CSE2431 – Lecture Midterm Review

The Ohio State University
College of Engineering

Midterm Review

Instructor: Luan Duong, Ph.D.

CSE 2431: Introduction to Operating Systems

Midterm – What to expect

- Simply put, **almost all the slides** (except those mentioned in lectures as "*optional*") covering the following topics: Computer Architecture general, Processes, Threads, CPU Scheduling, Memory structure, Dynamic Relocation, Paging, Swapping.

Midterm – What to expect

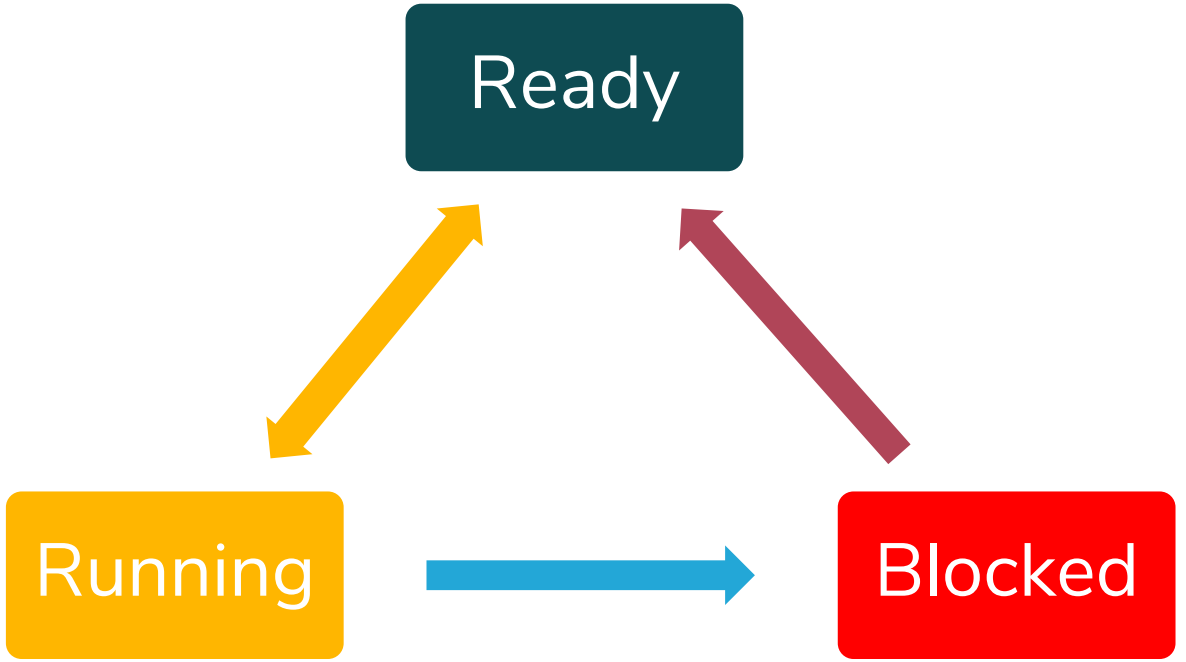
- **NOT** the History of Computer. History should be learned, but this is not a History course.
- Processes
- CPU Scheduling (Process/Job Scheduling)
- Threads (notice the difference between Processes and Threads)
- Virtual Memory:
  - Overview, Bit Operations (don't forget this, as you will need to remember how to do the bit operations in the exam by yourself)
  - Dynamic Relocation
  - Paging
  - Swaping

Process

- What is a process?
- What is time sharing?
- What does OS do to create/schedule a process? What is a context switch?
- What is a PCB? Process Control Block: storing process state, PID, program counter (PC), CPU registers info, PID of parents…
- Fork/exec/wait

Process

- Processes have different states: Running, Ready and Blocked.

