Лабораторная работа 1

РАСПРЕДЕЛЕННАЯ СИСТЕМА СБОРА ИНФОРМАЦИИ

Цель работы:

Получить навыки организации параллельной обработки большого количества клиентских соединений при помощи механизма портов завершения Win32. Получить навыки реализации шифрования передаваемых данных при помощи средств CryptoAPI. Получить навыки извлечения информации о системе.

Задача:

Разработать распределенную систему сбора информации о компьютере, состоящую из сервера и клиента, взаимодействующих через сокеты.

Разработать архитектуру системы.

Существует компьютерная сеть. Есть центральный компьютер, на который должна собираться информацию обо всех остальных компьютерах в сети. Сбор информации должен осуществляться в автоматическом режиме. Для этого на все компьютеры внедряется агент, который представляет собой серверную часть системы. На центральном компьютере запускается клиентская часть для запроса информации. Необходимо выбрать, кто инициирует передачу информации: клиент или сервер; кто постоянно работает (готов принять запрос): клиент или сервер; сервер stateless или statefull. Обосновать предлагаемую архитектуру.

Разработать прикладной протокол для запроса и передачи по сети следующей информации о системе:

- Тип и версия ОС
- Текущее время
- Время, прошедшее с момента запуска ОС
- Информация об используемой памяти
- Типы подключенных дисков (локальный / сетевой / съемный, файловая система)
- Свободное место на локальных дисках
- Права доступа в текстовом виде к указанному файлу/папке/ключу реестра
- Владелец файла/папки/ключа реестра

Требования:

- Использовать сокеты (posix или WinSock, но не обертки из библиотек MFC или аналогичных);
- Для передачи каждого типа информации должен существовать свой отдельный запрос;

Формат ответов должен быть формализован и пригоден для машинной обработки, а не только для визуального восприятия человеком

Разработать программу-сервер, которая, будучи запущенной, способна отвечать на запросы клиента по разработанному протоколу. Требования:

- Windows 7/8/10 все SP;
- Консольное приложение без интерактивного взаимодействия с пользователем;
- На консоль выводится диагностическая информация (подключение/отключение клиентов, принимаемые и обрабатываемые запросы);
- Параллельная схема обработки запросов (Использовать порты завершения Win32).

Разработать программу-клиента. Требования к пользовательскому интерфейсу:

- Задание адреса сервера;
- Задание типа запроса;
- Инициация запроса;
- Вывод присланной сервером информации;
- Формат вывода прав доступа должен включать SID субъекта, имя субъекта, типы установленных ACE, область действия установленных прав, номера установленных битов в маске доступа, названия установленных битов для текущего типа объекта (на русском или английском языке или в виде названий констант по MSDN);
- Формат вывода владельца объекта должен содержать SID и имя владельца;
- Текущее время и время, прошедшее с момента запуска ОС, должны выводиться в секундах, минутах, часах, днях и т.д.

Реализовать шифрование передаваемых данных.

- Все сообщения между клиентом и сервером передавались в зашифрованном виде, с использованием CryptoAPI;
- Данные должны передаваться с использованием одного из алгоритмов симметричного шифрования с сеансовым ключом;
- Для первоначальной передачи сеансового ключа необходимо использовать один из алгоритмов асимметричного шифрования.

Теоретические сведения

Механизм IO Completion ports в ОС Windows

Механизм "портов завершения ввода-вывода" (IO Completion ports) предназначен для создания масштабируемых сетевых приложений, которые могут обрабатывать десятки тысяч одновременных подключений и расходовать при этом минимум системных ресурсов. Суть механизма сводится к тому, что вызывающая программа инициирует какую-либо сетевую операцию, а уведомление о ее завершении получает позже. Во время выполнения операции вызывающая программа свободна и может выполнять любую другую полезную работу. Во время выполнения такой отложенной операции в памяти должны оставаться доступными буферы для приема-передачи на все время выполнения асинхронной операции.

Порт завершения создается с помощью функции *CreateIoCompletionPort*:

```
HANDLE io_port = CreateIoCompletionPort(INVALID_HANDLE_VALUE, NULL, 0, 0);
```

Для связи сокета с портом необходимо использовать ту же функцию, но передавать в нее описатель сокета и описатель существующего порта:

```
SOCKET s
CreateIoCompletionPort((HANDLE)s, io_port, 0, 0)
```

При установлении связи сокета и порта в третьем параметре указывается уникальный ключ, который может быть использован программой в каких-либо целях. Например, это может быть индекс элемента в массиве, хранящем текущие состояния клиентов. Ключ сообщается системой при завершении каждой отложенной операции.

