Operating Systems
CPU Scheduling

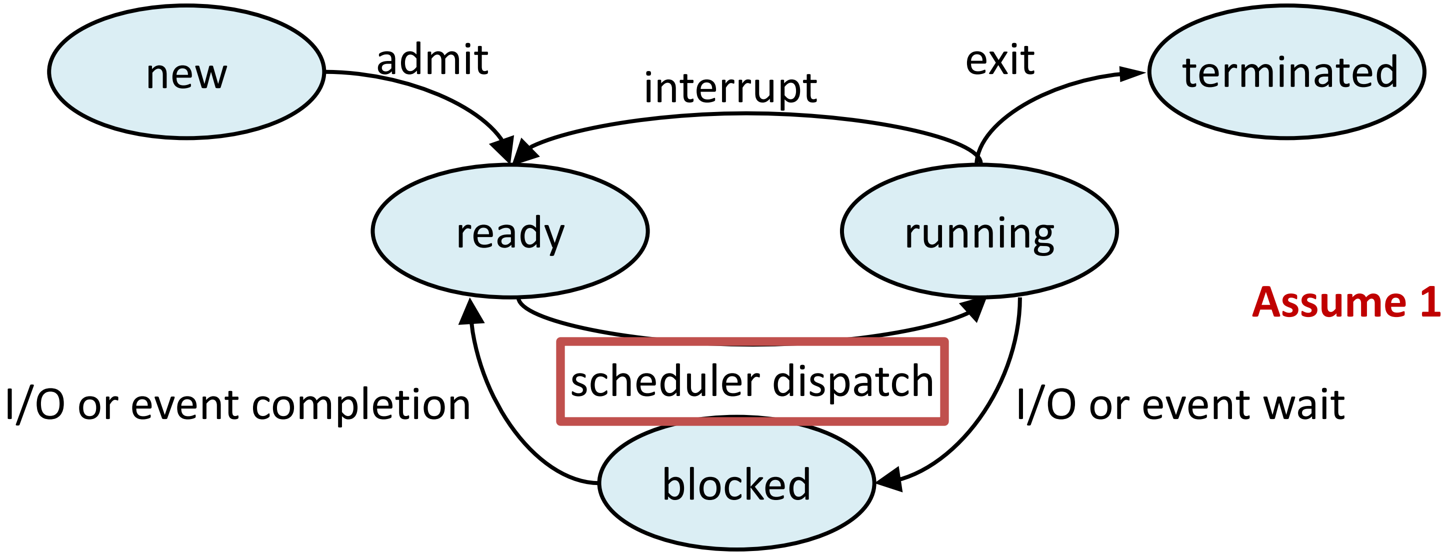
Yanyan Zhuang

Department of Computer Science

http://www.cs.uccs.edu/~yzhuang

What is CPU Scheduling?

- The five-state process model



Assume 1 CPU

CPU scheduling
Selects from among the processes/threads that are ready to execute, and allocates CPU time to it

Why CPU Scheduling?

- In support of multiprogramming
  - Uniprocessor systems
    - Batch, time-sharing processor
  - Real-time systems
    - Reliably guaranteeing deadlines
- What does CPU schedule?
  - Processes, threads

CPU Scheduling

- Process context
  - Minimal set of state info that must be stored to allow a process to be stopped and restarted later
  - Information stored in CPU
    - Contents of registers
      - Program counter, stack pointer
  - Information stored in RAM
    - PCB

