**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ   
РОССИЙСКОЙ ФЕДЕРАЦИИ**

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ**

**ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»**

**(БГТУ им. В.Г. Шухова)**



ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И УПРАВЛЯЮЩИХ СИСТЕМ

**Лабораторная работа №2**

по дисциплине: Компьютерные сети

тема: «Протокол сетевого уровня IPX»

Выполнил: ст. группы ПВ-223

Игнатьев Артур Олегович

Проверили:

Рубцов Константин Анатольевич

Белгород 2025 г.

**Цель работы:** изучить протоколы IPX/SPX, основные функции библиотеки Winsock и разработать программу для приема/передачи пакетов.

**Краткие теоретические сведения**

Протокол **IPX** (Internetwork Packet Exchange) является оригинальным протоколом сетевого уровня стека Novell, разработанным в начале 80-х годов на основе протокола Internetwork Datagram Protocol (IDM) компании Xerox

Протокол IPX соответствует сетевому уровню модели OSI и поддерживает только дейтаграммный (без установления соединений) способ обмена сообщениями. В сети NetWare самая быстрая передача данных при наиболее экономном расходовании памяти реализуется именно протоколом IPX.

Для надежной передачи пакетов используется протокол транспортного уровня **SPX** (Sequenced Packet Exchange), который работает с установлением соединения и восстанавливает пакеты при их потере или повреждении. Если по каким-то причинам пакет не дошел до получателя, выполняется его повторная передача. Следовательно, последовательность отправления совпадает с последовательностью получения пакетов. Обмен пакетами на уровне сеанса связи реализован с помощью протокола SPX, который построен на базе IPX.

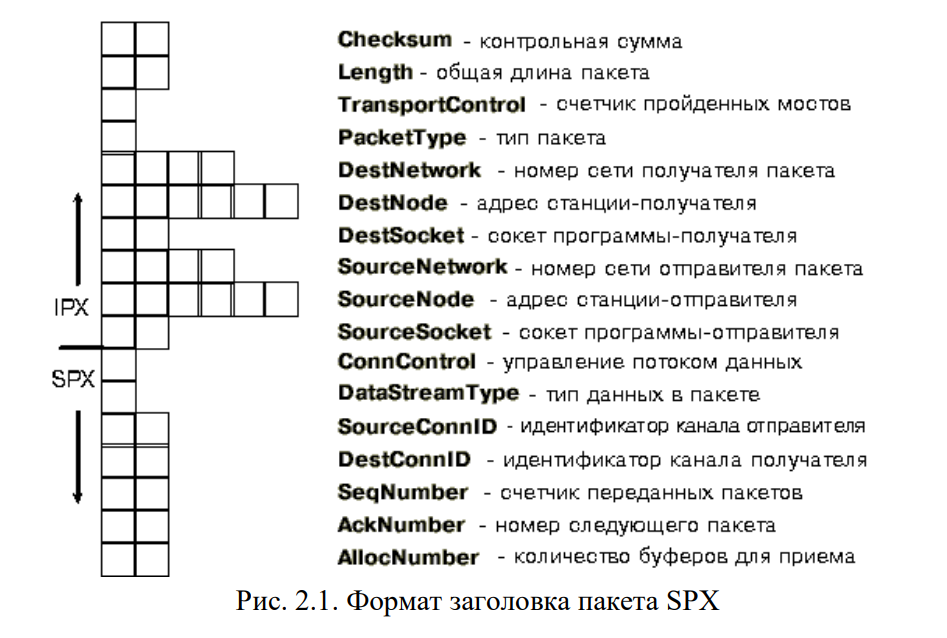
Фирма Novell в сетевой операционной системе NetWare применяла протокол IPX для обмена датаграммами и протокол SPX для обмена в сеансах.

Для некоторых приложений (например, для программ, передающих файлы между рабочими станциями) удобнее использовать сетевой протокол более высокого уровня, обеспечивающий гарантированную доставку пакетов в правильной последовательности. Разумеется, программа может сама следить за тем, чтобы все переданные пакеты были приняты. Однако в этом случае придется делать собственную надстройку над протоколом IPX - собственный протокол передачи данных.