Подобно тому, как реализуется цикл ожидания событий в других рассмотренных механизмах, в данном случае необходимо реализовать бесконечный цикл ожидания завершающихся событий. Такой цикл может исполняться в одном или нескольких потоках, что в полной мере позволяет использовать преимущества многопроцессорных систем. Чтение уведомления о завершенном событии из порта завершения осуществляется функцией *GetQueuedCompletionStatus*. При вызове функции сообщается уникальный ключ, присвоенный при связи сокета и порта и указатель на структуру *OVERLAPPED,* указанную в начале операции.

При разработке сетевых приложений с использованием механизма портов завершения необходимо использовать расширенные функции для работы с сокетами:

WSASocket — для создания сокета;

AcceptEx — для принятия подключения;

WSASend / WSASendTo — для отправки данных;

WSARecv / WSARecvFrom — для приема данных.

Функции отличаются от стандартных наличием дополнительного параметра — указателя на структуру *OVERLAPPED*, хранящей некоторые данные, необходимые для подсистемы ввода-вывода ОС. При вызове функции данные структуры должны быть заполнены нулями. При завершении операции ввода-вывода система сообщает указатель на переданную в начале операции структуру *OVERLAPPED*. Обычно это позволяет определить, какая именно операция была завершена.

В случае, если какие-либо операции были начаты, но сокет был закрыт функцией *closesocket*, уведомление об этих операциях все равно поступит в порт завершения ввода-вывода. При этом буферы с данными должны оставаться доступными до момента завершения операций. Чтобы отменить все начатые действия с сокетом, необходимо вызвать *CancelIo*. Вызов функции отменяет незавершенные операции, но не удаляет

уведомления об уже завершенных операциях из порта. Чтобы однозначно определить момент, когда буферы могут быть освобождены, программа может самостоятельно добавить событие в порт завершения. Эта операция выполняется функцией *PostQueuedCompletionStatus*.

