The Doctor on Night Duty

A doctor in a clinic has a small examination room where she examines her patients. After the initial examination, the doctor sends the patient to the lab for tests. Once the test is done, the patient visits the doctor again to receive prescription. (This means that each patient MUST visit the doctor twice). Attached to her examination room is a waiting room with four chairs where patients can sit and wait if the doctor is currently examining another patient. When there are no patients who need help, the doctor takes a nap. If a patient arrives and finds the doctor sleeping, the patient awakens the doctor. If a patient arrives and finds the doctor currently examining another patient, the patient sits on one of the chairs in the waiting room and waits. If no chairs are available, the patient goes away to have coffee from the clinic's cafeteria and will come back later.
Using multithreading, write a program (using C or C++) that coordinates the activities of the doctor and the patients. Use mutex locks and semaphores to ensure that two patients are not in the examination room at the same time and that the patients waiting in the waiting room get their turn. More details are provided below.

The Doctor and the Patients

You must create *n* patients (where *n* is input by the user) using PThreads, with each patient running as a separate thread. The doctor runs as a separate thread as well. Patient threads will alternate between being seen by the doctor to visiting the lab or the cafeteria. If the doctor is available, patient will be treated immediately. Otherwise, the patient will either sit in a chair in the waiting room or, if no chairs are available, will go to the cafeteria for a random amount of time and will return to see the doctor. If a patient arrives and notices that the doctor is taking a nap, the patient must notify the doctor using a semaphore. When the doctor finishes examining a patient, the patient must visit the lab for a random amount of time and then return to see the doctor again. The patient might have to wait in the waiting room again if the doctor is busy; or drink coffee from the cafeteria if there is no empty chair in the waiting room. Meanwhile, the doctor must check to see if there are patients waiting for help in the waiting room. If so, the doctor must help each of these patients. If no patients are present, the doctor may return to napping.

Perhaps the best option for simulating the time that patients spend in the lab or drinking coffee in the cafeteria, as well as the doctor treating a patient, is to have the appropriate threads sleep for a random period.

Each patient thread should print its state (drinking coffee, waiting for doctor, getting first examination, getting second examination, visiting the lab, waking up doctor, etc.) along with its patient number. Similarly, the doctor should print its state (examining, checking for next patient, sleeping etc.). A sample output is shown on the following page.

Pthreads are covered in section 4.4.1, while mutex locks and semaphores are covered in section 5.9.4 of your textbook.

GRADING CRITERIA:

- Your application uses the appropriate number of threads for patients (as input by the user) and the doctor. [2]
- Your application uses mutexes and semaphores properly, in the right places, and in the correct sequence. [8]
- Output is correctly displayed and follows the correct flow. [4]
- Return values for system calls are checked and errors handled (if any). [2 marks]
- Clean up is done, i.e., threads are properly joined, and any memory allocated is properly deallocated (if required). [2 marks]
- Code is well written, properly formatted and commented. Readme.txt file is provided with the correct compilation string and specific contribution made by group members (if applicable). [2 marks]

SAMPLE OUTPUT:

Sample output is shown below. Indentation allows for clear distinction between messages.
Note: The output is meant for guidance pertaining to messages only. Actual sequence of events will vary based on the number of threads, how random numbers are generated, and how scheduler allocates CPU to each thread. (Note that for this assignment, arbitrary selection of waiting patients is allowed)

```
                Doctor sleeping
    Waiting Room empty. Patient 1 wakes up the doctor
Doctor examining Patient 1 for 3 seconds. Chairs occupied = 0
Patient 1 getting first examination
        Patient 2 waiting. Chairs occupied = 1
        Patient 3 waiting. Chairs occupied = 2
        Patient 4 waiting. Chairs occupied = 3
        Patient 5 waiting. Chairs occupied = 4
    Waiting room full. Patient 6 drinking coffee for 1 seconds
            Patient 1 visiting the lab for 2 seconds
Doctor examining Patient 2 for 1 seconds. Chairs occupied = 3
Patient 2 getting first examination
        Patient 7 waiting. Chairs occupied = 4
    Waiting room full. Patient 1 drinking coffee for 1 seconds
    Waiting room full. Patient 6 drinking coffee for 3 seconds
            Patient 2 visiting the lab for 3 seconds
Doctor examining Patient 3 for 1 seconds. Chairs occupied = 3
Patient 3 getting first examination
        Patient 1 waiting. Chairs occupied = 4
            Patient 3 visiting the lab for 3 seconds
Doctor examining Patient 4 for 1 seconds. Chairs occupied = 3
Patient 4 getting first examination
            Patient 4 visiting the lab for 2 seconds
Doctor examining Patient 1 for 1 seconds. Chairs occupied = 2
Patient 1 getting second examination
                Patient 1 going home
Patient 5 getting first examination
Doctor examining Patient 5 for 2 seconds. Chairs occupied = 1
        Patient 6 waiting. Chairs occupied = 2
```

Sample Output. Note that this is a subset of the program's output