMO echo_Server\n");
        // Шаг 1 _ подключение библиотеки
        if (WSAStartup(0x202, (WSADATA *)&buff[0]))
        {
                printf("WSAStartup error: %d\n", WSAGetLastError());
                return _1;
        }
        // Шаг 2 _ создание сокета
        SOCKET my_sock;
        my_sock = socket(AF_INET, SOCK_DGRAM, 0);
        if (my_sock == INVALID_SOCKET)
        {
                printf("Socket() error: %d\n", WSAGetLastError());
                WSACleanup();
```

```
		return _1;
	}
	// Шаг 3 _ связывание сокета с локальным адресом
	sockaddr_in local_addr;
	local_addr.sin_family = AF_INET;
	local_addr.sin_addr.s_addr = INADDR_ANY;
	local_addr.sin_port = htons(PORT);
	if (bind(my_sock, (sockaddr *)&local_addr, sizeof(local_addr)))
	{
		printf("bind error: %d\n", WSAGetLastError());
		closesocket(my_sock);
		WSACleanup();
		return _1;
	}
	// Шаг 4 обработка пакетов, присланных клиентами
	while (1)
	{
		sockaddr_in client_addr;
		int client_addr_size = sizeof(client_addr);
		int bsize = recvfrom(my_sock, &buff[0], sizeof(buff)_1, 0,
			(sockaddr *)&client_addr, &client_addr_size);
		if (bsize == SOCKET_ERROR)
			printf("recvfrom() error: %d\n", WSAGetLastError());
		// Определяем IP_адрес клиента и прочие атрибуты
		HOSTENT *hst;
		hst = gethostbyaddr((char *)&client_addr.sin_addr, 4, AF_INET);
		printf("+%s [%s:%d] new DATAGRAM!\n",
			(hst) ? hst_>h_name:"Unknown host",
			inet_ntoa(client_addr.sin_addr),
			ntohs(client_addr.sin_port));
		// добавление завершающего нуля
		buff[bsize] = 0;
		// Вывод на экран
		printf("C=>S:%s\n", &buff[0]);
		// посылка датаграммы клиенту
		sendto(my_sock, &buff[0], bsize, 0,
			(sockaddr *)&client_addr, sizeof(client_addr));
	} return 0;
}
```

Пример реализации UDP клиента

```
// пример простого UDP_клиента
#include <stdio.h>
#include <string.h>
#include <winsock2.h>
#include <windows.h>
#define PORT 666
#define SERVERADDR "127.0.0.1"
```

```
int main(int argc, char* argv[])
{
	char buff[10 * 1014];
	printf("UDP DEMO Client\nType quit to quit\n");
	// Шаг 1 _ инициализация библиотеки Winsocks
	if (WSAStartup(0x202, (WSADATA *)&buff[0]))
	{
		printf("WSAStartup error: %d\n", WSAGetLastError());
		return _1;
	}
	// Шаг 2 _ открытие сокета
	SOCKET my_sock = socket(AF_INET, SOCK_DGRAM, 0);
	if (my_sock == INVALID_SOCKET)
	{
		printf("socket() error: %d\n", WSAGetLastError());
		WSACleanup();
		return _1;
	}
	// Шаг 3 _ обмен сообщений с сервером
	HOSTENT *hst;
	sockaddr_in dest_addr;
	dest_addr.sin_family = AF_INET;
	dest_addr.sin_port = htons(PORT);
	// определение IP_адреса узла
	if (inet_addr(SERVERADDR))
		dest_addr.sin_addr.s_addr = inet_addr(SERVERADDR);
	else
	if (hst = gethostbyname(SERVERADDR))
		dest_addr.sin_addr.s_addr = ((unsigned long **)hst_>h_addr_list)[0][0];
	else
	{
		printf("Unknown host: %d\n", WSAGetLastError());
		closesocket(my_sock);
		WSACleanup();
		return _1;
	}
	while (1)
	{
		// чтение сообщения с клавиатуры
		printf("S<=C:"); fgets(&buff[0], sizeof(buff)_1, stdin);
		if (!strcmp(&buff[0], "quit\n")) break;
		// Передача сообщений на сервер
		sendto(my_sock, &buff[0], strlen(&buff[0]), 0,
			(sockaddr *)&dest_addr, sizeof(dest_addr));
		// Прием сообщения с сервера
		sockaddr_in server_addr;
		int server_addr_size = sizeof(server_addr);
		int n = recvfrom(my_sock, &buff[0], sizeof(buff)_1, 0,
			(sockaddr *)&server_addr, &server_addr_size);
		if (n == SOCKET_ERROR)
```

