CECS 326: Group project[1]

Objectives: Interprocess communication and basic coordination using message queue[2]

This assignment is to be completed in a group[3] of 3 or 4 students. I recommend that each group selects a leader. The requirements of the system is somewhat complex because there are several software dependencies, a great number of possible "paths of execution" and intermittent software bugs. Get started as soon as message queue is covered in lab; your group will need the entire time for: testing, redesigning & recoding. If you need assistance with the assignment, you should ask for help ASAP. If you should need assistance with a difficult member of your group, you can & should speak with me in private about issues confronting your group immediately.

Software requirements:

Implement a software system which involve 4 programs executing concurrently: ProbeA, ProbeB, ProbeC and DataHub. The Hub receives data from the 3 probes. It outputs the identity (PID) of the probe and the data that was sent from its probe. The Hub needs to execute in a manner which does not arbitrarily delays the execution of any probe software.

BEHAVIOR OF EACH PROBE

Each probe produces its own reading[4] to send to the DataHub, until the probe terminates. While each probe sends the Hub its valid reading (see below) it produces, each probe behaves differently: For each reading ProbeA sends to the Hub, it waits to receive an acknowledgement (return message) from the Hub, before it continues to generate its next reading. Unlike ProbeA, the other 2 probes must not receive any acknowledgement from the Hub.

VALID READING

A valid reading is defined as: A random integer[5] that is divisible by its magic_seed[6] (see footnote). That is, any random integer which is not divisible by any magic_seed is discarded. The 3 magic_seeds (alpha, beta, rho) must satisfy this mathematical condition: alpha > 3 * beta > rho. ProbeA would use alpha as its magic_seed, beta is used in ProbB and rho goes with ProbeC. Furthermore, each magic_seed needs to be chosen to be sufficiently large, depending on the speed of your CPU[7]. You may try 997, 257, 251 initially, although there may require some experimentation in choosing them.

PROBE TERMINATION

Each probe terminates differently. ProbeA terminates when it produces a random integer smaller than 100. After the Hub had received a total of 10,000 messages, the Hub forces ProbeB to exit. This requires using the force_patch file provided by your instructor. Lastly, the user uses a kill command in a separate terminal to kill ProbeC. This can be accomplished with the kill_patch file provided by your instructor. Finally, the DataHub terminates after all 3 probes had exited.

---

1 This project can be completed after the lab lecture on the sample program on message queue.

2 This is an abstraction and simplification of a monitoring system. For instance, one may require a system that monitors wildlife or marine life. One sensor counts elephants in the wild, a second detects zebras, and a third monitors tigers.

3 If you need assistance with finding a group, let me know.

4 Simulated by a random integer value

5 Use the full range of random values generated by function rand(), i.e., do not truncate with a mod operator

6 Each probe is assigned its own magic_seed: alpha, beta or rho. It's best to chose alpha, beta and rho to be prime numbers.

7 When the magic_seed is too small, a fast CPU would fill the message queue instantly before the next probe gets a chance to execute.

There are a number of ways to design[8] (e.g., handling non-existing queue exception, protocol for exchanging messages) & implement this system. Each design likely has advantages & disadvantages. Therefore, it is important that students explain the rationale of their chosen design & implementation.

Programming notes:

1. Students will construct 4 separate programs (fewer is possible too). The magic_seed values can either be hard coded or user input or command line switch. Each program needs to execute with the maximum degree of concurrency[9], without unnecessary delay.
2. Student needs to design the content of the message structure and the communication protocol. That is, this document only specifies data items that are relevant to the tasks the programs perform. Student may include other pieces of information and/or other messages that are needed to ensure that the system operates correctly and cohesively. Nonetheless, minimize overhead. Remember to include the mtype variable as the very first field of the message structure and the message structure must be a fix-sized object; it cannot include pointers, linked lists, string object, vectors, etc.
3. Assume lossless communication (obviously not true in real life).
4. State and document additional constraints and assumptions in your design and implementation.
5. The programs should not execute in any predictable order, unless absolutely necessary, e.g., the program which creates the message queue must start first or the program that deallocates the message queue must be last to exit (although it may finish its tasks before the others).
6. Besides the patch files, all interprocess communication will be accomplished with a single message queue. Remove the queue automatically when the last program exits. The queue should be empty when it's deleted.
7. Avoid using the *sleep( )* function or do not use a busy wait loop[10] – a loop structure whose primary purpose is to halt or sequence program execution until a certain condition is satisfied.
8. Race conditions (e.g., using the queue before it is created, deleting the queue too soon, taking a message that is intended for another recipient, etc.) are to be avoided.
9. It is easiest to execute each program in its own separate terminal/window.

Submit hard copy of your source code & documentation, a cover page with description of the software, and your information (names of each member of the group, date, the course number, email address). Show your instructor the execution of your programs.

Note on working with message queue:
Students can learn more about a system call using Linux command 'man 2 syscall' where syscall is one of: msgget, msgsnd, msgrcv, or msgctl. The msgget( ) function as shown in the sample program B would return an error (-1) if the queue already exists -- not deleted on previous execution. This situation may be handled manually with two Linux terminal commands: 'ipcs' and 'ipcrm msg msqid' -- see 'man ipcs' for more info. To remove the unallocated queue, use the ipcrm command with the msqid value from the ipcs command.

Below is a sample output of the ipcs command. When a garbage queue is present, there would be an entry under the Message Queues section.

---

8 The above specification is purposely general, in order to allow students to explore a broad range of options in choosing a design and implement for this system. Therefore, a student may produce a slightly different programming solution. Consequently, some implementations will be more responsive, more reliable, more scalable, more robust, and easier to maintain than others. It would be appropriate to document, explain, and justify your design choices and assumptions. When in doubt about the requirements of your programs, please ask questions.

9 Always allowing the highest number of processes to execute concurrently.

10 Typically present when the programmer uses the prohibited IPC_NOWAIT option.

```
$ ipcs
------ Shared Memory Segments --------
key      shmid  owner  perms  bytes   nattch  status
------ Semaphore Arrays --------
key      semid  owner  perms  nsems
------ Message Queues --------
key      msqid  owner  perms  used-bytes      messages

$
```