**山东师范大学**

**实验报告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 院系部所 | 信息科学与工程学院 | | 课程名称 | | 计算机操作系统 |
| 所在班级 | 计工本1702 | 学号 | 201711010202 | 姓名 | 王汝芸 |
| 实验编号 | 10 | 题目 | 实现文件读/写的三种方式比较 | | |

一、实验目的和要求

通过对无缓冲、有缓冲和异步三种方式实现文件读写操作的比较，进一步理解操作系统有关文件系统I/O的概念，为选择文件I/O方式提供依据。

二、实验环境

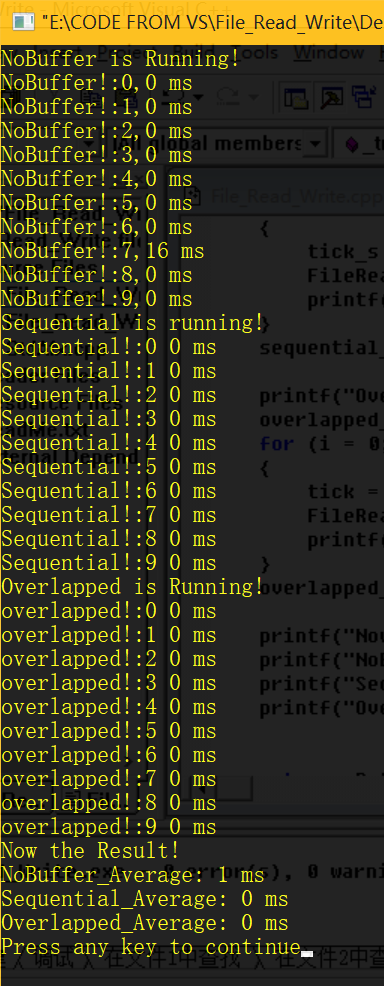
VC++6.0

Windows 10

三、实验内容及实施

使用本章实验一、实验二和实验三建立的三个函数：FileReadWrite\_NoBuffer( )、FileReadWrite\_Sequential\_Scan( )、和FileReadWrite\_Overlapped( )，分别用无缓冲、有缓冲和异步方式实现稳文件的读/写操作。

四、实验结果

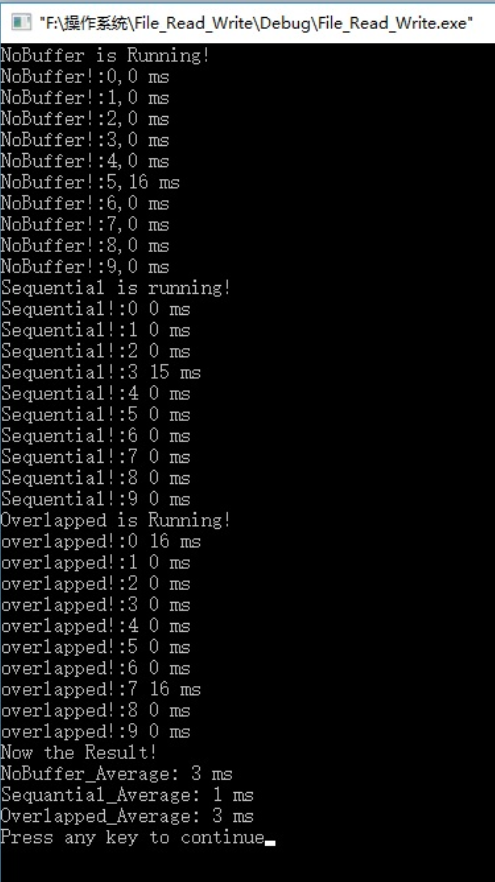


五、实验分析

分别调用三个函数各10次，最后得到的三种方式的平均花费时间如上图所示。

以上数据和实验指导书结论相差较大，分析原因可能为固态硬盘导致。

用室友的机械硬盘又运行一次，得到结果虽然个别时间变长，但差距并不大，结果如下图：



六、实验讨论（可选）





七、附录 （可选）

1 *// File\_Read\_Write.cpp : Defines the entry point for the console application.*

2 *//*

3

4 #include "stdafx.h"

5 #include "File\_Read\_Write.h"

6

7 #ifdef \_DEBUG

8 #define new DEBUG\_NEW

9 #undef THIS\_FILE

10 **static char** THIS\_FILE[] = \_\_FILE\_\_;

11 #endif

12

13 DWORD BufferSize = 2048;

14 **char** buf[2048];

15

16 */////////////////////////////////////////////////////////////////////////////*

