Министерство образования и науки Российской Федерации
Санкт-Петербургский политехнический университет Петра Великого
—
Институт компьютерных наук и технологий
Кафедра «Информационная безопасность компьютерных систем»

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ № 4

РАСПРЕДЕЛЕННАЯ СИСТЕМА СБОРА ИНФОРМАЦИИ О РАБОЧИХ СТАНЦИЯХ В СЕТИ

по дисциплине «Безопасность современных информационных технологий»

Выполнил
студент гр. 33508/3 Проценко Е.Г.
_____

Руководитель Иванов Д.В.
_____

Санкт-Петербург
2016

СОДЕРЖАНИЕ

1 ЦЕЛЬ РАБОТЫ

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

2 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!!!”.

```
int main_activity()
{
    do_help();

    while (true)
    {
        int cmd;
        std::cout << ">>> ";
        std::cin >> cmd;

        switch (cmd)
        {
        case COMMANDS::HELP:
            do_help();
            break;
        case COMMANDS::QUIT:
            return 0;
        case COMMANDS::HELLO:
            do_get_hello();
            break;
        default:
            std::cout << "Unknown Command" << std::endl;
        }
    }
}
```

```
int do_get_hello()
{
    std::string msg = "";
    BYTE b = REQUEST_BIT;
    b += COMMANDS::HELLO;
    msg += b;

    send_message(msg);
    msg = recv_message();

    std::cout << msg << std::endl;
    return 0;
}
```

Рисунок 1 – Механизм отправки запроса на сервер

```
int main_activity()
{
    while (true)
    {
        std::string msg;

        msg = recv_message();
        if (msg.length() == 0)
        {
            std::cout << "Error : main_activity : msg.lenght == 0" << std::endl;
            return -1;
        }
        char cmd = msg.c_str()[0];
        cmd -= REQUEST_BIT;

        switch (cmd)
        {
        case COMMANDS::HELLO:
            do_send_hello();
            std::cout << "Hello" << std::endl;
            break;
        default:
            std::cout << "Unknown Command" << std::endl;
        }
    }
}
```

```
int do_send_hello()
{
    std::string msg = "Hello!!!";
    send_message(msg);

    return 0;
}
```

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.

```
int send_message(std::string msg)
{
    int iResult = send(client_socket, msg.c_str(), (int)strlen(msg.c_str()), 0);
    if (iResult == SOCKET_ERROR)
    {
        PRINT_ERROR("send_message", "send");
        CLEAN_UP();
        return 1;
    }
}

std::string recv_message()
{
#define BUFLEN 256

    std::string msg = "";
    int iResult;
    char buf[BUFLEN];

    do {
        iResult = recv(client_socket, buf, BUFLEN, 0);
        if (iResult > 0)
        {
            buf[iResult] = 0;
            msg += buf;
        }
        else
        {
            PRINT_ERROR("recv_message", "recv");
            return "";
        }
    } while (iResult == BUFLEN);

#undef BUFLEN

    return msg;
}
```

Рисунок 3 – Отправка и прием сообщений

Расширяем интерфейс сервера, рис 4.

```
switch (cmd)
{
case COMMANDS::HELLO:
    do_send_hello();
    std::cout << "Hello" << std::endl;
    break;
case COMMANDS::VERSION:
    send_message(do_get_os_version());
    std::cout << "OS Version requested" << std::endl;
    break;
case COMMANDS::TIME:
    send_message(do_get_current_time());
    std::cout << "Time requested" << std::endl;
    break;
case COMMANDS::TICKS:
    send_message(do_get_ticks());
    std::cout << "Ticks requested" << std::endl;
    break;
case COMMANDS::MEMORY:
    send_message(do_get_memory_info());
    std::cout << "Memory info requested" << std::endl;
    break;
case COMMANDS::DISKS:
    send_message(do_get_disks_info());
    std::cout << "Disks info requested" << std::endl;
    break;
case COMMANDS::OWNER:
    send_message(do_get_owner(msg.c_str() + 1));
    std::cout << "Owner requested" << std::endl;
    break;
case COMMANDS::ACL_ACE:
    send_message(do_get_acl(msg.c_str() + 1));
    std::cout << "ACL requested" << std::endl;
    break;
default:
    std::cout << "Unknown Command" << std::endl;
}
```

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.

```
int main_activity()
{
    do_help();

    while (true)
    {
        int cmd;
        std::cout << ">>> ";
        std::cin >> cmd;

        switch (cmd)
        {
        case COMMANDS::HELP:
            do_help();
            break;
        case COMMANDS::QUIT:
            return 0;
        case COMMANDS::HELLO:
            do_get_hello();
            break;
        case COMMANDS::OWNER:
            send_request_param(cmd);
            break;
        case COMMANDS::ACL_ACE:
            send_request_param(cmd);
            break;
        default:
            if (cmd >= COMMANDS::VERSION && cmd <= COMMANDS::TIME)
                send_request(cmd);
            else
                std::cout << "Unknown Command" << std::endl;
        }
    }
}
```

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.

```cpp
int send_request_param(int cmd)
{
    std::string msg = "";
    BYTE b = REQUEST_BIT;
    b += cmd;
    msg += b;

    std::cout << "Enter path : ";
    std::string path;
    bool need_fix = false;
    if (path[0] == '\"') need_fix = true;

    int x = -1;
    do{
        if (x != -1)
            msg += ' ';

        std::cin >> path;
        msg += path;
    } while ((x = std::cin.rdbuf()->in_avail()) > 1);

    if (need_fix)
    {
        msg.pop_back();
        msg = msg.c_str() + 1;
    }

    send_message(msg);
    msg = recv_message();

    std::cout << msg << std::endl;
    return 0;
}
```

```cpp
int send_request(int cmd)
{
    std::string msg = "";
    BYTE b = REQUEST_BIT;
    b += cmd;
    msg += b;

    send_message(msg);
    msg = recv_message();

    std::cout << msg << std::endl;
    return 0;
}
```

Рисунок 6 – Отправка и обработка запросов

3 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.

4 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.

2) Что такое наследование прав доступа?

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

3) 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;
}
```