Assignment 3

CSE 681 SOFTWARE MODELING & ANALYSIS

PROF. GREGORY WAGNER

8/24/2022

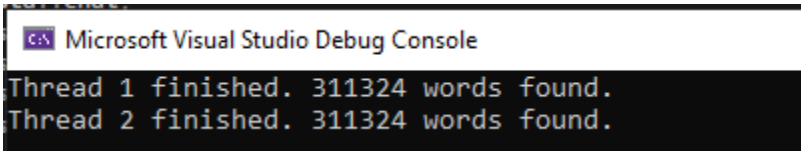
Anthony Redamonti
SYRACUSE UNIVERSITY

Introduction

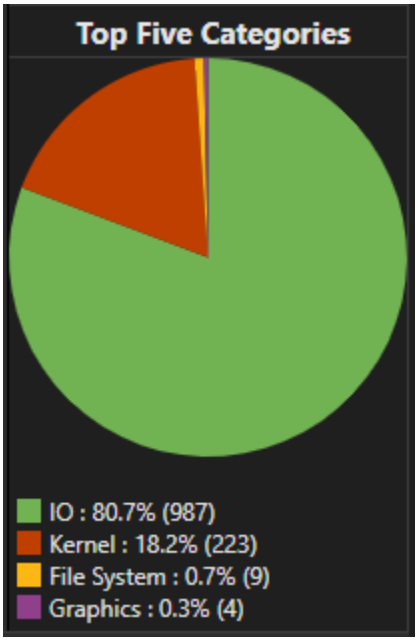
The following programs were written in C++ using the Visual Studio 2022 IDE on a Windows 10 OS. The complete works of William Shakespeare are contained in a text file. The text file is copied so that each thread or process has its own file. The multithreaded project creates 2 threads, each performing the same task. The threads will parse the file and count the total number of words. It will print the total number of words along with its thread I.D. The multiprocessing application performs the same function, except using 2 processes. The output of each program is below.

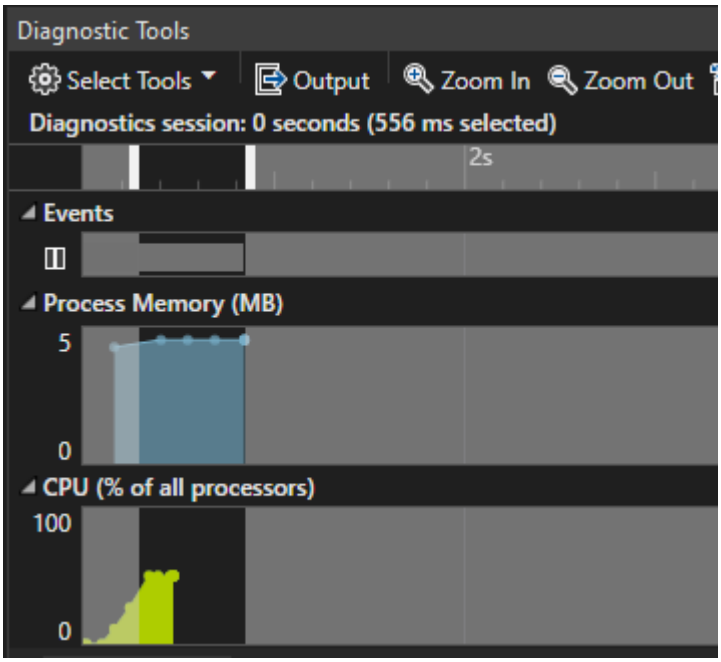
Output Using Two Threads:

Below is the output of the multithreaded program.



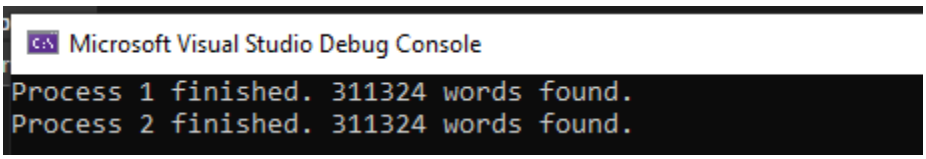
The total amount of process memory used is about 5MB. 80% of the tasks performed by the program involved I/O, which makes sense as it is constantly reading from a file. The total CPU performance got up to as high as 50% of all processors being used.