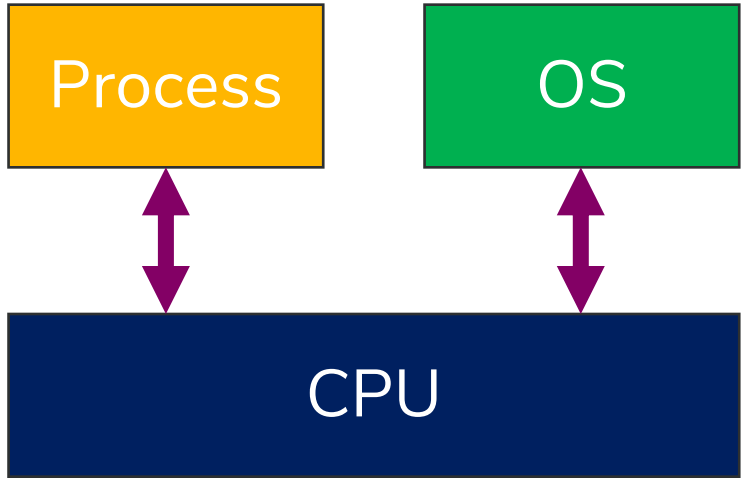


CPU Scheduling

- What is Limited Direct Execution?
  - Different between user-mode and kernel-mode
  - Trap- and return-from-trap instruction
- Difference between System call vs. Normal function call
- How to pause a process? How to resume a process? What is a **context switch**

Process scheduling

- Solution 2: Limited Direct Execution
  - A process sends its instructions directly to the CPU
  - OS collaborates with CPU to control the behavior(s) of the process.
  - Advantage: low overhead
  - Disadvantages: hard to control process behavior(s)
  - OS usually uses this approach

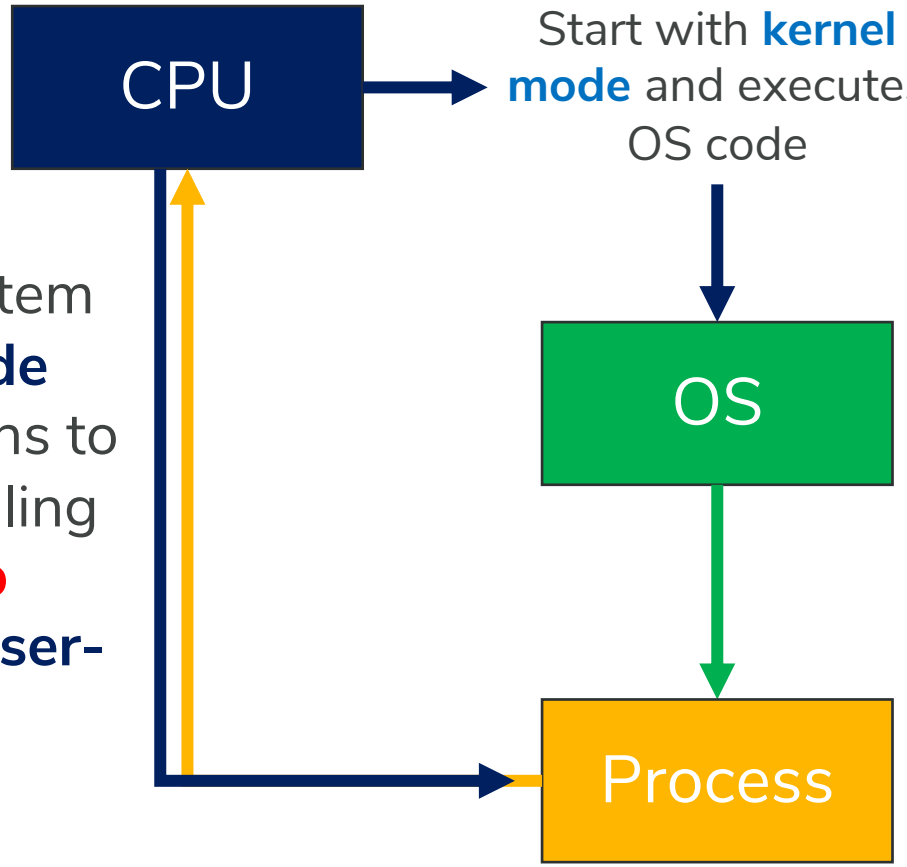


Process scheduling

- Trap- and return-from-trap instruction
  - Modern CPU provides a special trap instruction that performs the following two operations **atomically**.
    - **Switch** from user mode to kernel mode
    - **Jump** to an address defined in the trap table
  - Key idea: we allow user's code to change kernel bit, but **after** doing so, the user's code **must jump** to a **system call**!
    - Then it is safe to make trap non-restricted.
  - OS makes sure a system call **switches to user mode** before returning
    - CPU provides a special return-from-trap instruction to do that