Следующий пример демонстрирует использование перечисленных функций и реализует простой сервер. От подключившегося клиента ожидается строка, завершаемая символом перевода строки ('\n'). Приняв такую строку, сервер отправляет клиенту сообщение с размером полученной строки и закрывает соединение.

```
#include <windows.h>
#include <winsock2.h>
#include <mswsock.h>
#include <stdio.h>

#pragma comment(lib, "ws2_32.lib")
#pragma comment(lib, "mswsock.lib")

#define MAX_CLIENTS (100)
#define WIN32_LEAN_AND_MEAN

struct client_ctx

{
	int socket;
	CHAR buf_recv[512];		// Буфер приема
	CHAR buf_send[512];		// Буфер отправки
	unsigned int sz_recv;		// Принято данных
	unsigned int sz_send_total;	// Данных в буфере отправки
	unsigned int sz_send;		// Данных отправлено
                                                        // Структуры OVERLAPPED для уведомлений о
завершении
	OVERLAPPED overlap_recv;
	OVERLAPPED overlap_send;
	OVERLAPPED overlap_cancel;

	DWORD flags_recv; // Флаги для WSARecv

};


// Прослушивающий сокет и все сокеты подключения хранятся
// в массиве структур (вместе с overlapped и буферами)
struct client_ctx g_ctxs[1 + MAX_CLIENTS];
int g_accepted_socket;
HANDLE g_io_port;

// Функция стартует операцию чтения из сокета

void schedule_read(DWORD idx)

{
	WSABUF buf;
	buf.buf = g_ctxs[idx].buf_recv + g_ctxs[idx].sz_recv;
	buf.len = sizeof(g_ctxs[idx].buf_recv) - g_ctxs[idx].sz_recv;
   memset(&g_ctxs[idx].overlap_recv, 0, sizeof(OVERLAPPED));
   g_ctxs[idx].flags_recv = 0;
   WSARecv(g_ctxs[idx].socket, &buf, 1, NULL, &g_ctxs[idx].flags_recv, &g_ctxs[idx].overlap_recv, NULL);
}
// Функция стартует операцию отправки подготовленных данных в сокет
void schedule_write(DWORD idx)

{
	WSABUF buf; buf.buf = g_ctxs[idx].buf_send + g_ctxs[idx].sz_send;
```

```
	buf.len = g_ctxs[idx].sz_send_total - g_ctxs[idx].sz_send;
	memset(&g_ctxs[idx].overlap_send, 0, sizeof(OVERLAPPED));
	WSASend(g_ctxs[idx].socket, &buf, 1, NULL, 0, &g_ctxs[idx].overlap_send, NULL);
}
// Функция добавляет новое принятое подключение клиента

void add_accepted_connection()

{
	DWORD i; // Поиск места в массиве g_ctxs для вставки нового подключения
	for (i = 0; i < sizeof(g_ctxs) / sizeof(g_ctxs[0]); i++)
	{
		if (g_ctxs[i].socket == 0)
		{
			unsigned int ip = 0;
			struct sockaddr_in* local_addr = 0, *remote_addr = 0;
			int local_addr_sz, remote_addr_sz;
			GetAcceptExSockaddrs(g_ctxs[0].buf_recv, g_ctxs[0].sz_recv, sizeof(struct sockaddr_in) + 16,
sizeof(struct sockaddr_in) + 16, (struct sockaddr **) &local_addr, &local_addr_sz, (struct sockaddr **) &remote_addr,
&remote_addr_sz);
			if (remote_addr) ip = ntohl(remote_addr->sin_addr.s_addr);
			printf(" connection %u created, remote IP: %u.%u.%u.%u\n", i, (ip >> 24) & 0xff, (ip >> 16) & 0xff,
(ip >> 8) & 0xff, (ip) & 0xff);
			g_ctxs[i].socket = g_accepted_socket;
			// Связь сокета с портом IOCP, в качестве key используется индекс массива
			if (NULL == CreateIoCompletionPort((HANDLE)g_ctxs[i].socket, g_io_port, i, 0))
			{
				printf("CreateIoCompletionPort error: %x\n", GetLastError());
				return;
			}
			// Ожидание данных от сокета
			schedule_read(i);
			return;
		}
	}
	// Место не найдено => нет ресурсов для принятия соединения
	closesocket(g_accepted_socket);
	g_accepted_socket = 0;
}
// Функция стартует операцию приема соединения

void schedule_accept()
{
	// Создание сокета для принятия подключения (AcceptEx не создает сокетов)
	g_accepted_socket = WSASocket(AF_INET, SOCK_STREAM, 0, NULL, 0, WSA_FLAG_OVERLAPPED);
	memset(&g_ctxs[0].overlap_recv, 0, sizeof(OVERLAPPED));
	// Принятие подключения.
	// Как только операция будет завершена - порт завершения пришлет уведомление. // Размеры буферов должны
быть на 16 байт больше размера адреса согласно документации разработчика ОС
	AcceptEx(g_ctxs[0].socket, g_accepted_socket, g_ctxs[0].buf_recv, 0, sizeof(struct sockaddr_in) + 16, sizeof(struct
sockaddr_in) + 16, NULL, &g_ctxs[0].overlap_recv);
}

int is_string_received(DWORD idx, int* len)
{
	DWORD i;
	for (i = 0; i < g_ctxs[idx].sz_recv; i++)
	{
		if (g_ctxs[idx].buf_recv[i] == '\n')
		{
			*len = (int)(i + 1);
			return 1;
		}
	}
	if (g_ctxs[idx].sz_recv == sizeof(g_ctxs[idx].buf_recv))
	{
```

```
                *len = sizeof(g_ctxs[idx].buf_recv);
                return 1;
        }
        return 0;
}
void io_serv()
{
        WSADATA wsa_data;

        if (WSAStartup(MAKEWORD(2, 2), &wsa_data) == 0)
        {
                printf("WSAStartup ok\n");
        }
        else
        {
                printf("WSAStartup error\n");
        }
        struct sockaddr_in addr;
        // Создание сокета прослушивания
        SOCKET s = WSASocket(AF_INET, SOCK_STREAM, 0, NULL, 0, WSA_FLAG_OVERLAPPED);
        // Создание порта завершения
        g_io_port = CreateIoCompletionPort(INVALID_HANDLE_VALUE, NULL, 0, 0);
        if (NULL == g_io_port)
        {
                printf("CreateIoCompletionPort error: %x\n", GetLastError());
                return;
        }
        // Обнуление структуры данных для хранения входящих соединений
        memset(g_ctxs, 0, sizeof(g_ctxs));
        memset(&addr, 0, sizeof(addr)); addr.sin_family = AF_INET; addr.sin_port = htons(9000);
        if (bind(s, (struct sockaddr*) &addr, sizeof(addr)) < 0 || listen(s, 1) < 0) { printf("error bind() or listen()\n"); return; }
