CS 344: Operating Systems I

01.25: Scheduling (101)

M/W 12:00 – 1:50 PM (LINC #200)

Sanghyun Hong

sanghyun.hong@oregonstate.edu

Oregon State University

SAIL
Secure AI Systems Lab

Notice

- Announcement
  - Chat GPT (#random channel)
  - Misc.: we are here to help!

Notice

- Deadlines
  - ~~(1/23 11:59 PM) Programming assignment 1~~
  - (1/30 11:59 PM) Midterm quiz 1
  - (2/06 11:59 PM) Programming assignment 2

Recap

- Part I: Threads
  - Provide abstraction
    - What is a thread?
    - How is it different from a process?
    - How does OS run threads?
  - Offer standard libraries
    - How do we create/run/kill a thread?
    - How does OS manage the thread(s) we ran?
  - Manage resources
    - (Note) We will talk about this in the “scheduling” and “synchronization” classes

Topics for today

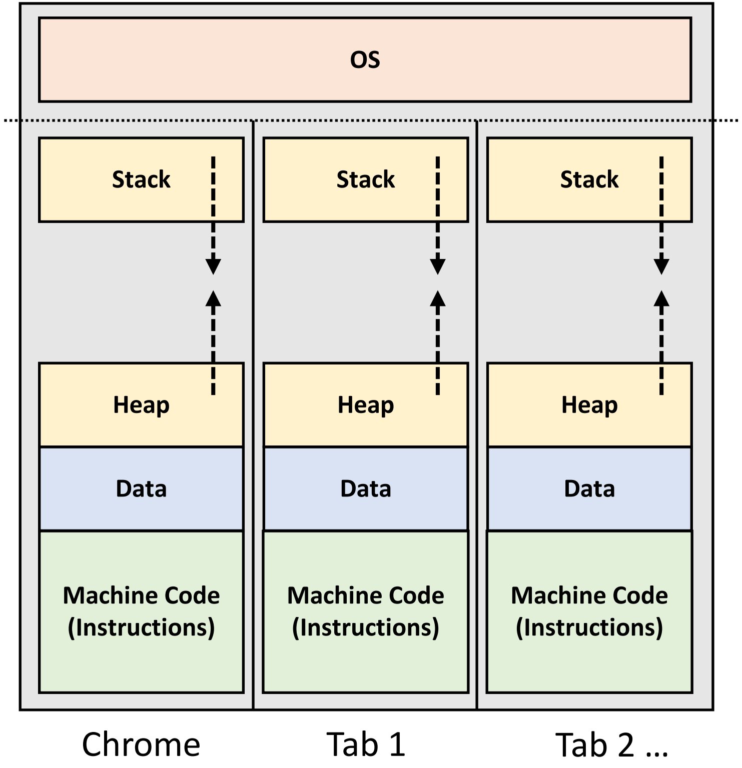
- Part I: Scheduling
  - Provide abstraction
    - What is scheduling?
    - What does OS achieve by scheduling?
  - Offer standard libraries
    - (Note) We will talk about this more in the “synchronization” lecture
  - Manage resources
    - What happens during scheduling?
    - How OS performs scheduling?
    - How OS implements this scheduling?

Problem: multiple program, yet limited processors

- Your Chrome browser:
  - Open multiple websites (tabs)
    - Tab 1: Open Canvas website
    - Tab 2: Stack Overflow
    - Tab 3: Discord website
    - … (many more 10+)
  - 4-8 CPUs (Processors)

How Can OS Address This Problem?

A Chrome tab ≈ A process


OS
Stack
Heap
Data
Machine Code (Instructions)
Stack
Heap
Data
Machine Code (Instructions)
Stack
Heap
Data
Machine Code (Instructions)
Chrome
Tab 1
Tab 2 …

Provide abstraction: scheduling

- (Process/thread) scheduling:
  - **Definition:** the action of assigning resources to perform tasks
  - **Example:** your Chrome browser
    - An OS assigns each tab (process) to one of the processors
    - The OS takes over the processor and assigns to another process
    - … (continues)

Provide abstraction: scheduling – cont'd

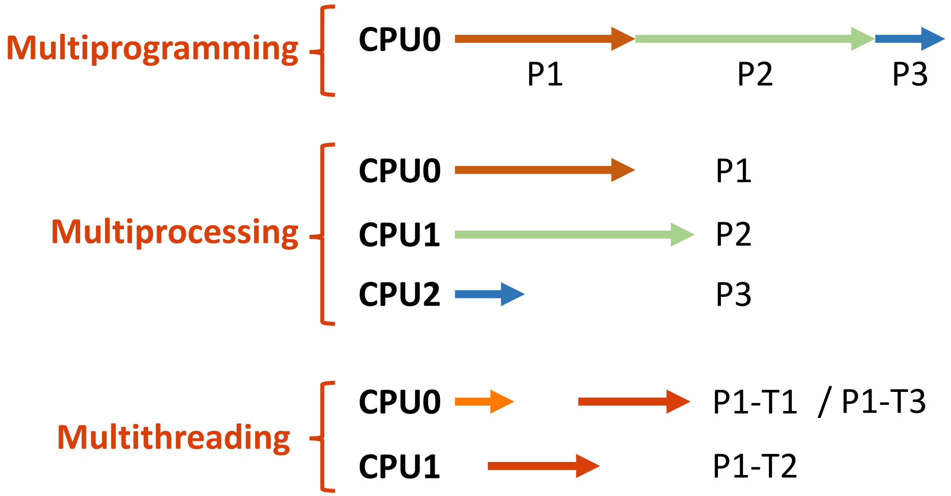
- Goal:
  - Generate illusion
  - **Illusion:**
    - Make you feel that you're running 100+ processes at the same time
    - But in truth, it's not

Topics for today

- Part I: Scheduling
  - Provide abstraction
    - What is scheduling?
    - What does OS achieve by scheduling?
  - Offer standard libraries
    - (Note) We will talk about this more in the “synchronization” lecture
  - Manage resources
    - What happens during scheduling?
    - How OS performs scheduling?
    - How OS implements this scheduling?