Process scheduling

- At boot time:



- **Executes** the system call in **kernel mode**
- System call returns to **user mode** by calling **return-from-trap**
- Process runs in u**ser-mode** again

- Sets up trap table
- Performs other initializations
- Switches to **user-mode**
- Starts a program/process in user-mode

- May make system call, jumping to a location in trap table and switching to **kernel mode**, by using the trap instruction

System call vs. normal function call

System call	Function call
Must run in **kernel mode**	Can run in any mode
Made by a **trap instruction**	Made by a jump instruction
Must be registered in the **trap table**	Can be put in any location (in mem)
Comes with **two context switches**	Only with one context switch

Context Switching

- Switches CPU from one process to another
- Performed by scheduler (dispatcher) with a special hardware support (timer)
- It includes:
  - Saving the state of the old process (such as registers);
  - Loading the state of the new process;
  - Flushing memory cache;
  - Changing memory mapping [Translation Lookaside Buffer (TLB)]

Context Switch (more)

- Context switches can occur when
  - A program makes a system call
  - A timer event is triggered
  - ……
- Context switches have additional overhead
  - OS needs to save the state of current process and load the state of another process
  - Frequent context switches can be bad for performance

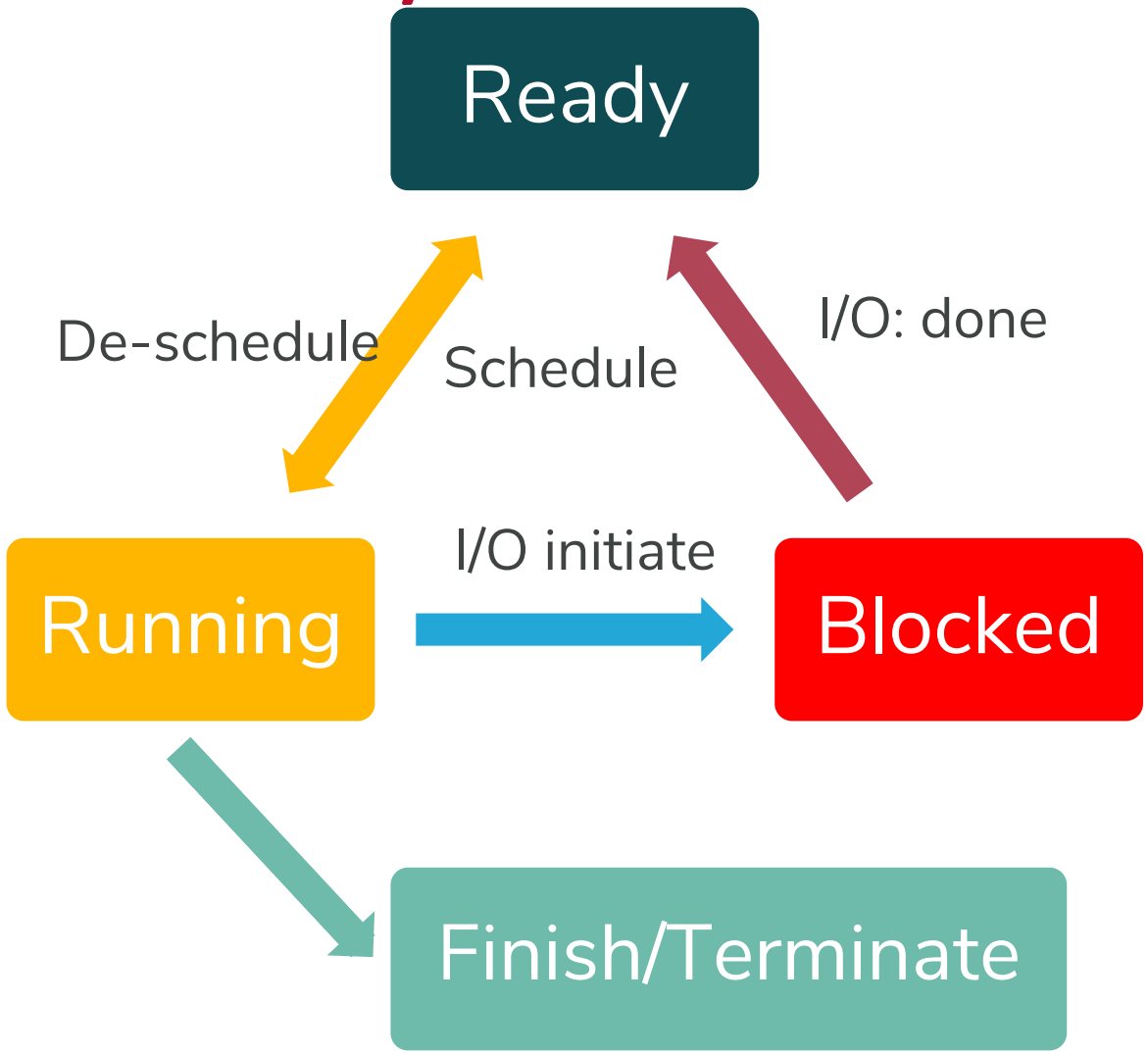
CPU Scheduling for a Process/Job

Non-preemptive scheduling:

The running process keeps the CPU until it voluntarily gives up the CPU (when? Process exits, Switches to waiting state) (No **de-schedule**)

Preemptive scheduling:

The running process can be interrupted and must release the CPU (it is forced to give up the CPU)



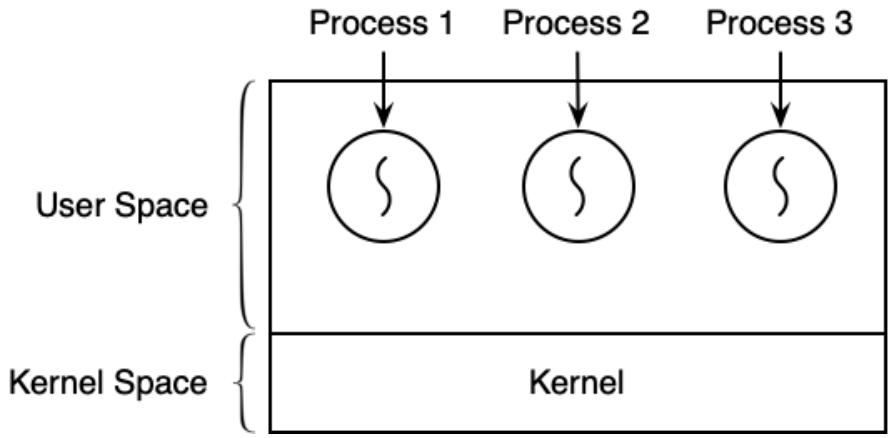
CPU Scheduling

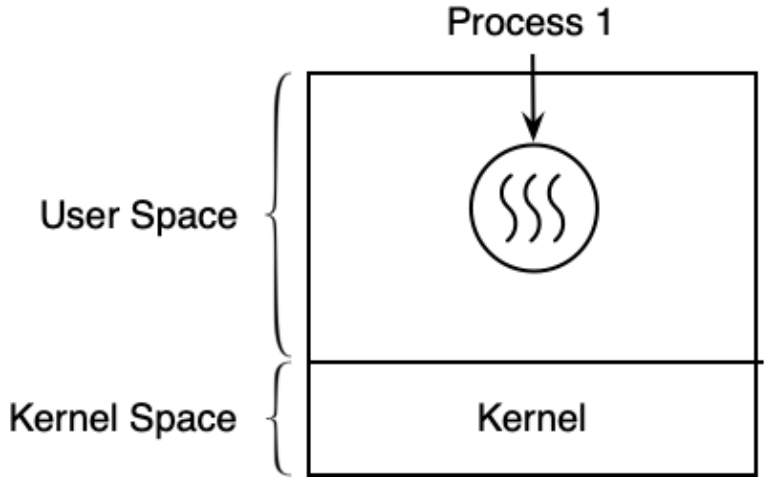
- **Type of Scheduling:**
  - Preemptive / Non-preemptive
  - Priority / Non-priority
- **Scheduling metrics**: turnaround time; response time; fairness, …
- **Scheduling Algorithms:**
  - Non-preemptive: FIFO/FCFS, SJF
  - Preemptive: Shortest-Time-to-Completion-First, Round Robin
  - Priority: Multi-level Feedback Queue (MLFQ)
  - More advanced? Multi-processor scheduling, Real-time scheduling

