CPU Scheduling
(Chapters 7-11)
Mechanism and Policy

Mechanism

enables a functionality

Policy

determines how that functionality should be used

Mechanisms should not determine policies!
Kernel Operation (conceptual, simplified)

Initialize devices
Initialize “first process”

while (TRUE) {
    while device interrupts pending
        - handle device interrupts
    while system calls pending
        - handle system calls
    if run queue is non-empty
        - select a runnable process and switch to it
    otherwise
        - wait for device interrupt
}
The Problem

You are the cook at the State Street Diner
Customers enter and place orders 24 hours a day
Dishes take varying amounts of time to prepare

What are your goals?
Minimize average turnaround time?
Minimize maximum turnaround time?

Which strategy achieves your goal?
Context matters!

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 an 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 next process to run from the ready queue

Disk scheduler selects next read/write operation

Network scheduler selects next packet to send or process

Page Replacement scheduler selects page to evict
Scheduling processes

OS keeps PCBs on different queues

Ready processes are on ready queue - OS chooses one to dispatch

Processes waiting for I/O are on appropriate device queue

Processes waiting on a condition are on an appropriate condition variable queue

OS regulates PCB migration during life cycle of corresponding process
Why scheduling is challenging

Processes are not created equal!

CPU-bound process: long CPU bursts
- mp3 encoding, compilation, scientific applications

I/O-bound process: short CPU bursts
- index a file system, browse small web pages

Problem
- don't know jobs type before running it
- jobs behavior can change over time
Job Characteristics

Job: A task that needs a period of CPU time

A user request: e.g., mouse click, web request, shell command...

Defined by:

Arrival time
When the job was first submitted

Execution time
Time needed to run the task in isolation

Deadline
By when the task must have completed (e.g. for videos, car brakes...)

Metrics

Response time
How long between job's arrival and first time job runs?

Total waiting time
How much time on ready queue but not running?
sum of “red” intervals below

Execution time: sum of “green” intervals

Turnaround time: “red” + “green”
Time between a job's arrival and its completion

Throughput: jobs completed/unit of time
Other Concerns

**Fairness**: Who get the resources?
Equitable division of resources

**Starvation**: How bad can it get?
Lack of progress by some job

**Overhead**: How much useless work?
Time wasted switching between jobs

**Predictability**: How consistent?
Low variance in response time for repeated requests
The Perfect Scheduler

Minimizes response time and turnaround time for each job
Maximizes overall throughput
Maximizes resource utilization ("work conserving")
Meets all deadlines
Is fair: everyone makes progress, no one starves
Is Envy-Free: no job wants to switch its schedule with another
Has zero overhead

Alas, no such scheduler exists...
When does the Scheduler Run?

Non-preemptive

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

Preemptive

- all of the above, plus timer and other interrupts
- when processes can’t be trusted
- incurs some context switching overhead
Context switch overhead

Cost of saving registers

Cost of scheduler determining which process to run next

Cost of restoring register

Cost of flushing caches
  L1, L2, L3, TLB
Basic Scheduling Algorithms

FIFO (First In First Out)

SJF (Shortest Job First)

EDF (Earliest Deadline First)
  preemptive

Round Robin
  preemptive

Shortest Remaining Time First (SRTF)
  preemptive
FIFO

Jobs with compute time 12, 3, 3

Job arrival

Average Turnaround Time: 
\[(12 + 15 + 18)/3 = 15\]
FIFO

Jobs with compute time 12, 3, 3

Job arrival

Average Turnaround Time:
\[
\frac{12 + 15 + 18}{3} = 15
\]

Average turnaround time very sensitive to arrival time!
FIFO Roundup

The Good
Simple
Low overhead
No starvation

The Bad
Average turnaround time
very sensitive to arrival time

The Ugly
Not responsive to interactive tasks
How to minimize average turnaround time?
SJF: Shortest Job First

Schedule jobs in order of estimated completion time
**SJF: Shortest Job First**

Schedule jobs in order of estimated completion time

\[ C_1 \]

0 \hspace{1cm} 1 \hspace{1cm} 2.5

2.5

3

2

4
SJF: Shortest Job First

Schedule jobs in order of estimated completion time

\[
\begin{array}{c|cccccc}
    & 0 & 1 & 2.5 & 4.5 & 7 & 10 & 14 \\
\hline
C_1 & & & & & & & \\
\end{array}
\]

