Analysis

The order of the graphs in the second, third page goes from lots of files in a single directory (1,2,4,8,16,32,64,128,256,512 files in the single directory), a long subdirectory chains (a folder will contain a .CSV file as well as another directory, and so on), and combinations of the two (long subdirectory chains with a lot of files at the very end of the subdirectory). The second page shows real time, third page shows the user time and the fourth (final) page shows the sys time. All these should be obvious if you were to look at what the line of each graph is called.

Here are the descriptions of real, user, sys
Real – The real time between the invocation and termination
User – the User's CPU time
ex) When a program runs through an array
Sys – The system's CPU time
ex) When a program executes system call such as exec, fork and more.

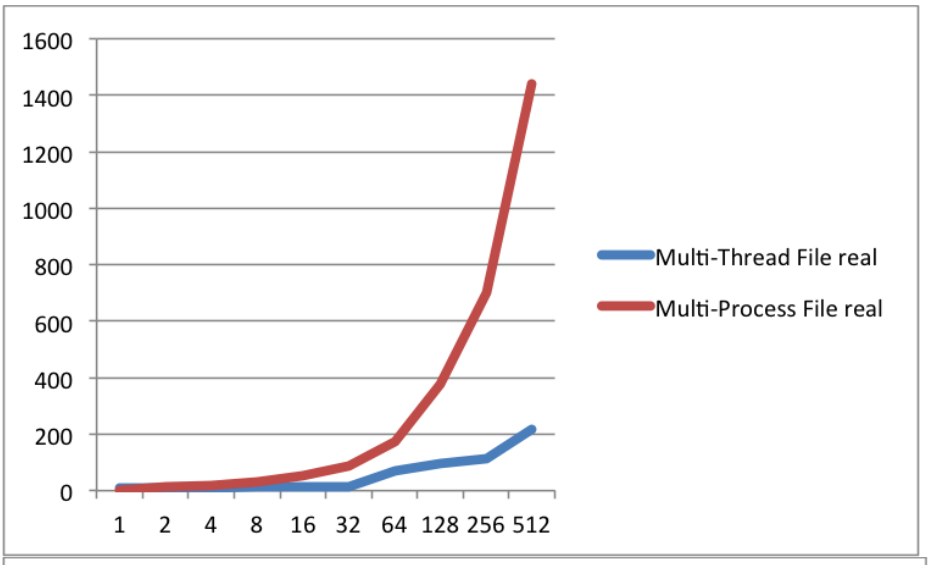
Also, next time time unix command is used, we should be using perf stat instead. Perf stat is a better benchmarking command…

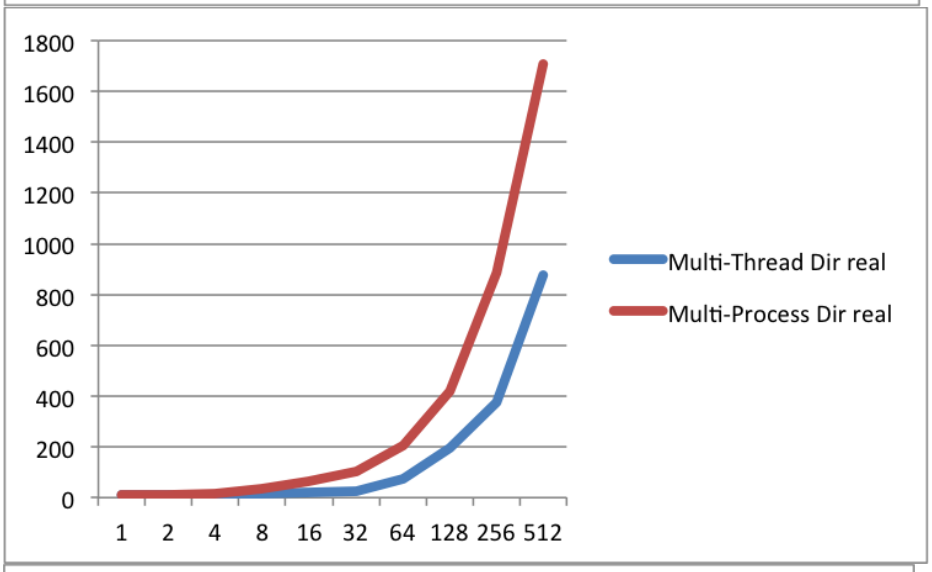
I averaged the time by running the program three times in two different machines (one iLab (factory.cs.rutgers.edu) and the other one a linux server set up in my house) and averaging the time.