        printf("Listening: %hu\n", ntohs(addr.sin_port));
        // Присоединение существующего сокета s к порту io_port.
        // В качестве ключа для прослушивающего сокета используется 0
        if (NULL == CreateIoCompletionPort((HANDLE)s, g_io_port, 0, 0))
        {
                printf("CreateIoCompletionPort error: %x\n", GetLastError());
                return;
        }
        g_ctxs[0].socket = s;
        // Старт операции принятия подключения.
        schedule_accept();
        // Бесконечный цикл принятия событий о завершенных операциях
        while (1)
        {
                DWORD transferred;
                ULONG_PTR key;
                OVERLAPPED* lp_overlap;
                // Ожидание событий в течение 1 секунды
                BOOL b = GetQueuedCompletionStatus(g_io_port, &transferred, &key, &lp_overlap, 1000);
                if (b)
                {
                        // Поступило уведомление о завершении операции
                        if (key == 0) // ключ 0 - для прослушивающего сокета
                        {
                                g_ctxs[0].sz_recv += transferred;
                                // Принятие подключения и начало принятия следующего
                                add_accepted_connection();
                                schedule_accept();
                        }
                        else
                        {
                                // Иначе поступило событие по завершению операции от клиента. // Ключ key - индекс в
массиве g_ctxs
                                if (&g_ctxs[key].overlap_recv == lp_overlap)
                                {
```

```
                                        int len;
                                        // Данные приняты:
                                        if (transferred == 0)
                                        {
                                                // Соединение разорвано
                                                CancelIo((HANDLE)g_ctxs[key].socket);
                                                PostQueuedCompletionStatus(g_io_port, 0, key,
&g_ctxs[key].overlap_cancel);
                                                continue;
                                        }
                                        g_ctxs[key].sz_recv += transferred;
                                        if (is_string_received(key, &len))
                                        {
                                                // Если строка полностью пришла, то сформировать ответ и начать его
отправлять
                                                sprintf(g_ctxs[key].buf_send, "You string length: %d\n", len);
                                                g_ctxs[key].sz_send_total = strlen(g_ctxs[key].buf_send);
                                                g_ctxs[key].sz_send = 0; schedule_write(key);
                                        }
                                        else
                                        {
                                                // Иначе - ждем данные дальше
                                                schedule_read(key);
                                        }
                                }
                                else if (&g_ctxs[key].overlap_send == lp_overlap)
                                {
                                        // Данные отправлены
                                        g_ctxs[key].sz_send += transferred;
                                        if (g_ctxs[key].sz_send < g_ctxs[key].sz_send_total && transferred > 0)
                                        {
                                                // Если данные отправлены не полностью - продолжить отправлять
                                                schedule_write(key);
                                        }
                                        else
                                        {
                                                // Данные отправлены полностью, прервать все коммуникации,
                                                // добавить в порт событие на завершение работы
                                                CancelIo((HANDLE)g_ctxs[key].socket);
                                                PostQueuedCompletionStatus(g_io_port, 0, key,
&g_ctxs[key].overlap_cancel);
                                        }
                                }
                                else if (&g_ctxs[key].overlap_cancel == lp_overlap)
                                {
                                        // Все коммуникации завершены, сокет может быть закрыт
                                        closesocket(g_ctxs[key].socket); memset(&g_ctxs[key], 0, sizeof(g_ctxs[key]));
                                        printf(" connection %u closed\n", key);
                                }
                        }
                }
                else
                {
                        // Ни одной операции не было завершено в течение заданного времени, программа может
                        // выполнить какие-либо другие действия
                        // ...
                }
        }
}
int main()

{
        io_serv();
        return 0;
}
```

