1. Министерство образования и науки Российской Федерации
2. Санкт-Петербургский Политехнический Университет Петра Великого
3. —
4. Институт компьютерных наук и технологий
5. Высшая школа программной инженерии

**Курсовая Работа**

«**Сервис резервного копирование данных**»

1. по дисциплине «Конструирование программного обеспечения»
2. Выполнил
3. студент гр. 3530904/80105 Туренко В. Н.

<*подпись*>

1. Проверил
2. Старший преподаватель Иванов А. С.

<*подпись*>

1. Санкт-Петербург
2. 2020

**Постановки проблемы**

Многие из нас встречались с такой проблемой как потеря или искажение важных, или необходимых данных таких как как фото, документы, проекты и т.д. Подобные проблемы могут возникнуть как по неловкости и халатности, так и от программных или аппаратных сбоев. Эта проблема была решена в моем проекте, мною была написана служба, которая осуществляет резервное копирование данных из указанных директорий в заранее прописанные директории с использованием технологий компрессии, прописанные директории в фоновом режиме.

**Выработка требований**

**Анализ предметной области.**

Разрабатываемое мною ПО может использоваться широким спектром людей, так как для многих людей и организаций критически важна сохранность информации в любой непредвиденной ситуации. В первую очередь это ПО отлично подходит для использования на серверном оборудовании

**Сбор требований.**

В первую очередь программа должна быть надежной и поддерживаться разными версиями OC Windows. Так же ПО должно быть интуитивно понятно обычному пользователю.

**Классификация требований.**

* Уделить на этапе разработки и тестирования необходимое количество времени для проведения тестирования, чтобы обеспечить достаточную отказоустойчивость ПО.
* Написать необходимую документацию и гайды для потенциальных пользователей, доступную для понимания широкому кругу лиц
* Написать само ПО которое будет выполнять резервное копирование данных
* Оптимизировать скорость и потребления ресурсов ПО, так как программа постоянно работает в фоновом режиме потребление ресурсов должно быть максимально оптимизированно

**Разрешение противоречий.**

Без сомнения, одному разработчику в такие сроки выполнить все эти условия будет невозможно, процесс оптимизации и тестирования будет значительно урезан, чтобы успеть реализовать основную часть

**Назначение приоритетов.**

Самым главным требованием написания кода в указанный срок, после чего идет отладка и тестирования и в оставшиеся время написания документации, финальные тесты.

**Разработка архитектуры и детальное проектирование.**

**Задача.**

Написать программу, работающую в качестве сервиса (службы) Windows (версии 7-10). Программа должна осуществлять резервное копирование данных из указанной директории.

**Ход работы**

Была написана программа, реализующая сервис Windows, создающий резервные копии файлов в заданной директории. SCP – приложение управления сервисом, использующее функции SCM. Реализуется для того, чтобы можно было напрямую контролировать сервис, избегая прямого взаимодействия пользователя и SCM.

Логика работы с сервисом через реализованный SCP, использующий SCM функции, выглядит следующим образом:

1. Установка сервиса
2. Запуск сервиса
3. Остановка сервиса
4. Удаление сервиса

При установке сервиса в систему выполняются следующие действия Регистрация сервиса в системе, посылка SCM сообщения о будущем нахождении сервиса, создание процессом SCM раздела в реестре для сервиса

В течение работы сервиса пользователь имеет возможность изменять путь к исходной папке/файлу и путь к архиву, хранящему резервную копию исходной папки/файла через файл конфигурации.

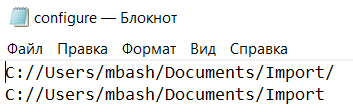
****

Рисунок 1. Пример содержания конфигурационного файла

Отладочные сообщения и сообщения об ошибках выводятся в log файл.

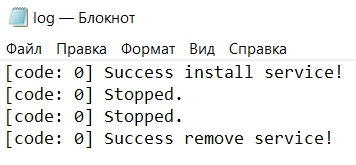


Рисунок 2. Пример содержания log файла

Архив создаётся программой 7zip, взаимодействие с которой производится через командную строку (функцию system ()).

