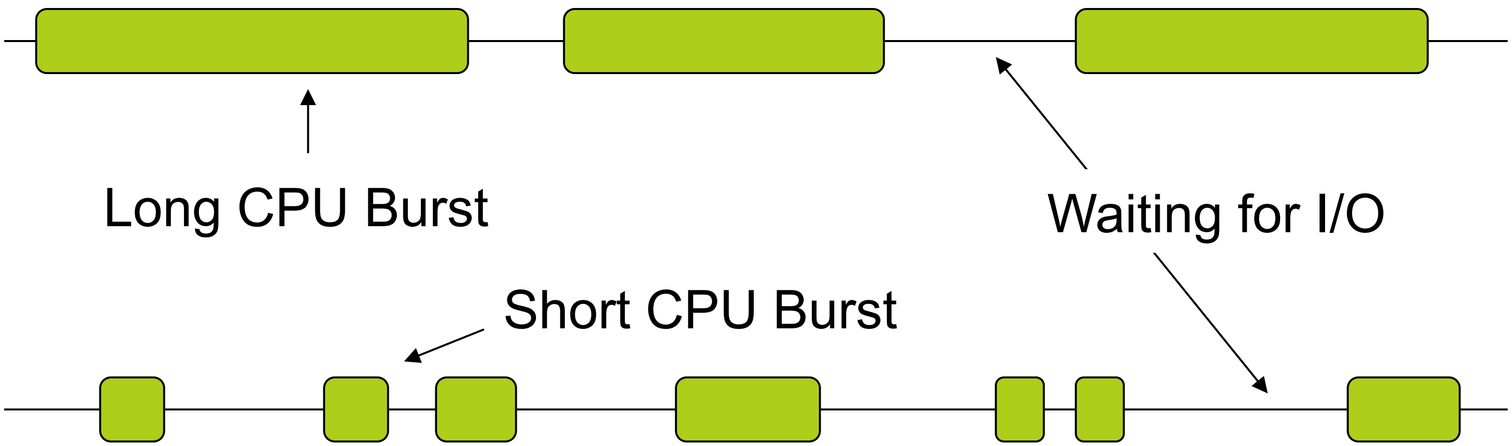
Scheduling Algorithms

Chapters 5

CPU Bursts



Ideally, we want the CPU **and** disk to be *always* busy.

CPU-Bound: Characterized by long CPU bursts
I/O-Bound: Characterized by short CPU bursts

Scheduling Decision Points

- Standard Approaches:
  - Process Creation
  - Process Termination
  - Blocking Call is issued
  - Process Yields voluntarily
- Pre-emptive
  - Blocking Call completes
  - Timer interrupt

Scheduling Algorithms

- Scheduling Criteria (Goals)
  - different goals for different systems
    - Batch
    - Interactive/General Purpose
    - Real-Time
- Scheduling algorithms attempt to maximize one or more "measurements"

Scheduling Algorithms

- For each algorithm discussed, we will assume we know the amount of time the CPU burst will last
  - Time is defined in clock cycles:
    - Faster computers have smaller clock cycles, but our numbers will remain the same

- Non-Preemptive: FCFS, SJF,
- Preemptive: Least Left First (LLF), Round Robin, Priority, Lottery

Summary

- Scheduling Algorithms are either Pre-emptive or non-preemptive
- Each favor a particular set of criteria:
  - CPU Utilization, Wait Time, Responsiveness, etc
  - Implementation Concerns: **Speed**
- Evaluate through simulation and observation

