Critical Sections

Report I
Concurrent Programming ID1217

Emil Ståhl
Student ID: 970410

Concurrent Programming ID1217
Report I

Emil Ståhl

January 29, 2020

1 Introduction

This report covers the implementation of a number of programs written in C that utilizes parallel execution with multiple threads. A thread is a subset of a process. Each process can have multiple threads executing concurrently that share all segments except the stack. Due to this, threads require mutually exclusive operations (mutexes) as well as conditional variables that will be discussed later in this report. The main topic covered in this work is to get a deeper understanding of the problems that a multithreaded program results in and how they can be solved.

2 The programs and its purposes

The work consists of three different multithreaded programs written in C. Below is a description of each program and its purposes.

2.1 Matrixsum

This program initializes a matrix of given size with random values and returns the total sum of all elements as well as the minimum and maximum element found in the matrix. The program will split up the matrix into different strips. Since the program uses multiple threads each thread will be given one or more strips of the matrix to work on. Each thread will store its result in a global struct. When all threads are done with its task the result are printed by the main method.

2.2 Parallel Quicksort

Quicksort is a divide-and-conquer sorting algorithm. It works by selecting a 'pivot' element from the array and partitioning the other elements into two sub-arrays, according to whether they are less than or greater than the pivot. The sub-arrays are then sorted recursively. This implementation utilizes recursive parallelism which means that the sub-arrays are sorted concurrently.

2.3 The Unix tee command

The tee command reads the standard input and writes it to both the standard output and to the file filename. This implementation of the tee command does the work in three different threads. One thread for reading standard input, one for writing to standard output, and one for writing to the specified file.

3 Main problems and solutions

Below is a description of the different problems that each implementation resulted in and how they were solved.

3.1 Matrix sum

Since the program utilizes multiple threads that all manipulates the same global struct there will therefore arise race conditions. To solve this the implementation is extended with four different mutual exclusion locks which are;

```
pthread_mutex_t minLock; // mutex lock to update the minimum element
pthread_mutex_t maxLock; // mutex lock to update the maximum element
pthread_mutex_t rowLock; // mutex lock to update the current row
pthread_mutex_t totalLock; // mutex lock to update the total sum
```

Before a thread enters the critical sections that manipulates the content of the global struct it needs to take a lock and unlock it when done. If the lock is not free at the moment the thread tries to take it the thread will sleep until the lock is free. Furthermore, since the main thread are the one that prints the results it needs to be suspended until the other threads has terminated. This is done by calling `pthread_join()`. The `pthread_join()` function shall suspend execution of the calling thread until the target thread terminates, unless the target thread has already terminated.

3.2 Parallell Quicksort

As stated above, this implementation utilizes recursive parallelism i.e. the subarrays are sorted concurrently in separate threads. However, since each thread only manipulates its own partition of the array there is no risk that race conditions will occur which means that mutual exclusion locks are not needed. The `pthread_join()` function is still required in order to wait for the threads to terminate before printing.

The most challenging part of this implementation was to get the recursive parallelism to work as intended. Another problem that required some effort was how to pass the arguments to the threads since the. `pthread_join()` only takes one argument. Therefore a struct was created that holds the low and high values of the array.

3.3 The Unix tee command

The main problems of this was that nothing was printed or written when utilizing multiple threads. The reason why this problem occurs is because that the printing and writing threads start their execution before the reading thread has done its work and initialized the global char array buffer with the content from standard input.

The solution to this is to extend the program with a conditional variable. The printing and writing thread is in this way suspended on a condition until the reading thread broadcasts that it has done its work.

4 Evaluation

A test was performed for each program:

4.1 Matrix sum

```
emilstahl$ ./maxmin 10 5 100
[ 5  42  73  79  44  91  43  29  64  79 ]
[ 94  58  82  47  95  32  84  5  54  90 ]
[ 76  39  96  1  11  70  35  34  5  4 ]
[ 33  76  76  83  76  17  52  6  66  81 ]
[ 41  35  60  1  50  94  42  0  58  90 ]
[ 81  47  59  51  38  73  33  93  96  55 ]
[ 34  69  10  57  37  8  31  32  45  35 ]
[ 55  28  17  63  21  85  17  89  52  9 ]
[ 49  83  95  27  46  41  55  78  71  96 ]
[ 58  0  41  54  99  82  13  75  68  81 ]
The execution took 0.24 ms to complete
The total sum of all the elements is 5230
The maximum element is 99 at position [5,10]
The minimum element is 0 at position [8,5]
```

4.2 Parallel Quicksort

```
emilstahl$ ./quicksort 20 1000
Initializing array with 20 elements between 0 and 1000
> [48 792 667 163 863 334 135 308 548 269 42 133 861 882 747 275 58 632 749 479]
Sorting array...
Sorted in 2.805000 ms
Sorted array:
> [42 48 58 133 135 163 269 275 308 334 479 548 632 667 747 749 792 861 863 882]
```

4.3 The Unix tee command

```
emilstahl$ ./tee file1.txt newfile.txt
Attempting to write to file...

"This shall be written to STDOUT and the file"

Write to file newfile.txt succeded
Emils-MBP-15:3. tee emilstahl$
```

5 Conclusions

This work has focused on implementing mutex locks and conditional variables in a number of multithreaded programs. The main topics covered was to better understand the advantages and problems that a multithreaded system results in.