**Системное программное обеспечение**

**Лабораторная работа №1**

Съестов Дмитрий Вячеславович

P3217

#include <stdio.h>

#include <Windows.h>

#include <chrono>

#include <cmath>

#include <iostream>

#include <fstream>

#include <string>

using namespace **std**;

using namespace **std::chrono**;

#define **COPYMETHOD\_C** 0

#define **COPYMETHOD\_WINDOWS** 1

#define **COPYMETHOD\_COPYFILE** 2

#define **COPYMETHOD\_BENCHMARK** 3

#define **COPYMETHOD\_INVALID** -1

#define **FILENAME\_SIZE** 16

#define **INITIAL\_FILESIZE\_KB** 1 << 7 *//128 KB*

int **copy\_c**(char\* originalName, char\* copyName)

{

    char buffer[BUFSIZ];

    FILE\* oldFile = **fopen**(originalName, "rb");

if (oldFile == NULL)

    {

**perror**(originalName);

        return EXIT\_FAILURE;

    }

    FILE\* newFile = **fopen**(copyName, "wb");

    if (newFile == NULL)

    {

**perror**(copyName);

        return EXIT\_FAILURE;

    }

    size\_t inBytes, outBytes;

    while ((inBytes = **fread**(buffer, 1, BUFSIZ, oldFile)) > 0)

    {

        outBytes = **fwrite**(buffer, 1, inBytes, newFile);

        if (outBytes != inBytes)

        {

**perror**("Failed to copy file via C functions.");

            return EXIT\_FAILURE;

        }

    }

**fclose**(oldFile);

**fclose**(newFile);

return EXIT\_SUCCESS;

}

int **copy\_windows**(char\* originalName, char\* copyName)

{

    CHAR buffer[BUFSIZ];

HANDLE hIn = **CreateFile**(originalName, GENERIC\_READ, 0, NULL, OPEN\_EXISTING, 0, NULL);

if (hIn == INVALID\_HANDLE\_VALUE)

    {

        cout << "Failed to open input file: error code " << **GetLastError**();

        return EXIT\_FAILURE;

    }

HANDLE hOut = **CreateFile**(copyName, GENERIC\_WRITE, 0, NULL, OPEN\_ALWAYS, 0, NULL);

if (hOut == INVALID\_HANDLE\_VALUE)

{

        cout << "Failed to open output file: error code " << **GetLastError**();

        return EXIT\_FAILURE;

    }

    DWORD nIn, nOut;

while (**ReadFile**(hIn, buffer, BUFSIZ, &nIn, NULL) && nIn > 0)

    {

**WriteFile**(hOut, buffer, nIn, &nOut, NULL);

        if (nIn != nOut)

        {

            cout << "Failed to copy file: error code " << **GetLastError**();

            return EXIT\_FAILURE;

        }

    }

**CloseHandle**(hIn);

**CloseHandle**(hOut);

return EXIT\_SUCCESS;

}

int **copy\_copyfile**(char\* originalName, char\* copyName)

{

int success = **CopyFile**(originalName, copyName, FALSE);

if (!success) cout << "Failed to copy file: error code " << **GetLastError**();

    return success ? EXIT\_SUCCESS : EXIT\_FAILURE;

}

void **calculateAveragePoints**(string filename)

{

    ifstream input(filename, ios\_base::in);

    int count = 0;

    float sum = 0;

int semester, points;

string discipline, teacher;

    while (input >> semester >> discipline >> points >> teacher)

    {

        sum += points;

        count++;

    }

    input.**close**();

float avg = sum / count;

if (!**isnan**(avg)) cout << "Average points: " << avg << endl;

    else cout << "Incorrect file format; unable to parse points." << endl;

}

int **getCopyMethod**(string argument)

{

if (argument == "c") return COPYMETHOD\_C;

else if (argument == "windows") return COPYMETHOD\_WINDOWS;

else if (argument == "copyfile") return COPYMETHOD\_COPYFILE;

else if (argument == "benchmark") return COPYMETHOD\_BENCHMARK;

return COPYMETHOD\_INVALID;

}

time\_t **execTime**(int (\*function)(char\*, char\*), char\* originalName, char\* copyName)

{

auto startTime = **steady\_clock::now**();

int result = **function**(originalName, copyName);

if (result == EXIT\_FAILURE) return -1;

auto duration = duration\_cast<microseconds>(**steady\_clock::now**() - startTime);

return duration.**count**();

