pa5

Multithreaded Book Order System

David Awad

Mariam Tsilosani

Design

The Book Order System was built as a solution to the producer consumer problem. In which we can get large amounts of input but only given a small, fixed space to handle it.

One way to handle this problem is to make multiple threads that handle the different inputs all at once so that we don't have to store the input for any longer than just to read the information we need.

So we have a producer thread and various consumer threads that are waiting for the producer's output so it can be processed.

Running the program will result in a colored terminal output that will show the producer and consumers *interleaving* and trading their access to this shared data structure. As the program runs the successful orders happen the consmer threads will log them to the user.

Implementation

The key to this problem is that the number of categories is *crucial* to the success of the program.

We have to keep track of a lot of different data for both the customers and the orders.

```
typedef struct order{
    char *bookTitle;
    double price;
    char *category;
    double remain;
    int custID;
    char processed;
} order;

typedef struct orderNode{
    order *order;
    struct orderNode *next;
} orderNode;
```

```
typedef struct category{
   char* name;
   orderNode* list;
   pthread_mutex_t mutex;
}category;

typedef struct customer{
   char *name;
   int custID;
   double credit;
   char *address;
   char *state;
   char *zip;
} customer;

typedef struct custNode{
   customer *customer;
   orderNode *failedOrders;
   orderNode *successOrders;
   struct custNode *next;
} custNode;
```

We start by creating n threads,(where n is the number of categories) that will then wait for the producer to propagate a structure that the producer and consumer threads are sharing.

The producer starts the process by reading each line of the input order file and then creating a struct out of the tokenized input. It then locks a mutex in the row of the category for that particular category.

The consumer threads are waiting for the producer to place the struct inside of the "row" of the array of orders that the consumers are responsible for. The consumer will see the order as soon as the mutex applied to that row is unlocked.

We're left with a program that very efficiently handles space by only storing a maximum of (n) orders at a time.

Usage

The program can be run by running the command `make`.

The makefile will generate an executable called `bookOrder`.

The program requires the following format.

```
./bookOrder <database file> <orders file> <category file>
```

Our BookOrder program is very careful about the validity of the input as well as it's thread management. The final source code amounted to 573 lines.

pa5