CS370 Operating Systems

Colorado State University
Yashwant K Malaiya

Fall 2016 Lecture 13

Slides based on

- Text by Silberschatz, Galvin, Gagne
- Various sources

CPU Scheduling: Objectives

- CPU-scheduling algorithms
- Evaluation criteria

Questions from last time

- Shortest Job first
  - Starvation?
- Preemptive vs non-preemptive? Context switch forced often because the time slice expires.
- Exponential averaging: how do we know the actual length of the burst?
- Can we have a discussion board on canvas for discussions outside the classroom?
- Why do we care about the waiting time? It seems it will be difficult to rearrange the queue.

Questions

- Wait vs Ready state waiting for IO event
- Advantage of multithreaded programs
- What is CPU “burst” Program running continuously on CPU without an I/O
- Preemptive scheduling: why it helps, even though it has overhead?
  - Fairness
  - May reduce waiting time
  - Context switch often <1%

Notes

- PA2 due 9/29, Help session this Thurs 5-6, CSB 325
- HW1 due 10/4, no late period
- Midterm Oct 7 (see announcement or grading)
- Project: topics, teams, details after midterm
  - Poster session Dec 9 10AM-Noon

FCFS Scheduling (Cont.)

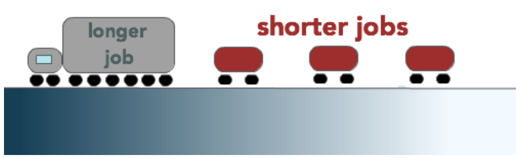
Suppose that the processes arrive in the order:

$$P_2 , P_3 , P_1$$

- The Gantt chart for the schedule is:

- Waiting time for $P_1 = 6; P_2 = 0; P_3 = 3$
- Average waiting time: $(6 + 0 + 3)/3 = 3$
- Much better than previous case
- But note -Throughput: 3/30 = 0.1 per unit
- **Convoy effect** - short process behind long process
  - Consider one CPU-bound and many I/O-bound processes

The Convoy Effect, visualized



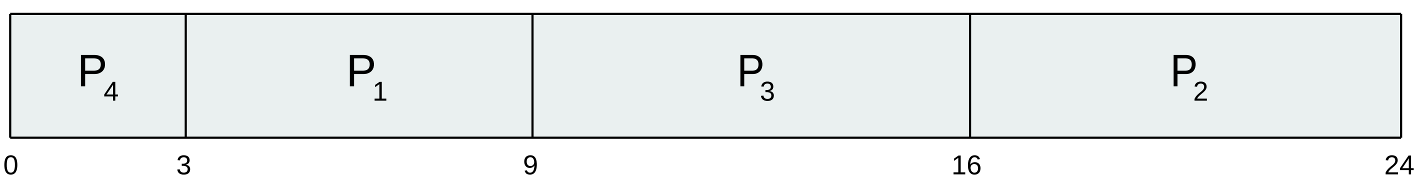
Shortest-Job-First (SJF) Scheduling

- Associate with each process the length of its next CPU burst
  - Use these lengths to schedule the process with the shortest time
- Reduction in waiting time for short process *GREATER THAN* Increase in waiting time for long process
- SJF is optimal – gives **minimum average waiting time** for a given set of processes
  - The difficulty is knowing the length of the next CPU request
  - Could ask the user

Example of SJF

Process	Burst Time
P_1	6
P_2	8
P_3	7
P_4	3

- All arrive at time 0.
- SJF scheduling chart



- Average waiting time for P_1,P_2,P_3,P_4 = (3 + 16 + 9 + 0) / 4 = 7

Determining Length of Next CPU Burst

- Can only estimate the length – should be similar to the recent bursts
  – Then pick process with shortest predicted next CPU burst

