Operating Systems

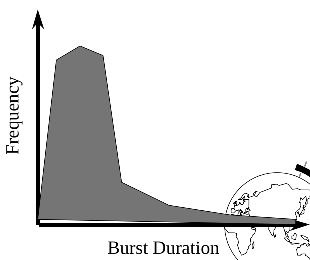
Process Scheduling
(Ch 4.2, 5.1 - 5.3)

Schedulers

- Short-Term
  - "Which process gets the CPU?"
  - Fast, since once per 100 ms
- Long-Term (batch)
  - "Which process gets the Ready Queue?"
- Medium-Term
  - "Which Ready Queue process to memory?"
  - Swapping
- We'll be talking about Short-Term, unless otherwise noted

CPU-IO Burst Cycle

```
add
read
```
(I/O Wait)
```
store
increment
write
```
(I/O Wait)



When does OS do Scheduling?

- Four times to re-schedule
  1. Running to Waiting (I/O wait)
  2. Running to Ready (*time slice*)
  3. Waiting to Ready (I/O completion)
  4. Termination
- #2 and #3 optional ==> "Preemptive"
- Timing may cause unexpected results
  - updating shared variable
  - kernel saving state

Question

- What performance criteria related to processes should the scheduler seek to optimize?
  - Ex: CPU minimize time spent in queue
  - Others?

Scheduling Criteria

1 CPU utilization (typically, 40% to 90%)
2 Throughput (processes/time, higher is better)
3 Waiting time (in queue, lower is better), or ...
3 Turn-around time (in queue plus run time)

- Maximize #1, #2 Minimize #3
- Note, response time often OS metric but is beyond short-term scheduler
  - Self-regulated by users (go home)
  - Bounded ==> Variance!

Scheduling Algorithms

- General
  - FCFS
  - SFJ
  - Priority
  - Round-Robin
- Specific
  - NT
  - Linux

First-Come, First-Served

Process	Burst Time
A	8
B	1
C	1

Gantt Chart

A	B	C

0 8 9 10

- Avg Wait Time (0 + 8 + 9) / 3 = 5.7

Shortest Job First

Process	Burst Time
A	8
B	1
C	1

B	C	A

0 1 2 10

- Avg Wait Time (0 + 1 + 2) / 3 = 1
- Optimal Avg Wait
- Prediction tough … Ideas?

Priority Scheduling

- Special case of SJF

Process	Burst Time	Priority
A	8	2
B	1	1
C	1	3

B	A	C

0 1 9 10

- Avg Wait Time (0 + 1 + 9) / 3 = 3.3

Priority Scheduling Criteria?

Priority Scheduling Criteria

- Internal
  - open files
  - memory requirements
  - *CPU time used* - *time slice expired (RR)*
  - *process age* - *I/O wait completed*
- External
  - $
  - department sponsoring work
  - *process importance*
  - *super-user (root)* - *nice*

Round Robin

- Fixed time-slice and Preemption

Process	Burst Time
A	5
B	3
C	3

A	B	C	A	B	C	A	B	C	A

8 9 11

- Avg = (8 + 9 + 11) / 3 = 9.3
- FCFS? SJF?

SOS: Dispatcher

- How is the next process chosen?
- Line 79 has an infinite loop. Why?
- There is no return from the `Dispatcher()` function call. Why not?
- See "TimerInterruptHandler()"
- Linux:
  - `/usr/src/linux/kernel/sched.c`
  - `/usr/src/linux/include/linux/sched.h`
  - `linux-pcb.h`

Round Robin Fun

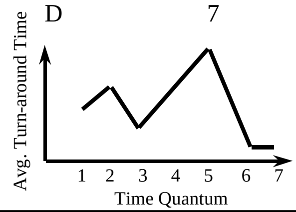
Process	Burst Time
A	10
B	10
C	10

- Turn-around time?
  - q = 10
  - q = 1
  - q --> 0

More Round Robin Fun

Process	Burst Time
A	6
B	3
C	1
D	7

Rule:
80% within
one quantum



Fun with Scheduling

Process	Burst Time	Priority
A	10	2
B	1	1
C	2	3

- Gantt Charts:
  - FCFS
  - SJF
  - Priority
  - RR (q=1)
- Performance:
  - Throughput
  - Waiting time
  - Turnaround time

