Mert Kelkit - 150115013
Furkan Nakıp - 150115032
29 December 2018

CSE 3033 Project 3 Report

Matrix and Queue Manipulations With Multithreading

In this project, our main objective is to synchronize 4 type of threads with matrix and queue operations.

We have 4 types of threads:

1- Generate Threads:

These threads are responsible for generating and enqueueing to queue #1 5x5 sub matrices filled with random values between 1 and RANDOM_LIM (100 in our case).

2- Log Threads:

Log threads are responsible for appending 5x5 sub matrices created by generate threads. Main purpose is to read queue #1 and insert sub matrix to the big matrix correctly .

3- Mod Threads:

Mod threads also read from queue #1, get first element of the sub matrix, take modulo of each element in this sub matrix. Then create new sub matrix with calculated values. These threads enqueue created matrices to the queue #2.

4- Add Threads:

Add threads read from queue #2, sum all elements in the sub matrix contained in queue #2. After that, update a global sum value with local sum calculated before.

Also there is a main thread which is responsible for creating threads and waiting for all threads to terminate.

This program expects arguments from user with that format:

./program -d [one dimension size of big matrix] -n [num of generate threads] [num of log threads] [num of mod threads] [num of add threads]

If there will be any illegal argument, our program will print error and exits the program. Also size of the big matrix must be a multiply of 5, because we are combining 5x5 sub matrices to create a big matrix. If we say size of big matrix is N, there must be (N/5) x (N/5) sub matrices.

Explaining Mutexes, Condition Variables and Semaphores That We Used:

Here is the list of synchronization tools that we used:

```
/* Required mutex and semaphore variables - Described in project report */
pthread_mutex_t generate_mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t log_mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t mod_mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t add_mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t checker_mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t empty_generate_mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t empty_mod_mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t printf_mutex = PTHREAD_MUTEX_INITIALIZER;

pthread_cond_t empty_generate_queue = PTHREAD_COND_INITIALIZER;
pthread_cond_t empty_mod_queue = PTHREAD_COND_INITIALIZER;
pthread_cond_t dequeue_cond = PTHREAD_COND_INITIALIZER;

sem_t peek_semaphore;
```

Generate Mutex: Used in generate thread routine while updating global current sub matrix counter, enqueueing to queue #1 which is global.

Log Mutex: Used in log thread routine while updating global log counter, peeking the global queue #1 (getting values stored inside of head of the queue #1).

Mod Mutex: Used in mod thread routine while updating global mod counter, peeking the global queue #1 (getting values stored inside of head of the queue #1). After that, this mutex is also used for enqueue part of mod threads; incrementing current mod matrix counter and enqueue operation for queue #2.

Add Mutex: Used in add thread routine while updating the global add counter, dequeueing from the queue #2.

Checker Mutex: This mutex is used in peek function for queue #1. In our program, mod and log threads should read from queue #1 concurrently. After one mod thread and one log thread read from queue #1, there must be a dequeue operation for queue #1. We used that mutex for checking both mod and log thread read from the queue #1. If not, thread that arrives first waits for the other one. After that, both threads will exit from the peek function after dequeueing. This mutex is used together with condition variable dequeue_cond.

Our usage of checker mutex goes below:

```
// Increment checkers field of the of queue
pthread_mutex_lock(&checker_mutex);
submatrix_queue->checkers++;

// If two threads checked and red data from the head
if(submatrix_queue->checkers == 2) {
    // Print info
    pthread_mutex_lock(&printf_mutex);
    printf("%s thread red from queue. dequeueing...\n", caller);
    pthread_mutex_unlock(&printf_mutex);
    // And current head is ready to be dequeued
    dequeue_generate();
}
// If only one thread red from the queue, wait for other one to read
else {
    // Print info
    pthread_mutex_lock(&printf_mutex);
    printf("%s thread red from queue. waiting for dequeue...\n", caller);
    pthread_mutex_unlock(&printf_mutex);
    // Wait until other thread read head
    pthread_cond_wait(&dequeue_cond, &checker_mutex);
    // After other thread reads, it'll dequeue
    // And we can continue...
}
pthread_mutex_unlock(&checker_mutex);
```

Empty Generate/Mod Mutex: These mutexes are used in enqueue operations for queues queue #1 and queue #2. They are used together with condition variables empty_generate/mod_queue. These mutexes are used for checking there will be any enqueue operations or not to an empty queue.

