ECE350 Lab3 Report

Group 57

Algorithms & Data Structures

Circular Queue

To maintain the mailboxes of each tab, we added a circular queue data structure to our TCB struct. The circular queue consists of the following components: a kernel-owned block of allocated memory, which will contain a contiguous set of messages and associated metadata, as well as the size of that block of memory; pointers to the first and last messages in the mailbox; the number of remaining bytes available in the mailbox.

The methods that we're supporting for our circular queue data structure are enqueue and dequeue.

Enqueue

The enqueue method accepts a message + metadata, as well as the receiver mailbox to add the message to. The message gets added to the mailbox at the end of the queue of messages. Assuming the arguments to the method are valid and there's enough space in the mailbox, the two cases to consider are when we do and don't need to wrap around from the end of the allocated memory to the front of the memory block. If the message we're adding doesn't reach the end of the allocated memory block, we can simply add it behind the current tail message, and change the tail pointer to this new message. In the other case, we copy the message + metadata structure byte by byte, wrapping around to the front of the memory block when we get to the end of the allocated memory.

Dequeue

The dequeue method returns the first message in the queue, along with its associated metadata, and updates variables associated with the circular queue, such as the head pointer and the remaining bytes available. There is a special case if the head message is stored across the boundary of the mailbox's allocated memory block. In that case, we have to build up the message + metadata struct byte by byte, wrapping around to the front of the memory block somewhere in the process.

Test Cases

Case 1: Sending and receiving messages

1. Task 1 creates mailbox
2. Task 1 sends a DEFAULT type message with data “Z” to itself
3. Task 1 receives message to get the sent message with data “Z”

Case 2: Sending and receiving messages between 2 tasks

1. Task 1 creates mailbox
2. Task 1 lowers its priority and raises Task 2’s priority to run Task 2
3. Task 2 creates mailbox
4. Task 2 sends a DEFAULT type message with data “Z” to Task 1’s mailbox
5. Task 2 lowers its priority to run Task 1
6. Task 1 receives message to get Task 2’s sent message
7. Task 1 sends a DEFAULT type message with data “RAPS” to Task 2’s mailbox
8. Task 1 raises Task 2’s priority to run Task 2
9. Task 2 receives Task 1’s message with data “RAPS”

Case 3: Verifying correct message order

1. Task 1 creates mailbox
2. Task 1 lowers its priority and raises Task 2’s priority to run Task 2
3. Task 2 creates mailbox
4. Task 2 sends a DEFAULT type message with data “Z” to Task 1’s mailbox
5. Task 2 sends a DEFAULT type message with data “RAPS” to Task 1’s mailbox
6. Task 2 lowers its priority to run Task 1
10. Task 1 receives message to get the first message with data “Z”
11. Task 1 receives message to get the second message with data “RAPS”