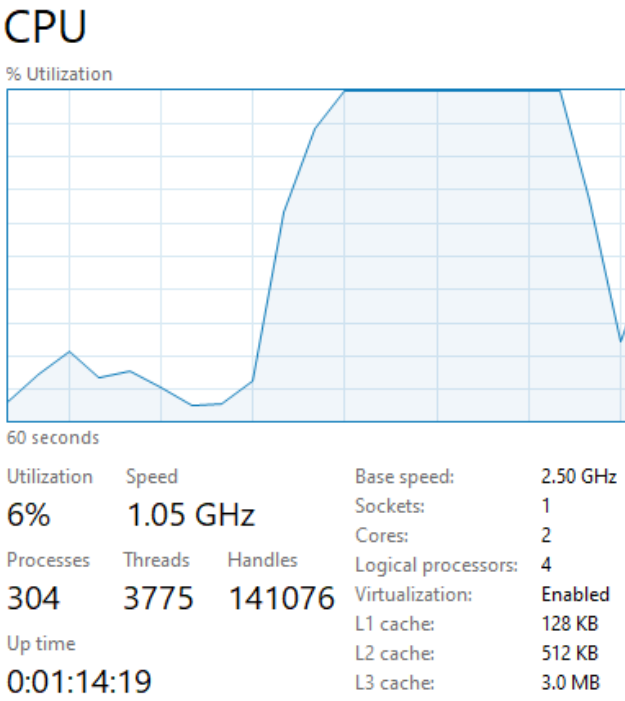


Output Using Two Processes:

Below is the output of the multiprocessing application.



The total amount of process memory used was about 5MB. The total CPU performance got up to as high as 28% of all processors being used. Since the Visual Studio IDE can only monitor the parent process, the task manager must be used to monitor CPU utilization to more accurately measure the performance of the multiprocessing application. Monitoring it showed that the CPU resources got up to as high as 100% for 7 seconds. The disk usage rate remained the same for both programs, 5%.



Conclusion:

The multithreading application performed better than the multiprocessing application on the Windows 10 OS machine. The multiprocessing application required more CPU bandwidth and took 300ms longer to run than the multithreading application.

Main.cpp file for the multithreading application

```
/*
 * Anthony Redamonti
 * Professor Gregory Wagner
 * CSE 681 Software Modeling
 * Assignment 3
 * 8/24/2022
 *
 * Main.cpp
 *
 * The following program creates two threads that perform the same task.
 * Each thread has its own text file containing the complete works of
 * Shakespeare. Each thread will parse the complete works of Shakespeare
 * and count the total number of words. Each thread will then output the
 * total number of words along with its thread ID.
 *
 */

#include <iostream>
#include <string>
#include <thread>
#include <fstream>

using std::endl;
using std::cout;
using std::ifstream;
using std::ofstream;
using std::string;
using std::thread;

// Method called by the thread.
// Counts the number of words in the file and displays it to
// the console.
void threadMethod(const char * inputFile, string threadId){
    ifstream ifstreamObj;
    string data;
    ifstreamObj.open(inputFile);
    int numberOfWords{ 0 };
    while (!ifstreamObj.eof()) {
        std::getline(ifstreamObj, data);
        for (int i = 0; i < data.length(); i++) {
            if (data.at(i) == ' ') {
                numberOfWords = numberOfWords + 1;
            }
        }
    }
    printf("Thread %d finished. %d words found.\n", stoi(threadId), numberOfWords);
}

int main(){

    const char* file1 = "CompleteWorksOfShakespeare1.txt";
    const char* file2 = "CompleteWorksOfShakespeare2.txt";

    // start the threads
    thread thread1(threadMethod, file1, "1");
    thread thread2(threadMethod, file2, "2");
```

```
    // main thread will wait for thread1 and thread2 to finish.
    thread1.join();
    thread2.join();
}
```

