|  |  |
| --- | --- |
| **Assignment** | |
| **Course Code** | CSC302A |
| **Course Name** | Operating Systems |
| **Programme** | B.Tech |
| **Department** | CSE |
| **Faculty** | FET |

|  |  |
| --- | --- |
| **Name of the Student** | Satyajit Ghana |
| **Reg. No.** | 17ETCS002159 |
| **Semester/Year** | 5/2019 |
| **Course Leader(s)** | Ms. Naveeta |



|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Declaration Sheet | | | | | | | | |
| Student Name | Satyajit Ghana | | | | | | | |
| Reg. No | 17ETCS002159 | | | | | | | |
| Programme | B.Tech | | | | | Semester/Year | 05/2019 | |
| Course Code | CSC302A | | | | | | | |
| Course Title | Operating Systems | | | | | | | |
| Course Date |  | | to |  | | | | |
| Course Leader | Ms. Naveeta | | | | | | | |
| **Declaration**  The assignment submitted herewith is a result of my own investigations and that I have conformed to the guidelines against plagiarism as laid out in the Student Handbook. All sections of the text and results, which have been obtained from other sources, are fully referenced. I understand that cheating and plagiarism constitute a breach of University regulations and will be dealt with accordingly. | | | | | | | | |
| Signature of the Student | |  | | | | | Date |  |
| Submission date stamp  (by Examination & Assessment Section) | |  | | | | | | |
| Signature of the Course Leader and date | | | | | Signature of the Reviewer and date | | | |
|  | | | | |  | | | |

# Contents

[Declaration Sheet ii](#_Toc22735078)

[Contents iii](#_Toc22735079)

[List of Figures iv](#_Toc22735080)

[1 Question 1 5](#_Toc22735081)

[1.1 Development of the Application 5](#_Toc22735082)

[1.1.1 Using Sequential Approach 6](#_Toc22735083)

[1.1.2 Using Multithreaded Approach 6](#_Toc22735084)

[1.2 Comparison of Execution time and Analysis 7](#_Toc22735085)

[1.2.1 Analysis of the performance statistics 10](#_Toc22735086)

[2 Question 2 11](#_Toc22735087)

[2.1 Number of page faults that occur when FIFO, LRU, and Optimal page replacement algorithms are used respectively 11](#_Toc22735088)

[2.2 Diagram of the probability density function of distance strings based on LRU 12](#_Toc22735089)

[2.3 Recommendation of an optimal number of physical page frames appropriate for the given string of accesses 12](#_Toc22735090)

[Appendix A 13](#_Toc22735091)

[Bibliography 20](#_Toc22735092)

# List of Figures

[Figure 1‑1 ccount help 5](#_Toc22735073)

[Figure 1‑2 ccount sample output 5](#_Toc22735074)

[Figure 1‑3 ccount - sample test file 6](#_Toc22735075)

# Question 1

Solution to Question No. 1

## Development of the Application

Complete Source Code of the Program is attached in Appendix A

The approach to the problem is to have two methods, sequential and multithreaded approach, the option is taken as a command line argument for the same. In both the methods the common task done by main thread is to combine the result obtained from the threads.

