МИНИСТЕРСТВО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ВОЛГОГРАДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

КАФЕДРА ЭВМ И СИСТЕМ

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ КОМПЬЮТЕРНЫХ СЕТЕЙ

Методические указания к лабораторным работам

Волгоград
2010

УДК 681.31

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ КОМПЬЮТЕРНЫХ СЕТЕЙ. Методические указания к лабораторным работам. /Сост. Деревенсков С.О., Стрельников О. И.; Волгоград.гос.техн.ун-т.- Волгоград, 2010. – 30 с.

Содержатся сведения, необходимые для изучения студентами транспортных и прикладных протоколов семейства TCP/IP, основных приемов программирования Internet-приложений на основе этих протоколов с использованием программных интерфейсов сокетов BSD UNIX и Windows Sockets 2.

Предназначены для студентов, обучающихся по направлению 5528 "Информатика и вычислительная техника" и специальности 2201 "Вычислительные машины, комплексы, системы и сети" всех форм обучения.

Ил. 4. Табл. 3. Библиогр. - 4 назв.

Рецензент _____

Печатается по решению редакционно-издательского совета Волгоградского государственного технического университета

Лабораторная работа № 1

Программирование сетевых серверов

Цель работы: изучение транспортных и прикладных протоколов семейства TCP/IP, структуры сетевых приложений, основных приемов программирования Internet-приложений на основе этих протоколов с использованием программных интерфейсов сокетов BSD UNIX и Windows Sockets 2.

Основные сведения

Архитектура сетевого взаимодействия

Развитие сетевых технологий и связанных с ними протоколов обмена данными уже на самых начальных этапах показало очевидную необходимость стандартизации этого процесса. Необходимость стандартизации связана, в первую очередь, с тем, что большинство компьютерных сетей гетерогенны, то есть объединяют компьютерные системы, построенные на совершенно различных аппаратных платформах и использующие различные операционные системы и прикладное программное обеспечение. Однако все эти разнородные системы должны взаимодействовать друг с другом, что невозможно без наличия неких стандартов, существующих как де-юре, так и де-факто.

В 1983 году Международной организацией по стандартизации (ISO) была предложена модель взаимодействия открытых систем (Open Systems Interconnections), обычно называемая семиуровневой моделью OSI.

Согласно этой модели, любая система, участвующая в сетевом взаимодействии, должна строиться по модульному принципу и включать в себя модули семи уровней:

уровень приложений (application layer);
уровень представления данных (presentation layer);
сеансовый уровень (session layer);
транспортный уровень (transport layer);
сетевой уровень (network layer);
канальный уровень (data link layer);
физический уровень (physical layer).

Модули каждого уровня должны обладать стандартными интерфейсами и подключаться в соответствии с ними, образуя “конвейер” обработки данных на передающей и принимающей сторонах.

В реальных системах модули, соответствующие каждому из уровней, функционируют на основе одного или нескольких сетевых протоколов, а набор протоколов всех уровней образует так называемое **семейство** или **стек протоколов**. При этом функции модулей некоторых уровней модели OSI могут выполняться в реальной системе единым аппаратным или программным модулем.

В лабораторной работе рассматриваются вопросы программирования сетевых приложений, использующих наиболее распространенное в настоящее время семейство протоколов – семейство TCP/IP. Протоколы этого семейства

используются как в локальных, так и в глобальных вычислительных сетях, в том числе в Internet. Семейство протоколов TCP/IP поддерживается наиболее распространенными операционными системами, в частности, всеми версиями Windows и UNIX.

При разработке семейства протоколов TCP/IP предполагалось, что коммуникационная структура включает три основных объекта:

процессы, которые обмениваются данными;

хосты, на которых функционируют эти процессы;

сеть передачи данных, к которой подключены хосты.

Исходя из этих соображений, все семейство протоколов TCP/IP разделено на четыре уровня:

прикладной (application layer);

транспортный (host layer);

межсетевой (gateway layer);

сетевой (network layer).

Все протоколы семейства TCP/IP стандартизированы Консультативным советом по координации деятельности Internet (Internet Engineering Task Force - IETF). Стандарты IETF свободно распространяются этой организацией и доступны в Internet в виде рекомендаций серии RFC (Request For Comments). RFC регулируют все вопросы, связанные с использованием семейства протоколов TCP/IP и смежные с ними. Например, описание базового протокола межсетевого уровня – протокола IP, приведено в RFC-791.

Взаимодействие коммуникационных модулей, реализующих протоколы TCP/IP, показано на рис. 1.

В процессе передачи данных они последовательно обрабатываются и передаются от верхнего уровня к нижнему на передающей стороне, и от нижнего к верхнему – на принимающей.

Для передачи данных от одного процесса к другому процессы должны однозначно идентифицироваться в коммуникационной среде. При этом модули каждого уровня используют свою схему адресации. Согласно RFC-790, для идентификации процесса в коммуникационной среде TCP/IP используются следующие адреса:

1) адрес сетевого порта, с которым связан процесс (номер порта);
2) номер транспортного протокола;
3) логический IP-адрес хоста;
4) физический адрес сетевого интерфейса (например, MAC-адрес для Ethernet).

Коммуникационные модули работают как мультиплексоры Nx1 на передающей стороне и как демультиплексоры 1xN – на принимающей стороне. Процесс передачи данных иллюстрируется рис. 2.

Для передачи данных прикладной процесс обращается к программному интерфейсу сокетов операционной системы. Интерфейс сокетов рассматривается далее в этой лабораторной работе. **Сокет** представляет собой программную абстракцию, описывающую конечную точку сетевого соединения. Прикладные процессы могут записывать данные в сокет и читать их из сокета. Все остальные

операции по передаче данных между процессами выполняются средствами операционной системы с использованием соответствующих модулей стека протоколов TCP/IP. При передаче через сокет указывается сетевой порт назначения, который фактически представляет собой номер процесса на принимающей стороне, которому направляются данные. Кроме того, указывается IP-адрес хоста назначения и тип сокета, соответствующий номеру используемого транспортного протокола.

При использовании стека протоколов TCP/IP возможна передача данных посредством транспортных протоколов TCP или UDP. Протокол TCP (RFC-793) является протоколом с надежной доставкой, для чего используется процедура предварительного установления соединения и создается виртуальный канал связи между процессами. При этом процесс на принимающей стороне получает поток данных, в точности соответствующий потоку данных, сформированному процессом на передающей стороне. С точки зрения процессов, участвующих в обмене, объем передаваемых данных неограничен и они нефрагментированы. Протокол UDP (RFC-768) ориентирован на передачу фрагментов данных конечной длины (максимально 568 байт при использовании максимальной длины IP-датаграммы по умолчанию), называемых **датаграммами** (datagrams). Протокол UDP не гарантирует доставку переданной датаграммы принимающей стороне. Соответствующие проверки и квитирование должны выполняться модулями, реализующими прикладной протокол (обычно – прикладными процессами).

Операционная система вызывает модуль транспортного протокола в зависимости от типа сокета. Модуль транспортного протокола формирует пакет TCP или UDP, добавляя к данным заголовок с указанием порта назначения и вызывает модуль IP. Этот модуль, в свою очередь, формирует IP-датаграмму, в которой к пакету транспортного протокола добавляется IP-заголовок. Затем вызывается драйвер сетевого интерфейса, который формирует из IP-датаграммы пакет для передачи по физической среде, например, кадр Ethernet.

При получении пакета, переданного по физической среде, драйвер сетевого интерфейса принимающей стороны начинает обратную процедуру – по заголовку принятого пакета определяется модуль верхнего уровня, которому надо направить содержимое пакета. Если это содержимое – IP-датаграмма, вызывается модуль IP. Он, в свою очередь, по IP-заголовку определяет модуль транспортного протокола. Модуль транспортного протокола из своего заголовка узнает номер порта и передает данные прикладному процессу, связанному с этим портом.

Программный интерфейс сокетов

Интерфейс сокетов реализован средствами операционной системы и представляет собой API для доступа прикладных процессов к модулям протоколов транспортного уровня семейства TCP/IP.

Впервые интерфейс сокетов был разработан в университете г. Беркли (США) и использован в версиях UNIX ветви BSD. В настоящее время он поддерживается всеми версиями UNIX. Интерфейс сокетов Windows реализован как отдельная DLL-библиотека, текущей версией является Windows Sockets 2. Интерфейс Windows Sockets создан на основе сокетов BSD UNIX и включает в себя полный набор

функций этого интерфейса без каких-либо изменений, а также дополнительно ряд функций, специфичных для реализации сокетов в Windows (так называемые WSA-функции).

В связи с этим вначале будут рассмотрены сокеты BSD UNIX, а затем – основные отличительные особенности реализации сокетов в Windows.

В BSD UNIX интерфейс сокетов является частью API ядра, т.е. представляет собой набор системных вызовов. При этом UNIX поддерживает специальный тип файла – сокет. Для обмена данными через сокет в UNIX могут использоваться как специальные системные вызовы для работы с сокетами, так и системные вызовы **read**, **readv**, **write**, **writev**, используемые для работы с файлами различных типов.

Перед началом обмена данными каждый процесс должен создать и открыть сокет при помощи функции **socket**. Этой функции передаются три параметра:

формат адреса (семейство адресов);

тип сокета;

используемый протокол.

Формат адреса определяет используемое семейство адресов (протоколов) и задается при помощи констант AF_*. При использовании TCP/IP указывается AF_INET (в UNIX сокеты могут использоваться с различными семействами протоколов).

Для семейства AF_INET поддерживаются следующие типы сокетов:

сокет потока (тип задается константой SOCK_STREAM) – используется для обмена данными через протокол TCP, при этом в качестве третьего параметра функции **socket** следует указать IPPROTO_TCP либо нуль;

сокет датаграмм (SOCK_DGRAM) – используется для обмена данными через протокол UDP, при этом в качестве третьего параметра функции **socket** следует указать IPPROTO_UDP либо нуль;

сокет низкого уровня (SOCK_RAW) – предназначен для взаимодействия прикладных программ непосредственно с модулем IP, что позволяет прикладной программе самостоятельно формировать IP-датаграмму (в настоящей лабораторной работе сокеты низкого уровня не рассматриваются).

Процедура обмена через сокеты потока показана на рис. 3. На рисунке процесс – инициатор соединения назван клиентом, а процесс ожидающий соединения – сервером.

После создания сокета сервер вызывает функцию **bind**, которая связывает сокет с указанным локальным IP-адресом (IP-адресом одного из сетевых интерфейсов компьютера, где запущен сервер). В качестве параметров она получает дескриптор сокета, указатель на структуру **sockaddr** и длину этой структуры в байтах. В случае AF_INET в качестве второго параметра в UNIX используется указатель на структуру **sockaddr_in**, приведенный к типу **struct sockaddr ***:

```
struct sockaddr_in{
        u_char          sin_len;
        u_char          sin_family;
```

u_short sin_port;
struct in_addr sin_addr;
char sin_zero[8]; };
struct in_addr { **u_long** s_addr; };

В Windows Sockets структуры **sockaddr_in** и **in_addr** определены несколько иначе, но их основные поля sin_family, sin_port и sin_addr.s_addr имеют тот же смысл. Определение названных структур можно найти в заголовочных файлах winsock.h или winsock2.h. Для удобства программиста в этих файлах также определены типы:

typedef **struct sockaddr** FAR * **LPSOCKADDR**;
typedef **struct sockaddr_in** **SOCKADDR_IN**;

В поле sin_port указывается номер сетевого порта либо нуль. В последнем случае система выбирает любой свободный порт в диапазоне 1024 – 5000.

