**Operating Systems**

**PROJECT 3**



**BACHELOR'S DEGREE IN COMPUTER SCIENCE AND ENGINEERING**

**Year 2018/2019**

David Rico Menéndez 100384036

Javier Ruiz Plaza 100383436

# INDEX

[**INDEX**](#_gjdgxs) **1**

[**Code Description**](#_pxy6adxvtodq) **3**

[Declaration of global variables for concurrency\_layer.c](#_zciacrwjgz8f) 3

[init\_concurrency\_mechanisms()](#_gjk2vslcltup) 3

[destroy\_concurrency\_mechanisms()](#_gjk2vslcltup) 3

[broker(void\* args)](#_gjk2vslcltup) 3

[operation\_executer(void \* args)](#_gjk2vslcltup) 4

[stats\_reader(void \*args)](#_gjk2vslcltup) 4

[checkFlag(int\* flagDir, pthread\_mutex\_t exit mutex)](#_chxlgi6jq2hj) 4

[main()](#_wed4gkyjkp3j) 5

[**Testing**](#_dwx7k3ien7dc) **6**

[**Conclusion**](#_fahpbfhlxgrx) **8**

[Problems encountered:](#_tyjcwt) 8

[Concurrency](#_gx7jcsnuuwuz) 8

[Multiple brokers and batch files duplications](#_y3x05s953hjs) 8

[Exit mutex](#_fbbp8letmd3s) 8

[Empty batch file](#_gnsday30r5ko) 8

# 

# 

# 

# Code Description

## Declaration of global variables for concurrency\_layer.c

In this file, we are using 2 different mutexes (1 for Producer-Consumer operations, and 1 more for Writing-Reading control flow),also 2 conditional variables for the Producer-Consumer operations in this case for the emptiness and fullness of the queue,

each of them is declared in the first lines. Also, we declare an int (flagReaders) that would be used for controlling the reader operations.

## init\_concurrency\_mechanisms()

This method will be on charge of initializing all the mutexes for his correct use. To do so, we make sure also that, in case of error, notice it to the user.

The mutex exit\_mutex is NOT initialized at this method. This is because, since it is created in the main, and we are not able to pass arguments to the current method, plus it is used as an argument from main function at concurrent\_market.c, we decided to do mutex\_init at the file referred before.

## destroy\_concurrency\_mechanisms()

This method, as the name said, takes all the existent mutexes and destroy them to save memory at the end of the processes.

exit\_mutex is also destroyed where created and initialized, at concurrent\_market.c

## broker(void\* args)

Each broker acts as a producer in the producer consumer operations.

In this method, each of the n brokers generates and iterator after reading the batch file.

This batch file can be the same, as in our implementation each broker can receive a different batch file as long as it’s specified in the correspondent structure.Then, we get into a while loop that will continue if there are pending operations left. At this point, next step is to create a new operation, stored at op1. Then, we will lock the queue since we are about to start doing some enqueue operations, but first we check if it is full, in that case the broker will wait for a signal indicating that the queue is not full anymore, this signal is sended from operation executor after processing an operation. Once all its ok, the broker will get the operation into the queue and leave unlocking the queue mutex.

## operation\_executer(void \* args)

The operation executor acts as the consumer in the producer consumer operations.

This method will gather the parameters from the arguments after that it will go on a for loop as long as the flag is not active, when the flag gets activated it will stop when the queue is empty. Inside the while it will block the queue and then check if the queue is empty or not, in case it is it will wait for a signal about the emptiness of the queue, this signal is sended by the broker when an operation is put in the queue. After that it will dequeue an operation and process it. Finally it unblocks the queue mutex .

## stats\_reader(void \*args)

The stats reader will read will save the arguments it receives before going into the loop that controls it. In the loop that will be going on as long as the flag is not active, the thread will block the queue print the market status and unblock the queue, after that the thread will go to sleep for a determined amount of time until the next iteration of the loop.

