RAMAIAH UNIVERSITY OF APPLIED SCIENCES

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

List of Figures

1 Question 1

Solution to Question No. 1

1.1 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.

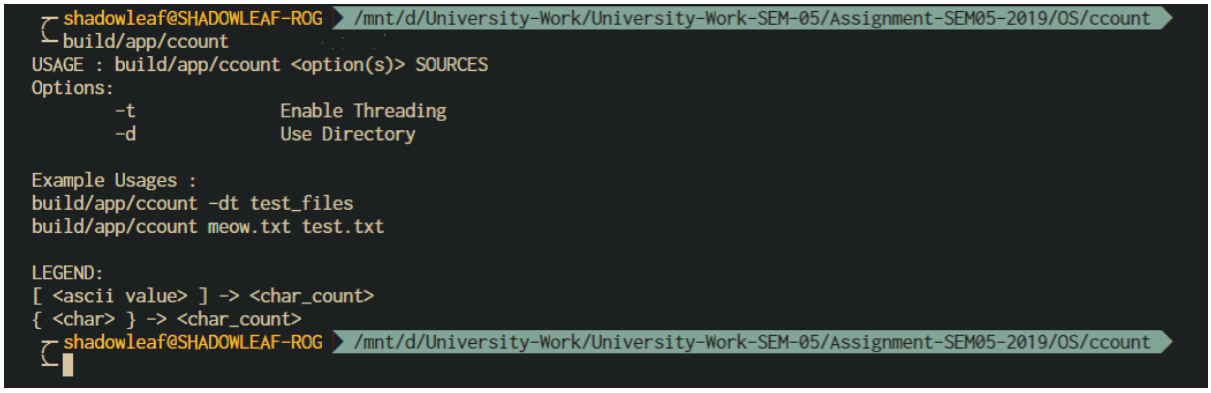

Figure 1-1 ccount help

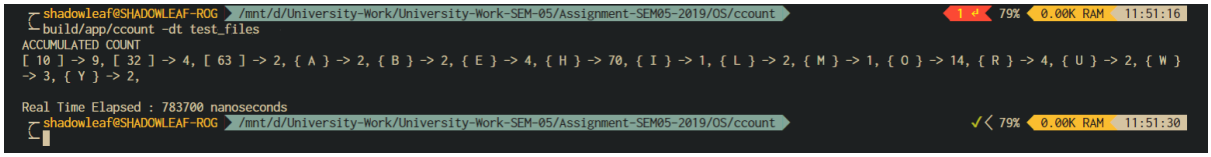

Figure 1-2 ccount sample output

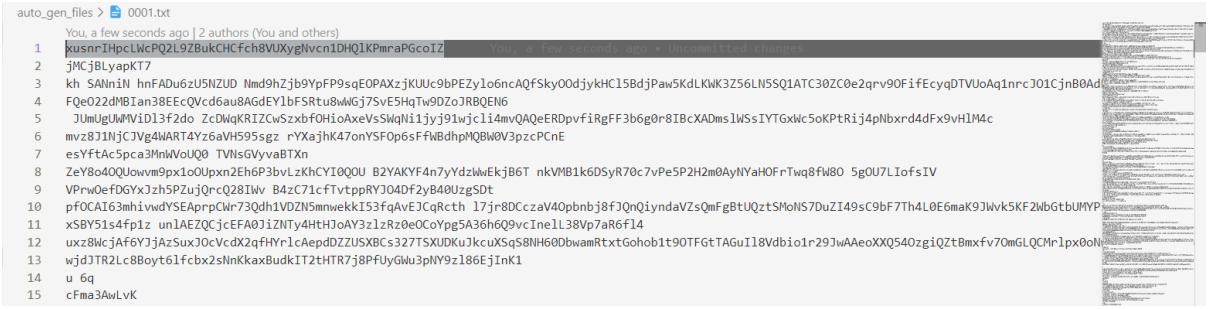

Figure 1-3 ccount - sample test file

1.1.1 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.

1.1.2 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(&bro
mine::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 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.

1.2 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
```

1.2.1 Analysis of the performance statistics

Table 1 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.

2 Question 2

Solution to Question 2

2.1 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
```

2. 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
```

3. 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
```

2.2 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
```

2.3 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

```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_pt
r<bromine::file> p) {
            return static_cast<void*>(p.get());
        });

        // generate and run threads
        std::vector<pthread_t> threads = bromine::threader::gen_worker_threads(&bro
mine::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 << " nanosecond
s" << 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
```

```hpp
#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

```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)(vo
id*), 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> th
reads) {
    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