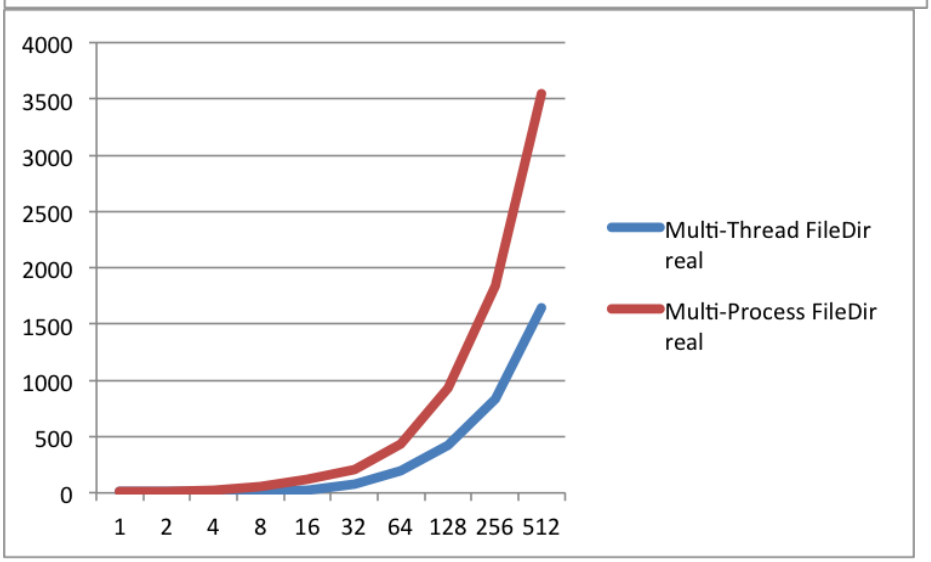
Initially, the growth might seem to grow in an exponential rate due to how the graph line looks like, but take into consideration that the x component of each graph goes by in an 1,2,4,8,16,32,64,128,256,512 way.

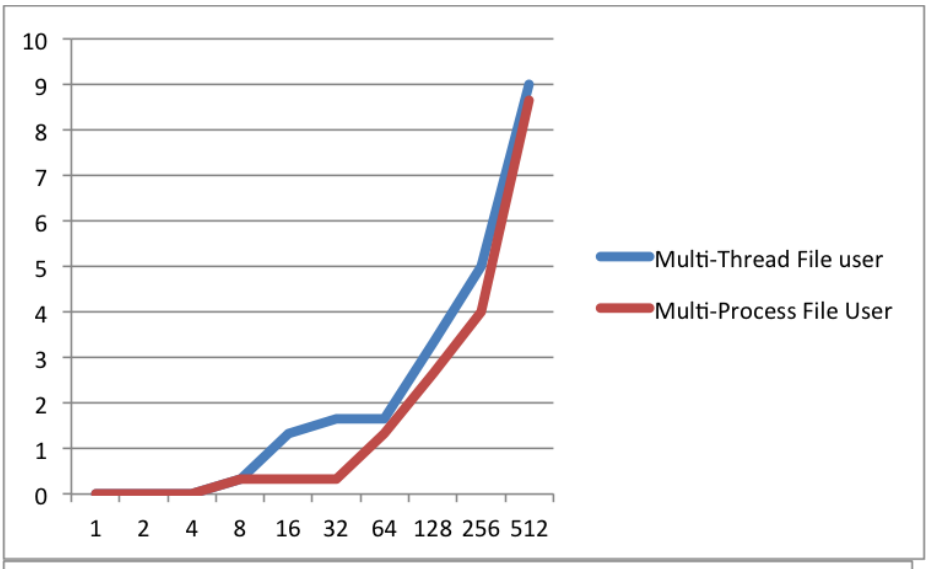
It also appeared that a single directory that contains a bunch of file is a lot faster than a long subdirectory. One of the theories that I thought of is that the code takes longer time is due to the fact that it goes through each directories every time and then sort through each files in that directories, which is the same of that in a single directory. And the fact that the third and final search is the slowest (almost twice as small) is due to it having both the single directory files as well as the subdirectory files.