Average Turnaround time (att): \( \frac{39}{6} = 6.5 \)

Would a different schedule produce a lower turnaround time?
SJF: Shortest Job First

Schedule jobs in order of estimated completion time

\[
\begin{array}{c}
0 & 1 & 2.5 & 4.5 & 7 & 10 & 14
\end{array}
\]

Average Turnaround time (att): \( \frac{39}{6} = 6.5 \)

Would a different schedule produce a lower turnaround time?
<table>
<thead>
<tr>
<th>Colloquium Date</th>
<th>Speaker</th>
<th>Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tuesday, January 19</td>
<td>Sarah Dean</td>
<td>ML</td>
</tr>
<tr>
<td>Tuesday, January 26</td>
<td>Ranjay Krishna</td>
<td>Vision</td>
</tr>
<tr>
<td>Thursday, January 28</td>
<td>Josh Alman</td>
<td>Theory</td>
</tr>
<tr>
<td>Tuesday, February 2</td>
<td>Akshitha Sriraman</td>
<td>Systems</td>
</tr>
<tr>
<td>Thursday, February 4</td>
<td>Bryan Wilder</td>
<td>AI</td>
</tr>
<tr>
<td>Tuesday, February 9</td>
<td>Caroline Lemieux</td>
<td>Software Engineering</td>
</tr>
<tr>
<td>Thursday, February 11</td>
<td>Jonathan Gammell</td>
<td>Robotics</td>
</tr>
<tr>
<td>Tuesday, February 16</td>
<td>Kexin Rong</td>
<td>Databases</td>
</tr>
<tr>
<td>Thursday, February 18</td>
<td>Manuel Rigger</td>
<td>Software Engineering</td>
</tr>
<tr>
<td>Tuesday, February 23</td>
<td>Angelique Maria Taylor</td>
<td>Robotics</td>
</tr>
<tr>
<td>Colloquium Date</td>
<td>Speaker</td>
<td>Area</td>
</tr>
<tr>
<td>----------------------------</td>
<td>----------------------------</td>
<td>------------------</td>
</tr>
<tr>
<td>Thursday, February 25</td>
<td>Abhishek Gupta</td>
<td>Robotics</td>
</tr>
<tr>
<td>Tuesday, March 2</td>
<td>Reuth Mirsky</td>
<td>AI</td>
</tr>
<tr>
<td>Thursday, March 4</td>
<td>Mina Konakovic Lukovic</td>
<td>Graphics</td>
</tr>
<tr>
<td>Thursday, March 11</td>
<td>Fraser Brown</td>
<td>Security</td>
</tr>
<tr>
<td>Tuesday, March 16</td>
<td>Jonathan Frankle</td>
<td>ML</td>
</tr>
<tr>
<td>Thursday, March 18</td>
<td>Riad Wahby</td>
<td>Security</td>
</tr>
<tr>
<td>Tuesday, March 23</td>
<td>Sanmi Koyejo</td>
<td>ML</td>
</tr>
<tr>
<td>Thursday, March 25</td>
<td>Huacheng Yu</td>
<td>Theory</td>
</tr>
<tr>
<td>Thursday, April 1</td>
<td>Michał Deriziński</td>
<td>ML</td>
</tr>
<tr>
<td>Tuesday, April 6</td>
<td>Leilani Gilpin</td>
<td>AI</td>
</tr>
<tr>
<td>Thursday, April 8</td>
<td>J ia-Bin Huang</td>
<td>Vision</td>
</tr>
<tr>
<td>Tuesday, April 13</td>
<td>Maggie Makar</td>
<td>ML</td>
</tr>
<tr>
<td>Thursday, April 15</td>
<td>Sanjiban Choudhuri</td>
<td>Robotics</td>
</tr>
<tr>
<td>Tuesday, April 20</td>
<td>Danfei Xu</td>
<td>Robotics</td>
</tr>
<tr>
<td></td>
<td>Aditi Raghunathan</td>
<td>ML</td>
</tr>
<tr>
<td>Tuesday, April 27</td>
<td>Mingmin Zhao</td>
<td>IoT/Networking</td>
</tr>
</tbody>
</table>
Graduate School?

