CS 4348/5348 Operating Systems -- Project 3

The goal of Project 3 is for you to practice programming using sockets. It is also designed to let you practice a bit of concurrency control using semaphores. In this project, you need to modify the system to include multiple computers (processes) and they interact with the printer manager via sockets. The printer process includes three components, the printer manager (the main thread), the simulated printer (now becomes a child thread), and the communicator threads (child treads). They interacts via shared address space and synchronized by semaphores.

1 The Printer Process

The major functionalities and interactions of the three types of threads in the printer process are given below.

- The printer manager (the main thread) listens on the server socket. When a connection is established with a returned new socket, the manager puts the socket and other related information in a "connection queue" (a bounded buffer).
- A free communicator thread waits to read from the connection queue and if succeeds, serves the connection.
- The communicator, once wins a connection, reads from the socket and place the read messages in its message queue (also a bounded buffer).
- The printer thread waits and reads printer requests from the message queues of the communicators. It processes each read printer request like before.

1.1 Printer Manager

In printer manager, you can implement two functions: printer_manager_init(…) and printer_manager(…). The major steps for printer_manager_init(…) are given as follows:

- Create a printer thread.
- Create $\boldsymbol{NC}$ communicator threads. (NC: number of communicators)
- Create and initialize a "connection queue" of size $\boldsymbol{CQS}$. (CQS: connection queue size)
- Create necessary semaphores and initialize them for synchronization control on the connection queue.
- Create a server socket to listen to connections. After finishing, print a message to indicate the readiness of the printer manager.

The major steps of printer_manager(…) is as follows.

- Call printer_manager_init(…).
- Listen on the server socket for connections.
- When receiving a connection, put it in the connection queue.

Synchronization between the printer manager and the communicators is similar to that of the bounded buffer problem. You can use the same solution for synchronizing them.

1.2 The Communicators

You need to implement a new function, communicator(…), which is a thread function. The major steps of each communicator are given below.

Create and initialize a message queue with $\boldsymbol{MQS}$ message slots. (MQS: message queue size)
Create and initialize semaphores for synchronizing accesses on the message queue.
Loop till the termination flag is on
  Wait on the connection queue. The wait can be realized by semaphore(s).
  Read the connection information and start serving the connection.
  Loop till the connection is closed or the termination flag is on
    Wait for messages from the connection. If a message is read, put it in its message queue.

Note that now printer may accept multiple connections from multiple computers, we need to have one thread to wait on and read from each connection (cannot have one thread to wait on multiple connections). In fact, it is the best (in terms of performance) to use the "**select**" system call to wait on multiple channels. But in this project, we choose to use multiple threads to give you the chance to practice programming for synchronization.

1.3 The Simulated Printer

Your simulated printer will be a thread now. It performs the same tasks as before, spooling and printing program outputs on a simulated paper. But instead of reading printer requests from the pipe, it now reads from the message queues of the NC communicators. Thus, you need to modify printer_init(…) and printer_main(…).

The major steps in printer_init(…) include:

- Initialize the simulated paper (like before).
- Create and initialize a **sync_pc** semaphore for synchronizing between the printer and the communicators.
- After finish initialing, you can print a message to indicate the readiness of the printer.

Note that the synchronization is not exactly the same as the bounded buffer problem. The printer should not get stuck on a single message queue when that queue is empty. You are not allowed to do busy waiting either, i.e., not to continuously checking all queues to find a message available. You need to have an additional layer of synchronization, i.e., let the printer block on the sync_pc semaphore when there is no message in any queue.

Functionality for printer_main(…) is given as follows.

```
Loop till the termination flag is on
  Wait on sync_pc semaphore.
  Once through sync_pc, do a preliminary check on each message queue and find the ones that are not empty.
  Read printer commands from the non-empty message queues.
  Reset sync_pc semaphore (careful design needed).
```

Note that if there are multiple messages ready in one or more message queues, you do not repeatedly wait on sync_pc semaphore. This can reduce the overhead on multiple lockings and multiple counter updates in semaphore wait function. However, while you are going through the above loop, some communicator may get new messages and signals sync_pc again. You need to take care of the sync_pc reset protection.

2 Simulated Computer

Your computer program in Project 2 will mostly remain the same, except for the communication mechanism encapsulated in the Print Component print.c.

2.1 Modifying the Computer

For simulating multiple computers, you need to run multiple versions of it on multiple windows (should try to run them on a few different physical computers).

Also, when starting a computer process, you need to give a **computer ID** (CID) as a **command line input**. Within a single computer, PID is sufficient for process identification. But when communicating with the printer process, you need to concatenate CID and PID to form the new ID to ensure unique identification of each process on the printer.

2.2 Modifying the Print Component

In print.c, you need to change pipe to socket when sending print requests. Also, the PID in all printer request messages should be replaced by CID+PID. For function print_init(…), you need to prepare a client socket and call "connect" to establish a connection with the printer manager. To make sure the printer manager and the printer are ready before you send a connection request and the printer requests, you should start the computer process **after** seeing the two readiness messages.

3 System Setup

The printer manager creates the server socket and the printers and computers need to know its IP address and port number. Together with the other new parameters and existing ones, config.sys in Project 3 will have the following parameters (in the order of the listing)

- IP address and port number of the printer manager.
- The existing system parameters, M, TQ, and PT.
- The new parameters, NC, CQS, MQS.

4 Run Time Output and Testing

For testing and grading purpose, you need to provide clear outputs to show the behaviors of your program. Below are some situations that you definitely should display the corresponding messages:

- What was required in Project 2.
- When a communicator wins a connection.
- When the printer starts and ends (before waiting) reading the printer requests. Your messages should clearly show your design of the synchronization, but not to over crowd the display.
- When any other important events happen.

The list above may not be complete. You can add more situations for displaying messages. TA may add additional ones as well.

5 Project Submission

You need to submit your program **before** midnight (11:59pm) of the due date (check the web page for the due date). Use UTD elearning system for submission: "elearning.utdallas.edu".

Your submission should include the following:

- The same set of files for Project 2, but modified.
- Any other source code files you may have added.
- A “design” file to discuss your design for the synchronization between the communicators and the printer.
- A “readme” file
    - Indicate how to compile and run your programs in case it deviates from the specification
    - If you did not finish the project, you need to specify which part(s) you have completed and which part(s) are missing
- You can zip or tar all your submission files together and then submit it on elearning