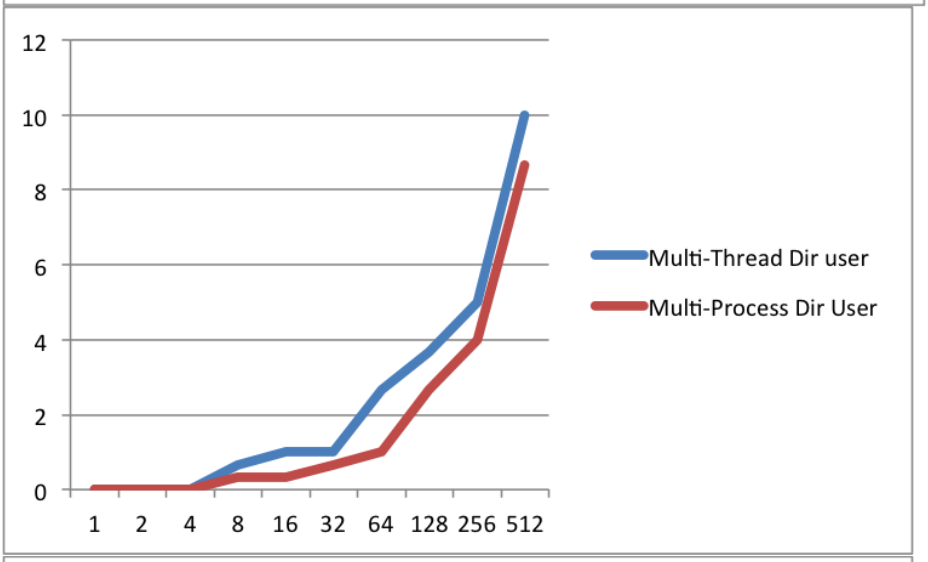
Multi threading is a lot faster in real time. But multi threading is a little slower at user as the program takes longer time running through each struct array created for the Allfiles.csv. The reason why multi threading is faster than multi processing is due to faster task switching as well as less overhead. For multi processing, each process will need to create its own virtual memory space, thus, will take a longer time. Threads are faster than processes because threads are able to share memory easily since they are in the same memory space compared to processes.

