CS 702: Midterm Exam Practice Problems

Problem 1: Suppose we have two implementations of a caching multithreaded web server. Implementation #1 uses user-level threads and can process a cache hit in 15ms of CPU time and a cache miss in 15ms of CPU time plus 35ms during which it must wait for I/O. Implementation #2 uses kernel-level threads and can process cache hits in 20ms of CPU time and cache misses in 40ms of CPU time plus 35ms during which it must wait for I/O.

(a) Assuming that 75% of requests are for pages in the cache, which implementation can process more requests per unit time?

(b) What if only 60% of requests are for pages that are in the cache?

In both cases explain your answers, and assume that the kernel can interleave I/O operations.

Problem 2: Recall the proposed use of select to avoid having one user-level thread block all others when it blocks for I/O (see p. 92). User level threads would call select before a potentially blocking I/O call in order to determine if the call would block. Consider the following pseudocode implementation of this idea:

```c
while (!select(cin))
    ; /* do nothing */

int data;
cin >> data;
```

Is this an ideal solution to the problem (that is, is this as good as kernel-level threads)? (Hint: what is the process doing while it waits for input to become available?)

Problem 3: Draw a diagram showing the parent/child relationships among the processes created by the following code fragment. Also show which call to fork created each process.

```c
pid = fork(); /* Call #1 */
if (pid != 0)
    fork(); /* Call #2 */
fork(); /* Call #3 */
```

Problem 4: Consider Peterson’s solution to the mutual exclusion problem (see pp. 105-6). Explain how this solution satisfies the criterion “no process should have to wait forever to enter its critical region” (criterion #4 on p. 102). Consider the following attempt to generalize Peterson’s solution to to three processes. Assume N has been #defined to 3 and that the processes are numbered 0, 1, and 2. Does this proposal meet all the criteria for solutions to the mutual exclusion problem?

```c
void enter_region(int process)
    int other1 = (process + 1) % 3; /* other1 and other2 are */
    int other2 = (process + 2) % 3; /* ids of the other proc */
    interested[process] = TRUE;
    turn = other1;
    while (turn != process
        && (interested[other1] || interested[other2]))
        ; /* do nothing */

void leave_region(int process)
    interested[process] = false;
```
Problem 5: Complete the following pseudocode definition of a condition variable using semaphores.

class ConditionVariable
{
    Semaphore sem = ???;
    Semaphore mutex = 1;
    // declare any counters you need

    void wait()
    {

    }

    void signal()
    {

    }

    void broadcast()
    {

    }
}
Problem 6: This problem refers to five processes with the following arrival times and times to completion.

<table>
<thead>
<tr>
<th>Process</th>
<th>Arrival</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>$P_2$</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>$P_3$</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>$P_4$</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>$P_5$</td>
<td>10</td>
<td>1</td>
</tr>
</tbody>
</table>

(a) Compute the turnaround time for each process if using (non-preemptive) Shortest Job First, breaking ties in favor of the process that arrived earliest.

(b) Find a nonpreemptive schedule that minimizes total turnaround time for the above set of processes. The CPU may be idle.

(c) Compute the turnaround time for each process if using Shortest Remaining Time First where jobs can be preempted when other jobs arrive. This time, break ties in favor of the process that arrived latest.