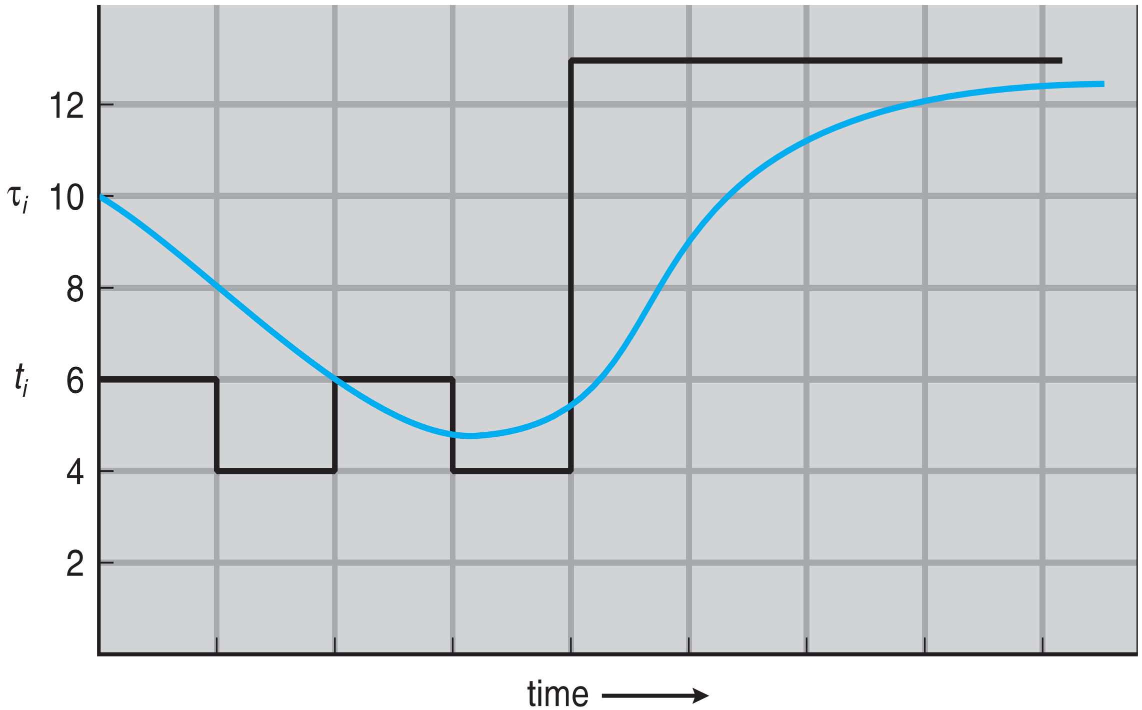
- Can be done by using the length of previous CPU bursts, using *exponential averaging*

$$\begin{aligned}
&1.\ t_n = \text{actual length of } n^{th} \text{ CPU burst} \\
&2.\ \tau_{n+1} = \text{predicted value for the next CPU burst} \\
&3.\ \alpha, 0 \le \alpha \le 1 \\
&4.\ \text{Define: } \tau_{n+1} = \alpha\, t_n + (1-\alpha)\tau_n .
\end{aligned}$$

- Commonly, α set to ½
- Preemptive version called **shortest-remaining-time-first**

Colorado State University

Prediction of the Length of the Next CPU Burst



Blue points: guess
Black points: actual
$\alpha = 0.5$

Ex:
$0.5 \times 6 + 0.5 \times 10 = 8$

CPU burst (t_i)		6	4	6	4	13	13	13	…
"guess" (τ_i)	10	8	6	6	5	9	11	12	…

Examples of Exponential Averaging

- α =0
  - $\tau_{n+1} = \tau_n$
  - Recent history does not count
- α =1
  - $\tau_{n+1} = \alpha\, t_n$
  - Only the actual last CPU burst counts
- $\tau_{n+1} = \alpha\, t_n + (1 - \alpha)\tau_n.$
- If we expand the formula, substituting for τ_n, we get:

$$\tau_{n+1} = \alpha\, t_n + (1 - \alpha)\alpha\, t_{n-1} + \ldots$$
$$+ (1 - \alpha)^j \alpha\, t_{n-j} + \ldots$$
$$+ (1 - \alpha)^{n+1} \tau_0$$

- Since both α and (1 - α) are less than or equal to 1, each successive term has less weight than its predecessor

Widely used for predicting stock-market etc

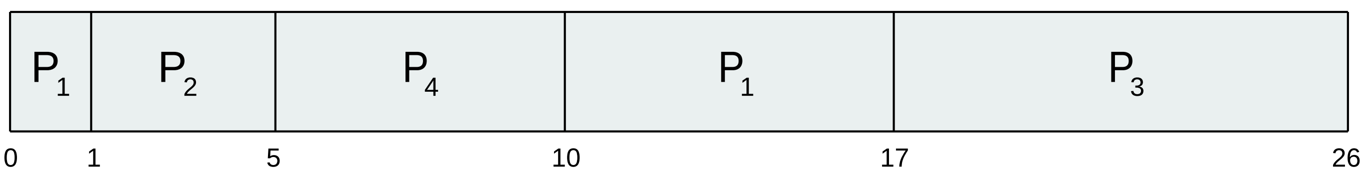
Colorado State University

Shortest-remaining-time-first (**preemptive** SJF)

