CPU Scheduling
(Chapters 7-11)

CS 4410
Operating Systems

[R. Agarwal, L. Alvisi, A. Bracy, M. George, F.B. Schneider, E.G. Sirer, R. Van Renesse]
The Problem

You’re the cook at State Street Diner
  - customers continuously enter and place orders 24 hours a day
  - dishes take varying amounts to prepare

What is your *goal*?
  - minimize average turnaround time?
  - minimize maximum turnaround time?
  - maximize throughput

Which *strategy* achieves your goal?
Goals depend on context

What if instead you are:

• the owner of an (expensive) container ship and have cargo across the world
• the head nurse managing the waiting room of the emergency room
• a student who has to do homework in various classes, hang out with other students, eat, and occasionally sleep
Schedulers in the OS

- **CPU Scheduler** selects a process to run from the run queue
- **Disk Scheduler** selects next read/write operation
- **Network Scheduler** selects next packet to send or process
- **Page Replacement Scheduler** selects page to evict

We’ll focus on **CPU Scheduling**
Kernel Operation (conceptual, simplified)

1. Initialize devices
2. Initialize “first process”
3. while (TRUE) {
   • while device interrupts pending
     - handle device interrupts
   • while system calls pending
     - handle system calls
   • **if run queue is non-empty**
     - select process and switch to it
   • otherwise
     - wait for device interrupt
}


Job Characteristics

Job or Task
• e.g., mouse click, web request, shell command, …

Job Arrival time

Job Execution time
• Time needed to run the task without contention

Nomenclature warning: no agreement on any of these terms or the ones that follow
Important Metrics of Scheduling

- **Execution Time**: sum of green periods
- **Total Waiting Time**: sum of red periods
- **Turnaround Time**: sum of both
Performance Terminology

**Turnaround time:** How long?
- User-perceived time to complete some job.

**Response time:** When does it start?
- User-perceived time before job can produce first output.

**Total Waiting Time:** How much thumb-twiddling?
- Time on the run queue but not running.
Throughput: How many jobs over time?
• The rate at which jobs are completed.

Predictability: How consistent?
• Low variance in turnaround time for repeated jobs.

Overhead: How much useless work?
• Time lost due to switching between jobs.

Fairness: How equal is performance?
• Equality in the resources given to each job.

Starvation: How bad can it get?
• The lack of progress for one job, due to resources given to higher priority jobs.
The Perfect Scheduler

- Minimizes response time and turnaround time
- Maximizes throughput
- Maximizes utilization (aka “work conserving”):
  - keeps all devices busy
- Meets deadlines:
  - think watching a video, car brakes, etc.
- Is Fair:
  - everyone makes progress, no one starves
- Is Envy-Free:
  - no job wants to switch its schedule with another

No such scheduler exists! 😞
When does scheduler run?

**Non-preemptive**

Job runs until it voluntarily yields CPU:
- job blocks on an event (e.g., I/O or P(sem))
- job explicitly *yields*
- job terminates

**Preemptive**

All of the above, plus:
- Timer and other interrupts
  - When jobs cannot be trusted to yield explicitly
- Incurs some context switching overhead
Jobs switch between CPU & I/O bursts

CPU-bound jobs: Long CPU bursts

Matrix multiply

I/O-bound jobs: Short CPU bursts

emacs

Problems:
• don’t know job’s type before running
• jobs also change over time
Basic scheduling algorithms:

- First in first out (FIFO)
- Shortest Job First (SJF)
- Round Robin (RR)
First In First Out (FIFO)

Processes (jobs) $P_1$, $P_2$, $P_3$ with execution time 12, 3, 3
All have same arrival time

Scenario 1: schedule order $P_1$, $P_2$, $P_3$

Average Turnaround Time: \[rac{(12+15+18)}{3} = 15\]

Scenario 2: schedule order $P_2$, $P_3$, $P_1$

Average Turnaround Time: \[rac{(3+6+18)}{3} = 9\]
FIFO Roundup

The Good

+ Simple
+ Low-overhead
+ No Starvation

The Bad

- Average turnaround time very sensitive to schedule order

The Ugly

- Not responsive to interactive jobs
How to minimize average turnaround time?
Shortest Job First (SJF)

Schedule in order of estimated execution\(^\dagger\) time

Scenario: each job takes as long as its number

Average Response Time: \( \frac{(1+3+6+10+15)}{5} = 7 \)

\( \text{P}_1 \)  \( \text{P}_2 \)  \( \text{P}_3 \)  \( \text{P}_4 \)  \( \text{P}_5 \)

Time 0  1  3  6  10  15

Would another schedule improve avg turnaround time?

