PDC - Lab 4

Aadhitya Swarnesh I

[redacted]

8 - August - 2020

Question 1 :

Use OpenMP to implement a producer-consumer program in which some of the threads are producers and others are consumers. The producers read text from a collection of files, one per producer. They insert lines of text into a single shared queue. The consumers take the lines of text and tokenize them. Tokens are “words”.

```
#include<omp.h>
#include<stdio.h>
#include<stdlib.h>
#include<dirent.h>
#include<string.h>
#include <unistd.h>

#define MAX_FILE_COUNT 100
#define MAX_FILE_NAME_LENGTH 50
#define MAX_SENTENCE_COUNT 999
#define MAX_SENTENCE_LENGTH 500
#define CONSUMER_COUNT 2

int main()
{
```

```
    // Get all the names of the files in the corpus
directory.
    struct dirent *de;
    DIR *dir = opendir("./corpus/");
    char *file_names[MAX_FILE_COUNT];
    int file_count = 0;
    while ((de = readdir(dir)) != NULL)
    {
        //printf("%s\n", de->d_name);
        if (file_count > 1)
        {
            file_names[file_count-2] = de->d_name;
        }
        file_count ++;
    }
    file_count -= 2;
    printf("There is a total of %d files to be read.\n",
file_count);
    closedir(dir);

    /* Use parallel programming paradgims to make the
producer to scan text from the files
          and place them in sentence array, while the
consumers tokenize them.  */

    // Ask for threads one for each producer and
CONSUMER_COUNT threads for consumers.
    omp_set_num_threads(file_count + CONSUMER_COUNT);
    char sentences[MAX_SENTENCE_COUNT]
[MAX_SENTENCE_LENGTH];
    int front=0, back=0;
    int production_over = 0;       // To indicate the
completion of production.
```

