CPSC 410

Project 4

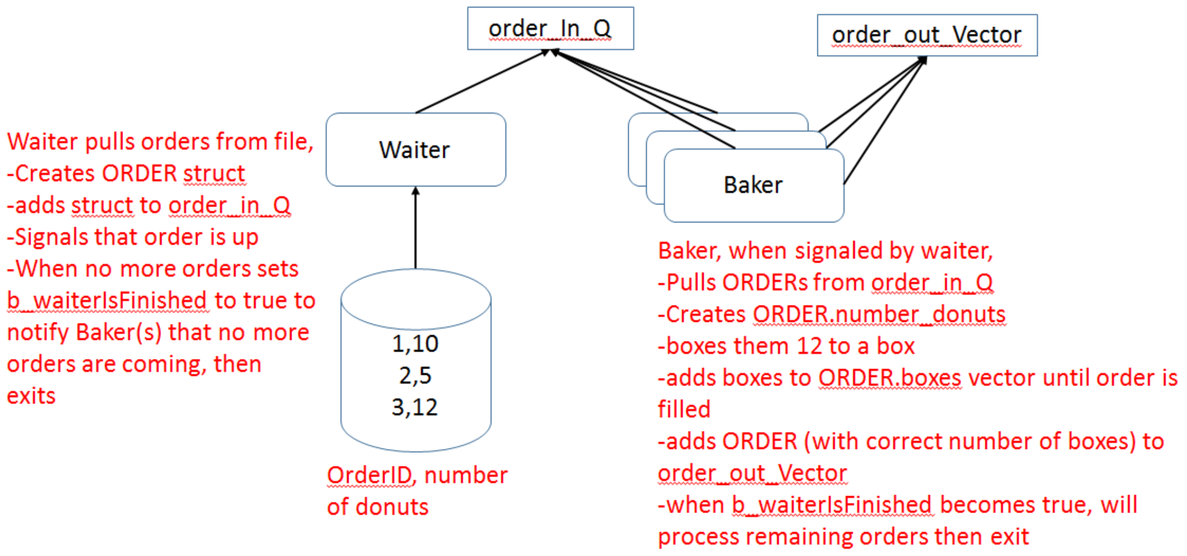
Motivation: Topics covered by this project;

- Threads, mutexes, condition variables

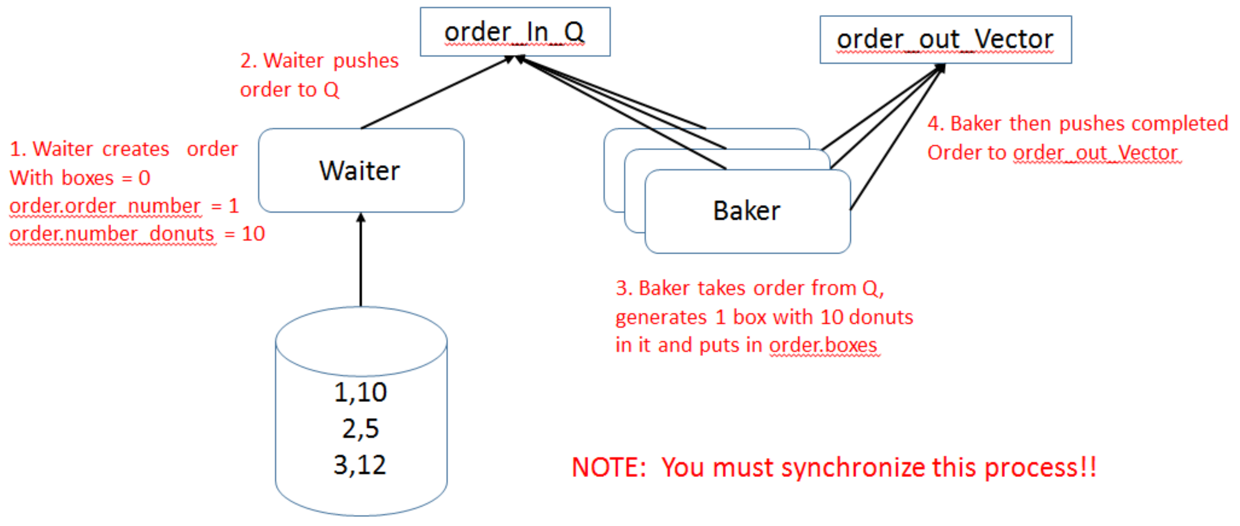
Overview

The best ideas often occur while your daydreaming, where you wandering mind connects disparate bits of information in a way that produces something new and innovative.   This is not one of those.

