Lecture: Scheduling

Questions:
Difference between alloc/scheduling?
Pre-emptive vs. non?
Different policies: how behave?
(+’s, -’s)

Review
Processes + low-level mechanisms
how to pre-empt, save/restore

⇒ Resources: what OS’s manage
things that a process needs/uses
(CPU, disk, memory)

Today: CPU resource scheduling

Resources: Not all alike
Preemptible
Can take resource away, use for something else, give back later
e.g. CPU

Non-preemptible
Once given away, can’t really take back (until volunteered)
e.g. Disk block, terminal

Key: Given resource/demands ⇒ determine how to manage it
Decisions about Resources

Allocation: which process gets which resource?

Space Sharing: divide up resource, give some to each when resources not easily preemptible

E.g. File space

Scheduling: how long process keeps resource

(best order)

Time Sharing: more resources can be requested than available

Resource must be preemptible

E.g. CPU scheduling

Role of Dispatcher vs. Role of Scheduler

Dispatcher: low-level mechanism

Must context switch

{save state in PCB for old
load state of new
}

Scheduler: higher-level policy

which to run

could have both

space + time

Sharing of CPU in parallel system, memory
Performance Metrics

- Minimize waiting time
  - Minimize turn-around time
  - Minimize "cumulative wait" →

- Minimize overhead
  ↓ context switches (costly?)

- Fairness
  each process = same % of CPU

When does Sched make decisions?

Minimal: Non-preemptive

1) Process blocks on I/O
2) Process terminates
   ⇒ (remain scheduled until relinquished)

Also: Pre-emptive

+ (I/O interrupt)
  1) event completes: blocked → ready
  2) timer interrupts
     ⇒ can run what it wants to
Algorithms: Overview

1. Process: CPU activity, I/O activity interleaved
   - if CPU bound: long CPU bursts
   - I/O bound: short CPU bursts
   - Best algorithm depends on workload, environment
   - Specialized: only certain kinds of jobs
   - Much knowledge about them
   - General purpose: want to do well for all types

Projects

FCFS

> Simplest algorithm:
> Not preemptive
> Why bad?
  - Wait/service time and the depends on order of arrival
  - Unfair to later jobs
  - E.g. 3 jobs (A, B, C) arrive at same time

Gantt Chart

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>24</td>
<td>3</td>
</tr>
<tr>
<td>A</td>
<td>3</td>
<td>C</td>
</tr>
</tbody>
</table>

Cum. wait time: (0 + 24 + 27) / 3 = 17
Turnaround: \( \frac{24 + 27 + 36}{3} = 27 \)

Note: in multiprogrammed world, can put job @ end of Q upon I/O

Book

Discussion

> How to implement?
  - FIFO queue
  - Simple

What worked well here:
  - Simple 2 job example
  - A: 100
  - B: 20
  - Use throughout
Instance of Convoq Effect
(Slow truck in front of fast cars)

CPU

\[
\begin{array}{c|c|c|c}
A & B & C & A \\
\hline
& & & \\
\end{array}
\]

Disk

\[
\begin{array}{c|c|c|c|c|c}
\text{Idle} & \text{Idle} & \text{Write} & \text{Read} & \text{Idle} & ?
\end{array}
\]

- General problems
- Can also reduce I/O device utilization
- The short job waiting time

**SJT (Shortest Job First)**

- Should minimize wait/turnaround time

\[
\begin{array}{c|c|c}
1 & 2 & 3 \\
\hline
A & B & C \\
\end{array}
\]

Avg wait: \((6 + 3 + 6)/3 = 3\)

Avg Turn Around: \(3 + 6 + 30 = 36\)

- If \(T_A = T_B = T_C \Rightarrow FCFS\)

- Proverbially optimal (w/o preemption)
  
  Insight: Moving short jobs up improves them standing a lot while only hurting the long jobs a little often

- But, not practical
  
  How to know job length?
  
  (Maybe can use past \(\Rightarrow\) predict future)
ST to CF (Shortest Time to Completion First)

STCF = SJF w/ preemption

Problem: SJF is fine IF all jobs are in system

BUT, some arrive later!

\[
\text{Avg. Cum. wait} = \frac{(0 + 7 + 9 + 15)}{4} = 7.75
\]

\[
\text{Avg. wait} = \frac{(9 + 0 + 2 + 15)}{4} = 6.5
\]

w/ preemption!

10/04

What worked well:

- let them figure it out
- A:100, B:20
- A has run for: 175 vs 17

Note: total time is always same
(no magic here)

RR

\[
\text{Aug. Cum. wait} = \frac{\text{(low response time)}}{2}
\]

Advantages

- Fair, low average wait if job lengths vary widely
- Good interactive perf
- Low response time

=> preemption if still running at end
But, RR has problems

Poor when jobs nearly identical

\[
\begin{array}{c|c|c|c}
A & 10 & \ \ & \ \ \\
\hline
B & 20 & \ \ & \ \ \\
C & 30 & \ \ & \ \ \\
\end{array}
\]

Even FCFS is better!

Avg turn around time = \( (10 + 20 + 30) / 3 = 20 \)

\[
\begin{array}{c|c|c|c}
A & B & C \ \ & \ \ \\
\hline
\ \ & 20 & 29 & 30 \\
\end{array}
\]

= \( (20 + 29 + 30) / 3 = 29 \)

But, avg wait/response time is good!

\[
\begin{array}{c|c|c|c}
A & B & C \ \ & \ \ \\
\hline
\ \ & 1 & 2 & \ \ \\
\end{array}
\]

FCFS: \( (0 + 10 + 20) / 3 = 10 \) more interactive

Problem #2: Perf depends on length of time slice

if too high \( => FCFS \)

e.g. A 1 ms compute, 10 ms I/O

B just compute

CPU

Disk
> If time-slice too low, context switch overhead

\[ TS = 1 \text{ ms} \]

But STCF is still better

> How to approximate? \Rightarrow PRIORITIES

> Each process has priority run highest, RR among equal