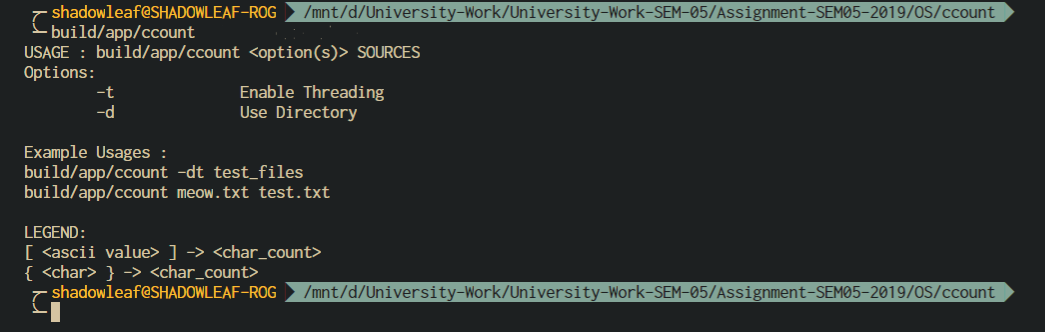


Figure ‑ ccount help

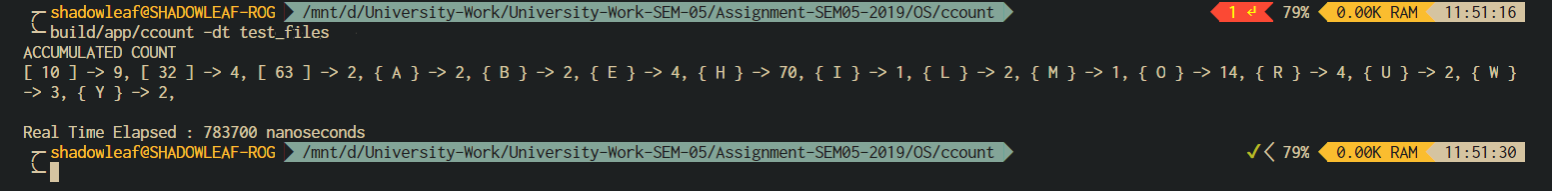


Figure ‑ ccount sample output

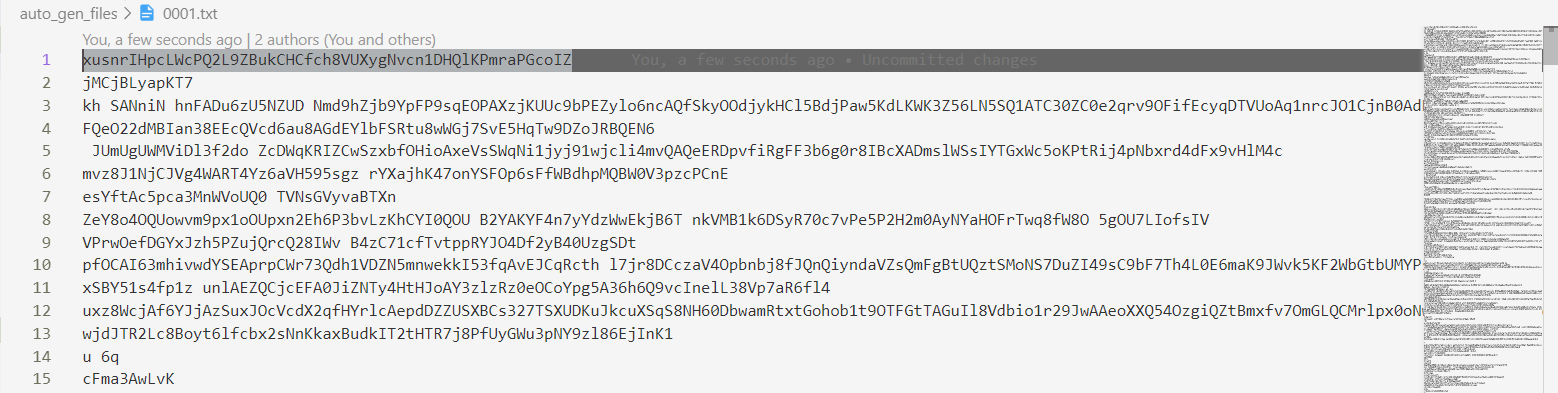


Figure ‑ ccount - sample test file

The test files are generated using Bash Scripting in Linux, the source code for the script can be found in generate\_files.sh.

### Using Sequential Approach

The sequential approach is pretty straight forward, the files are read from the disk one by one by the main thread and the result is pushed to a vector, here all the work is done by the main thread, i.e. disk I/O, counting the characters and also accumulating the results.

The sequential approach code is as simple as

        for (auto& file : files) {

**std**::map<char, int>\* ccount = new **std**::**map**<char, int>();

            \*ccount = file.**get**()->**get\_char\_count**();

            results.**emplace\_back**(static\_cast<void\*>(ccount));

        }

Here we iterate over all the files in the directory and then run get\_char\_count() for each of the file, which counts the characters and the results are stored in the results vector.

The function definition of get\_char\_count can be found in Appendix A. It’s a simple iterate over all the contents of the file character-by-character and then add the count into a std::map.

### Using Multithreaded Approach

In the Multithreaded approach the each of the file is assigned as a task to different thread, the main thread then waits for all these threads to complete their work and the result is pushed to the result vector, the main thread then does the accumulation of the results.

The threads are generated using the gen\_worker\_threads function,

*// generate and run threads*

**std**::vector<pthread\_t> threads = **bromine**::**threader**::**gen\_worker\_threads**(&**bromine**::**file**::threadable\_ccount\_fun, fargs);

this function generates threads.size() number of threads and runs them using pthread\_create, the function arguments are taken as fargs, the function then returns the thread ids of the threads generated.