В поле sin_addr.s_addr заносится 32-разрядное число – IP-адрес локального сетевого интерфейса, с которым связывается сокет. Вместо этого может быть указана константа INADDR_ANY, тогда система сама определит требуемый IP-адрес.

Значение IP-адреса может быть также получено с использованием следующих функций:

1) **inet_addr** – формирует 32-разрядное значение IP-адреса из текстовой строки вида “a.b.c.d”; функция **inet_ntoa** выполняет обратную операцию;
2) **gethostbyname** – по доменному имени компьютера возвращает указатель на структуру **hostent**, которая в частности содержит доменное имя хоста, его псевдонимы (альтернативные доменные имена), а также список IP-адресов сетевых интерфейсов, установленных на указанном компьютере.

Доменное имя локального компьютера можно получить при помощи функции **gethostent**, которая также возвращает указатель на структуру **hostent**. Эта же структура возвращается функцией **gethostbyaddr**, параметром которой является IP-адрес, указанный в виде текстовой строки вида “a.b.c.d”. В Windows вместо функции **gethostent** используется функция **gethostname**, возвращающая строку – имя компьютера.

В сети могут находиться компьютеры различной архитектуры, имеющие разный порядок байт в машинном слове. Поэтому при передаче информации по сети используется так называемый сетевой порядок байт. Функции **htons** и **htonl** преобразуют соответственно 16 и 32-разрядные числа к сетевому порядку байт, а функции **ntohs** и **ntohl** выполняют обратное преобразование к машинному слову.

Ниже приведен пример использования функции **bind** при программировании для UNIX. Использование этой функции в приложениях Windows можно изучить на примере программы – сервера, листинг которой представлен в Приложении 1.

```
int sockfd; u_short nport = 500;
struct sockaddr_in servaddr;
sockfd = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
bzero(&servaddr, sizeof(servaddr));
servaddr.sin_port = htons(nport); servaddr.sin_family = AF_INET;
```

servaddr.sin_addr.s_addr = **inet_addr**(“194.226.46.40”);
bind(sockfd, (struct sockaddr *)&servaddr, **sizeof**(servaddr));

После связывания сокета сервер вызывает функцию **listen**, предназначенную для организации очереди поступивших по сети от клиентов запросов на установление соединения. В качестве параметров она получает дескриптор сокета и целое число – максимальный размер очереди запросов. В случае переполнения очереди модуль TCP будет уничтожать принимаемые пакеты TCP и не будет посылать подтверждения на них, что в итоге будет приводить к повторной передаче со стороны клиента.

Затем сервер вызывает функцию **accept**, которая извлекает первый запрос из очереди и создает новый сокет – полную копию сокета, дескриптор которого передан ей в качестве параметра. Этот сокет будет использован для обслуживания нового соединения. Функция **accept** через второй и третий параметры возвращает указатель на структуру **sockaddr** (который для AF_INET приводится к типу **sockaddr_in *** в UNIX или **LPSOCKADDR** в Windows) и длину этой структуры. В результате сервер узнает IP-адрес и порт удаленного сокета, через который клиентом производится попытка установить соединение.

Обычно для обслуживания каждого соединения сервер порождает новый процесс. В UNIX новый процесс порождается системным вызовом **fork** :

```
sockfd = socket ( . . .);
bind (sockfd, . . .);
listen (sockfd, 5);
for (;;)
{       newsockfd = accept (sockfd, . . .);
        if (fork() == 0)
                { /* новый процесс, обслуживающий соединение */
                close (sockfd);
                /* прием – передача через сокет newsockfd */
                close (newsockfd);
                exit(0);       }
        else
                { /* процесс - сервер*/
                close (newsockfd);
                }
}
```

Для обслуживания нового соединения в Windows, как правило, через функцию CreateThread создается параллельный поток, которому передается идентификатор нового сокета, возвращаемый функцией **accept**. Вновь созданный поток производит обмен данными через этот сокет и по завершении обмена закрывает сокет. После вызова CreateThread родительский поток вновь вызывает **accept** для ожидания очередного соединения.

Если **accept** вызывается при пустой очереди, то вызвавший процесс блокируется до тех пор, пока не появится очередной запрос. Для того чтобы избежать блокировки, в UNIX с помощью системного вызова **fcntl** можно указать

для дескриптора сокета флаг O_NONBLOCK, тогда при пустой очереди **accept** будет возвращаться с ошибкой. Это касается и
других рассматриваемых функций, которые по умолчанию являются блокирующими.

Для установления соединения клиент вызывает функцию **connect**, которая имеет тот же набор параметров, что и **bind**, но в структуре **sockaddr** указывается IP-адрес и номер порта сокета сервера, с которым устанавливается соединение.

После установления соединения стороны могут обмениваться данными. При обмене через сокеты потока обычно используются функции **send** и **recv**, но в UNIX стороны могут использовать и файловые функции **write**, **read**, **writev**, **readv**. Все названные функции не имеют параметров, содержащих адреса получателя и отправителя, и поэтому могут использоваться только для сокетов потока после установления соединения. Функции **sendto** и **recvfrom** имеют соответствующие параметры (указатели на структуру **sockaddr**) и могут использоваться для обмена через сокеты любого типа.

При обмене через сокеты потока не сохраняются границы сообщений, поэтому стороны могут использовать произвольную комбинацию из перечисленных выше функций записи и чтения. Передаваемая информация рассматривается просто как поток байт, аналогично файловому вводу/выводу.

Процедура обмена через сокеты датаграмм показана на рис. 4. На рисунке процесс – отправитель назван клиентом, а процесс – получатель – сервером.

При обмене через сокеты датаграмм сохраняются границы сообщений. Функция **sendto** может отправить блок данных длиной не более максимального размера UDP-датаграммы. Каждому вызову **sendto** должен соответствовать вызов **recvfrom**, причем **recvfrom** сразу получает всю информацию (датаграмму), направленную соответствующим вызовом **sendto**. Названные функции являются блокирующими. Процесс, вызвавший **sendto**, блокируется, пока эта функция не отправит данные или истечет установленный системой тайм-аут. Функция **recvfrom** блокирует процесс, пока не поступит очередная датаграмма. В UNIX можно устранить блокирование, установив при помощи системного вызова **fcntl** для сокета режим O_NONBLOCK. Функция **sendto** позволяет также рассылать широковещательные сообщения, для чего в структуре **sockaddr**, являющейся параметром **sendto**, следует вместо адреса получателя указать константу INADR_BROADCAST.

Существуют следующие основные отличия реализации Windows Sockets от сокетов BSD UNIX:

1) так как сокеты Windows реализованы в виде DLL-библиотеки, то прежде чем использовать соответствующие функции приложение должно инициализировать библиотеку вызовом функции **WSAStartup**, а по окончании работы следует отключиться от библиотеки с помощью функции **WSACleanup**;
2) функции сокетов BSD UNIX являются системными вызовами и поэтому они возвращают в случае ошибки –1, а код ошибки помещается в переменную errno; функции Windows Sockets возвращают определенное

для каждой функции значение (обычно SOCKET_ERROR), а характер ошибки можно узнать с помощью функции **WSAGetLastError**;

3) приложения Windows не используют для сокетов функции файлового ввода/вывода;
4) функция **close** BSD UNIX заменена на **closesocket**;
5) в BSD UNIX для того, чтобы избежать блокировок при вызове функций, следует использовать системный вызов **fcntl** как указывалось выше; в Windows Sockets для этого используется функция **WSAAsyncSelect**, через параметр которой передается определяемый программистом код сообщения, которое будет поступать в функцию окна приложения в случае, если произойдет некоторое событие, связанное с сокетом (сокет готов для чтения, готов для записи, установлено соединение и т.п.); фактически сообщение поступает тогда, когда вызов соответствующей блокирующей функции не приведет к блокировке.

В версии Windows Sockets 2 имеются также функции **WSASend**, **WSASendTo**, **WSARecv**, **WSARecvFrom**, **WSAAccept**, **WSAConnect** и другие, которые имеют большие функциональные возможности по сравнению с аналогичными функциями BSD UNIX.

Для работы с функциями Windows Sockets необходимо использовать заголовочные файлы winsock.h (версия Windows Sockets 1.1) или winsock2.h (версия 2.0). Более подробную информацию о Windows Sockets можно получить из справочника Microsoft Windows Sockets 2 Reference (файл sock2.hlp), поставляемого со всеми системами программирования для Windows.

В приложениях к настоящей лабораторной работе приведены исходные тексты приложений tcpclient и tcpserver, иллюстрирующие использование интерфейса Windows Sockets.

Ниже приведен пример программы-сервера, использующего протокол TCP в качестве транспортного протокола:

```
//
// Trivial TCP Server
// Copyright (C) Oleg Strelnikov, 2000-2005
//
#include <stdio.h>
#include <winsock2.h>
const u_short port = 5001;
int main() {
	printf("TCP Server\n");
	// Windows Sockets 2.0
	WSADATA wsaData;
	if (WSAStartup(MAKEWORD(2, 0), &wsaData) != 0) {
		printf("Can't startup Windows Sockets\n");
		return 0;
	}
	printf("Windows Sockets started\n");
```

```
//
SOCKET s = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
if (s == INVALID_SOCKET) {
        printf("Can't create socket\n");
        return 0;
}
printf("Socket Created\n");
//
sockaddr_in sai;
memset(&sai, 0, sizeof (sai));
sai.sin_family = AF_INET;
sai.sin_port = htons(port);
sai.sin_addr.s_addr = INADDR_ANY;
//
if (bind(s, (sockaddr*)(&sai), sizeof (sai)) == SOCKET_ERROR) {
        printf("Bind error\n");
        return 0;
}
printf ("Bind OK\n");
//
if (listen(s, 1) == SOCKET_ERROR) {
        printf("Listen error\n");
        return 0;
}
printf("Listen OK\n");
while (true) {
        //
        printf("\nAccepting...\n");
        struct sockaddr_in saic;
        int addrlen;
        SOCKET sc = accept(s, (sockaddr *) &saic, &addrlen);
        if (sc == INVALID_SOCKET) {
                printf("Accept error\n");
                break;
        }
        printf("Connection opened: %s\n", inet_ntoa(saic.sin_addr));
        printf("Data exchanging...\n");
        //
        char msg[] = "SERVER-HELLO\r\n";
        printf("Sending message...\n");
        send(sc, msg, sizeof (msg), 0);
        //
        while (true) {
                char recvchar;
                int received = recv(sc, &recvchar, 1, 0);
```

```
                if (received == SOCKET_ERROR) {
                        printf("\nSocket Error: receive\n");
                        return 0;
                }
                else if (received == 0) {
                        printf("\nConnection closed by remote host\n");
                        break;
                }
                else {
                        printf("%c", recvchar);
                }
        } //while
        //
        closesocket(sc);
    } //while
    closesocket(s);
    return 0;
} //main()
```

