W4118: scheduling

Instructor: Junfeng Yang

References: Modern Operating Systems (3rd edition), Operating Systems Concepts (8th edition), previous W4118, and OS at MIT, Stanford, and UWisc

Outline

- Introduction to scheduling
- Scheduling algorithms

Direction within course

- Until now: interrupts, processes, threads, synchronization
  - Mostly mechanisms

- From now on: resources
  - Resources: things processes operate upon
    - E.g., CPU time, memory, disk space
  - Mostly policies

Types of resource

- Preemptible
  - OS can take resource away, use it for something else, and give it back later
    - E.g., CPU
- Non-preemptible
  - OS cannot easily take resource away; have to wait after the resource is voluntarily relinquished
    - E.g., disk space
- Type of resource determines how to manage

Decisions about resource

- Allocation: which process gets which resources
  - Which resources should each process receive?
  - Space sharing: Controlled access to resource
  - Implication: resources are not easily preemptible

- Scheduling: how long process keeps resource
  - In which order should requests be serviced?
  - Time sharing: more resources requested than can be granted
  - Implication: Resource is preemptible

Role of Dispatcher vs. Scheduler

- Dispatcher
  - Low-level mechanism
  - Responsibility: context switch
- Scheduler
  - High-level policy
  - Responsibility: deciding which process to run
- Could have an allocator for CPU as well
  - Parallel and distributed systems

When to schedule?

- When does scheduler make decisions?
  When a process
  1. switches from running to waiting state
  2. switches from running to ready state
  3. switches from waiting to ready
  4. terminates
- Minimal: nonpreemptive
  - ?
- Additional circumstances: preemptive
  - ?

Outline

- Introduction to scheduling
- Scheduling algorithms

Overview of scheduling algorithms

- Criteria: workload and environment
- Workload
  - Process behavior: alternating sequence of CPU and I/O bursts
  - CPU bound v.s. I/O bound
- Environment
  - Batch v.s. interactive?
  - Specialized v.s. general?

Scheduling performance metrics

- Min waiting time: don't have process wait long in ready queue
- Max CPU utilization: keep CPU busy
- Max throughput: complete as many processes as possible per unit time
- Min response time: respond immediately
- Fairness: give each process (or user) same percentage of CPU

First-Come, First-Served (FCFS)

- Simplest CPU scheduling algorithm
  - First job that requests the CPU gets the CPU
  - Nonpreemptive

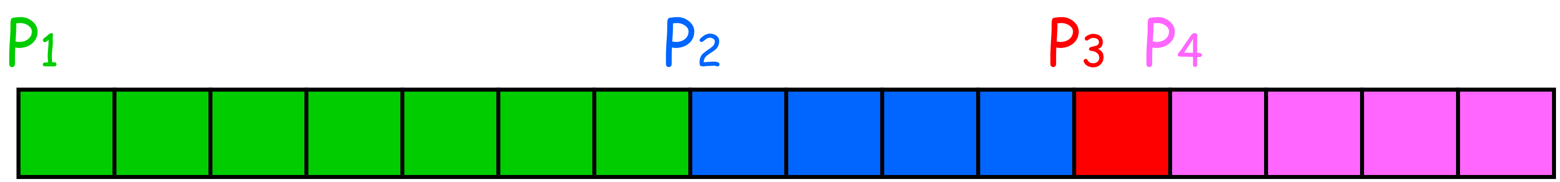
- Implementation: FIFO queue

Example of FCFS

Process	Arrival Time	Burst Time
P_1	0	7
P_2	0	4
P_3	0	1
P_4	0	4

- Gantt chart

Schedule:



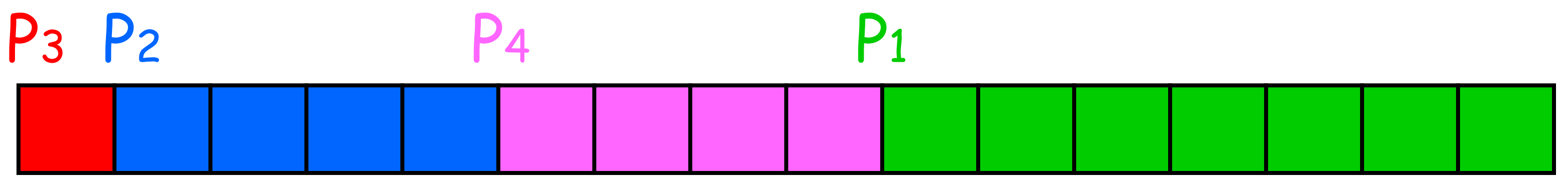
- Average waiting time: (0 + 7 + 11 + 12)/4 = 7.5

Example of FCFS: different arrival order

Arrival order: P_3 P_2 P_4 P_1

- Gantt chart

Schedule:



- Average waiting time: (9 + 1 + 0 + 5)/4 = 3.75

FCFS advantages and disadvantages

- Advantages
  - Simple
  - Fair

- Disadvantages
  - waiting time depends on arrival order
  - Convoy effect: short process stuck waiting for long process
  - Also called head of the line blocking

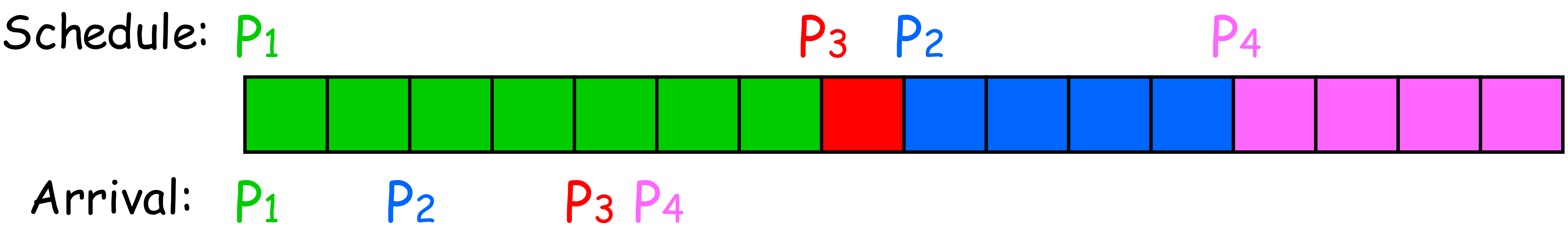
Shortest Job First (SJF)

- Schedule the process with the shortest time
- FCFS if same time

Example of SJF (w/o preemption)

Process	Arrival Time	Burst Time
P_1	0	7
P_2	2	4
P_3	4	1
P_4	5	4

- Gantt chart



- Average waiting time: (0 + 6 + 3 + 7)/4 = 4

Shortest Job First (SJF)

- Schedule the process with the shortest time
  - FCFS if same time
- Advantages
  - Minimizes average wait time. Provably optimal if no preemption allowed
- Disadvantages
  - Not practical: difficult to predict burst time
    - Possible: past predicts future
  - May starve long jobs

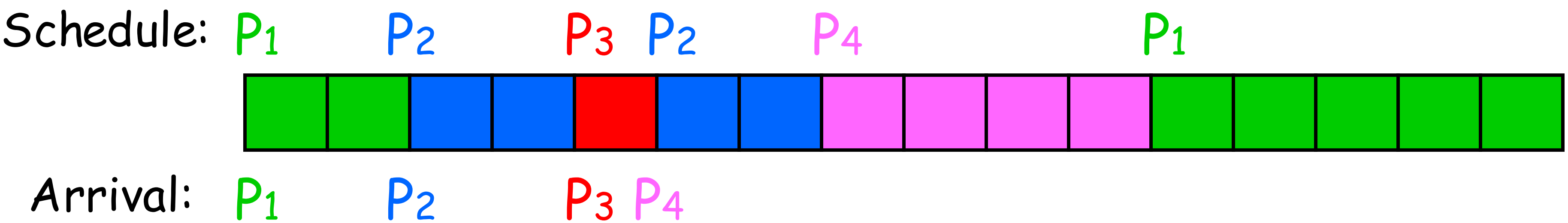
Shortest Remaining Time First (SRTF)

