UNIVERSITY of WISCONSIN-MADISON
Computer Sciences Department

CS 537 — A. Arpaci-Dusseau
Intro to Operating Systems — Spring 2000

# CPU Scheduling

## Questions answered in these notes:

- What is the difference between allocation and scheduling?
- What is preemptive vs. non-preemptive scheduling?
- How do different algorithms behave?
  What are their advantages and disadvantages?

## Reading

- Chapter 5

# Direction within Course

## Until now: Processes

- Process implementation
- Process synchronization/ deadlock

## From now on: Resources

- Resources are things operated upon by processes
- Example: CPU time, disk space, disk access, memory

## Today: Managing the CPU by Scheduling



# Resources

## Preemptible

- Can take resource away, use it for something else, give it back later
- Example: CPU

## Non-preemptible

- Once given resource, it can't be reused until voluntarily relinquished
- Examples: File space, terminal

## Given set of resources and set of requests for those resources:
Type of resource determines how OS manages it



# Decisions about Resources

## *Allocation*: Which process gets which resources

- Which resources should each process receive?
- **Space sharing**: Control access to resource
- Implication: Resources are not easily preemptible
- Example: File space

## *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
- Example: Processor scheduling

# Role of Dispatcher vs. Scheduler

## Dispatcher

- Low-level mechanism
- Responsibility: Context-switch
  - Save execution state of old process in PCB
  - Load execution state of new process from PCB to registers
  - Change scheduling state of process (**running, ready,** or **blocked**)
  - Switch from kernel to user mode
  - Jump to instruction in user process

## Scheduler

- Higher-level policy
- Responsibility: Deciding which process to run

## Could have an Allocator for CPU as well

- Parallel and Distributed systems



# Scheduling Performance Metrics

## Minimize waiting time

- Do not have process wait long in ready queue

## Maximize resource utilization

- Keep CPU and disks busy

## Minimize overhead

- Reduce context switches (number and cost)

## Minimize response time

- Keystrokes for interactive jobs (e.g., word processors)

## Distribute resources equitably

- Give each user (or process) same percentage of CPU



# Preemptive Scheduling

## When does scheduler need to make a decision?

## Minimal amount: Nonpreemptive

- Two cases
  1.
  2.
- Implication:
  Process remains scheduled until voluntarily relinquishes CPU

## Additional circumstances: Preemptive

- More cases
  1.
  2.
- Implication:
  Higher priority job can interrupt lower priority jobs whenever ready



# Scheduling Algorithms

## Process (Job) Model

- Process alternates between CPU and I/O bursts
- CPU-bound job: Long CPU bursts
- I/O-bound job: Short CPU bursts

## Do not know what type of job each is before executing

- Need to perform “well” on all job types

## Scheduling Algorithms

- First-Come-First-Served (FCFS)
- Shortest-Job-First (SJF) or Shortest-Time-Completion-First (STCF)
- Round-Robin (RR)
- Priority Scheduling
- Multilevel Feedback Queue Scheduling (Solaris TS in next notes)

# First Come First Served (FCFS)

## Simplest CPU scheduling algorithm

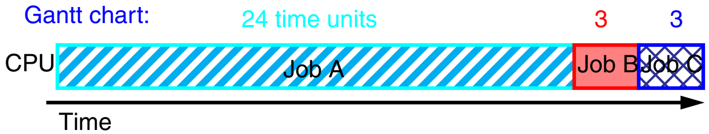
- First job that requests the CPU is allocated the CPU
- **Nonpreemptive**

## Advantage: Simple implementation with FIFO queue

## Disadvantage: Waiting time depends upon arrival order

- Unfair to later jobs (especially if long jobs arrive first)

Example: Three jobs arrive nearly simultaneously (A, B, and C)




Average Waiting Time:

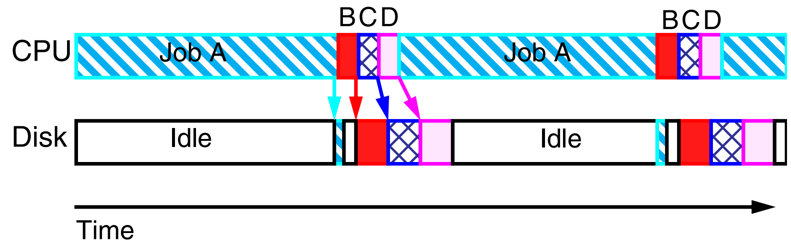
- Uniprogramming: Run job to completion
- Multiprogramming: Put job at back of queue when perform I/O