**SPX** – протокол последовательного обмена пакетами (Sequenced Packet Exchange Protocol), разработанный Novell. Система адресов протокола SPX аналогична системе адресов протокола IPX и также состоит из 3 частей: номера сети, адреса станции и сокета.

Протокол SPX использует такой же блок ECB для передачи и приёма пакетов, что и протокол IPX. Однако, пакет, передаваемый при помощи протокола SPX, имеет более длинный заголовок. Дополнительно к 30 байтам стандартного заголовка пакета IPX добавляется еще 12 байт (рис. 2.1).

* Поле **ConnControl** представляет собой как набор битовых флагов, управляющих передачей данных по каналу SPX.
* Поле **DataStreamType** состоит из однобитовых флагов, которые используются для классификации данных, передаваемых или принимаемых при помощи протокола SPX.
* Поле **SourceConnID** содержит номер канала связи передающей программы, присвоенный драйвером SPX при создании канала связи. Этот номер должен указываться функции передачи пакета средствами SPX.



* Поле **DestConnID** содержит номер канала связи принимающей стороны. Так как все пакеты приходят на один номер сокета и могут принадлежать разным каналам связи (на одном сокете можно открыть несколько каналов связи), необходимо классифицировать приходящие пакеты по номеру канала связи.
* Поле **SeqNumber** содержит счетчик пакетов, переданных по каналу в одном направлении. На каждой стороне канала используется свой счетчик. После достижения значения FFFFh счетчик сбрасывается в нуль, после чего процесс счета продолжается. Содержимым этого поля управляет драйвер SPX, поэтому программа не должна менять его значение.
* Поле **AckNumber** содержит номер следующего пакета, который должен быть принят драйвером SPX. Содержимым этого поля управляет драйвер SPX, поэтому программа не должна менять его значение.
* Поле **AllocNumber** содержит количество буферов, распределенных программой для приема пакетов. Содержимым этого поля управляет драйвер SPX, поэтому программа не должна менять его значение.

**Windows Sockets API (WSA)** (сокр. Winsock) – техническая спецификация, которая определяет, как сетевое программное обеспечение Windows будет получать доступ к сетевым сервисам.

**Winsock** – это интерфейс сетевого программирования для Microsoft Windows. Winsock основывается на сокетной парадигме, изложенной в документах под названием Berkley System Distribution от University of California в Berkley.

Основные операционные среды (Unix подобные системы, Windows) базируются в настоящее время на идеологии соединителей (socket). Эта технология была разработана в университете г. Беркли (США) для системы Unix, поэтому соединители иногда называют соединителями Беркли. Соединители реализуют механизм взаимодействия не только партнеров по телекоммуникациям, но и процессов в ЭВМ вообще.

Winsock включает в себя несколько стилей программирования. Первый – это стандартный однопотоковый стиль с блокированием потока определенными командами, второй – с использованием оконных процедур и третий – с использованием асинхронных процедур. Стандартная модель программирования от Berkley является de facto для сетей TCP/IP, но под Windows можно использовать эту библиотеку для программирования протоколов IPX/SPX.

Winsock предназначен для использования во всех версиях MS Windows, начиная с 3.0. Для того чтобы программа могла корректно 21 работать с библиотекой Winsock необходимо проверить версию библиотеки Winsock, а так же вообще наличие этой библиотеки в системе. Библиотека функции Winsock расположена в файле wsock32.dll (ws2\_32.dll для версии 2.0 этой библиотеки) или winsock.dll для 32-бит и 16-бит приложений соответственно. Также, необходимо подключить заголовочные файлы winsock.h (winsock2.h), а для работы с протоколами IPX и SPX еще и заголовочный файл wsipx.h.

