Operating Systems

Lecture 2.1 - Processes and Threads

Golestan University

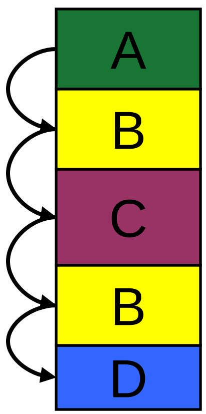
Hossein Momeni
momeni@iust.ac.ir

What is a process?

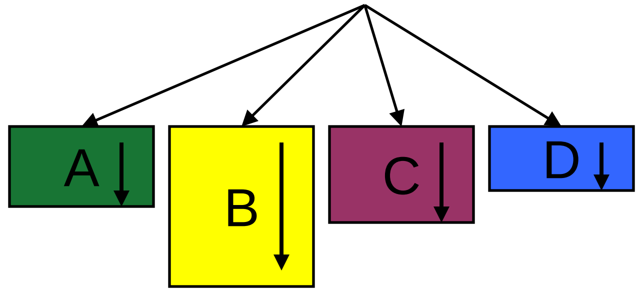
- Process= Program, Input/Output and State
- Program state
  - Program counter (current location in the code)
  - CPU registers
  - Stack pointer
- Only one process can be running in the CPU at any given time!
  - All executable programs are organized to multi ordinal process
  - Each process have a virtual CPU (CPU switch)

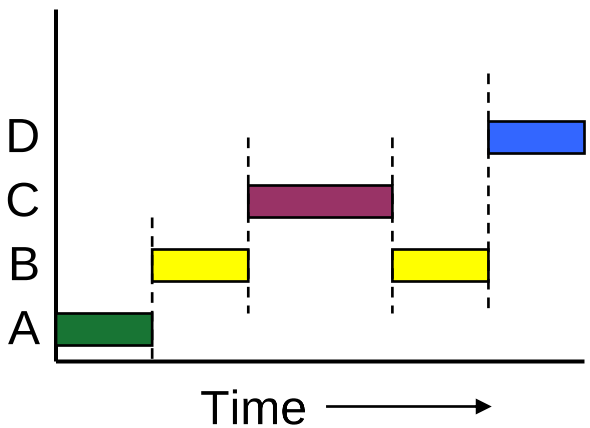
The process model

Single PC
(CPU's point of view)



Multiple PCs
(process point of view)





- Multiprogramming of four programs
- Conceptual model
  - 4 independent processes
  - Processes run sequentially
- Only one program active at any instant!
  - That instant can be very short…

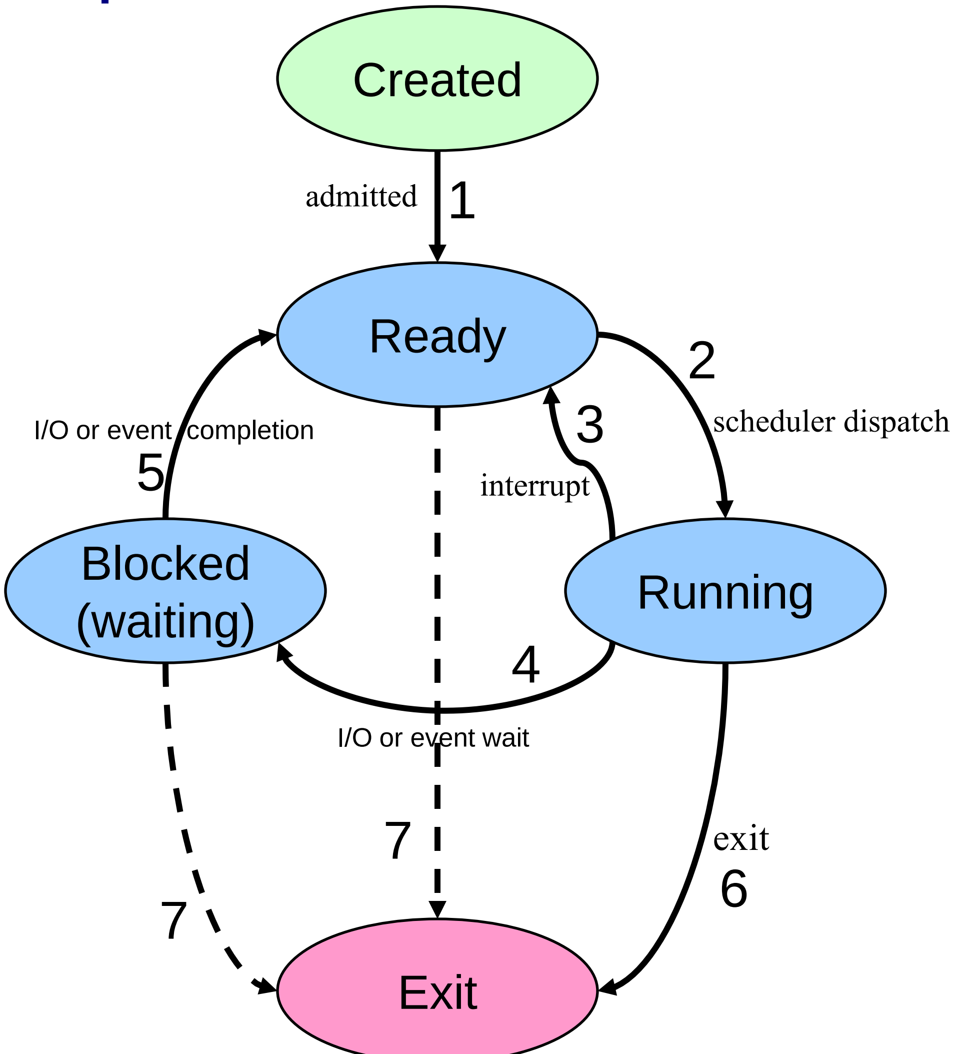
When is a process created?