## контекстная диаграмма системы

**Вывод**

В ходе курсовой работы я научился работать со службами windows и, используя win api, написал свою программу, реализующую службу, создающую резервную копию данных. Для данной службы реализовал SCP, имеющее функции создания, запуска, остановки и удаления службы. Ознакомился с основными этапами разработки

**Исходный код**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <windows.h>

#include <tchar.h>

#include <stdio.h>

#include <direct.h>

#include <iostream>

#include <string>

#include <sstream>

#include <fstream>

#include <list>

#include <stack>

#pragma comment (lib, "advapi32.lib")

#pragma comment (lib, "rpcrt4.lib")

#pragma comment(lib, "User32.lib")

using namespace std;

TCHAR serviceName[30] = TEXT("backup\_copy");

SERVICE\_STATUS serviceStatus;

SERVICE\_STATUS\_HANDLE serviceStatusHandle = 0;

HANDLE stopServiceEvent = NULL;

struct STRUCTConfigfile

{

string frompathconfigfile;

string topathconfigfile;

string zipnameconfigfile;

std::list<string> maskfileconfigfile;

};

std::list<STRUCTConfigfile> TASKS;

std::list<string> FilesFromConfigFile;

// LPCTSTR: LP — Long Pointer, C — Constant, T — TCHAR, STR — string

int FileExists(LPCTSTR fname)

{

return (::GetFileAttributes(fname) != DWORD(-1));

}

#define \_SECOND ((long long) 10000000)// change long long to int64\_t

#define \_MINUTE (60 \* \_SECOND)

#define \_HOUR (60 \* \_MINUTE)

#define \_DAY (24 \* \_HOUR)

ULONGLONG ConvertFILETINEToULONGLONG(FILETIME ft) // Convert FILETIME to ULONGLONG

{

ULONGLONG qwResult;

qwResult = (((ULONGLONG)ft.dwHighDateTime) << 32) + ft.dwLowDateTime;

qwResult += 30 \* \_DAY;

return qwResult;

}

ULONGLONG TimeLastModification(const TCHAR \* filename) // Определяем время в FILETIME, затем передает его в вышестоящую функцию

{

HANDLE hFile = NULL;

hFile = CreateFile(filename, GENERIC\_READ, 0, NULL, OPEN\_EXISTING, 0, NULL);

FILETIME lpLastWriteTime;

if (!hFile)

{

return 0;

}

GetFileTime(hFile, NULL, NULL, &lpLastWriteTime);

CloseHandle(hFile);

return ConvertFILETINEToULONGLONG(lpLastWriteTime);

}

void ParserConfigFIle() // Распарсить конфиг файл

{

TCHAR path[\_MAX\_PATH + 1];

GetModuleFileName(0, path, sizeof(path) / sizeof(path[0]));

int i = \_tcslen(path);

while (path[i] != \_T('\\'))

path[i--] = 0;

\_tcscat(path, \_T("BackupConfigFile.txt"));

fstream ConfigFile(path);

string line;

string delimiter = "|";

while (std::getline(ConfigFile, line))

{

STRUCTConfigfile temp;

temp.frompathconfigfile = '\"';

temp.topathconfigfile = '\"';

int pos = 0;

pos = line.find(delimiter);

temp.frompathconfigfile += line.substr(0, pos);

line.erase(0, pos + delimiter.length());

pos = line.find(delimiter);

temp.topathconfigfile += line.substr(0, pos);

line.erase(0, pos + delimiter.length());

pos = line.find(delimiter);

temp.zipnameconfigfile = line.substr(0, pos);

line.erase(0, pos + delimiter.length());

string temp\_mask;

while ((pos = line.find(delimiter)) != std::string::npos)

{

temp\_mask = line.substr(0, pos);

temp.maskfileconfigfile.push\_back(temp\_mask);

line.erase(0, pos + delimiter.length());

}

temp\_mask = line;

temp.maskfileconfigfile.push\_back(temp\_mask);

TASKS.push\_back(temp);

}

ConfigFile.close();