You are to model a donut order fulfillment system. A waiter takes orders for donuts, while bakers fill and box these orders (see below).


order_In_Q
order_out_Vector
Waiter
Baker
1,10
2,5
3,12
Waiter pulls orders from file,
-Creates ORDER struct
-adds struct to order_in_Q
-Signals that order is up
-When no more orders sets b_waiterIsFinished to true to notify Baker(s) that no more orders are coming, then exits
OrderID, number of donuts
Baker, when signaled by waiter,
-Pulls ORDERs from order_in_Q
-Creates ORDER.number_donuts
-boxes them 12 to a box
-adds boxes to ORDER.boxes vector until order is filled
-adds ORDER (with correct number of boxes) to order_out_Vector
-when b_waiterIsFinished becomes true, will process remaining orders then exit

As an example, suppose the Waiter is pulling in order 1, for 10 donuts (see above). The process will follow this outline


order_In_Q
order_out_Vector
2. Waiter pushes order to Q
1. Waiter creates order With boxes = 0 order.order_number = 1 order.number_donuts = 10
Waiter
Baker
4. Baker then pushes completed Order to order_out_Vector
3. Baker takes order from Q, generates 1 box with 10 donuts in it and puts in order.boxes
1,10
2,5
3,12
NOTE: You must synchronize this process!!

File_io (all is given to you)

This set of functions is similar to what you did in project 2. Note that it contains orders now.
You are given a file that has an unknown number of rows, and 2 columns of integers. Assume there are no malformed rows (ie != 2 columns). It looks like the following (mine will be much more elaborate).

```
1,10
2,5
3,3
5,15
10,11
```

This file can have any name, but I will call it testdata.txt for the purposes of this document. The first column is order_number, second is number_donuts . File_IO provides functions to load this data into a vector and sort it by order_number.

Baker

You are given a header file and a stripped .cpp file. Please fill in the cpp file. Please see header file for function explanation.
Additional notes: Baker must not start processing orders until they are available. Do this by waiting for a signal from condition variable cv_order_inQ (defined in 410_proj4.cpp and externed in externs.h). Baker will also wait on b_WaiterIsFinished, when this happens no more orders are coming.

Waiter
You are given a header file and a stripped .cpp file. Please fill in the cpp file. Please see header file for function explanation
Additional notes: You must use condition variable cv_order_inQ (defined in 410_proj4.cpp and externed in externs.h) to signal the baker that there are orders available in the order_in_Q or to signal that there are no more orders available (b_WaiterIsFinished)

410_proj4.cpp
I have given you a stripped down implementation. Use this file to launch a thread for the waiter and multiple bakers. There is a useful function audit_results() that should help you determine whether orders were filled properly.

Other Information
Box.cpp and .h, used by baker when boxing donuts
Datastructs.h, holds ORDER info
PRINT.h contains helpful threadsafe print functions that you can use.

Externs.h contains external references to global variables, include wherever you need those global variables.

> (extern is a promise to the compiler. It states that an externed variable is not defined in this file, but is in another. When all object files linked (remember compilation is preprocess->compile->link) the linker will find it. It's the only way to work with globals that must be shared amongst different compilation units without passing them as function references.)

Documentation and Testing
Make sure you comment each function and the program as a whole. I recommend you test each module separately (which implies you write test code to verify its behavior).

Directory Structure
Please do not modify the project directory structure

To Turn In
waiter.cpp, baker.cpp ONLY!

Grading
I will drop your waiter.cpp and baker.cpp into the appropriate places in my project. Therefore these are the only files that I need from you.

20% waiter.cpp and baker.cpp files submitted as described in 'ToTurn In'
25% condition variables correctly used for signaling between baker and waiter
30% 1 waiter and 1 baker work correctly in separate threads
25% 1 waiter and multiple bakers work correctly

Other Hints

I will test with many bakers so use condition variable `notify_all()`.