```
                {
                        printf("recvfrom() error: %d\n", WSAGetLastError());
                        closesocket(my_sock); WSACleanup();
                        return _1;
                }
                buff[n] = 0;
                // Вывод принятого с сервера сообщения на экран
                printf("S=>C:%s", &buff[0]);
        }
        // Шаг последний _ выход
        closesocket(my_sock);
        WSACleanup();
        return 0;
}
```

Пример реализации сокетов в Delphi

Пример кода TCP-сервера, взаимодействующего с несколькими клиентами и работающего схеме запрос-ответ:

```
var Sockets : array of TSocket;
Addr:TSockAddr;
Data:TWSAData;
Len, I, J:Integer;
FDSet:TFDSet;
begin
        WSAStartup($101, Data);
        SetLength(Sockets, 1);
        Sockets[0]: = Socket(AF_Inet, Sock_Stream, 0);
        Addr.sin_family: = AF_Inet;
        Addr.sin_port: = HToNS(5514);
        Addr.sin_addr.S_addr: = InAddr_Any;
        FillChar(Addr.Sin_Zero, SizeOf(Addr.Sin_Zero), 0);
        Bind(Sockets[0], Addr, SizeOf(TSockAddr));
        Listen(Sockets[0], SoMaxConn);

        while True do
        begin
        // 1. Формирование множества сокетов
        FD_Zero(FDSet);
        for I: = 0 to High(Sockets) do
        FD_Set(Sockets[I], FDSet);
        // 2. Проверка готовности сокетов
        Select(0, @FDSet, nil, nil, nil);
        // 3. Чтение запросов клиентов тех сокетов, которые готовы к этому
        I: = 1;
        while I <= High(Sockets) do
        begin
                if FD_IsSet(Sockets[I], FDSet) then
```