- Now we add the concepts of varying arrival times and preemption to the analysis

Process	*Arrival* Time	Burst Time
P_1	0	8
P_2	1	4
P_3	2	9
P_4	3	5

- *Preemptive* SJF Gantt Chart



- Average waiting time for P1,P2,P3,P4

  = [(10-1)+(1-1)+(17-2)+(5-3)]/4 = 26/4 = 6.5 msec

Colorado State University

Priority Scheduling

- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer ≡ highest priority)
  - Preemptive
  - Nonpreemptive

- SJF is priority scheduling where priority is the inverse of predicted next CPU burst time
- Problem ≡ **Starvation** – low priority processes may never execute
  - Solution ≡ **Aging** – as time progresses increase the priority of the process

MIT had a low priority job waiting from 1967 to 1973 on IBM 7094! ☺

Colorado State University

Example of Priority Scheduling

Process	Burst Time	Priority
P_1	10	3
P_2	1	1 (highest)
P_3	2	4
P_4	1	5
P_5	5	2

- Arrived at time 0 in order P1,P2, P3, P4,P5
- Priority scheduling Gantt Chart

- Average waiting time for P1, .. P5: (6+0+16+18+1)/5 = 8.2 msec

Round Robin (RR) with time quantum

- Each process gets a small unit of CPU time (**time quantum** q), usually 1-10 milliseconds. After this, the process is preempted, added to the end of the ready queue.
- If there are n processes in the ready queue and the time quantum is q, then each process gets $1/n$ of the CPU time in chunks of at most q time units at once. No process waits more than $(n-1)q$ time units.
- Timer interrupts every quantum to schedule next process
- Performance
  - q large $\Rightarrow$ FIFO
  - q small $\Rightarrow$ q must be large with respect to context switch, otherwise overhead is too high (overhead *typically in 0.5% range*)

Example of RR with Time Quantum = 4

Process	Burst Time
P_1	24
P_2	3
P_3	3

- Arrive a time 0 in order P1, P2, P3: The Gantt chart is:

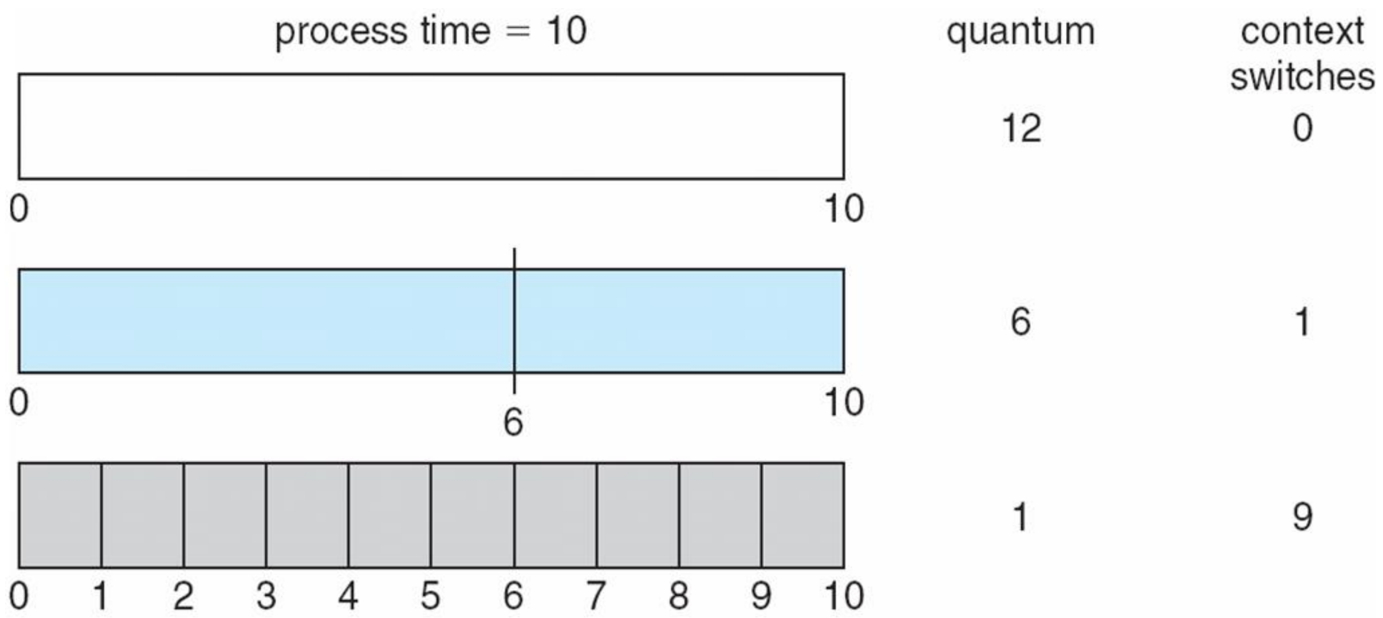
P_1	P_2	P_3	P_1	P_1	P_1	P_1	P_1

0 4 7 10 14 18 22 26 30

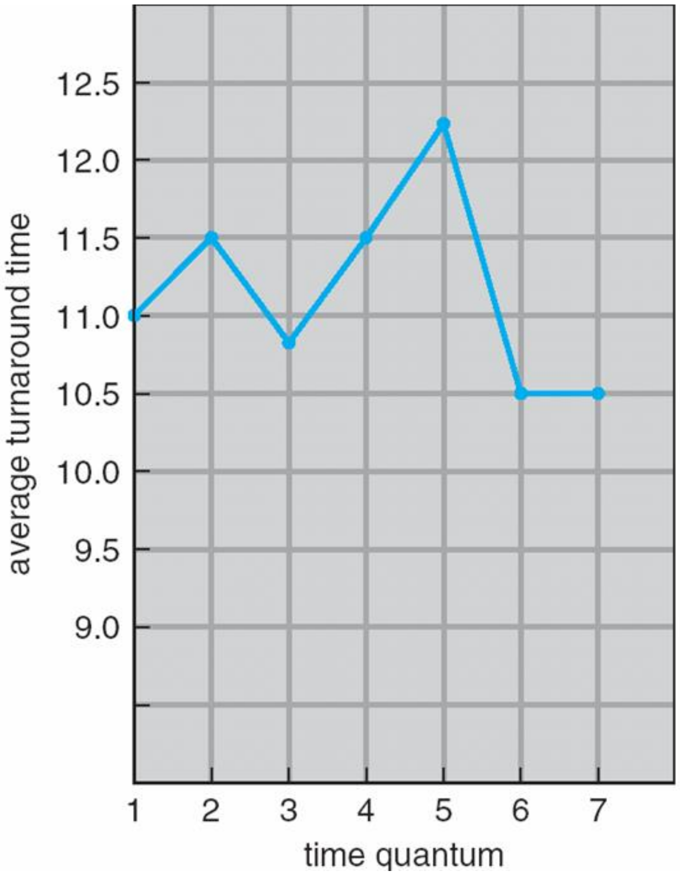
- Waiting times: 10-4 =6, 4, 7, average 17/3 = 5.66 units
- Typically, higher average turnaround than SJF, but better ***response***
- q should be large compared to context switch time
- q usually 10ms to 100ms, context switch < 10 µsec

Response time: Arrival to beginning of execution
Turnaround time: Arrival to finish of execution

Time Quantum and Context Switch Time



Turnaround Time Varies With The Time Quantum



process	time
P_1	6
P_2	3
P_3	1
P_4	7

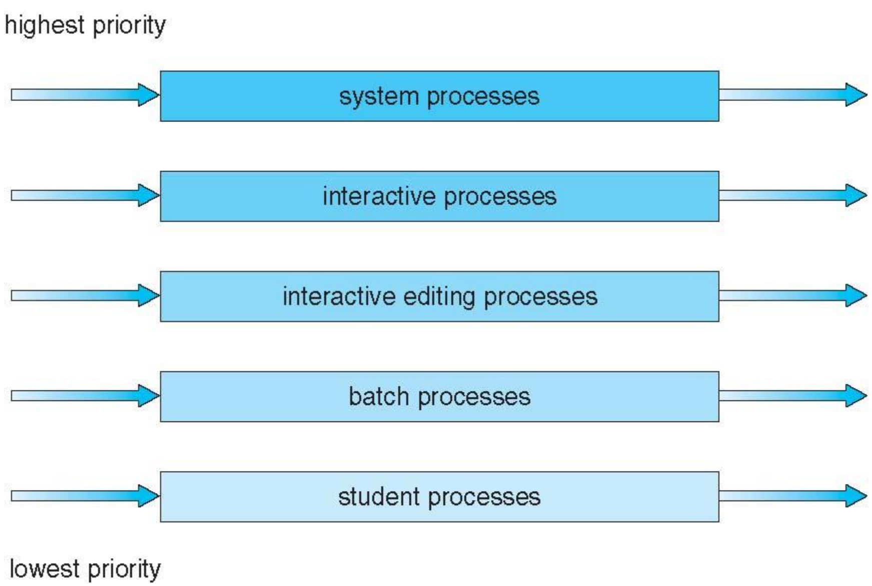
Rule of thumb: 80% of CPU bursts should be shorter than q