Порядок выполнения работы

Напишите одну из следующих программ по заданию преподавателя. Прикладной протокол разрабатывается студентами. Транспортный протокол TCP. Язык программирования — C или C++, используемый программный интерфейс — Windows Sockets. Продемонстрируйте работоспособность программы при помощи программы **telnet.**

1. Программа – сервер, моделирующая работу отделения банка. Сервер должен предоставлять клиенту возможность открытия и закрытия счета, перечисления и снятия денег со счета, перевода некоторых сумм на другие счета и выполнения других подобных операций.
2. Программа – сервер, моделирующая работу железнодорожной кассы. Сервер должен предоставлять клиенту возможность получения справки о поездах, следующих до указанной станции, расписании их движения, наличия свободных мест, заказа билетов различных классов и выполнения других подобных операций.
3. Программа – сервер, позволяющая получить информацию о товарах и услугах некоторых фирм, их стоимости, сформировать заказ и выполнять другие подобные операции.
4. Программа – сервер, позволяющая получить информацию о наличии книг в библиотеке по разным критериям, добавлять новую книгу, удалять потерянную и выполнять другие подобные операции.
5. Программа – сервер, позволяющая получить список файлов указанного каталога на компьютере, где функционирует сервер, переименовать,

скопировать в другой каталог или удалить указанный файл, запустить программу на выполнение, выполнить перезагрузку компьютера.

Содержание отчета

Отчет по лабораторной работе подготавливается в электронном виде и должен содержать следующие сведения:
название и цель работы;
листинг разработанной программы с комментариями;
результаты работы программы.

Контрольные вопросы

1. Расскажите об архитектуре сетевого взаимодействия TCP/IP, назначении и функционировании модулей, реализующих протоколы каждого уровня.
2. Расскажите об основных особенностях протоколов транспортного уровня TCP/IP.
3. Расскажите о программировании приложений с использованием сокетов потока.
4. Расскажите о программировании приложений с использованием сокетов датаграмм.
5. Расскажите об основных отличиях интерфейса Windows Sockets от интерфейса сокетов BSD UNIX.

Лабораторная работа № 2

Программирование сетевых клиентов

Цель работы: изучение транспортных протоколов семейства TCP/IP, структуры сетевых приложений, основных приемов программирования Internet-клиентов на основе этих протоколов с использованием программных интерфейсов сокетов BSD UNIX и Windows Sockets 2.

Основные сведения

Основные функции программного интерфейса сокетов, используемые при программировании сетевых клиентов совпадают с функциями для серверов, рассмотренными в предыдущей лабораторной работе. Отличает сетевой клиент наличие вызова функции **connect.**

Ниже приведен пример сетевого клиента.

```
//
// Trivial TCP client
// Copyright (C) Oleg Strelnikov, 2004-2005
//

#include <stdio.h>
#include <winsock2.h>

const u_short port = 5001;
const char ip[] = "127.0.0.1";

int main() {

	printf("TCP Client\n");

	 //   Windows Sockets 2.0
	 WSADATA wsaData;
	 if (WSAStartup(MAKEWORD(2, 0), &wsaData) != 0) {
		printf("Can't startup Windows Sockets\n");
		return 0;
	 }
	 printf("Windows Sockets started\n");

	 //
	 SOCKET s = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
	 if (s == INVALID_SOCKET) {
		printf("Can't create socket\n");
		return 0;
```

```
        }
        printf("Socket Created\n");

        //
        sockaddr_in sai;
        memset(&sai, 0, sizeof (sai));
        sai.sin_family = AF_INET;
        sai.sin_port = htons(port);
        sai.sin_addr.s_addr = inet_addr(ip);

        //
        printf("\nConnecting...\n");
        if (connect(s, (sockaddr*)(&sai), sizeof (sai)) == SOCKET_ERROR) {
                printf("Connect error\n");
                return 0;
        }
        printf("Data exchanging...\n");

        //
        char msg[] = "CLIENT-HELLO\r\n";
        printf("Sending message...\n");
        send(s, msg, sizeof (msg), 0);

        //
        while (true) {

                char recvchar;
                int received = recv(s, &recvchar, 1, 0);

                if (received == SOCKET_ERROR) {
                        printf("\nSocket Error: receive\n");
                        return 0;
                }
                else if (received == 0) {
                        printf("\nConnection closed by remote host\n");
                        break;
                }
                else {
                        printf("%c", recvchar);
                }

        } //while
        closesocket(s);
        return 0;
} //main()
```

Порядок выполнения работы

Напишите одну из следующих программ по заданию преподавателя. Вариант задания и прикладной протокол совпадают с вариантом и прикладным протоколом предыдущей работы. Транспортный протокол TCP. Язык программирования — C или C++, используемый программный интерфейс — Windows Sockets. Продемонстрируйте работоспособность программы с использованием сервера, подготовленного на предыдущей лабораторной работе.

1. Программа – клиент, моделирующая работу отделения банка. Сервер должен предоставлять клиенту возможность открытия и закрытия счета, перечисления и снятия денег со счета, перевода некоторых сумм на другие счета и выполнения других подобных операций.
2. Программа – клиент, моделирующая работу железнодорожной кассы. Сервер должен предоставлять клиенту возможность получения справки о поездах, следующих до указанной станции, расписании их движения, наличия свободных мест, заказа билетов различных классов и выполнения других подобных операций.
3. Программа – клиент, позволяющая получить информацию о товарах и услугах некоторых фирм, их стоимости, сформировать заказ и выполнять другие подобные операции.
4. Программа – клиент, позволяющая получить информацию о наличии книг в библиотеке по разным критериям, добавлять новую книгу, удалять потерянную и выполнять другие подобные операции.
5. Программа – сервер, позволяющая получить список файлов указанного каталога на компьютере, где функционирует сервер, переименовать, скопировать в другой каталог или удалить указанный файл, запустить программу на выполнение, выполнить перезагрузку компьютера.

Содержание отчета

Отчет по лабораторной работе подготавливается в электронном виде и должен содержать следующие сведения:
название и цель работы;
листинг разработанной программы с комментариями;
результаты работы программы.

Контрольные вопросы

1. Расскажите об архитектуре сетевого взаимодействия TCP/IP, назначении и функционировании модулей, реализующих протоколы каждого уровня.
2. Расскажите об основных особенностях протоколов транспортного уровня TCP/IP.

3. Расскажите о программировании приложений с использованием сокетов потока.
4. Расскажите о программировании приложений с использованием сокетов датаграмм.
5. Расскажите об основных отличиях интерфейса Windows Sockets от интерфейса сокетов BSD UNIX.

Лабораторная работа № 3

Программирование сетевых приложение с использованием прикладных протоколов

Цель работы: изучение прикладных протоколов семейства TCP/IP, структуры сетевых приложений, основных приемов программирования Internet-приложений на основе этих протоколов

Основные сведения

В настоящее время разработано несколько сотен протоколов прикладного уровня, предназначенных для предоставления различных сетевых услуг. К числу наиболее известных из них относятся протокол передачи гипертекста HTTP, протокол передачи файлов FTP, протоколы электронной почты SMTP, POP3, IMAP4.

Все эти протоколы реализуются модулями прикладного уровня – приложениями или системными программами. Отличительной особенностью рассматриваемых протоколов является обмен информацией при помощи текстовых строк, содержащих команды, данные и ответы на команды, предусмотренные спецификацией протокола. В определенном смысле обмен при помощи прикладных протоколов напоминает диалог людей.

Рассмотрим некоторые прикладные протоколы.

Протокол передачи файлов FTP (File Transfer Protocol, RFC-959) предназначен для передачи файлов между участниками соединения. В качестве протокола транспортного уровня используется TCP. FTP создает два TCP-соединения: одно для обмена командами и ответами на них, другое – для обмена данными.

Сервер FTP всегда находится в состоянии ожидания соединения со стороны клиента по порту 21 TCP, который используется для обмена командами.
Инициатором соединения по каналу команд является клиент, т.е. сервер на соответствующем порту использует функции **listen** и **accept**, а клиент посылает на этот порт запрос на установление соединения при помощи функции **connect**. Для обмена данными FTP-сервер по умолчанию использует порт 20. Клиент по умолчанию использует для обмена данными тот же порт, что и для команд. Однако чаще всего предпочтительнее использовать для обмена данными отдельный порт и со стороны клиента. Для указания серверу номера порта данных клиента используется команда PORT. Инициатором соединения по каналу данных стандартно является сервер, т.е. клиент на соответствующем порту использует функции **listen** и **accept**, а сервер посылает на этот порт запрос на установление соединения при помощи функции **connect**. В некоторых случаях клиент может взять на себя инициативу соединения по каналу данных. Для этого сервер должен быть переведен в пассивный режим командой PASV.

Основные команды протокола FTP приведены в таблице 1, более полную информацию можно получить из RFC-959. К обязательным командам относятся USER, PORT, TYPE, MODE, STRU, RETR, STOR, QUIT, NOOP, остальные команды могут сервером не поддерживаться.

Таблица 1

Основные команды протокола FTP

Команда	Описание
USER *имя пользователя*	Регистрация пользователя (большинство серверов поддерживают имя anonymous для анонимного доступа)
PASS *пароль*	Пароль пользователя (произвольная строка для пользователя anonymous)
TYPE *тип*	Выбор типа передаваемых данных (по умолчанию – тип A – 7-битовая кодировка US ASCII; тип I – 8-битовая передача (двоичные данные); тип E – кодировка EBCDIC)
STRU *структура*	Структура передаваемых данных (по умолчанию – структура F – данные неструктурированы; структура R – данные разделены на записи символами CR LF; структура P – данные разделены на страницы с заголовками специального формата)
MODE *режим*	Режим передачи данных (по умолчанию – режим S – поток байт; режим B – блоки специального формата; режим C – передача с компрессией)
PORT *IP-адрес и порт*	Установка порта данных клиента
PASV	Перевод сервера в пассивный режим
LIST *путь и имя каталога*	Получить содержимое указанного каталога (если не указан – текущего каталога)
PWD	Получить путь и имя текущего каталога
CWD *путь и имя каталога*	Сменить текущий каталог на указанный
RETR *путь и имя файла*	Получить указанный файл
STOR *путь и имя файла*	Записать указанный файл
REST *смещение*	Задает смещение в байтах от начала файла, начиная с которого будет выполняться следующая команда RETR (используется для “докачки” файлов)
DELE *путь и имя файла*	Удалить указанный файл
RNFR *путь и имя файла* RNTO *путь и имя файла*	Переименовать файл (команды следуют друг за другом, RNFR содержит старое имя, RNTO – новое)

MKD *путь и имя каталога*	Создать указанный каталог
RMD *путь и имя каталога*	Удалить указанный каталог
NOOP	Нет операции
ABOR	Прервать выполнение текущей операции, закрыть соединение по каналу данных
QUIT	Завершить текущий сеанс, закрыть соединение по каналам команд и данных

После установления соединения по каналу команд, клиент посылает серверу команды USER и PASS. Затем при необходимости могут быть установлены параметры обмена командами TYPE, STRU и MODE. Далее клиент посылает серверу команды, необходимые для обмена данными. Соединение по каналу данных существует только во время передачи данных. Соединение по порту команд существует постоянно, до тех пор пока клиент не пошлет серверу команду QUIT.