Preliminaries on terminology

- **Definitions:**
  - Multiprogramming vs. multi-processing vs. multi-threading
    - Multi-programming: multiple jobs (or processes)
    - Multi-processing: multiple processors (CPUs)
    - Multi-threading: multiple threads

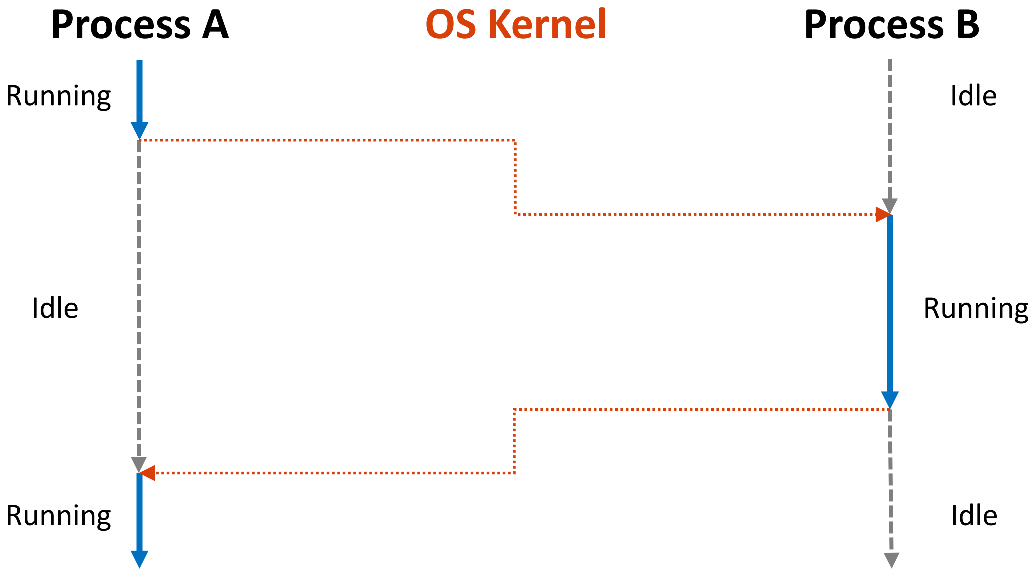


Manage resources: what happens during scheduling?

- **Context switch**
  - **Definition:** OS stores the current process’s status and loads the new process’s one
  - **Informal:** OS takes a CPU from one process and gives it to another

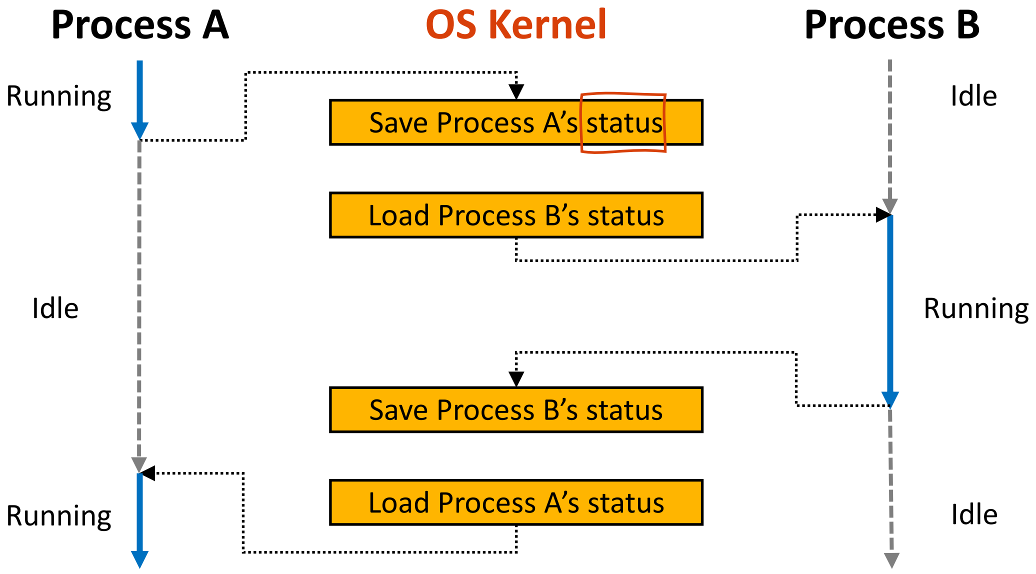
Manage resources: what happens during scheduling?

- **Context switch**
  - **Definition:** OS stores the current process's status and loads the new process's one
  - **Informal:** OS takes a CPU from one process and gives it to another



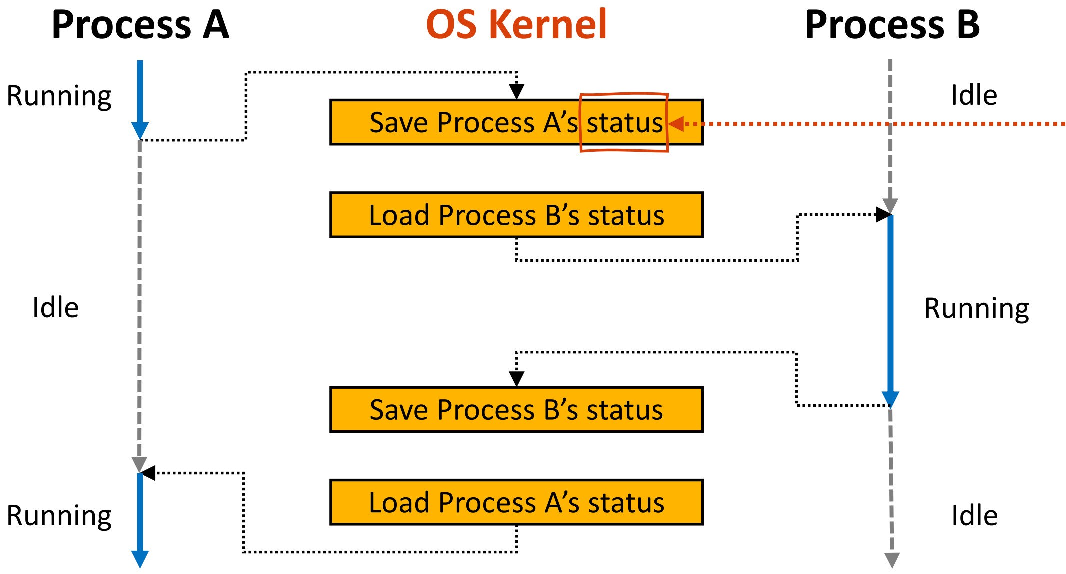
Manage resources: what happens during scheduling?

- **Context switch**
  - **Definition:** OS stores the current process's status and loads the new process's one
  - **Informal:** OS takes a CPU from one process and gives it to another



Manage resources: what happens during scheduling?

