CSE451 Winter 2013 Project #3
Out: January 30, 2013
Due: February 20, 2013 by 11:59 PM (late assignments will lose 20% per day)

Objectives

In the first two projects you learned about some of the internals of the Windows Operating System. Your third project, done in groups of three, will practice using threads and synchronization to write a multi-threaded multi-buffered copy file program. The principal objectives of this project are:

1. To practice thinking and writing multi-threaded applications
2. Design and implement a multi-buffering program
3. Working on a group project
4. Look at performance aspects

Getting Started

Unlike the previous projects this project involves writing a single user mode program Windows program in C. You may use Visual Studio to do the development. Located in O:\cse\courses\cse451\13wi\Project3 is the base skeleton code to get you started.

Things you will need to know how to do are:

1. Manipulate files with open, create, read, write, and close.
2. Start, and synchronize threads
3. Time your operation

Your assignment:

You are writing a command line program to copy one or more files to another location. The skeleton code already parses the command line and calls two blank routines that you will need to supply. The syntax for the program is:

```
MTCOPY [/T:number] [/B:size] [/A] [/V] <source>+ destination

  /T:number - Specifies the number of threads and buffers to use
in the copy function.  The default value is 1.

  /B:size - Specifies the maximum buffer size to use in the copy
function.  The default value is 4096.

  /A – Specifies if the I/O is to be issued asynchronously. The
default value is to do synchronous I/O.

  /V – Verbose switch used to report the total time needed to
copy the file(s).  It also prints the MB per second throughput.
```

```
  source - Specifies one or more source files to be copied. The
program handles wildcard provided it is linked with the
wsetargv.obj that is in the project directory.

  destination - Specifies the directory for the new file(s).
```

The program is to open and read each source file, and create and write each destination file. The two routines you need to write are:

```
  MtFileCopy( ThreadCount, BufferSize, SrcDst, Verbose ), and
  MtFileCopyAsync( ThreadCount, BufferSize, SrcDst, Verbose )
```

Where:

```
  ThreadCount – Specifies in the synchronous case the number of
threads used to do the copy.  In the asynchronous case it is the
number of buffers used to do the copy.

  BufferSize – Specifies the buffer size.

  SrcDst – Specifies each source and destination file.  SrcDst is
a pointer to an array of pointers (which is null terminated).
Each pointer in the array points to a pair of pointers.  The
paired pointers point to the source and destination file names.
A diagram might help illustrate this.

  +--------+  +----------+       +----------+
  | SrcDst |->| 1stFile |------>| SrcName | -> "Letter.doc "
  +--------+  | 2ndFile |->...  | DstName | -> "z:bin\letter.doc"
              | 3rdFile |->...  +----------+
              | Null    |
              +----------+

In this example, SrcDst points to an array containing three
pointers. 1stFile points to two more pointers, each pointing to a
string.  The source file is "Letter.doc" and the destination file
is "z:bin\letter.doc".

  Verbose – Specifies the function is to printout throughput
statistics.
```

Your goal is to achieve as much parallelism and efficiency as possible utilizing the specified number of threads and buffer size. In the synchronous case each thread will have exactly one buffer that it uses to read and write data. MtFileCopy must do all I/O synchronously and should be smart enough to not use 10 threads to copy a single two byte file; but it might use all of its threads to copy a single 1GB file or to copy 100 small files. MtFileCopyAsync must use a single thread to do all I/O asynchronously and be event based.

After completing the program your continued assignment is to analyze its performance using various combinations of threads, files, buffer sizes, file sizes, and mixed media. There is a command script in Project3's distribution directory called "mtcopytest.cmd"

that tests various combinations. Feel free to use this command script as your test base, under the usual disclaimer that "your millage may vary." It has been tested on the lab machines. Do not let this sample command script restrict your creativity. Be creative in your testing.

You will need to do a complete concise write-up reporting your results in order to receive full credit. Besides reporting actual results your write-up will need to speculate on why your program exhibits certain behavior. For example, why certain configurations are better performing than others, and what conclusions can be made from this.

N.B. it is easy to test the correctness of your program by comparing the source and destination files.

Also look at the sample function called CSE451Cat to see an example of how open and read file, and how to compute time differences.

Turn-in:

Be prepared to turn in the following on Catalyst:

1. Executables images of your test program
2. Source code for your test program
3. A write up listing the students who worked on the project and your performance analysis

Grading Criteria:

Your project will be graded on the following scale.

10 points Correctness
20 points Multi-threaded case
20 points Asynchronous case
20 points Readability
10 points Elegance
20 points Write-up

Late assignments will lose 20% per day.

Addendum

I suggest that you look at the MSDN website (http://msdn.microsoft.com/en-us/library/default.aspx) for a complete description of the Win32 APIs.

File operations

Three basic file operations are: CreateFile, ReadFile, and WriteFile. You will also need to look through the File Management Functions listed on the MSDN website to see the other functions that will query/set file sizes and enumerate directories.

File operations can be a little tricky because one of the attributes of an opened file is whether you want to do synchronous or asynchronous I/O. Synchronous Read and Write calls do not return until the operation completes; whereas, asynchronous Read and Write calls return immediately, and the user is later signaled via an event when the operation completes. In Windows terminology this is also called Overlapped I/O.

Now, according to the MSDN website there is a serialization restriction if you use multiple threads to issue multiple requests to a file handle opened for synchronous I/O. What this means is that while you may be attempting to read or write from multiple places in a file simultaneously it will be serialized if you use a handle opened for synchronous I/O. The way I would work around this restriction is to use multiple handles (i.e., I would call CreateFile multiple times on the same file with a proper share mode). Then I can issue multiple I/O on a single file and know that they will not be serialized.

Thread operations and synchronizations

The basic operations to create a new thread, and synchronize threads using mutexes are: CreateThread, CreateMutex, ReleaseMutex, WaitForSingleObject, and WaitForMultipleObjects.