Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики

Факультет Компьютерных Технологий и Управления

Кафедра Информатики и прикладной математики

**Системное программное обеспечение**

Лабораторная работа №1

Выполнил:

Хлопков Д. С.

Группа: Р3217

Преподаватель:

Зыков Анатолий Геннадьевич

Санкт-Петербург

2018 г.

**Листинг программы:**

*lab1.cpp*

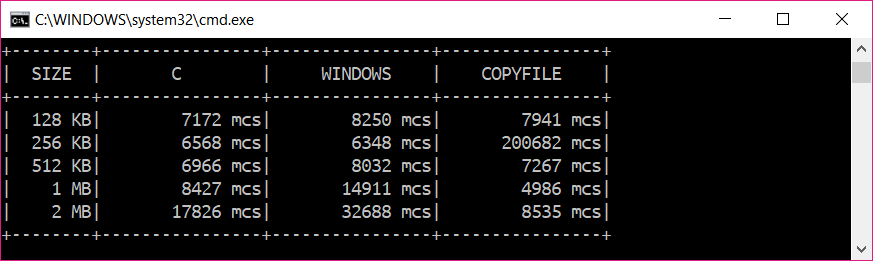
1. #include <stdio.h>
2. #include <Windows.h>
3. #include <chrono>
4. #include <cmath>
5. #include <iostream>
6. #include <fstream>
7. #include <string>
8. #include "cpy.h"
9. using namespace std;
10. using namespace std::chrono;
12. enum methods{
13. COPYMETHOD\_C,
14. COPYMETHOD\_WINDOWS,
15. COPYMETHOD\_COPYFILE,
16. COPYMETHOD\_BENCHMARK,
17. COPYMETHOD\_INVALID = -1
18. };

21. #define FILENAME\_SIZE           16
22. #define INITIAL\_FILESIZE\_KB     1 << 7     //128 KB
24. void calculateAveragePoints(string filename)
25. {
26. ifstream input(filename, ios\_base::in);
28. int count = 0;
29. float sum = 0;
31. int semester, points;
32. string discipline, teacher;
33. while (input >> semester >> discipline >> points >> teacher)
34. {
35. sum += points;
36. count++;
37. }
39. input.close();
41. float avg = sum / count;
42. if (!isnan(avg)) cout << "Average points: " << avg << endl;
43. else cout << "Incorrect file format; unable to parse points." << endl;
44. }
46. int getCopyMethod(string argument)
47. {
48. if (argument == "c") return COPYMETHOD\_C;
49. else if (argument == "windows") return COPYMETHOD\_WINDOWS;
50. else if (argument == "copyfile") return COPYMETHOD\_COPYFILE;
51. else if (argument == "benchmark") return COPYMETHOD\_BENCHMARK;
52. return COPYMETHOD\_INVALID;
53. }
55. time\_t execTime(int (\*function)(char\*, char\*), char\* originalName, char\* copyName)
56. {
57. auto startTime = steady\_clock::now();
58. int result = function(originalName, copyName);
59. if (result == EXIT\_FAILURE) return -1;
61. auto duration = duration\_cast<microseconds>(steady\_clock::now() - startTime);
62. return duration.count();
63. }
65. int benchmark()
66. {
67. char originalName[FILENAME\_SIZE], copyName[FILENAME\_SIZE];
68. tmpnam\_s(originalName, FILENAME\_SIZE);
69. tmpnam\_s(copyName, FILENAME\_SIZE);
71. FILE\* original;
73. cout << "+--------+----------------+----------------+----------------+" << endl <<
74. "|  SIZE  |       C        |     WINDOWS    |    COPYFILE    |" << endl <<
75. "+--------+----------------+----------------+----------------+" << endl;
77. size\_t size\_kb = INITIAL\_FILESIZE\_KB, oldSize\_kb = 0;
78. time\_t time1, time2, time3;
80. char buffer[1024];
81. for (int i = 0; i < 1024; i++) buffer[i] = '1';
83. for (int i = 0; i < 5; i++)
84. {
85. for (int k = 0; k < size\_kb - oldSize\_kb; k++)
86. {
87. original = fopen(originalName + 1, "ab");
88. fwrite(buffer, 1, 1024, original);
89. fclose(original);
90. }
92. size\_t displaySize = size\_kb;
93. bool megabytes = displaySize >= 1024;
94. if (megabytes) displaySize = displaySize >> 10;
95. time1 = execTime(&copy\_c, originalName + 1, copyName + 1);
96. time2 = execTime(&copy\_windows, originalName + 1, copyName + 1);
97. time3 = execTime(&copy\_copyfile, originalName + 1, copyName + 1);
98. printf("|%5i %cB|%12i mcs|%12i mcs|%12i mcs|**\n**", displaySize, megabytes ? 'M' : 'K', time1, time2, time3);
100. oldSize\_kb = size\_kb;
101. size\_kb = size\_kb << 1;
102. }
104. remove(originalName + 1);
105. remove(copyName + 1);
107. cout << "+--------+----------------+----------------+----------------+" << endl;
108. return EXIT\_SUCCESS;
109. }
111. bool validateArgs(int argc, char\* argv[])
112. {
113. if (argc != 2 && argc != 4 && argc != 5) return false;
114. if (argc == 5 && (string)argv[4] != "--avg") return false;
116. int copyMethod = getCopyMethod(argv[1]);
117. if (copyMethod == COPYMETHOD\_INVALID) return false;
118. if (argc == 2 == !(copyMethod == COPYMETHOD\_BENCHMARK)) return false;
120. return true;
121. }
123. int main(int argc, char\* argv[])
124. {
125. bool argsAreValid = validateArgs(argc, argv);
126. if (!argsAreValid)
127. {
128. cout << "Usage:" << endl << "lab1 {c|windows|copyfile} <OLD\_FILENAME> <NEW\_FILENAME> [--avg]" << endl << endl;
129. cout << "For benchmark:" << endl << "lab1 benchmark" << endl;
130. return EXIT\_FAILURE;
131. }
133. int copyExitcode;
134. switch (getCopyMethod(argv[1]))
135. {
136. case COPYMETHOD\_C:
137. copyExitcode = copy\_c(argv[2], argv[3]);
138. if(copyExitcode == EXIT\_FAILURE) return EXIT\_FAILURE;
139. break;
141. case COPYMETHOD\_WINDOWS:
142. copyExitcode = copy\_windows(argv[2], argv[3]);
143. if(copyExitcode == EXIT\_FAILURE) return EXIT\_FAILURE;
144. break;
146. case COPYMETHOD\_COPYFILE:
147. copyExitcode = copy\_copyfile(argv[2], argv[3]);
148. if(copyExitcode == EXIT\_FAILURE) return EXIT\_FAILURE;
149. break;
151. case COPYMETHOD\_BENCHMARK:
152. return benchmark();
153. }
155. cout << "File was succesfully copied." << endl;
156. if (argc == 5) calculateAveragePoints(argv[2]);
158. return copyExitcode;
159. }