- **Context switch**
  - **Definition:** OS stores the current process's status and loads the new process's one
  - **Informal:** OS takes a CPU from one process and gives it to another



Recall: Process control block

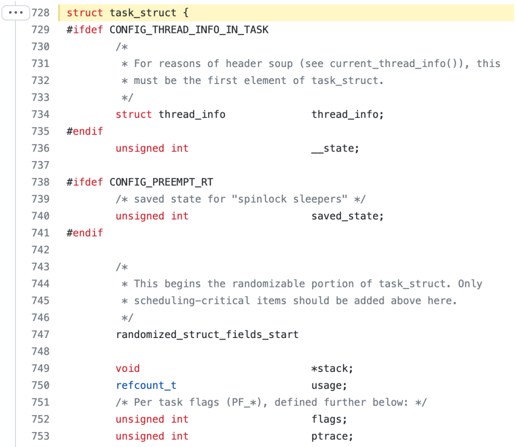
A structure in OS that contains a set of information required to run a process on a CPU. Recall that Linux has *task_struct*.

- CPU#
- Program counter
- Instruction pointer
- Heap/stack pointer
- Process state [!]
- …

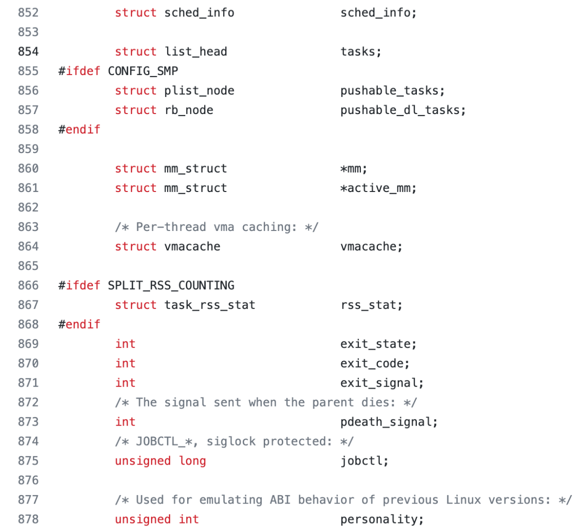
Revisit: process context

- (Linux) has the process context
  - Code
    - Program counter
    - Instruction pointer
  - Stack and heap
    - Stack pointer
    - Heap pointer
  - Running context
    - Process state (ID, …)
    - Execution flags
    - CPU # to run
    - (OS II) Scheduling policy
    - (OS II) Mem. virtualization
  - …

Process Context: A set of information that OS requires to run a process on a CPU, different from CPU vendors (ex. In Linux, it's defined as *task_struct*, Link)



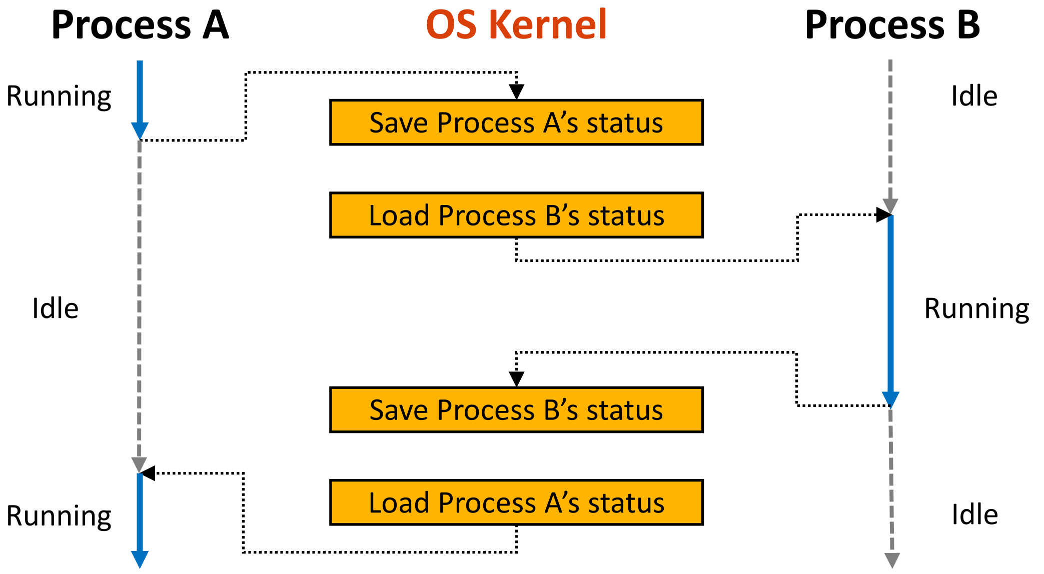
```
struct task_struct {
#ifdef CONFIG_THREAD_INFO_IN_TASK
        /*
         * For reasons of header soup (see current_thread_info()), this
         * must be the first element of task_struct.
         */
        struct thread_info              thread_info;
#endif
        unsigned int                    __state;

#ifdef CONFIG_PREEMPT_RT
        /* saved state for "spinlock sleepers" */
        unsigned int                    saved_state;
#endif

        /*
         * This begins the randomizable portion of task_struct. Only
         * scheduling-critical items should be added above here.
         */
        randomized_struct_fields_start

        void                            *stack;
        refcount_t                      usage;
        /* Per task flags (PF_*), defined further below: */
        unsigned int                    flags;
        unsigned int                    ptrace;
```



```
        struct sched_info               sched_info;

        struct list_head                tasks;
#ifdef CONFIG_SMP
        struct plist_node               pushable_tasks;
        struct rb_node                  pushable_dl_tasks;
#endif

        struct mm_struct                *mm;
        struct mm_struct                *active_mm;

        /* Per-thread vma caching: */
        struct vmacache                 vmacache;

#ifdef SPLIT_RSS_COUNTING
        struct task_rss_stat            rss_stat;
#endif
        int                             exit_state;
        int                             exit_code;
        int                             exit_signal;
        /* The signal sent when the parent dies: */
        int                             pdeath_signal;
        /* JOBCTL_*, siglock protected: */
        unsigned long                   jobctl;

        /* Used for emulating ABI behavior of previous Linux versions: */
        unsigned int                    personality;
```

Manage resources: what happens during scheduling?

