Jeffrey Cocklin

4-20-17

CSCE 311 section 1

Project 5 Ports

For the Ports project, I began by implementing the Message constructor, with a call to super that passes along the length of the message. Next I went to implement PortCB by first introducing a new variable avlBufferSpace, to track the changes made to a port’s buffer. I then initialize the variable in the constructor after making a call to super with no parameters. The do create method first grabs the current task, and if it is null returns null. In the case the current task is not null, I check if the number of ports assigned to the task have not exceeded the limit, if so I return null. Otherwise, I create a new Port, and try to add it to the task, if it fails I return null. If it succeeds, the port is assigned to the task, and given status PortLive before returning the newPort. Implementing do\_destroy only required that the port the method was called on have its status set to PortDestroyed, the appropriate threads notified, and the subsequent removal of the port from its task. Afterward I only need to reset the task of the port to null.

The do\_send method proved to be challenging, but manageable; I started by checking to see if the msg was not null, and not larger than PortBufferLength. If the msg failed on either condition failure is return and the method ends. In the case of success a new SystemEvent is created called MyEvent, and the current thread is suspended on MyEvent. Following that, I enter a while loop that will check a series of conditions, and set the values of Booleans flags; noSpace, livePort, and liveThread. As the names would suggest these flags represent the current status of Space in the msg buffer, the status of the Port, and the status of the currentThread, respectively. If liveThread is false, the current thread is remove, and I exit the loop, and failure is return. If livePort is false, all threads suspended on MyEvent are notified, and the loop is exited, and failure is returned. If noSpace is true, the currentThread is suspended on the port. In the event that noSpace is false I break the loop and check if liveThread and livePort are true, if not I return failure. In the case that noSpace is false, and liveThread and livePort are true, I check the buffer to see if it was previous empty, and save the result as a Boolean, the new message is then appended, and if the buffer was previous empty, the threads suspended on the port are notified. Next the avlBufferSpace is decremented by the length of the message, should the avlBufferSpace become negative, it is reset to zero. Afterward the threads on MyEvent are notified and Success is return. The do\_receive method works in much the same way.

The do\_ receive method grabs the current task, and if the currentTask is the same as the receiving port’s currentTask, the method proceeds, if not the method ends and null is returned. In the case of success, a new SystemEvent MyEventR is created, I then enter a while loop, where I check if the port buffer is empty, if so I suspend the thread on the port, and Boolean flag empty is set to true. Much like in the previous paragraph the conditions of liveThread, and livePort are checked, and if the result of either of these conditions is false, the method ends and returns null. In the case empty is false and livePort and liveThread are true, the message is removed from the buffer. The AvlBufferSpace is then increased by the length of message that was removed, if the increase exceeds PortBufferLength available space is set to PortBufferLength. Finally, threads suspended on the port are then notified, followed by threads suspended on MyEventR, and the message is returned.