# CMPSCI 677: Operating Systems

**Homework 1**

Spring 2004

*Due:* Tuesday, February 10 (in class). Off-campus students: one week from when you watch Lecture 2.

**Note:** Feel free to use an undergraduate operating systems text as a reference for this assignment.

---

1. True or false questions. Justify your answer in one or two sentences.

   (a) All interactive time-shared systems are also multiprogrammed systems.

   (b) Traps are a common mechanism used by the OS to implement *all* of the following: system call, page fault, and illegal memory access.

   (c) Only a parent process can kill a child process.

   (d) Kernel threads have faster context switches than user-level threads.

   (e) Shared data that is only read but not written to by cooperating threads need not be protected by a critical section.

   (f) A synchronization problem that requires counting semaphores can never be implemented using locks.

   (g) Overlays allow contiguous memory allocation techniques to support process sizes that are larger than the size of physical memory.

   (h) Compaction algorithms are required in a memory system that uses pure segmentation.

   (i) Thrashing is less likely if you use a global page replacement scheme.

   (j) Direct memory access (DMA) transfers increase contention on the system bus.

2. Write short answers

   (a) You are in charge of writing a device driver for a 56Kbps modem on your PC. Assuming you decide to use *polling* for communication between the OS and the modem hardware, explain the steps involved in writing a data packet from the OS to the modem.

   (b) Since the shortest job first scheduling algorithm has provably optimal average waiting times, would you use it to schedule processes in a conventional operating system. Why or why not?

   (c) What is second chance page replacement and why do many operating systems prefer it to LRU?

   (d) Does Banker's algorithm do deadlock prevention or deadlock avoidance? Why?

   (e) Give a 1-2 sentence definition of a system call.

   (f) Why does a pure paging scheme not suffer from external fragmentation?

3. Consider a precedence graph in which program segment $S_2$ must execute only after $S_1$, and $S_3$, $S_4$ and $S_5$ must execute only after $S_2$.

   Assume that each of the $S_i$'s is executed in a separate process $P_i$. You may assume that the processes are unrelated (i.e., the processes need not be created), and that each process $P_i$ has exactly one computation step $S_i$. Give the pseudo-code for the individual processes using each of the following: (a) Semaphores (b) Locks (c) UNIX fork() and waitpid() system calls. Your C program syntax need not be correct. But the use of the fork() and waitpid() system calls must be correct. You should permit the maximum amount of concurrency possible. Also, discuss the appropriateness of monitors to achieve this synchronization.

4. Write pseudo code to solve the following synchronization problem using locks. Consider a candy shop where each customer takes a number and waits until a sales person calls their number. The sales person services customers in the order of their ticket numbers. Write two routines *Enter* and *Service* to solve this problem using locks‘. Assume that the *Enter* routine is used by each new customer and the *Service* routine is used by the sales person to service the next customer. Be sure to clearly declare and initialize all variables. How does your solution change if you were to use semaphores instead?

5. Operating systems enforce protection using the notion of a domain and by associating a set of access privileges with each domain. Many operating systems also support a *domain switch* where an application can begin execution in one domain and then switch to another domain (and thereby acquire the privileges associated with tha t domain). Give an example of an application that might benefit from this feature.

6. - In standard uniprocessor `Unix`, explain why the following command will not produce a useful result.
     `sort < foobar > foobar`
   - What (erroneous) result will it always produce ?
   - Why, and when, would the following command *sometimes* produce a useful result?
     `sort < foobar | ( cat > foobar )`
     where the paranthesis is `sh`-ell notation to execute the contained command in a sub-shell.