## checkFlag(int\* flagDir, pthread\_mutex\_t exit mutex)

This is an auxiliary function that will check the status of the flag, it is used in the conditional of the while loops.

It will receive the direction of the flag in memory and the exit mutex. This function will simulate the protection for the writers-readers, there can be more than 1 reader at the time, and when so happens the value of flag cannot be changed.

At first it will block a readers mutex in , inside the critical section it will increase the number of readers, in case the readers are one the exit mutex will be blocked and then it will unblock the readers mutex. After that it will read the flag status and save its value in a variable. Finally will activate the readers mutex again and decrease the number of readers, in case the readers are zero at that instance it will unblock the exit mutex and unblock the readers. At the end it will return the value read from the flag.

Readers mutex is necessary so no more than 1 reader can increment the number of readers at the same time nor block the exit mutex.

## 

## main()

In the main execution of the program first we establish the number of brokers and readers that will be executed we left by default 1 broker and no reader.

After that we create the threads necessary for the execution, initialize the market and the concurrency mechanism and create the flag variable and mutex.

Then the structures for the creation of the threads are initialized and the correspondent thread is created in a for loop.

When the execution of all the brokers are finished and waited for by the join mechanism the exit variable will be activated provoking the rest of the threads to end, these will also be collected by join.

At the end in order to free resources the market and concurrent mechanisms will be destroyed before exiting.

# 

# Testing

|  |  |  |  |
| --- | --- | --- | --- |
| TEST | DESCRIPTION | EXPECTED | WORKS? |
| Test with 1 broker and a exec operator | Basic test | Broker and operator work concurrently but only one can operate the queue at the same time | OK |
| Test with 3 brokers and a exec operator | Various brokers in the system | Each broker inserts into the queue the specified batch file and only one can access the queue | OK |
| Test with 500 brokers and a exec operator | What happens when there is a large amount of brokers | Each broker inserts into the queue the specified batch file and only one can access the queue,takes a large amount of time | OK |
| Test with 0 brokers and a exec operator | What if there is no brokers | After market is created it will finish immediately | OK |
| Test with 1 broker and a exec , large batch file(over 1000 operations) | What happens when the batch file is large | Execution very similar to 500 brokers and a small batch file(due to our implementation) | OK |
| Test with 1 broker , a exec operator and batch file is empty | What if the batch file is empty | Execution ends as if there is no brokers | NOT OK  Executions waits for input in the terminal |
| Test with 1 broker,a exev operator and 1 reader | Test the readers | All works ok | OK |
| Test with 1 broker,a exev operator and 50 readers | Test with a large amount of readers | Executions can be very long as readers also block the queue for reading, depends on the priority of the processes | OK |
| Test with 1 broker and a exec operator , market file is empty, batch file exists | Test what happens if there is no market | At the time exec operator tries to process the queue error about the operations not being found will display | OK |
| Test with 2 brokers each one with a different file | Test with 2 different batch files | Each broker will implement in the queue it's correspondent batch operations | OK |

## 

## 

# Conclusion

## Problems encountered:

### Concurrency

At the beginning we developed all the functions taking into account the concurrency, to have an overview of how to plan and put mutexes and rest of tools we had to use. Later on we have encountered problems with our concurrency, but after many reviews we realised that the concurrency is no more than a consumer-producer and readers-writers problem.

### Multiple brokers and batch files duplications

When there are more than 1 broker and receives the same batch file as input our implementation will put in the queue a duplicate of the batch file. We don't know if what it’s intended is that the brokers reading the same file will only input the operations once.

### Exit mutex

The exit mutex is completely implemented in the main function, as it has to be passed as a parameter in the structures for the operation exec and the readers. We tried to implement it so it’s initialized in the init\_concurrent\_mechanism() function, didnt work so well so we left it in the main.

### Empty batch file

When an empty batch file is used the system after initializing the market will stop and wait for user input. We don’t know what was causing it, we looked deeply at stringcopy and memcpy but didn't seem to be the problem. We looked everywhere but did not encounter the problem with that one.

## 