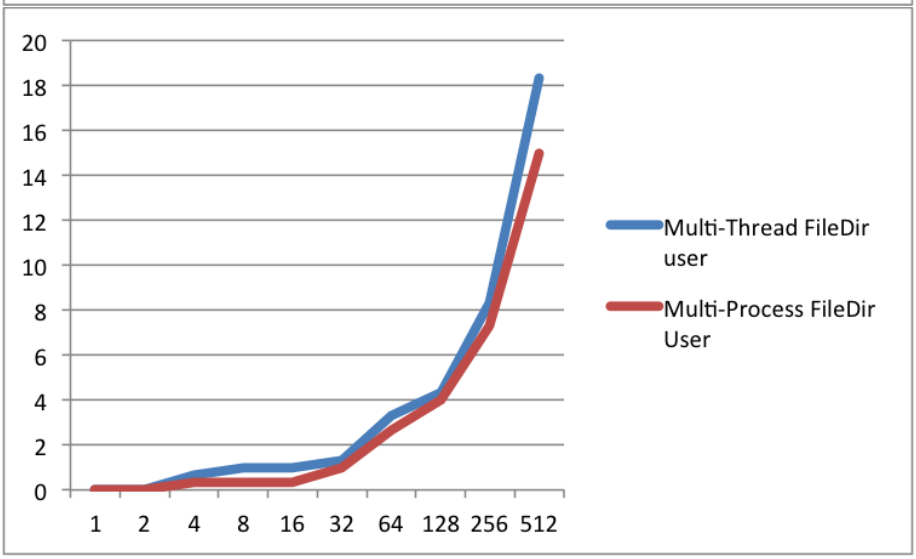

1600
1400
1200
1000
800
600
400
200
0
1 2 4 8 16 32 64 128 256 512
Multi-Thread File real
Multi-Process File real

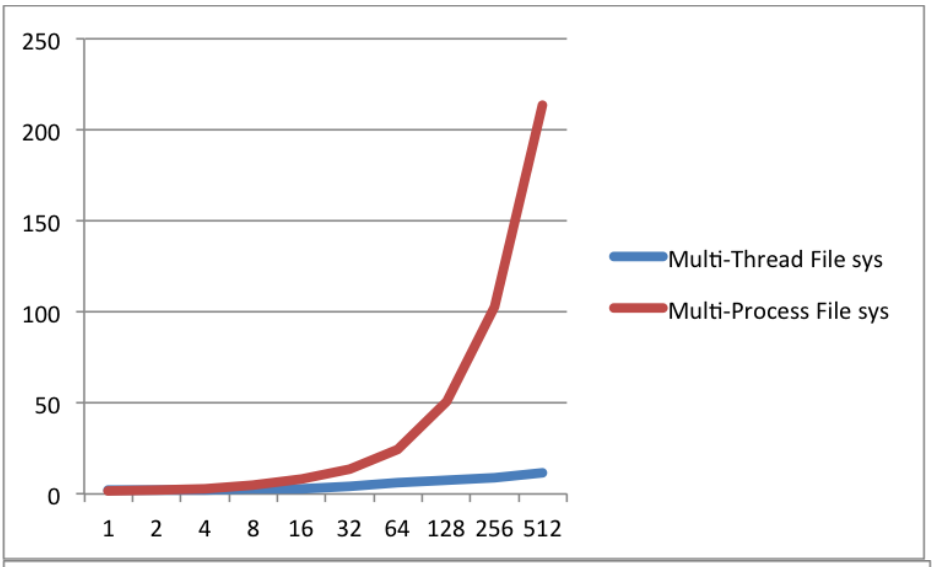

1800
1600
1400
1200
1000
800
600
400
200
0
1 2 4 8 16 32 64 128 256 512
Multi-Thread Dir real
Multi-Process Dir real