Протоколы систем электронной почты

Любая система электронной почты включает три основных компонента:

1) пользовательский агент – программа, предназначенная для подготовки писем пользователем и их передачи транспортному агенту;
2) транспортный агент – программа-сервер электронной почты, предназначенная для передачи почтовых сообщений по сети;
3) доставочный агент – программа, предназначенная для чтения и обработки пользователем почты, принятой транспортным агентом.

Протокол обмена электронной почты SMTP (Simple Mail Transfer Protocol, RFC-821) предназначен для обмена почтовыми сообщениями между двумя транспортными агентами либо между пользовательским и транспортным агентом.

В терминах SMTP инициатор соединения рассматривается как отправитель почты, а другая сторона – как получатель. Получатель ожидает соединения по порту 25 TCP, отправитель использует любой свободный порт.

Основные команды SMTP приведены в таблице 2. Команды HELO, MAIL FROM: , RCPT TO: , DATA, RSET, NOOP, QUIT являются обязательными, остальные команды могут не поддерживаться.

Таблица 2

Основные команды протокола SMTP

Команда	Описание
HELO *домен*	Идентифицирует домен отправителя
MAIL FROM:<*адрес*>	Идентифицирует адрес отправителя (аргумент *адрес* может быть опущен)
RCPT TO: <*адрес*>	Идентифицирует адрес получателя
DATA	Идентифицирует начало передачи почтового сообщения
RSET	Прекращает текущий сеанс, все выполненные операции и переданные данные аннулируются.

VRFY *имя пользователя*	Проверяет наличие пользователя в домене
EXPN *идентификатор списка*	Позволяет получить список пользователей домена
NOOP	Нет операции
QUIT	Прекращает текущий сеанс, закрывает соединение

Первой передаваемой командой является HELO, идентифицирующая домен отправителя. Затем следует команда MAIL FROM: , которая определяет почтовый адрес отправителя. Этот адрес используется сервером электронной почты для информирования отправителя о невозможности отправки письма в текущий момент и о других ошибках. Далее отправляется одна или несколько команд RCPT TO: , определяющих почтовые адреса получателей. Команда DATA предшествует передаче текста почтового сообщения. Признаком окончания передачи является символ “.”, расположенный в начале пустой строки.

Протокол SMTP не накладывает никаких ограничений на формат передаваемого сообщения. Формат полей заголовка письма определяется RFC-822 и используется при обработке письма доставочным агентом. С точки зрения SMTP электронное сообщение – это набор строк в 7-битовой кодировке US ASCII, заканчиваемый пустой строкой с начальным символом “.”.

Для передачи с письмом двоичной информации разработан стандарт Multipurpose Internet Mail Extensions (MIME), описанный, в частности, в RFC-1341. Этот стандарт позволяет передавать в теле письма информацию в 8-битовой кодировке, для чего используются различные схемы кодирования. Перед передачей тело письма или его часть кодируется в 7-битовую кодировку, а на принимающей стороне доставочный агент производит обратное преобразование.

Следует отметить, что в настоящее время серверы электронной почты используют расширение протокола SMTP – ESMTP (RFC-1651, RFC-1652). Этот протокол содержит дополнительные команды, позволяющие непосредственно передавать данные в 8-битовой кодировке. Протокол ESMTP в настоящей лабораторной работе не рассматривается.

Транспортные агенты могут выполнять перенаправление (relaying) почты, если это не запрещено в настройках агента. При получении очередного письма транспортный агент проверяет, не является ли адресат пользователем домена, который обслуживает этот агент. Если это так, полученное письмо помещается в почтовый ящик пользователя. В противном случае делается DNS запрос о хосте адресата и в случае положительного ответа письмо направляется тому транспортному агенту, в домене которого имеется соответствующая запись DNS. Таким образом на пути от отправителя к получателя письмо может последовательно пройти несколько транспортных агентов. Если очередной агент не может в настоящее время установить соединение со следующим агентом, письмо помещается в очередь и через определенные интервалы времени производятся попытки установить соединение заданное в настройках агента число раз. Информация об этом посылается по электронной почте отправителю

письма.

Используемый доставочным агентом протокол POP3 (Post Office Protocol version 3, RFC-1081) является простейшим протоколом для работы пользователя с содержимым своего почтового ящика. Он позволяет забрать сообщения из почтового ящика сервера на машину клиента и удалить сообщения из ящика.

Для обеспечения работы протокола на машине, где размещаются почтовые ящики, должен функционировать POP3-сервер, а на машине клиента – POP3-клиент. POP3-сервер находится в режиме ожидания входящего соединения по порту 110 TCP. Соединение инициирует клиент. Клиент посылает серверу команды, а сервер – ответы на эти команды и содержимое почтового ящика.

Основные команды протокола POP3 приведены в таблице 3.

Таблица 3. Основные команды протокола POP3

Команда	Описание
USER *имя пользователя*	Регистрация пользователя
PASS *пароль*	Пароль пользователя
STAT	Получить статистику о почтовом ящике
LIST	Показать содержимое почтового ящика
RETR *номер*	Получить сообщение с указанным номером
DELE *номер*	Удалить сообщение с указанным номером
QUIT	Завершить сеанс, закрыть соединение

Обмен начинается с фазы аутентификации, на которой клиент посылает команды USER и PASS, происходит регистрация клиента и проверка его прав.

На фазе транзакции клиент работает с почтовым ящиком при помощи команд STAT, LIST, RETR, DELE.

После посылки клиентом команды QUIT сервер выполняет фазу завершения, на которой реально производятся все изменения, внесенные пользователем в текущем сеансе: удаляются удаленные командой DELE сообщения, помечаются сообщения, прочтенные командой RETR, но оставленные в ящике.

Задание 1

1.1. Используя утилиту **telnet,** установите соединение по каналу команд с сервером FTP.
1.2. Зарегистрируйтесь на сервере и установите параметры обмена.
1.3. Переведите сервер в пассивный режим, запустите вторую копию утилиты **telnet** и установите соединение по каналу данных. Повторяйте эту процедуру перед выполнением каждой команды, передающей данные от сервера.
1.4. Получите содержимое корневого каталога сервера и нескольких его подкаталогов.
1.5. Получите несколько файлов с FTP-сервера.
1.6. Создайте подкаталог в каталоге **incoming** сервера, отправьте файл в этот каталог и получите его обратно. Удалите созданный файл и подкаталог.

1.7. Закройте соединение с сервером FTP.
1.8. Запишите посылаемые Вами команды и ответы сервера в протокол лабораторной работы.

Задание 2

2.1. Используя утилиту **telnet,** установите соединение с SMTP-сервером.
2.2. Проверьте, зарегистрированы ли на сервере адресаты Ваших почтовых сообщений.
2.3. Передайте несколько почтовых сообщений одному адресату.
2.4. Передайте одно почтовое сообщение сразу нескольким адресатам.
2.5. Закройте соединение с сервером SMTP.
2.6. Запишите посылаемые Вами команды и ответы сервера в протокол лабораторной работы.

Задание 3

3.1. Используя утилиту **telnet,** установите соединение с POP3-сервером.
3.2. Зарегистрируйтесь на сервере и получите статистику о почтовом ящике.
3.3. Получите список сообщений в почтовом ящике.
3.4. Последовательно получите все сообщения из ящика и удалите их.
3.5. Закройте соединение с сервером POP3.
3.6. Запишите посылаемые Вами команды и ответы сервера в протокол лабораторной работы.

Содержание отчета

Отчет по лабораторной работе подготавливается в электронном виде и должен содержать следующие сведения:
название и цель работы;
описание выполняемых операций и результаты их выполнения по каждому заданию.

Контрольные вопросы

1. Расскажите об архитектуре сетевого взаимодействия TCP/IP, назначении и функционировании модулей, реализующих протоколы каждого уровня.
2. Расскажите об основных особенностях протоколов прикладного уровня TCP/IP.
3. Расскажите о программировании приложений с использованием протокола FTP.
4. Расскажите о программировании приложений с использованием протоколов SMTP и POP3.

Лабораторная работа № 4

Изучение основных сетевых утилит

Цель работы: Изучить основные команды утилиты openssl.

Основные сведения

В связи с развитием Internet, в том числе коммерческих приложений, возникает задача защиты информации при передачи по глобальным и локальным сетям и при хранении на серверах Internet. В частности, если пользователь А передает сообщение В, которое может быть перехвачено злоумышленником, то система должна обеспечить конфиденциальность, целостность, идентификацию отправителя.

Конфиденциальность означает, что никто, кроме В не может раскрыть смысла сообщения. Целостность подразумевает, что если сообщение было изменено при передаче, то В может установить этот факт. Идентификация отправителя – достоверное установление авторства отправителя сообщения.

В системах цифровой связи для обеспечения защиты передаваемой информации, в частности для обеспечения конфиденциальности, аутентификации и идентификации используются криптографические методы, выполняющие преобразование:

$$T_C = f(T_O, K), \quad (1)$$

где T_O – исходные защищаемые данные, подлежащие зашифрованию перед передачей, T_C – зашифрованные данные, K – ключ, известный сторонам обмена, f – криптографическое преобразование.

Для получения исходного текста из зашифрованного выполняется обратное преобразование относительно T_O:

$$T_O = f^{-1}(T_C, K). \quad (2)$$

Решение исходной задачи защиты T_O определяется практической невозможностью решения уравнения (1) относительно T_O, в частности, относительно K.

В общем случае ключ K должен быть известен участникам обмена данными и только им, при этом ключ, используемый как для зашифрования (1), так и для расшифрования (2) называется закрытым, а система шифрования – системой с закрытым ключом или симметричной системой шифрования.

В частном случае ключ K может быть представлен в виде пары $K = (K_O, K_C)$, причем для шифрования применяется преобразование:

$$T_C = g(T_O, K_O), \quad (3)$$

где g – функция криптографического преобразования.

