Scheduling

Basic Notions

- Given
  - A set of sequential *processes*
  - A number of *processors*

  a *schedule* is an assignment of processors to processes over time.
- A *scheduler* is a component that determines a schedule
- A *scheduling strategy* is principle for constructing schedules
- A *scheduling algorithm* is an implementation of a strategy
- A scheduling strategy may be *static* (planning) or *dynamic*

Scheduling Objectives

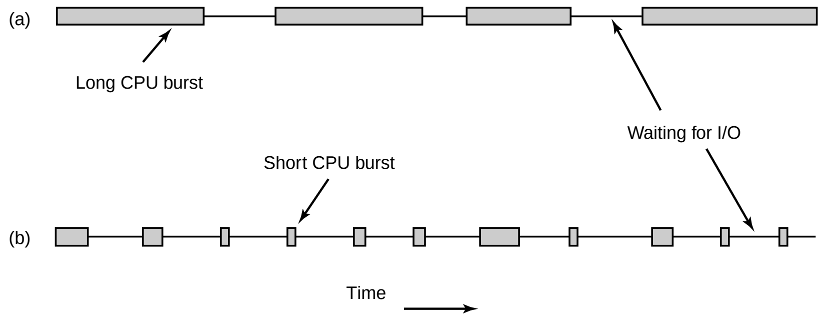
Process Types

- *Batch* processes — High CPU/IO ratio
- *Interactive* processes — Low CPU/IO ratio
- *Real-time* processes — deadlines, timeliness

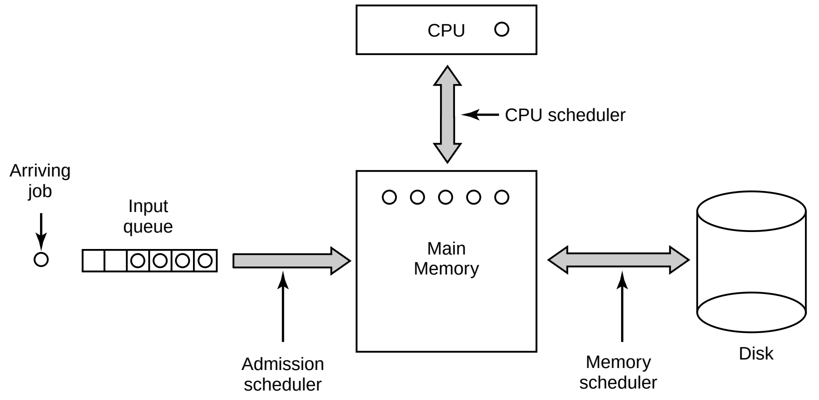
Goals

- *Fairness* — every process/user/client get a fair share
- Good *utilization* — no resource is unnecessarily idle
- High *throughput*
- Acceptable *response times*
- *Dead-lines* must be met

Process Examples



Multi-level Scheduling



Common Scheduling Strategies

Throughput-oriented

- *First-come-first-served* FCFS
- *Shortest-job-first* — requires behaviour knowledge

Fairness-oriented

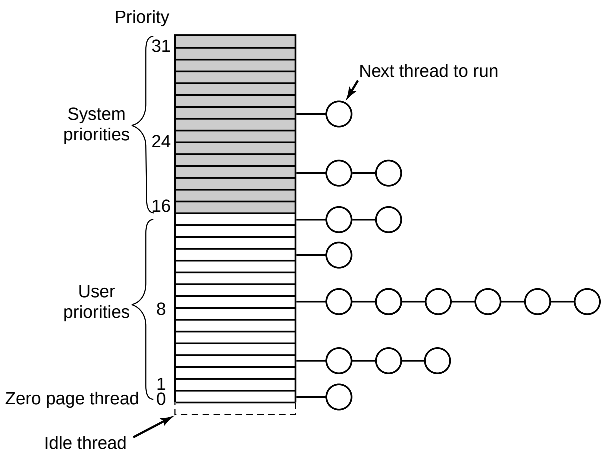
- *Round Robin* — preemption at end of *time slice*
- Elaboration: *Priority classes* with dynamic *boost*

Real-time

- *Fixed priorities* with strict preemption
- Dynamic priorities: Least Slack Time, *Earliest Deadline First*, ...

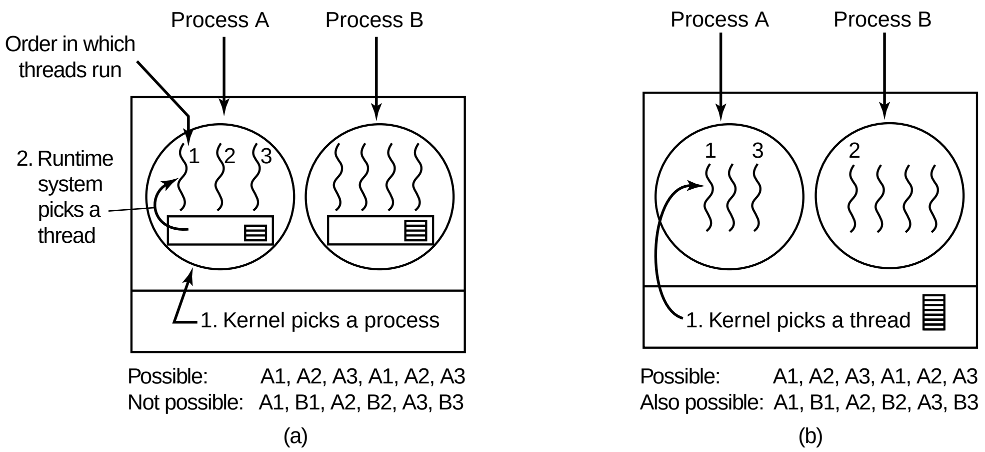
Priority Classes

Windows example



- User priorities *boosted* (+1..+8) by I/O completion
- Boosted threads *drop down* when slice expires

Thread Scheduling



- Prevailing approach: Kernel threads scheduled *globally*

Multi-processor Scheduling Issues

Traditional Symmetrical Multi-Processing (SMP)

- Any thread may execute on any processor
- Global prioritized *ready queue*
- *Strictness:* The M processors execute the M top priority threads
- Disadvantages:
  - Strictness $\Rightarrow$ inter-processor interruption
  - Processor jumping $\Rightarrow$ cache contents lost
  - Locking contention on ready queue

Trends

- Threads may have *affinities* to processors
- Linux: *Local ready queues* + *load balancing*