Today

- Process Scheduling
- Process Management

Project 1

- Questions?

> Operating systems are like underwear — nobody really wants to look at them.
> -- Bill Joy
> Co-Founder Sun Microsystems

Core-and-Driver Analogy

zipcar®
wheels when you want them

Core #1 **Core #2**

The machine has a bank of CPU cores for threads to run on.

The OS allocates cores to threads (the "drivers").

Cores are hardware. They go where the driver tells them.

OS can force a switch of drivers any time.

Review: CPU Scheduling Policy

- In designing the CPU scheduler there are two major policy questions that must be answered:
  - Under what circumstances will the scheduler be invoked?
    - Non-preemptive vs. Preemptive scheduling
  - When the scheduler is invoked, what criterion will it use to select from the ready queue the next process to run?
    - Scheduling Algorithm

Review: Scheduling Opportunities

- There are four opportunities for the CPU scheduler to select a new process to run:
  1. The running process blocks (running → waiting)
  2. A new process is created (new → ready)
  3. The running process is interrupted (running →ready)
     - Or yields
     - A process may also unblock. (waiting →ready)
  4. A process exits. (running→terminated)

Review

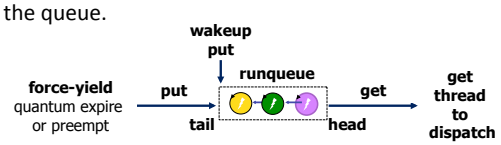
- What are metrics we can use to determine process/thread scheduling efficiency?
- What are algorithms we can use to schedule jobs?

Review: Scheduler Metrics

- Response time or latency, responsiveness
  - How long does it take to complete a task or request? (R)
  - Say a task takes D time units of work (its service demand)
    - But how long does it spend waiting for service?
- Throughput
  - How many tasks/requests complete per unit of time? (X)
  - Utilization: what % of time is each core/device busy? (U)
- Meet deadlines, reduce jitter for periodic tasks
  - e.g., videos and other continuous media

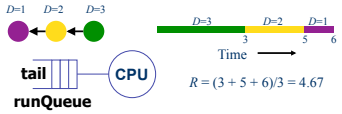
A simple policy: FCFS

- The most basic scheduling policy is first-come-first-served (FCFS), also called first-in-first-out (FIFO).
- FCFS is like the checkout line at the Kwik-e-mart
- Maintain a queue ordered by time of arrival.
- `GetNextToRun` selects from the front (head) of the queue.



Evaluating FCFS

- How well does FCFS achieve the goals?
- Throughput. FCFS is as good as any non-preemptive policy.
  - ….if the CPU is the only schedulable resource in the system.
- Fairness. FCFS is intuitively fair…sort of.
  - "The early bird gets the worm"…and everyone is fed… eventually.
- Response time. Long jobs keep everyone else waiting.
  - Consider service demand (D) for a process/job/thread.

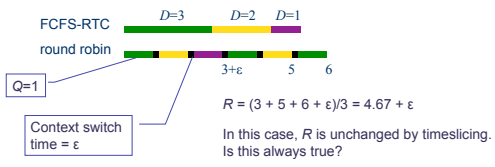


Non-Preemptive vs Preemptive

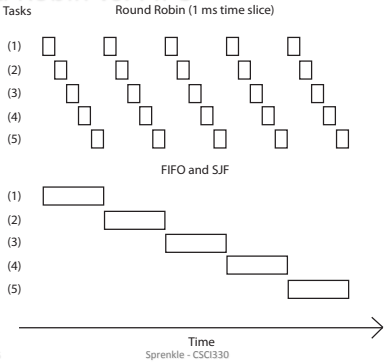
- Depending upon which scheduling opportunities are used by a scheduler, the scheduling can be:
  - **Non-Preemptive**: The scheduler will allow the running process to continue to run as long as it remains ready (i.e., doesn't block or exit).
  - **Preemptive**: The scheduler may set aside the running process in favor or another at any scheduling opportunity
    - Enables time-sharing, priority scheduling

Preemptive FCFS: Round Robin

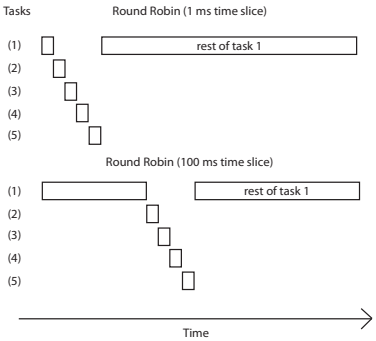
- Preemptive timeslicing is one way to improve fairness of FCFS.
- If job does not block or exit, force an involuntary context switch after each quantum Q of CPU time.
- FCFS without preemptive timeslicing is "run to completion" (RTC).
- FCFS with preemptive timeslicing is called round robin.



Round Robin vs. FIFO



Round Robin



Evaluating Round Robin

$R = (5+6)/2 = 5.5$

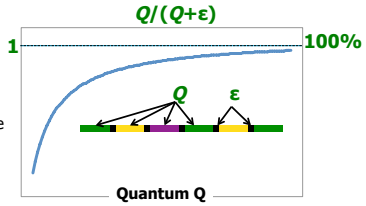
$R = (2+6+\varepsilon)/2 = 4 + \varepsilon$

- **Response time.** RR reduces response time for short jobs.
  - For a given load, wait time is proportional to the job's total service demand D.
- **Fairness.** RR reduces variance in wait times.
  - But: RR forces jobs to wait for other jobs that arrived later.
- **Throughput.** RR imposes extra context switch overhead.
  - Degrades to FCFS-RTC with large Q.