}

void MySystem(char \*szPath)

{

STARTUPINFO si;

memset(&si, 0, sizeof(si));

si.cb = sizeof(si);

PROCESS\_INFORMATION pi;

memset(&pi, 0, sizeof(pi));

bool result = false;

if (CreateProcess(NULL, szPath, NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi))

{

DWORD dwWait = WaitForSingleObject(pi.hProcess, INFINITE);

if (dwWait == WAIT\_OBJECT\_0) result = true;

CloseHandle(pi.hProcess);

CloseHandle(pi.hThread);

}

//return result;

}

void AddFileToZip(const TCHAR \* path, const string command, STRUCTConfigfile MyStruct, TCHAR mode)

{

string temp = MyStruct.frompathconfigfile;

temp.erase(temp.begin());

\_chdir(temp.c\_str());

stringstream buf;

string comm = path;

comm.insert(comm.begin(), \_T('\"'));

comm += \_T("7za\\7za.exe");

comm += +\_T('\"');

comm += \_T(" a -tzip -ssw -mx5 -r0 ");

comm += \_T(" -bso0 -bse0 -bsp0 ");

comm += MyStruct.topathconfigfile;

comm += \_T("\\") + MyStruct.zipnameconfigfile + \_T(".zip") + \_T('\"') + \_T(" ");

if (mode == 'w')

{

comm += MyStruct.frompathconfigfile + \_T("\\");

comm += command + \_T("\"");

}

else

{

string temp1 = command;

size\_t pos = temp1.find(temp + \_T("\\"));

temp1.erase(0, pos + temp.length());

temp1.erase(temp1.begin());

comm += \_T("\"") + temp1 + \_T("\"");

}

MySystem((char \*)comm.c\_str());

}

void GetListFileByMask(const TCHAR \*path\_dir, const string folderName\_file)

{

WIN32\_FIND\_DATA ffd\_file;

WIN32\_FIND\_DATA ffd\_dir;

HANDLE hFind\_file = INVALID\_HANDLE\_VALUE;

HANDLE hFind\_dir = INVALID\_HANDLE\_VALUE;

std::stack<string> st;

string folderName\_dir;

st.push(path\_dir);

while (!st.empty())

{

folderName\_dir = st.top();

st.pop();

hFind\_dir = FindFirstFile((folderName\_dir + \_T("\\") + \_T("\*")).c\_str(), &ffd\_dir);

if (INVALID\_HANDLE\_VALUE != hFind\_dir)

{

do

{

if (ffd\_dir.dwFileAttributes & FILE\_ATTRIBUTE\_DIRECTORY)

{

string check(ffd\_dir.cFileName);

if (check != \_T(".") && check != \_T(".."))

{

string NewPath = folderName\_dir + \_T("\\");

NewPath += ffd\_dir.cFileName;

st.push(NewPath);

}

}

} while (FindNextFile(hFind\_dir, &ffd\_dir) != 0);

}

FindClose(hFind\_dir);

hFind\_file = FindFirstFile((folderName\_dir + \_T("\\") + folderName\_file).c\_str(), &ffd\_file);

if (INVALID\_HANDLE\_VALUE == hFind\_file)

{

FindClose(hFind\_file);

continue;

}

do

{

if (!(ffd\_file.dwFileAttributes & FILE\_ATTRIBUTE\_DIRECTORY))

{

string temp = folderName\_dir + \_T("\\");

temp += ffd\_file.cFileName;

FilesFromConfigFile.push\_back(temp);

}

} while ((FindNextFile(hFind\_file, &ffd\_file) != 0));

FindClose(hFind\_file);

}

}

void MakeBackUpAccordingToConfigFile()

{

TCHAR path[\_MAX\_PATH + 1];

GetModuleFileName(0, path, sizeof(path) / sizeof(path[0]));

int i = \_tcslen(path);

while (path[i] != \_T('\\'))

path[i--] = 0;

for (auto it = TASKS.begin(); it != TASKS.end(); it++)