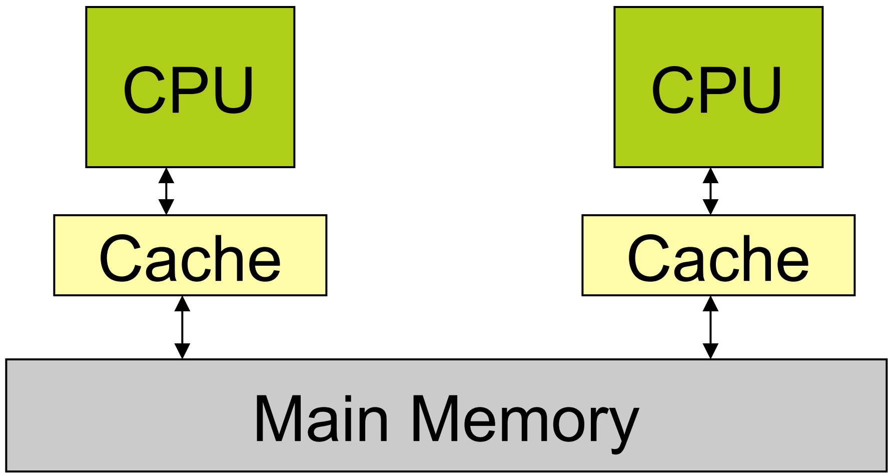
Multiprocessor Scheduling

- Goal: Load Sharing
- Assumptions: Homogenous set of CPU's
  - Note - CPU's may have dedicated bus to specific I/O (heterogeneous)

Asymmetric Multiprocessing ← Easy, but wasteful

Most efficient → SMP: Symmetric Multiprocessing

Processor Affinity



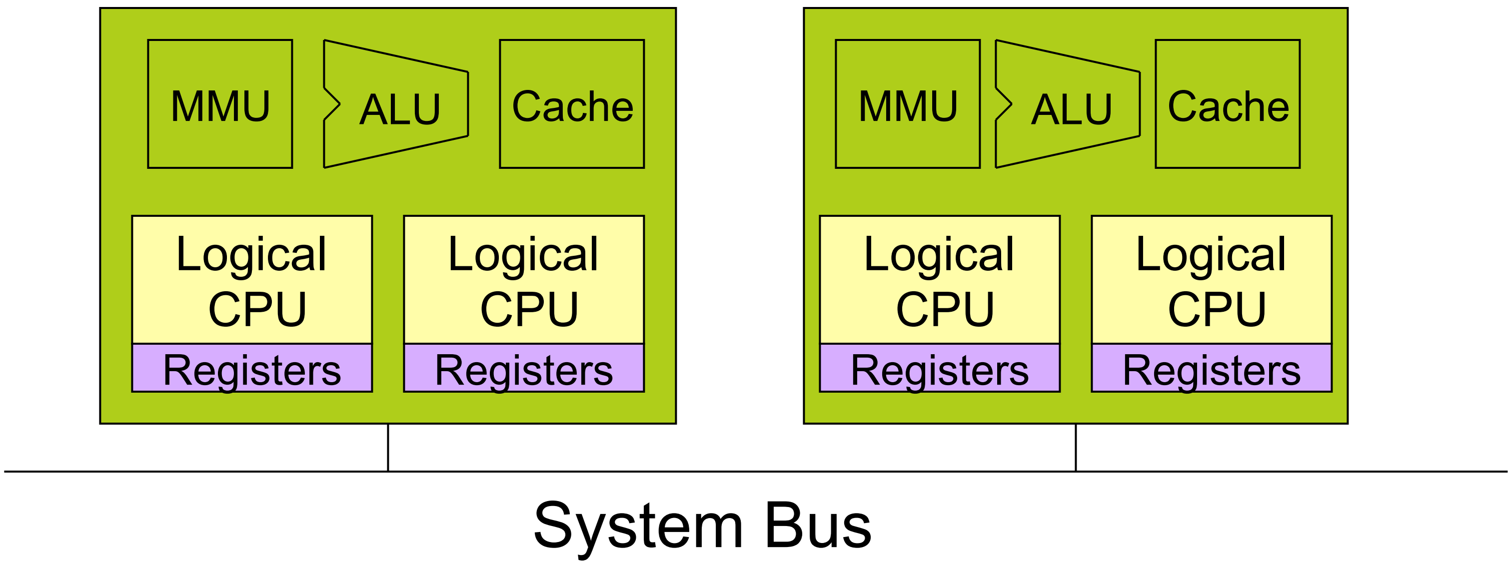
- *Process Migration*: Time Consuming
- *Processor Affinity*: Attempt to keep a process on the same CPU throughout its lifetime

Load Balancing

- When each CPU has separate ready queues, must prevent one from being overloaded
  - Push Migration
  - Pull Migration

  - How does this affect Processor Affinity?
  - Is this necessary when a common ready queue is used?