Main.cpp file for the multiprocessing application (Parent)

```
/*
 * Anthony Redamonti
 * Professor Gregory Wagner
 * CSE 681 Software Modeling
 * Assignment 3
 * 8/24/2022
 *
 * Main.cpp
 *
 * The following program defines a parent process that creates 2 child processes.
 * The parent will pass as arguments a file path to a text file and a processes
 * ID. The child process will count the number of words in the file and print the
 * number of words along with its process ID.
 */

#include <iostream>
#include <string>
#include <thread>
#include <fstream>
#include <Windows.h>

using std::endl;
using std::cout;
using std::ifstream;
using std::ofstream;
using std::string;
using std::thread;

int main(){

	// there are two seperate files. One for each process.
    string file1 = "CompleteWorksOfShakespeare1.txt";
    string file2 = "CompleteWorksOfShakespeare2.txt";
	string fileArray[2];
	fileArray[0] = file1;
	fileArray[1] = file2;

	// create the processes in a loop.
	for (int i = 0; i < 2; i++) {

		//************Start the processes**************
		STARTUPINFO si;
		PROCESS_INFORMATION pi;

		// create an array to hold the file.
		wchar_t wCharArray[100];

		// local variable for iterating through character array.
		int index{ 0 };
		for (int j = 0; j < fileArray[i].size(); j++) {
			wCharArray[j] = fileArray[i][index];
			index = index + 1;
		}

		wCharArray[index] = ' ';
		index = index + 1;
```

```
            // insert the process ID.
            wCharArray[index] = i+1;
            index = index + 1;

            // end the string with the null character
            wCharArray[index] = 0;

            // convert the process number in string form to LPWSTR
            LPWSTR allArgsLpwstr = wCharArray;

            ZeroMemory(&si, sizeof(si));
            si.cb = sizeof(si);
            ZeroMemory(&pi, sizeof(pi));

            // Start the child process.
            if (!CreateProcess(
                  L"C:\\Users\\aredamonti\\projects\\Software
Modeling\\Assignment3\\Assignment3Process\\x64\\Debug\\Assignment3Process.exe",
                  allArgsLpwstr,  // Command line
                  NULL,           // Process handle not inheritable
                  NULL,           // Thread handle not inheritable
                  FALSE,          // Set handle inheritance to FALSE
                  0,              // No creation flags
                  NULL,           // Use parent's environment block
                  NULL,           // Use parent's starting directory
                  &si,            // Pointer to STARTUPINFO structure
                  &pi)            // Pointer to PROCESS_INFORMATION structure
                  )
            {
                  printf("CreateProcess failed (%d).\n", GetLastError());
                  return -6;
            }

            // Close process and thread handles.
            CloseHandle(pi.hProcess);
            CloseHandle(pi.hThread);
      }
}
```

Main.cpp file for the multiprocessing application (Child)

```
/*
 * Anthony Redamonti
 * Professor Gregory Wagner
 * CSE 681 Software Modeling
 * Assignment 3
 * 8/24/2022
 *
 * The following program defines a child process that is called by a parent
 * process. The parent process passes 2 arguments to the child process: an
 * input file path (must be .txt extension) and a process ID.
 *
 * The child process will count the words in the file and display it to the
 * console along with its process ID.
 *
 */

#include <iostream>
#include <fstream>
#include <string>

using std::endl;
using std::cout;
using std::ifstream;
using std::ofstream;
using std::string;
using std::stoi;

int main(int argc, char* argv[])
{
    // convert the data to a string
    string inputFile = argv[0];

    // process ID
    string processIDStr = argv[1];
    char lastChar = processIDStr[processIDStr.length() - 1];

    ifstream ifstreamObj;
    string data;

    // parse the input text file. Count the number of words.
    try {
        ifstreamObj.open(inputFile);
        int numberOfWords{ 0 };
        while (!ifstreamObj.eof()) {
            std::getline(ifstreamObj, data);
            for (int i = 0; i < data.length(); i++) {
                if (data.at(i) == ' ') {
                    numberOfWords = numberOfWords + 1;
                }
            }
        }

        // display the number of words to the user.
        printf("Process %d finished. %d words found.\n", lastChar, numberOfWords);
    }
    catch (std::exception exceptionObj) {
```

```
        cout << "Exception: " << exceptionObj.what() << endl;
    }
}
```