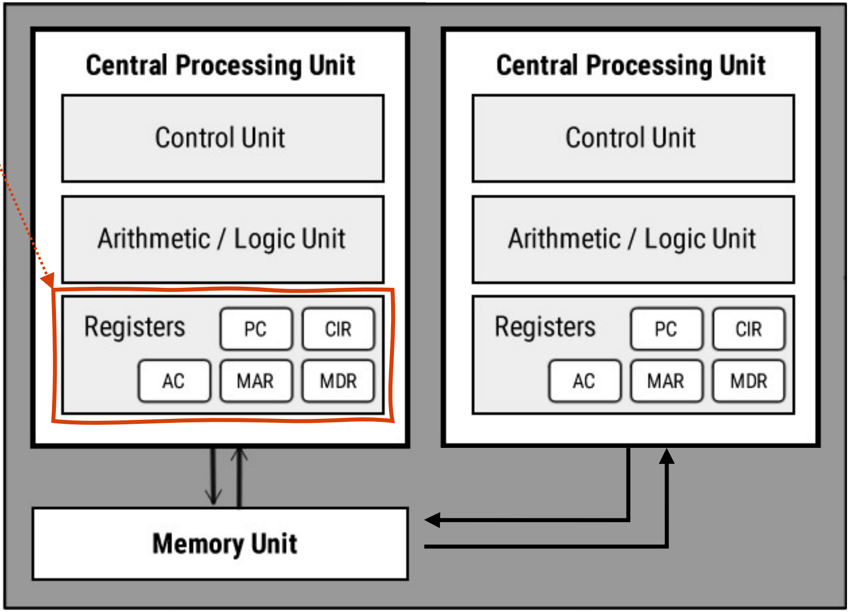
- **Context switch**
  - **Definition:** OS stores the current process's status and loads the new process's one
  - **Informal:** OS takes a CPU from one process and gives it to another



Revisit: Process Context to a CPU

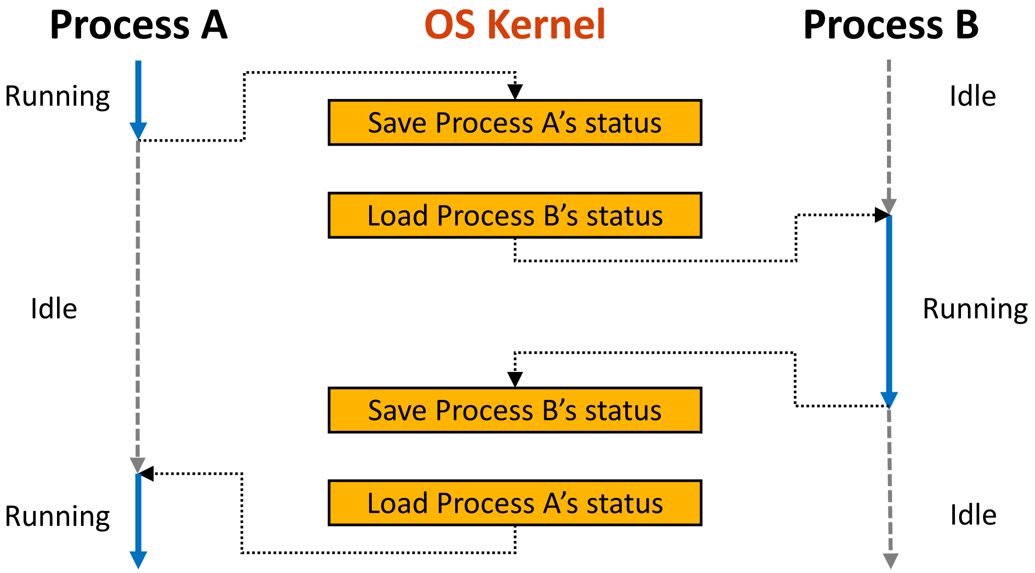
- (Linux) has the process context
  - Code
    - Program counter
    - Instruction pointer
  - Stack and heap
    - Stack pointer
    - Heap pointer
  - Running context
    - Process state (ID, …)
    - Execution flags
    - CPU # to run
    - (OS II) Scheduling policy
    - (OS II) Mem. virtualization
  - …

Process Context: A set of information that OS requires to run a process on a CPU, different from CPU vendors (ex. In Linux, it's defined as *task_struct*, Link)



Manage resources: what happens during scheduling?

- **Context switch**
  - **Definition:** OS stores the current process's status and loads the new process's one
  - **Informal:** OS takes a CPU from one process and gives it to another



Process A

OS Kernel

Process B

Running

Idle

Save Process A's status

Load Process B's status

Idle

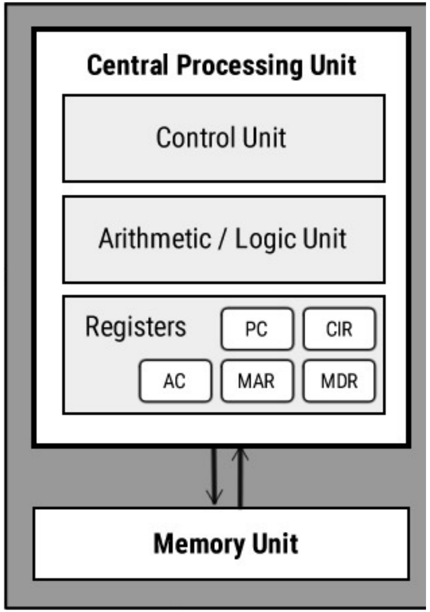
Running

Save Process B's status

Load Process A's status

Running

Idle



Manage resources: what happens during scheduling?

- **Context switch**
  - **Definition:** OS stores the current process's status and loads the new process's one
  - **Informal:** OS takes a CPU from one process and gives it to another