Illustration
q=7:
Turnaround time:
6,9,10,17 av = 10.5
Similarly for q =1, ..6

Multilevel Queue

- Ready queue is partitioned into separate queues, e.g.:
  - **foreground** (interactive)
  - **background** (batch)
- Process permanently in a given queue
- Each queue has its own scheduling algorithm, e.g.:
  - foreground – RR
  - background – FCFS
- Scheduling must be done between the queues:
  - Fixed priority scheduling; (i.e., serve all from foreground then from background). Possibility of starvation. Or
  - Time slice – each queue gets a certain amount of CPU time which it can schedule amongst its processes; i.e., 80% to foreground in RR, 20% to background in FCFS

Multilevel Queue Scheduling

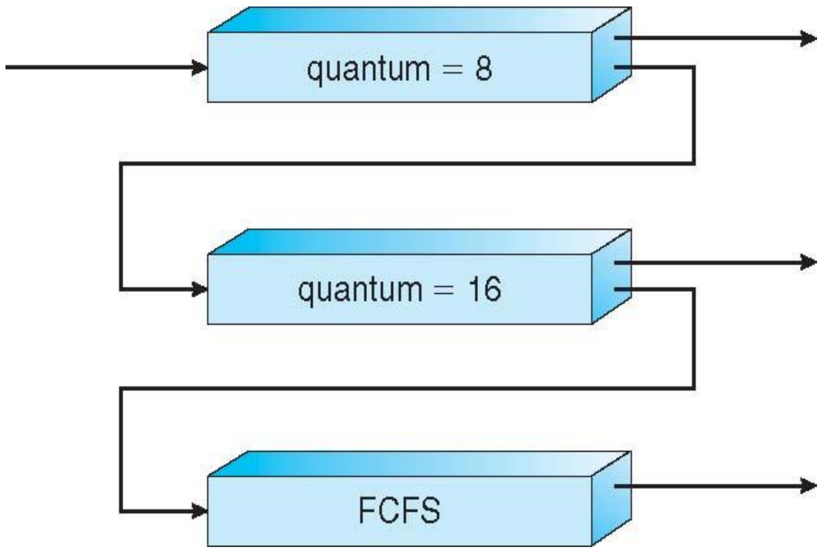


Multilevel Feedback Queue

- A process can move between the various queues; aging can be implemented this way
- Multilevel-feedback-queue scheduler defined by the following parameters:
  - number of queues
  - scheduling algorithms for each queue
  - method used to determine when to upgrade a process
  - method used to determine when to demote a process
  - method used to determine which queue a process will enter when that process needs service

Example of Multilevel Feedback Queue

- Three queues:
  - Q_0 – RR with time quantum 8 milliseconds
  - Q_1 – RR time quantum 16 milliseconds
  - Q_2 – FCFS (no time quantum limit)



- Scheduling
  - A new job enters queue Q_0 which is served FCFS
    - When it gains CPU, job receives 8 milliseconds
    - If it does not finish in 8 milliseconds, job is moved to queue Q_1
  - At Q_1 job is again served FCFS and receives 16 additional milliseconds
    - If it still does not complete, it is preempted and moved to queue Q_2

CPU-bound: priority falls, quantum raised,
I/O-bound: priority rises, quantum lowered

Colorado State University

Thread Scheduling

- Thread scheduling is similar
- Distinction between user-level and kernel-level threads
- When threads supported, threads scheduled, not processes