The gen\_worker\_threads function is more elaborated as

**std**::**vector**<pthread\_t> **bromine**::**threader**::**gen\_worker\_threads**(void\* (\*thread\_fun)(void\*), **std**::**vector**<void\*> fargs) {

*// create a vector of worker threads*

**std**::vector<pthread\_t> **worker\_threads**(fargs.**size**());

    for (int i = 0; i < (int)worker\_threads.**size**(); i++) {

**pthread\_create**(&worker\_threads[i], NULL, thread\_fun, fargs[i]);

    }

    return worker\_threads;

}

The results are accumulated using pthread\_join function, that waits for all the threads that were spawned earlier to complete and stores the data in a vector.

## Comparison of Execution time and Analysis

For comparing the results, mostly CPU usage is affected, since in the single threaded application, only one the available cores are used, to do the testing **perf**, a testing utility for Linux was used that provided satisfiable results, the testing utility was created in bash scripting for automation, the source code for which can be found in testing.sh

Testing Bench specifications is as follows

CPU: Intel Core i3-6006U @2.00Ghz 3M Cache [ 2 Cores 4 Threads(HyperThreading ON) ]

RAM: 4GB LPDDR4

DISK: Seagate 500GB Barracuda 5400rpm

OS: Manjaro (Arch Linux) [ Linux Kernel 4.9 ]

COMPILER: GCC 9.1 with POSIX Thread

**Single Threaded – perf stat**

ACCUMULATED COUNT

[ 10 ] -> 251225, [ 32 ] -> 249547, { 0 } -> 249984, { 1 } -> 249856, { 2 } -> 249470, { 3 } -> 250628, { 4 } -> 250043, { 5 } -> 250199, { 6 } -> 250399, { 7 } -> 249785, { 8 } -> 250736, { 9 } -> 250580, { A } -> 249349, { B } -> 249697, { C } -> 250086, { D } -> 250345, { E } -> 250988, { F } -> 249419, { G } -> 250584, { H } -> 250114, { I } -> 250605, { J } -> 249629, { K } -> 250470, { L } -> 249376, { M } -> 250031, { N } -> 249913, { O } -> 249845, { P } -> 249611, { Q } -> 249478, { R } -> 249601, { S } -> 249083, { T } -> 250680, { U } -> 250305, { V } -> 249700, { W } -> 249404, { X } -> 249967, { Y } -> 250037, { Z } -> 250192, { a } -> 249858, { b } -> 249909, { c } -> 250924, { d } -> 249329, { e } -> 250550, { f } -> 249603, { g } -> 250269, { h } -> 250333, { i } -> 249218, { j } -> 249101, { k } -> 249405, { l } -> 250815, { m } -> 250157, { n } -> 250132, { o } -> 250876, { p } -> 249498, { q } -> 250057, { r } -> 249850, { s } -> 249597, { t } -> 250008, { u } -> 249903, { v } -> 249462, { w } -> 250003, { x } -> 248996, { y } -> 250862, { z } -> 250324,

Real Time Elapsed : 16722432861 nanoseconds

Performance counter stats for 'build/app/ccount -d auto\_gen\_files':