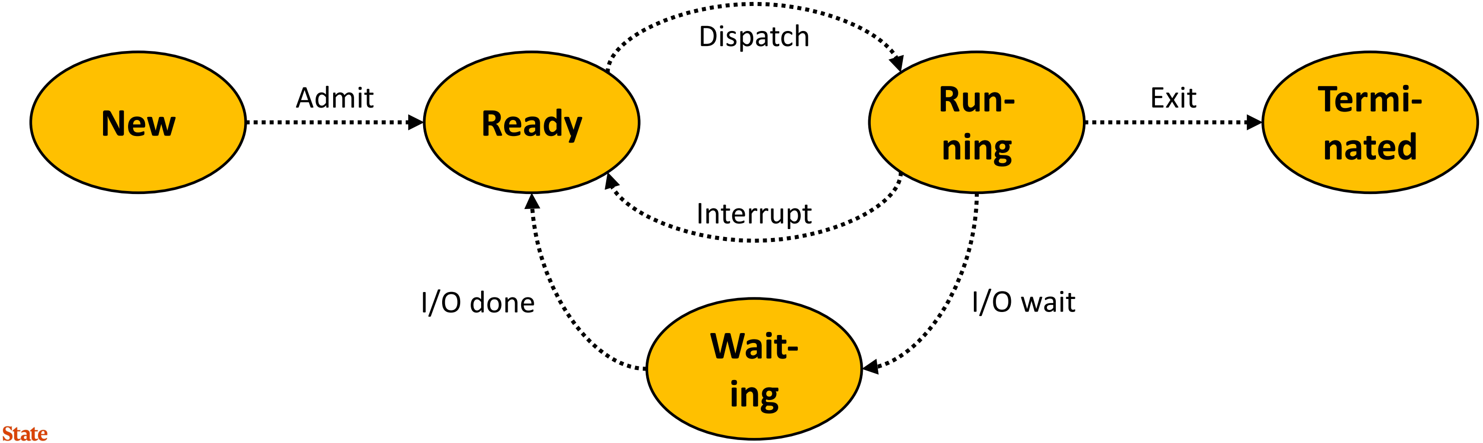
- No free lunch
  - Context switching takes $\sim 5\ \mu s$ on average
  - OS typically runs 100+ processes
  - Too many context switching makes a system unable to respond…

Manage resources: what happens during scheduling?

- A process can have **five states**:
  - **New:** a process (or thread) is being created (by fork())
  - **Ready:** the process is waiting to run
  - **Running:** the process is running on a CPU(or CPUs)
  - **Waiting:** the process is waiting for some events to occur (*e.g.*, a data loaded from storage)
  - **Terminated:** the process has finished execution; waiting for removal

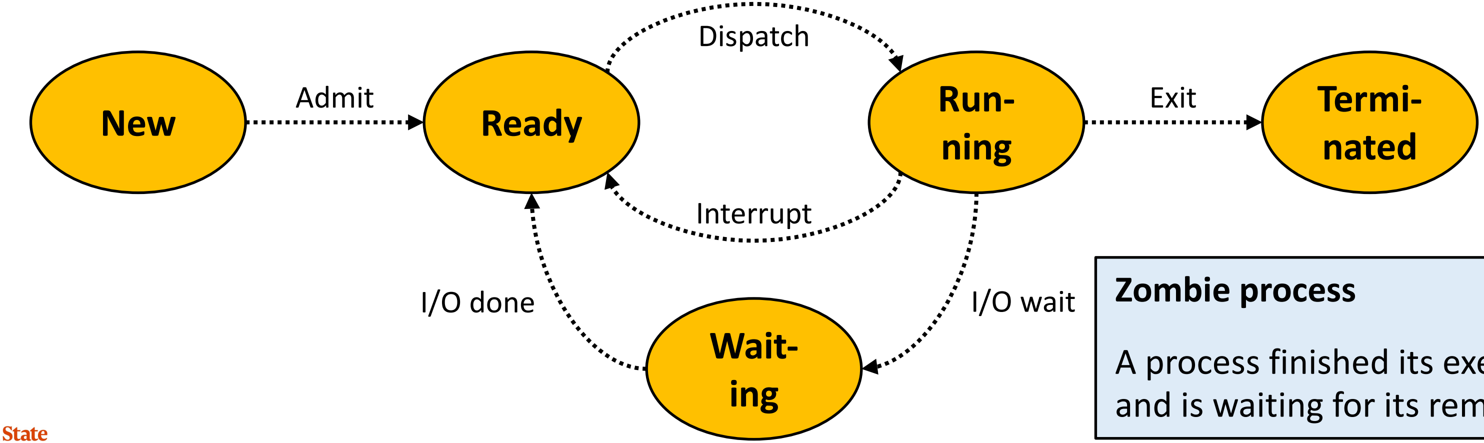
Manage resources: what happens during scheduling?

- A process can have **five states**:
  - **New:** a process (or thread) is being created (by fork())
  - **Ready:** the process is waiting to run
  - **Running:** the process is running on a CPU(or CPUs)
  - **Waiting:** the process is waiting for some events to occur (*e.g.*, a data loaded from storage)
  - **Terminated:** the process has finished execution; waiting for removal
- State transition (life cycle):



Manage resources: what happens during scheduling?

- A process can have **five states**:
  - **New:** a process (or thread) is being created (by fork())
  - **Ready:** the process is waiting to run
  - **Running:** the process is running on a CPU(or CPUs)
  - **Waiting:** the process is waiting for some events to occur (*e.g.*, a data loaded from storage)
  - **Terminated:** the process has finished execution; waiting for removal
- State transition (life cycle):