}

int **benchmark**()

{

char originalName[FILENAME\_SIZE], copyName[FILENAME\_SIZE];

**tmpnam\_s**(originalName, FILENAME\_SIZE);

**tmpnam\_s**(copyName, FILENAME\_SIZE);

FILE\* original;

cout << "+--------+----------------+----------------+----------------+" << endl << "| SIZE | C | WINDOWS | COPYFILE |" << endl << "+--------+----------------+----------------+----------------+" << endl;

size\_t size\_kb = INITIAL\_FILESIZE\_KB, oldSize\_kb = 0;

time\_t time1, time2, time3;

char buffer[1024];

for (int i = 0; i < 1024; i++) buffer[i] = '1';

for (int i = 0; i < 10; i++)

{

for (int k = 0; k < size\_kb - oldSize\_kb; k++)

{

original = **fopen**(originalName + 1, "ab");

**fwrite**(buffer, 1, 1024, original);

**fclose**(original);

}

size\_t displaySize = size\_kb;

bool megabytes = displaySize >= 1024;

if (megabytes) displaySize = displaySize >> 10;

time1 = **execTime**(&copy\_c, originalName + 1, copyName + 1);

time2 = **execTime**(&copy\_windows, originalName + 1, copyName + 1);

time3 = **execTime**(&copy\_copyfile, originalName + 1, copyName + 1);

**printf**("|%5i %cB|%12i mcs|%12i mcs|%12i mcs|\n", displaySize, megabytes ? 'M' : 'K', time1, time2, time3);

oldSize\_kb = size\_kb;

size\_kb = size\_kb << 1;

}

**remove**(originalName + 1);

**remove**(copyName + 1);

cout << "+--------+----------------+----------------+----------------+" << endl;

return EXIT\_SUCCESS;

}

bool **validateArgs**(int argc, char\* argv[])

{

if (argc != 2 && argc != 4 && argc != 5) return false;

if (argc == 5 && (string)argv[4] != "--avg") return false;

int copyMethod = **getCopyMethod**(argv[1]);

if (copyMethod == COPYMETHOD\_INVALID) return false;

if (argc == 2 == !(copyMethod == COPYMETHOD\_BENCHMARK)) return false;

return true;

}

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

{

bool argsAreValid = **validateArgs**(argc, argv);

if (!argsAreValid)

{

cout << "Usage:" << endl << "lab1 {c|windows|copyfile} <OLD\_FILENAME> <NEW\_FILENAME> [--avg]" << endl << endl;

cout << "For benchmark:" << endl << "lab1 benchmark" << endl;

return EXIT\_FAILURE;

}

int copyExitcode;

switch (**getCopyMethod**(argv[1]))

{

case COPYMETHOD\_C:

copyExitcode = **copy\_c**(argv[2], argv[3]);

if(copyExitcode == EXIT\_FAILURE) return EXIT\_FAILURE;

break;

case COPYMETHOD\_WINDOWS:

copyExitcode = **copy\_windows**(argv[2], argv[3]);

if(copyExitcode == EXIT\_FAILURE) return EXIT\_FAILURE;

break;

case COPYMETHOD\_COPYFILE:

copyExitcode = **copy\_copyfile**(argv[2], argv[3]);

if(copyExitcode == EXIT\_FAILURE) return EXIT\_FAILURE;

break;

case COPYMETHOD\_BENCHMARK:

return **benchmark**();

}

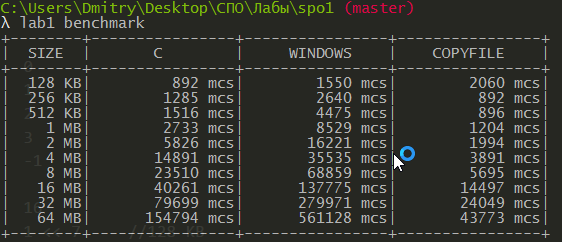
cout << "File was succesfully copied." << endl;

if (argc == 7) **calculateAveragePoints**(argv[2]);

return copyExitcode;

}

**Сравнение способов**



Преимущества каждого способа:

* Копирование через функции С доступно на любой платформе.
* Копирование через функции WinAPI возвращает не только количество обработанных байтов, но и булево значение, соответствующее результату операции.
* Копирование через CopyFile – самый простой способ, т.к. требуются только имена двух файлов. Также он позволяет отменить операцию копирования.