Scheduling competition

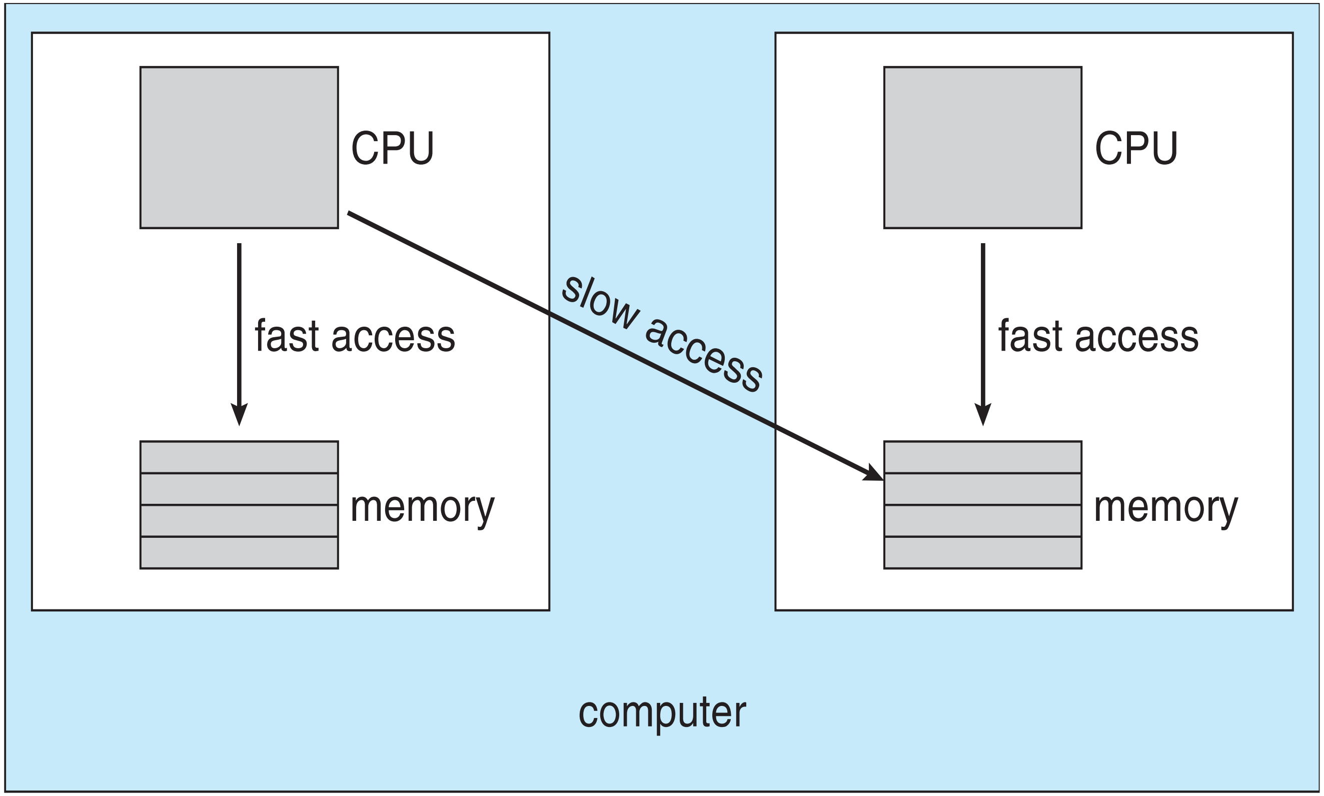
- Many-to-one and many-to-many models, thread library schedules user-level threads to run on LWP
  - Known as **process-contention scope** (**PCS**) since scheduling competition is within the process
  - Typically done via priority set by programmer
- Kernel thread scheduled onto available CPU is **system-contention scope** (**SCS**) – competition among all threads in system

LWP layer between kernel threads and user threads in some older OSs

Multiple-Processor Scheduling

- CPU scheduling more complex when multiple CPUs are available.
- **Assume Homogeneous processors** within a multiprocessor
- **Asymmetric multiprocessing** – individual processors can be dedicated to specific tasks at design time
- **Symmetric multiprocessing (SMP)** – each processor is self-scheduling,
  - all processes in common ready queue, **or**
  - each has its own private queue of ready processes
    - Currently, most common
- **Processor affinity** – process has affinity for processor on which it is currently running because of info in cache
  - **soft affinity: try but no guarantee**
  - **hard affinity can specify processor sets**

NUMA and CPU Scheduling



Note that memory-placement algorithms can also consider affinity
Non-uniform memory access (NUMA), in which a CPU has faster access to some parts of main memory.