1. Министерство образования и науки Российской Федерации
2. Санкт-Петербургский политехнический университет Петра Великого
3. —
4. Институт компьютерных наук и технологий
5. **Кафедра «Информационная безопасность компьютерных систем»**

**ОТЧЕТ**

**по лабораторной работе № 4**

# Распределенная система сбора информации о рабочих станциях в сети

1. по дисциплине «Безопасность современных информационных технологий»
2. Выполнил
3. студент гр. 33508/3 Проценко Е.Г.

\_\_\_\_\_\_\_\_\_\_

Руководитель Иванов Д.В.

\_\_\_\_\_\_\_\_\_\_

1. Санкт-Петербург
2. 2016

1. Содержание

[1 Цель работы 3](#_Toc468381766)

[2 Результаты работы 5](#_Toc468381767)

[3 Вывод 10](#_Toc468381768)

[4 Ответы на контрольные вопросы 11](#_Toc468381769)

[Приложение А 12](#_Toc468381770)

# Цель работы

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

# Результаты работы

Разработка программы началась с создания простого клиента и сервера, где клиент может отправить на сервер сообщение с запросом ответа, сервер в ответ отправляет строку “Hello!!!”.

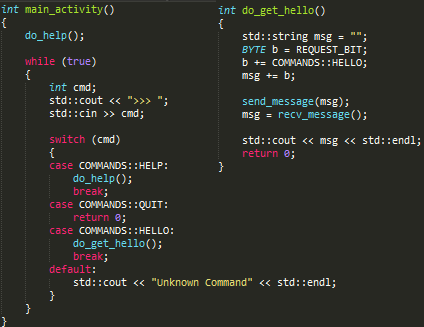


Рисунок 1 – Механизм отправки запроса на сервер

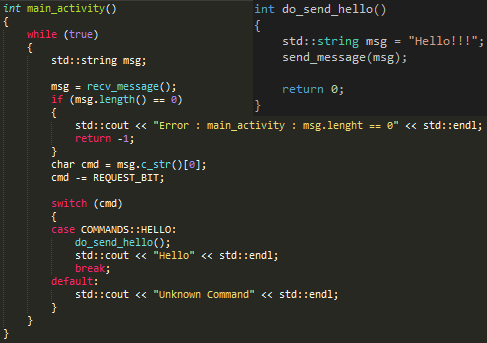


Рисунок 2 – Механизм обработки запроса на сервере

Зачем нужна переменная cmd? Это относится к части протокола взаимодействия между клиентом и сервером. При отправке запроса, клиент должен сформировать данные таким образом, чтобы сервер корректно их обработал. Так как клиент может выполнять несколько различных запросов, нужно выделить некоторое количество битов/байтов для идентификации типа запроса. Для этого используется первый байт сообщение. То есть, каждый запрос (request) от клиента содержит хотя бы один байт данных.

В каждом пакете байт, идентифицирующий тип запроса всегда стоит на первом месте (если передается более одного байта, например, нужно передать путь к файлу, то байт команды будет первым).

Набор команд и их идентификаторов характеризуются следующим фрагментом кода:

enum COMMANDS { VERSION = 1, TICKS, MEMORY, DISKS, OWNER, ACL\_ACE, TIME, HELLO = 95, HELP = 90, QUIT = 99 };

Таблица 1 – Команды и их идентификаторы

|  |  |  |
| --- | --- | --- |
| Команда | Идентификатор | Описание |
| Version | 1 | Запросить версию ос |
| Ticks | 2 | Время с начала работы ос |
| Memory | 3 | Информация о памяти устройства |
| Disks | 4 | Информация о дисках |
| Owner\* | 5 | Владелец файла/папки/ключа реестра |
| Acl\_ace\* | 6 | ACL файла/папки/ключа реестра |
| Time | 7 | Системное время |

Здездочкой помечены те запросы, которые требуют от клиента дополнительной информации: путь к файлу/папке/ключу реестра. Например, запрос на владельца файла может иметь следующий вид:

|  |  |
| --- | --- |
| 5 | C:\OgreSDK\qwe.txt |
| 1 байт | 19 байт (включая ‘\0’) |

Если пользователь прилагает ключ реестра, то путь к нему должен начинаться с ключевых слов: "**CLASSES\_ROOT**", "**CURRENT\_USER**", "**MACHINE**" или "**USERS**". Например: “MACHINE\SOFTWARE\Microsoft\DFS”.