16,726.46 msec task-clock # 1.000 CPUs utilized

27 context-switches # 0.002 K/sec

0 cpu-migrations # 0.000 K/sec

460 page-faults # 0.028 K/sec

33,374,743,105 cycles # 1.995 GHz

25,169,226,542 instructions # 0.75 insn per cycle

4,870,832,127 branches # 291.205 M/sec

99,689,709 branch-misses # 2.05% of all branches

16.728176104 seconds time elapsed

8.150538000 seconds user

8.535908000 seconds sys

**Multithreaded – perf stat**

ACCUMULATED COUNT

[ 10 ] -> 251225, [ 32 ] -> 249547, { 0 } -> 249984, { 1 } -> 249856, { 2 } -> 249470, { 3 } -> 250628, { 4 } -> 250043, { 5 } -> 250199, { 6 } -> 250399, { 7 } -> 249785, { 8 } -> 250736, { 9 } -> 250580, { A } -> 249349, { B } -> 249697, { C } -> 250086, { D } -> 250345, { E } -> 250988, { F } -> 249419, { G } -> 250584, { H } -> 250114, { I } -> 250605, { J } -> 249629, { K } -> 250470, { L } -> 249376, { M } -> 250031, { N } -> 249913, { O } -> 249845, { P } -> 249611, { Q } -> 249478, { R } -> 249601, { S } -> 249083, { T } -> 250680, { U } -> 250305, { V } -> 249700, { W } -> 249404, { X } -> 249967, { Y } -> 250037, { Z } -> 250192, { a } -> 249858, { b } -> 249909, { c } -> 250924, { d } -> 249329, { e } -> 250550, { f } -> 249603, { g } -> 250269, { h } -> 250333, { i } -> 249218, { j } -> 249101, { k } -> 249405, { l } -> 250815, { m } -> 250157, { n } -> 250132, { o } -> 250876, { p } -> 249498, { q } -> 250057, { r } -> 249850, { s } -> 249597, { t } -> 250008, { u } -> 249903, { v } -> 249462, { w } -> 250003, { x } -> 248996, { y } -> 250862, { z } -> 250324,

Real Time Elapsed : 6572352826 nanoseconds

Performance counter stats for 'build/app/ccount -dt auto\_gen\_files':

25,699.24 msec task-clock # 3.907 CPUs utilized

10,448 context-switches # 0.407 K/sec

189 cpu-migrations # 0.007 K/sec

1,446 page-faults # 0.056 K/sec

51,249,351,391 cycles # 1.994 GHz

27,129,213,412 instructions # 0.53 insn per cycle

5,412,543,554 branches # 210.611 M/sec

102,972,422 branch-misses # 1.90% of all branches

6.576939478 seconds time elapsed

10.477496000 seconds user

14.992854000 seconds sys

**Single Threaded – perf report**

# Total Lost Samples: 0

#

# Samples: 71K of event 'cycles:u'

# Event count (approx.): 3241922147

#

# Overhead Command Shared Object Symbol

72.42% ccount ccount [.] bromine::file::get\_char\_count

19.09% ccount [unknown] [k] 0xffffffffa260015f

6.60% ccount libpthread-2.29.so [.] \_\_libc\_read

1.24% ccount ccount [.] read@plt

0.12% ccount ccount [.] main

0.08% ccount [unknown] [k] 0xffffffffa2600b07

0.08% ccount libstdc++.so.6.0.26 [.] std::\_Rb\_tree\_insert\_and\_rebalance

0.07% ccount libc-2.29.so [.] malloc

0.07% ccount libc-2.29.so [.] \_int\_malloc

0.04% ccount ld-2.29.so [.] \_dl\_lookup\_symbol\_x

0.03% ccount ccount [.] operator delete@plt

0.03% ccount libstdc++.so.6.0.26 [.] std::local\_Rb\_tree\_decrement

0.02% ccount libc-2.29.so [.] \_int\_free

0.02% ccount libstdc++.so.6.0.26 [.] operator new

**Multithreaded – perf report**

# Total Lost Samples: 0

#

# Samples: 90K of event 'cycles:u'

# Event count (approx.): 26392820082450

#

# Overhead Command Shared Object Symbol

40.00% ccount ccount [.] bromine::file::get\_char\_count

30.00% ccount libpthread-2.29.so [.] \_\_libc\_read

10.00% ccount [unknown] [k] 0xffffffffa260015f

10.00% ccount libpthread-2.29.so [.] \_\_pthread\_disable\_asynccancel

10.00% ccount libpthread-2.29.so [.] \_\_pthread\_enable\_asynccancel

0.00% ccount ccount [.] read@plt

0.00% ccount libc-2.29.so [.] malloc

0.00% ccount [unknown] [k] 0xffffffffa2600b07

0.00% ccount libc-2.29.so [.] \_int\_malloc

0.00% ccount ccount [.] main

0.00% ccount libc-2.29.so [.] \_int\_free

0.00% ccount libstdc++.so.6.0.26 [.] std::\_Rb\_tree\_insert\_and\_rebalance

0.00% ccount ld-2.29.so [.] \_dl\_lookup\_symbol\_x

### Analysis of the performance statistics

Table Performance Analysis

|  |  |  |
| --- | --- | --- |
| **Performance Parameter** | **Single-Threaded** | **Multi-Threaded** |
| Time Elapsed | 16.72817 secs | 6.57693 secs |
| IPC | 0.75 ins/cycle | 0.53 ins/cycle |
| Context Switches | 27 | 10,448 |
| CPU’s Utilized | 1.000 | 3.907 |

From the above performance parameters, we can clearly observe that the multi-threaded program is approximately 3X faster than the single-threaded. The processor used here had 2 cores and 4 hyperthreaded cores, hence the CPU’s utilized in multi-threaded is approximately 4, or all of them, in single threaded only one CPU is utilized. Another thing to note is that, since we had 10000 files and hence 10000 threads that are created, the number of context switches is relatively very high compared to the single threaded program.

