Process/CPU scheduling

Saptarshi Ghosh and Mainack Mondal
CS39002

Spring 2020-21

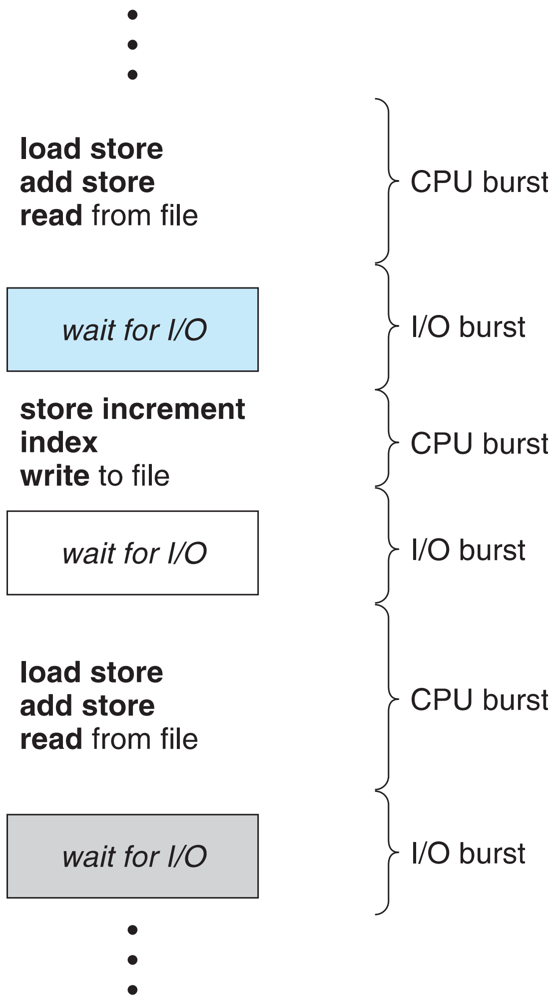
Why process scheduling: Recap

- CPU scheduling forms the central idea behind multiprogramming OS
  - By switching the CPU among processes, the OS better utilizes the computer system
  - Modern OS also consider lightweight process like entities called threads (to be discussed later…)

Still: Why do we need scheduling today

- Recall
  - CPU burst: the process is being executed in the CPU
  - I/O burst: the process is waiting for I/O to be done
  - A Process alternates between CPU and I/O burst

Why is CPU and I/O burst important?



Maximum CPU utilization obtained with multiprogramming

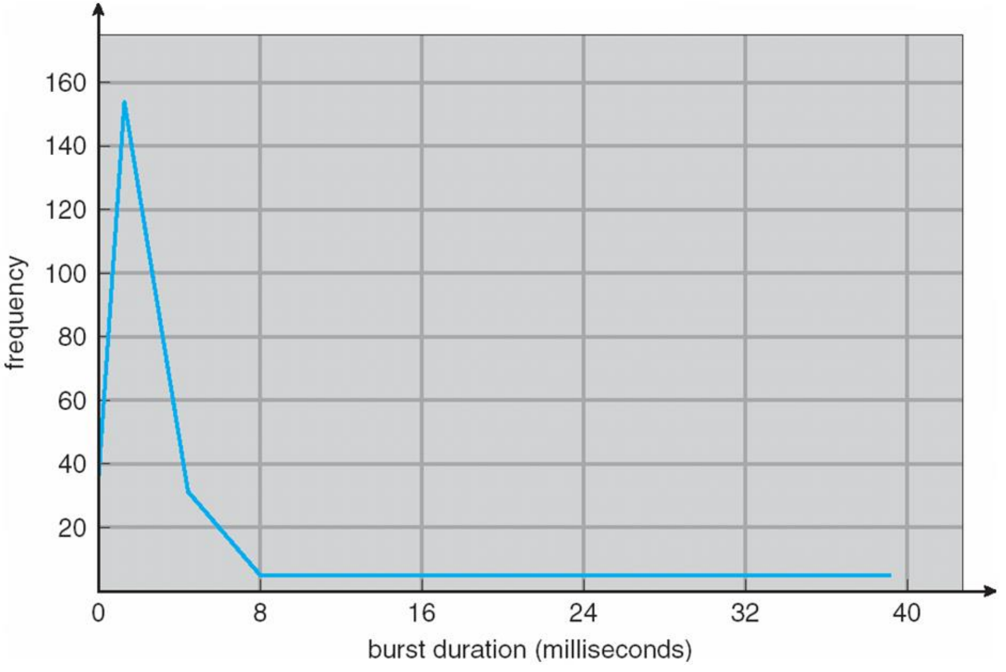
CPU–I/O Burst Cycle – Process execution consists of a cycle of CPU execution and I/O wait

CPU burst followed by I/O burst

CPU burst distribution is of main concern

Characteristics of CPU bursts

- Typically, CPU bursts follow an exponential or hyper-exponential distribution
  - Large number of short CPU bursts
  - Small number of large CPU bursts