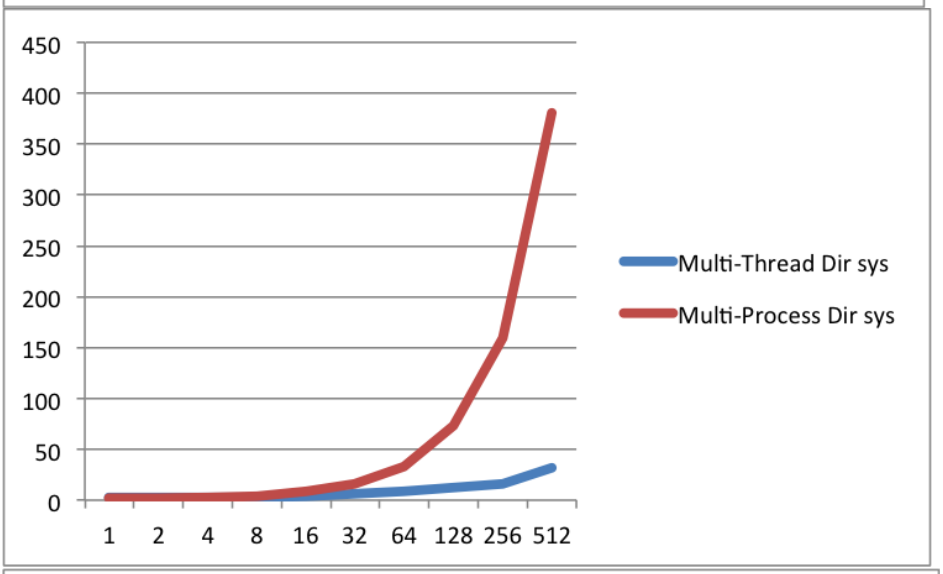

4000
3500
3000
2500
2000
1500
1000
500
0
1 2 4 8 16 32 64 128 256 512
Multi-Thread FileDir real
Multi-Process FileDir real

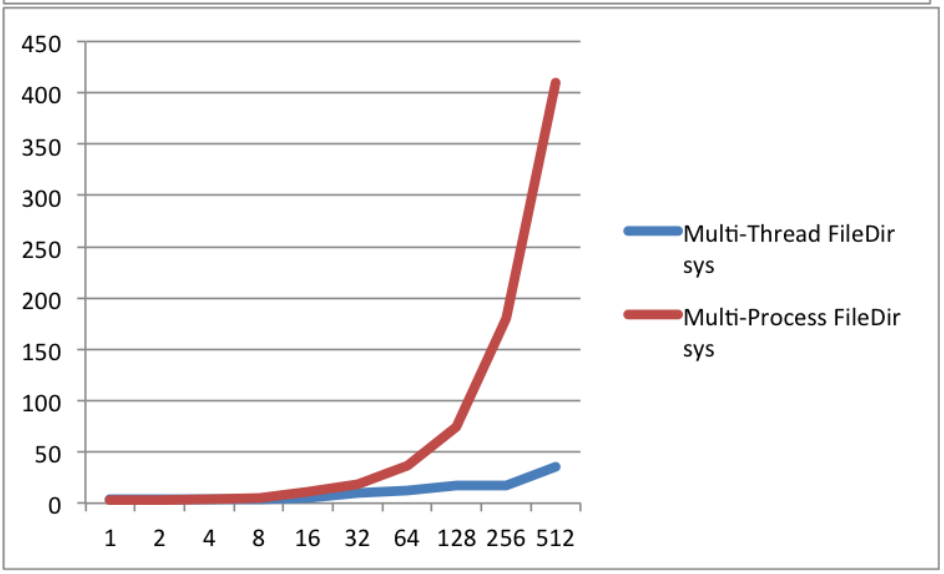

10
9
8
7
6
5
4
3
2
1
0
1 2 4 8 16 32 64 128 256 512
Multi-Thread File user
Multi-Process File User


12
10
8
6
4
2
0
1 2 4 8 16 32 64 128 256 512
Multi-Thread Dir user
Multi-Process Dir User


20
18
16
14
12
10
8
6
4
2
0
1 2 4 8 16 32 64 128 256 512
Multi-Thread FileDir user
Multi-Process FileDir User


250
200
150
100
50
0
1 2 4 8 16 32 64 128 256 512
Multi-Thread File sys
Multi-Process File sys


450
400
350
300
250
200
150
100
50
0
1 2 4 8 16 32 64 128 256 512
Multi-Thread Dir sys
Multi-Process Dir sys


450
400
350
300
250
200
150
100
50
0
1 2 4 8 16 32 64 128 256 512
Multi-Thread FileDir sys
Multi-Process FileDir sys