|  |  |
| --- | --- |
| **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 | <semester>/<year> | |
| Course Code | <course code> | | | | | | | |
| Course Title | <course title> | | | | | | | |
| Course Date |  | | to |  | | | | |
| Course Leader | <course leader(s)> | | | | | | | |
| **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](#_Toc21363770)

[Contents iii](#_Toc21363771)

[List of Figures iv](#_Toc21363772)

[1 Question 1 5](#_Toc21363773)

[1.1 Development of the Application 5](#_Toc21363774)

[1.1.1 Using Sequential Approach 5](#_Toc21363775)

[1.1.2 Using Multithreaded Approach 5](#_Toc21363776)

[1.2 Comparison of Execution time and Analysis 5](#_Toc21363777)

[2 Question 2 6](#_Toc21363778)

[2.1 Number of page faults that occur when FIFO, LRU, and Optimal page replacement algorithms are used respectively 6](#_Toc21363779)

[2.2 Diagram of the probability density function of distance strings based on LRU 6](#_Toc21363780)

[2.3 Recommendation of an optimal number of physical page frames appropriate for the given string of accesses 6](#_Toc21363781)

[Appendix A 7](#_Toc21363782)

[Bibliography 14](#_Toc21363783)

# List of Figures

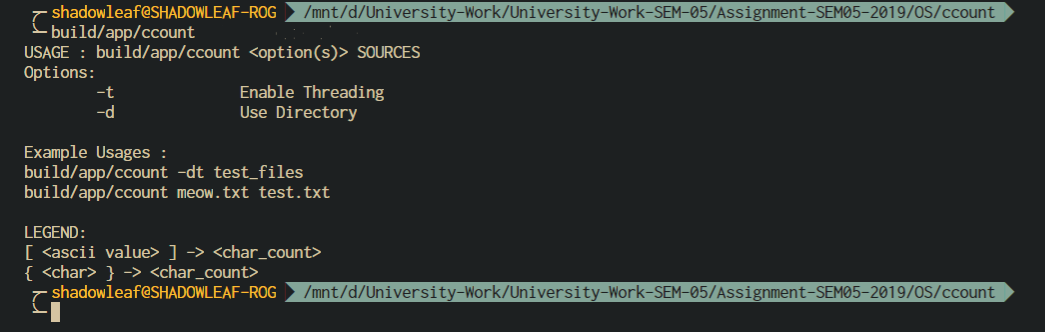
# Question 1

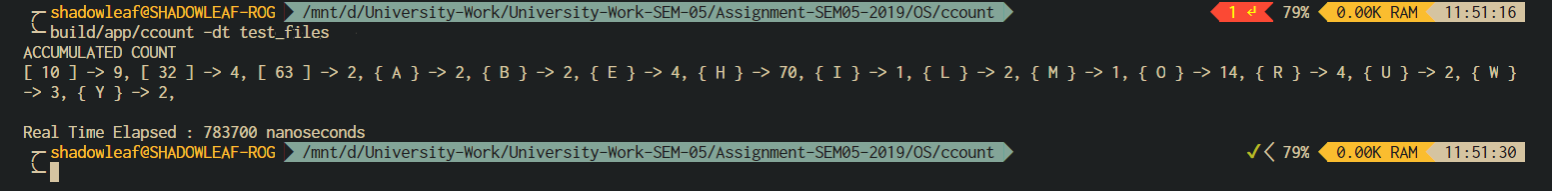
Solution to Question No. 1

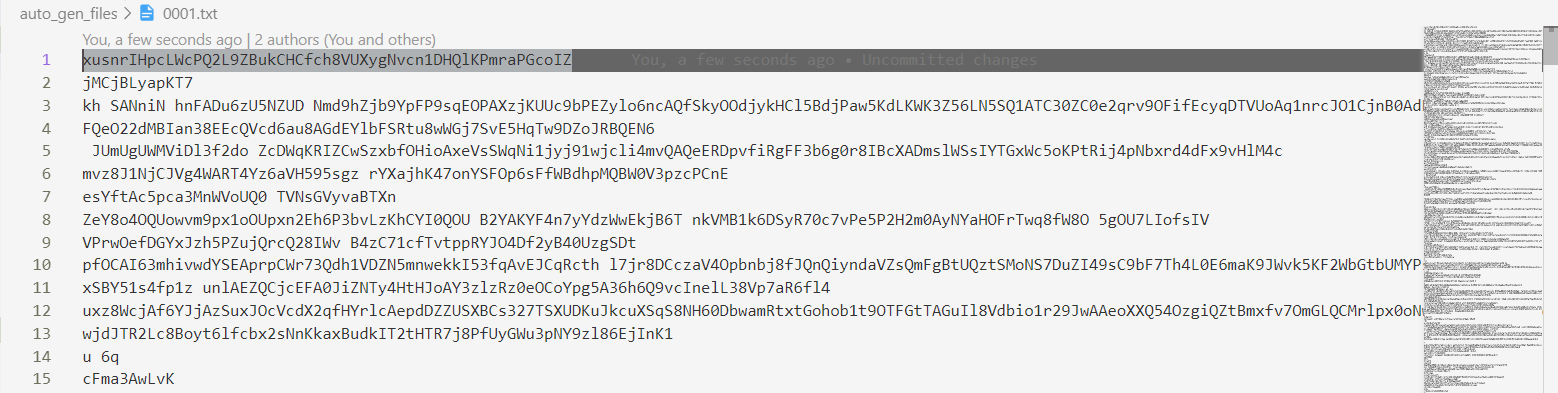
## Development of the Application

Complete Source Code of the Program is attached in Appendix A

The approach to the problem is to have to 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.







### 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.

### 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.

## 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.

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

# Question 2

Solution to Question 2

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

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

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

# 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 <iostream>

#include <map>

#include <memory>

#include <numeric>

#include <vector>

#include <chrono>

*// 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) {

**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 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);

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

    } else {*// SEQUENCIAL*

        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));

        }

    }

*// var 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;

**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;

}

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;

    bool isclosed = true;

*//  std::map<char, int> ccount;*

   public:

**file**() {

        file\_name = "";

        fd = 0;

    };

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

**~file**();

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

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

    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*

*\*/*

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

    this->**open**(this->file\_name);

}

*/\*\**

*\* Destructor*

*\*/*

**bromine**::**file**::**~file**() {

    if (!isclosed) {

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

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

    }

    this->isclosed = true;

}

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);

    }

}

**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) {

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

        ccount[buffer[0]]++;

    }

    return ccount;

}

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

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

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

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

}

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

    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*

**threader.cpp**

#include "threader.hpp"

#include <pthread.h>

#include <iostream>

#include <map>

#include <memory>

#include "file\_ops.hpp"

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

**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;

}

**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;

}

# Bibliography