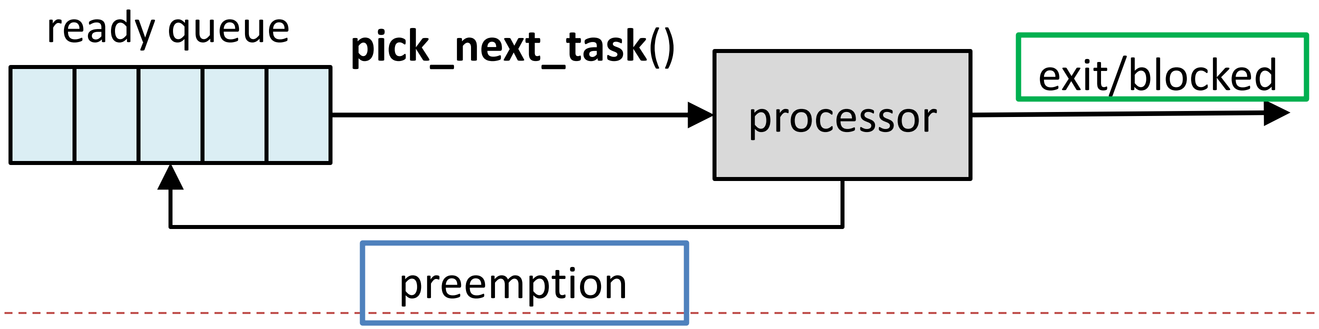
CPU Scheduling

- CPU scheduling may take place at
  - Clock interrupts
  - I/O interrupts
  - Blocking system call
  - Termination

  $\left.\begin{array}{l}\text{Clock interrupts}\\ \text{I/O interrupts}\end{array}\right\}$ preemptive

  $\left.\begin{array}{l}\text{Blocking system call}\\ \text{Termination}\end{array}\right\}$ non-preemptive

- Non-preemptive
  - Scheduling only when current process terminates or gives up control
- Preemptive
  - Processes can be forced to give up control



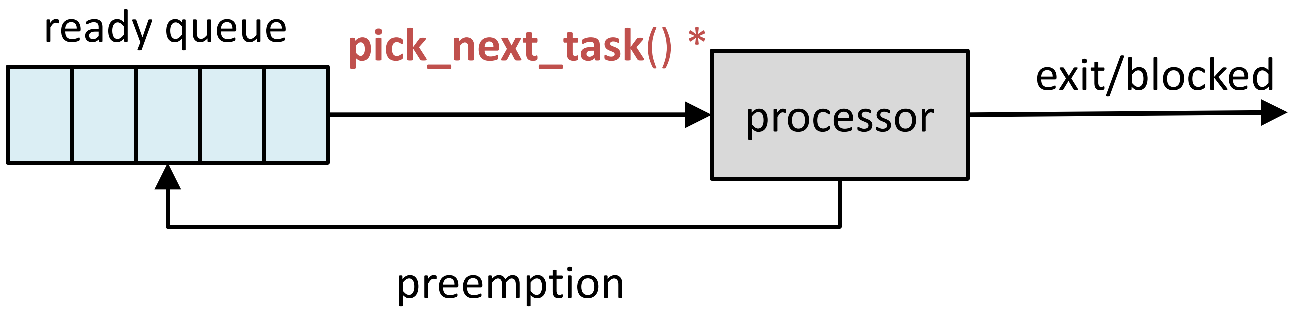
Scheduling Goals

- User oriented → minimize
  - Response time (wait time): the time that the ***first*** response is received (interactivity)
  - Turnaround time: the time that the task **finishes**
  - Unpredictability: variations in different runs
- System oriented → maximize
  - Throughput: # of tasks that finish per time unit
  - Utilization: the percentage of time the CPU is busy
  - Fairness: avoid starvation

Response time = first response time – arrival time
Turnaround time = finish time – arrival time

Scheduling Policies

- Basic policies
  - First-Come First-Served (FCFS)
  - Shortest-Job-First (SJF)
  - Round Robin (RR)
- Implementation: pick the next ready task to run
  - How we design pick_next_task()
- Performance: response time and turnaround time



Scheduling Policies

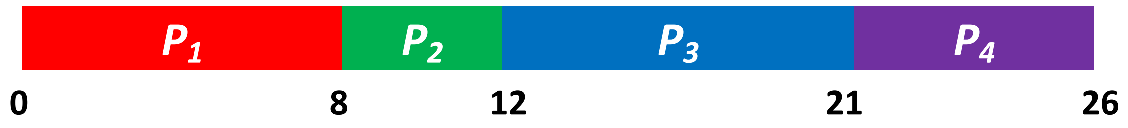
- Batch Systems
  - **First-Come First-Serve**
  - **Shortest Job First**
  - Shortest Remaining Time Next
- Interactive Systems
  - **Round-Robin**
  - Priority Scheduling
  - Multiple Queues
  - Shortest Process Next
  - Guaranteed Scheduling
  - Lottery Scheduling
- Real-time Systems
  - Rate Monotonic Scheduling
  - Earliest Deadline First Scheduling

