A barbershop contains a barber, the barber's chair, and N chairs for waiting customers. When there are no customers, the barber sits in his chair and sleeps. As soon as a customer arrives, he either awakens the barber or, if the barber is cutting someone else's hair, sits down in one of the vacant chairs; waiting customers have their hair cut in order of arrival. If all of the chairs are occupied, the newly arrived customer simply leaves. For this problem set we will model the behavior of this system as concurrent threads of execution - one for the barber, and one for each of the customers.

Question 1 – monitor and semaphore solution

Provide pseudo-code for an object which has two methods, barber() and customer():

- barber() does not return, but instead loops forever sleeping and cutting hair
- a thread entering the customer() method corresponds to arrival of a customer at the shop; the thread returns from customer() when the customer would leave. (I.e after getting a haircut, or immediately if the shop is full)

Note that there are lots of solutions to this problem on the web, but the version described here is a bit more complicated because of the requirement that customer() not return until the haircut is finished.

a) Semaphore solution: A straightforward solution uses 4 semaphores - one for a mutex to guard shared variables ("mutex"), one for customers waiting in line ("waiting"), one for the sleeping barber ("sleeping"), and one for a customer getting his/her hair cut ("sitting"). The only shared variable is "numinshop", a count of the total number of customers in the shop ("waiting" + "sitting").

```plaintext
barber:
    while true
        loop forever
            while true
                wait(mutex) complicated threadsafe "while numinshop > 0"
                tmp = numinshop
                signal(mutex)
                if tmp == 0 break
                signal(waiting) tell a customer it's time to get their hair cut
                ... cut hair ...
                signal(sitting) tell the customer he/she is done
                wait(mutex) and decrement the number in the shop.
                numinshop--
                signal(mutex)
            end loop
            wait(sleeping) no one in shop go to sleep
        end loop
```

customer:
  \begin{verbatim}
  wait(mutex) \quad safely check how many customers are in the shop
  \text{tmp} = \text{numinshop} \quad if not full, increment the customer count
  if \text{numinshop} < N+1
    \text{numinshop}++
  \text{signal(mutex)}

  if \text{tmp} < N+1
    if \text{tmp} == 0
      \text{signal(sleeping)} \quad if the shop isn't full, go in
      \text{wait(waiting)} \quad if no one else here, wake the barber
      \text{wait(sitting)} \quad wait until it's our turn
      \text{end if}
  \end{verbatim}

b) Monitor solution based on the semaphore:

\begin{verbatim}
monitor barbershop =
  int \text{numinshop}
  condition waiting, sleeping, sitting

barber =
  while true
    while \text{numinshop} > 0
      \text{signal(waiting)} \quad ... cut hair ...
      \text{signal(sitting)}
      \text{numinshop}--
    end while
    \text{wait(sleeping)}
  end while

customer =
  \text{numinshop}++
  if \text{numinshop} == 1
    \text{signal(sleeping)}
    \text{wait(waiting)}
    \text{wait(sitting)}
  \end monitor
\end{verbatim}

Finally, a bakery algorithm version of the monitor solution. Note that this version can use broadcast() instead of signal(), and thus can be implemented in a straightforward fashion with a Java monitor.

\begin{verbatim}
monitor barbershop =
  int \text{cutting}, \text{next_num}
  condition waiting, sleeping, sitting

barber:
  while true
    while \text{cutting} == \text{next_num} \quad -- no one in the shop
      \text{wait(sleeping)}
      \text{signal(waiting)} \quad -- get customer from queue
      \text{... cut hair ...}
      \text{signal(sitting)}
      \text{cutting}++ \quad -- next customer's turn
    end while
  \end while
\end{verbatim}
Question 5 – Markov model

The barbershop with 5 seats and exponentially distributed wait and haircut times can be modeled as a Markov process with 6 states corresponding to the number of occupied chairs. Do it.

![Markov model diagram]

a) What is the steady state probability of each state?

\[
\begin{align*}
1.0 \, P_0 & - 0.8 \, P_1 = 0 \\
-1.0 \, P_0 + (0.8 + 0.9) \, P_1 - 0.8 \, P_2 & = 0 \\
-0.9 \, P_1 + (0.8 + 0.8) \, P_2 - 0.8 \, P_3 & = 0 \\
-0.8 \, P_2 + (0.8 + 0.7) \, P_3 - 0.8 \, P_4 & = 0 \\
-0.7 \, P_3 + (0.8 + 0.6) \, P_4 - 0.8 \, P_5 & = 0 \\
-0.6 \, P_4 + (0.8 + 0.5) \, P_5 - 0.8 \, P_6 & = 0 \\
P_0 + P_1 + P_2 + P_3 + P_4 + P_5 + P_6 & = 1.0
\end{align*}
\]

\( P_0 = 0.128327, \ P_1 = 0.160409, \ P_2 = 0.180460, \ P_3 = 0.180460, \ P_4 = 0.157902, \ P_5 = 0.118427, \ P_6 = 0.074017 \)

The average number of customers in the shop is just:

\( 1 \, P_1 + 2 \, P_2 + 3 \, P_3 + 4 \, P_4 + 5 \, P_5 + 6 \, P_6 = 2.7305 \)

b) What is the probability of a customer turning away due to a full shop?

If there was a constant memoryless arrival rate, then it would just be \( P_6 \). As mentioned in class, however, this is a bit tricky. In state 6 customers arrive at a rate of 0.4, for a total rejection rate \( R \) of 0.4\( \cdot \)\( P_6 \) or 0.0296 rejections per second. The total customer arrival rate is 1.0 \( P_0 + 0.9 \, P_1 + \ldots + 0.4 \, P_6 = 0.7270 \) arrivals/sec; the fraction which result in rejection is thus 0.0296 / 0.7270 or 0.0407.

An alternate way to calculate this is to say that the barber cuts hair at a rate \( R_B = 0.8 \cdot (1 - P_0) \), giving an arrival rate including rejections of \( R_B + R \); thus the rejected fraction is \( R / (R_B + R) \) or again 0.0407.
c) What is the expected time in the shop (including haircut) if the shop is not full

The expected number of customers in the shop, conditioned on there not being 6 customers, is:

\[
(0 \cdot P_0 + 1 \cdot P_1 + 2 \cdot P_2 + 3 \cdot P_3 + 4 \cdot P_4 + 5 \cdot P_5) / (P_0 + P_1 + P_2 + P_3 + P_4 + P_5) = 2.4692
\]

At 1.25 seconds/haircut, the naive answer would be 1.25 \cdot (1 + 2.4692); unfortunately as in (b) this isn't correct. The average number of customers seen by arriving customers is \((1.0 \cdot 0 \cdot P_0 + 0.9 \cdot 1 \cdot P_1 + 0.8 \cdot 2 \cdot P_2 + ... + 0.4 \cdot 6 \cdot P_6) / (\text{total customer arrival rate})\), or \(1.665 / 0.7270 = 2.290\), and giving an expected time of \(1.25 \cdot (1 + 2.290)\).

(even this isn't quite correct, as it doesn't correct for the full shop case. However, it's late and I want to post this so people can look at it.)