Symmetric Multi-**threading**



- Multiple CPU's are *emulated* on one chip.
- OS may/may not be aware.
  - Performance improved if made aware

Scheduling: Sun Solaris

- Kernel Threads
- Priority Based Scheduling:
  - Real-time, System, Time Sharing, Interactive
  - Within each priority, different scheduling algorithm
- Time Sharing and Interactive have dynamically changing priorities
  - Varies between 0 - 60 (low to high)

Changing Priorities

			Priority after...	
	Priority	Quantum Time	Quantum Expiration	Sleep/IO Completion
Lowest	0	200	0	50
	10	160	0	51
	20	120	15	52
	35	80	25	54
	50	40	40	58
Highest	59	20	49	59

What type of process frequently completes its quantum?

Scheduling: Windows

- Kernel Threads
  - Preemptive - Priority Scheduling
  - Highest priority thread **always** runs
  - Context Switch when…
    - Higher Priority thread is ready
    - Thread terminates
    - time quantum expires
    - blocking call issued

Priority Levels

- 32 Level priority
  - 2 Classes: variable and real-time
  - Variable: 1-15
  - Real-time: 16-31
  - Memory management: 0

Windows Priority Classes

Priorities

Relative Priorities

	real-time	high	above normal	normal	below normal	idle
time-critical	31	15	15	15	15	15
highest	26	15	12	10	8	6
above normal	25	14	11	9	7	5
normal	24	13	10	8	6	4
below normal	23	12	9	7	5	3
lowest	22	11	8	6	4	2
idle	16	1	1	1	1	1

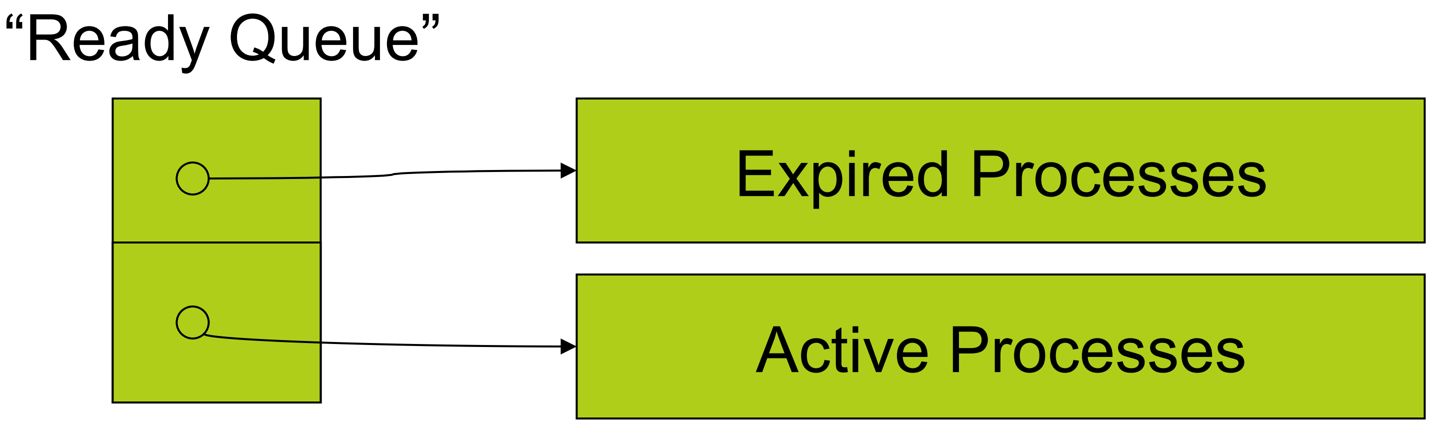
Priority lowered after preemption

Priority elevated after I/O completion

Linux Scheduling

- Pre-emptive, Priority Based
  - Classes:
    - Real-time (0-99)
    - Nice (100-140)
  - Higher Priority gets **longer** quantum
    - Low priority - 10ms
    - High priority - 200ms
  - Notion of “expired” versus “active”

Split Ready Queue



- Only Active Processes will run
- When a process uses its quantum, marked expired
- When active queue empties, pointers are swapped and process repeats