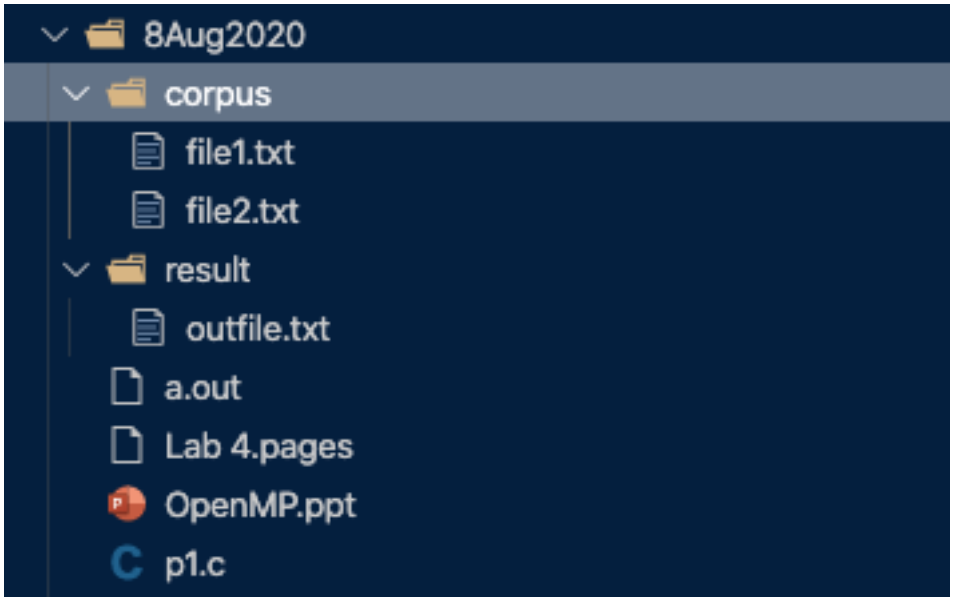
```
    // Create an output file containing the tokenized
words.
    FILE *output_file;
    output_file = fopen("result/outfile.txt", "w");

    #pragma omp parallel shared(sentences, front, back,
production_over, output_file)
    {
        int num_threads = omp_get_num_threads();
        if (num_threads >= (file_count + CONSUMER_COUNT))
{
            // We have enough threads and hence we can
continue.
            int thread_num = omp_get_thread_num();
            if (thread_num < file_count) {
                // Producer Threads
                // Read file i for thread i
                char *temp = "corpus/";
                char cur_file_name[MAX_FILE_NAME_LENGTH];
                strcat(cur_file_name, temp);
                strcat(cur_file_name,
file_names[thread_num]);
                FILE *filePointer;
                filePointer = fopen(cur_file_name, "r");
                char cur_sentence[MAX_SENTENCE_LENGTH];
                while(fgets(cur_sentence,
MAX_SENTENCE_LENGTH, filePointer) != NULL) {
                    strtok(cur_sentence, "\n");
                    #pragma omp critical(crit)
                    {
                        strcpy(sentences[back++],
cur_sentence);
                        printf("Thread num : %d --
Reading %s\n", omp_get_thread_num(), sentences[back-1]);
```

```
                    }
                    sleep(1);
                }
                fclose(filePointer);
                production_over ++;
                printf("Thread num : %d -- Completed
Reading.\n", omp_get_thread_num());
            }
            else {
                // Consumer Threads
                #pragma omp single
                {
                    int num_consumer_threads =
omp_get_num_threads() - file_count;
                }
                while ((front<back) ||
(production_over<file_count))
                {
                    if (front == back) {
                        sleep(1);
                    }
                    else {
                        char
cur_sentence[MAX_SENTENCE_LENGTH];
                        // Take a sentence from the
queue.
                        #pragma omp critical(crit)
                        {
                            strcpy(cur_sentence,
sentences[front++]);
                            printf("Thread num : %d --
Tokenizing %s\n", omp_get_thread_num(), cur_sentence);
                        }
                        if (strlen(cur_sentence) > 0) {
```

```
                            // Tokenize the sentence.
                            char
tokenized[MAX_SENTENCE_LENGTH];
                            char *token =
strtok(cur_sentence, " ");
                            while(token != NULL) {
                                strcat(tokenized, token);
                                strcat(tokenized, "\n");
                                token = strtok(NULL, "
");
                            }
                            // Store the tokenized words
in the output file
                            #pragma omp
critical(output_critical)
                            {
                                fputs(tokenized,
output_file);
                                //printf("Thread num : %d
-- Printing %s\n", omp_get_thread_num(), tokenized);
                            }
                            strcpy(tokenized, "");
                        }
                    }
                    sleep(1);
                }
            }
        }
        else {
            // We do not have enough threads.
            // As an expanded version, we can come up
with methods to handle this situation in a better manner.
            #pragma omp single
            {
```

```
                printf("Not Enough threads are avalible.
\n");
            }
        }
        #pragma omp barrier
    }
    fclose(output_file);
    printf("All the text has been tokenized
successfully!!\n");
}
```

```
(base) Aadhityas-MacBook-Air:8Aug2020 aadhitya$ gcc-10 -fopenmp p1.c
(base) Aadhityas-MacBook-Air:8Aug2020 aadhitya$ ./a.out
There is a total of 2 files to be read.
Thread num : 0 -- Reading i am fine
Thread num : 1 -- Reading hi how are you
Thread num : 0 -- Reading same to you.
Thread num : 1 -- Reading are you fine
Thread num : 0 -- Completed Reading.
Thread num : 1 -- Completed Reading.
Thread num : 2 -- Tokenizing i am fine
Thread num : 3 -- Tokenizing hi how are you
Thread num : 3 -- Tokenizing same to you.
Thread num : 2 -- Tokenizing are you fine
All the text has been tokenized successfully!!
(base) Aadhityas-MacBook-Air:8Aug2020 aadhitya$
```



Question 2 :