```
                if Recv(Sockets[I], …) <= 0 then
                begin
                        // Связь разорвана, надо закрыть сокет
                        // и удалить его из массива
                        CloseSocket(Sockets[I]);
                        for J: = I to High(Sockets) - 1 do
                                Sockets[J] : = Sockets[J + 1];
                                Dec(I);
                                SetLength(Sockets, Length(Sockets) - 1)
                        end
                        else
                        begin
                                // Поучены данные от клиента, надо ответить
                                Send(Sockets[I], …)
                        end;
                        Inc(I)
        end;
        // 4. Проверка подключения нового клиента
        if FD_IsSet(Sockets[0], FDSet) then
        begin
                // Подключился новый клиент
                SetLength(Sockets, Length(Sockets) + 1);
                Len: = SizeOf(TSockAddr);
                Sockets[High(Sockets)]: = Accept(Sockets[0], @Addr, @Len)
        end
end;
```

2. Задание на лабораторную работу.

Задания представленные ниже выполняются с использованием WinAPI и WSA Socket.

Варианты задания:

Распределенное вычисление площади поверхности фигуры одним из предложенных по варианту методов (в таблице варианты 1-20). Исходными данными для вычисления являются: ● отрезок [A;B]; ● точность вычислений; ● количество клиентов решающих задачу.	4-5 баллов
Распределенное вычисление объема поверхности фигуры одним из предложенных по варианту методов (в таблице варианты 21-32). Исходными данными для вычисления являются: ● отрезок [A;B] и [C;D]; ● точность вычислений; ● количество клиентов решающих задачу.	4-5 баллов
Задания написаны в таблице	6-8 баллов
алгоритм выбрать в соответствии с вариантом задания. Есть один сервер, к которому подключено N клиентов. На сервере указываете матрицу,	9-10 баллов

которая считывается и её части рассылаются клиентам (не полностью !). В результате вы должны получить вектор решений, который сохраняется в текстовый файл. В программе также задается количество клиентов.	

Обязательные компоненты отчета:

- Краткое описание алгоритма
- Описание передаваемых по сети данных
- Верификация с любым мат. пакетом (mathcad, matlab, octave или scilab)
- Сравнение скорости решения для одного клиента, двух и четырех

Варианты согласуются с преподавателем:

1.	Вычислить площадь фигуры, описанную следующей функцией f(x)= $e^{-x}sin(x)$ методом трапеций.
2	Вычислить площадь фигуры, описанную следующей функцией f(x)= $e^{x}arctg(x)$ методом центральных прямоугольников.
3	Вычислить площадь фигуры, описанную следующей функцией f(x)= $x^{3}cos(x)$ методом левых прямоугольников.
4	Вычислить площадь фигуры, описанную следующей функцией f(x)= $e^{cos(x)}x^{2}$ методом центральных прямоугольников.
5	Вычислить площадь фигуры, описанную следующей функцией f(x)= $x^{2}2^{\sqrt[3]{xcos(x)sin(x)}}$ методом правых прямоугольников.
6	Вычислить площадь фигуры, описанную следующей функцией f(x)= $x^{3}sin(x)$ методом Симпсона.
7	Вычислить площадь фигуры, описанную следующей функцией f(x)= $-x^{4}cos(x)$ методом Монте-Карло.
8	Вычислить площадь фигуры, описанную следующей функцией f(x)= $\frac{e^{xcos(x)}}{x^{2}+1}$ методом трапеций.
9	Вычислить площадь фигуры, описанную следующей функцией f(x)= $\frac{2^{xcos(x)sin(x)}}{sin^{2}x+cos^{2}x}$ методом центральных прямоугольников.
10	Вычислить площадь фигуры, описанную следующей функцией f(x)= $x^{3}2^{\sqrt[3]{xsin(x)}}$ методом левых прямоугольников.
11	Вычислить площадь фигуры, описанную следующей функцией f(x)= $x^{3}2^{\sqrt[3]{xcos(x)}}$ методом центральных прямоугольников.
12	Вычислить площадь фигуры, описанную следующей функцией f(x)= $x^{2}2^{\sqrt[3]{xsin(x)}}$ методом правых прямоугольников.
13	Вычислить площадь фигуры, описанную следующей функцией f(x)= $4^{x}e^{cos(x)}$ методом Симпсона.

14	Вычислить площадь фигуры, описанную следующей функцией f(x)=$\frac{e^{xsin(x)}}{x^2+1}$ методом трехточечной квадратуры Гаусса-Лежандра.
15	Вычислить площадь фигуры, описанную следующей функцией f(x)=$2^{x\sqrt{e^{sin(x)cos(x)}}}$ методом трапеций.
16	Вычислить площадь фигуры, описанную следующей функцией f(x)=$2^{x\sqrt{e^{sin(x)}}}$ методом центральных прямоугольников.
17	Вычислить интеграл функции f(x)=$4^x sin(x)$ методом левых прямоугольников.
18	Вычислить площадь фигуры, описанную следующей функцией f(x)=$e^{cos(x)}2^x$ методом центральных прямоугольников.
19	Вычислить площадь фигуры, описанную следующей функцией f(x)=$e^{-cos(x)}2^{\sqrt[3]{sin(x)}}$ методом правых прямоугольников.
20	Вычислить площадь фигуры, описанную следующей функцией f(x)=$x^3e^{cos(x)}$ методом Симпсона.
