Virginia Tech
Bradley Department of Electrical and Computer Engineering
ECE-3574: Applied Software Design Spring 2016

Homework 5

Submission Details

You must submit the solutions for this homework as an electronic submissions using Scholar (under ECE3574 → Assignments → Homework 5). The submission must be a gzipped tar file (.tar.gz) with your source code. Include all necessary project files, but no binary or compiled files. Your program will be run to evaluate its correctness, and the source code will be reviewed for adherence to the Qt programming style. Your program must run on Ubuntu 14.04.1 and compile/build using the GNU C/C++ compiler and the qmake/make tools. The following information must be included at the top of each of your source files as comments: your full name, your student ID number, your email address, class (ECE 3574), and the title of the assignment (Homework 4). The submitted file must be given a name in the following form: ***LAST_FIRST_hw5.tar.gz*** where LAST is your last or family name and FIRST is your first or given name. Paper, email or Drop Box submissions will not be accepted. All work must be submitted by the announced due date/time. **Late submissions will not be accepted!** (Don't do it! You have been warned!)

Questions

Use the Homework 4 forum in the Discussion Board area of the class web site to ask questions about this assignment. Do not post questions that contain specific information about the solution.

Honor Code

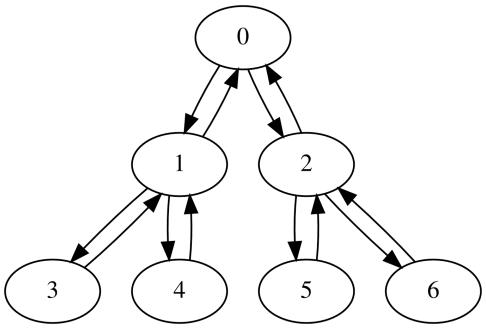
As stated in the syllabus, in working on homework and projects, discussion and cooperative learning are allowed. However, copying or otherwise using another person's detailed solutions to assigned problems is an honor code violation. See syllabus for details.

Learning Objectives

The primary learning objectives of this assignment include learning to use concurrent programming abstractions including POSIX message passing and inter-process communication. Additional learning objectives include being able to reason about concurrent execution of programs.

Specifications

1. Using POSIX message passing (message queues) write an application that communicates temperature values between various processes.
2. There will be 7 processes. Each one an instance of the same executable.
    a. You will run 7 instances of the process and they will be identified by a unique number as the first argument that will be passed in at the command prompt, numbers 0 to 6.
    b. Each process will create their own mailbox with a different number. For each process provided numbers 0 to 6 they create a mailbox /70 for 0, /71 for 1, etc.
    c. Additionally, the program will be provided with a floating point number as the second argument indicating the temperature.
    d. In the first step of this assignment the processes will create, in this order, processes 6, 5, 4, 3, 2, 1, 0 with a small delay in-between to ensure that each process's respective message queues have been created.
    e. These processes will communicate as a binary tree. While each process opens and creates its own mailbox it will communicate with 1, 2 or 3 other mailboxes. It will open the mailbox of its parent and children if they exist.
        i. Process 0 is parent to process 1 and 2
        ii. Process 1 is parent to process 3 and 4 and child to process 0
        iii. Process 2 is parent to process 5 and 6 and child to process 0
        iv. Processes 3 and 4 are children to process 1
        v. Processes 5 and 6 are children to process 2



    f. Each process will accept the following 2 command line parameters

```
./external <id> <initial-temperature>
```

    For example,

```
./external 6 100.0
./external 5  45.0
./external 4 120.0
./external 3  67.7
./external 2  20.0
./external 1  36.9
./external 0  98.6
```

