CSE 451
Problem Set #1

Due: 9:00pm, Friday, October 31, 2014
To be done INDIVIDUALLY

1. A river crossing is shared by both cannibals and missionaries. One boat is available to cross the river, but it only seats **four** people and must always carry a full load. In order to guarantee the safety of the missionaries, you cannot put three cannibals and one missionary in the same boat -- because the cannibals would gang up and eat the missionary. Similarly, you cannot put three missionaries in the same boat as a cannibal -- because the missionaries would gang up and convert the cannibal ;-). All other combinations are safe.

Two procedures are needed: *MissionaryArrives* and *CannibalArrives*, called by a missionary or cannibal when he/she arrives at the river bank. (Since they are duals, you only need to write **one** of the procedures.) The procedures arrange the arriving missionaries and cannibals into safe boatloads; once the boat is full, one thread calls *Rowboat*. Once the call to *Rowboat* returns, and not before, the four threads representing the people in the boat can return/proceed. (A long rope is tied to the boat, so it can be pulled empty back to the start.)

You must use locks and Mesa-style condition variables to implement the solution, using the best practices as defined in class and the textbook. There should be no busy-waiting. There must also be no useless waiting: missionaries and cannibals should not wait if there are enough of them for a safe boatload.

2. A **synchronous** buffer is one where the thread placing an item into the buffer waits until the thread retrieving the item has gotten it, and then the sender can return.

SyncBuf::put(item): put item into the buffer and return only once the item has been retrieved by some thread.

SyncBuf::get(): wait until there is an item in the buffer, and then return it.

Any number of threads can concurrently call SyncBuf::get and put; the implementation pairs off puts and gets – each put item should be returned exactly once. The threads do not need to be paired in any particular order.

You must use locks and Mesa-style condition variables to implement the solution, using the best practices as defined in class and the textbook. There should be no busy-waiting. There must also be no useless waiting: the put and get should return if there is a corresponding call to the other routine.