Для проверки и отладки сервера в качестве клиентской программы можно использовать telnet, запустив утилиту с подключением к локальному адресу: *telnet 127.0.0.1 9000*

CryptoAPI

CryptoAPI — интерфейс программирования приложений, который обеспечивает разработчиков Windows-приложений стандартным набором функций для работы с криптопровайдером. Входит в состав операционных систем Microsoft. Большинство функций CryptoAPI поддерживается начиная с Windows 2000.

CryptoAPI поддерживает работу с асимметричными и симметричными ключами, то есть позволяет шифровать и расшифровывать данные, а также работать с электронными сертификатами. Набор поддерживаемых криптографических алгоритмов зависит от конкретного криптопровайдера.

Криптопровайдер (далее CSP (*CryptographyServiceProvider*)) — это независимый модуль, позволяющий осуществлять криптографические операции в операционных системах Microsoft, управление которым происходит с помощью функций CryptoAPI. Проще говоря, это посредник между операционной системой, которая может управлять им с помощью стандартных функций CryptoAPI, и исполнителем криптографических операций (это может быть как программа, так и аппаратный комплекс). Криптопровайдером называют независимый модуль, обеспечивающий непосредственную работу с криптографическими алгоритмами. Каждый криптопровайдер должен обеспечивать:

- Реализацию стандартного интерфейса криптопровайдера;
- Работу с ключами шифрования, предназначенными для обеспечения работы алгоритмов, специфичных для данного криптопровайдера;
- Невозможность вмешательства третьих лиц в схему работы алгоритмов.

Криптопровайдеры реализуются в виде динамически загружаемых библиотек (DLL). Таким образом, достаточно трудно повлиять на ход алгоритма, реализованного в криптопровайдере, поскольку компоненты криптосистемы Windows должны иметь цифровую подпись (то есть подписывается и DLL криптопровайдера). У криптопровайдеров должны отсутствовать возможности изменения алгоритма через установку его параметров. Таким образом решается задача обеспечения целостности алгоритмов криптопровайдера. Задача обеспечения целостности ключей шифрования решается с использованием контейнера ключей. Функции работы с криптопровайдерами можно разделить на следующие группы:

- Функции инициализации контекста и получения параметров криптопровайдера;
- Функции генерации ключей и работы с ними;
- Функции шифрования/расшифровывания данных;
- Функции хеширования и получения цифровой подписи данных.

Базовые функции CryptoAPI

CryptAcquireContext - Используется для создания контейнера ключей с определенным CSP.

```
BOOL WINAPI CryptAcquireContext
(
  _Out_  HCRYPTPROV *phProv,
  _In_   LPCTSTR pszContainer,
  _In_   LPCTSTR pszProvider,
```

```
  _In_   DWORD dwProvType,
  _In_   DWORD dwFlags
);
```

phProv – указатель а дескриптор CSP**.**
pszContainer – имя контейнера ключей.
pszProvider – имя CSP.
dwProvType – тип CSP.
dwFlags – флаги.

CryptGenKey **-** Данная функция предназначена для генерации сеансового ключа, а также для генерации пар ключей для обмена и цифровой подписи.

```
BOOL WINAPI CryptGenKey
(
  _In_   HCRYPTPROV hProv,
  _In_   ALG_ID Algid,
  _In_   DWORD dwFlags,
_Out_  HCRYPTKEY *phKey
);
```

hProv– дескриптор CSP.
Algid – идентификатор алгоритма.
dwFlags – флаги.
phKey – указатель на дескриптор ключа.

CryptImportKey **-** Функция предназначена для получения из каналов информации значения ключа.

```
BOOL WINAPI CryptImportKey
(
  _In_   HCRYPTPROV hProv,
  _In_   BYTE *pbData,
  _In_   DWORD dwDataLen,
  _In_   HCRYPTKEY hPubKey,
  _In_   DWORD dwFlags,
_Out_  HCRYPTKEY *phKey
);
```

hProv – дескриптор CSP.
pbData – импортируемый ключ представленный в виде массива байт.
dwDataLen –длина данных в pbData.
hPubKey - дескриптор ключа, который расшифрует ключ содержащийся в pbData.
dwFlags - флаги.
phKey – указатель на дескриптор ключа. Будет указывать на импортированный ключ.