3. The whole idea of the system is as follows:
    a. Processes 6 through 0 create their own mailbox and wait on their own mailbox to receive a message to start, except 0 which starts itself. You may wait on the start message to connect to the other mailboxes or count on a startup delay between the processes.
    b. After process 0 starts it should send a message indicating its current temperature.
    c. When a process receives a temperature from its parent it will calculate its value according to the following equation and transmit this number to its children
    `down_temp = (temp + parent_temp)/2.0`
    d. Processes 3, 4, 5 and 6 do not, and cannot, transmit their temperature to their children. Instead they transmit its down_temp back to their parent.
    e. When a process receives a temperature from its children it will calculate its value according to the following equation and transmit this number to its parent
    `up_temp = (temp + child_a_temp + child_b_temp)/3.0`
    f. Process 0 does not, and cannot, transmit its temperature to its parent. Instead, if it still unstable as discussed later, it will transmit its up_temp back to its children.
    g. When down_temp and up_temp are calculated it immediately becomes the node's temperature.
    h. When up_temp is calculated it must be printed to output.
4. In the case where the up_temp is less than $1/100^{th}$ degree different from the temp for process 0 it is no longer unstable and it should send a signal, which other processes must propagate, that will cause each process to print a message indicating that it's the final temperature and then exit the process.
5. When printing the calculated up_temp value it should print
    `Process #<num>: current temperature <up_temp>`
    Where <num> is the process number 0 to 6 and <up_temp> is the calculated value.
6. When printing the final temperature for each process it should print
    `Process #<num>: final temperature <temp>`
    Where <num> is the process number 0 to 6 and <temp> is the calculated value. Note the difference in the text. This is used to indicate that the exit message is received and propagated properly.

Example

Let the initial temperatures be the following:

```
Process 0 temp =   0.0
Process 1 temp = 100.0
Process 2 temp = 200.0
Process 3 temp = 300.0
Process 4 temp = 400.0
Process 5 temp = 500.0
Process 6 temp = 600.0
```

Iteration	P0	P1	P2	P3	P4	P5	P6
initial	0	100	200	300	400	500	600
1	133.333	150	250	175	225	300	350
2	176.852	161.111	236.111	158.333	183.333	245.833	270.833
3	190.072	169.599	223.765	163.657	176.157	226.157	238.657
4	194.004	176.526	215.415	171.746	177.996	216.538	222.788
5	195.166	181.801	209.694	178.506	181.631	210.624	213.749

6	195.509	185.678	205.682	183.495	185.057	206.527	208.09
7	195.61	188.488	202.833	187.044	187.825	203.561	204.343
8	195.64	190.511	200.798	189.547	189.937	201.391	201.782
9	195.649	191.964	199.342	191.311	191.506	199.805	200.001
final	195.649	191.964	199.342	191.311	191.506	199.805	200.001

Note that you may omit the output for iteration 9 as long there is text which indicates that it is printing the final temperature for a given process. Also note that only process 0 needs to have a temperature that changes no more than 0.01 degrees.

Notes

1. Standard error checking for command line arguments is required. If wrongly formatted parameters are entered, the process must print out an error message

2. External IDs in the correct range are expected. If an incorrect ID is entered on the command line, again, the program must print out an error message

3. It is an error to try to run a process with the same ID twice. You can detect this by using the IPC_EXCL (exclusive) flag when creating the mailboxes. Detect this and all other errors and properly report them.

4. There is no specific requirement in using Blocking or Non-blocking send/receive functions for the mailboxes. A non-blocking implementation will be granted with an extra-credit of 10.

5. When using message queue there are a few functions that you'll likely want to use to manipulate these queues. From the terminal you can type "man <function name>" where function name is any of the following and get documentation for any of these commands:

- mq_open
- mq_close
- mq_unlink
- mq_send
- mq_receive
- mq_getattr
- mq_setattr

  Note that the header for these functions is given in the documentation and mentions that you need to link with librt. This can be done by adding a line in your *.pro file: "LIBS += -lrt" or by adding -lrt to the gcc command line compilation tool.

Grading

Qt Style: 10%
Programs work as specified: 90%