Printf Mutex: This mutex is used for printing jobs of threads properly. Acquired before and released after printf functions inside threads that print information.

Empty Generate/Mod Queue Condition: These are used in threads that require reading queue #1 and queue #2. If any queue is empty while a thread is trying to read the that queue, these condition variables will be in wait status if there will be further enqueue operations to the queue.

Dequeue Condition: This condition variable used for determining dequeue operation. As described above, we accept 2 threads inside of the peek function. After these 2 threads read their data, we need to do dequeue operation. If one thread arrives and reads its data before the other thread, it should wait other thread to read its data too. While waiting, this condition variable will be signaled from dequeue function.

Peek Semaphore: This semaphore used for letting 2 threads coming inside the peek function for queue #1 since 2 types of threads need to reach queue #1 concurrently. Because of locks inside log and mod thread routines, one of the 2 threads arrive peek function must be a mod thread, other one must be a log thread.

Sample Execution:

We executed our program with these arguments in order to put them into report simply:

```
Program arguments:  -d 15 -n 9 8 9 8
```

A part of output of the program goes below:

```
Add Thread 2 has local sum 725 by [1, 0] submatrix. Global sum before/after update: 2452/3177.
Add Thread 7 has local sum 607 by [0, 2] submatrix. Global sum before/after update: 1845/3177.
Mod Thread 5 created this matrix from submatrix [1, 0]:
[[  0   68  10  21  32  ]
 [  68  0   1   65  20  ]
 [  10  35  17  4   26  ]
 [  18  52  30  13  61  ]
 [  30  21  42  61  20  ]]
Mod Thread 3 created this matrix from submatrix [2, 2]:
[[  0   3   4   1   4   ]
 [  1   3   1   4   2   ]
 [  5   1   0   0   4   ]
 [  3   2   2   4   3   ]
 [  4   3   2   3   3   ]]
Mod Thread 0 created this matrix from submatrix [2, 0]:
[[  0   2   6   5   3   ]
 [  0   3   3   0   5   ]
 [  8   5   3   2   5   ]
 [  2   6   6   2   2   ]
 [  3   6   6   2   4   ]]
Add Thread 2 has local sum 1093 by [1, 2] submatrix. Global sum before/after update: 3266/4421.
Mod Thread 7 created this matrix from submatrix [1, 2]:
[[  0   46  35  52  0   ]
 [  62  49  22  70  18  ]
 [  62  28  9   11  71  ]
 [  30  51  85  85  29  ]
 [  39  54  56  84  45  ]]
Add Thread 3 has local sum 62 by [2, 2] submatrix. Global sum before/after update: 4359/4421.
Add Thread 1 has local sum 89 by [2, 0] submatrix. Global sum before/after update: 3177/4421.
Global sum: 4421
Process finished with exit code 0
```

Inside of output file:

```
The big matrix is:
[[  69  92  38  71  89  59  20  81  25  6   70  83  13  71  17  ]
 [  15  82  66  19  45  37  72  25  25  97  24  53  7   86  10  ]
 [  51  77  19  20  76  81  69  20  43  69  37  94  18  20  23  ]
 [  23  95  8   73  62  10  91  76  3   98  36  85  68  46  86  ]
 [  36  31  88  80  48  88  9   47  3   55  98  66  13  77  36  ]
 [  69  68  10  90  32  57  88  48  91  54  90  46  35  52  90  ]
 [  68  69  1   65  20  28  71  66  91  12  62  49  22  70  18  ]
 [  79  35  17  4   26  1   12  57  64  80  62  28  99  11  71  ]
 [  87  52  99  82  61  94  89  90  22  17  30  51  85  85  29  ]
 [  99  90  42  61  20  68  31  96  23  21  39  54  56  84  45  ]
 [  9   74  69  41  30  8   18  100 55  13  6   93  88  49  10  ]
 [  36  12  57  9   86  53  6   36  33  23  49  75  67  64  38  ]
 [  53  50  93  65  50  45  42  7   55  61  83  97  90  96  52  ]
 [  92  33  42  65  65  49  28  2   44  98  99  14  26  46  75  ]
 [  30  33  6   20  49  17  90  84  4   63  10  69  8   39  9   ]]
Global sum is 4421
```

Resources:

[1] http://akademik.marmara.edu.tr/zuhal.altuntas

[2] https://stackoverflow.com/

[3] https://pubs.opengroup.org/

[4] http://linux.die.net/