17 *// The one and only application object*

18

19 CWinApp theApp;

20

21 **using namespace** std;

22

23 **void** **FileReadWrite\_NoBuffer**(**char**\*source, **char**\*destination);

24 **void** **FileReadWrite\_Sequential\_Scan**(**char**\* source, **char**\*destination);

25 **void** **FileReadWrite\_Overlapped**(**char**\* source, **char**\*destination);

26

27 **int** **\_tmain**(**int** argc, TCHAR\* argv[], TCHAR\* envp[])

28 {

29 **int** i, nRetCode = 0;

30

31 DWORD tick, tick\_s;

32 DWORD nobuffer\_start\_time, sequential\_start\_time, overlapped\_start\_time;

33 DWORD nobuffer\_end\_time, sequential\_end\_time, overlapped\_end\_time;

34 DWORD nobuffer\_average\_time = 0;

35 DWORD sequential\_average\_time = 0;

36 DWORD overlapped\_average\_time = 0;

37

38 **printf**("NoBuffer is Running!\n");

39 nobuffer\_start\_time = **GetTickCount**();

40

41 **for** (i = 0; i < 10; i++)

42 {

43 tick = **GetTickCount**();

44 **FileReadWrite\_NoBuffer**("source.txt", "nobuffer.txt");

45 **printf**("NoBuffer!:%d,%d ms\n", i, **GetTickCount**() - tick);

46 }

47 nobuffer\_end\_time = **GetTickCount**();

48

49 **printf**("Sequential is running!\n");

50 sequential\_start\_time = **GetTickCount**();

51 **for** (i = 0; i < 10; i++)

52 {

53 tick\_s = **GetTickCount**();

54 **FileReadWrite\_Sequential\_Scan**("source.txt", "Sequential.txt");

55 **printf**("Sequential!:%d %d ms\n", i, **GetTickCount**() - tick\_s);

56 }

57 sequential\_end\_time = **GetTickCount**();

58

59 **printf**("Overlapped is Running!\n");

60 overlapped\_start\_time = **GetTickCount**();

61 **for** (i = 0; i < 10; i++)

62 {

63 tick = **GetTickCount**();

64 **FileReadWrite\_Overlapped**("source.txt", "overlapped.txt");

65 **printf**("overlapped!:%d %d ms\n", i, **GetTickCount**() - tick);

66 }

67 overlapped\_end\_time = **GetTickCount**();

68

69 **printf**("Now the Result!\n");

70 **printf**("NoBuffer\_Average: %d ms\n", (nobuffer\_end\_time - nobuffer\_start\_time) / 10);

71 **printf**("Sequential\_Average: %d ms\n", (sequential\_end\_time - sequential\_start\_time) / 10);

72 **printf**("Overlapped\_Average: %d ms\n", (overlapped\_end\_time - overlapped\_start\_time) / 10);

73

74

75 **return** nRetCode;

76 }

77

78 **void** **FileReadWrite\_NoBuffer**(**char**\*source, **char**\*destination)

79 {

80 HANDLE handle\_src, handle\_dst;

81 DWORD NumberOfByteRead, NumberOfByteWrite;

82 BOOL cycle;

83 **char**\*buffer;

84 buffer = buf;

85

86 handle\_src = **CreateFile**(source,

87 GENERIC\_READ,

88 0,

89 NULL,

90 OPEN\_EXISTING,

91 FILE\_FLAG\_NO\_BUFFERING,

92 NULL

93 );

94 handle\_dst = **CreateFile**(destination,

95 GENERIC\_WRITE,

96 NULL,

97 NULL,

98 CREATE\_ALWAYS,

99 NULL,

100 NULL);

101 **if** (handle\_src == INVALID\_HANDLE\_VALUE || handle\_dst == INVALID\_HANDLE\_VALUE)

102 {

103 **printf**("File Create Fail!\n");

104 **exit**(1);

105 }

106 cycle = TRUE;

107

108 **while** (cycle)

109 {

110 NumberOfByteRead = BufferSize;

111 **if** (!**ReadFile**(handle\_src, buffer, NumberOfByteRead, &NumberOfByteRead, NULL))

112 {

113 **printf**("Read File Error!%d\n", **GetLastError**());

114 **exit**(1);

115 }

116 **if** (NumberOfByteRead < BufferSize)cycle = FALSE;

117 **if** (!**WriteFile**(handle\_dst, buffer, NumberOfByteRead, &NumberOfByteWrite, NULL))

118 {

119 **printf**("Write File Error!%d\n", **GetLastError**());

120 **exit**(1);

121 }

122 }

123 **CloseHandle**(handle\_src);

