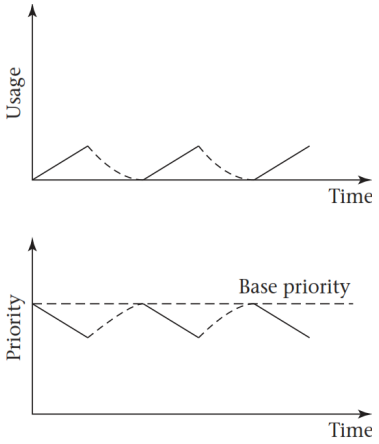
1) *Decay usage schedulers* (e.g., Mac OS X / MS Windows) dynamically adjust priorities of waiting threads higher, and running threads lower.

	Mac OS X
Thread priority:	"base priority" - recent usage
When adjust priorities?	when a thread changes state
Adjust running thread usage by:	old usage + current burst * current load
Adjust waiting thread usage by:	old usage * $(5/8)^n$, where n is # of 1/8 sec. "*exponential decay*"



a) Why are priorities updated when a thread changes state and not continually?

b) The values (5/8) and 1/8 sec. are empirical parameter defaults that can be tuned by the system administrator.

- What would be the general effect of increasing from (5/8) to (7/8)?

- What would be the general effect of decreasing from 1/8 second to 1/16 second?

c) In Microsoft Windows, a thread coming out of a wait state has its priority elevated from the base priority depending on what it was waiting. Why would waiting on a disk be a small boost, but waiting on input from the keyboard be a large boost?

d) In Microsoft Windows, a thread using the processor has its priority reduced linearly downward toward the base priority. To avoid starving a runnable thread with base priority, it periodically boosts the priority of any thread not dispatched otherwise. Why does the Mac OS X scheduler not have a similar starvation problem?

2. *Proportional-Share scheduling* - allow the user (or system) to specify the proportion of CPU time each thread should be allocated. Three main types of proportional-share scheduling:

a) weighted round-robin scheduling (WRR) - round-robin so threads with higher weights getting longer-time slices

Consider three threads we want to run with the following proportions. If we want to run all three in 32 ms, how big of time slices would each be allocated?

Thread	Proportion of CPU	CPU Time-Slice (ms)
T1	4	
T2	3	
T3	1	

Draw the Gantt chart for this schedule

b) *weighted fair queuing* (WFQ)/*stride scheduling*/*virtual time round-robin scheduling* (VTRR) - uniform time-slice, but smaller allocation threads sit out some round-robin rotations through the runnable-thread list.

Consider three threads we want to run with the following proportions. If we want to run all three in 32 ms with a 4 ms uniform time-slice, what percentage of the round-robin rotations would a thread participate?

Thread	Proportion of CPU	Percentage participation in round-robin rotations
T1	4	
T2	3	
T3	1	

Draw the Gantt chart for this schedule

c) *lottery scheduling* - uniform time slice, but large allocation threads win lottery more often

Consider three threads we want to run with the following proportions.

Thread	Proportion of CPU	Range of lottery tickets for thread
T1	4	
T2	3	
T3	1	

If there is a 4 ms uniform time-slice and the random numbers used for scheduling are 3, 5, 1, 4, 5, 8, 2, 7, 8, 1, 6, then draw the Gantt chart for this schedule.

3. Lottery scheduling is rarely used on real systems because a thread might have bad luck and not "win" the scheduling lottery for a long time (e.g., a second). Which performance criteria (throughput or response time) is most effected by a lottery scheduler?

4. "Recent" versions of Linux use the *Completely Fair Scheduler* (*CFS*) to schedule on a processor (with an independent load-balancing mechanism to distribute threads across processors). Characteristics of CFS are:

- each thread has a niceness level (-20 [highest priority] to 19 [lowest priority]). Each niceness level has a weight associated with it (e.g., niceness 0 →1024, 5 →335, etc. via some geometric progression).
- the scheduler round-robins through the threads trying to complete a "pass" in a target time (e.g. 6 ms default on uniprocessor)
- each thread is given a time slice proportional to its weight/(total weights of all runnable threads) (a minimum time slice default is used if proportional time slice is less)

a) Which type of proportional-share scheduling is CFS?

b) Which performance criteria (throughput or response time) is most effected by adjusting the "target time" for a round-robin pass?

c) Which performance criteria (throughput or response time) is most effected by adjusting the "minimum time slice"?