**Основные функции API для работы с протоколом IPX**

* **WSAStartup** (WORD wVersionRequested, LPWSADATA lpWSAData) инициализирует библиотеку Winsock. В случае успеха возвращает 0. Дальше можно использовать любые остальные функции этой библиотеки, иначе возвращает код возникшей ошибки. WwVersionRequested – это необходимая минимальная версия библиотеки, при присутствии которой приложение будет корректно работать. Младший байт содержит номер версии, а старший – номер ревизии. LpWSAData – структура, в которую возвращается информация по инициализированной библиотеке (статус, версия и т.д.).
* **WSAGetLastError** (void) возвращает код ошибки, возникшей при выполнении последней операции. После работы с библиотекой, её необходимо выгрузить из памяти.
* **WSACleanup** (void) осуществляет очистку памяти, занимаемой библиотекой Winsock. Функция деинициализирует библиотеку Winsock и возвращает 0, если операция была выполнена успешно, иначе возвращает SOCKET\_ERROR. Расширенный код ошибки можно получить при помощи функции **WSAGetLastError**. Порядок байт на машинах PC отличается от порядка, используемого в сетях, поэтому необходимы некоторые преобразования определенных данных, например, номера порта, чтобы он был правильным при использовании функций библиотеки Winsock. Ниже приведены функции преобразования порядка байт:
* **u\_short htons**(u\_short hostshort);
* **u\_long htonl**(u\_long hostlong);
* **u\_long ntohl**(u\_long netlong);
* **u\_short ntohs**(u\_short netshort);

В качестве параметра передаётся число, которое необходимо преобразовать. Функция возвращает преобразованное число.