Для получения исходного текста выполняется преобразование:

$$T_O = g^{-1}(T_C, K_C). \quad (4)$$

Основные команды OpenSSL

Команда	Назначение
dsaparam	Генерация параметров p, q, g для алгоритма DSA
gendsa	Генерация секретного ключа для алгоритма DSA
genrsa	Генерация секретного ключа для алгоритма RSA
version	Информация о версии openssl
dhparam	Генерация параметра (простого числа) для алгоритма DH
pkcs12	Создание цифрового удостоверение PKCS#12
req	Создание запросов сертификата (CSR) PKCS#10 и корневых сертификатов X.509
s_client	Клиентский терминал для подключения к серверу SSL
s_server	Серверный терминал для подключения клиентов SSL
x509	Создание сертификатов подписыванием запросов

Структуры openssl, такие как открытые и закрытые ключи, сертификаты и запросы сертификатов, могут храниться и передаваться в одном из трех форматов:

ASN1 – двоичный формат;

PEM – формат ASN1 закодированный алгоритмом base64 для представления в виде последовательности печатных символов;

текстовый – представление в текстовом (строковом) виде.

Рассмотрим типичные задачи

1. Создание корневого сертификата

openssl req -new -x509 -text -newkey rsa:<Bits> -out <CAFile> -keyout <KeyFile> -days <D> -sha1 -nodes

Параметры:

-new – создать новый запрос сертификата или корневой сертификат;

-x509 – создать новый корневой сертификат;

-text – включить в текст сертификата текстовые комментарии;

-newkey rsa:<Bits> - использовать алгоритм RSA с указанной длиной закрытого ключа в битах;

-out <CAFile> - имя файла для сохранения создаваемого сертификата;

-keyout <KeyFile> - имя файла для создаваемого ключа;

-days <D> - количество дней действительности сертификата;

-sha1 – использовать алгоритм хэширования SHA1;

-nodes – не шифровать файл с закрытым ключом алгоритмом DES (на практике не рекомендуется указывать данный параметр).

Команда в интерактивном режиме предлагает при необходимости ввести пароль для шифрования файла с закрытым ключом, а также параметры сертификата. Если соответствующий параметр не должен присутствовать в сертификате, необходимо указать символ ".". Обязательно должен быть указан по крайней мере параметр "Common Name".

Пример:

openssl req -new -x509 -text -newkey rsa:2048 -out localhost1.crt -keyout localhost1.key -days 3650 -sha1 -nodes

Generating a 2048 bit RSA private key

.....+++

.....+++

writing new private key to 'localhost1.key'

You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.

Country Name (2 letter code) [AU]:.
State or Province Name (full name) [Some-State]:.
Locality Name (eg, city) []:
Organization Name (eg, company) [Internet Widgits Pty Ltd]:.
Organizational Unit Name (eg, section) []:
Common Name (eg, YOUR name) []:Localhost CA
Email Address []:

localhost1.crt

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 0 (0x0)
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: CN=Localhost CA
        Validity
            Not Before: Nov  8 14:06:59 2005 GMT
            Not After : Nov  6 14:06:59 2015 GMT
        Subject: CN=Localhost CA
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (2048 bit)
                Modulus (2048 bit):
                    00:c8:db:36:d8:18:3f:93:d4:69:e4:71:cb:ad:49:
                    2e:a6:ec:b9:de:99:6b:53:db:59:63:1b:89:cb:b8:
                    3e:a5:14:58:36:4a:59:02:f6:2d:34:84:d0:31:99:
                    8f:45:1e:e7:62:bd:49:f4:95:ff:e2:c6:e9:41:5a:
                    52:66:e4:f3:a3:6e:4f:19:4a:c6:a8:34:04:59:7e:
                    f9:4a:3d:ac:23:4c:0b:a2:40:37:c6:cc:69:27:e6:
                    24:20:27:94:91:5a:d9:db:ab:90:a5:95:d4:68:76:
                    7d:96:f8:af:71:e7:64:cb:55:1e:c9:fe:a6:f2:4d:
                    a9:20:cd:77:63:c0:a0:5a:bd:7b:ec:b0:aa:9c:76:
                    4a:a4:d9:03:f8:d0:14:b5:b5:ce:a9:ca:58:33:d6:
                    84:bd:93:f0:ba:d5:86:2a:8d:ed:f2:21:85:92:e2:
```

```
                    43:33:69:67:db:19:ce:42:d2:7d:dc:76:23:0f:91:
                    10:ce:9b:a6:5e:69:aa:20:22:2c:49:78:44:5a:63:
                    98:4a:9d:ec:cb:38:8b:0e:f2:7e:fb:0f:51:e3:b3:
                    15:f2:97:0f:4f:a7:ac:d9:be:6b:be:0c:12:aa:d4:
                    fa:68:ec:be:67:9e:2c:c4:6d:c2:cc:69:12:3f:60:
                    0c:85:cd:28:06:ec:c4:ea:78:6e:75:81:b7:0a:fe:
                    11:53
                Exponent: 65537 (0x10001)
        X509v3 extensions:
            X509v3 Subject Key Identifier:
                8A:57:B9:8F:1E:69:2B:FA:0B:AA:2D:63:BB:85:8C:92:B4:F
1:AB:A6
            X509v3 Authority Key Identifier:
                keyid:8A:57:B9:8F:1E:69:2B:FA:0B:AA:2D:63:BB:85:8C:9
2:B4:F1:AB:A6
                DirName:/CN=Localhost CA
                serial:00

            X509v3 Basic Constraints:
                CA:TRUE
    Signature Algorithm: sha1WithRSAEncryption
        00:42:33:01:0d:61:23:3f:ce:08:95:bd:d2:6c:f6:90:c1:d8:
        2f:55:fb:da:7c:7e:b3:a1:b7:b4:23:68:a5:08:9e:77:b9:27:
        a7:66:5a:17:24:00:23:66:d0:44:38:7a:31:ce:48:63:0a:47:
        bb:37:e2:b6:7c:8a:1b:2b:65:d3:f4:50:b5:d3:84:45:9b:10:
        ff:b0:b0:55:fc:ad:8e:20:b7:0d:96:13:d9:12:c2:a4:0d:ad:
        80:06:eb:3f:44:ed:f9:35:16:6d:87:9c:66:10:6c:25:59:09:
        88:94:92:f9:a0:fd:a5:e2:f6:10:8d:12:51:c6:b9:57:7b:1d:
        97:5b:c4:9a:ff:55:d8:37:99:46:ca:83:21:98:56:fd:a7:81:
        86:27:2f:5f:a7:aa:c6:f0:c1:d9:93:53:97:ba:b8:35:a9:bc:
        48:db:0e:52:8d:64:05:25:cc:5b:8d:2f:8e:3d:96:b6:6c:ca:
        26:52:31:23:a0:9a:a6:76:62:43:c7:78:3c:59:c4:5b:d4:b8:
        3e:cf:04:11:ba:a0:90:54:ce:1f:a7:2f:f2:65:89:5c:0b:18:
        7b:0c:1f:1e:51:3e:09:a1:a7:1b:87:24:24:21:95:ce:44:83:
        d2:62:ef:4d:f5:c5:30:1d:76:ed:89:6d:bf:5f:43:34:c5:f3:
        f4:b3:28:a5
-----BEGIN CERTIFICATE-----
MIIDGTCCAgGgAwIBAgIBADANBgkqhkiG9w0BAQUFADAXMRUwEwYDVQQDEwxMb2Nh
bGhvc3QgQ0EwHhcNMDUxMTA4MTQwNjU5WhcNMTUxMTA2MTQwNjU5WjAXMRUwEwYD
VQQDEwxMb2NhbGhvc3QgQ0EwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwggEKAoIB
AQDI2zbYGD+T1GnkccutSS6m7LnemWtT21ljG4nLuD6lFFg2SlkC9i00hNAxmY9F
HudivUn0lf/ixulBWlJm5POjbk8ZSsaoNARZfvlKPawjTAuiQDfGzGkn5iQgJ5SR
Wtnbq5CclldRodn2W+K9x52TLVR7J/qbyTakgzXdjwKBavXvssKqcdkqk2QP40BS1
tc6pylgz1oS9k/C61YYqje3yIYWS4kMzaWfbGc5C0n3cdiMPkRDOm6ZeaaogIixJ
eERaY5hKnezLOIsO8n77D1HjsxXylw9Pp6zZvmu+DBKq1Ppo7L5nnizEbcLMaRI/
YAyFzSgG7MTqeG51gbcK/hFTAgMBAAGjcDBuMB0GA1UdDgQWBBSKV7mPHmkr+guq
LWO7hYyStPGrpjA/BgNVHSMEODA2gBSKV7mPHmkr+guqLWO7hYyStPGrpqEbpBkw
FzEVMBMGA1UEAxMMTG9jYWxob3N0IENBggEAMAwGA1UdEwQFMAMBAf8wDQYJKoZI
hvcNAQEFBQADggEBAABCMwENYSM/zgiVvdJs9pDB2C9V+9p8frOht7QjaKUInne5
J6dmWhckACNm0EQ4ejHOSGMKR7s34rZ8ihsrZdP0ULXThEWbEP+wsFX8rY4gtw2W
E9kSwqQNrYAG6z9E7fk1Fm2HnGYQbCVZCYiUkvmg/aXi9hCNElHGuVd7HZdbxJr/
Vdg3mUbKgyGYVv2ngYYnL1+nqsbwwdmTU5e6uDWpvEjbDlKNZAUlzFuNL449lrZs
yiZSMSOgmqZ2YkPHeDxZxFvUuD7PBBG6oJBUzh+nL/JliVwLGHsMHx5RPgmhpxuH
```

```
JCQhlc5Eg9Ji7031xTAddu2Jbb9fQzTF8/SzKKU=
-----END CERTIFICATE-----
```

Здесь сертификат представлен в текстовом формате и в формате PEM.

localhost1.key