CryptExportKey **-** Функция экспорта ключа для его передачи по каналам информации. Возможны различные варианты передачи ключа, включая передачу публичного ключа, пары ключей, а также передачу секретного или сеансового ключа.

```
BOOL WINAPI CryptExportKey
(
  _In_    HCRYPTKEY hKey,
  _In_    HCRYPTKEY hExpKey,
  _In_    DWORD dwBlobType,
  _In_    DWORD dwFlags,
  _Out_   BYTE *pbData,
 _Inout_  DWORD *pdwDataLen
);
```

hKey – дескриптор экспортируемого ключа.
hExpKey – ключ, с помощью которого будет зашифрован hKey при экспорте.
dwBlobType – тип экспорта.
dwFlags – флаги.
pbData – буфер для экспорта. Будет содержать зашифрованный hKey с помощью hExpKey.
pdwDataLen – длина буфера на вход. На выходе – количество значащих байт.

CryptDecrypt – Основная базовая функция расшифровывания данных.

```
BOOL WINAPI CryptDecrypt
(
  _In_    HCRYPTKEY hKey,
  _In_    HCRYPTHASH hHash,
  _In_    BOOL Final,
  _In_    DWORD dwFlags,
  _Inout_  BYTE *pbData,
 _Inout_  DWORD *pdwDataLen
);
```

hKey – дескриптор ключа, которым будем расшифровывать.
hHash – дескриптор хеш-объекта. Нужен, если мы должны расшифровать и найти хеш одновременно.
Final - TRUE, если это последний блок на расшифровку и FALSE,если нет.
dwFlags– флаги.
pbData – указатель на буфер с шифртекстом.
pdwDataLen – вход – длина шифртекста, выход – длина расшифрованного текста.

***CryptEncrypt* -** Основная базовая функция шифрования данных.

```
BOOL WINAPI CryptEncrypt
(
  _In_    HCRYPTKEY hKey,
  _In_    HCRYPTHASH hHash,
  _In_    BOOL Final,
  _In_    DWORD dwFlags,
  _Inout_  BYTE *pbData,
  _Inout_  DWORD *pdwDataLen,
  _In_    DWORD dwBufLen
);
```

hKey – дескриптор ключа, которым будем шифровать.
hHash – дескриптор хеш-объекта. Нужен, если мы хотим зашифровать и найти хеш одновременно.
Final – TRUE,если это последний блок на расшифровку и FALSE,если нет.
dwFlags – флаги.
pbData – буфер с открытым текстом.
pdwDataLen– вход – длина открытого текста, выход – длина шифртекста.
dwBufLen – длина буфера.

CryptDestroyKey - Функция предназначена для освобождения ранее полученного хэндла ключа. Функцию следует вызывать всегда для предотвращения утечек памяти в приложении.

```
BOOL WINAPI CryptDestroyKey(_In_  HCRYPTKEYhKey);
```

hKey – дескриптор ключа.

CryptReleaseContext - Данная функция предназначена для освобождения контекста криптопровайдера.

```
BOOL WINAPI CryptReleaseContext
(
  _In_  HCRYPTPROVhProv,
_In_  DWORD dwFlags
);
```

hProv – дескриптор провайдера.
dwFlags – флаги.

CryptCreateHash -Данная функция предназначена для создания хеш-объекта.

```
BOOL WINAPI CryptCreateHash
(
  _In_   HCRYPTPROV hProv,
  _In_   ALG_ID Algid,
  _In_   HCRYPTKEY hKey,
  _In_   DWORD dwFlags,
  _Out_  HCRYPTHASH *phHash);
```

hProv – дескриптор CSP.
Algid – идентификатор алгоритма хеширования.
hKey – ключ, если алгоритм хеширования требует ключа (HMAC, MAC)
dwFlags – флаги
phHash – указатель на дескриптор хеш-объекта (в него будет помещен созданный хеш объект)

CryptHashData-Основная функция хеширования данных.

```
BOOLWINAPICryptHashData
(
  _In_  HCRYPTHASH hHash,
  _In_  BYTE *pbData,
  _In_  DWORDdwDataLen,
_In_  DWORD dwFlags
);
```

hHash – дескриптор хеш-объекта
pbData – буффер с данными
dwDataLen – длина данных
dwFlags – флаги