- Processes can be created in two ways
  - System initialization: one or more processes created when the OS starts up
  - Execution of a process creation system call
- System calls can come from
  - User request to create a new process
  - Execute a batch job

When do processes end?

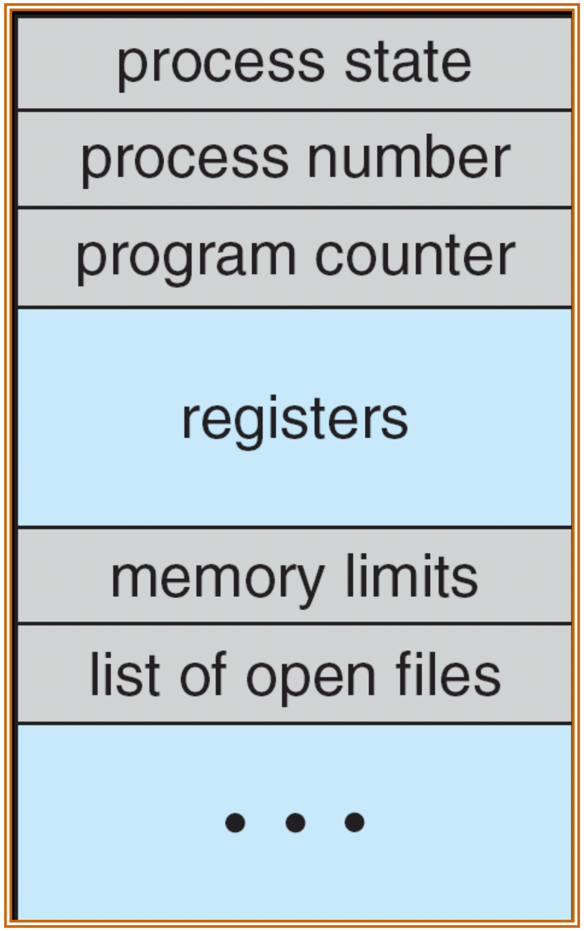
- Conditions that terminate processes can be
  - Voluntary
    - Normal exit
    - Error exit
  - Involuntary
    - Fatal error
    - Killed by another process

