CSC 159 – Midterm Study Guide

Operating System Fundamentals

- A **Operating System** is a piece of software that supports the basic function of a computer (scheduling, apps, peripherals).
- The **OS** is a layer of SW often considered the most important SW in a system. It is sandwiched between the HW and application SW. It forms an interface with the underlying HW.
- The **role** of the **Operating System** is to control the allocation, operation, isolation, and arbitrate access to different resources.
  - **Manage Resources:** CPU time, memory allocation, storage, HW/peripherals, networking, security, etc.
  - **HW Abstraction:** System architecture, scheduling, inter-process communication, user interfaces, multitasking.
  - **System Services:** A common set of mechanisms for a process to interact with the OS.
- **OS Key Components:**
  - **Kernel:** The heart of the OS. Has complete control over the system.
    - Provides services and interfaces to processes.
  - **Program/Processes:** User/Kernel Processes and their compiled binary
    - A process is an instance of a program running in memory, a set of instructions for the CPU. They may operate in 1 of 2 contexts.
      - User Context: Visible to the user.
        - Processes and user programs. Can only modify and interact with its own data structures. Does not directly interact with kernel data structures.
      - Kernel Context: Internal, or part of the OS.
        - Process interrupts, kernel data structures, scheduler, etc.
  - **Drivers/HW Interfaces:** SW that manages/controls HW. May provide abstraction or interface.
    - Can be built into the kernel or separate.
    - Manages and controls a specific piece of HW (simple or complex).
  - **Memory Management:** Paged memory to virtual memory. Allocation of memory.
  - **Data Storage/Transfer:** Abstract data access (file systems).
    - Abstract read/write ops. File systems and loading from non-volatile to RAM.
    - Facilitate the exchange of data either internally or externally.
  - **User Interfaces:** Human interfaces (console, graphical), Non-human (system interconnects, etc.)
  - **Bootloader*:** Piece of SW that loads the OS into memory.
    - Often bundled with the OS but not generally considered a part of the OS.
- **Kernel Types:**
  - **Monolithic:** Everything (programs, services, etc.) in the kernel.
  - **Micro:** Programs separated from the kernel
  - **Nano:** Small embedded systems
  - **Hybrid:** A cross between 2 or more types
- **Backwards Compatibility**
  - **Real Mode:** 16-bit instruction set; limited memory can be addressed.
  - **Protected Mode:** 32-bit instruction set, segmentation, addressable virtual memory, retained 16-bit instructions.

  - **Long mode:** 64-bit instruction set, larger addressable memory, drops some 16-bit and 32-bit instructions.
  - During boot the system typically transitions: Real -> Protected -> Long (if 64bit)

SPEDE (System Programmers Educational Development environment)

- **SPEDE** is a framework that consists of a set of basic functionalities (c libraries, compiler, debugger) and a bootloader (FLAMES).
  - **SPEDE_Host:** Development environment with compilation and debugging tools.
  - **SPEDE_Target:** MS-DOS based environment with FLAMES bootloader to launch our OS.
- **Control Registers:**
  - **CR0:** State information.
  - **CR1:** Intel Reserved.
  - **CR2:** Page fault address.
  - **CR3:** Virtual memory page directory base address.
- **Segment Registers:**
  - **CS:** Code segment, indicates the code segment where your code runs.
  - **DS:** Data segment, indicates the data segment that your code may access.
  - **ES, FS, GS:** Extra segments, far pointer addressing.
  - **SS:** Stack segment, indicates the stack segment where your program is located.
- **General Registers:**
  - **EAX, AX, AH, AL:** Accumulator register - I/O, arithmetic, interrupt calls.
  - **EBX, BX, BH, BL:** Base register – Base pointer for memory access, interrupt return values.
  - **ECX, CX, CH, CL:** Counter register – Loop counter, interrupt values.
  - **EDX, DX, DH, DL:** Data register - I/O, arithmetic, interrupt calls.

Kernel Bootstrap

- **Boot Process:**
  - At boot, the CPU is operating in real mode with interrupts and the memory management unit (MMU) disabled. Nothing is loaded in RAM with memory segmentation always being enabled.
  - The BIOS or UEFI is loaded into memory and executes. This initializes and enumerates all HW/busses/devices. It also enumerates all boot devices, selects one, and loads the stage 1 bootloader via the master boot record (512 bytes).
  - Once loaded the phase 1 bootloader is executed. This code detects the phase 2 bootloader on disk and loads it into memory so it can execute.
  - Once the stage 2 bootloader is loaded the CPU transitions from real mode to protected mode. Sets up a stack, loads the kernel into memory, and transfers control to it.
    - In x86-64 it transitions into long mode
  - The kernel then initializes all data structs, variables, and memory, initializes any HW required, enables interrupts, and enters the kernel run loop, able to run user processes.
- **Kernel Data Structures:**
  - Used to maintain the state of the kernel and OS. All need to be initialized since the content of RAM is indeterministic at boot.
- **Hardware Initialization:**
  - Certain HW may need to be configured (beyond what the BIOS did) to operated. These may be enumerated and configured by the kernel.