```
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEAyNs22Bg/k9Rp5HHLrUkupuy53plrU9tZYxuJy7g+pRRYNkpZ
AvYtNITQMZmPRR7nYr1J9JX/4sbpQVpSZuTzo25PGUrGqDQEWX75Sj2sI0wLokA3
xsxpJ+YkICeUkVrZ26uQpZXUaHZ9lvivcedky1Ueyf6m8k2pIM13Y8CgWr177LCq
nHZKpNkD+NAUtbXOqcpYM9aEvZPwutWGKo3t8iGFkuJDM2ln2xnOQtJ93HYjD5EQ
zpumXmmqICIsSXhEWmOYSp3syziLDvJ++w9R47MV8pcPT6es2b5rvgwSqtT6aOy+
Z54sxG3CzGkSP2AMhc0oBuzE6nhudYG3Cv4RUwIDAQABAoIBAQCARF3VykZvpWqq
VHF5jA6a0vovyVHdD2sZ+3AYBOUcjs12fiwx0/wyWc6KnkH8JiwlxjQAWfo7XEOB
VKjixLtCVyN0R2ht6ioM7SVN3+yZ7epQByqXbdqJfFL9pckeuiH+QqXw2Kn4vBKf
4thHqTmZEBbFcDY07ptfM1dXAKBm0REvYUKaFVbmmjvcfaYNjLjxnK7NvyLT7FnK
E9rrtFBWVYP+wBNPpRx4L1RcpSBIH1xyQY4U9urJCrJKUNg/qatYYnm2/d5XDY4+
FGzr2OcfovCtLkOtGy/L1SBZ272bNL7c1V2xsTDfQtjV3ZtEeyEc6XXW0s30PrEV
eePDM1vBAoGBAOeXPq0+a2q6+epbQRH+TOaRwc+4V5A6dyISc/RE6sF/FRiuYjbN
vRxkG6J49UA7i+eEURRYUn3oWkvOkybsK305ACBXMtlDibGM65yQuha6QsHA0JoA
rnivrWe5nP+RewPyOlb3zCPYOnqrUPwQBL+3uRKdaC9WsmLzgmJ/VU2pAoGBAN4G
sekKBceLa9g49wFHHOu5p0xNgp8m5poco5nHNHWdUQReAPQN1VNV11BXtzzQQtll
b5jjxS8ExJnRCbVHwc92Duvw08miLnyoQLTbF0lZesyF8uKn965ohofHfFCK46B5
8xqwZWjOhfruVZDTfgJZ5ICsrSVmrTokJcj6ZyybAoGAbMaPO9P7pA/aXocqEwd8
mzeO7r/I8Qdk+W3tZKgSE3xbLlKJ8u/DiVhD2pYxq0/MsJtnccTiSh0efEi0uS1o
Z9KkCoyqZ33faLOl3s3jedVWkaa80/jJ+gmZywRVh3yPaRUW7c5J5fQj04Yv7ZSg
hmQG24Vvb/yqoOt7LERejrECgYAHR8brslVwLqUxqOSImTFclXDj8qK230qugQPo
VfzlYmw2rXJwjkeRfP4bd1Td3KJkiIv5QnbDzO85b/5WgGzdhayFlhcLxJKrqcS9
aXXBBPse28GUJxfbEzheGW99IKDIMVeEZ4ZbwnHoQYpyvkyifRoeEgt3b398qZTH
m1ZowwKBgEOtUt7S4MIWlsM7lSKd20dYNdctx8Vnbty6wqFmRQ21M55SQFF/zw+j
0CY9Syc9S4lDbp71jY7Z3LCJABFizZucROAzXm7Iut06+8pakqVCqV0g7AIEeAeU
gL8hJ/QCvBVsf3MNMC8FmUf8LTTyBxIvA4U3KjGjmxaHMLh+TloT
-----END RSA PRIVATE KEY-----
```

Закрытый ключ представлен в формате PEM.

2. Создание запроса сертификата

openssl req -new -text -newkey rsa:<Bits> -out <CSRFile> -keyout <KeyFile> -sha1 -nodes

Назначение параметров точно такое же, как и в случае создания корневого сертификата. Отсутствие параметра -x509 означает, что будет создан не корневой сертификат, а запрос сертификата. Параметр -days не используется, поскольку срок действия сертификата определяется во время подписывания запроса сертификата. Запросы сертификатов в дальнейшем должны быть преобразованы в сертификаты. Если создаваемый запрос сертификата будет использоваться для создания сертификата сервера, то параметр "Common Name" создаваемого запроса должен содержать доменное имя или IP-адрес сервера.

Пример:

openssl req -new -text -newkey rsa:2048 -out localhost2.csr -keyout localhost2.key -sha1 -nodes

Generating a 2048 bit RSA private key

.....+++

.....+++

writing new private key to 'localhost2.key'

You are about to be asked to enter information that will be incorporated into your certificate request.

What you are about to enter is what is called a Distinguished Name or a DN.

There are quite a few fields but you can leave some blank

For some fields there will be a default value,

If you enter '.', the field will be left blank.

Country Name (2 letter code) [AU]:.

State or Province Name (full name) [Some-State]:.

Locality Name (eg, city) []:

Organization Name (eg, company) [Internet Widgits Pty Ltd]:.

Organizational Unit Name (eg, section) []:

Common Name (eg, YOUR name) []:localhost

Email Address []:

Please enter the following 'extra' attributes

to be sent with your certificate request

A challenge password []:

An optional company name []:

localhost2.csr

```
Certificate Request:
    Data:
        Version: 0 (0x0)
        Subject: CN=localhost
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (2048 bit)
                Modulus (2048 bit):
                    00:ad:2c:56:e3:9b:5b:45:d8:87:11:45:9c:5c:1b:
                    ad:79:5c:c7:ec:c2:93:18:75:45:0b:81:0d:62:a6:
                    6a:b7:e1:9a:08:89:f9:31:6c:31:4c:e7:22:c4:ad:
                    52:30:67:0a:76:c4:a6:1e:0b:16:07:70:c2:94:62:
                    ae:74:3b:b6:f7:e9:4a:64:11:ed:02:d0:5c:15:09:
                    e8:c6:f3:47:3d:be:1b:7b:0a:15:9f:fe:6a:a8:8f:
                    4b:04:e6:71:0b:9b:ca:23:fc:ca:80:9d:3b:5a:1c:
                    b6:c3:6f:c5:ca:e1:70:26:dd:a1:a8:9c:cb:40:6b:
                    74:22:40:bc:be:30:67:b8:1c:78:92:0e:a5:ac:fb:
                    72:ab:86:6d:88:24:84:53:c2:64:d9:26:ff:d6:e2:
                    10:ed:de:db:fb:79:c7:71:d5:6b:d3:55:2e:6d:b0:
                    7e:68:73:11:38:0d:5a:1a:aa:53:8a:b1:ac:d6:77:
```

```
                    c2:4b:cc:e4:71:68:14:5e:ea:b4:f8:c1:1c:cf:15:
                    66:cb:6a:d5:7b:4a:2a:ae:ec:6e:f0:9d:d3:a6:da:
                    85:4e:14:02:87:87:d3:39:d4:b1:38:e0:86:68:7f:
                    a6:06:05:b9:9b:da:4c:12:b2:ea:49:ec:fa:b9:e1:
                    84:5c:a2:c9:8b:2f:06:91:b9:2e:21:3a:6c:4e:63:
                    47:e9
                Exponent: 65537 (0x10001)
        Attributes:
            a0:00
    Signature Algorithm: sha1WithRSAEncryption
        93:99:de:c5:6b:d4:69:8c:a8:d9:7c:4b:d0:09:6b:5a:9b:86:
        60:27:5d:cb:b3:79:7e:fa:66:10:b4:1f:7c:cd:4e:f5:05:9a:
        0a:13:96:73:34:71:b2:d7:ff:ea:26:91:e0:4b:0d:79:c7:91:
        68:1b:50:7d:cc:58:3e:63:77:65:a2:db:73:47:d4:14:d2:9c:
        12:d8:0f:27:39:83:81:f7:fe:8f:c3:bf:cb:c2:4f:a9:c8:9a:
        ba:9a:03:8d:6b:4a:53:cd:2f:45:e5:ab:a7:32:b7:f3:1a:7f:
        4d:07:e2:71:76:3f:8b:00:e7:b4:e1:ea:ea:36:04:98:b9:7f:
        8e:0b:8e:0e:19:a0:85:ad:72:d7:a2:2c:9c:ad:ac:11:df:a8:
        03:c8:68:d0:4b:87:09:fa:c3:bc:9f:ae:e1:00:0c:6d:23:2e:
        16:60:99:b7:fa:e5:35:c7:44:43:84:c3:77:b0:fb:af:93:f2:
        43:d6:d4:6a:71:70:76:c4:66:b1:1b:b5:f0:6f:9d:da:f5:84:
        b9:50:95:54:1b:0f:3a:65:ac:cc:4e:27:cf:8d:13:ea:45:4c:
        70:37:03:b3:85:b7:be:07:02:75:72:dc:05:91:38:16:94:3f:
        50:83:a2:a8:43:0e:21:24:99:d9:fc:a8:9f:0f:41:77:2d:51:
        d0:ed:67:0c
-----BEGIN CERTIFICATE REQUEST-----
MIICWTCCAUECAQAwFDESMBAGA1UEAxMJbG9jYWxob3N0MIIBIjANBgkqhkiG9w0B
AQEFAAOCAQ8AMIIBCgKCAQEArSxW45tbRdiHEUWcXButeVzH7MKTGHVFC4ENYqZq
t+GaCIn5MWwxTOcixK1SMGcKdsSmHgsWB3DClGKudDu29+lKZBHtAtBcFQnoxvNH
Pb4bewoVn/5qqI9LBOZxC5vKI/zKgJ07Why2w2/FyuFwJt2hqJzLQGt0IkC8vjBn
uBx4kg6lrPtyq4ZtiCSEU8Jk2Sb/1uIQ7d7b+3nHcdVr01UubbB+aHMROA1aGqpT
irGs1nfCS8zkcWgUXuq0+MEczxVmy2rVe0oqruxu8J3TptqFThQCh4fTOdSxOOCG
aH+mBgW5m9pMErLqSez6ueGEXKLJiy8GkbkuITpsTmNH6QIDAQABoAAwDQYJKoZI
hvcNAQEFBQADggEBAJOZ3sVr1GmMqNl8S9AJa1qbhmAnXcuzeX76ZhC0H3zNTvUF
mgoTlnM0cbLX/+omkeBLDXnHkWgbUH3MWD5jd2Wi23NH1BTSnBLYDyc5g4H3/o/D
v8vCT6nImrqaA41rSlPNL0Xlq6cyt/Maf00H4nF2P4sA57Th6uo2BJi5f44Ljg4Z
oIWtcteiLJytrBHfqAPIaNBLhwn6w7yfruEADG0jLhZgmbf65TXHREOEw3ew+6+T
8kPW1GpxcHbEZrEbtfBvndr1hLlQlVQbDzplrMxOJ8+NE+pFTHA3A7OFt74HAnVy
3AWROBaUP1CDoqhDDiEkmdn8qJ8PQXctUdDtZww=
-----END CERTIFICATE REQUEST-----
```

3. Подпись запроса сертификата и получение (не корневого) сертификата

openssl x509 -req -in <CSRFile> -out <CRTFile> -CAkey <KeyFile> -CA <CAFile> -CAcreateserial -text -sha1 -days <D>

Параметры:

-req – подписать запрос сертификата, полученный на входе и выдать подписанный сертификат на выходе;

-in <CSRFile> - имя файла с подписываемым запросом;

-out <CRTFile> - имя файла с созданным сертификатом;

-CAkey <KeyFile> - имя файла с ключом подписывающего сертификата;

-CAcreateserial – если отсутствует файл с номером сертификата, создать;

-text – включить текстовый комментарий в файл с сертификатом;
-sha1 – использовать алгоритм хэширования SHA1;
-days <D> - установить срок действия сертификата.

Пример:
openssl x509 -req -in localhost2.csr -out localhost2.crt -CAkey localhost1.key -CA localhost1.crt -CAcreateserial -text -sha1 -days 3650

localhost2.crt