- If new process arrives w/ shorter CPU burst than the remaining for current process, schedule new process
  - SJF with preemption

- Advantage: reduces average waiting time

Example of SRTF

- Gantt chart



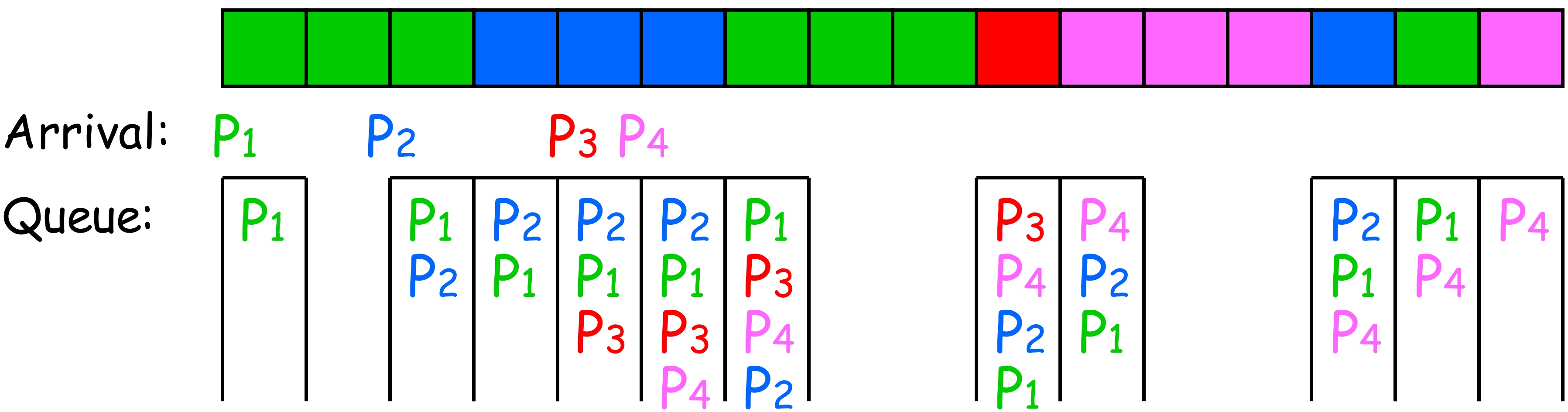
- Average waiting time: $(9 + 1 + 0 + 2)/4 = 3$

Round-Robin (RR)

- Practical approach to support time-sharing
- Run process for a time slice, then move to back of FIFO queue
- Preempted if still running at end of time-slice
- How to determine time slice?

Example of RR: time slice = 3

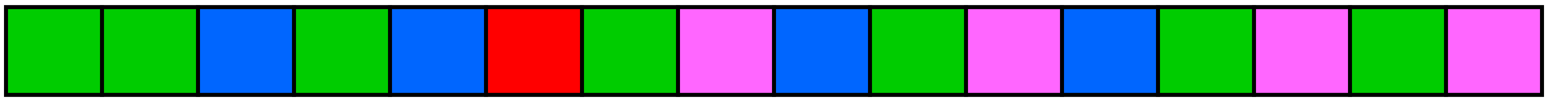
- Gantt chart with time slice = 3



- Average waiting time: (8 + 8 + 5 + 7)/4 = 7
- Average response time: (0 + 1 + 5 + 5)/4 = 2.75
- # of context switches: 7