First-Come, First-Serve (FCFS)

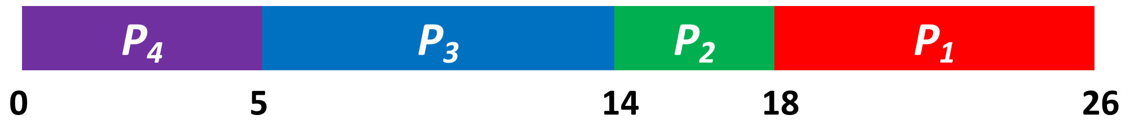
- CPU schedules the task that arrived earliest, non-preemptive

Change arrival time to P1:3, P2:2, P3:1, P4:0
Burst time the same

Process	Arrival Time	Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5



Average turnaround time = ((8-0)+(12-1)+(21-2)+(26-3)) / 4 = 15.25
Average response time = (0+(8-1)+(12-2)+(21-3)) /4 = 8.75



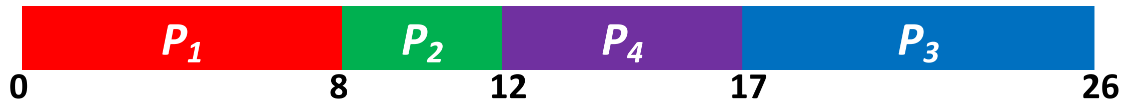
Average turnaround time = ((5-0)+(14-1)+(18-2)+(26-3)) / 4 = 14.25
Average response time = (0+(5-1)+(14-2)+(18-3)) /4 = 7.75

Shortest Job First (SJF)

- CPU schedules the task with the shortest remaining time

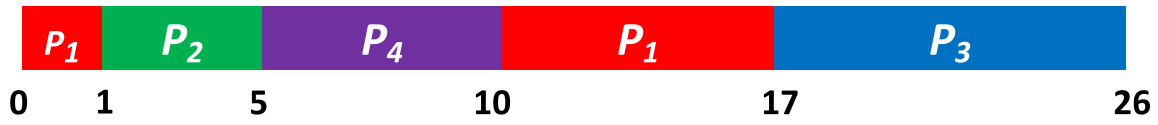
Process	Arrival Time	Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

non-preemptive



Average turnaround time = ((8-0)+(12-1)+(26-2)+(17-3)) / 4 = 14.25
Average response time = (0+(8-1)+(17-2)+(12-3)) /4 = 7.75

preemptive



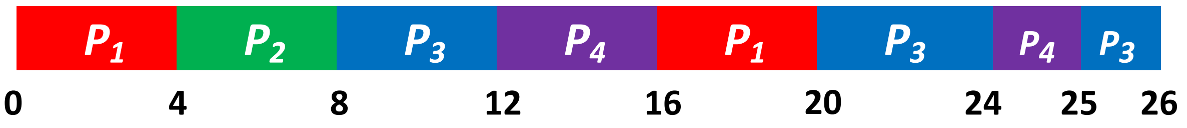
Average turnaround time = ((17-0)+(5-1)+(10-3)+(26-2)) / 4 = 13
Average response time = (0+(1-1)+(5-3)+(17-2)) /4 = 4.25

Round Robin (RR)