Рассмотрим механизм отправки (send\_message()) и приема сообщений (recv\_message()). На клиенте и сервере этот интерфейс взаимодействия имеет одинаковый вид.



Рисунок 3 – Отправка и прием сообщений

Расширяем интерфейс сервера, рис 4.

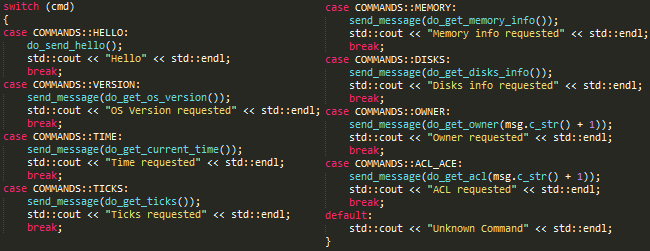


Рисунок 4 – Расширение интерфейса сервера

При запросе каждого типа информации вызывается свой обработчик. Пример нескольких из них приведены в Приложении А.

Теперь нужно расширить круг запросов, которые может отправить клиент, рис 5. Функция main\_acivity() вызывается сразу после подключения клиента к серверу. Функция do\_help() – выводит справку о поддерживаемых командах.



Рисунок 5 – Поддержка команд на клиенте

Функции send\_request() и send\_request\_param() собирают пакет в формат протокола, отправляют его и принимают ответ, рис 6. Главная задача – это поставить на первое место пакета бит команды. Для функции send\_request\_param вызывается дополнительный запрос на указание пути, далее выполняется его обработка и добавление в пакет.

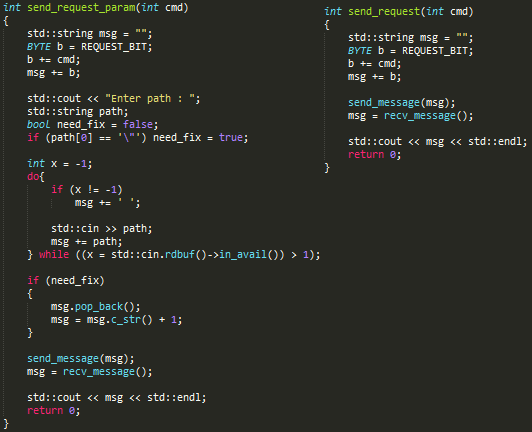


Рисунок 6 – Отправка и обработка запросов

# Вывод

В данной лабораторной работе было реализовано клиент серверное приложение, где клиент может по запросу собирать системную информацию о сервера, такую как время, память, диски и другое.

Сокеты, которые использовались в данной работе являются простым механизмом взаимодействия, но на его основе можно разобраться с другими механизмами взаимодействия.

# Ответы на контрольные вопросы

1. **Какова структура списков контроля доступа в ОС Windows?**

Список контроля доступа содержит список записей контроля доступа (access-control entries, ACEs). ACE бывают разрешающими или запрещающими. Каждая запись содержит набор битовых флагов, сопоставленных определенным правам доступа, и идентификатор SID попечителя (trustee) - пользователя или группы, к которой эти права применены.

1. **Что такое наследование прав доступа?**

Механизм, при котором права доступа субъекта к объектам настраиваются с учетом прав родительского субъекта.

1. **Для чего используются well-known SID?**

Под well-known SID понимаются группы SID, идентифицирующие общих пользователей или общие группы. Их значения остаются постоянными во всех операционных системах. Эта информация полезна при устранении неполадок, связанных с безопасностью. Кроме того, она полезна в случае возможных неполадок отображения, которые можно увидеть в редакторе ACL. В редакторе ACL вместо имени пользователя или группы может отображаться идентификатор SID.

# Приложение А

#pragma warning(disable : 4996)

#include <windows.h>

#include <time.h>

#include "info\_collector.h"

#include <iostream>

#include <Aclapi.h>

#include <Sddl.h>

std::string do\_get\_os\_version()

{

OSVERSIONINFO osvi;

ZeroMemory(&osvi, **sizeof**(OSVERSIONINFO));

osvi.dwOSVersionInfoSize = **sizeof**(OSVERSIONINFO);

GetVersionEx(&osvi);

**int** ver = osvi.dwMajorVersion \* **10** + osvi.dwMinorVersion;

//if ((osvi.dwMajorVersion == 5) && (osvi.dwMinorVersion == 0))