A search engine can be implemented using a farm of servers; each contains a subset of data that can be searched. Assume that this farm server has a single front-end that interacts with clients who submit queries. Implement the above server form using master-worker pattern.

```
#include<omp.h>
#include<stdio.h>
#include<stdlib.h>
#include<dirent.h>
#include<string.h>
#include<unistd.h>
#include<stdbool.h>

#define MAX_FILE_COUNT 10
#define MAX_FILE_NAME_LENGTH 20
#define MAX_WORD_LENGTH 20
#define MAX_NUM_THREADS 4
#define MAX_SENTENCE_LENGTH 100

int main()
{
    // Get all the names of the files in the corpus
directory.
    struct dirent *de;
    DIR *dir = opendir("./corpus/");
    char *file_names[MAX_FILE_COUNT];
    int file_count = 0;
    while ((de = readdir(dir)) != NULL)
    {
        //printf("%s\n", de->d_name);
        if (file_count > 1)
        {
            file_names[file_count-2] = de->d_name;
```

```
        }
        file_count ++;
    }
    file_count -= 2;
    printf("----> There is a total of %d files to be
read.\n", file_count);
    closedir(dir);

    char word[MAX_WORD_LENGTH];
    printf("Enter the word to be searched : ");
    scanf("%s", word);

    int work[MAX_NUM_THREADS][100];
    int work_count[MAX_NUM_THREADS];
    memset(work, 0, sizeof(work));
    memset(work_count, 0, sizeof(work_count));

    bool flag = false;

    #pragma omp parallel shared(work, work_count, flag)
    {
        #pragma omp master
        {
            //Allocate work to other worker threads.
            int num_threads = omp_get_num_threads();
            printf("----> Number of avalible threads :
%d.\n", num_threads);
            int thread_num = 1;
            for (int i=0; i<file_count; i++) {
                work[thread_num][work_count[thread_num]+
+] = i;
                thread_num ++;
                if (thread_num == num_threads-1) {
                    thread_num = 1;
```

```
                }
            }
        }
        #pragma omp barrier
        int thread_num = omp_get_thread_num();
        if (thread_num > 0) {
            printf("----> Thread : %d has %d files to
read.\n", thread_num, work_count[thread_num]);
            for (int i=0; i<work_count[thread_num]; i++)
{
                printf("----> Thread : %d is reading file
: %s\n", thread_num, file_names[work[thread_num][i]]);

                // Read contents from file and search
                FILE *filePointer;
                char *temp = "corpus/";
                char cur_file_name[MAX_FILE_NAME_LENGTH];
                strcpy(cur_file_name, temp);
                strcat(cur_file_name,
file_names[work[thread_num][i]]);
                filePointer = fopen(cur_file_name, "r");
                char cur_sentence[MAX_SENTENCE_LENGTH];
                while(fgets(cur_sentence,
MAX_SENTENCE_LENGTH, filePointer) != NULL) {
                    if (strstr(cur_sentence, word)!=
NULL) {
                        printf("A match has been found in
%s.\n", file_names[work[thread_num][i]]);
                        flag = true;
                        break;
                    }
                    strcpy(cur_sentence, "");
                }
                fclose(filePointer);
```

```
            }
        }
    }
    if (flag == false) {
        printf("There was no match found for the word in
any of the servers.\n");
    }
}
```

```
(base) Aadhityas-MacBook-Air:8Aug2020 aadhitya$ gcc-10 -fopenmp p2.c
(base) Aadhityas-MacBook-Air:8Aug2020 aadhitya$ ./a.out
----> There is a total of 3 files to be read.
Enter the word to be searched : hi
----> Number of avalible threads : 4.
----> Thread : 1 has 2 files to read.
----> Thread : 1 is reading file : file2.txt
----> Thread : 2 has 1 files to read.
----> Thread : 2 is reading file : file3.txt
A match has been found in file3.txt.
----> Thread : 1 is reading file : file1.txt
----> Thread : 3 has 0 files to read.
A match has been found in file1.txt.
(base) Aadhityas-MacBook-Air:8Aug2020 aadhitya$
```