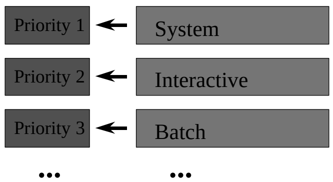
More Fun with Scheduling

Process	Arrival Time	Burst Time
A	0.0	8
B	0.4	4
C	1.0	1

- Turn around time:
  - FCFS
  - SJF
  - q=1 CPU idle
  - q=0.5 CPU idle

Multi-Level Queues

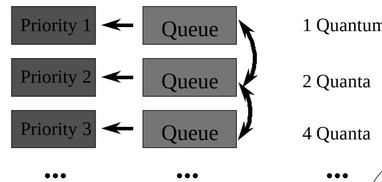
- Categories of processes, each at a priority level



- Run all in 1 first, then 2 …
- Starvation!
- Divide between queues: 70% 1, 15% 2 …

Multi-Level Feedback Queues

- Allow processes to move between prio levels
- Ex: time slice expensive but want interactive



- Consider process needing 100 quanta
  - 1, 4, 8, 16, 32, 64 = 7 swaps!
- Favor interactive users
- Most general. Used in WinNT and Linux

Outline

- Processes ✓
  - PCB ✓
  - Interrupt Handlers ✓
- Scheduling
  - Algorithms ✓
  - WinNT ←
  - Linux

Windows NT Scheduling

- Basic scheduling unit is a thread
  - For now, just think of a thread as a process
- Priority based scheduling per thread
- Preemptive operating system
- No shortest job first, no quotas

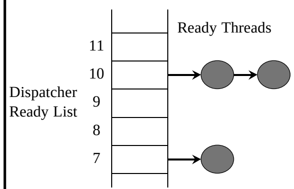
Priority Assignment

- NT kernel uses 31 *priority levels*
  - 31 is the highest; 0 is system idle thread
  - Realtime priorities: 16 - 31
  - Dynamic priorities: 1 - 15
- Users specify a *priority class*:
  - realtime (24) , high (13), normal (8) and idle (4)
  - and a *relative priority*:
    - highest (+2), above normal (+1), normal (0), below normal (-1), and lowest (-2)
  - to establish the *starting priority*
- Threads also have a *current priority*

Quantum

- Determines how long a thread runs once selected
- Varies based on:
  - NT Workstation or NT Server
  - Intel or Alpha hardware
  - Foreground/Background application threads
    - NOTE: NT 4.0 increases quantum for foreground threads while NT 3.5 increased priorities. Why?

Dispatcher Ready List



- Keeps track of all threads in the ready state
- Queue of threads assigned to each level
- (Multi-level feedback queue)

Selecting the Ready Thread

- Locates the highest priority thread that is ready to execute
- Scans dispatcher ready list
- Picks front thread in highest priority nonempty queue

- *When is this like round robin?*

Boosting and Decay

- When does the “feedback” occur?
- *Boost* priority
  - Event that “wakes” blocked thread
  - Boosts never exceed priority 15 for *dynamic*
  - *Realtime* priorities are not boosted
- *Decay* priority
  - by one for each quantum
  - decays only to starting priority (no lower)

Starvation Prevention

- Low priority threads may never execute
- “Anti-CPU starvation policy”
  - thread that has not executed for 3 seconds
  - boost priority to 15
  - double quantum
- Decay is swift not gradual after this boost

Linux Process Scheduling

- Two classes of processes:
  - Real-Time
  - Normal
- Real-Time:
  - Always run Real-Time above Normal
  - Round-Robin or FIFO
  - “Soft” not “Hard”

Linux Process Scheduling

- Normal: *Credit-Based*
  - process with most credits is selected
  - time-slice then lose a credit (0, then suspend)
  - no runnable process (all suspended), add to *every* process:
    `credits = credits/2 + priority`
- Automatically favors I/O bound processes

Questions

- True or False:
  - FCFS is optimal in terms of avg waiting time
  - Most processes are CPU bound
  - The shorter the time quantum, the better
- What is the *idle thread*? Where did we see it?