Process states

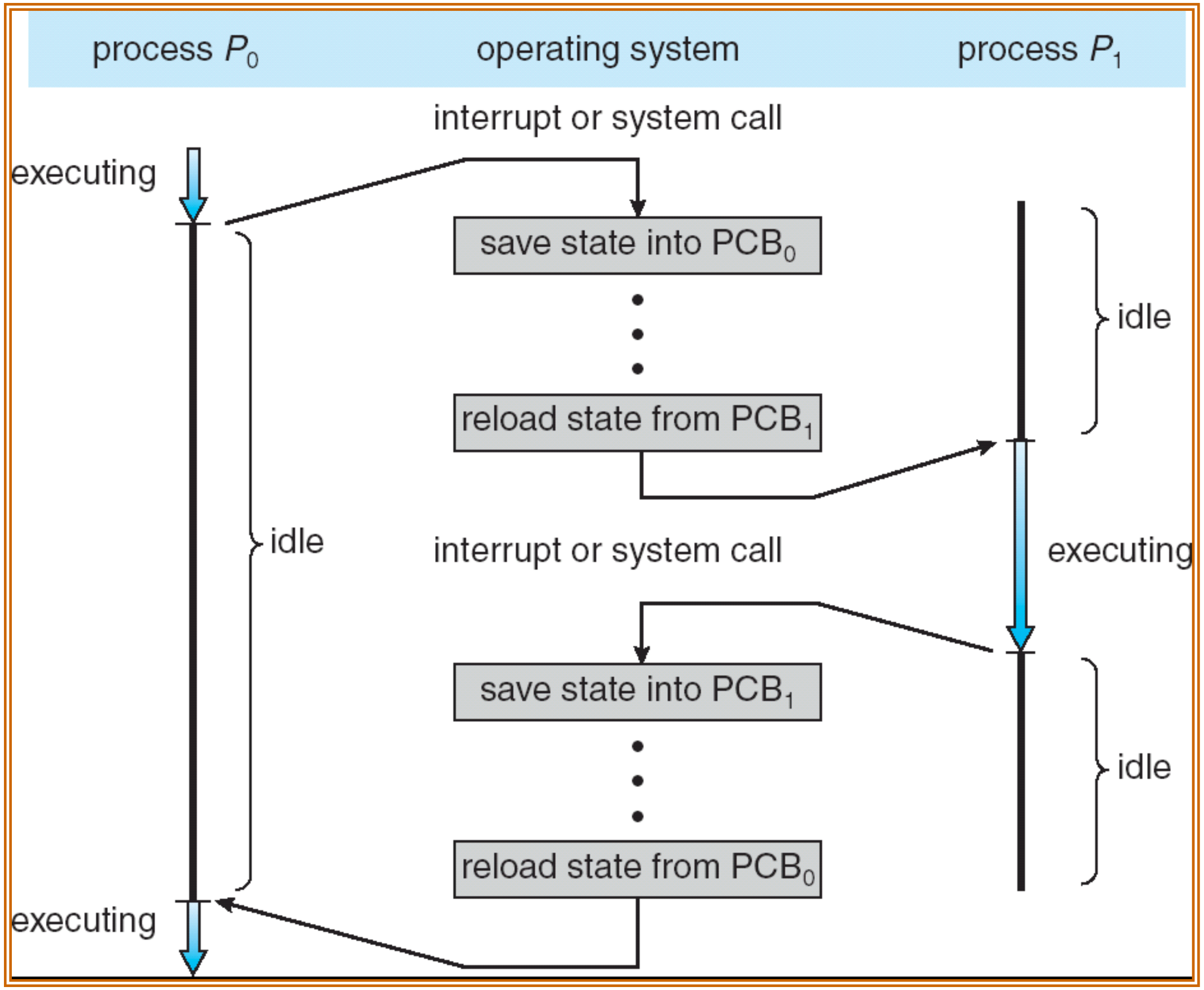


- Process in one of 5 states
  - Created
  - Ready
  - Running
  - Blocked
  - Exit
- Transitions between states
  - 1 - Process enters ready queue
  - 2 - Scheduler picks this process
  - 3 - Scheduler picks a different process
  - 4 - Process waits for event (such as I/O)
  - 5 - Event occurs
  - 6 - Process exits
  - 7 - Process ended by another process

Process Implementation: Process Control Block (PCB)



