Operating System

7 questions in two pages, total 100 points

Note: Questions must be answered with explanations, a simple “yes” or “no” does not help the credits

1. (15%) Compare and contrast the UNIX inode-scheme for allocation of disk blocks with the FAT system used by MS-DOS/Windows.
2. (15%) Suppose that we are going to study an unknown operating system Q. There is a process creation primitive *qfork*() in Q, and we know that it is similar to the UNIX process creation primitive *fork* in that it returns some kind of process identification of the child process to the parent, whereas it returns 0 to the child. However, we do not know whether it creates a heavy weight process (like *fork*) or a thread. Please design a test (write a program) to determine which the case is and explain how it works. You may also want to make use of the primitives *qwait*(*pid*), which waits for the process/thread with identity *pid* to terminate, and *qexit*(), which terminates a process/thread.
3. (10%) Explain the difference between preemptive and non-preemptive CPU scheduling. Which scheme is ‘the best’ for a general purpose OS, would you say?
4. (15%) Given a paging system with a 48-bit logical address, 8KB pages, and 4 bytes per page entry, suppose that the maximum physical memory size is 64GB, and the system is byte-addressable. Please answer the following questions:
   (a). What is the number of bits for physical addresses?
   (b). If only simple paging (only single level, no hash, etc.) is used in the system, what is the size of the page table?
   (c). Suppose that multi level paging is used in the system, how many level do we have?
5. (15%) Please explain the following terms.
   (a). spinlock
   (b). inverted page table
   (c). resource allocation graph
   (d). remote method invocation
   (e). virtual machine
6. (20%) The first know correct software solution to the critical-section problem for two processes was developed by Dekker. The two processes, P_0 and P_1, share the following variables:

   boolean flag[2]; int turn;

   the structure of process P_i (i==0 or 1) is shown in Figure 1; the other process is P_j (j==1 or 0). The three requirements for the critical-section problem are: mutual exclusion, progress, and bounded waiting.

(a) Please explain the meaning of each of the three requirements (mutual exclusion, progress, and bounded waiting). (6 %)

(b) Please prove the algorithm satisfies all three requirements. (9 %)

(c) What the initial value for the variables flag[0] and flag[1] should be (True/False, or arbitrary) ? Why? (3%)

(d) What the initial value for the variable turn should be (0 or 1)? Can it be arbitrary one of them (0 or 1)? Why? (2%)

```
01. do {
02.    flag[i]=TRUE;
03.    While (flag[j]){
04.      if (turn == j) {
05.        flag[i]=false;
06.        while (turn == j); // do nothing
07.        flag[i]= TRUE;
08.      } // if
09.    } // while loop
10.    // critical section code here
11.    turn = j;
12.    flag[i]=FALSE;
13.    // remainder section code here
14. } while (TRUE);
```

Figure 1. the structure of process Pi in Dekker's algorithm.

Figure 1.

7. (10%) Figure 2 shows the two semaphore operations: wait() and signal().

(a) If the wait() and signal () semaphore operations are not executed atomically, then mutual exclusion may be violated. Why. (5%)

(b) What happen if we set the initial value of semaphore S to 0 ? why ? (5%)

```
// typedef int semaphore;

01. wait (semaphore S) {
02.    While S <= 0; // no-op
03.    S--;
04. }

01. signal (semaphore S) {
02.    S++;
03.}
```

Definition of wait() and signal() primitives of a semaphore S.

Figure 2.