Jeff DiPaola

CS 214

11/26/14

Assignment 5 – Overview

I would like to begin by calling your attention to some of the assumptions I made throughout the development of this assignment:

- **Input files will be of the same (correct) format** – I am assuming that the input files passed to the argument will always be formatted correctly and contain the proper information. That being said I do check to be sure that they are valid files, that they are present, and that the user ID number as well as the category are valid.
- **The Category file will always have three categories in it** – I assumed that the category file will always be formatted the same (one category per line) and that there will always be three categories. In a real implementation of this program categories wouldn't constantly be changing; there would really only be the categories that are set up in the database. This dependency on the three categories can be changed if it were required later, however, at this point *the category file must contain 3 categories or the program will give you an error and will close*. This is done because a lot of other factors in the program depend on how many categories are there. The category names, however, can change.
- **Each queue can hold 5 orders** – I am unsure what the specifications of the iLab machines are, however, each queue is currently set up to hold 5 orders. With a total of 3 queues that's a total of 15 orders that could possibly be stored in queues at any point in time. This did not yield any errors while testing so I am assuming this doesn't exceed the memory of the iLab machines.

The program itself is developed into three different C files: "reader.c", "books.c", and "books.h". Reader takes care of the actual reading into memory of the categories and customer databases. This was taken from the search assignment and modified slightly to work with this one. Books.h takes care of all the header work for books.c including all the global variable declarations, includes, and the declaration of our book order structure. Finally, books.c does all the heavy lifting for our program creating the producer and consumers and actually completing the assignment requirements.

In order for our producer and consumer system to work appropriately there were a few things that had to happen to be sure that orders weren't getting lost. To begin, a mutex had to be placed around the queue holding the orders (called "orders"). This prevents an order being taken from a queue by a consumer while the producer is placing. Additionally, I started with the consumer only checking if there were orders in its queue to decide if it should remain active or not. This also led to a loss in orders because the queue would get emptied while the producer was still running! To correct this I also created a "running" Boolean that is set to true while the producer

is running and becomes false once it has completed. This also requires a mutex around it to be sure that a consumer doesn't get stuck thinking the producer is running when it's actually not.

I also found that, in some cases, the purchased books for a customer would be different. I attributed this behavior to the nature of the producer and consumer. The producer is dropping orders into different queues all for the same user. This then can end with the orders getting processed in a different sequence depending on which consumer is running when on that particular execution of the code. For that reason, sometimes customers will have different orders on different runs.

Another aspect of my program that I noticed was that the way I coded it has the orders coming out in the opposite order they did in the given example output. This is just due to the nature of how I had set up my linked list and has no effect on the actual program itself. Since the program instructions did not specify the order that the successful and unsuccessful orders must be in, I left it as is.