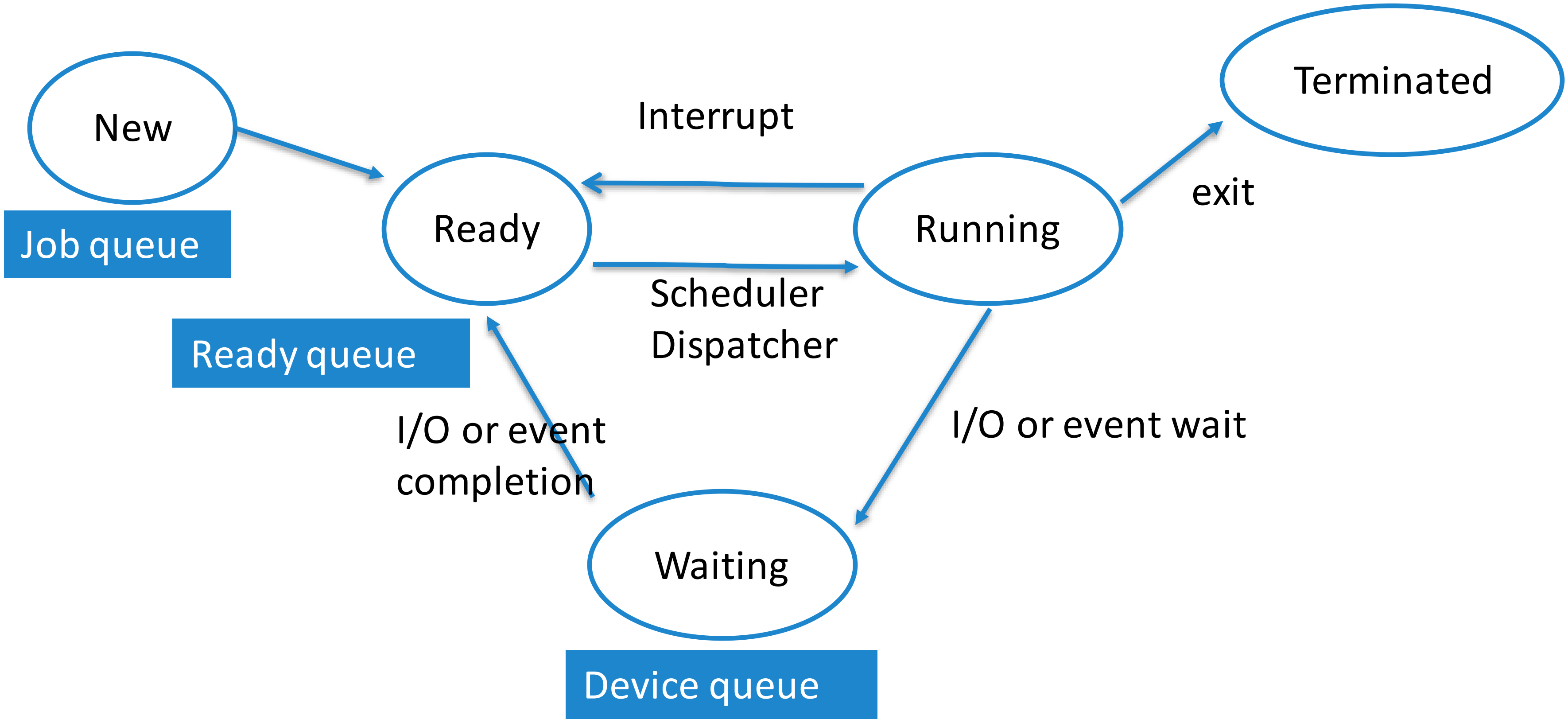


Characteristics of CPU bursts

- Typically, CPU bursts follow an exponential or hyper-exponential distribution
    - Large number of short CPU bursts
    - Small number of large CPU bursts

- Whenever the CPU becomes free, the OS selects one of the processes in the ready queue for executing next
    - Decided by the short-term scheduler or CPU scheduler

Recap: Process state diagram



Schedulers

- Short-term scheduler (or CPU scheduler) –
  - selects which process should be executed next in CPU
  - Sometimes the only scheduler in a system
  - Short-term scheduler is invoked frequently (milliseconds) --> must be fast

Schedulers

- Short-term scheduler (or CPU scheduler) –
  - selects which process should be executed next in CPU
  - Sometimes the only scheduler in a system
  - Short-term scheduler is invoked frequently (milliseconds) --> must be fast

- Long-term scheduler (or job scheduler) – selects which processes should be brought into the ready queue
  - Long-term scheduler is invoked infrequently (seconds, minutes) --> okay if it is relatively slow
  - The long-term scheduler controls the degree of multiprogramming

When is scheduler called?

- A processes switches from RUNNING to WAITING
  - E.g., I/O, wait() call
- A processes switches from RUNNING to READY
  - E.g., timer interrupt, I/O interrupt
- A processes switches from WAITING to READY
  - E.g., completion of I/O, child terminates
- A process terminates
  - E.g., exit() call

Non-preemptive scheduling

- Scheduling happens only when
  - A processes switches from RUNNING to WAITING
  - A process terminates

- Once CPU is allocated to a process, it keeps the CPU for as long as the process requires

Pre-emptive scheduling

- CPU can be forcibly taken away from a running process
  - E.g., due to timer interrupt upon completion of time slice
  - May result in race condition
  - In Many OS, core kernel routines are non-preemptive

Dispatcher

- Dispatcher module gives control of the CPU to the process selected by the short-term scheduler.
- Functions
  - switching context
  - switching to user mode
  - jumping to the proper location in the user program to restart that program

- Dispatch latency should be low --> dispatcher should be as fast as possible

Next ...

- How to schedule?
- In other words, how to select the process to be run next, from among the processes in the ready queue?