Operating Systems Instructors: Zaeem Asif, Lehmia Kiran and Umar Suleman
Midterm Exam I Max Marks: **55 marks**
Fall 2012 Time Allowed: 90 min

Name: Roll no:

Note: Attempt questions in the designated areas only. State your assumptions clearly if required. There are two sections. From section II you have to attempt only 1 fixed question!

SECTION I - (Attempt All)

1. Answer these short questions **(25 marks)**

   a. What is dual mode in CPU and what type of hardware support is needed to implement it?

In order to ensure the proper execution of the OS, we must be able to distinguish between the execution of operating-system code and user-defined . At the very least, we need two separate modes of operation: user mode and kernel mode (also called supervisor mode, system mode, or privileged mode). A bit, called the mode bit is added to the hardware of the computer to indicate the current mode: kernel (0) or user (1).

   b. Give two reasons why caches are useful. What problems do they solve? What problems do they cause? if a cache can be made as large as device for which it is caching (for instance, a cache as large as disk), why not make it that large and eliminate the device?

Caches are useful when two or more components need to exchange data, and the components perform transfers at different speeds. Caches solve the transfer problem by providing a buffer of intermediate speed between the components. If the fast device finds the data it needs in the cache, it need not wait for the slower devices. The data in the cache must be kept consistent with the data in the components. If a component has a data value change, and the datum is also in the cache, the cache must also be updated. This is especially a problem on multiprocessor systems where more than one process may be accessing a datum. A component may be eliminated by an equal-sized cache, but only it: (a) the cache and the component have equivalent state-saving capacity (that is, if the component retains its data when electricity is removed, the cache must retain data as well), and (b) the cache is affordable, because faster storage tends to be more expensive.

   c. Draw a state diagram to explain *process state model. The diagram should* show the transitions are valid between the states. Also comment an event that might cause such a transition from each state to the other.

- Five states: New, Ready, Running, Blocked, Exit
- New : A process has been created but has not yet been admitted to the pool of executable processes.
- Ready : Processes that are prepared to run if given an opportunity. That is, they are not waiting on anything except the CPU availability.
- Running: The process that is currently being executed. (Assume single processor for simplicity.)
- Blocked : A process that cannot execute until a specified event such as an IO completion occurs.
- Exit: A process that has been released by OS either after normal termination or after abnormal termination (error).

d. Precisely differentiate Soft vs Hard interrupts

A hardware interrupt causes the processor to save its state of execution via a context switch, and begin execution of an interrupt handler. Software interrupts are usually implemented as instructions in the instruction set, which cause a context switch to an interrupt handler similar to a hardware interrupt.

e. Precisely differentiate User Mode vs Kernel Mode

- Kernel Mode

Name: Roll no:

- In Kernel mode:

the executing code has complete and unrestricted access to the underlying hardware. It can execute any CPU instruction and reference any memory address. Kernel mode is generally reserved for the lowest-level, most trusted functions of the operating system. Crashes in kernel mode are catastrophic; they will halt the entire PC.

- User Mode

In User mode, the executing code has no ability to directly access hardware or reference memory. Code running in user mode must delegate to system APIs to access hardware or memory. Due to the protection afforded by this sort of isolation, crashes in user mode are always recoverable. Most of the code running on your computer will execute in user mode.

f. Precisely differentiate Symmetric vs asymmetric Multiprocessing

Symmetric multiprocessing treats all processors as equals, and I/O can be processed on any CPU.
Asymmetric multiprocessing has one master CPU and the remainder CPUs are slaves. The master distributes takes among the slaves, and I/O is usually done by the master only. Multiprocessor can save money by not duplicating power supplies, housings, and peripherals. They can execute programs more quickly and can have increased reliability. They are also more complex in both hardware and software then uniprocessor systems.

g. If in RR scheduling the time quantum is very large such that the biggest CPU burst of any loaded process is less than the time quantum then the scheduling result will be equal to which other scheduling algorithm? Why?

FCFS *algorithm,*

Name: Roll no:

2. There are five processes A to E to run. Their arrival times are 0, 1, 3, 9 and 12 seconds, respectively. Their processing times are 3, 5, 2, 5 and 5 seconds, respectively. What is the turn avg waiting stime using first-come-first-served, shortest-job-first and round-robin (with 1 second quantum) scheduling? Draw Gantt charts and show proper working. [3 + 3.5 + 3.5 = 10 marks]

FCFS:

0 3 8 10
15 20

A	B	C	D	E

Avg waiting time:
0+(3-1)+(8-3)+(10-9)+(15-12)/5
=2.2

Shortest job first(Non-preemptive)

A	C	B	D	E

0 3 5 10
15 20

Avg. waiting time:
0+(3-3)+(5-1)+(10-9)+(15-12)/5
=2.6

Round Robin:

1: A runs alone. 2-3: A and B. 4-6: A, B and C, and A finishes. 7-8: B and C, and
C finishes. 9: B. 10-11: B and D, and B finishes. 12: D. 13-18: D and E, and D
finishes. 19-20: E, and E finishes.

3. (Processes, Context switch, System calls) The CDC 6600 computers could handle up to 10 I/O processes simultaneously on a single CPU using an interesting form of round-robin scheduling called processor sharing. A context switch occurred after each instruction, so instruction 1 came from process 1, instruction 2 came from process 2, etc. The context switching was done by special hardware, and the overhead was zero. If a process needed T seconds to complete in the absence of competition, what is the maximum amount of time it would need if processor sharing were used with N processes? Assume there are less than 10 processes waiting to run. **(10 marks)**

It will need NT seconds.

Name: Roll no:

Name: Roll no:

4. [CS-C students ONLY] Consider the following two threads, to be run concurrently in a shared memory (all variables are shared between the two threads) **(3+4+3 = 10 marks)**

Thread A
```
for (i=0; i<5; i++) {
      x = x + 1;
}
```

Thread B
```
for (j=0; j<5; j++) {
      x = x + 2;
}
```

Assume a single-processor system, that load and store are atomic, that x is initialized to 0 *before either thread starts*, and that x must be loaded into a register before being incremented (and stored back to memory afterwards). The following questions consider the final value of x after both threads have completed.

a. Give a *concise* proof why $x \le 15$ when both threads have completed

Each x=x+1 statement can either do nothing (if erased by Thread B) or increase x by 1.
Each x=x+2 statement can either do nothing (if erased by Thread A) or increase x by 2.
Since there are 5 of each type, and since x starts at 0, x is >= 0 and x<= (5x1)+(5x2)=15

b. Give a *concise* proof why $x \neq 1$ when both threads have completed.

Every store into x from either Thread A or B is ¥ 0, and once x becomes ¥ 0, it stays ¥ 0. *The only way for x=1 is for the last x=x+1 statement in Thread A to load a zero and store a one. However, there are at least four stores from Thread A previous to the load for the last statement, meaning that it couldn't have loaded a zero.*

c. Suppose we replace '**x = x+2**' in Thread B with an atomic double increment operation **atomicIncr2(x)** that cannot be preempted while being executed. What are all the possible final values of x? Explain.

Final values are 5, 7, 9, 11, 13, or 15. The x=x+2 statements can be "erased" by being between the load and store of an x=x+1 statement. However, since the x=x+2 statements are atomic, the x=x+1 statements can never be "erased" because the load and store phases of x=x+2 cannot be separated. Thus, our final value is at least 5 (from Thread A) with from 0 to 5 successful updates of x=x+2.

☺ GOOD LUCK ☺