\( \text{\dagger with preemption, remaining execution time} \)
FIFO vs. SJF

Effect on the short jobs is huge. Effect on the long job is small.
Shortest Job First Prediction

How to approximate duration of next CPU-burst
• Based on the durations of the past bursts
• Use past as a predictor of the future

• No need to remember entire past history!

Use exponential moving average:

\[ \tau_n \quad \text{predicted duration of n}^{\text{th}} \text{ CPU burst} \]

\[ \tau_{n+1} \quad \text{predicted duration of (n+1)}^{\text{th}} \text{ CPU burst} \]

\[ \tau_{n+1} = \alpha \tau_n + (1- \alpha) t_n \]

0 ≤ \( \alpha \) ≤ 1, \( \alpha \) determines weight placed on past behavior
SJF Roundup

The Good

+ Optimal average turnaround time

The Bad

– Pessimal variance in turnaround time
– Needs estimate of execution time

The Ugly

– Can starve long jobs
Round Robin (RR)

• Each job allowed to run for a quantum
• Context is switched (at the latest) at the end of the quantum

What is a good quantum size?
• Too long, and it morphs into FIFO
• Too short, and much time is wasted on context switching
• Typical quantum: about 100X cost of context switch (~100ms vs. << 1 ms)
Effect of Quantum Choice in RR

Tasks

Round Robin (1 ms time slice)

- (1)
- (2)
- (3)
- (4)
- (5)

Rest of Task 1

Tasks

Round Robin (100 ms time slice)

- (1)
- (2)
- (3)
- (4)
- (5)

Rest of Task 1

Time
Round Robin vs. FIFO

Tasks of same length that start ~same time

At least it’s fair?

Optimal avg. turnaround time!

Round Robin (1 ms time slice)

FIFO and SJF
More Problems with Round Robin

Mixture of one I/O Bound tasks + two CPU Bound Tasks
I/O bound: compute, go to disk, repeat

$\Rightarrow$ *RR doesn’t seem so fair after all*....

[Diagram showing the execution of tasks with I/O Bound and CPU Bound requests and the associated time intervals.]
RR Roundup

The Good
- No starvation
- Can reduce response time

The Bad
- Context switch overhead
- Mix of I/O and CPU bound

The Ugly
- Bad avg. turnaround time for equal length jobs
Priority-based scheduling algorithms:

- Priority Scheduling
- Real-Time Scheduling
- Multi-level Queue Scheduling
- Multi-level Feedback Queue Scheduling
Priority Scheduling

• Assign a number to each job and schedule jobs in (increasing) order

• Can implement any scheduling policy
  • e.g., reduces to SJF if $\tau_n$ is used as priority

• To avoid starvation, improve job’s priority with time (aging)
Real-Time Scheduling

Real-time processes have timing constraints
• Expressed as deadlines or rate requirements

Common RT scheduling policies
• Earliest deadline first (EDF) (priority = deadline)
• Priority Inheritance
  • Recall priority inversion: high priority process wants to get lock held by low priority process
  • Solution: High priority process (needing lock) temporarily donates priority to lower priority process (with lock)
Multi-Level Queue Scheduling

Multiple ready queues based on job “type”

- system jobs
- interactive jobs
- background batch jobs

Different queues may be scheduled using different algorithms

- Queue classification difficult
  (Job may have CPU-bound and interactive phases)
- No queue re-classification
Multi-Level Feedback Queues

- Like multilevel queue, but assignments are not static
- Jobs start at the top
  - Use your quantum? move down
  - Don’t? Stay where you are

Need parameters for:
- Number of queues
- Scheduling alg. per queue
- When to upgrade/downgrade job
Thread Scheduling

Threads share code & data segments
- **Option 1:** Ignore this fact
- **Option 2:** Gang scheduling*
  - all threads of a process run together (pink, green)

- **Option 3:** Space-based affinity*
  - assign tasks to processors (pink → P1, P2)
    + Improve cache hit ratio

*multiprocessor only