- Like FCFS, but with limited time slices, preemptive

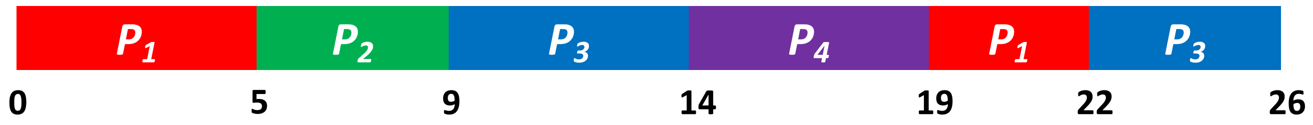
Process	Arrival Time	Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

q=4



Average turnaround time = ((20-0)+(8-1)+(26-2)+(25-3)) / 4 = 18.25
Average response time = (0+(4-1)+(8-2)+(12-3)) /4 = 4.5

q=5

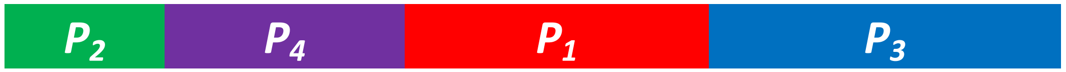


Average turnaround time = ((22-0)+(9-1)+(26-2)+(19-3)) / 4 = 17.5
Average response time = (0+(5-1)+(9-2)+(14-3)) /4 = 5.5

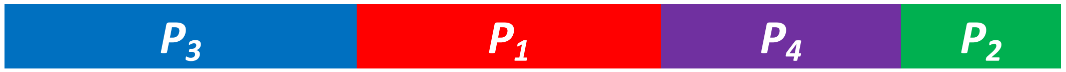
Priority Scheduling

- CPU schedules highest priority first, FCFS within same priority level

Process	Priority	Burst Time
P_1	3	8
P_2	1	4
P_3	4	9
P_4	2	5



Process	Priority	Burst Time
P_1	2	8
P_2	4	4
P_3	1	9
P_4	3	5



Put it together

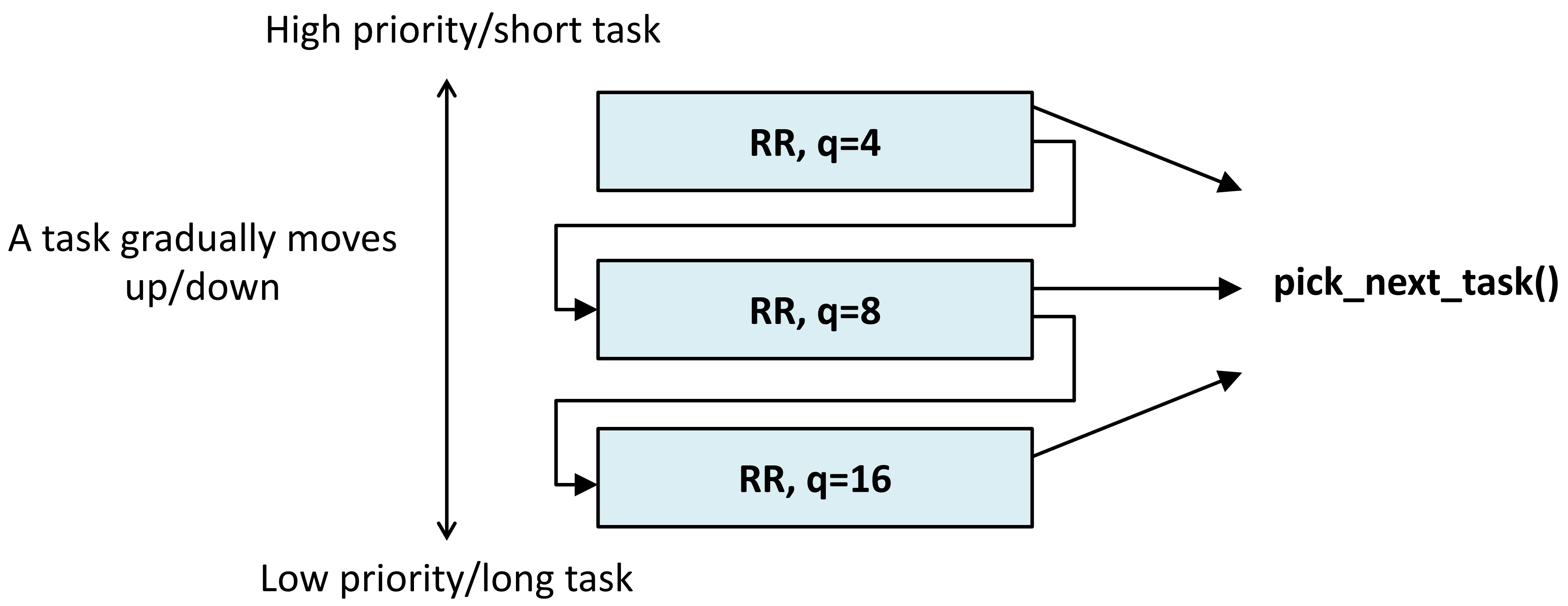
	Turnaround time	Response time
FCFS	15.25	8.75
SJF-preemptive	**13**	**4.25**
RR (q=5)	17.5	5.5

