Threads Assignment

Store your solutions in a subdirectory of your home directory named `Boats` by 2:00PM EDT April 15, 2016. Use the filenames that suggest which version they solve, e.g., `solution1` for the base case and `solutions2` for the complete case. Note: I will compile your programs using the files in `/u1/junk/cs670`.

Three types of people arrive at a dock ready for a boat ride: captains, sailors, and passengers. In order to sail, a boat requires one captain and two sailors. The boat can carry up to three passengers. Normally a boat will sail when all six people have boarded the boat. However, if only two passengers are on the boat and one of them has waited `MAXWAIT` time units, the boat will sail with only two passengers. In addition, if a boat is full of (three) passengers but needs one sailor, a waiting captain can replace *one* sailor. Begin writing your program to handle the base case (1 captain, 2 sailors, 3 passengers), when that works well, consider the enhancements.

I will provide a program that generates arrivals at the dock, and I will provide a module that implements the `board()` and `sail()` actions. Note: the current version only supports the base case for the problem (see above). Your job is to read my arrival generator (as a pipe), create threads for each arrival, and synchronize boarding and sailing. You will also need to recycle thread ids. Make sure that all data that is accessed by multiple threads is protected by mutexes.

The arrival data will be read from the pipe, one line at a time. The lines will contain a name (no spaces, with a 15 character maximum) and a role (C, S, or P).

The `board()` function will expect a pointer to the data for the person boarding the boat, as follows. See `/u1/junk/cs670/boat.h` for further details.

```
typedef struct person {
   char *name;
   int role;
   int arrival_time;
   int board_time;      /* my modules will assign this field */
   int sail_time;       /* my modules will assign this field */
   void *other;         /* available for your use, as you see fit */
} PERSON;
```

My module will keep track of the people who have boarded using this format. When one of your threads executes the sail function, it print data for each person on the boat, check for correctness, and empty the boat. Your threads are responsible for memory management of the PERSON data structures. My module will copy your data as needed.

Your main thread should respond to the SIGINT signal by printing a list of those waiting in line (including roles and arrival times). My module will provide a `boat_report()` function that prints the data for people for have boarded on `stdout`.

Your job is the write the main program and three threads (one for each role). Study the *River Crossing Problem* from the *Little Book of Semaphores* for guidance on getting started.

All times will be in seconds. Your timer should use the `gettime()` function I provided in `boat.c` to obtain the current time (in seconds). Don't forget to include `boat.h`.

The following files are available in `/u1/junk/cs670`:

`boat.h`

`boat.c`

`generator` has been installed in `/usr/local/bin`

DO NOT COPY THESE FILES, use the copies in `/u1/junk/cs670`.

The `generator` program accepts an integer command line argument which seeds the random number generator. With no argument it uses a seed of 2.