Peter G. Mavronicolas
Old Dominion University
CS 571, Spring 2023
Instructor: Jay Morris

CS 571 Project

```
#include <iostream>
#include <thread>
#include <vector>
#include <mutex>
#include <cstdlib>
#include <ctime>
#include <chrono>
#include <fstream>
```

Table 1.1 Displaying libraries imported into the project. Note that <ctime> was not used.

Introduction

Initially, I performed online research of producers and consumers utilizing semaphores and mutex to manage thread traffic. During my research, the following libraries shown above were used. Input/Output streams were displayed with <iostream> and <fstream> was used to output data to files. The <vector> library was used to store data from *struct BufferItems* and <mutex> was used to prevent and allow entry of queued tasks into the thread. <cstdlib> is used for memory management and general-purpose functions built into C++. Although I imported both <ctime> and <chrono> together, I finally decided on using <chrono> to measure elapsed time and set delays following mutex unlocking.

Summary of Findings

After running simulations with the generated data for Producers(p) and Consumers (c), using (9) combinations of 2,5,10, I generated the following results which were derived from the *global.txt* file within this submission. Local.txt was also created and displayed data pertaining to the consumer. Overall, *Aggregate Sales* ($) remained relatively consistent within a range of $545,000- $554,000 +- 500.Total Time was measured in seconds and remained constant through each simulation at 1.68091e+09.

	Run1	Run2	Run3	Run4	Run5	Run6	Run7	Run8	Run9
P	2	2	2	5	5	5	10	10	10
C	2	5	10	2	5	10	2	5	10
Aggregate Sales ($)	$553471	$553318	$553318	$546353	$545432	$548195	$531616	$531442	$531442
Total Time (seconds)	1.68091e+09	1.68091e+09	1.68091e+09	1.68091e+09	1.68091e+09	1.68091e+09	1.68091e+09	1.68091e+09	1.68091e+09

Figure 1.1 Displaying (9) simulations (Columns, Row 1-9) with every pair of combinations for the numbers 2,5,10 representing Producers(P) and Consumers(C) shown as rows. Aggregate Sales ($) and Total Time (seconds) are also shown as rows.

Shared Variables

Below are sections of code from the source code that identify the initialization and declaration of shared variables.

```
///Shared Variables

int year = 16;
int aggregateSales = 0;
int itemsGenerated= 0;
int consumerID = 0;

//synchronization to lock and unlock thread
mutex m;



//arrays
int storeWideTotalSales[10]={};
int monthWiseTotalSales[12]={};

//time variables to calculate elapsed time
chrono::time_point<chrono::system_clock> startTime; // start time
chrono::time_point<chrono::system_clock> endTime; // end time

//Struct for Buffer Item
    struct BufferItems{
    int randomDays, randomMonth, year, storeID, registerNum;
    float salesAmt;
    };

//vectors
vector<BufferItems> buffer; //vector to store struct Buffer Items
```

Semaphores

Parts of the code containing semaphores referenced movement into and out of the thread. The following code snippets are found in the main function.

```
//semaphore threads
vector<thread> producers; // vector to store producers
vector<thread> consumers; // vector to store consumers


   /// Creating Semaphore for Producers
   for (int i = 1; i <= p; i++){
       producers.push_back(thread(producer, i + 1));
   }

   /// Creating Semaphore for Consumers
   for (int i = 1; i <= c; i++){
       consumers.push_back(thread(consumer, i + 1));
   }
```

This code snippet below is called in the *void producer (int storeID)* function.

```
// created a sales record to place back into the shared buffer
buffer.push_back(record);
```

Finally, the last code snippet is found in the void consumer(int consumerID) function.

```
if (buffer.size() > 0) {
    // Taking record from buffer
    BufferItems record = buffer[buffer.size() - 1];
    buffer.pop_back();
```

Conclusion

Coding with semaphores and mutex locks can be quite challenging but once you familiarize yourself with the functions and their importance in monitoring the thread, you quickly realize how important they can be. I was reminded how grand central dispatch, one of the most important topics in Swift coding challenges, relates to the use of semaphores and mutex locks when regulating the main thread. I found this project to be a thorough refresher for the C++ language and look forwarding to applying my skills in future job interviews.

Works Cited

1) *Chegg.com*. Chegg Study Questions and Answers | Chegg.com. (n.d.). Retrieved April 7, 2023, from https://ww.chegg.com/homework-help/questions-and-answers
2) WonderCsabo , doynaxdoynax 4, Phil H, Nelspike, Zeta, & madxmadx 6. (1960, September 1). *Generating a random date*. Stack Overflow. Retrieved April 7, 2023, from https://stackoverflow.com/questions/20271163/generating-a-random-date
3) Soni, V. (2021, May 9). *Different ways to initialize an array in C++*. OpenGenus IQ: Computing Expertise & Legacy. Retrieved April 7, 2023, from https://iq.opengenus.org/different-ways-to-initialize-array-in-cpp/
4) *C++ DO/while loop*. C++ Do While Loop. (n.d.). Retrieved April 7, 2023, from https//www.w3schools.com/cpp/cpp_do_while_loop.asp
5) GeeksforGeeks. (2023, January 20). *Chrono in C++*. GeeksforGeeks. Retrieved April 7, 2023, from https://www.geeksforgeeks.org/chrono-in-c/
6) Ahmad, Bertrand, Shital, , ranuranu, & vveil. (1956, December 1). *How to append text to a text file in C++?* Stack Overflow. Retrieved April 7, 2023, from https://stackoverflow.com/questions/2393345/how-to-append-text-to-a-text-file-in-c
7) C++ Pointers. (n.d.). Retrieved April 7, 2023, from https://www.w3schools.com/cpp/cpp_pointers.asp
8) Austin. (2015, March 19). *Semaphores C++11 when multithreading*. Austin G. Walters. Retrieved April 7, 2023, from https://austingwalters.com/multithreading-semaphores/