	Throughput	Response time	Starvation
FCFS	Depends	Depends	No
SJF-preemptive	High	Good	Yes
RR	Can be low	Good	No
Priority scheduling	Can be high	Can be good	Can remove

⇩

Multilevel Feedback Queue

Multilevel Feedback Queue

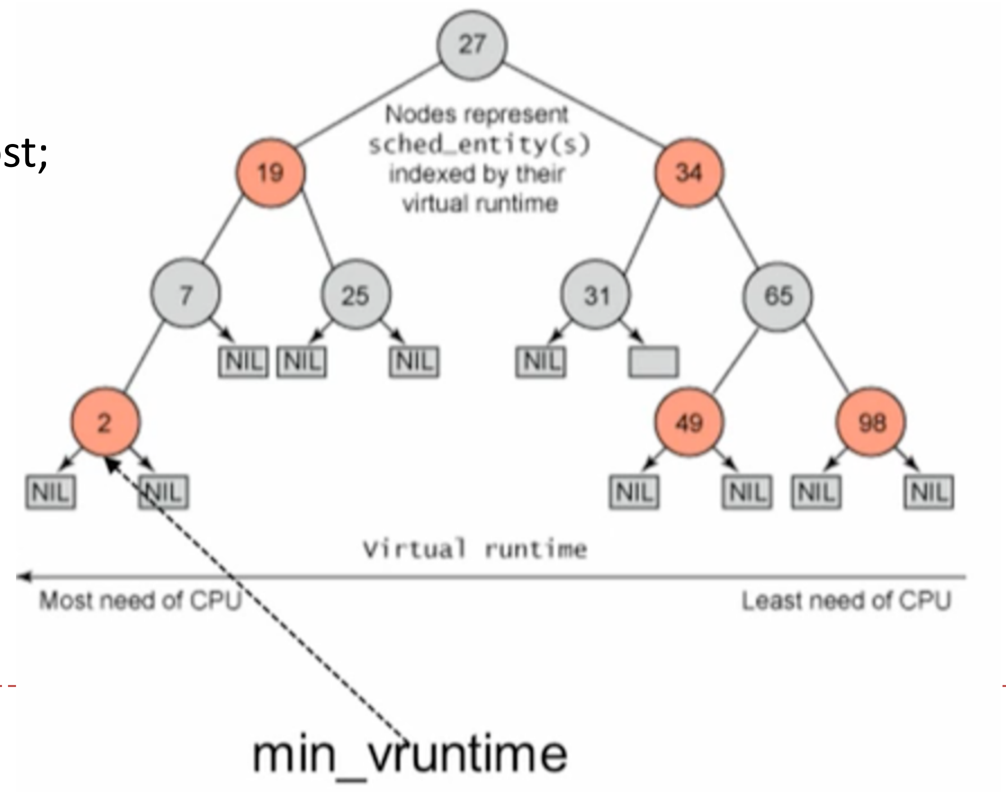


Windows XP, Mac OS X, Linux 2.6.22 and before

A Close Look at the State-of-Art

- Linux Completely Fair Scheduler (CFS)
  - Separate ready queue per processor
  - Red-black tree based ready queue
  - Proportional fair sharing

```
pick_next_task()
Remove leftmost task curr_leftmost;
Run the task;
vruntime + = runtime;
if vruntime > curr_leftmost's
Preempt task;
Put it back to the sorted tree;
Remove curr_leftmost;
```



