CMPSCI 677: Operating Systems

Homework 1

Spring 2011

Due: 5pm, Thu, Jan 27 (via SPARK).

Note: Feel free to use an undergraduate operating systems text as a reference for this assignment. This assignment is mandatory, although it will not count towards your final grade.

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

   (a) On a uniprocessor, a CPU can execute one process while another process is performing a context switch.

   (b) It is desirable for a CPU scheduler to give priority to I/O-bound threads over CPU-bound threads.

   (c) "Hold and Wait", one of the conditions required for deadlock to occur, means that the process is holding the CPU while waiting for a resource.

   (d) Thrashing occurs when there is not enough physical memory allocated to a process to keep the pages the process is actively using in memory.

   (e) Contiguous allocation is a reasonable way to store files on write-once disks, such as traditional CD-ROMs.

   (f) Ethernet is commonly used to connect machines on a wide area network.

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

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

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

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

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

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

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

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

   (o) Since a kernel thread is a thread that the kernel knows about, kernel threads have faster context switches than user-level threads.

2. Write short answers

   (a) 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?

(b) What is LRU page replacement algorithm?

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

(d) Explain how direct memory access (DMA) works and list some of its advantages.

(e) What is a process control block (PCB) and what information does it store?

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

(g) Consider an OS that improves disk performance using disk read-ahead. For what kinds of file access patterns is disk read-ahead useful? Why is prefetching using disk read-ahead easier than prefetching virtual pages into memory?

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

   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. The local laundromat has just entered the computer age. As each customer enters, (s)he puts coins into slots at one of two stations and selects the number of washes she will need. The stations are connected to a central computer that automatically assigns available machines and outputs tokens that also identify the machines to be used. The customer places her laundry into a machine and inserts the appropriate token into the machine. When a machine is done with a wash, it informs the computer that it is available again. The computer maintains a boolean array `available` to represent if a machine is available, and a semaphore `nfree` that indicates how many machines are available. The code to allocate and release machines is as follows.

   All the elements of the `available` array are initialized to true, and `nfree` is initialized to NMACHINES.

```
semaphore nfree;
boolean available[NMACHINES];

allocate()
{
  P(nfree);
  for (int i=0; i< NMACHINES; i++)
    if (available[i] == TRUE) {
      available[i] = FALSE;
      return i;
    }
}
```

```
release(int machine)
{
  available[machine] = TRUE;
  V(nfree);
}
```

Explain how the program works. Does it work the way one would expect it to? If not, how can you modify it to work correctly?

5. Consider a disk that employs the shortest seek time first (SSTF) scheduling algorithm. Assume that the disk has 100 tracks that are numbered from 1 to 100. Further assume that the current disk head position is track number 52.

   (a) In what order does the disk service requests for blocks that are stored on the following tracks: 67 40 19 82? Assume tht all four requests arrive simultaneously.

   (b) What is the total number of tracks across which the disk seeks to satisfy these requests?

   (c) What order are the requests serviced assuming the SCAN (elevator) disk scheduling policy? Assume that the disk head is currently moving towards track 100 and that the disk head always seeks to the highest and lowest numbered tracks (i.e., tracks 100 and 1) before reversing direction.

   (d) What is the total number of tracks across which the disk seeks in case of SCAN?

   (e) Explain why SSTF scheduling tends to favor middle tracks over the innermost and outermost tracks of a disk.

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.