# Convoy Effect

## Short-running jobs stuck waiting for long-running jobs

- Example: 1 CPU-bound job; 3 I/O-bound jobs



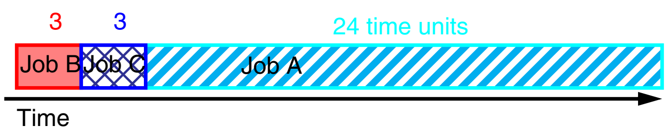

## Problems

- Reduces utilization of I/O devices
- Hurts waiting time of short jobs



# Shortest Job First (SJF)

## Minimize average wait time if run shortest job first




- FCFS if same time
- Average Waiting Time:

## Provably optimal (given no preemption)

- Moving shorter job before a longer job improves waiting time of short job more than harms waiting time of long job
- Reduces average waiting time

## Not practical: Cannot predict burst time

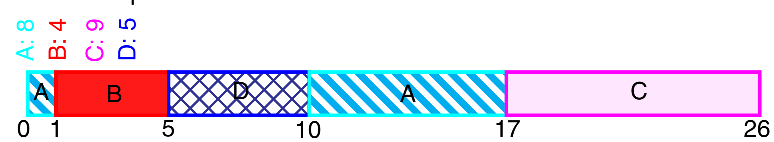
- Use past behavior to predict future behavior



# Shortest Time to Completion First (STCF)

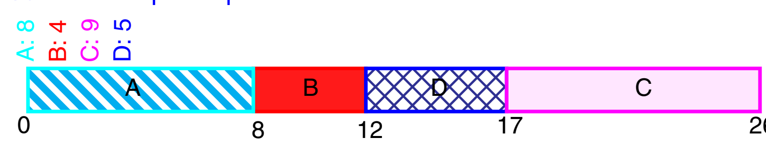
## STCF == SJF with preemption

- New process arrives w/ shorter CPU burst than that remaining for current process




- Average Wait Time:

## SJF without preemption




- Average Wait Time:

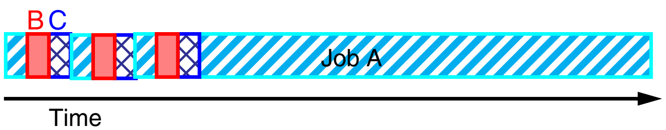
# Round-Robin (RR)

## Practical approach to support time-sharing

- Run job for a **time-slice** and then move to back of FIFO queue
- **Preempted** if still running at end of time-slice

## Advantages

- Fair allocation of CPU across jobs
- Low average waiting time when job lengths vary widely




- Average Waiting Time:
- Compare to waiting time for FCFS and SJF



# Disadvantages of Round-Robin

## Poor average waiting time when job lengths are identical

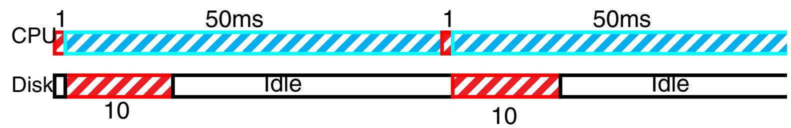
- Imagine 10 jobs each requiring 10 time slices
- RR: All complete after about 100 time slices
- Even FCFS is better!

## Performance depends on length of time slice

- If time-slice too high, degenerate to FCFS

  1 job w/ 1ms compute and 10ms I/O; 1 job always computes

  Time-Slice = 50ms



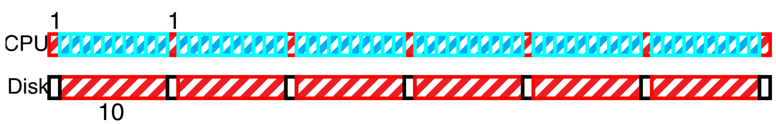



# Disadvantages of RR Continued

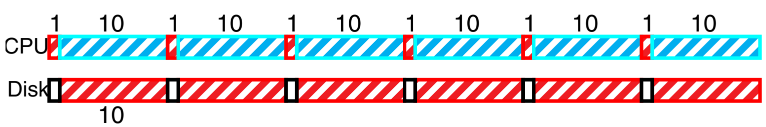
- If time-slice too low, pay overhead of context-switch too frequently

  1 job w/ 1ms compute and 10ms I/O; 1 job always computes

- Time-Slice = 1ms




## STCF is still ideal




## How do we approximate this ideal?



# Priorities

## Each process has a priority

- Run highest priority **ready** job in system (some may be blocked)
- Round-robin among processes of equal priority
- Can be preemptive or nonpreemptive

## Is a large integer a high or a low priority???

- Solaris: User priorities range from 0 to 59
  59 is the highest
- Text book: Low numbers indicate high priority ...

## Data structure question

- Keep all processes in same queue?
- Separate processes in a queue for each priority?

# Setting Priorities

## Priority can be static

- Some jobs always have higher priority than others
- Problem: **Starvation**

## Priority can be dynamically chosen by system

- Multilevel Feedback Queue Scheduling
- Decrease priority of compute-bound jobs
- Increase priority of interactive and I/O-bound jobs
- Many different policies possible...

## Example: Solaris Time-Sharing scheduler...