

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.

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

```
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.

```
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?

```
void enter_region(int process)
    int other1 = (process + 1) % 3; /* other1 and other2 are */
    int other2 = (process + 2) % 3; /* ids of the other procs */
    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 to implement a counting semaphore using a monitor (see p. 115) and condition variables. `up` may need to update the count and/or signal to other processes that a resource has become available. `down` may need to update the count and/or wait for resources to become available. You may add additional counters or condition variables as you see fit.

```
monitor counting_semaphore
{
  int count = N; /* N is the number of resources available */
  condition_variable non_zero;

  procedure up()
  {

  }

  procedure down()
  {

  }
}
```