ID1217 Homework 1

Lucas Villarroel

February 2021

1 Introduction

This report presents the solution of two different problems having to do with critical sections, locks, barriers and condition variables. All solutions are solved in C using the pthread-library.

2 Problems

2.1 Problem 1: Sum, Min and Max of Matrix Elements

For this assignment a parallel program is given, this program uses a variable number of workers which are given a strip of the matrix. Each worker calculates the sum of their respective strip before reaching a barrier, once all workers are done with their strip, worker(0) will sum up the strip sums which gives the total sum of the matrix elements.

2.1.1 Min and max

The first part of this problem is to extend the program to also determine the maximum and minimum elements of the matrix. The solution is similar to the previous implementation. Let the workers calculate a sum, min, and max local to their strip. Then let worker(0) compare the other workers results to determine the actual sum, min and max.

2.1.2 Without barrier

Instead of using the barrier, three shared variables are introduced: sum, min and max. To prevent race conditions these variables will have to be protected using mutual exclusion. The workers still calculate their local sum, minimum and maximum. But instead of using a barrier, we let the workers access and if needed also update the shared variables. The workers will do this by taking a mutex lock.

2.1.3 Bag of tasks

Instead of letting workers compute a predetermined strip size of the matrix, each row of the matrix is represented as a task. The workers will then pick the next task in the bag. This will be determined by yet another shared variable which is a counter used to delegate tasks to the workers. Once the counter reaches the number of rows in the matrix, the bag is empty, there are no more tasks.

```
while (row < size) {
    /* take row lock and increment if there are tasks left */
    pthread_mutex_lock(&rowLock);
    if (row < size) {
      first = row;
      row++;
      last = row;
    } else {
      pthread_mutex_unlock(&rowLock);
      break;
    }
    pthread_mutex_unlock(&rowLock);

    // calculate local sum, min max
    ...
}
```

Figure 1: Bag of tasks implementation.

2.2 Problem 2: Diff

The task is to develop a parallel multi-threaded program which compares corresponding lines of two text files and prints both lines to standard output if they are different. If one file is longer, the extra lines are also printed.

This is a classic producer-consumer problem. There will be two producers reading from two different file and placing lines in two different buffers, the consumer will then read these lines, compare them and print if they are different.

The solution uses two circular buffers to accomplish this. There are two shared variables indicating the buffer sizes, in other words, how many elements in the buffer are not yet consumed. These must be protected through mutual exclusion, therefore a lock is used when accessing these variables. The producers will increment their respective size variables, while the consumer will read from both buffers and decrement the size values.

The producers will also need to stop putting elements into the buffer if the buffers become full. To accomplish this a condition variable is used. If the buffer a producer is writing to is full, the producer will wait on a condition which the consumer will broadcast once it has consumed and decremented the buffer sizes.

The consumer will also need to suspend on a condition, if one or more buffers are empty with more lines to be read. In this case, when the producers add to the buffers, they will send a signal on this condition, letting the consumer know it can now continue.

To keep track of when the producers were done, a counter was used. The counter value represents the number of producers which are finished placing elements into their respective buffers.