*cpy.h*

1. #include <stdio.h>
2. #include <Windows.h>
3. #include <iostream>
4. #include <fstream>
5. #include <string>
6. using namespace std;
8. int copy\_c(char\* originalName, char\* copyName)
9. {
10. char buffer[BUFSIZ];
12. FILE\* oldFile = fopen(originalName, "rb");
13. if (oldFile == NULL)
14. {
15. perror(originalName);
16. return EXIT\_FAILURE;
17. }
19. FILE\* newFile = fopen(copyName, "wb");
20. if (newFile == NULL)
21. {
22. perror(copyName);
23. return EXIT\_FAILURE;
24. }
26. size\_t inBytes, outBytes;
27. while ((inBytes = fread(buffer, 1, BUFSIZ, oldFile)) > 0)
28. {
29. outBytes = fwrite(buffer, 1, inBytes, newFile);
31. if (outBytes != inBytes)
32. {
33. perror("Failed to copy file via C functions.");
34. return EXIT\_FAILURE;
35. }
36. }
38. fclose(oldFile);
39. fclose(newFile);
41. return EXIT\_SUCCESS;
42. }
44. int copy\_windows(char\* originalName, char\* copyName)
45. {
46. CHAR buffer[BUFSIZ];
48. HANDLE hIn = CreateFile(originalName, GENERIC\_READ, 0, NULL, OPEN\_EXISTING, 0, NULL);
49. if (hIn == INVALID\_HANDLE\_VALUE)
50. {
51. cout << "Failed to open input file: error code " << GetLastError();
52. return EXIT\_FAILURE;
53. }
55. HANDLE hOut = CreateFile(copyName, GENERIC\_WRITE, 0, NULL, OPEN\_ALWAYS, 0, NULL);
56. if (hOut == INVALID\_HANDLE\_VALUE)
57. {
58. cout << "Failed to open output file: error code " << GetLastError();
59. return EXIT\_FAILURE;
60. }
62. DWORD nIn, nOut;
63. while (ReadFile(hIn, buffer, BUFSIZ, &nIn, NULL) && nIn > 0)
64. {
65. WriteFile(hOut, buffer, nIn, &nOut, NULL);
66. if (nIn != nOut)
67. {
68. cout << "Failed to copy file: error code " << GetLastError();
69. return EXIT\_FAILURE;
70. }
71. }
73. CloseHandle(hIn);
74. CloseHandle(hOut);
76. return EXIT\_SUCCESS;
77. }
79. int copy\_copyfile(char\* originalName, char\* copyName)
80. {
81. int success = CopyFile(originalName, copyName, FALSE);
82. if (!success) cout << "Failed to copy file: error code " << GetLastError();
83. return success ? EXIT\_SUCCESS : EXIT\_FAILURE;
84. }

**Сравнение производительности**



**Вывод**

В ходе выполнения лабораторной работы были изучены 3 способа копирования файлов. При помощи библотеки C, с использованием Windows, с использованием вспомогательной функции Windows. Были выявлены следующие отличия:

* При помощи библиотеки C можно копировать файлы на любой платформе
* Копирование через функции WinAPI возвращает не только количество обработанных байтов, но и булево значение, соответствующее результату операции.
* Копирование при помощи CopyFile самый простой способ, а также исходя из проведенного сравнения производительности время копирования файла меньше всего увеличивается при увеличении размера файла