From the perf report we can determine the sub-routines in the program that cause the major overhead, this tells us how the work is distributed among the threads. The major overhead in our program is bromine::file::get\_char\_count, in single threaded this has 72.42% overhead, while in multithreaded its brought down to 40.00%, since now we have opened multiple files all at once, the get\_char\_count overhead is not there anymore, suppose a thread is currently executing ccount, the other thread might be opening a file, hence the overhead is distributed among reading and processing the file, as we can see in the perf report of multithreaded, the \_\_libc\_read function takes up about 30.00% overhead, in single threaded, the file opening and ccount is sequential, i.e. one single thread can open the file and then process the file, majority of the time is spent on processing the file, and only one single CPU is utilized.

# Question 2

Solution to Question 2

## Number of page faults that occur when FIFO, LRU, and Optimal page replacement algorithms are used respectively

1. FIFO

16 Page Faults

0 1 2 3 2 3 0 4 5 2 3 1 4 3 2 6 3 2 1 2

0 1 2 3 3 3 0 4 5 2 3 1 4 4 2 6 3 3 1 2

0 1 2 2 2 3 0 4 5 2 3 1 1 4 2 6 6 3 1

0 1 1 1 2 3 0 4 5 2 3 3 1 4 2 2 6 3

P P P P P P P P P P P P P P P P

1. LRU

14 Page Faults

0 1 2 3 2 3 0 4 5 2 3 1 4 3 2 6 3 2 1 2

0 1 2 3 2 3 0 4 5 2 3 1 4 3 2 6 3 2 1 2

0 1 2 3 2 3 0 4 5 2 3 1 4 3 2 6 3 2 1

0 1 1 1 2 3 0 4 5 2 3 1 4 3 2 6 3 3

P P P P P P P P P P P P P P

1. Optimal Page Replacement

10 Page Faults

0 1 2 3 2 3 0 4 5 2 3 1 4 3 2 6 3 2 1 2

0 0 0 0 0 0 0 4 5 5 5 1 4 4 4 6 6 6 1 1

1 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3

2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2

P P P P P P P P P P

## Diagram of the probability density function of distance strings based on LRU

Assume LRU algorithm is used.

0 1 2 3 2 3 0 4 5 2 3 1 4 3 2 6 3 2 1 2

0 1 2 3 2 3 0 4 5 2 3 1 4 3 2 6 3 2 1 2

0 1 2 3 2 3 0 4 5 2 3 1 4 3 2 6 3 2 1

0 1 1 1 2 3 0 4 5 2 3 1 4 3 2 6 3 3

0 0 0 1 2 3 0 4 5 2 2 1 4 4 4 6 6

1 2 3 0 4 5 5 5 1 1 1 4 4

1 1 1 0 0 0 0 5 5 5 5 5

0 0 0 0 0

String

Distance x x x x 2 2 4 x x 5 5 6 5 3 4 x 3 3 5 2

x indicates the infinity distance.

P(1) = 0

P(2) = 3/20 = 0.15

P(3) = 3/20 = 0.15

P(4) = 2/20 = 0.1

P(5) = 4/20 = 0.2

P(6) = 1/20 = 0.05

P(infinity) = 7/20 = 0.35

## Recommendation of an optimal number of physical page frames appropriate for the given string of accesses

Based on the diagram 5 frames would be a good choice.

# Appendix A

Source Code for Program in Question 1

Project Structure

* app
  + file\_ops.cpp
  + file\_ops.hpp
  + main.cpp
  + threader.cpp
  + threader.hpp

**main.cpp**

*// C++ Includes*

#include <algorithm>

#include <chrono>

#include <iostream>

#include <map>

#include <memory>

#include <numeric>

#include <vector>

*// C Includes*

#include <dirent.h>

#include <fcntl.h>

#include <pthread.h>

#include <stdio.h>

#include <sys/stat.h>

#include <sys/types.h>

#include <unistd.h>

*// User defined Includes*

#include "file\_ops.hpp"