**switch** (ver)

{

**case** **50**:

**return** "Windows 2000";

**case** **51**:

**return** "Windows XP";

**case** **52**:

**return** "Windows XP 64-Bit Edition";

**case** **60**:

**return** "Windows Vista";

**case** **61**:

**return** "Windows 7";

**case** **62**:

**return** "Windows 8 or higher";

**case** **63**:

**return** "Windows 8.1";

**case** **100**:

**return** "Windows 10";

**default:**

**return** "Unknown";

}

}

// Get current date/time, format is YYYY-MM-DD.HH:mm:ss

std::string do\_get\_current\_time() {

**time\_t** now = time(**0**);

**struct** tm tstruct;

**char** buf[**80**];

tstruct = \*localtime(&now);

// Visit http://en.cppreference.com/w/cpp/chrono/c/strftime

// for more information about date/time format

strftime(buf, **sizeof**(buf), "%Y-%m-%d.%X", &tstruct);

**return** buf;

}

std::string do\_get\_ticks()

{

**int** msec = GetTickCount();

**int** sec = ((msec + **500**) / **1000**);

**int** minutes = (**int**)(sec / **60**);

sec %= **60**;

**int** hours = (**int**)(minutes / **60**);

minutes %= **60**;

std::string msg = std::to\_string(hours);

msg += minutes < **10** ? ":0" : ":";

msg += std::to\_string(minutes);

msg += sec < **10** ? ":0" : ":";

msg += std::to\_string(sec);

**return** msg;

**return** std::to\_string(GetTickCount());

}

std::string do\_get\_memory\_info()

{

MEMORYSTATUSEX statex;

statex.dwLength = **sizeof** (statex);

GlobalMemoryStatusEx(&statex);

std::string msg = "";

msg += "Percents of memory in use : ";

msg += std::to\_string(statex.dwMemoryLoad);

msg += "**\n**";

msg += "Total MB of physical memory : ";

msg += std::to\_string(statex.ullTotalPhys / **1024** / **1024**);

msg += "**\n**";

msg += "Free MB of physical memory : ";

msg += std::to\_string(statex.ullAvailPhys / **1024** / **1024**);

msg += "**\n**";

msg += "Total MB of paging file : ";

msg += std::to\_string(statex.ullTotalPageFile >> **20**);

msg += "**\n**";

msg += "Free MB of paging file : ";

msg += std::to\_string(statex.ullAvailPageFile >> **20**);

msg += "**\n**";

msg += "Free MB of virtual file : ";

msg += std::to\_string(statex.ullTotalVirtual >> **20**);

msg += "**\n**";

msg += "Free MB of virtual file : ";

msg += std::to\_string(statex.ullAvailVirtual >> **20**);

**return** msg;

}

std::string do\_get\_disks\_info()

{

std::string msg = "";

**int** n;

**char** dd[**4**];

DWORD dr = GetLogicalDrives();

**for** (**int** i = **0**; i < **26**; i++)

{

n = ((dr >> i) & **0x00000001**);

**if** (n == **1**)

{

dd[**0**] = **char**(**65** + i); dd[**1**] = ':'; dd[**2**] = '\\'; dd[**3**] = **0**;

**long** **long** **int** FreeBytesAvailable = **0**;

**long** **long** **int** TotalNumberOfBytes = **0**;

**long** **long** **int** TotalNumberOfFreeBytes = **0**;

BOOL status = GetDiskFreeSpaceExA(

dd, // directory name

NULL, // bytes available to caller

(PULARGE\_INTEGER)&TotalNumberOfBytes, // bytes on disk

(PULARGE\_INTEGER)&TotalNumberOfFreeBytes // free bytes on disk

);

**if** (status)

{

msg += dd[**0**];

msg += dd[**1**];

msg += " Free/Total : ";

msg += std::to\_string(TotalNumberOfFreeBytes >> **20**);

msg += "/";

msg += std::to\_string(TotalNumberOfBytes >> **20**);

status = GetDriveTypeA(dd);

**switch** (status)

{

**case** **0**:

msg += " (DRIVE\_UNKNOWN)";

**break**;

**case** **1**:

msg += " (DRIVE\_NO\_ROOT\_DIR)";

**break**;

**case** **2**:

msg += " (DRIVE\_REMOVABLE)";

**break**;

**case** **3**:

msg += " (DRIVE\_FIXED)";

**break**;

**case** **4**:

msg += " (DRIVE\_REMOTE)";

**break**;

**case** **5**:

msg += " (DRIVE\_CDROM)";

**break**;

**case** **6**:

msg += " (DRIVE\_RAMDISK)";

**break**;

}

msg += '\n';

}

}

}

msg.pop\_back();

**return** msg;

}