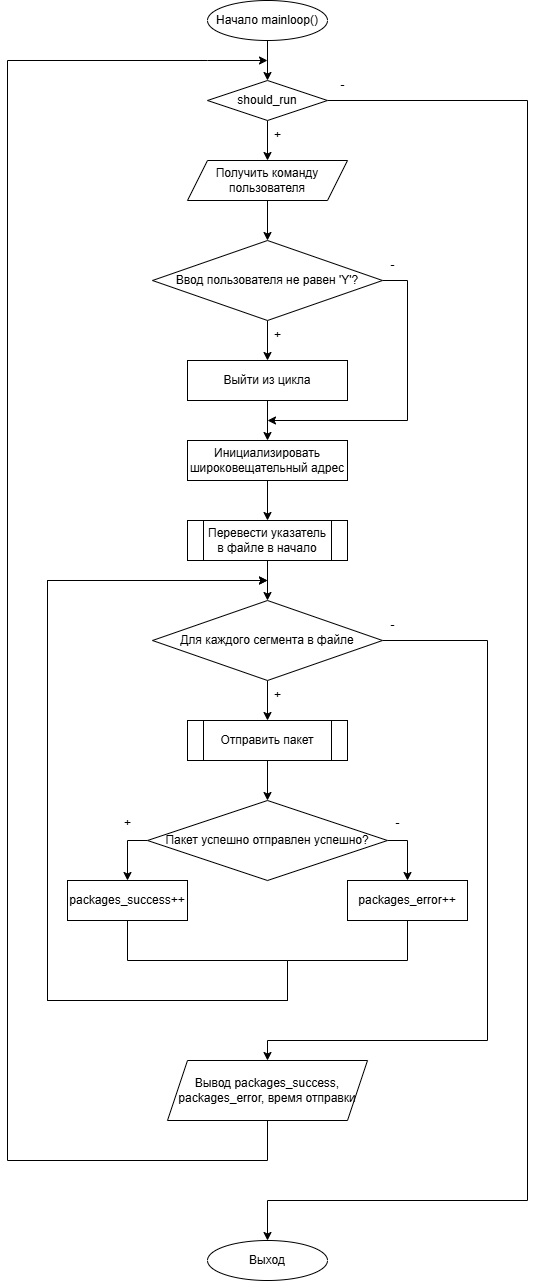
* **socket** (int af, int type, int protocol) возвращает либо дескриптор созданного сокета, либо ошибку INVALID\_SOCKET. Расширенный код ошибки можно получить при помощи функции WSAGetLastError.   
  Параметр af содержит сведения о семействе протоколов (AF\_INET, AF\_IPX). В данной лабораторной работе необходимо использовать константу AF\_IPX. Параметр type – тип передаваемых данных (поток или дейтаграммы). В данной лабораторной работе для IPX необходимо использовать константу SOCK\_DGRAM, а для SPX – константу SOCK\_SEQPACKET, которая означает, что пакеты будут отсылаться последовательно и в порядке очереди. Параметр protocol – протокол передачи данных. Для протокола IPX используется константа NSPROTO\_IPX, для SPX – NSPROTO\_SPX.
* Чтобы работать дальше с созданным сокетом его нужно привязать к какому-нибудь локальному адресу и порту. Этим занимается функция **bind** (SOCKET s, const struct sockaddr FAR\* name, int namelen). Здесь s – дескриптор сокета, который данная функция именует; name – указатель на структуру имени сокета; namelen – размер, в байтах, структуры name. Если порт установить в 0, то система сама пытается подыскать свободный порт. Если в качестве адреса указать константу INADDR\_ANY (0) для сетей TCP/IP или 0 в сетях IPX/SPX, то система попытается использовать все доступные адреса для сокета.
* **listen** (SOCKET s, int backlog) переводит сокет в состояние “прослушивания” (для протокола SPX). Здесь s – дескриптор сокета; backlog – это максимальный размер очереди входящих сообщений на соединение. Эта функция используется сервером, чтобы информировать ОС, что он ожидает запросы связи на данном сокете. Без такой функции всякое требование связи с этим сокетом будет отвергнуто.
* **connect** (SOCKET s, const struct sockaddr FAR\* name, int namelen) используется процессом-клиентом для установления связи с сервером по протоколу SPX. В случае успешного установления соединения connect возвращает 0, иначе SOCKET\_ERROR и номер ошибки можно получить при помощи функции WSAGetLastError.
* **accept** (SOCKET s, struct sockaddr FAR\* addr, int FAR\* addrlen) используется для принятия связи на сокет. Здесь s – дескриптор сокета; addr – указатель на структуру sockaddr; addrlen – размер структуры addr. Сокет должен быть уже слушающим в момент вызова функции. Если сервер устанавливает связь с клиентом, то данная функция возвращает новый сокет-дескриптор, через который и производит общение клиента с сервером. Пока устанавливается связь клиента с сервером, функция блокирует другие запросы связи с данным сервером, а после установления связи “прослушивание” запросов возобновляется.
* В случае автоматического распределения адресов и портов узнать какой адрес и порт присвоен сокету можно при помощи функции **getsockname** (SOCKET s, struct sockaddr FAR\* name, int FAR\* namelen). Здесь s — дескриптор сокета; name — структура sockaddr, в 24 которую система поместит данные; namelen — размер, в байтах, структуры name. Если операция выполнена успешно, возвращает 0, иначе возвращает SOCKET\_ERROR и номер ошибки можно получить при помощи функции WSAGetLastError.
* Передача данных по протоколу IPX осуществляется с помощью функции **sendto** (SOCKET s, const char FAR \* buf, int len, int flags, const struct sockaddr FAR \* to, int tolen). Здесь s - дескриптор сокета; buf - указатель на буфер с данными, которые необходимо переслать; len - размер (в байтах) данных, которые содержатся по указателю buf; flags - совокупность флагов, определяющих, каким образом будет произведена передача данных; to - указатель на структуру sockaddr, которая содержит адрес сокета-приёмника; tolen - размер структуры to. Если операция выполнена успешно, возвращает количество переданных байт, иначе возвращает SOCKET\_ERROR и номер ошибки можно получить при помощи функции WSAGetLastError.
* Передача данных по протоколу SPX осуществляется с помощью функции **send** (SOCKET s, const char FAR \* buf, int len, int flags). Здесь s - дескриптор сокета; buf - указатель на буфер с данными, которые необходимо переслать; len - размер (в байтах) данных, которые содержатся по указателю buf; flags - совокупность флагов, определяющих, каким образом будет произведена передача данных. Если операция выполнена успешно, возвращает количество переданных байт, иначе возвращает SOCKET\_ERROR и номер ошибки можно получить при помощи функции WSAGetLastError.
* Прием данных по протоколу IPX осуществляется с помощью функции **recvfrom** (SOCKET s, char FAR\* buf, int len, int flags, struct sockaddr FAR\* from, int FAR\* fromlen). Если операция выполнена успешно, возвращает количество полученных байт, иначе возвращает SOCKET\_ERROR и номер ошибки можно получить при помощи функции WSAGetLastError.
* Прием данных по протоколу SPX осуществляется с помощью функции **recv** (SOCKET s, char FAR\* buf, int len, int flags). Если операция выполнена успешно, возвращает количество полученных байт, иначе возвращает SOCKET\_ERROR и номер ошибки можно получить при помощи функции WSAGetLastError.