Zombie process

A process finished its execution and is waiting for its removal

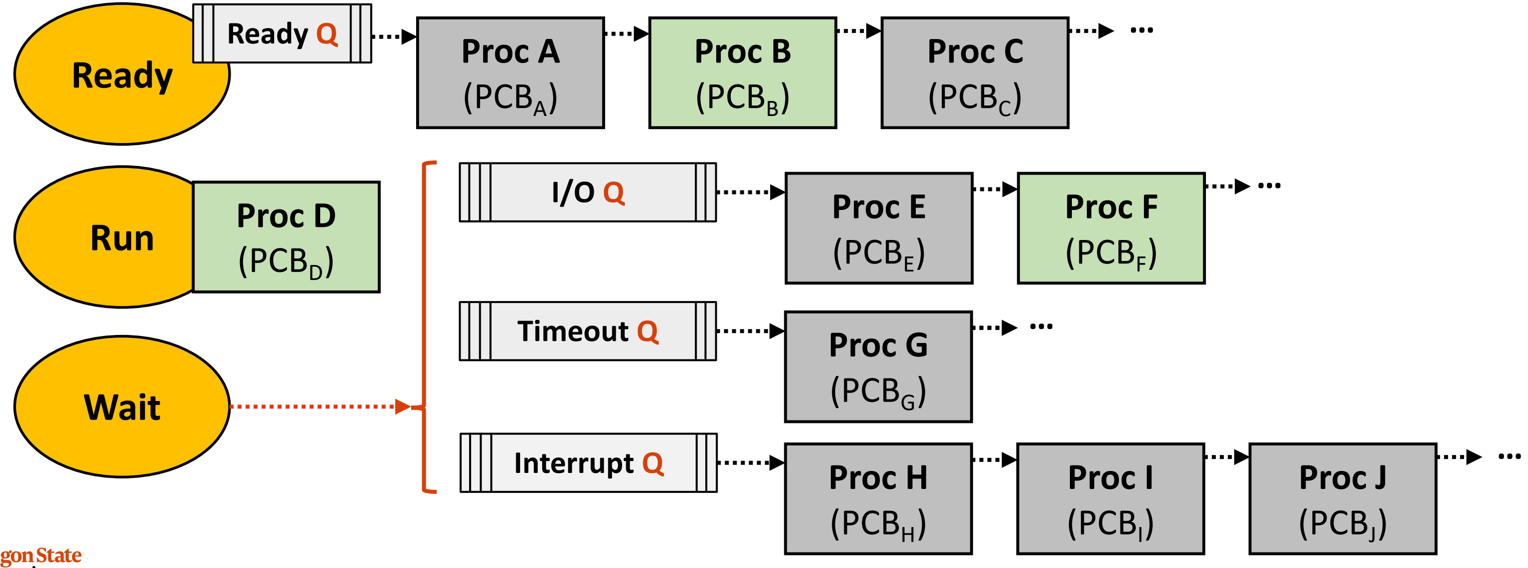
Topics for today

- Part I: Scheduling
  - Provide abstraction
    - What is scheduling?
    - What does OS achieve by scheduling?
  - Offer standard libraries
    - (Note) We will talk about this more in the "synchronization" lecture
  - Manage resources
    - What happens during scheduling?
    - How OS performs scheduling?
    - How OS implements this scheduling?

Manage resources: how OS performs scheduling?

- **Scheduling**
  - **Definition:** an OS activity that schedules processes in different states
  - **Note:** OS implements queues to hold multiple processes in the same state

- **Illustration (single CPU)**



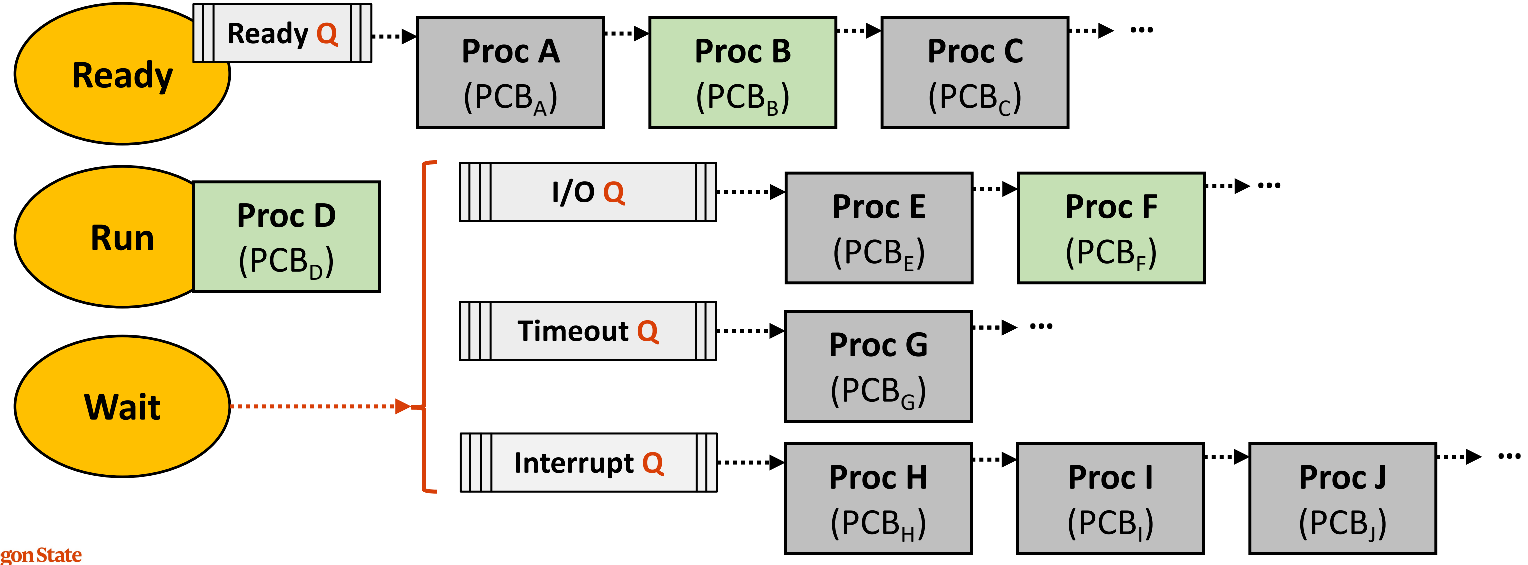
Manage resources: how OS performs scheduling?

- **Scheduling**
  - **Definition:** an OS activity that schedules processes in different st
  - **Note:** OS implements queues to hold multiple processes in the sa

Illustrated Example

1. Kicks out Proc D (timeout)
2. Runs Proc B
3. Puts Proc F in the ready Q
   (I/O has done, in this case)

- **Illustration (single CPU)**



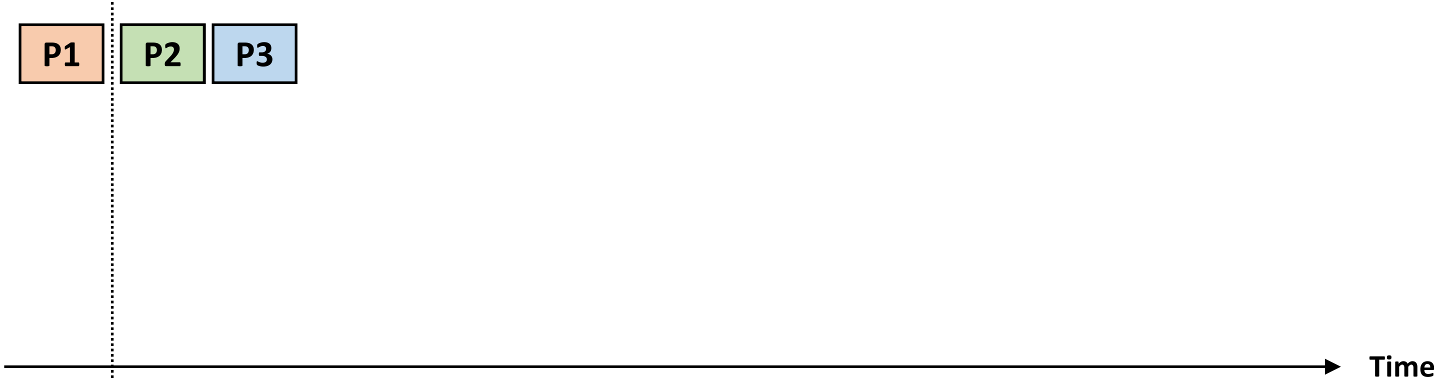
Scheduling example: high-level view

- 3 Processes in Chrome:
  - **P1:** Download movies
  - **P2:** Open Canvas
  - **P3:** Search StackOverflow
- Example
  - **New** :
  - **Ready**:
  - **Run** :
  - **Wait** :
  - **Term..**:

  - **Scenario:** open a website for downloading movies

Scheduling example: high-level view

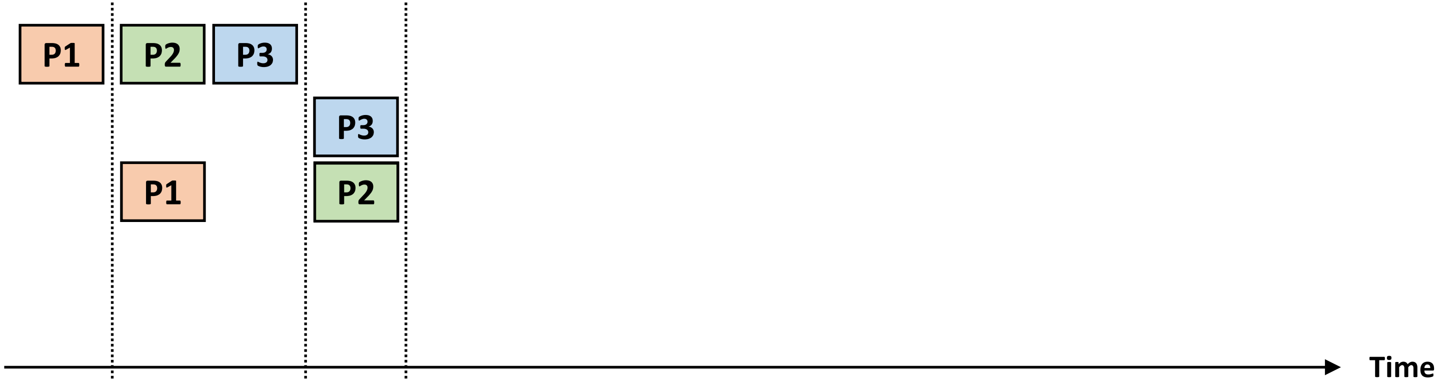
- 3 Processes in Chrome:
  - **P1:** Download movies
  - **P2:** Open Canvas
  - **P3:** Search StackOverflow
- Example
  - **New** :
  - **Ready**:
  - **Run** :
  - **Wait** :
  - **Term..**:



  - **Scenario:** the website opened and open two other websites

Scheduling example: high-level view

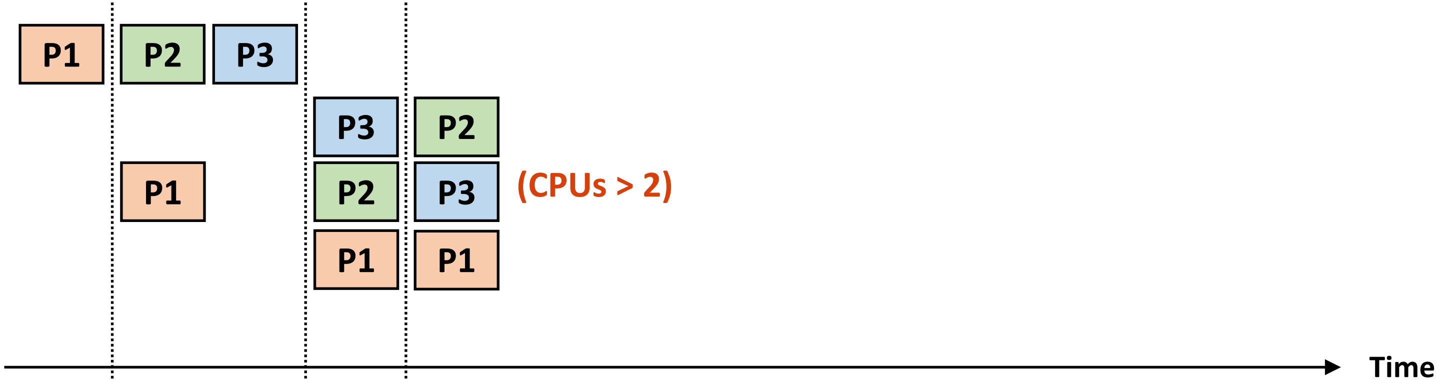
- 3 Processes in Chrome:
  - **P1:** Download movies
  - **P2:** Open Canvas
  - **P3:** Search StackOverflow
- Example
  - **New** :
  - **Ready**:
  - **Run** :
  - **Wait** :
  - **Term..**:



  - **Scenario:** downloads started, and you focus on Canvas

Scheduling example: high-level view

- 3 Processes in Chrome:
  - **P1:** Download movies
  - **P2:** Open Canvas
  - **P3:** Search StackOverflow
- Example
  - **New** :
  - **Ready**:
  - **Run** :
  - **Wait** :
  - **Term..**:



  - **Scenario:** while downloading, you start searching StackOverflow

Scheduling example: high-level view

- 3 Processes in Chrome:
  - **P1:** Download movies
  - **P2:** Open Canvas
  - **P3:** Search StackOverflow
- Example

