8: Sharing the Chores

CSCI 4547 / 6647 Systems Programming

Fall 2017

1 Stages

Assignments 8 and 9 will both implement a game using concurrency, but the concurrency will be implemented in two very different ways.

P8. A server process that interacts with four client processes through sockets. All five interact through a protocol.

P9. A process with four subsidiary threads. All five interact through shared memory.

1.1 Goals for P8

1. To implement application logic for Mom and the Kids.
2. To define a communication protocol.
3. To implement a client program that implements the Kid logic.
4. To implement a server program that implements Mom's logic.
5. To concurrently operate the Mom and four Kids.

2 Scenario

The family. Mom has four children, Ali, Cory, Lee, and Pat. She also has an unending list of chores that need to be done every week. On Saturday, Mom posts the job list and everybody knows that Mom will make them keep doing chores until she signals the end of the work session. They do not know that Mom plans to make them work approximately 3.5 hours (21 time units).

Varied jobs. There are many kinds of jobs, and Mom rates each one on three scales (1..5):

1. Q: From quick to time consuming. A quick job = 1 time unit; a time consuming job = 5. Each time unit is 10 minutes,
2. P: From pleasant =1 to very dirty, unpleasant, or sweaty = 5
3. E: From easy work = 1 to heavy work = 5

Mom rewards each worker-child with points for every completed job. The number of points is: Q*(P+E). For example, Cleaning the toilet = quick, a little unpleasant, light work = 1*(4+1) = 5 points. Mowing the lawn is slow, fairly sweaty, and strenuous = 5*(4+5) = 45 points.

Mom has an infinite number of jobs of all sorts, which she posts as needed. However, no more than 10 are posted at any given time. On Saturday morning, early, she posts the initial set of jobs for the day. Each kid must ask to see the bulletin board, look at the posted jobs, and choose one according to his/her preferences. The kid then tells Mom which job was chosen. Mom will remove the chosen job from the job-board and put it in a vector of done jobs, with the kid's ID# on it. Then she will replace it in the job table by another random job.

3 Implementing a Kid

Kids are moody. From week to week, a child may have different job-preferences. Sometimes children are *lazy* and won't do heavy work. Sometimes they are *prissy* and won't do dirty work. *Overtired* children like short jobs because they can't concentrate on a job for long. A child can feel *greedy* sometimes – and go for the biggest available reward. Finally, everybody is *impatient* sometimes and just grabs the last job on the list. The kids select jobs, do them, and report back to Mom.

The first thing a child should do when it "comes alive" is to select a mood from the list of five moods (lazy, prissy, overtired, greedy, impatient). This will be used in a switch to determine which job-selection process is used:

- A lazy child chooses the easiest job.
- An overtired child chooses the shortest job.
- A prissy child chooses the cleanest job.
- A greedy child chooses the biggest payoff.
- An impatient child takes job 9 every time.

Next, create a socket and connect. Wait for Mom to send you the job table. Choose a job and tell Mom what you chose. Choose another if she tells you your first choice is taken. Then simulate doing it by sleeping for the required number of time periods (seconds). When it is done (that is, you wake up), print out the completed job and report to Mom that the job is done. Then wait for further instructions: either a fresh copy of the job table or the message that it is time to quit.

At the end of the work time, Mom will send the "time to quit" message instead of the job list. There is nothing more to do, so you can leave the house.

4 Implementing Mom

Mom should print the banner, then randomly create 10 jobs. For each, choose three slow/dirty/heavy coefficients, then call a Job constructor with 3 parameters.

Print the banner, then create a welcome socket and listen. Wait until all children (1, then 2, finally 4) have arrived. When a child arrives, send back an ACK message with the child's ID# (0,1,2, or 3) . After #3 arrives, look at the clock and record the time. Enter the polling loop and keep the kids busy. When a Kid sends a job-choice message, and the choice is good, record the Kids ID in the job and move the job to the "done" vector. When a child sends a job-done message, look at the clock. If 3.5 or more hours (21 time periods) have elapsed, send the kid a time-to-quit message. Otherwise, send back the current job table.

After all four children have received the quit message, Calculates each child's earnings (using the vector of completed jobs) and add an added $5 to the child with the greatest number of points. Print the final results. Then terminate.

Print adequate debugging output throughout.

5 Other Instructions.

5.1 Types needed

Moods In the Kid program only, you need an enumerated type that names the five moods.

Job. This is a class with 3 short ints between 1 and 5 indicating how slow/dirty/heavy the job is, plus one for the value of the job, and one for a kid's ID. This definition is needed by both Mom and Kids. You need a print(), a default constructor, a constructor with parameters and a setter for the kidId field.

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 are kept entirely separate and (potentially) run on different machines. 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.

The test plan. Make a test plan, using fake input where necessary, that calls every function at least once to ensure that the functions all work and produce the expected results. DO NOT SKIP THIS STEP.