Design File

The C program simulates a multi-threaded printer manager server that takes connections from clients and handles messages. Semaphores and mutexes are used by the server to maintain correct synchronisation and prevent race conditions.

Semaphores:

1. **sync_pc**: The communication between the communicators and the printer is synchronised using this semaphore. Ensuring that the printer's main thread only processes messages when there are messages available in any of the message queues is its primary duty

    1. Initialization:

sync_pc is initialised in the printer_init() function with the following line:

sem_init(&sync_pc, 0, 0)

This initialises the semaphore with a value of 0, meaning that there are no messages available initially.

2. Communicator Threads:
   The following line in the client_handler() function signals the sync_pc semaphore when a message is received from a client and put to a message queue:
   Indicating that a new message has become accessible in the message queue, sem_post(&sync_pc); increases the value of the semaphore by 1

3. Printer main thread:
   In the printer_main() function, the printer main thread waits for the sync_pc semaphore with the following line:

sem_wait(&sync_pc);

The sem_wait function blocks the printer main thread until the value of the semaphore is greater than 0, meaning that there's at least one message available in the message queues. When a client handler thread adds a message to a message queue and signals the semaphore, the printer main thread proceeds to process the messages from the non-empty message queues.

2. **queue_semaphore**:
   To restrict entry to the connection_queue, this semaphore is employed. It guarantees that only one thread at a moment can modify the queue.

Mutexes:

1. **printer_mutex**: The printer_mutex is used in the printer_main() method to control access to the message_queues array. To guarantee exclusive access to the message queues while reading their contents, the printer process makes use of it. The pthread_mutex_lock(&printer_mutex) function is used to safeguard the sync_pc semaphore from possible race conditions that might develop if multiple threads attempt to access it concurrently.
   A mutex called printer_mutex makes sure that only one process can access a crucial area of the code that uses the sync_pc semaphore. As shown here

   1. Initialization: The printer_mutex is initialised in the printer_init() function with the following line:
      pthread_mutex_init(&printer_mutex, NULL);

      Printer main thread: In the printer_main() function, after the printer main thread has waited for the sync_pc semaphore and it's ready to process messages from non-empty message queues, it locks the printer_mutex with the following line:

      pthread_mutex_lock(&printer_mutex);
      This ensures that only one thread at a time can access the sync_pc semaphore and the non-empty message queues.

      Printer main thread (unlocking):The following line releases the printer_mutex after the printer main thread has handled messages from non-empty message queues:

      Other threads can access the sync_pc semaphore, the non-empty message queues, and secure the printer_mutex by using the command pthread_mutex_unlock(&printer_mutex).

2. **MessageQueue.mutex:** This mutex, which is a component of the MessageQueue structure, serves to restrict access to the data in each individual message queue. (messages, front, and rear). It guarantees that only one thread at a moment can modify a message queue.

Functions and their functionalities:

1. **printer_init():** Initiates the printer.out output file, initialises the printer_mutex and sync_pc semaphore, and starts the printer.

2. **printer_main():** Changes the front index of the message queues while reading messages from non-empty message queues and writing them to the output file. To guarantee exclusive access to message queues while receiving, it makes use of the printer_mutex.

3. **printer_manager_init():** Initiates a listening socket, binds it to a particular address and port, and starts the server by listening for connections. Additionally, it generates client_handler threads to control client connections and initialises the queue_semaphore.

4. **printer_manager():** Calls the printer_manager_init() function.

5. **client_handler():** a procedure that is carried out by each client_handler node. It manages client connections by reading messages from the clients, adding the messages to the proper message queue, and controlling termination signs.

6. **init_message_queues()**:Sets the front and back indexes to 0 and initialises the mutex for each queue to initialise the message_queues array.

7. **add_message_to_queue()**: If the specified message queue is not already filled, it adds a message to it. It ensures exclusive access while modifying the list by using the mutex of the queue.

8. **print_message_queue():** Prints the contents of the specified message queue.

9. **add_connection_to_queue():** enables exclusive access by adding a fresh client connection to the connection_queue while using the queue_semaphore.

10. **remove_connection_from_queue():** Removes a client connection from the connection_queue, returns it, and uses the queue_semaphore to guarantee exclusive access.

11. **is_message_queue_empty():** Checks if the specified message queue is empty.