{

string zippathconfigfile;

zippathconfigfile = it->topathconfigfile + \_T("\\") + it->zipnameconfigfile + \_T(".zip");

zippathconfigfile.erase(zippathconfigfile.begin());

ULONGLONG ZipTimeLastModification;

if (!FileExists(zippathconfigfile.c\_str()))

{

ZipTimeLastModification = 0;

for (auto it\_masks = it->maskfileconfigfile.begin(); it\_masks != it->maskfileconfigfile.end(); it\_masks++)

{

AddFileToZip(path, \*it\_masks, \*it, 'w');

}

}

else

{

ZipTimeLastModification = TimeLastModification(zippathconfigfile.c\_str());

string temp = it->frompathconfigfile;

temp.erase(temp.begin());

string new\_temp;

for (auto it\_masks = it->maskfileconfigfile.begin(); it\_masks != it->maskfileconfigfile.end(); it\_masks++)

{

GetListFileByMask(temp.c\_str(), \*it\_masks);

}

for (auto it\_filesbymasks = FilesFromConfigFile.begin(); it\_filesbymasks != FilesFromConfigFile.end(); it\_filesbymasks++)

{

if (ZipTimeLastModification < TimeLastModification(it\_filesbymasks->c\_str()))

AddFileToZip(path, it\_filesbymasks->c\_str(), \*it, 'a');

}

}

}

}

int other\_main()

{

int i = 0;

while (WaitForSingleObject(stopServiceEvent, 0) != WAIT\_OBJECT\_0)

{

ParserConfigFIle();

MakeBackUpAccordingToConfigFile();

TASKS.clear();

Sleep(3000);

}

return 0;

}

void WINAPI ServiceControlHandler(DWORD controlCode)

{

switch (controlCode)

{

case SERVICE\_CONTROL\_INTERROGATE:

break;

case SERVICE\_CONTROL\_SHUTDOWN:

case SERVICE\_CONTROL\_STOP:

serviceStatus.dwCurrentState = SERVICE\_STOP\_PENDING;

SetServiceStatus(serviceStatusHandle, &serviceStatus);

SetEvent(stopServiceEvent);

return;

case SERVICE\_CONTROL\_PAUSE:

break;

case SERVICE\_CONTROL\_CONTINUE:

break;

default:

if (controlCode >= 128 && controlCode <= 255)

// user defined control code

break;

else

// unrecognised control code-

break;

}

SetServiceStatus(serviceStatusHandle, &serviceStatus);

}

void WINAPI ServiceMain(DWORD argc, TCHAR\* argv[])

{

serviceStatus.dwServiceType = SERVICE\_WIN32;

serviceStatus.dwCurrentState = SERVICE\_STOPPED;

serviceStatus.dwControlsAccepted = 0;

serviceStatus.dwWin32ExitCode = NO\_ERROR;

serviceStatus.dwServiceSpecificExitCode = NO\_ERROR;

serviceStatus.dwCheckPoint = 0;

serviceStatus.dwWaitHint = 0;

serviceStatusHandle = RegisterServiceCtrlHandler(serviceName, ServiceControlHandler);

if (serviceStatusHandle)

{

// service is starting

serviceStatus.dwCurrentState = SERVICE\_START\_PENDING;

SetServiceStatus(serviceStatusHandle, &serviceStatus);

// do initialisation here

stopServiceEvent = CreateEvent(0, FALSE, FALSE, 0);

// running

serviceStatus.dwControlsAccepted |= (SERVICE\_ACCEPT\_STOP | SERVICE\_ACCEPT\_SHUTDOWN);

serviceStatus.dwCurrentState = SERVICE\_RUNNING;

SetServiceStatus(serviceStatusHandle, &serviceStatus);

other\_main();

// service was stopped

serviceStatus.dwCurrentState = SERVICE\_STOP\_PENDING;

SetServiceStatus(serviceStatusHandle, &serviceStatus);

// do cleanup here

CloseHandle(stopServiceEvent);

stopServiceEvent = 0;