- **Interrupts:**
  - Interrupts are configured with each interrupt type having a handler (service routine)
  - Before handling interrupts. Interrupts must be enabled, and the specific interrupt unmasked.
  - PIC = Programable interrupt controller. Sometimes multiple are used to service more interrupts. The PIC triggers an interrupt.
    - **SW Interrupt**: Must dismiss IRQ, after processing interrupt.
    - **HW Interrupt**: Do not dismiss anything. By returning we are finished processing.
- **Trapframes:**
  - Represent the state of the CPU for a process. The frame is stored on the stack and the PCB contains a pointer to it.

System Calls

- **Kernel/User Barrier:**
  - The kernel has visibility into every process. User processes only have visibility into themselves.
  - If a user process needs to interact with the kernel. It does so through a system call. These are common services (process/memory management, system info, communication, and HW interfacing) that the kernel provides to all processes.
- **System Calls:**
  - Typically triggered via software interrupts to enter the kernel context. The kernel than processes and handles the system call requested. Eventually returning to the user context, allowing the process to access the data requested within its own state.
  - Data is typically sent to the user process via the kernel placing data into the specific register in the trapframe of the calling process.
  - Data is typically received from a user process by the kernel by it accessing and reading a specific register in the trapframe of the calling process.

Quiz – OS Fundamentals & Boot Process

- In an x86 system, memory segmentations is enabled at boot.
- Memory, CPU Time, Security, and Networking are typically managed by an OS.
- Boot loaders begin operation in real mode.
- It is generally the role of the kernel to enable interrupts.
- Portions of the OS must have intimate knowledge of the underlying HW: **TRUE**
- Access to HW by SW is always performed through a driver: **TRUE**
- User applications can generally be programmed to be platform independent: **TRUE**
- An OS can be written to be platform independent: **FALSE**
- In a multi-stage bootloader design, the stage 1 boot loader initializes the OS kernel: **FALSE**
- Processes only operate in user-context: **FALSE**
- Interrupts are disabled by default at boot.
- The OS controls the allocation, operation, isolation, and arbitration of access to different resources.
- Since the BIOS performs basic initialization of HW the kernel does not need to perform additional configuration: **FALSE**
- Interrupts that are configured may not be triggered until they are unmasked: **TRUE**
- System services provide a common set of mechanisms for processes to interact with the HW, kernel, other processes, and external systems.

- The BIOS or UEFI performs the following, loads boot records to start the boot loader, enumerates boot devices, and performs basic initialization of HW to provide input/output operations.
- When the kernel starts, all data structures and variables need to be initialized since all memory is ‘0’ by default: **FALSE**
- A process will be the **only** set of instructions that a CPU executes until it has exceeded it’s time slice: **FALSE**
- Interrupt service routines are registered/programmed into which of the following? **IDT**
- Processes may operate in kernel or user context.

Quiz – Processes

- A process is a set of instructions to be executed and an instance of a program.
- The active time of a process is counted so that the process scheduler can determine when to unscheduled it.
- Data about a process resides in the Process Control Block.
- A process that is executed immediately begin executing it’s instructions: **FALSE**
- When a context switch occurs, CPU state is saved on the stack.
- When a context switch occurs from user context to kernel context, CPU registers are pushed to the stack.
- The process scheduler runs when the kernel starts, after an interrupt occurs, and within the kernel context.
- The timer interrupt is used as a mechanism to trigger the process scheduler: **TRUE**
- CPU registers describe the state of a process: **TRUE**
- In a single CPU (single core) system, we perceive that multiple processes may be running simultaneously due to time slicing and context switching.
- In a time-sharing kernel, the process scheduler will unscheduled a process when it exceeds its time slice: **TRUE**
- One process will always be scheduled so the CPU will be able to execute instructions: **TRUE**
- The CPU trapframe contains the segment registers, interrupt number, and the general registers.
- Kernel context is only entered when an interrupt occurs: **TRUE**

Quiz – System Calls

- The mechanism for processes to interact with the kernel or OS outside of it’s own process space is through system calls.
- When using CPU registers to exchange data with the kernel during a single system call, you may use the same register to send multiple pieces of data: **FALSE**
- System calls can be used to exchange data between the kernel and runtime processes and enter the kernel context to access a kernel service.
- Data can be sent or received from the kernel as part of a system call via the use of CPU registers.
- System calls can be used to expose services that the kernel provides: **TRUE**
- To be more efficient, user processes can bypass the overhead of a system call and access kernel data structures directly: **FALSE**
- Any type of data can be exchanged with the kernel when using a CPU register during a system call: **TRUE**

- A system call triggered via a software interrupt is composed of the kernel system call handler, user context API, and the interrupt service routine.
- System calls are the only mechanism in which a context switch occurs between user space and kernel space: **FALSE**
- Beyond the implementation of system calls using interrupts, dedicated instructions have been designed in modern system architectures to be more efficient.
- A system call will always immediately return execution to the process that performed the system call: **FALSE**
- In a system call, a single CPU register can be used to send data to the kernel as well as to receive data from the kernel: **TRUE**
- Which of the following allows a context switch to occur to facilitate a system call: **Software Interrupt**