#include "threader.hpp"

void **help**(char name[]);

int **main**(int argc, char\*\* argv) {

*// vector of files*

**std**::vector<**std**::shared\_ptr<**bromine**::file>> files;

    bool is\_directory = false;

    bool is\_threaded = false;

    int opt;

    while ((opt = **getopt**(argc, argv, ":dt")) != -1) {

        switch (opt) {

            case 'd':

                is\_directory = true;

                break;

            case ':':

**std**::cout << "MISSING FOLDER NAME";

**help**(argv[0]);

**exit**(EXIT\_FAILURE);

                break;

            case '?':

**std**::cout << "UNKNOWN OPTION : " << argv[optind];

                break;

            case 't':

                is\_threaded = true;

                break;

        }

    }

    if (is\_directory) {

*// directory path used*

        DIR\* d;

        struct **dirent**\* dir;

        d = **opendir**(argv[optind]);

        if (d) {

            while ((dir = **readdir**(d)) != nullptr) {

                if (dir->d\_type == DT\_REG) {

**std**::string **rel\_path**(dir->d\_name);

                    rel\_path = **std**::**string**(argv[optind]) + "/" + rel\_path;

**std**::shared\_ptr<**bromine**::file> **ptr**(new **bromine**::**file**(rel\_path));

                    files.**emplace\_back**(ptr);

                }

            }

        } else {

**std**::cerr << "ERROR OPENING DIRECTORY " << argv[2] << **std**::endl;

**exit**(EXIT\_FAILURE);

        }

    } else {

        if (argc == 1) {

**help**(argv[0]);

**exit**(EXIT\_FAILURE);

        }

        for (; optind < argc; optind++) {

*// files are specified in argv*

            for (int i = 1; i < argc; i++) {

**std**::shared\_ptr<**bromine**::file> **ptr**(new **bromine**::**file**(argv[optind]));

                files.**emplace\_back**(ptr);

            }

        }

    }

**std**::vector<void\*> results;

    auto start = **std**::**chrono**::**high\_resolution\_clock**::**now**();

    if (is\_threaded) {*// MULTITHREADED*

*// transform the vector of file arguments into void\* vector*

**std**::vector<void\*> **fargs**(files.**size**());

**std**::**transform**(files.**begin**(), files.**end**(), fargs.**begin**(), [](**std**::**shared\_ptr**<**bromine**::**file**> p) {

            return static\_cast<void\*>(p.**get**());

        });

*// generate and run threads*

**std**::vector<pthread\_t> threads = **bromine**::**threader**::**gen\_worker\_threads**(&**bromine**::**file**::threadable\_ccount\_fun, fargs);

*// join the threads and get the results*

        results = **bromine**::**threader**::**get\_threads\_results**(threads);

    } else {*// SEQUENCIAL*

        for (auto& file : files) {

*// please use shared\_ptr or unique\_ptr*

**std**::map<char, int>\* ccount = new **std**::**map**<char, int>();

            \*ccount = file.**get**()->**get\_char\_count**();

            results.**emplace\_back**(static\_cast<void\*>(ccount));

        }

    }

*// variable to store the accumulated results*

**std**::map<char, int> accumulated\_vals;

*// accumulate the results in the main thread*

    for (auto& result : results) {

**std**::map<char, int> ccount = \*static\_cast<**std**::map<char, int>\*>(result);

        for (auto& elem : ccount) {

            if (accumulated\_vals[elem.first]) {

                accumulated\_vals[elem.first] += elem.second;

            } else {

                accumulated\_vals[elem.first] = elem.second;

            }

        }

    }

    auto end = **std**::**chrono**::**high\_resolution\_clock**::**now**();

**std**::cout << "ACCUMULATED COUNT" << **std**::endl;

*// print the character count*

**bromine**::**file**::**print\_ccount**(accumulated\_vals);

    auto time\_taken = **std**::**chrono**::**duration\_cast**<**std**::**chrono**::**nanoseconds**>(end - start).**count**();

*// time\_taken \*= 1e-9;*

**std**::cout << "\nReal Time Elapsed : " << **std**::fixed << time\_taken << " nanoseconds" << **std**::endl;

}

*// for printing the help documentation*

void **help**(char name[]) {

**std**::cerr << "USAGE : " << name << " <option(s)> SOURCES\n"

              << "Options:\n"

              << "\t-t\t\tEnable Threading\n"

              << "\t-d\t\tUse Directory\n"

              << "\nExample Usages : \n"

              << name << " -dt test\_files\n"

              << name << " meow.txt test.txt\n"

              << "\nLEGEND:"