21	Вычислить объем фигуры, описанный следующей функцией f(x)=$\frac{e^{xcos(y)}}{y^2+1}$ методом трапеций.
22	Вычислить объем фигуры, описанный следующей функцией f(x)=$\frac{2^{ycos(x)sin(x)}}{sin^2y+cos^2y}$ методом центральных прямоугольников.
23	Вычислить объем фигуры, описанный следующей функцией f(x)=$y^3 2^{\sqrt[3]{ysin(x)}}$ методом левых прямоугольников.
24	Вычислить объем фигуры, описанный следующей функцией f(x)=$x^3 2^{\sqrt[3]{ycos(y)}}$ методом центральных прямоугольников.
25	Вычислить объем фигуры, описанный следующей функцией f(x)=$x^2 2^{\sqrt[3]{ysin(x)}}$ методом правых прямоугольников.
26	Вычислить объем фигуры, описанный следующей функцией f(x)=$4^y e^{cos(x)}$ методом Симпсона.
27	Вычислить объем фигуры, описанный следующей функцией f(x)=$\frac{e^{ysin(x)}}{x^2+1}$ методом трехточечной квадратуры Гаусса-Лежандра.
28	Вычислить объем фигуры, описанный следующей функцией f(x)=$2^{y\sqrt{e^{sin(x)cos(y)}}}$ методом трапеций.
29	Вычислить объем фигуры, описанный следующей функцией f(x)=$4^y sin(x)$ методом левых прямоугольников.
30	Вычислить объем фигуры, описанный следующей функцией f(x)=$e^{cos(x)}2^y$ методом центральных прямоугольников.
31	Вычислить объем фигуры, описанный следующей функцией f(x)=$e^{-cos(x)}2^{\sqrt[3]{sin(y)}}$ методом правых прямоугольников.

32	Вычислить объем фигуры, описанный следующей функцией f(x)=$y^3e^{cos(x)}+x^2$ методом Симпсона.
33	Распределенное умножение матриц, распределение по строкам (блоками)
34	Распределенное умножение матриц, распределение по столбцам (блоками)
35	Распределенное умножение матриц, распределение по строкам (циклически)
36	Распределенное умножение матриц, распределение по столбцам (циклически)
37	Распределенное умножение матриц, клеточное распределение
38	Распределенное умножение матриц, через внешнее произведение
39	Распределенная сортировка, для сбора от клиентов использовать mergesort, на клиентах использовать сортировку вставкой
40	Распределенная сортировка, для сбора от клиентов использовать mergesort, на клиентах использовать сортировку mergesort
41	Распределенная сортировка, для сбора от клиентов использовать mergesort, на клиентах использовать сортировку quicksort
42	Распределенная сортировка, для сбора от клиентов использовать mergesort, на клиентах использовать сортировку heapsort
43	Распределенная сортировка, для сбора от клиентов использовать mergesort, на клиентах использовать сортировку treesort (симметричный обход бинарного дерева
44	Распределенная одномерная оптимизация методом дихотомии
45	Распределенная одномерная оптимизация методом градиентного спуска
46	Решение СЛАУ через обратную матрицу, для получения обратной матрицы использовать распределение по столбцам
47	Решение СЛАУ через обратную матрицу, для получения обратной матрицы использовать распределение по строкам
48	Найти обратную матрицу (используя метод Гаусса), распределение по строкам (циклическое)
49	Найти обратную матрицу (используя метод Гаусса), распределение по столбцам (циклическое)
50	Решение СЛАУ методом Гаусса, распределение по столбцам (блоками)
51	Решение СЛАУ методом Гаусса, распределение по строкам (блоками)
52	Решение СЛАУ методом Гаусса, распределение по столбцам (циклическое)
53	Решение СЛАУ методом Гаусса, распределение по строкам (циклическое)
54	Решение СЛАУ методом Гаусса, клеточное распределение

55	Решение СЛАУ методом Жардана-Гауса, распределение по столбцам (циклическое)
56	Решение СЛАУ методом Жардана-Гауса, распределение по строкам(циклическое)
57	Найти определитель матрицы, распределение по строкам (циклическое)
58	Найти определитель матрицы, распределение по столбцам (циклическое)
59	Найти обратную матрицу (через единичную), распределение по строкам (циклическое)
60	Найти обратную матрицу (через единичную), распределение по столбцам (циклическое)

3. Контрольные вопросы

1. Стек протоколов TCP/IP
2. IP-адресация
3. Механизм сокетов
4. Особенности реализации сокетов TCP/IP в Win API.
5. Основные функции необходимые для реализации сетевого взаимодействия по протоколу TCP в ОС Windows.
6. Основные функции необходимые для реализации сетевого взаимодействия по протоколу UDP в ОС Windows
7. Отличия протоколов TCP от UDP.
8. Последовательность вызова функций для взаимодействия по протоколу TCP на стороне клиента в ОС Windows
9. Последовательность вызова функций для взаимодействия по протоколу UDP на стороне клиента в ОС Windows
10. Последовательность вызова функций для взаимодействия по протоколу TCP на стороне сервера в ОС Windows
11. Последовательность вызова функций для взаимодействия по протоколу UDP на стороне сервера в ОС Windows
12. Отличия блокирующих от неблокирующих сокетов в ОС Windows.