CryptDeriveKey - Функция предназначена для генерации сеансового ключа на основе хеша данных. То есть данная функция генерирует один и тот же сеансовый ключ, если ей передаются одинаковые значения хеша данных. Функция полезна в случае генерации сеансового ключа на основе пароля.

```
BOOL WINAPI CryptDeriveKey
(
  _In_     HCRYPTPROV hProv,
  _In_     ALG_ID Algid,
  _In_     HCRYPTHASH hBaseData,
  _In_     DWORD dwFlags,
  _Inout_  HCRYPTKEY *phKey
);
```

hProv – дескриптор CSP
Algid – идентификатор алгоритма шифрования. (для ключа)
hBaseData – дескриптор хеш-объекта
dwFlags– флаги
phKey – указатель на дескриптор ключа.(в него будет помещен сгенерированный ключ)

Схема шифрования с использованием сеансового ключа:

1) Клиент генерирует асимметричный ключ–пару ключей публичный/приватный
2) Клиент посылает публичный ключ серверу
3) Сервер генерирует сеансовый ключ
4) Сервер получает публичный ключ клиента
5) Сервер шифрует сеансовый ключ публичным ключом клиента и отправляет получившееся зашифрованное сообщение клиенту
6) Клиент получает зашифрованное сообщение и расшифровывает его с помощью своего приватного ключа
7) У клиента и сервера есть сеансовый ключ. Теперь можно использовать симметричное шифрование для защищенного обмена сообщениями

Получение информации о системе

Тип и версия ОС

Самый удобный способ узнать тип и версию ОС – это функция GetVersionEx(). Пример использования:

```
OSVERSIONINFOEXosvi;
       ZeroMemory(&osvi, sizeof(OSVERSIONINFOEX));
       osvi.dwOSVersionInfoSize = sizeof(OSVERSIONINFOEX);
       GetVersionEx((LPOSVERSIONINFOA)&osvi);
```

После этого в структуре osvi можно найти два поля (osvi.dwMajorVersion и osvi.dwMinorVersion), которые характеризуют версию ОС.

Значение этих элементов для всех версий Windows:

Версия ОС	MajorVersion	MinorVersion
Windows 95	4	0
Windows 98	4	10
WindowsMe	4	90
Windows 2000	5	0
Windows XP	5	1
Windows 2003	5	2
Windows Vista	6	0
Windows 7	6	1
Windows 8	6	2
Windows 8.1	6	3

Примечание: ОС Windows версии выше 8.1 не определяются данной функцией.

Текущее время
Используется функция GetSystemTime(). Пример использования:

```
SYSTEMTIME sm;
GetSystemTime(&sm);
```

Тогда в элементах структуры smбудет хранится информация о системном времени.

sm.wDay - день
sm.wMonth - месяц
sm.wYear- год
sm.wHour - час
sm.wMinute - минуты
sm.wSecond - секунды

Время, прошедшее с момента запуска ОС

Используется функция GetTickCount, которая возвращает количество миллисекунд, прошедших с запуска системы. Информацию можно получить например так:

```
Int hour, min, sec, msec = GetTickCount();
hour = msec / (1000 * 60 * 60);
min = msec / (1000 * 60) - hour * 60;
sec = (msec / 1000) - (hour * 60 * 60) - min * 60;
```

Информация об используемой памяти
Используется функция GlobalMemoryStatus. Пример использования:

```
MEMORYSTATUS stat;
GlobalMemoryStatus(&stat);
```

Теперьвструктуреstat хранится информация о используемой памяти.

stat.dwLength- длина структуры в байтах
stat.dwMemoryLoad- загрузка памяти в процентах
stat.dwTotalPhys- максимальное количество физической памяти в байтах
stat.dwAvailPhys- свободное количество физической памяти в байтах
stat.dwTotalPageFile- максимальное количество памяти для программ в байтах
stat.dwAvailPageFile- свободное количество памяти для программ в байтах
stat.dwTotalVirtual- максимальное количество виртуальной памяти в байтах
stat.dwAvailVirtual- свободное количество виртуальной памяти в байтах

Свободное место на локальных дисках
Удобно узнать свободное место на локальных дисках в два шага.