              << "\n[ <ascii value> ] -> <char\_count>"

              << "\n{ <char> } -> <char\_count>"

              << **std**::endl;

}

**file\_ops.hpp**

#pragma once

#include <map>

#include <string>

namespace **bromine** {

class **file** {

   private:

*// file name*

**std**::string file\_name;

*// the file descriptor*

    int fd;

*// to check if the file is closed*

    bool isclosed = true;

   public:

**file**() {

        file\_name = "";

        fd = 0;

    };

**file**(**std**::**string** file\_name);

**~file**();

*// fetch the file descriptor*

    int **get\_fd**() { return this->fd; };

*/\*\**

*\* Returns the character count*

*\* @param none*

*\* @return character count as a map*

*\*/*

**std**::**map**<char, int> **get\_char\_count**();

*/\*\**

*\* Opens the file*

*\* @param file name as a string*

*\* @return none*

*\*/*

    void **open**(**std**::**string** file\_name);

**std**::**string** **get\_file\_name**() { return this->file\_name; };

*// char count function*

    static void **print\_ccount**(const **std**::**map**<char, int>& ccount);

*// threadable char count*

    static void\* **threadable\_ccount\_fun**(void\*);

};

}*// namespace bromine*

**file\_ops.cpp**

*// user includes*

#include "file\_ops.hpp"

*// system includes*

#include <fcntl.h>

#include <stdio.h>

#include <sys/stat.h>

#include <sys/types.h>

#include <unistd.h>

#include <iostream>

#include <memory>

#include <utility>

*/\*\**

*\* Constructor*

*\* @param string file\_name*

*\* @return object of bromine::file*

*\*/*

**bromine**::**file**::**file**(**std**::**string** file\_name) : **file\_name**(file\_name) {

    this->**open**(this->file\_name);

}

*/\*\**

*\* Destructor*

*\* close the file if it was open*

*\*/*

**bromine**::**file**::**~file**() {

    if (!isclosed) {

*// for debugging*

*// std::cout << "CLOSING " << this->file\_name << std::endl;*

**close**(this->fd);

    }

    this->isclosed = true;

}

*/\*\**

*\* Opens the file, given the file\_name*

*\* @param string file\_name*

*\* @return none*

*\*/*

void **bromine**::**file**::**open**(**std**::**string** file\_name) {

    this->file\_name = file\_name;

    if (this->fd = ::**open**(this->file\_name.**c\_str**(), O\_RDONLY); this->fd != -1) {

*// std::cout << "SUCCESSFULLY OPENED " << file\_name << std::endl;*

        this->isclosed = false;

    } else {

        throw **std**::**runtime\_error**("ERROR OPENING FILE : " + file\_name);

    }

}

*/\*\**

*\* Gets the character count from the file*

*\* @param none*

*\* @return map of char, int which is the character count*

*\*/*

**std**::**map**<char, int> **bromine**::**file**::**get\_char\_count**() {

**std**::map<char, int> ccount;

    if (this->fd < 0) {

**std**::cerr << "FILE ERROR" << **std**::endl;

        return ccount;

    }

    char buffer[2];

*// seek to start of the file*

**lseek**(this->fd, 0, SEEK\_SET);

    while (**read**(this->fd, &buffer, 1) == 1) {

*// for debugging*

*// std::cout << "#" << buffer[0] << "#";*

*// increment the character count of the character*

        ccount[buffer[0]]++;

    }

    return ccount;

}

*/\*\**

*\* Threadable Character Cound function which can be passed*

*\* to gen\_worker\_threads function in threader library*

*\*/*

void\* **bromine**::**file**::**threadable\_ccount\_fun**(void\* file\_obj) {

**bromine**::file\* file = static\_cast<**bromine**::file\*>(file\_obj);

*// how stupid am i to use RAW pointer ?*

*// never use RAW pointer, use unique pointer and Move Semantics*

*// and make use of RVO (Return Value Optimization)*

    auto ccount = new **std**::**map**<char, int>(file->**get\_char\_count**());

*// thread count needs to be type casted to void\**

    return static\_cast<void\*>(ccount);

}

*/\*\**

*\* Prints the Character Count*

*\* @param map<char, int> ccount*

*\* @return none*

*\*  ccount*

*\* @return none*

*\*/*

void **bromine**::**file**::**print\_ccount**(const **std**::**map**<char, int>& ccount) {