Example of RR: smaller time slice

- Gantt chart with time slice = 1

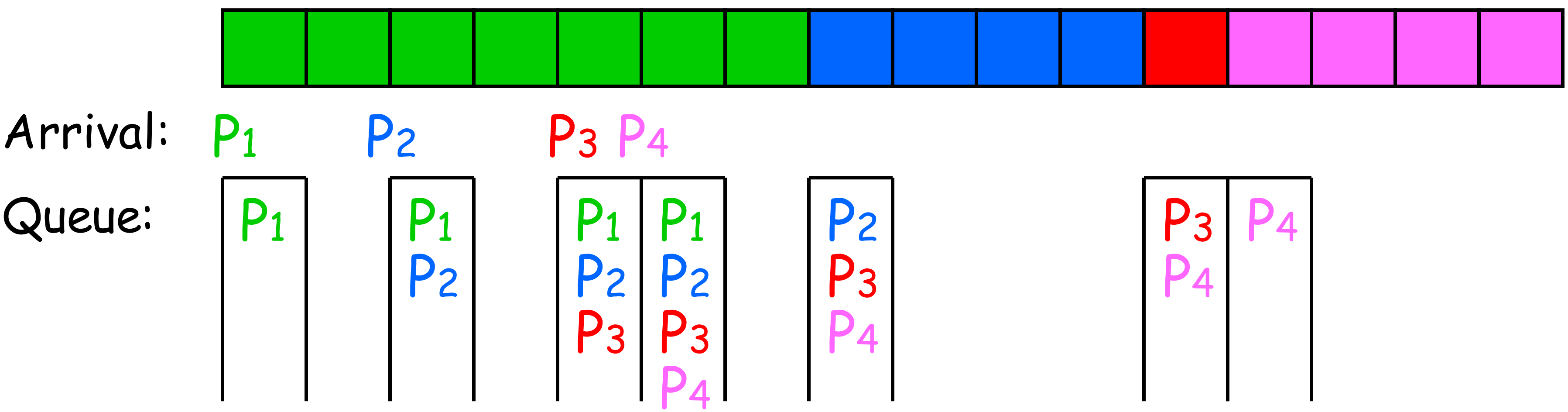


Arrival:	P_1		P_2		P_3	P_4										
Queue:	P_1	P_1	P_2 P_1	P_1 P_2	P_2 P_3 P_1	P_3 P_1 P_4 P_2	P_1 P_4 P_2	P_4 P_2 P_1	P_2 P_1 P_4	P_1 P_4 P_2	P_4 P_2 P_1	P_2 P_1 P_4	P_1 P_4	P_4 P_1	P_1 P_4	P_4

- Average waiting time: (8 + 6 + 1 + 7)/4 = 5.5
- Average response time: (0 + 0 + 1 + 2)/4 = 0.75
- # of context switches: 14

Example of RR: larger time slice

- Gantt chart with time slice = 10



- Average waiting time: (0 + 5 + 7 + 7)/4 = 4.75
- Average response time: same
- # of context switches: 3 (minimum)

RR advantages and disadvantages

- Advantages
  - Low response time, good interactivity
  - Fair allocation of CPU across processes
  - Low average waiting time when job lengths vary widely
- Disadvantages
  - Poor average waiting time when jobs have similar lengths
    - Average waiting time is even worse than FCFS!
  - Performance depends on length of time slice
    - Too high → degenerate to FCFS
    - Too low → too many context switches, costly

Priorities

- A priority is associated with each process
  - Run highest priority ready job (some may be blocked)
  - Round-robin among processes of equal priority
  - Can be preemptive or nonpreemptive
- Representing priorities
  - Typically an integer
  - The larger the higher or the lower?

Setting priorities

- Priority can be statically assigned
  - Some always have higher priority than others
  - Problem: starvation
- Priority can be dynamically changed by OS
  - Aging: increase the priority of processes that wait in the ready queue for a long time

Priority inversion

- High priority process depends on low priority process (e.g. to release a lock)
  - Another process with in-between priority arrives?
- Solution: priority inheritance
  - Inherit highest priority of waiting process
  - Must be able to chain multiple inheritances
  - Must ensure that priority reverts to original value