Book Order Readme
Andrew Moore and Paul Jones

Design

Overview

Our implementation of the multithreaded book order system utlizes multiple processes[1] which communicate using shared memory. We have one consumer process per customer listed in the database file. In addition, we have only one program on disk that can act as a either producer, consumer, or overseer, and only one executable file is used.

This program is layed out with one "overseer" process to synchronize and order the operations of the producers and the consumers. The overseer will do all the input parsing, create child processes, dictate when the production/consumption is to stop/start, and close the processes, using condition flags from the child processes.

Process

1. Read input files into lists of structs for the book orders / customers.
2. Acquire shared memory segment.
3. Spawn child processes - one for producer, and a consumer for every customer in the database.
4. Wait for the children to signal "ready".
5. Signal "start"
6. Start checking for all to have signaled "done".
7. Collect and print a final report.
8. Send a "stop" signal to all children, remove shared memory chunk.

The producer, nicknamed ProducerBot, is created, signals itself "ready" (through a dedicated byte in shared memory) and then waits for the overseer to produce a "start" flag in shared memory. Then, it starts adding book order ID's one at a time to the fixed-length-queue (refer to the section on layout of shared memory) in a loop. When it is complete, it signals done to the overseer and waits for a stop signal.

Each consumer, nicknamed with their name a given in the database file, has a similar process to the producer. They signal ready, wait for start, consume, signal response, and repeat until done. Then, put current credit balance in shared memory, signal done to overseer and wait for the stop signal.

The consumption method works as follows:

1. Peek at top item in queue.
2. If it is for this customer, dequeue it.

[1] Which is extra credit!

3. If it can be bought with available credit, buy it. Replace its ID number with -1, signaling successful purchase.
4. If it cannot be bought, replace its ID with -2, signifying failed order.
5. If this is the last order or the last order has been recently consumed, break the loop and signal done.

Process Syncronization

We synchronize the actions of our processes with a simple system of flags in shared memory. These dedicated bytes allow the processes to send simple, predetermined binary messages to each other - for example, am I done yet? Should I start consuming yet? etc. The overseer process enforces the synchronization by waiting for conditions to be satisfied at certain points in the code. For example, at the end of consumption, processes will signal a "done" flag. The overseer will wait until all of the done flags are set before removing the shared memory & children so that no consumers are cut off before they are done.

Shared Memory Layout/Design

The shared memory is layed out in order as follows

1. "Start" flag
2. "Stop" flag
3. "ready" flags
4. "done" flags
5. "error" flags (unused)
6. queue start index (used for the static length queue)
7. data for the queue (1 int per order)
8. money response data
9. order response data
10. 10 bytes free space.