CPU Switch From Process to Process



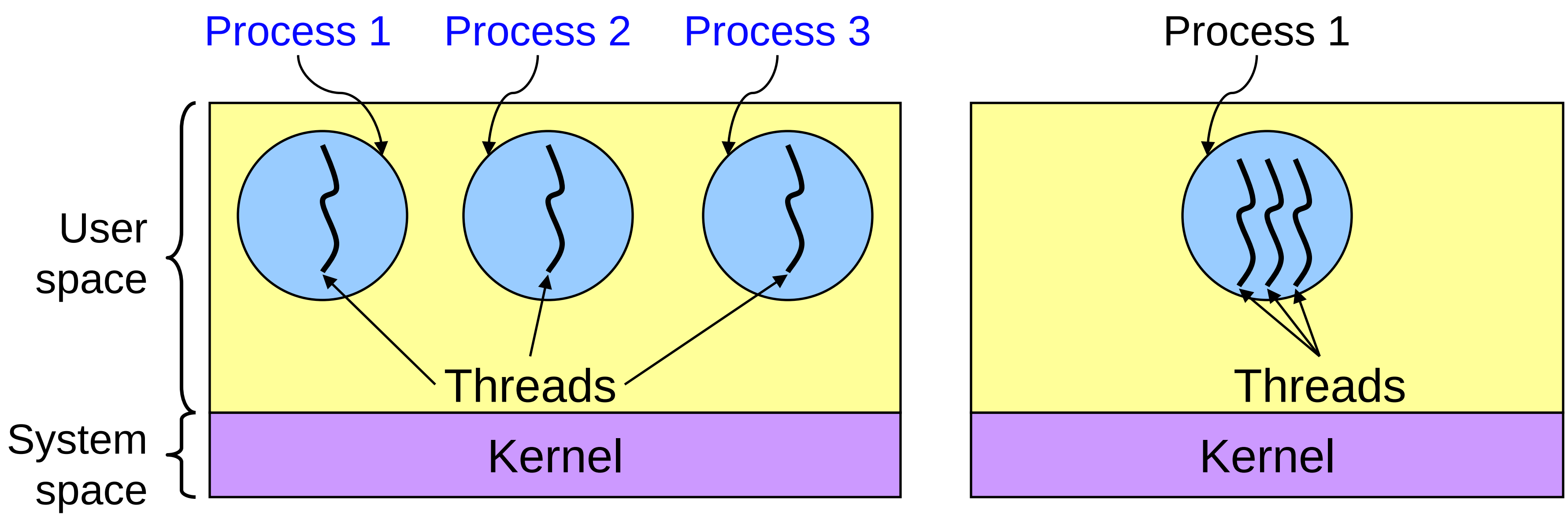
Process Scheduling Queues

- **Job queue** – set of all processes in the system
- **Ready queue** – set of all processes residing in main memory, ready and waiting to execute
- **Device queues** – set of processes waiting for an I/O device
- Processes migrate among the various queues

Threads

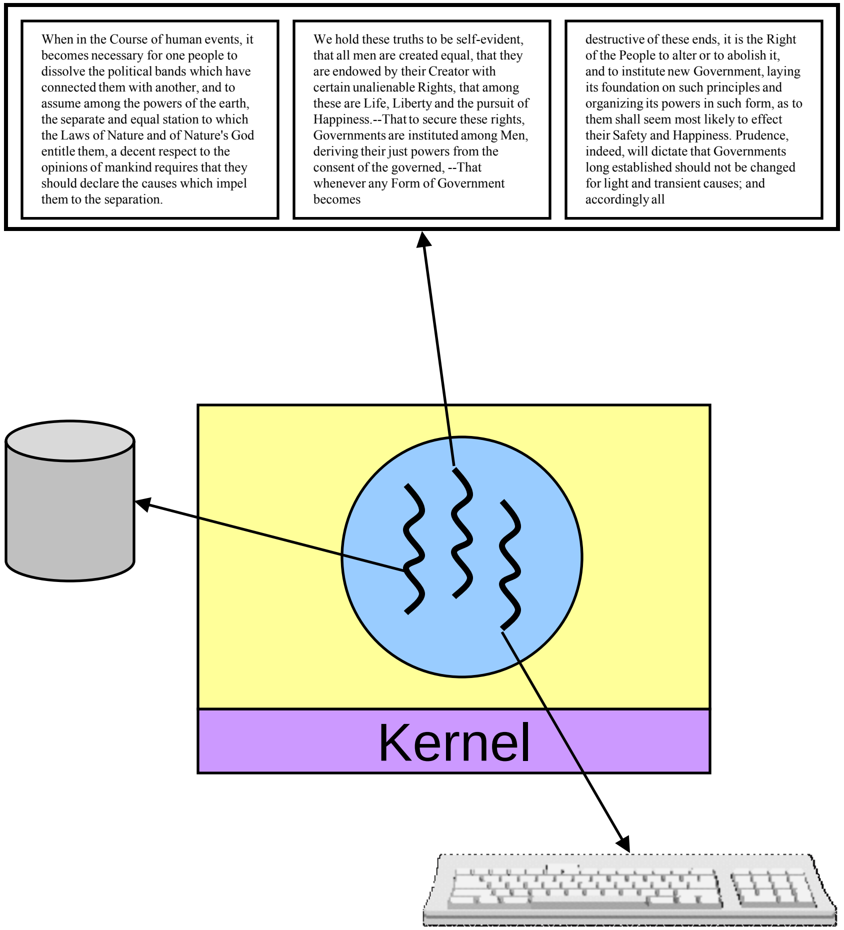
Threads: “processes” sharing memory

- Process == address space
- Thread == program counter / stream of instructions
- Thread == Lightweight process
- Two examples
  - Three processes, each with one thread
  - One process with three threads

