#include "../include/concurrency\_layer.h"

#include <pthread.h>

#include <unistd.h>

#include <stdio.h>

#include <stdlib.h>

pthread\_mutex\_t enqueue\_mutex;

pthread\_cond\_t not\_full;

pthread\_cond\_t not\_empty;

//Initialize the concurrency mechanisms

void init\_concurrency\_mechanisms(){

pthread\_mutex\_init(&enqueue\_mutex,NULL);

pthread\_cond\_init(&not\_full, NULL);

pthread\_cond\_init(&not\_empty, NULL);

}

//Destroy the concurrency mechanisms

void destroy\_concurrency\_mechanisms(){

pthread\_mutex\_destroy(&enqueue\_mutex);

pthread\_cond\_destroy(&not\_full);

pthread\_cond\_destroy(&not\_empty);

}

//Checks a condition variable that needs to be locked for checking

int check\_exit(pthread\_mutex\_t \*mutex, int \*exit) {

pthread\_mutex\_lock(mutex);

int ret;

if (\*exit == 0) {

pthread\_mutex\_unlock(mutex);

return ret = 1;

}

else {

pthread\_mutex\_unlock(mutex);

return ret = 0;

}

}

//The broker will work whenever the operation\_executer is not working thanks to the producer-consumer mechanism

void\* broker(void \* args){

//Extracting information from the argument

broker\_info \*info\_b1 = args;

char batch\_oper[256];

strcpy (batch\_oper, info\_b1->batch\_file);

stock\_market \* my\_market = info\_b1->market;

operations\_queue \*q = my\_market->stock\_operations;

//Creating iterator and some structures to add transactions to the queue later

iterator \*it = new\_iterator(batch\_oper);

stock new\_stock;

operation new\_op;

//While loop that only affects when there are operations to read

while((next\_operation(it, new\_stock.id, &new\_op.type, &new\_stock.total\_shares, &new\_stock.total\_value)) > -1){

//A new operation is created

new\_operation(&new\_op, new\_stock.id, new\_op.type, new\_stock.total\_shares, new\_stock.total\_value);

//Producer-Consumer concurrency mechanism

pthread\_mutex\_lock(&enqueue\_mutex);

//Execute a while lopp for waiting until we can introduce new operations in the queue

while(operations\_queue\_full(q)){

pthread\_cond\_wait(&not\_full, &enqueue\_mutex);

}

//Adds the operation previously created to the queue

enqueue\_operation(q, &new\_op);

//Send a signal to say that there are operations in the queue

pthread\_cond\_signal(&not\_empty);

pthread\_mutex\_unlock(&enqueue\_mutex);

}

//Iterator destroyed

destroy\_iterator(it);

}

//The operation\_executer will work whenever the broker is not working thanks to the producer-consumer mechanism

void\* operation\_executer(void \* args){

//Obtaining the values from the argument

exec\_info \*info\_ex1 = args;

stock\_market \*market = info\_ex1->market;

pthread\_mutex\_t \*exit\_mutex = info\_ex1->exit\_mutex;

operations\_queue \*q = market->stock\_operations;

operation \*op = q->operations;

//Locking mutex to read the variable exit

pthread\_mutex\_lock(exit\_mutex);

int \*exit=info\_ex1->exit;

//Unlocking the mutex, as the variable has been already read

pthread\_mutex\_unlock(exit\_mutex);

while(check\_exit(exit\_mutex, exit)){

//Producer-Consumer concurrency mechanism

pthread\_mutex\_lock(&enqueue\_mutex);

//Execute a while lopp for waiting until we can pull new operations out of the queue

while(operations\_queue\_empty(q)){

pthread\_cond\_wait(&not\_empty, &enqueue\_mutex);

}

//Dequeuing the operation and proccesing

dequeue\_operation(q, op);

//Send a signal to say that more operations can be added to the queue

pthread\_cond\_signal(&not\_full);

//Processing the operation pulled from the queue

process\_operation(market, op);

op = malloc(sizeof(operation));

pthread\_mutex\_unlock(&enqueue\_mutex);

}

//If exit signal has appeared we execute all pending operations

while(!operations\_queue\_empty(q)){

pthread\_mutex\_lock(&enqueue\_mutex);

//Dequeuing the operation and proccesing

dequeue\_operation(q, op);

//Processing the operation pulled from the queue

process\_operation(market, op);

op = malloc(sizeof(operation));

pthread\_mutex\_unlock(&enqueue\_mutex);

}

}

void\* stats\_reader(void \* args){

//Obtaining the values from the argument

reader\_info \*info\_re1 = args;

stock\_market \*market = info\_re1->market;

pthread\_mutex\_t \*exit\_mutex = info\_re1->exit\_mutex;

unsigned int frequency = info\_re1->frequency;

//Locking mutex to read the variable exit

pthread\_mutex\_lock(exit\_mutex);

int \*exit=info\_re1->exit;

//Unlocking the mutex, as the variable has been already read

pthread\_mutex\_unlock(exit\_mutex);

while(check\_exit(exit\_mutex, exit)){

//Printing the status of the market while taking care there is no update of the market in that moment

pthread\_mutex\_lock(&enqueue\_mutex);

print\_market\_status(market);

pthread\_mutex\_unlock(&enqueue\_mutex);

usleep(frequency);

}

}