1. [10 points]
Requests are issued to an io device at the rate of 100 requests/second. The requests are of two types: 1/3 are type 1 and the remainder are type 2. The average completion time (from entry to departure) over all requests is 55 ms (milliseconds). The average completion time for type 1 requests is 33 ms.

a. What is the average number of requests (of both types) at the io device (both waiting and served)? Explain briefly.

b. What is the average number of type 2 requests at the io device. Explain briefly.

Solution

Part a [4 pt]
Overall throughput \( X = 100 \) req/s
Avg completion time \( R = 55 \) ms
From Little's Law, avg number of requests \( N = X \times R \)
\[ = 100 \text{ req/s} \times 55 \text{ ms} = 100 \times 55 \times 10^{-3} = 5.5 \]

Part b [6 pt]
Type 1 throughput \( X_1 = 100/3 \) req/s
Type 1 avg completion time \( R_1 = 33 \) ms
From Little's Law, avg number of type 1 requests \( N_1 = X_1 \times R_1 \)
\[ = (100/3) \text{ req/s} \times 33 \text{ ms} = 100 \times 11 \times 10^{-3} = 1.1 \]
So avg number of type 2 requests \( N_2 = N - N_1 = 5.5 - 1.1 = 4.4 \)

2. [10 points] Here is an attempted 2-user spinlock for a multi-cpu system. Does it ensure that at most user holds the lock at any time? If yes, explain very briefly. If no, give an evolution ending with both users holding the lock.

Solution

NO. For example, evolution 0.s1, 1.s1..s3, 1.s2..s3 ends with both threads holding the lock. Here is a more detailed look at this evolution.

<table>
<thead>
<tr>
<th>flag[0]</th>
<th>flag[1]</th>
<th>turn</th>
</tr>
</thead>
<tbody>
<tr>
<td>initially</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>0.s1</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>1.s1</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>1.s2</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>1.s3</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>0.s2</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>0.s3</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>

Grading if you said YES (i.e., at most one thread holds lock at any time)

- attempted a good analysis (identifying the cases, etc.) [5 pt]
- poor analysis (e.g., treating s1, s2 as atomic) [2 pt]
3. **[20 points]** Here is a skeleton (multi-cpu) implementation of a condition variable cv associated with lock lck. Complete the implementation by filling in the boxes. Do not change what is already given.

**Variables**
- runq: run queue
- runqL: spinlock protecting runq
- readyq: ready queue
- readyqL: spinlock protecting readyq
- lck: lock associated with cv
- cvq: pcb queue for cv
- cvqL: spinlock protecting cvq
- supply other static variables if needed

**Functions**
- scheduler(): assumes runqL and readyqL are free when called
- updateRunqPcb()
- cv.signal():
  - cvqL.acq() if (cvq not empty)
  - readyqL.acq()
  - move a pcb from cvq to readyq
  - readyq.rel()
  - cvqL.rel()
- cv.wait():
  - supply code

**Solution**
No other variables are needed

```c
void cv.wait(){
    cvqL.acq(); [2 pt]
    runqL.acq(); [2 pt]
    lck.rel(); [2 pt]
    updateRunqPcb(); [2 pt]
    with ra ← a1 [1 pt]
    move my pcb to cvq [2 pt]
    cvqL.rel(); [2 pt]
    runqL.rel(); [2 pt]
    scheduler(); [2 pt]
    a1: lck.acq(); [3 pt]
}
```

[-1 pt] if lck.rel() comes after runqL.rel() or updateRunqPcb()
(Probably would not work if lck is a regular (not spin) lock.)

No points lost for acquiring and releasing readyqL, as long as solution is ok.