Overhead and Goodput

- **Context switching is overhead: "wasted effort".**
  - It is a cost that the system imposes in order to get the work done. It is not actually doing the work.

Efficiency or goodput
What percentage of the time is the busy resource doing useful work?

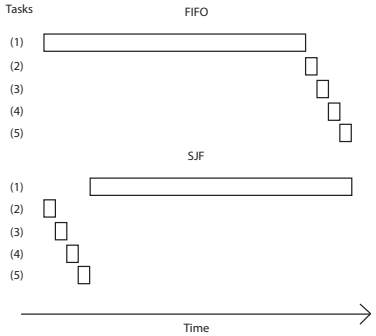


Minimizing Response Time: SJF (STCF)

- **Shortest Job First** (SJF) is provably optimal if the goal is to minimize average-case R.
  - Also called Shortest Time to Completion First (STCF) or Shortest Remaining Processing Time (SRPT)
  - Example: express lanes at the MegaMart
- Idea: get short jobs out of the way quickly to minimize the number of jobs waiting while a long job runs.
  - Intuition: longest jobs do the least possible damage to the wait times of their competitors.

$R = (1 + 3 + 6)/3 = 3.33$

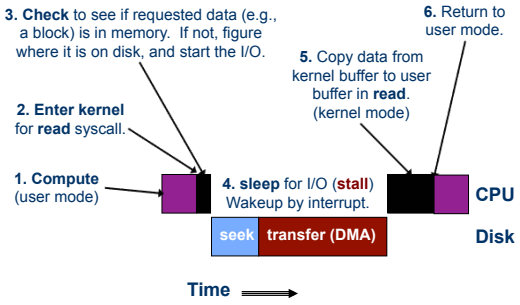
FIFO vs. SJF



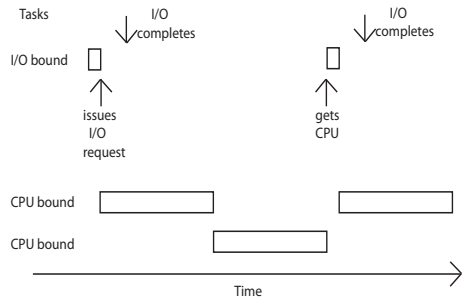
The Process Mix

- Two broad classes of processes:
  - **CPU Bound**: A process that is spending most of its time doing CPU operations.
  - **I/O Bound**: A process that is spending most of its time doing I/O operations.
- Processes can switch between being CPU Bound and being I/O Bound during their execution

Anatomy of a read

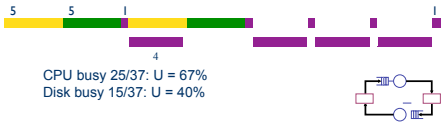


Mixed Workload



Two Schedules for CPU/Disk

1. Naive **Round Robin**



CPU busy 25/37: U = 67%
Disk busy 15/37: U = 40%

2. Add internal priority boost for I/O completion

CPU busy 25/25: U = 100%
Disk busy 15/25: U = 60%

33% improvement in utilization
When there is work to do,
U == efficiency.
More U means better throughput.

Estimating Time-to-Yield

- How to predict which job/task/thread will have the shortest demand on the CPU?
  - If you don't know, then guess.
    - Weather report strategy: predict future D from the recent past.
- We can "guess" well by using adaptive internal priority.
  - Common technique: multi-level feedback queue.
  - Set N priority levels, with a timeslice quantum for each.
  - If thread's quantum expires, drop its priority down one level.
    - "It must be CPU bound." (mostly exercising the CPU)
  - If a job yields or blocks, bump priority up one level.
    - "It must be I/O bound." (blocking to wait for I/O)

Example from Linux

Tasks are determined to be I/O-bound or CPU-bound based on an interactivity heuristic.

A task's interactiveness metric is calculated based on how much time the task executes compared to how much time it sleeps.

Note that because I/O tasks schedule I/O and then wait, an I/O-bound task spends more time sleeping and waiting for I/O completion.

This increases its interactive metric.

Multilevel Feedback Queue

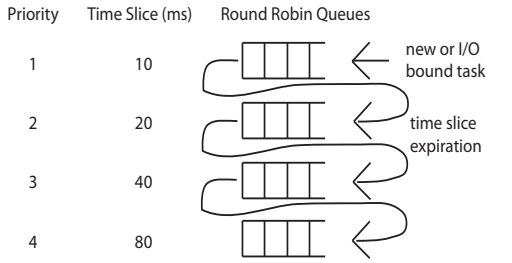
- Many systems (e.g., Unix variants) implement internal priority using a multilevel feedback queue.
- **Multilevel**. Separate queue for each of N priority levels.
  - Use RR on each queue
  - Look at queue i+1 only if queue i is empty.
- **Feedback**. Factor previous behavior into new job priority.

`GetNextToRun` selects job at the head of the highest priority queue: constant time, no sorting

ready queues indexed by priority

Priority of CPU-bound jobs decays with system load and service received.

Multilevel Feedback Queue: MFQ



TODO

- Project 1 due Wednesday