Функция **closesocket**(SOCKET s) служит для закрытия сокета. Возвращает 0, если операция была выполнена успешно, иначе возвращает SOCKET\_ERROR и номер ошибки можно получить при помощи функции WSAGetLastError.

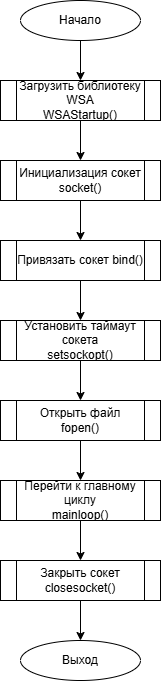
**Разработка программы. Блок-схемы программы**

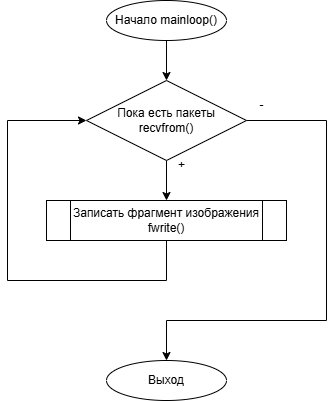
**Сервер IPX**





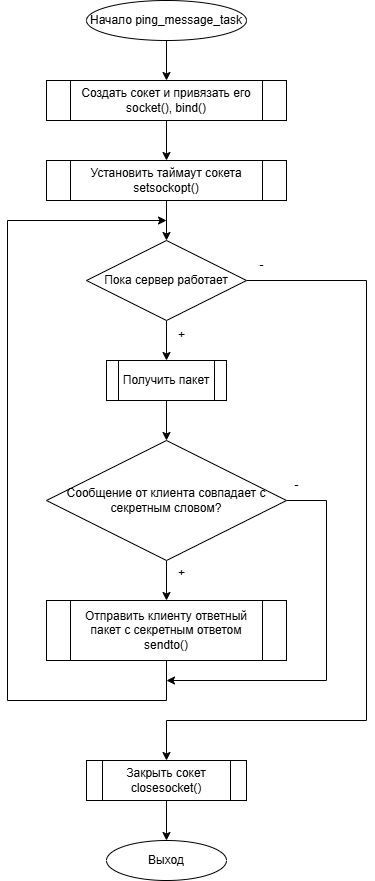
**Клиент IPX**

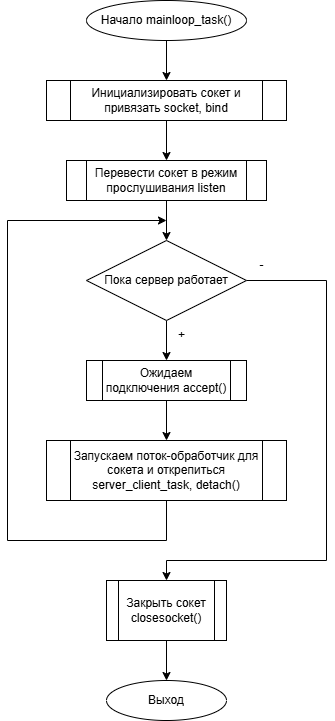


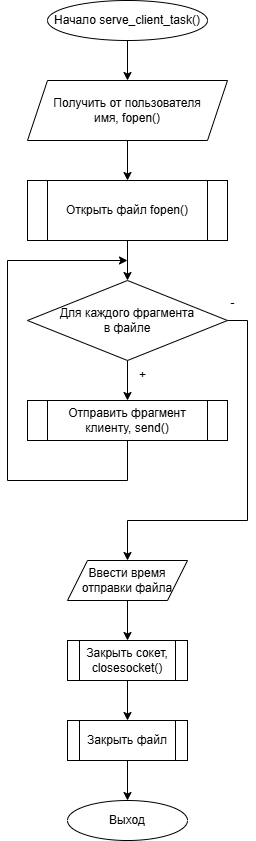


**Сервер SPX**









**Клиент SPX**