Resume building

March 3rd, 7:00 pm
Register at: https://tinyurl.com/GSW1Register

Perspectives on Graduate School

interaction with current graduate students
March 11, 6:00 pm
Register at: https://tinyurl.com/gradschoolperspectives

CMMRS

Apply at: https://cmmrs.mpi-sws.org/how-to-apply/
SJF Roundup

The Good
- Optimal average turnaround time

The Bad
- Pessimal variance in turnaround time
- Need to estimate execution time

The Ugly
- Can starve long jobs
Shortest Process Next (SJF for interactive jobs)

Enqueue in order of estimated completion time

Use recent history as indicator of near future

Let $\alpha$ determines weight placed on past behavior
Earliest Deadline First (EDF)

Schedule in order of earliest deadline

If a schedule exists that meets all deadlines, then EDF will generate that schedule!

does not even need to know the execution times of the jobs

Informal Proof

Let $S$ be a schedule of a set of jobs that meets all deadlines
Let $j_1$ and $j_2$ be two neighboring jobs in $S$ so that $j_1$.deadline > $j_2$.deadline
Let $S'$ be $S$ with $j_1$ and $j_2$ switched

$S'$ also meets all deadlines!
Repeat until sorted (i.e., bubblesort)

Resulting schedule is EDF
Earliest Deadline First (EDF)

Schedule in order of earliest deadline

If a schedule exists that meets all deadlines, then EDF will generate that schedule!

...but only if tasks only need the processor!

does not even need to know the execution times of the jobs

Informal Proof

Let S be a schedule of a set of jobs that meets all deadlines
Let and be two neighboring jobs in S so that \( .\text{deadline} > .\text{deadline} \)
Let S’ be S with and switched
S’ also meets all deadlines!
Repeat until sorted (i.e., bubblesort)
Resulting schedule is EDF
When EDF fails

Two jobs:

\( j_1 \): deadline at 12; 1 unit of computation, 10 of I/O

\( j_2 \): deadline at 10; 5 units of computation
When EDF fails

Two jobs:

: deadline at 12; 1 unit of computation, 10 of I/O

\( j_2 \) : deadline at 10; 5 units of computation
EDF Roundup

The Good

Meets deadlines if possible (but...)
Free of starvation

The Bad

Does not optimize other metrics

The Ugly

Cannot decide when to run jobs without deadlines
Round Robin

Each process is allowed to run for a quantum

Context is switched (at the latest) at the end of the quantum — preemption!

Next job to run is the one that hasn’t run for the longest amount of time

What is a good quantum size?
  Too long, and it morphs into FIFO
  Too short, and much time lost context switching

Typical quantum: about 100X cost of context switch
  (~100ms vs. << 1ms)
Round Robin vs FIFO

Jobs of about equal length (5 TU) start at about the same time

Average Turnaround time (RR):
\[
\frac{(5 + (10-1) + (15-2) + (20-3) + (25-4))}{5} = 13
\]

Average Turnaround time (FIFO/SJF):
\[
\frac{(21 + (22-1) + (23-2) + (24-3) + (25-4))}{5} = 21
\]
At least it is fair...?

Mix of one I/O-bound and two CPU-bound jobs

I/O-bound: compute; go to disk; repeat

I/O Bound

compute

go to disk

wait 190 ms

I/O Bound

compute

go to disk

I/O completes

CPU Bound

100ms quantum

CPU Bound

100ms quantum

Time
Round Robin Roundup

The Good
No starvation
Can reduce response time

The Bad
Overhead of context switching
Mix of I/O and CPU bound

The Ugly
Particularly bad average turnaround for simultaneous, equal length jobs
SJF

arrives at time 0; arrive at time 10

Average Turnaround Time:
\[ 100 + (110 - 10) + (120 - 10) / 3 \]
\[ = 103.33 \]
SJF + Preemption

arrives at time 0; arrive at time 10

With a preemptive scheduler — SRTF

At end of each quantum, scheduler selects job with the least remaining time to run next

Average Turnaround Time:
\[ \frac{100 + (110 - 10) + (120 - 10)}{3} = 103.33 \]

Shortest Remaining Time First

Often same job is selected, avoiding a context switch...

...but new short jobs see improved response time

Average Turnaround Time:
\[ \frac{(120 - 0) + (20 - 10) + (30 - 10)}{3} = 50 \]
SRTF Roundup

**The Good**
- Good response time and turnaround time of I/O bound processes

**The Bad**
- Bad turnaround time and response time for CPU bound processes
- Need estimate of execution for each job

**The Ugly**
- Starvation
Priority Scheduling

Assign a number (priority) to each job and schedule jobs in priority order.

