Name: _____

CS354 - Operating Systems
Spring 2006
Midterm Exam #1 – Take Home
March 1-6, 2004

Rules:

1. All work that you do on this take home exam must be completely and entirely your own work.

2. You may neither discuss the content of this exam nor ask questions regarding this exam of anyone other than your professor until you have turned it in on Monday March 6.

3. You may neither discuss the content of this course nor ask questions about the content of this course of anyone other than your professor until you have turned in this exam on Monday March 6.

4. The single exception to rule 3 is that you may discuss project #2 or potential topics for your presentations with your partner. However, be careful to steer well clear of anything that could be construed as violating rules 1 through 3.

Any violation of these rules constitutes cheating. All cases of cheating will be submitted to the College disciplinary system.

Each question is worth 10 points for a total of 100 points.

1. Most CPUs have an instruction to disable interrupts and another to enable interrupts. When interrupts are disabled, the CPU ignores interrupts that are generated by all devices. When interrupts are enabled, the CPU processes interrupts normally. For each of these instructions, explain whether or not it needs to be a kernel mode instruction in order to ensure that a multiprogramming OS can provide protection.

2. Explain the role of the interrupt vector. What is it? What does it do? When is it used?

3. Compare the benefits and drawbacks of the monolithic and microkernel operating system architectures in terms of performance and maintainability.

4. Consider the c++ program given below:

```
int main() {
  int pid1 = fork();           // A
  int pid2 = fork();           // B

  if (pid2 == 0) {
      int pid3 = fork();       // C
      int pid4 = fork();       // D
  }
  else {
      int pid5 = fork();       // E
  }

  waitAll();       // Assume this causes a process to wait
                   // for all of its child processes to exit.
}
```

Draw the process tree that would be created if this program were run. Label each node in the tree with the letter given to the right of the fork statement that created the process.

5. What output would be generated by running the following c++ program (Note: You are not allowed to actually run the program!) :

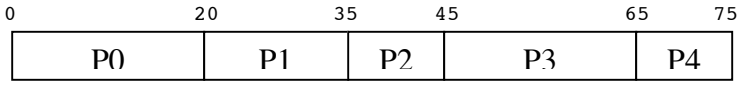
```
int main() {

      int a = 19;

      a = a + 2;
      int pid = fork();
      a = a + 9;

      if (pid != 0) {
            cout << "Parent: a=" << a << endl
            WAIT();
            cout << "Parent: a=" << a << endl
      }
      else {
            a = a + 3;
            cout << "Child: a=" << a << endl;
      }
}
```

6. In the lecture slides on "Processes", the slide titled "Process States in the OS" shows 5 process states and the transitions that exist between them. Redraw this diagram as you think it should look if it were extended to include both medium term and long term (job) scheduling. Briefly justify any additions or changes that you made.

7. Consider the table of processes shown below. The arrival time column indicates the time at which each process is inserted into the ready queue. The priority column indicates the priority of each process, with higher values indicating higher priority. The CPU burst column indicates the amount of CPU time that each process requires. Give a Gantt chart showing how these processes would be scheduled by a preemptive priority based scheduler. Be sure to indicate the time at which each context switch occurs on your Gantt chart. For the purposes of this question you can ignore the overhead for context switches.

Process ID	Arrival Time	Priority	CPU Burst
0	0	3	20
1	5	2	15
2	10	5	10
3	25	1	20
4	40	4	10

8. If the processes used in the previous problem were scheduled in FCFS order, the Gantt chart for the schedule would look as shown below:



Given the above schedule compute the following metrics:

i. Throughput
ii. Average turnaround time
iii. Average waiting time

9. Give a Gantt chart illustrating how the Linux OTHER scheduling algorithm would schedule the processes given in the table below. Assume that K = 2 and that the time quantum is 3. Also assume that if two processes in the ready queue have the same number of credits, the one that has run least recently has higher priority. Ignore the overhead for context switches.

Process ID	Arrival Time	CPU Burst
0	0	12
1	4	9
2	15	6

10. Most operating systems that support interactive processes will boost the priority of I/O bound processes and decrease the priority of CPU bound processes.

i. What are the benefits of this approach?
ii. How is this accomplished in the Solaris operating system?