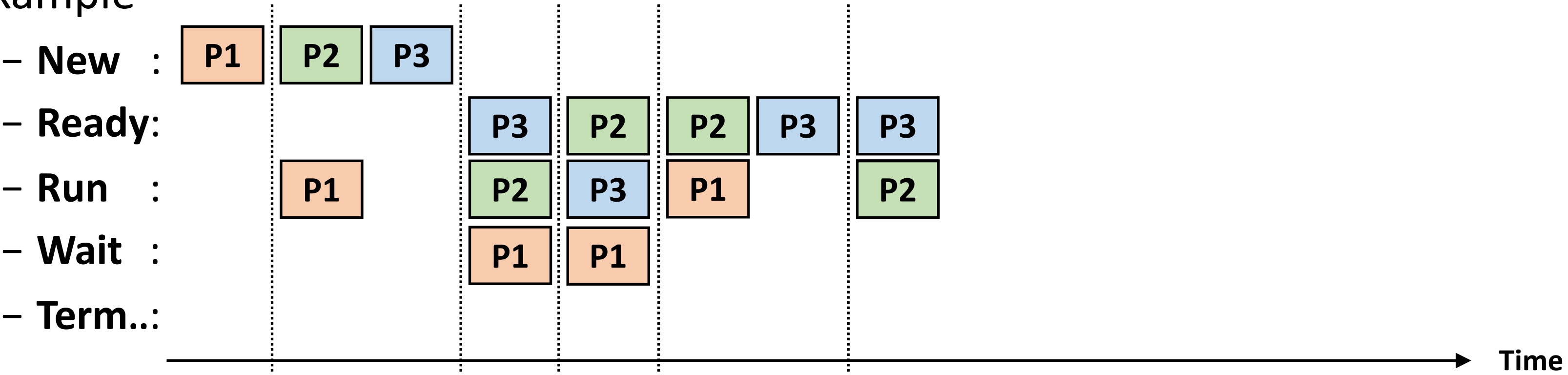
– **New** :	P1	P2 P3			
– **Ready**:			P3	P2	P2 P3
– **Run** :		P1	P2	P3	
– **Wait** :			P1	P1	P1
– **Term..**:					

Time

  - **Scenario:** downloading movies are done

Scheduling example: high-level view

- 3 Processes in Chrome:
  - **P1:** Download movies
  - **P2:** Open Canvas
  - **P3:** Search StackOverflow
- Example



– **New** :	P1	P2 P3				
– **Ready**:			P3	P2	P2 P3	P3
– **Run** :		P1	P2	P3	P1	P2
– **Wait** :			P1	P1		
– **Term..**:						

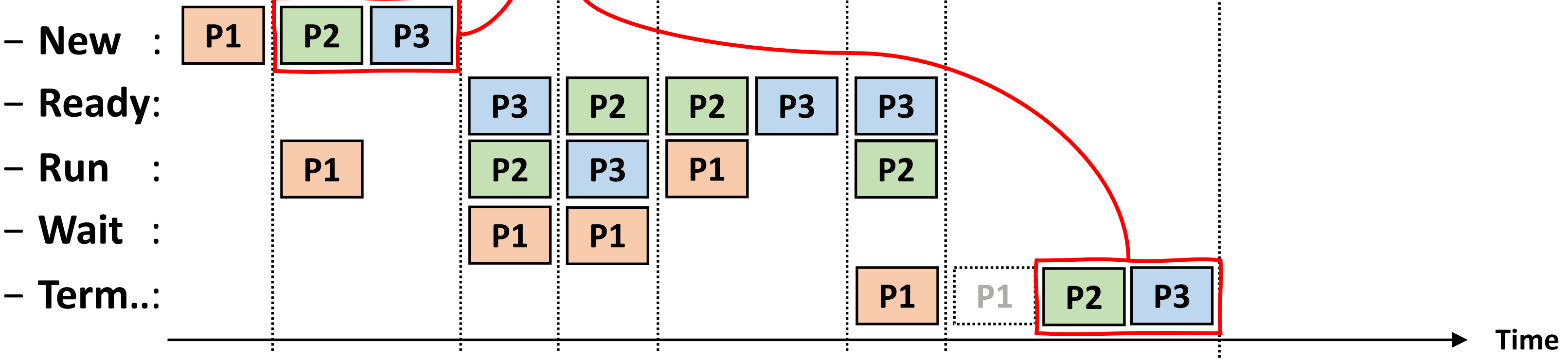
  - **Scenario:** close the download tab, and keep looking at Canvas

Scheduling example: high-level view

- 3 Processes in Chrome:
  - **P1:** Download movies
  - **P2:** Open Canvas
  - **P3:** Search StackOverflow
- Example

Linked list: OS keeps a list of processes for each state by using the queue structure. Based on a policy, OS searches the queue and find the process with the highest priority

Zombie process(es): A list of processes whose job has already terminated (exited) but has not removed from the process list. One case is that a parent process wait for a child’s exit status.



  - **Scenario:** close the other two tabs to go to bed

Topics for today

- Part I: Scheduling
  - Provide abstraction
    - What is scheduling?
    - What does OS achieve by scheduling?
  - Offer standard libraries
    - (Note) We will talk about this more in the “synchronization” lecture
  - Manage resources
    - What happens during scheduling?
    - How OS performs scheduling?
    - How OS implements this scheduling?

Manage resources: how OS implements scheduling?

- **(OS) Scheduler:**
  - **Definition:** An OS task (process) that manages the process scheduling activity

Manage resources: how OS implements scheduling?

- **(OS) Scheduler:**
  - **Definition:** An OS task (process) that manages the process scheduling activity

- **Implementation**

```
while ( <some condition,
            but eventually will be infinite>) {

    RunProcess( curProc );
    newProc = chooseNextProc();
    saveCurrentProc( curProc );
    LoadNextState( newProc );

}
```

  - It is also a process (an *infinite* loop)
  - The scheduler process terminates if we *stop* (turn-off) a computer
  - Example mechanisms that trigger scheduling, *e.g.*, yield and interrupt

Manage resources: how OS implements scheduling?

- **Problem:** how to choose a next process?
  - FIFO (first come, first served)
  - LIFO (last come, first served)
  - Shortest-job first (do a short-period job first)
  - Priority-based (do an important job first)
  - … (It’s an open-problem; You’ll learn in OS II)

Process Scheduling

- Multiple objectives:
  - **Fairness**: no monopoly
  - **Priority**: consider importance of a process
  - **Deadlines**: a process should be done before/by the time *T*
  - **Throughput**: maximize the number of tasks done
  - **Efficiency**: minimize the scheduling overheads
  - …

- **No silver bullet:**
  - Depends on the objectives (ex. NASA’s Perseverance)
  - Oftentimes, the objectives are in conflicts (ex. Priority vs. Fairness)
  - … (You’ll learn in OS II)

Topics for today

- Part I: Scheduling
  - Provide abstraction
    - What is scheduling?
    - What does OS achieve by scheduling?
  - Offer standard libraries
    - (Note) We will talk about this more in the “synchronization” lecture
  - Manage resources
    - What happens during scheduling?
    - How OS performs scheduling?
    - How OS implements this scheduling?

Thank You!

M/W 12:00 – 1:50 PM (LINC #200)

Sanghyun Hong

sanghyun.hong@oregonstate.edu

Oregon State University

SAIL
Secure AI Systems Lab