Scheduling

- Mostly Siberschatz 5.1-5.6
  - (Lewis & Berg Ch. 5 too)
- Scheduling determines who runs next/when
- Dispatcher puts that decision in action
  - switches contexts, switches to user mode, jumps to PC and gives CPU to selected process

- Scheduling decisions occur
  - process switches from running to waiting state
  - process switches from running to ready (something happened, e.g., interrupt, to cause it to stop running)
  - process switches from waiting to ready (I/O completes)
  - process terminates
- May be preemptive or non-preemptive

Scheduling criteria

- To guarantee min/max value, good average, low variance of values such as:
  - CPU utilization (max)
  - Throughput (max)
  - Turnaround time (min)
  - Waiting time (min)
  - Response time (min)
  - other…
- Different algorithms may have different impact on these
- Very workload dependent

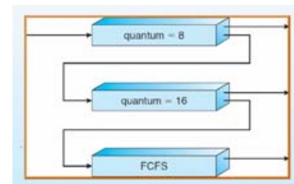
Scheduling algorithms

- First Come First Serve
  - non preemptive, can be bad for average waiting time
- Shortest Job First
  - preemptive, heuristic in job duration prediction (e.g., based on prior history…)
- Priority Scheduling
  - introduce priority aging to avoid starvation
- Round Robin
  - like FCFS with preemption, quantum must be large enough wrt context switch time, otherwise overhead too high

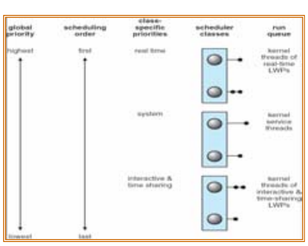
- Multilevel Queue
  - multiple queues, for different classes of processes (or different priorities)
  - within each queue, different scheduling algorithm may be applied
  - some objectives – interactive processes should get high priority, compute-bound process may use longer time quanta but non necessarily high priority
  - how does the scheduler know which process is what?

Multilevel Feedback-Queue Scheduling

- 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 a service



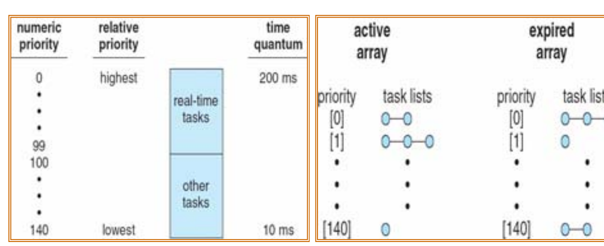
Example: Solaris



priority	time quantum	time quantum expired	return from sleep
0	200	0	50
5	200	0	50
10	160	0	51
15	160	5	51
20	120	10	52
25	120	15	52
30	80	20	53
35	80	25	54
40	40	30	55
45	40	35	56
50	40	40	58
55	40	45	58
59	20	49	59

0 = low priority; 59 = high priority
high priority task gets smaller quantum

Example: Linux



higher priority = longer time quantum
for non-RT tasks adjustment criteria sleep time:
long sleeps => interactive => priority – 5
short sleeps => compute intensive => priority + 5

Scheduling on Multi-processor Systems

- assumption homogeneous
  - in heterogeneous systems, availability of certain resources is a factor (e.g., device, floating point unit, different ISA…)
- objective well balanced systems
  - trade-off between load-balancing and processor-affinity

Scheduling in CMTs

- paper "Chip Multithreading Systems Need a New Operating System Scheduler"
  - emerging platforms today, and all platforms in future will be CMTs (at least Intel hopes so)
- *Question:* If we have n-way system where each core has m hardware threads, can we deliver performance = m*n individual cores running in parallel?
  - no but can we get close?
  - objective in paper – design scheduling to maximize performance, by maximizing utilization of CPU resources

Summary of paper ideas

- improve utilization of processor pipeline (shared resource) in CMT systems
  - idea: consider which units are needed when scheduling next thread, make sure as many of them can be doing work
  - need some easy metric to be able to differentiate between memory- and compute-intensive threads
  - cycles per instruction (CPI) is chosen to be that metric
- Results good for synthetic, but modest for real workloads
  - probably indicates that no reason to put circuitry in hardware which will maintain accurate CPI for the sake of scheduling
- Important for you -> factors which are to be considered
- Next iteration of this work looks at L2 cache utilization, and tries to co-schedule threads which are going to be able to share as much of the cache as possible