

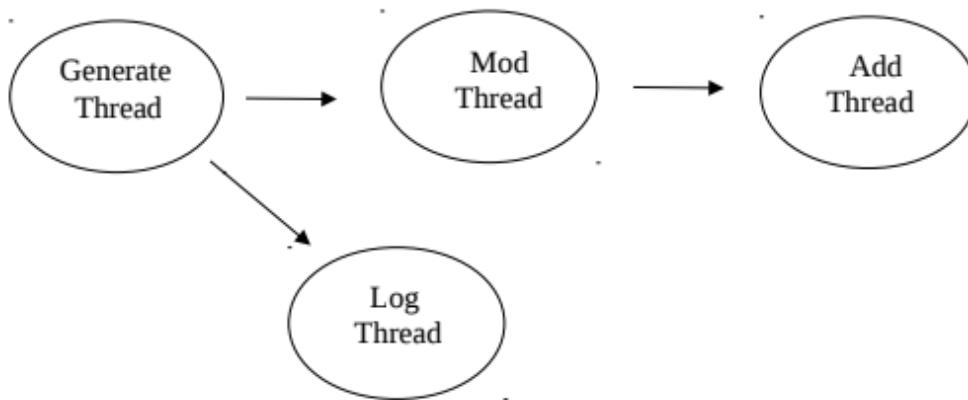
# OPERATING SYSTEMS PROJECT #3 REPORT

## PROJECT DESCRIPTION

Project's real aim is to create a C program that will create a matrix which is built with 5x5 sub matrices. These sub matrices will be computed randomly on different threads (Generate Threads). Size of the matrix has been given in the argument list (N).

After the creation of these sub matrices that as a whole describes the matrix, they will be pushed into 2 different threads that one of them will try to build the whole matrix (Log Threads) and another will try to get the modulo of each sub-matrix element with their 0th indexes element (Mod Threads).

Also there is an another thread that will get the output of the modulo threads and sum the sub matrix and update a global sum (Add Threads).



## USAGE

**gcc -g main.c -o main.o -lpthread:**  
Compiling

**./main.o -d 30 -n 15 5 8 6:**

*With -d option represents the size of the matrix ( 30x30 ),  
-n option represents the number of threads for each type of thread  
( Generate, Mod, Log and Add Threads)*

```
((base) Onur-MacBook-Pro:ParallelMatrixCreation olmaditekrar$ time ./main.o -d 5 -n 15 5 8 6
===== Program is starting =====

-----
Generate_127684608: |   Generate_127684608 generated following matrix:
36 03 98 28 20
59 42 97 84 55
13 17 94 29 77
09 17 35 59 04
30 79 91 64 26
This matrix is [0,0] submatrix
-----

Log_127148032 |   Log_127148032 just grabbed the [0,0] submatrix
-----

Mod_130367488 |   Mod_130367488 generated following matrix:
00 03 26 28 20
23 06 25 12 19
13 17 22 29 05
09 17 35 23 04
30 07 19 28 26
This matrix is generated by [0,0] submatrix
-----

Add_134123520 |   has local sum: 446 by [0,0] submatrix, global sum before/after update: 0/446
-----

=== Printing the LOG Matrix ===
36 03 98 28 20
59 42 97 84 55
13 17 94 29 77
09 17 35 59 04
30 79 91 64 26

real    0m0.008s
user    0m0.003s
sys     0m0.005s
```

## THREAD FUNCTIONS

### Generate Threads

Each Generate Thread is responsible of the submatrix creations. These submatrices will be  $(N/5) * (N/5)$  times. Each sub matrix has a unique index that will be used in the later on when creating the global matrix. It will represent row and column indexes to put the current submatrix in.

After the creation of the threads, we are initializing them by sending every thread to the `generate_thread_func`.

This function will get the first index that it can take ( we are using a mutex to make sure only one thread can get slot or index for the creation) and generate a sub matrix with the size of 5x5. Lastly it will increment the `last_generated_submatrix_index` variable to give other threads a starting point when they want to get an index to create sub matrix on.

After that it will queue the `generate_threads_queue` with that sub matrix.

Function has also error preventions inside it, it is checking two times if the total sub matrix number has reached or not by `last_generated_submatrix_index`. Because other threads can update the `last_generated_submatrix_index` value once we stepped out the mutex locks.

Also the first Generate Thread that enters the mutex lock to get slots will create the Queue.

### Log Threads

Each Log Thread is responsible for the creation of the big matrix that builded up with the sub matrixes that comes from Generate Threads.

It will check if the upper limit (Size of the Generate Queue) for the index is exceeded. If so, it will exit the current thread.

Each thread try to lock the slot-taking procedure and get the next available slot for themselves. When they get it they will increment the last logged submatrix index variable.

