**Heterogeneous and Cloud Computing**

**Homework #8 (Part A)**

Max Points: 15

**Due: Mon, Nov 24 2014 By 11:59 PM (EST)**

**Email-based help cut-off: 24 hours prior to due date**

|  |
| --- |
| **Objective**: Exploring the computational space spanning CPU and I/O bound tasks.  **Submission**: Once you have successfully tested your program and verified it meets all the formatting requirements, upload:   1. This file saved as a PDF document and named with the convention MUid\_Homework8\_PartA.pdf (where MUid is your Miami University Unique Id). 2. The C++ source files (version #1 and version #2) for this homework. |

# Grading Rubric:

|  |  |
| --- | --- |
|  | This is a graduate course and consequently the expectations in this course are higher. Accordingly, the C++ program submitted for this homework must be operational in order to qualify for earning a full score.  **NOTE: Program that do not compile, have methods longer than 25 lines, or just some skeleton code will be assigned zero score.** |

**-1 point**: For each style violations as reported by the C++ style checker (a slightly relaxed version from Google Inc. automatically downloaded by Makefile from Niihka)

**-3 points**: If the program does not include suitable comments at appropriate points in each method to elucidate flow of thought/logic in each method.

**-1 point**: For each warning message generated by g++, when compiling with -Wall (report all warnings) flag.

# Data download

This homework requires the use of a large data file (at least 1 GB in size) for conducting proper scientific experiments. You may download the data from:

|  |
| --- |
| wget ftp://ftp.ncdc.noaa.gov/pub/data/ghcn/daily/by\_year/2010.csv.gz |

You can gunzip of the data and use 2010.csv for testing.

# Homework Program Requirement

Develop a straightforward C++ program (do not use OpenCL or Map-Reduce constructs) coverts all characters in an input file to another character and writes it to an output file using only iterators and algorithms *i.e.*, without using any traditional looping constructs such as for-loop, range-based-for (huh, what is ranged-based- for?) while-loop, do-while-loop. Your program must accept the following command-line arguments in the following order:

1. The first command-line argument is input file.
2. The second command-line argument is the character to be converted.
3. The third command-line argument is the replacement character.
4. The fourth command-line argument is the output file to which the converted characters are to be written.

Ensure that the implementation with just iterators reads, process, and writes one character at time. This will result in a CPU-bound job.

***Sample Outputs:***

A sample output is shown below (and additional files are supplied for testing):

|  |
| --- |
| $ cat input.txt  11111111111111111  $ ./hw8\_v1 input.txt 1 2 out.txt  $ cat out.txt  22222222222222222  $ |

***Initial Timing Measurements:***

For timing measurements ensure your program is complied with the –O3 optimization flags (make SRC=hw8.cpp OPTS=-O3) and use the following command to measure time:

|  |
| --- |
| $ /usr/bin/time –v ./hw8\_v1 2010.csv 1 2 ~/2010\_out.csv |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Obs#** | **User Time (sec)** | **System Time (sec)** | **Elapsed Time (sec)** | **%CPU** |
| 1 |  |  |  |  |
| 2 |  |  |  |  |
| 3 |  |  |  |  |

Note: If implemented correctly the %CPU should be 99%.

Use the strace to track the top-three system calls (average of 3 runs) that consume most of the system time and record the observations in the table further below:

|  |
| --- |
| $ strace –c ./hw8\_1 2010.csv 1 2 ~/2010\_out.csv |

|  |  |  |  |
| --- | --- | --- | --- |
| **Syscall** | **%time** | **Time (seconds)** | **#Calls** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# Rewriting to reduce number of system calls

With a simple iterator based approach you should observe a large number of calls to read and write system calls. Create a new program (version #2) that functions exactly the same the previous with enhancement to reduce the number of system calls by reading, processing, and writing blocks of characters (instead of one character at a time) using a buffers of different sizes until you determine an optimum size where your program runs fastest (*i.e.*, elapsed time is lowest and if your program is implemented correctly your %CPU will be below 25%). The std::istream::read and std::istream::write methods will come in handy to read blocks of data.

***Timing data:***

In order to find the fastest running configuration, you will have to vary your buffer sizes (say in steps of 2048 bytes) and recording runtimes of your program in the table below. Plot the elapsed time to illustrate the trends you observe to illustrate the choice of buffer size.

|  |
| --- |
| $ /usr/bin/time –v ./hw8\_v2 2010.csv 1 2 ~/2010\_out.csv |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Buffer Size** | **Obs#** | **User Time** | **System Time** | **Elapsed Time** | **%CPU** |
| 1024 | 1 |  |  |  |  |
| 1024 | 2 |  |  |  |  |
| 1024 | 3 |  |  |  |  |
| 2048 | 1 |  |  |  |  |
| 2048 | 2 |  |  |  |  |
| 2048 | 3 |  |  |  |  |
| 4096 | 1 |  |  |  |  |
| 4096 | 2 |  |  |  |  |
| 4096 | 3 |  |  |  |  |
| Keep adding rows to this table as needed. Ensure your data clearly highlights that you have explored the solution space. | | | | | |

|  |
| --- |
| Insert chart showing trends in elapsed time with change in buffer size |

Use the strace to track the top-three system calls (average of 3 runs) that consume most of the system time for your optimal buffer size and record the observations in the table further below:

|  |
| --- |
| $ strace –c ./hw8\_v2 2010.csv 1 2 ~/2010\_out.csv |

|  |  |  |  |
| --- | --- | --- | --- |
| **Syscall** | **%time** | **Time (seconds)** | **#Calls** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

***Inference:***

Using the data collected discuss (4 to 5 sentences is not a discussion and warrants assigning zero score for this part of the homework) if the implementation is CPU bound or I/O bound. In addition, discuss which system calls are consuming time and why?

|  |
| --- |
|  |

# Submission

Once you have successfully tested your program and verified it meets all the requirements, upload:

1. This file saved as a PDF document and named with the convention MUid\_Homework8\_PartA.pdf (where MUid is your Miami University Unique Id).
2. The C++ source files (version #1 and version #2) for this homework.

Verify that your program meets all the requirements as stated in the grading rubric. No credit will be given for submitting skeleton code or programs that do not meet the base case. Ensure your C++ source files are named with the appropriate conventions. Ensure your solution meets the style guidelines by running it through the style checker. Upload each file individually. Do not upload zip/7zip/tar/gzip or any other archive file formats.