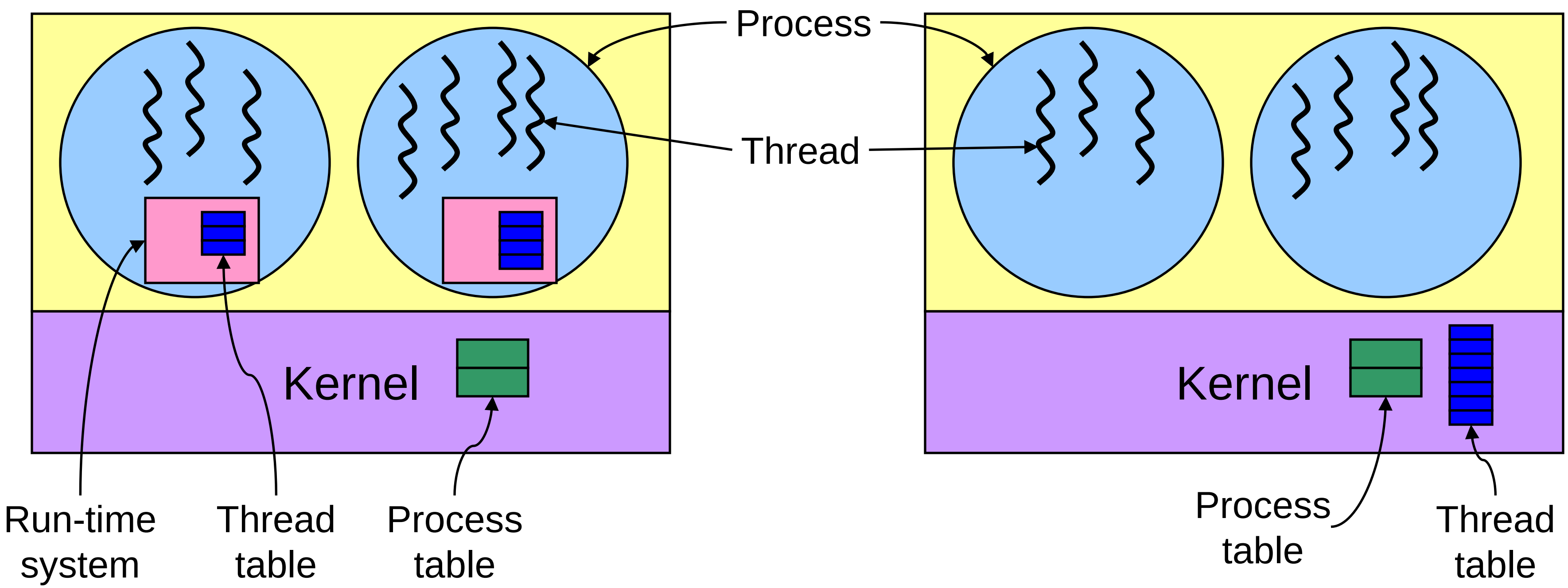


Why use threads?

- Simpler programming model
- Less waiting
- Threads are faster to create or destroy
  - No separate address space
- Overlap computation and I/O
- Example: word processor
  - Thread to read from keyboard
  - Thread to format document
  - Thread to write to disk



Implementing threads



User-level threads
+ No need for kernel support
- May be slower than kernel threads
- Harder to do non-blocking I/O
    - Page faults
    -System calls

Kernel-level threads
+ More flexible scheduling
+ Non-blocking I/O
- Not portable

- Single Threading Vs. Multi Threading
- Thread is a chain that include sum of the instructions
- Address space and some resources is shared:
  - Open files
  - Signal handlers
  - …
- Each thread has own Program counter and registers and stack

- Theads’s Goals:
  - Increase speed of execution
  - Improve performance (Data transmission between thread of a process)
  - Parallelism
  - Modularity and Granularity