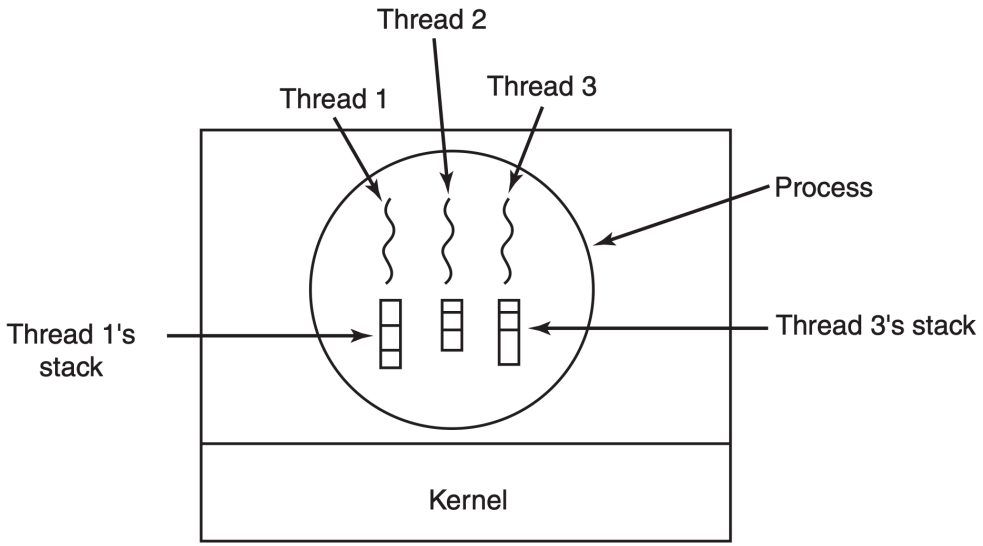
Threads

- What are threads?
- Difference between Threads and Processes?
- Different type of Threads and Thread Scheduling
  - User-level Threads
  - Kernel-level Threads
  - Hybrid Implementation
- Multi-Threading and Multi-Processing

Threads


Process 1
Process 2
Process 3
User Space
Kernel Space
Kernel


Process 1
User Space
Kernel Space
Kernel


Thread 2
Thread 1
Thread 3
Process
Thread 1's stack
Thread 3's stack
Kernel

Memory

- How is memory used by a program?
  - Jump/load/store … instructions
  - Data/Code section; stack; heap, …
- What does virtual memory mean?
  - Address space; virtual address; physical address
- malloc/free

Memory – Dynamic Relocation

- Address translation with contiguous allocation
  - Base and bounds registers
  - Internal and external fragmentation
- Segmentation
  - How to determine the correct segment?
- Free-space management
  - How to keep track of free slots?
  - Which slot to use? Best/Worst/First/Next fit

Memory – Paging

- Paging: basic idea and challenges
- How to do address translation?
- Single-array Page table:
  - Contents
  - How to calculate its size?
  - How to do translation?
  - What is its overhead?
  - What is TLB and MMU? Workflow with TLB; how to calculate TLB efficiency? How to handle TLB in a context switch?

Memory – Paging and Swapping

- Multi-level page tables
  - Basic idea
  - How to do the translation?
  - How to calculate its size?
- Swapping
  - Swap space, Present bit, and Page fault
  - Swapping policies: MIN, FIFO, LRU, Random
  - Thrashing