Every node in the Generate Queue has an index attached to them. As we described above, it is showing the where to put the matrix in the global matrix. So a Log Thread will have to compute the row and column starting point with the help of the functions for them.

After getting a slot and corresponding index and matrix for that slot, they will update the `log_thread_matrix` (global matrix).

`log_thread_matrix` is a matrix that we will output to a file (output.txt) later.

First log thread will create the global matrix.

### Mod Threads

Each Mod Thread is responsible for the creation of a queue that will get inputs from Generate Queue. Mod Threads will try to compute every element's modulo by the current matrix's first element. They are creating another Queue for as input to Add Threads.

As other thread functions they will check upper limits once that are executed. If so, they will decrement the last\_mod\_submatrix\_index and exit the current thread. And try to get a slot from the Generate Queue for sub matrix to be able to make computations on.

If it is the first Mod Thread that is executing, it will create the Mod Thread's Queue.

They will get the index of the current node, for the printing later on as problem statement expects.

These functions uses GenerateModMatrix function as a helper function. GenerateModMatrix function gets a QueueMatrix as input and outputs a mod matrix. It will do all the mod computations necessary.

Once the mod matrix is ready, we will enqueue the mod matrix to the Mod Threads Queue.

## Add Threads

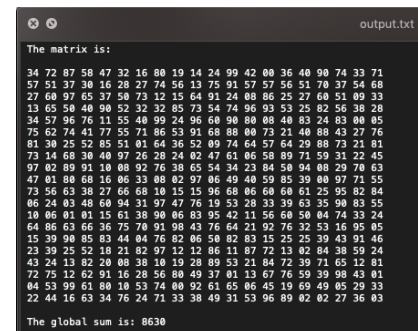
Each Add Thread is responsible for the contributing the global sum of the sub matrixes that is being creating and updated from the Mod Threads.

They will check if the upper limit (Last mod sub matrix index) is exceeded or not, if so they will decrement the last\_summed\_submatrix\_index.

As other thread functions, they will try to get a slot from the Mod Queue and update a local sum. We created a local sum because we want to find out old and new values for the global\_sum variable and it is making sure multiple Add Threads can work concurrently on different indexes.

## IMPLEMENTATION DETAILS

- We used Array Queue to establish a queue structure that we are controlling with the rear and front variables as well as capacity variable.
- We used another structure called QueueMatrix to hold a matrix and an index to itself. It is quite useful when we try to find out the real index that sub matrix corresponds to.
- In code, there are two implementations of the matrix. First one is matrix[5][5] and the second one is \*\*matrix. Each of them has purposes to use. We create different enqueue and printing functions for each of them.
- We used a printer mutex to make sure only one thread at the time can write into the stdout. You can't expect to get expected output when you are using threads.
- Output of the Log and Add Threads is written in a file called "output.txt"
- After program execution, we are freeing all the malloc'd variables for secure environment.



```
The matrix is:
34 72 87 58 47 32 16 80 19 14 24 99 42 00 36 40 90 74 33 71
57 51 37 30 16 28 27 74 56 13 75 91 57 57 56 51 70 37 54 68
27 68 97 65 37 58 73 22 15 64 91 24 88 86 25 27 68 51 09 33
13 65 58 48 98 52 32 32 85 73 54 74 96 93 53 25 82 56 38 28
34 57 96 76 11 55 48 99 24 96 68 98 88 08 48 83 24 83 00 05
75 62 74 41 77 55 71 86 53 91 68 88 88 73 21 48 88 43 27 76
81 38 25 52 85 51 01 64 36 52 89 74 64 57 64 29 88 73 21 81
73 14 68 38 48 97 26 28 24 02 47 61 86 58 89 71 59 31 22 45
97 82 89 91 10 08 92 76 38 65 54 34 23 84 58 94 88 29 78 63
47 81 88 68 16 86 33 88 82 97 86 49 48 59 85 39 88 97 71 55
73 56 63 38 27 68 68 18 15 96 68 86 68 88 61 25 95 82 84
86 24 83 48 68 94 31 97 47 76 19 53 28 33 39 63 35 98 83 55
10 06 01 81 15 61 38 98 06 83 95 42 11 56 68 58 04 74 33 24
64 86 63 66 36 75 70 91 88 43 76 64 21 92 76 32 53 16 95 85
15 39 98 85 83 44 84 76 82 86 58 82 83 15 25 35 39 43 91 46
23 39 25 52 18 21 82 97 12 12 86 11 87 72 13 82 84 38 59 24
43 24 13 82 20 08 38 10 19 28 89 53 21 84 72 39 71 65 12 81
72 75 12 62 91 18 28 56 88 49 37 81 13 67 76 59 39 98 43 81
84 53 99 61 88 19 53 74 88 92 61 65 86 45 19 69 49 85 29 33
22 44 16 63 34 76 24 71 33 38 49 31 53 96 89 82 82 27 36 03

The global sum is: 8630
```