Reduces to SRTF when using as priority the estimate of the execution time.

To avoid starvation:
- change job’s priority with time (aging)
- select jobs randomly, weighted by priority
Multi-level Feedback Queue (MFQ)

Scheduler learns characteristics of the jobs it is managing

Uses the past to predict the future

Favors jobs that used little CPU...

...but can adapt when the job changes its pattern of CPU usage
## The Basic Structure

<table>
<thead>
<tr>
<th>Queue</th>
<th>Jobs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q8</td>
<td>A</td>
</tr>
<tr>
<td>Q7</td>
<td></td>
</tr>
<tr>
<td>Q6</td>
<td></td>
</tr>
<tr>
<td>Q5</td>
<td>C</td>
</tr>
<tr>
<td>Q4</td>
<td></td>
</tr>
<tr>
<td>Q3</td>
<td></td>
</tr>
<tr>
<td>Q2</td>
<td>D</td>
</tr>
<tr>
<td>Q1</td>
<td></td>
</tr>
</tbody>
</table>

Queues correspond to different priority levels:
- higher is better

Scheduler runs job in queue i if no other job in higher queues.

Each queue runs RR.

Parameter:
- how many queues?

How are jobs assigned to a queue?
**Moving down**

Q8  ➔  A  ➔  B  
Q7  
Q6  
Q5  ➔  C  
Q4  
Q3  
Q2  
Q1  ➔  D

- Job starts at the top level
- If it uses full quantum before giving up CPU, moves down
- Otherwise, it stays where it is

What about I/O?

- Job with frequent I/O will not finish its quantum and stay at the same level

Parameter

- quantum size for each queue
Moving Up

A job's behavior can change
After a CPU-bound interval, process may become I/O bound

Must allow jobs to climb up the priority ladder...
As simple as periodically placing all jobs in the top queue, until they percolate down again

Parameter
time before jobs are moved up
Sneeeakekkyyyy...

Say that I have a job that requires a lot of CPU

Start at the top queue

If I finish my quantum, I’ll be demoted...

...just give up the CPU before my quantum expires!

Better accounting

fix a job’s time budget at each level, no matter how it is used
Linux’s “Completely Fair Scheduler” (CFS)

Let “Spent Execution Time” (SET) to be the amount of time that a process has been executing.

Scheduler selects process with lowest SET.

Let $\triangle$ be some time (typically, 50ms or so).

Let $N$ be the number of processes on the run queue.

Process runs for $\triangle/N$ time.

There is a minimum value too.

If it uses up this quantum, reinsert into the queue.

$\text{SET} += \frac{\triangle}{N}$

Computing of elapsed SET can be weighed by priority value.

Processes that move to a waiting queue, upon returning to the READY queue, have SET initialized to the minimum SET of any process on the READY queue.
Multiprocessor Scheduling: Sequential Applications

A web server
A thread per user connection
Threads are I/O bound (access disk/network)
   favor short jobs!

An MFQ, right?
Idle processors take task off MFQ
Only one processor at a time gets access to MFQ
If thread blocks, back on the MFQ
Single MFQ
Considered Harmful

Contention on MFQ lock

Limited cache reuse
since threads hop from processor to processor

Cache coherence overhead
processor need to fetch current MFQ state
on a uniprocessor, likely to be in the cache
on a multiprocessor, likely to be in the cache of another processor
2-3 orders of magnitude more expensive to fetch
To Each (Process), it's Own (MFQ)

Processors use affinity scheduling

each thread is run repeatedly on the same processors

maximizes cache reuse

more complex to achieve on a single MFQ

Idle processors can steal work from other processors

re-balance load at the cost of some loss of cache efficiency

only if it is worth the time of rewarming the cache!
Multiprocessor Scheduling: Parallel Applications

Application is decomposed in parallel tasks

- granularity roughly equal to available processors
- or poor cache reuse

Often (e.g., MapReduce) using bulk synchronous parallelism (BSP)

- tasks are roughly of equal length
- progress limited by slowest processor
Oblivious Scheduling
Each process time-slices its ready list independently

Gang Scheduling
Schedule all tasks from the same program together

Four applications, each with four threads

Length of BSP step determined by last scheduled thread!