CS162 Section 1

True/False

- Threads within the same process share the same heap and stack.
- *False: The heap is shared; each thread has it’s own stack.*

True/False

- Preemptive multithreading requires threads to give up the CPU using the yield() system call.
- *False: Preemptive multithreading uses interrupts to schedule context switches.*

True/False

- Despite the overhead of context switching, multithreading can provide speed-up even on a single-core cpu.
- *True: Context switch to avoid blocking on I/O.*

Short Answer

- What is the OS data structure that represents a running process?
- *Answer: PCB*

Short Answer

- What are some of the similarities and differences between interrupts and system calls? What roles do they play in preemptive and non-preemptive multithreading?

Interrupts

- An interrupt is an electronic signal to the processor from an external device indicating that an external event needs attention.
- Physical bus (line) connecting devices to the cpu.
- Alternative to polling (cpu constantly checks if an I/O device is ready).

Handling the Interrupt

- Hardware saves the PC.
- Use interrupt vector to determine what interrupt handler (aka interrupt service routine) to call.
- ISR is just a piece of code in the kernel.
- The interrupt handler saves the current process/thread to its PCB/TCB.
- ISR performs its job, often I/O.
- Call the scheduler.

Handling the Interrupt

- Prior to calling the interrupt handler, the hardware may disable (mask) certain interrupts.
- Why disable interrupts?

System Calls

- Also referred to as software interrupts or synchronous interrupts (as opposed to asynchronous hardware interrupts).
- Special instruction that causes a transition from user to kernel mode when executed.
- fork(), open(), etc.
- Handled the same as hardware interrupts: interrupt vector => ISR => scheduler

Traps/Exceptions

- Also falls into the category of software interrupts.
- Requires kernel intervention.
- Could be because of errors: divide by zero.
- Page Faults is another example.

True/False

- Every interrupt results in a transition from user to kernel mode. Hint: think Inception.
- *False: Another interrupt can occur while servicing the current interrupt.*

Concurrency Problem

- Java local variables live on the stack.
- Instance variables live on the heap.

Scenario 1

- No problem.
- Result is the same as if thread B and then thread A called add() serially.

Scenario 2

- Problem.
- As if thread B’s add() never occurred.

Scenario 3

- No problem.
- Result is the same as if thread A and then thread B called add() serially.