*// elem.first is the character*

*// elem.second is the character count*

    for (auto& elem : ccount) {

        if (**isalnum**(elem.first)) {

**std**::cout << "{ " << elem.first << " } -> " << elem.second << ", ";

        } else {

**std**::cout << "[ " << static\_cast<unsigned>(elem.first) << " ] -> " << elem.second << ", ";

        }

    }

**std**::cout << **std**::endl;

}

**threader.hpp**

#pragma once

#include <pthread.h>

#include <vector>

namespace **bromine** {

class **threader** {

   public:

    static **std**::**vector**<pthread\_t> **gen\_worker\_threads**(void\* (\*thread\_fun)(void\*), **std**::**vector**<void\*> fargs);

    static **std**::**vector**<void\*> **get\_threads\_results**(**std**::**vector**<pthread\_t>);

};

}*// namespace bromine*

*// namespace bromine*

**threader.cpp**

#include "threader.hpp"

#include <pthread.h>

#include <iostream>

#include <map>

#include <memory>

#include "file\_ops.hpp"

*/\*\**

*\* generates worker threads*

*\* @param pointer to the thread function, arguments to the thread function*

*\* @return vector of threads*

*\*/*

**std**::**vector**<pthread\_t> **bromine**::**threader**::**gen\_worker\_threads**(void\* (\*thread\_fun)(void\*), **std**::**vector**<void\*> fargs) {

*// create a vector of worker threads*

**std**::vector<pthread\_t> **worker\_threads**(fargs.**size**());

    for (int i = 0; i < (int)worker\_threads.**size**(); i++) {

**pthread\_create**(&worker\_threads[i], NULL, thread\_fun, fargs[i]);

    }

    return worker\_threads;

}

*/\*\**

*\* joins the generated threads and return the results*

*\* @param vector of threads*

*\* @return vector of void\* which are the results from the threads*

*\*/*

**std**::**vector**<void\*> **bromine**::**threader**::**get\_threads\_results**(**std**::**vector**<pthread\_t> threads) {

    auto results = **std**::**vector**<void\*>(threads.**size**());

    for (int i = 0; i < (int)threads.**size**(); i++) {

**pthread\_join**(threads[i], &results[i]);

    }

    return results;

}

**generate\_files.sh**

*#!/bin/bash*

for n in $(seq 1 $1); do

    < /dev/urandom tr -dc "\n [:alnum:]" | head -c$2 > $( **printf** %04d "$n" ).txt

done

**testing.sh**

*#!/bin/bash*

COLS=$(tput cols)

function **print\_head** {

    print\_line

**echo** -e "$1\n"

}

function **print\_line** {

**printf** '=%.0s' $(seq $COLS)

**printf** "\n"

}

start=$SECONDS

print\_head "Building"

rm -rf build 2> /dev/null

mkdir build

(cd build; cmake -DCMAKE\_BUILD\_TYPE=Release ..; make)

*# Perf Report Directory*

rm -r perf\_report &> /dev/null

mkdir perf\_report

print\_head "TESTING PROGRAM TIMES"

print\_head "Generating Files"

rm -rf auto\_gen\_files 2> /dev/null

mkdir auto\_gen\_files

(cd auto\_gen\_files; sh ../generate\_files.sh 400 40000)

**echo** -e "Generated 400 files with 40000 characters each\n"

print\_head "Single Threaded Run"

perf stat build/app/ccount -d auto\_gen\_files |& tee perf\_report/perf\_stat\_st.txt

print\_head "Multi Threaded Run"

perf stat build/app/ccount -dt auto\_gen\_files |& tee perf\_report/perf\_stat\_mt.txt

*# Run Perf Benchmarks and save to file*

print\_head "Running Perf . . . (could take a while) "

*# single threaded run*

perf record -s build/app/ccount -d auto\_gen\_files

mv perf.data perf\_report/perf\_st.data

perf report --stdio -i perf\_report/perf\_st.data |& tee perf\_report/perf\_report\_st.txt

*# multi threaded run*

perf record -s build/app/ccount -dt auto\_gen\_files

mv perf.data perf\_report/perf\_mt.data

perf report --stdio -i perf\_report/perf\_mt.data |& tee perf\_report/perf\_report\_mt.txt

**echo** -e "perf records and output saved to perf\_report/\n"

duration=$(( SECONDS - start ))

**echo** -e "testing took $duration seconds\n"

*# print\_head "Cleaning Up"*

*# rm -r auto\_gen\_files 2> /dev/null*

# Bibliography