6: Sharing the Chores

CSCI 4547 / 6647 Systems Programming

Spring 2022

0.1 Goals for P6

1. To define a communication protocol.
2. To implement a client and server programs that implements the same logic as in P5.
3. To concurrently operate the Mom and four Kids as 5 fully separate processes..

1 Scenario

Same family, same chores. Mom has four children, Ali, Cory, Lee, and Pat. She also has an unending list of chores that need to be done this week; 10 at a time will be posted. As before, there are many kinds of jobs, and there are moody kids, and Mom pays the kids for jobs completed.

2 Changes

There is no longer a block of shared memory. Instead, all communication is done by sending data and messages through sockets.

Locks vs. Protocols. Mutex locks are no longer relevant because the mom and kids are potentially on different machines. But sockets are 2-way streets, and the messages between Mom and the kids must happen in lock-step to prevent mess-ups. For this purpose, you need to invent a protocol, or systematic sequence of messages and responses that will convey all necessary information from one to the other. Define an enum type consisting of codes for all the message codes that Mom or a Kid expects to send or receive. (ACK, NACK, QUIT, and others). Use these codes consistently in your protocols; DO NOT send strings as messages.

3 Implementing Mom

Startup. Mom should print the banner and create a job table with 10 jobs. Then she must create a welcome socket and listen for kid-contacts. When a Kid comes, Mom must "hand off" the kid to a newly created worker socket. Because Mom will be receiving protocol messages and sending entire tables of integers through this socket, open a C-style I/O stream around it. This will allow Mom to use printf(), scanf(), and write() to read and write messages.

Wait until children 0 through 3 have all arrived. When each one arrives, send it an ACK message and the child's ID# (0,1,2, or 3) . After #3 arrives, look at the clock and record the time. Quitting time is 21 seconds later.

Then enter the polling loop and keep the kids busy. When a Kid-contact comes in with the message NEED A JOB,

1. Send the entire job table (60 short integers) to a kid.
2. Accept, from the kid, the choice of a job. Respond with an ACK or NACK message according to whether that job is still available when the kid's message is received. A NACK message should be immediately followed by a copy of the current job table.

When a kid finishes a job, Mom will get a JOB DONE message from the kid. She should record the Kid's work in her tables, and look at the clock. If 3.5 or more hours (21 time periods) have elapsed, she must send the kid a "TIME TO QUIT" message.

At the end of the work time, Mom will send the "time to quit" message instead of the job list. After all four children have received the quit message, Mom calculates each child's earnings (using the vector of completed jobs) and adds $5 to the child with the greatest number of points. Print the final results. Then she terminates

4 Implementing a Kid

Kid actions Next, create a socket and try to connect. Because this Kid will be receiving entire tables of short ints through the socket, wrap a C-style I/O stream around the socket so you can use scanf and printf for messages and read() to receive the table data.

1. Wait for Mom to send you your identity, then request the job table.
2. Receive the entire job table (60 integers) and store a copy to search. Then choose a job from the table and inform Mom of the table subscript 0..9). If Mom says NACK, repeat this step.
3. Otherwise simulate work by sleeping for the required number of time periods (seconds), then send a JOB DONE message back to Mom. Receive back either a QUIT message or a new copy of the current job table,
4. If a QUIT message is received, print out the kid's entire list of jobs done and the value of that list, then quit. There is nothing more to do, so the kid can finally leave the house.

5 Other Instructions.

5.1 Types needed

Print adequate debugging output throughout.

The protocol. Define an enum type consisting of all of the message-codes that a Mom or a Kid needs to send or receive (ACK, NACK, QUIT, and others). Use enum codes, not strings, for all messages.

Jobs. A job has a unique ID (a short int, starting at 10), a field to record a kid's identity (0 through 3), and 3 short ints between 1 and 5 indicating how slow/dirty/heavy the job is, plus one for the value of the job. This definition is needed by both Mom and Kids, so include it in both server and client. In the Job class, you need a print(), a default constructor, a constructor with parameters and a setter for the kidId field. You will need to revise the code you wrote for the thread program.

5.2 Testing

Write the logic for Mom, which will run in the server process, and the logic for a Kid to run in the client processes. The server and client code must be kept separate and compiled separately. To ensure a consistent compile, you need a makefile that will compile both. Write and compile one, then write and compile the other. Then test Mom with one child and verify that the protocol works. Then test with 2 kids, then with 4.