Boat Question

[Background] This question is adapted from previous years. It has nothing to do with the previous recommendation systems. Please treat it as a separate question letting you write a thread model for a real-world simulator.

A number of Hawaiian adults and children are trying to get from Oahu to Molokai. Unfortunately, they have only one boat that can carry maximally two children or one adult (but **not** one child and one adult). The boat can be rowed back to Oahu, but it requires a pilot to do so.

Requirements:

Arrange a solution to transfer everyone from Oahu to Molokai. You may assume that there are **at least two children**.

The method Boat.begin() should create a thread for each child or adult. We will refer to the thread that called Boat.begin() as the parent thread. Your mechanism cannot rely on knowing how many children or adults are present beforehand, although you are free to attempt to determine this among the threads (i.e. you can't pass the parameters adults and children in the method begin() to your threads, but you are free to have each thread increment shared variables to try and determine this value, if you wish).

To show that the trip is properly synchronized, make calls to the appropriate BoatGrader methods every time someone crosses the channel. When a child pilots the boat from Oahu to Molokai, call ChildRowToMolokai. When a child rides as a passenger from Oahu to Molokai, call ChildRideToMolokai. Make sure that when a boat with two people on it crosses, the pilot calls the ...RowTo... method before the passenger calls the ...RideTo... method.

Your solution must **have no busy waiting**, and it must eventually end. The simulation ends when the parent thread finishes running. Note that it is not necessary to terminate all the created threads -- you can leave them blocked waiting for a condition variable. While you cannot pass the number of threads created to the threads representing adults and children, you can and probably will need to use this number in begin() in order to determine when all the adults and children are across and you can return.

The idea behind this task is to use **independent** threads to solve a problem. You are to program the logic that a child or an adult would follow if that person were in this situation. For example, it is reasonable to allow a person to see how many children or adults are on the same island they are on. A person could see whether the boat is at their island. A person can know which island they are on. All of this information may be stored with each individual thread or in shared variables. So a counter that holds the number of children on Oahu would be allowed, so long as only threads that represent people on Oahu could access it.

What is not allowed is a thread that executes a **"top-down"** strategy for the simulation. For example, you may not create threads for children and adults, then have a controller thread simply send commands to them through communicators. The threads must act as if they were individuals. This also means you cannot serialize for any reason. Every person must be thinking, and acting, independently at all times.

Information that is not possible **in the real world** is also not allowed. For example, a child on Molokai cannot magically see all of the people on Oahu. That child may remember the number of people that he or she has seen leaving, but the child may not view people on Oahu as if it were there. (Assume that the people do not have any technology other than a boat!)

You will reach a point in your simulation where the adult and child threads believe that everyone is across on Molokai. At this point, you are allowed to do **one-way** communication from the adult/child threads to begin() (the parent thread) in order to inform it that the simulation may be over. It may be possible, however, that your adult and child threads are incorrect. Your simulation must handle this case without requiring explicit or implicit communication from begin() (the parent thread) to the adult/child threads.

ToDo:

Implement a class **Boat** in the files *boat.h* and *boat.cc* exposing the member function *Boat.begin()*. Your threads should call grading functions in *boatGrader.h* when making decisions. An example of interface usage in the final grading has been provided in *main.cc*.

Grading:

You will be graded by the correctness of your solution. Violation of design principles (refer to requirements) will result in demerit points.