3. Consider the following set of five processes where *arrival* is the time the process became ready, *t* is the total service time, and *e* is the external priority.

<table>
<thead>
<tr>
<th>process</th>
<th>arrival</th>
<th>t</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>p0</td>
<td>0</td>
<td>80</td>
<td>9</td>
</tr>
<tr>
<td>p1</td>
<td>15</td>
<td>25</td>
<td>10</td>
</tr>
<tr>
<td>p2</td>
<td>15</td>
<td>15</td>
<td>9</td>
</tr>
<tr>
<td>p3</td>
<td>85</td>
<td>25</td>
<td>10</td>
</tr>
<tr>
<td>p4</td>
<td>90</td>
<td>10</td>
<td>11</td>
</tr>
</tbody>
</table>

Assume that execution starts immediately at time 0 and there is no context switch overhead. For the following scheduling disciplines, draw a time diagram showing when each of the five processes executes. (In the case of a tie, assume that the process with the lower process number goes first.)

(a) FIFO
(b) SJF
(c) SRT
(d) RR (quantum = 1)

4. For each of the scheduling disciplines in the previous exercise, determine the average turnaround time for the five processes.

6. Consider *n* processes sharing the CPU in a RR fashion.

(a) Assuming that each context switch takes *s* milliseconds, determine the quantum size *q* such that the overhead resulting from process switching is minimized but, at the same time, each process is guaranteed to get its turn at the CPU at least every *t* seconds.

(b) If *n* = 100, *t* = 1, and *s* = 0.001, what would be the size of *q*? What if *s* increased to 0.01?

7. Consider a system using RR scheduling with a fixed quantum *q*. Every context switch takes *s* milliseconds. Any given process runs for an average of *t* milliseconds before it blocks or terminates.

(a) Determine the fraction of CPU time that will be wasted because of context switching for each of the following cases:
   i. *t* < *q*
   ii. *t* ≫ *q* (i.e., *t* is much greater than *q*)
   iii. *q* approaches 0

(b) Under what condition will the wasted fraction of CPU time be exactly 50%?
12. Consider four processes, \( p_0 \) through \( p_3 \). The diagram below shows the timing of each process when executed in isolation.

That means, process \( p_0 \) arrives at time 0; after one time unit it executes a \( P(s) \) operation (\( s \) is initially 1); after 3 more time units it executes a \( V(s) \) operation and terminates. Processes \( p_1 \) and \( p_3 \) are similar. Process \( p_2 \) does not execute any \( P \) or \( V \) operations. Assume that the execution of \( P \) and \( V \) is instantaneous, i.e., takes 0 time. Any context switch is also instantaneous. The priorities of the four processes are as follows: \( \text{priority}(p_0) < \text{priority}(p_1) < \text{priority}(p_2) < \text{priority}(p_4) \). Determine the start and end time of each of the four processes when executing concurrently on a single-processor system.

(a) without priority inheritance
(b) with priority inheritance
(c) without priority inheritance but making all CSs noninterruptible

(Hint: Draw timing diagrams similar to those in Figures 5-7 and Figure 5-8.)

13. Assume two processes compete for a CS implemented using simple spinning locks (Section 4.5.1). Prior to entering the CS, the process executes the statement

\[
\text{do TS(R, sb) while (!R);} 
\]

After exiting the CS, it executes the statement:

\[
\text{sb = 1;} 
\]

Show how this solution may lead to a deadlock under priority scheduling, whereas RR scheduling does not suffer from this problem.