Real-time Scheduling

Schedulable real-time system

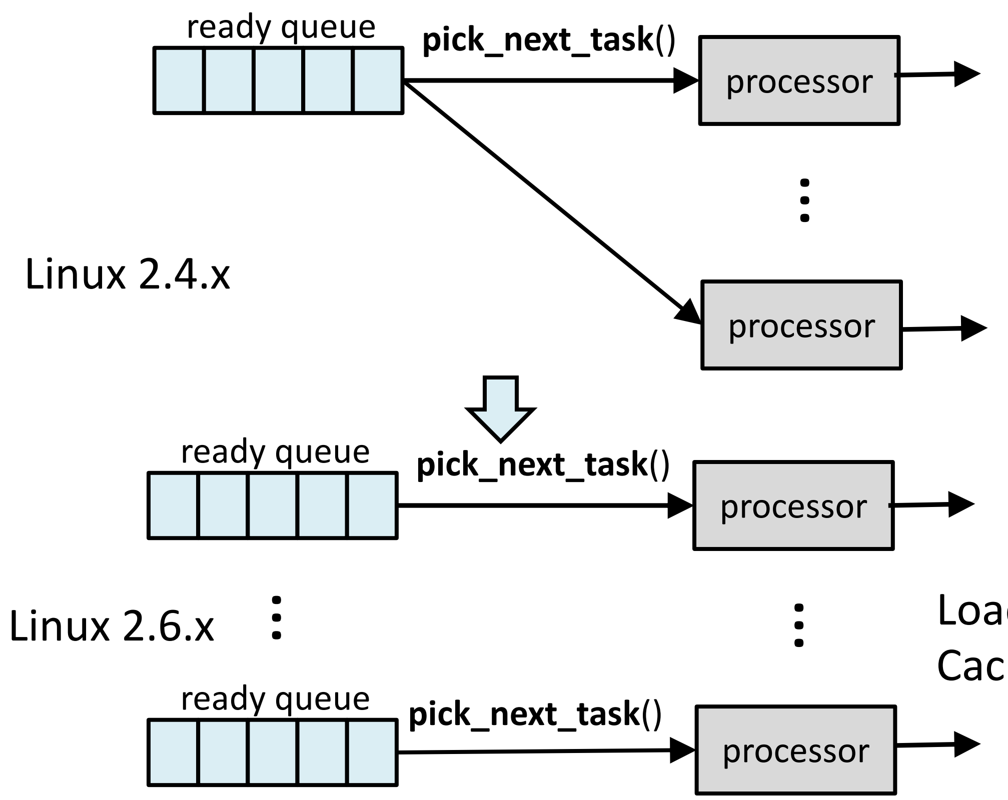
- Given
  - *m* periodic events
  - event *i* occurs within period P_i seconds and requires C_i seconds
- Then the load can only be handled (schedulable) if

$$\sum_{i=1}^{m} \frac{C_i}{P_i} \leq 1$$

- Example: a real-time system with three periodic events, with periods of 100, 200, and 500 ms, respectively. If these events require 50, 30, and 100 ms of CPU time per event, respectively, the system is schedulable
  - Process/context switching overhead is often an issue though!
  - Given the example, what would be the maximum CPU burst for a 4th job with a period of 500 ms?

Challenges on Emerging Hardware and Applications

- Multiprocessor → Many core



Multiprocessor = more powerful processor

pick_next_task() will be the bottleneck

Load balancing
Cache affinity

Problem:
Self-scheduling, global control is difficult

Summary

- The basic scheduling policies are important
  - Multilevel Feedback Queue = RR + Priority
  - CFS = RR + SJF + Priority + smart data structure
- Additional readings
  - Go to http://lxr.linux.no/linux+v2.6.24/kernel/
  - Read /kernel/sched.c, /kernel/sched_fair.c, /include/linux/sched.h (starting from the schedule(void) function)
  - See how the vruntime is actually updated
  - Documentation: http://www.kernel.org/doc/Documentation/scheduler/sched-design-CFS.txt
  - Another interesting scheduler: BFS
    - http://ck.kolivas.org/patches/bfs/bfs-faq.txt