```
Certificate:
    Data:
        Version: 1 (0x0)
        Serial Number: 4 (0x4)
        Signature Algorithm: sha1WithRSAEncryption
        Issuer: CN=Localhost CA
        Validity
            Not Before: Nov  8 20:20:49 2005 GMT
            Not After : Nov  6 20:20:49 2015 GMT
        Subject: CN=localhost
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (2048 bit)
                Modulus (2048 bit):
                    00:ad:2c:56:e3:9b:5b:45:d8:87:11:45:9c:5c:1b:
                    ad:79:5c:c7:ec:c2:93:18:75:45:0b:81:0d:62:a6:
                    6a:b7:e1:9a:08:89:f9:31:6c:31:4c:e7:22:c4:ad:
                    52:30:67:0a:76:c4:a6:1e:0b:16:07:70:c2:94:62:
                    ae:74:3b:b6:f7:e9:4a:64:11:ed:02:d0:5c:15:09:
                    e8:c6:f3:47:3d:be:1b:7b:0a:15:9f:fe:6a:a8:8f:
                    4b:04:e6:71:0b:9b:ca:23:fc:ca:80:9d:3b:5a:1c:
                    b6:c3:6f:c5:ca:e1:70:26:dd:a1:a8:9c:cb:40:6b:
                    74:22:40:bc:be:30:67:b8:1c:78:92:0e:a5:ac:fb:
                    72:ab:86:6d:88:24:84:53:c2:64:d9:26:ff:d6:e2:
                    10:ed:de:db:fb:79:c7:71:d5:6b:d3:55:2e:6d:b0:
                    7e:68:73:11:38:0d:5a:1a:aa:53:8a:b1:ac:d6:77:
                    c2:4b:cc:e4:71:68:14:5e:ea:b4:f8:c1:1c:cf:15:
                    66:cb:6a:d5:7b:4a:2a:ae:ec:6e:f0:9d:d3:a6:da:
                    85:4e:14:02:87:87:d3:39:d4:b1:38:e0:86:68:7f:
                    a6:06:05:b9:9b:da:4c:12:b2:ea:49:ec:fa:b9:e1:
                    84:5c:a2:c9:8b:2f:06:91:b9:2e:21:3a:6c:4e:63:
                    47:e9
                Exponent: 65537 (0x10001)
    Signature Algorithm: sha1WithRSAEncryption
        28:fa:68:f0:03:2f:b9:36:b5:79:0e:ee:bd:5b:c8:53:8f:5d:
        87:16:ff:41:30:8a:6b:37:d4:8e:e4:3a:36:3f:15:af:a3:1e:
        3d:1d:7a:e8:b8:7b:75:ba:b1:f7:e2:8a:95:93:18:fd:1f:0f:
        9e:89:2a:13:fe:a1:14:de:1a:3b:49:49:f4:24:0c:d7:f0:20:
        f2:54:bd:b7:18:c2:36:c8:a6:19:8d:e9:c8:dd:a5:53:c2:a6:
        a5:c9:44:f2:28:a3:53:82:6d:91:5b:31:28:34:7c:1c:1d:4e:
        e6:de:27:6f:fe:98:b4:6b:4f:55:e6:55:f4:fa:4b:16:fa:7b:
        a6:bb:1a:a9:94:81:dd:44:e5:c7:0d:30:cf:52:fe:97:25:35:
```

```
        d3:c5:05:b3:3d:df:bf:d1:90:c4:e3:60:e9:f8:2d:b3:b2:30:
        c5:05:39:ee:3d:e9:5f:17:01:11:53:a2:b3:7b:d7:11:65:4e:
        53:b9:3b:50:31:17:3f:97:63:a9:bf:52:5a:2b:3a:4d:4d:e8:
        05:06:df:9d:3e:c9:64:1c:8b:81:5b:53:8f:34:35:87:2a:9c:
        97:15:cc:12:45:b3:52:bf:8d:b5:96:00:7a:1c:ac:5c:45:f4:
        b0:0a:50:0c:07:72:69:87:4d:ce:f8:23:14:5a:86:9d:25:bf:
        b1:4e:be:b4
-----BEGIN CERTIFICATE-----
MIICnzCCAYcCAQQwDQYJKoZIhvcNAQEFBQAwFzEVMBMGA1UEAxMMTG9jYWxob3N0
IENBMB4XDTA1MTEwODIwMjA0OVoXDTE1MTEwNjIwMjA0OVowFDESMBAGA1UEAxMJ
bG9jYWxob3N0MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEArSxW45tb
RdiHEUWcXButeVzH7MKTGHVFC4ENYqZqt+GaCIn5MWwxTOcixK1SMGcKdsSmHgsW
B3DClGKudDu29+lKZBHtAtBcFQnoxvNHPb4bewoVn/5qqI9LBOZxC5vKI/zKgJ07
Why2w2/FyuFwJt2hqJzLQGt0IkC8vjBnuBx4kg6lrPtyq4ZtiCSEU8Jk2Sb/1uIQ
7d7b+3nHcdVr01UubbB+aHMROA1aGqpTirGs1nfCS8zkcWgUXuq0+MEczxVmy2rV
e0oqruxu8J3TptqFThQCh4fTOdSxOOCGaH+mBgW5m9pMErLqSez6ueGEXKLJiy8G
kbkuITpsTmNH6QIDAQABMA0GCSqGSIb3DQEBBQUAA4IBAQAo+mjwAy+5NrV5Du69
W8hTj12HFv9BMIprN9SO5Do2PxWvox49HXrouHt1urH34oqVkxj9Hw+eiSoT/qEU
3ho7SUn0JAzX8CDyVL23GMI2yKYZjenI3aVTwqalyUTyKKNTgm2RWzEoNHwcHU7m
3idv/pi0a09V5lX0+ksW+numuxqplIHdROXHDTDPUv6XJTXTxQWzPd+/0ZDE42Dp
+C2zsjDFBTnuPelfFwERU6Kze9cRZU5TuTtQMRc/l2Opv1JaKzpNTegFBt+dPslk
HIuBW1OPNDWHKpyXFcwSRbNSv421lgB6HKxcRfSwClAMB3Jph03O+CMUWoadJb+x
Tr60
-----END CERTIFICATE-----
```

4. Объединение сертификата с закрытым ключом в цифровые удостоверения PKCS#12

openssl pkcs12 -inkey <KeyFile> -in <CRTFile> -export -out <PFXFile> -nodes

Параметры:

-inkey <KeyFile> - имя файла с закрытым ключом;

-in <CRTFile> - имя файла с сертификатом;

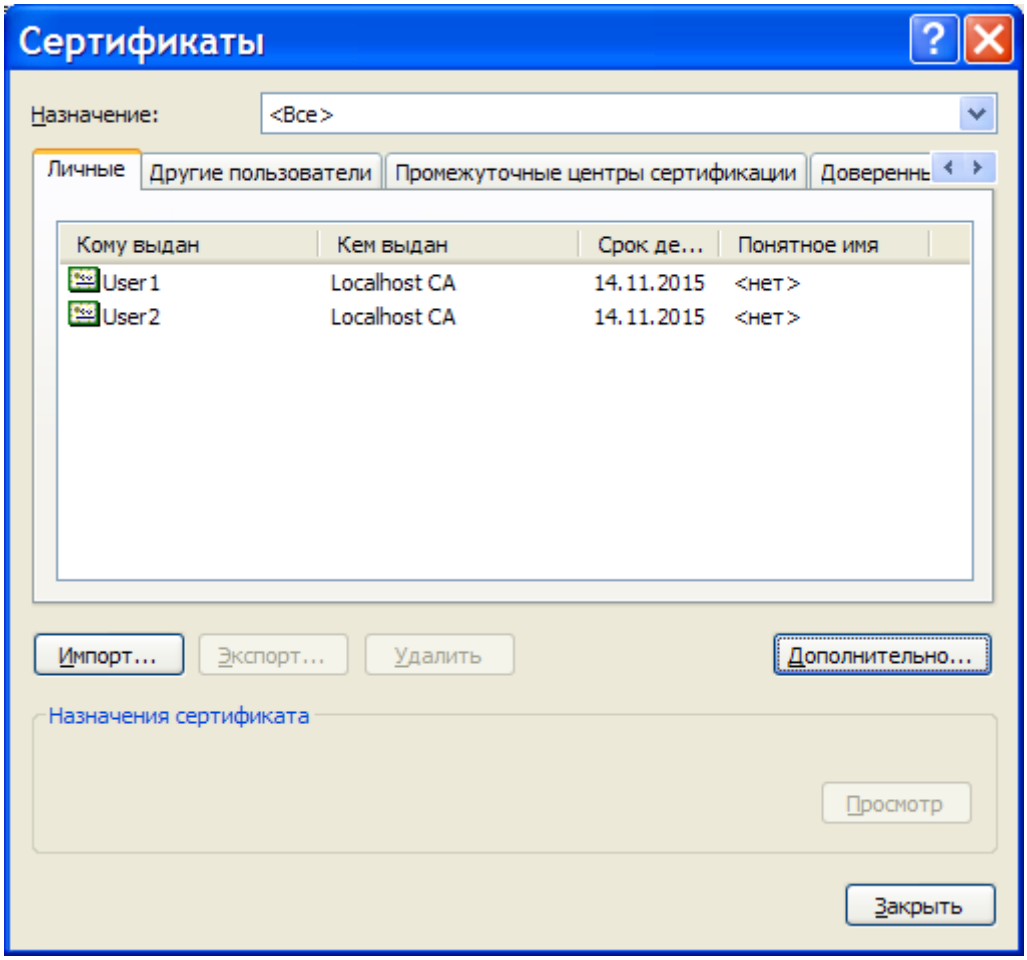
-export – объединить сертификат и закрытый ключ в цифровое удостоверение;

-out <PFXFile> - имя файла с создаваемым удостоверением.

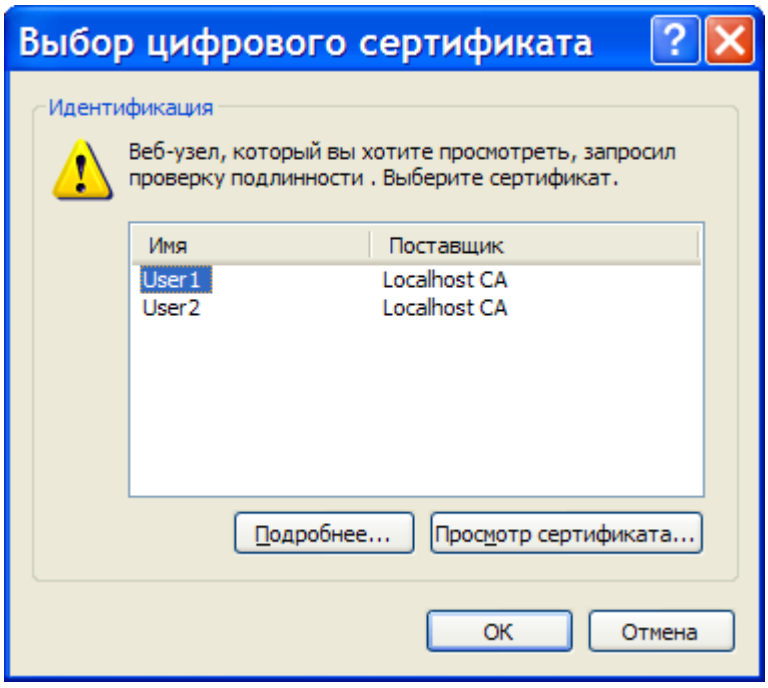
Пример:

openssl pkcs12 -inkey localhost2.key -in localhost2.crt -export -out localhost2.pfx -nodes

Допустим, что существуют пользовательские удостоверения user1.pfx и user2.pfx, принадлежащие соответственно пользователям User1 и User2. Если оба удостоверения установлены в Internet Explorer:



то при загрузке странице с сервера сервер запрашивает пользовательский сертификат, а Internet Explorer переадресует запрос сертификата пользователю:



При помощи пользовательских сертификатов, как и при помощи паролей, можно осуществлять аутентификацию пользователей при доступе к страницам.

Порядок выполнения работы

В данной и последующих двух лабораторных работах используется виртуальная машина. Виртуальная машина представляет собой образ физического диска с установленной на ней операционной системой. Для запуска такой операционной системы используется специальное программное обеспечение – менеджер (монитор, гипервизор) виртуальных машин. Примерами простейших реализаций менеджера виртуальных машин являются VMWare Player и Oracle VirtualBox. В распоряжении студентов имеется установленная программа VMWare Player и образ виртуальной машины в папке wxpro_netprog. Для запуска виртуальной операционной системы нужно открыть в программе VMWare Player файл wxpro_netprog/wxpro_vstu.vmx. В виртуальной операционной системе, во-первых, установлено все необходимое программное обеспечение, во-вторых, доступна учетная запись с правами администратора, в-третьих, имеется вся необходимая документация.

1. Создайте корневой сертификат и его закрытый ключ.
2. Создайте закрытый ключ сервера и запрос сертификата сервера.
3. Получите сертификат сервера, подписав запрос сертификата сервера закрытым ключом корневого сертификата.
4. Скопируйте сертификат сервера в папку и перезапустите web-сервер Apache
5. При помощи браузера Internet Explorer обратиться к ресурсам https://localhost/use, https://localhost/ask, https://localhost/require
6. Создайте закрытый ключ клиента и запрос сертификата клиента.
7. Получите сертификат клиента, подписав запрос сертификата клиента закрытым ключом корневого сертификата
8. Установите сертификат клиента в хранилище Windows / IE.
9. Повторите п. 5.
10. Соединитесь программой openssl с командой s_client с веб-сервером, используя ключ клиента. Выполните http-запрос содержимого ресурса https://localhost/text/readme.txt
11. Запустите программой openssl с командой s_server сервер на порту 40043 , используя ключ сервера
12. Запросите в браузере Internet Explorer ресурс https://localhost:40043/ossl, изучите запрос в консоли сервера и сформируйте правильный http-ответ. Результат наблюдайте в браузере

Содержание отчета

Отчет по лабораторной работе подготавливается ***в электронном виде*** и должен содержать файлы:

1. закрытого ключа корневого сертификата
2. корневого сертификата

3.

Контрольные вопросы

1. Основные команды программы OpenSSL

Лабораторная работа № 5

Программирование SSL-клиента

Цель работы: Изучение программного интерфейса OpenSSL.Изучение порядка действий, выполняемых клиентом SSL.

Основные сведения

Для программирования с использованием OpenSSL нужно в исходную программу на C++ добавить:

```
#define OPENSSL_NO_KRB5
#include <openssl/ssl.h>
#include <openssl/rand.h>
#include <openssl/x509v3.h>
#include <openssl/pkcs12.h>
```

Библиотека openssl поддерживает большое количество алгоритмов шифрования с открытым и закрытым ключом, а также алгоритмов хэширования. Для их использования нужно построить системную таблицу алгоритмов вызовом функции:

```
void OpenSSL_add_all_algorithms();
```

Для уничтожения данных таблиц после завершения работы нужно вызвать:

```
void EVP_cleanup();
```

Для внутренних структур данных openssl поддерживает следующие типы:

EVP_PKEY – закрытый ключ;
EVP_MD – алгоритм хэширования;
EVP_CIPHER – алгоритм шифрования с закрытым ключом;
RSA – набор ключей и параметров алгоритма RSA;
X509 – сертификат X.509;
X509_REQ – запрос сертификата;
X509_NAME – идентификационные данные в сертификате;
BIO – поток данных (файл, сокет, буфер в памяти) для ввода-вывода;
PKCS12 – цифровое удостоверение pkcs12.

Основные задачи сертификационного центра состоят в следующем:
1. Создание запроса сертификата
2. Подпись запроса сертификата – создание сертификата

3. Объединение сертификата с ключом – создание цифрового удостоверения

Запрос сертификата – не подписанный удостоверяющем центром сертификат. Выше показано содержимое файла localhost2.csr, который является примером запроса сертификата. Основные атрибуты запроса сертификата – идентификационные данные (distinguish name) владельца (subject), то есть кому выдается сертификат, версия, информация об используемых алгоритмах шифрования с открытым ключом и хэширования, открытый ключ и подпись сертификата. Открытый ключ формируется в паре с закрытым, который используется для создания цифровой подписи сертификата. В дальнейшем будем рассматривать только алгоритм шифрования с 2048-битным открытым ключом RSA и алгоритм хэширования SHA1.

Создание и уничтожение закрытого ключа выполняется функциями:

```
EVP_PKEY* EVP_PKEY_new();
void EVP_PKEY_free(EVP_PKEY*);
```

Сгенерировать случайную пару ключей RSA и связать их с созданным закрытым ключом можно функциями:

```
RSA* RSA_generate_key(int num, unsigned long e,
 void (*callback)(int, int, void*), void* cb_arg
);
int EVP_PKEY_assign_RSA(EVP_PKEY*, RSA*);
```

Параметры:

num – длина закрытого ключа; будем использовтьа 2048;

e – экспонента – специфический параметр алгоритма RSA; рекомендуемые значения – 3, 17 и 65537; будем использовать значение 65537 = 0x10001;

callback и cb_arg – поскольку процедура вычисления параметров может быть достаточно долгой, RSA_generate_key() периодически вызывает функцию callback с аргументом cb_arg, в которой можно отображать прогресс; в простейшем случае можно использовать 0 для обоих аргументов;

Первая функция возвращает структуру с параметрами RSA, вторая 0 в случае ошибки и 1 в случае успеха.

Для создания и уничтожения запроса сертификата используются функции:

```
X509_REQ* X509_REQ_new();
void X509_REQ_free(X509_REQ*);
```

Для создания и уничтожения сертификата:

```
X509* X509_new();
void X509_free(X509*);
```

Чтобы получить имя владельца запроса сертификата или сертификата вызываются функции:

```
X509_NAME* X509_REQ_get_subject_name(X509_REQ*);
X509_NAME* X509_get_subject_name(X509*);
```

Имена вновь созданных сертификатов являются пустыми. Каждое имя состоит из набора полей. Для того, чтобы отличать поля им присваиваются коды. Некоторые из полей являются стандартными. К стандартным относятся поля с кодами commonName – основное имя, countryName – страна, emailAddress – адрес электронной почты владельца и ряд других. Стандартные поля идентифицируются не только строковыми, но и числовыми кодами – NID. Для получения числового кода по строковому используется функция:

```
int OBJ_txt2nid(const char*);
```

Функция возвращает константу NID_undef, если имя не имеет числового идентификатора.

Для добавления поля в вызывается функция:

```
X509_NAME_add_entry_by_NID(X509_NAME* name, int nid,
 int type, unsigned char* bytes, int len, int loc, int set
);
```

Параметры:

name – имя, в которое устанавливается поле;

nid – числовой идентификатор поля;

type – формат значения; для строкового формата используется константа MBSTRING_ASC;

bytse – значение поля;

len – длина bytes; может быть −1 для строк;

loc и set определяют позицию поля в имени; значения, соответственно, −1 и 0 позволяют выбрать позицию автоматически.

Установка версии запроса сертификата:

```
int X509_REQ_set_version(X509_REQ*, long ver);
```

Для добавления открытого ключа в запрос сертификата и сертификат используются функции:

```
int X509_REQ_set_pubkey(X509_REQ*, EVP_PKEY*);
int X509_set_pubkey(X509*, EVP_PKEY*);
```

Идентификатор алгоритма хэширования SHA1 может быть получен:

EVP_MD const* EVP_sha1();

Для подписи запроса сертификата и сертификата используются функции:

int X509_REQ_sign(X509_REQ*, EVP_PKEY*, EVP_MD const*);
int X509_sign(X509*, EVP_PKEY*, EVP_MD const*);

Порядок выполнения работы

Разработать одну из программ согласно полученному у преподавателя варианту:

1. Программа – клиент, моделирующая работу отделения банка. Сервер должен предоставлять клиенту возможность открытия и закрытия счета, перечисления и снятия денег со счета, перевода некоторых сумм на другие счета и выполнения других подобных операций.
2. Программа – клиент, моделирующая работу железнодорожной кассы. Сервер должен предоставлять клиенту возможность получения справки о поездах, следующих до указанной станции, расписании их движения, наличия свободных мест, заказа билетов различных классов и выполнения других подобных операций.
3. Программа – клиент, позволяющая получить информацию о товарах и услугах некоторых фирм, их стоимости, сформировать заказ и выполнять другие подобные операции.
4. Программа – клиент, позволяющая получить информацию о наличии книг в библиотеке по разным критериям, добавлять новую книгу, удалять потерянную и выполнять другие подобные операции.
5. Программа – клиент, позволяющая получить список файлов указанного каталога на компьютере, где функционирует сервер, переименовать, скопировать в другой каталог или удалить указанный файл, запустить программу на выполнение, выполнить перезагрузку компьютера.

Содержание отчета

Отчетом по лабораторной работе является листинг разработанной программы, представляемый ***в электронном виде.***

Контрольные вопросы

1. *Основные функции программного интерфейса OpenSSL, используемые клиентом*
2. *Основные действия, выполняемые SSL-клиентом*

Литература

1. Золотов С. Протоколы Internet – СПб.: - BHV – Санкт-Петербург, 1998.- 304 с.
2. Семенов Ю.А. Протоколы и ресурсы Internet. - М.:Радио и связь, 1996. - 320 с.
3. Джамса К., Коуп К. Программирование для Internet в среде Windows. : Пер. с англ. - СПб., Питер, 1996. - 688 с.
4. Робачевский А.М. Операционная система UNIX. – СПб.: BHV – Санкт-Петербург, 1997. – 528 с.

Составители: Сергей Олегович Деревенсков, Стрельников Олег Иванович

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ КОМПЬЮТЕРНЫХ СЕТЕЙ
Методические указания к лабораторным работам.

Редактор Е.И. Кагальницкая

Темплан 2011 г. , поз. N ____

Подписано в печать _____ . Формат 60 x 84 1/16. Бумага газетная. Печать офсетная. Усл. печ. л. 1,86 . Уч. -изд.л. 1,81 . Тираж 200 экз. Заказ _____ .

Волгоградский государственный технический университет.
400131 Волгоград , пр. Ленина , 28.
РПК "Политехник" Волгоградского государственного технического университета.
400131 Волгоград , ул. Советская , 35.