124 **CloseHandle**(handle\_dst);

125

126 }

127

128 **void** **FileReadWrite\_Sequential\_Scan**(**char**\*source, **char**\*destination)

129 {

130 HANDLE handle\_src, handle\_dst;

131 DWORD NumberOfByteRead, NumberOfByteWrite;

132 BOOL cycle;

133 **char**\*buffer;

134 buffer = buf;

135

136 handle\_src = **CreateFile**(source,

137 GENERIC\_READ,

138 0,

139 NULL,

140 OPEN\_EXISTING,

141 FILE\_FLAG\_SEQUENTIAL\_SCAN,

142 NULL);

143 handle\_dst = **CreateFile**(destination,

144 GENERIC\_WRITE,

145 NULL,

146 NULL,

147 CREATE\_ALWAYS,

148 FILE\_FLAG\_SEQUENTIAL\_SCAN,

149 NULL);

150 **if** (handle\_src == INVALID\_HANDLE\_VALUE || handle\_dst == INVALID\_HANDLE\_VALUE)

151 {

152 **printf**("File Create Fail!\n");

153 **exit**(1);

154 }

155 cycle = TRUE;

156

157 **while** (cycle)

158 {

159 NumberOfByteRead = BufferSize;

160 **if** (!**ReadFile**(handle\_src, buffer, NumberOfByteRead, &NumberOfByteRead, NULL))

161 {

162 **printf**("Read File Error!%d\n", **GetLastError**());

163 **exit**(1);

164 }

165 **if** (NumberOfByteRead < BufferSize)cycle = FALSE;

166 **if** (!**WriteFile**(handle\_dst, buffer, NumberOfByteRead, &NumberOfByteWrite, NULL))

167 {

168 **printf**("Write File Error!%d\n", **GetLastError**());

169 **exit**(1);

170 }

171 }

172 **CloseHandle**(handle\_src);

173 **CloseHandle**(handle\_dst);

174 }

175

176 **void** **FileReadWrite\_Overlapped**(**char**\* source, **char**\*destination)

177 {

178 HANDLE handle\_src, handle\_dst;

179 DWORD NumberOfByteRead, NumberOfByteWrite, Error;

180 BOOL cycle;

181 **char**\*buffer;

182 buffer = buf;

183 OVERLAPPED overlapped;

184

185

186 handle\_src = **CreateFile**(source,

187 GENERIC\_READ,

188 0,

189 NULL,

190 OPEN\_EXISTING,FILE\_FLAG\_NO\_BUFFERING|

191 FILE\_FLAG\_OVERLAPPED,

192 NULL);

193 handle\_dst = **CreateFile**(destination,

194 GENERIC\_WRITE,

195 NULL,

196 NULL,

197 CREATE\_ALWAYS,

198 NULL,

199 NULL);

200 **if** (handle\_src == INVALID\_HANDLE\_VALUE || handle\_dst == INVALID\_HANDLE\_VALUE)

201 {

202 **printf**("File Create Fail!\n");

203 **exit**(1);

204 }

205 cycle = TRUE;

206 overlapped.hEvent = NULL;

207 overlapped.Offset = -BufferSize;

208 overlapped.OffsetHigh = 0;

209

210 **while** (cycle)

211 {

212 overlapped.Offset = overlapped.Offset + BufferSize;

213 NumberOfByteRead = BufferSize;

214 **if** (!**ReadFile**(handle\_src,

215 buffer,

216 NumberOfByteRead,

217 &NumberOfByteRead,

218 &overlapped))

219 {

220 **switch** (Error = **GetLastError**())

221 {

222 **case** ERROR\_HANDLE\_EOF:

223 cycle = FALSE;

224 **break**;

225 **case** ERROR\_IO\_PENDING:

226 **if** (!**GetOverlappedResult**(handle\_src,

227 &overlapped,

228 &NumberOfByteRead,

229 TRUE))

230 {

231 **printf**("GetOverlappedResult!%d\n", **GetLastError**());

232 **exit**(1);

233 }

234 **break**;

235 **default**:

236 **break**;

237 }

238 }

239 **if** (NumberOfByteRead < BufferSize)cycle = FALSE;

240 *//WRITE FILE*

241 **if** (!**WriteFile**(handle\_dst,

242 buffer,

243 NumberOfByteRead,

244 &NumberOfByteWrite,

245 NULL

246 ))

247 {

248 **printf**("Write File Error!%d\n", **GetLastError**());

249 **exit**(1);

250 }

251 }

252 **CloseHandle**(handle\_src);

253 **CloseHandle**(handle\_dst);

254

255 }

256

257