9: Chores using Threads

CSCI 4547 / 6647 Systems Programming
Fall 2017

1 Goals

1. To simulate doing the Saturday Morning Chores, as in Program 8, but by using threads and shared memory instead of sockets.
2. To use appropriate synchronization and locking with the shared memory.
3. To use a signal to send information to a thread.

2 Instructions

The main program. Print the banner, instantiate Mom and call her run() function. Run this program several times and submit all the results, including the list of jobs done by the children, their final scores, and the winner.

2.1 Adapt Parts of Program 8

You already have a Mood enumeration; use it here also. Your Job class can be used with minor modification: Change the integer KidID to a C-string kidName and add a boolean field, `done` to represent job status (not-complete or complete). The child will record its own name in the name field when he or she selects a job and will set the `done` field to true when the job is completed.

Create a Shared class. Into this class, put all the things that must be shared:

- A table of 10 jobs.
- A mutex lock to protect the table, initially unlocked. Remember that this structure must be locked and unlocked every time it is used.
- A boolean flag, `startFlag`, initially false.

Create a Mom class.
Put parts of the code from the Server process of P8 into this class, organized thus:

- Declare an array of C-strings to store the names of your four children. Chose your own names.
- Declare an array of 4 Kids and another array of 4 tids.
- Declare a vector to store the completed jobs.
- Declare a `time_t` variable to store the time at which the chores started and another to store the current time.
- Define a default Mom constructor.
- Define a function to create and initialize the job table. To do this, you need to first lock the table, then fill it with jobs. Initialized `done=false` and `kidName=""` for each new job.
- Define a function for scanning the job table. When a finished job is found, move the Job object to the vector of completed jobs, then replace it with a new job.

Define Mom's `run()` function.

- Create the job table.
- Create the kid threads and, for each one, a Kid object.
  - Instantiate Kid k using the kth name in the array of kidNames and a pointer to the Shared object. Store the Kid in Moms array of Kids.
  - Create a thread for the kid and store its tid in the array of tids. The parameter for thread creation should be a pointer to the current Kid. Have the thread execute a global static function named doKid(), described below.
- As soon as the 4th kid is running, store the current time in your `time_t` variable and send a start signal (use SIGUSR1) to one of the kids by using `pthread_kill( tid, SIGNAME )`
- Enter a loop.
  - Sleep for one second.
  - When the signal wakes you up, look at the clock and calculate the difference between the current time and the saved time. If it is 21 or more, leave the loop.
  - Otherwise, scan the job table, save the finished jobs, and replace them in the table by new jobs.
- When you leave the loop, send four QUIT signals, one to each of the four kids. Wait for them to join you.
- When all four kids have joined, use the vector of completed jobs to compute how much each kid has earned. Award an extra $5 to the kid with the highest earnings. Print the finished-job list and the winner's name If all is working correctly, Mom's computations will agree with the Kids own computations.

The Signal handler.
In shared memory, you need a flag that means "time to start" and another that means "time to quit". These flags will be visible to all threads. .

Your signal handler must catch the signals that Mom can send, figure out which signal was sent, and set the corresponding flag to true. This makes a record that can be checked by any thread at any time to determine whether the signal was sent.

Make a Kid class.
Put parts of the code from the Client process of P8 into this class, organized thus:

- Declare a vector to store the completed jobs.
- Define a constructor that will save the pointer to the Shared object and the Kid's name, received as parameters.
- Define a function for choosing and saving the mood of the day.
- Define a function for selecting a job that fits the daily mood. Avoid selecting a job that already has another Kid's name on it. When you select a job, put your name in its name field and return.
- Define a signal handler to catch the signals that Mom will send, figure out which signal was sent, and set one of the shared flags to true. This flag will be seen by all threads and will deliver the message simultaneously to all of them.

When a Kid thread starts up, it should:

- Create a signal-processing apparatus for the two signals it expects to receive. You will need an empty signal set and a signal set containing SIGUSR1 and SIGQUIT. You will need to set up and initialize a `sigAct` object and attach it to the thread with a call on `sigaction()`.
- Use a busy wait to wait until the `startFlag` becomes true, then enter a working loop:
  - Select a job (call your function).
  - Simulate doing the job by sleeping for the required number of seconds.
  - When you wake up, if the QUIT signal has been sent, break out of the work loop.
  - If not, move the current Job object to your vector of completed jobs and repeat the loop.
- When you break out of the work loop, print a message that you got the signal, then print the contents of your job vector and the amount of money you earned, then quit. Note that you will not be paid for the final partly-done job.