Первый – узнаем какие в целом на компьютере существуют локальные диски

```
char disks[26][3] = { 0 };
DWORD dr = GetLogicalDrives();

for (i = 0; i< 26; i++)
{
	n = ((dr>>i) & 0x00000001);
	if (n == 1)
	{
		disks[count][0] = char(65 + i);
		disks[count][1] = ':';
		count++;
	}
}
```

Размерность массива disks – 26, так как это максимальное количество локальных дисков. Далее в цикле мы пробегаемся по переменной, выданной нам функцией GetLogicalDrives() и, если находим диск, который существует, то записываем его букву с символом двоеточия в массив. Символ двоеточия нужен будет дальше для обращения к каждому диску. Теперь нам нужно узнать, какие из полученных дисков фиксированные. Это можно выяснить с помощью функции GetDriveType():

```
if (GetDriveTypeA(disks[i]) == DRIVE_FIXED)
{/*Выводим информацию о свободном дисковом пространстве*/}
```

Само свободное пространство можно найти функцией GetDiskFreeSpace().

Например:

```
GetDiskFreeSpaceA(disks[i], &s, &b, &f, &c);
freeSpace = (double)f * (double)s * (double)b / 1024.0 / 1024.0 / 1024.0;
```

Теперь в переменной freeSpace будет хранится свободной пространство на диске disks[i].

Права доступа к указанному файлу/папке/ключу реестра
Права доступа к любому объекту надо вытаскивать из ACL-списков.

Для начала надо воспользоваться функцией GetNamedSecurityInfo(). Например:

```
GetNamedSecurityInfo(path,SE_FILE_OBJECT,DACL_SECURITY_INFORMATION,NULL,
NULL,&a,NULL, &pSD)
```

Параметры функции:

Параметр	Пояснение
path	Путь к файлу/папке или сам ключ. Например: «C://1.txt»
SE_FILE_OBJECT	Флаг. Для файла и директории выставляется SE_FILE_OBJECT, для ключа - SE_REGISTRY_KEY.
DACL_SECURITY_INFORMATION	Определяет, какую информацию мы хотим получить.
&a	Структура PACL
&pSD	Структура PSECURITY_DESCRIPTOR

Далее по SID узнаем права на объект.

Пользуемся функцией GetAclInformation(), потом GetAce(). После этого можно узнать SID:

```
ACCESS_ALLOWED_ACE* pACE;
PSID pSID;
pSID = (PSID)(&(pACE->SidStart));
```

Теперь можно воспользоваться функцией LookupAccountSid() с полученными параметрамии, если она успешна, то сравнивать pACE->Mask со всеми константами, представляющими права доступа. Например GENERIC_ALL, GENERIC_READ, GENERIC_WRITE, GENERIC_EXECUTE для файлов и т.д.

Владелец файла/папки/ключа реестра
Владельца объекта тоже надо смотреть в ACL.

Действия аналогичны предыдущему пункту полностью от начала и до функции GetNamedSecurityInfo().

```
GetNamedSecurityInfo(path, SE_FILE_OBJECT, OWNER_SECURITY_INFORMATION,
&pOwnerSid, NULL, NULL, NULL, &pSD
```

GetNamedSecurityInfo() тоже выполняется почти так же, только в качестве третьего параметра используем константу OWNER_SECURITY_INFORMATION, так как теперь нам нужна информация о владельце, а четвертым параметром идет SID владельца файла.

Далее необходимо узнать имя пользователя по полученному SID. Это можно сделать функцией LookupAccountSid(). Например:

```
Char user[50]= "", domain[50] = "";
LookupAccountSid(NULL, pOwnerSid, user, &userLen, domain, &domainLen, &SidName);
```

Параметры функции:

Параметр	Пояснение
pOwnerSid	Структура PSIDpOwnerSid, в которой будет хранится SID владельца файла.
user	В этой переменной будет имя пользователя.
&userLen	Размер.
domain	В этой переменной будет домен пользователя
&domainLen	Размер.
&SidName	СтруктураSID_NAME_USE SidName.

Контрольные вопросы:

1) Какова структура списков контроля доступа в ОС Windows?
2) Что такое наследование прав доступа?
3) Для чего используются well-known SID?
4) Как выглядит схема шифрования с использованием сеансового ключа?
5) В чем преимущества использования сеансового ключа?