// service is now stopped

serviceStatus.dwControlsAccepted &= ~(SERVICE\_ACCEPT\_STOP | SERVICE\_ACCEPT\_SHUTDOWN);

serviceStatus.dwCurrentState = SERVICE\_STOPPED;

SetServiceStatus(serviceStatusHandle, &serviceStatus);

}

}

void Service()

{

SERVICE\_TABLE\_ENTRY serviceTable[] =

{

{ serviceName, ServiceMain },

{ 0, 0 }

};

StartServiceCtrlDispatcher(serviceTable);

}

void RunService()

{

SC\_HANDLE serviceControlManager = OpenSCManager(0, 0, SC\_MANAGER\_CONNECT);

if (serviceControlManager)

{

SC\_HANDLE hService = OpenService(serviceControlManager,

serviceName,

SERVICE\_ALL\_ACCESS);

if (hService)

{

if (!StartService(hService, NULL, NULL))

{

printf("ERROR: Start service\n");

return;

}

else

{

printf("OK: Start service\n");

}

}

}

CloseServiceHandle(serviceControlManager);

}

void StopService(void)

{

SC\_HANDLE serviceControlManager = OpenSCManager(0, 0, SC\_MANAGER\_CONNECT);

if (serviceControlManager)

{

SC\_HANDLE hService = OpenService(serviceControlManager, serviceName, SERVICE\_STOP);

if (hService)

{

SERVICE\_STATUS ss;

if (!ControlService(hService, SERVICE\_CONTROL\_STOP, &ss))

{

printf("ERROR: Stop service\n");

return;

}

else

{

printf("OK: Stop service\n");

}

}

}

CloseServiceHandle(serviceControlManager);

}

void InstallService()

{

SC\_HANDLE serviceControlManager = OpenSCManager(0, 0, SC\_MANAGER\_CREATE\_SERVICE);

if (serviceControlManager)

{

TCHAR path[\_MAX\_PATH + 1];

if (GetModuleFileName(0, path, sizeof(path) / sizeof(path[0])) > 0)

{

SC\_HANDLE service = CreateService(serviceControlManager,

serviceName,

serviceName,

SERVICE\_ALL\_ACCESS,

SERVICE\_WIN32\_OWN\_PROCESS,

SERVICE\_AUTO\_START,

SERVICE\_ERROR\_IGNORE,

path,

0, 0, 0, 0, 0);

if (service)

{

printf("OK: Install service\n");

if (!StartService(service, NULL, NULL)) // 1 +; 0 -

{

printf("ERROR: Start service\n");

}

else

{

printf("OK: Start service\n");

}

CloseServiceHandle(service);

}

else

{

printf("ERROR: Install service\n");

}

}

CloseServiceHandle(serviceControlManager);

}

}

void UninstallService()

{

SC\_HANDLE serviceControlManager = OpenSCManager(0, 0, SC\_MANAGER\_ALL\_ACCESS);

if (serviceControlManager)

{

SC\_HANDLE service = OpenService(serviceControlManager,

serviceName,

SERVICE\_QUERY\_STATUS | DELETE);

if (service)

{

SERVICE\_STATUS serviceStatus;

if (QueryServiceStatus(service, &serviceStatus))

{

if (serviceStatus.dwCurrentState == SERVICE\_STOPPED)

{

if (!DeleteService(service))

{

printf("ERROR: Delete service\n");

return;

}

else

{

printf("OK: Delete service\n");

}

}

}

CloseServiceHandle(service);

}

else

{

printf("ERROR: OpenService\n");

}

CloseServiceHandle(serviceControlManager);

}

}

int main(int argc, TCHAR\* argv[])

{

if (argc > 1 && lstrcmpi(argv[1], TEXT("install")) == 0)

{

InstallService();

}

else if (argc > 1 && lstrcmpi(argv[1], TEXT("uninstall")) == 0)

{

StopService();

UninstallService();

}

else if (argc > 1 && lstrcmpi(argv[1], TEXT("stop")) == 0)

{

StopService();

}

else if (argc > 1 && lstrcmpi(argv[1], TEXT("start")) == 0)

{

RunService();

}

else

{

Service();

}

return 0;

}