Linux Scheduling

- The three classes are:
  - SCHED_FIFO: First-in-first-out real-time threads
  - SCHED_RR: Round-robin real-time threads
  - SCHED_OTHER: Other, non-real-time threads
- Within each class multiple priorities may be used

A	minimum
B	middle
C	middle
D	maximum

(a) Relative thread priorities

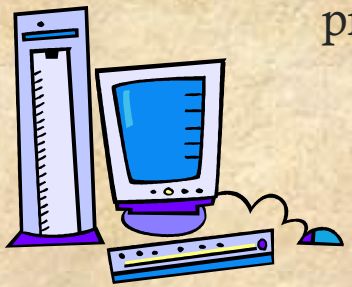
(b) Flow with FIFO scheduling

(c) Flow with RR scheduling

Figure 10.10 Example of Linux Real-Time Scheduling

Non-Real-Time Scheduling

- The Linux 2.4 scheduler for the SCHED_OTHER class did not scale well with increasing number of processors and processes
- Time to select the appropriate process and assign it to a processor is constant regardless of the load on the system or number of processors
- Linux 2.6 uses a new priority scheduler known as the O(1) scheduler
- Kernel maintains two scheduling data structures for each processor in the system



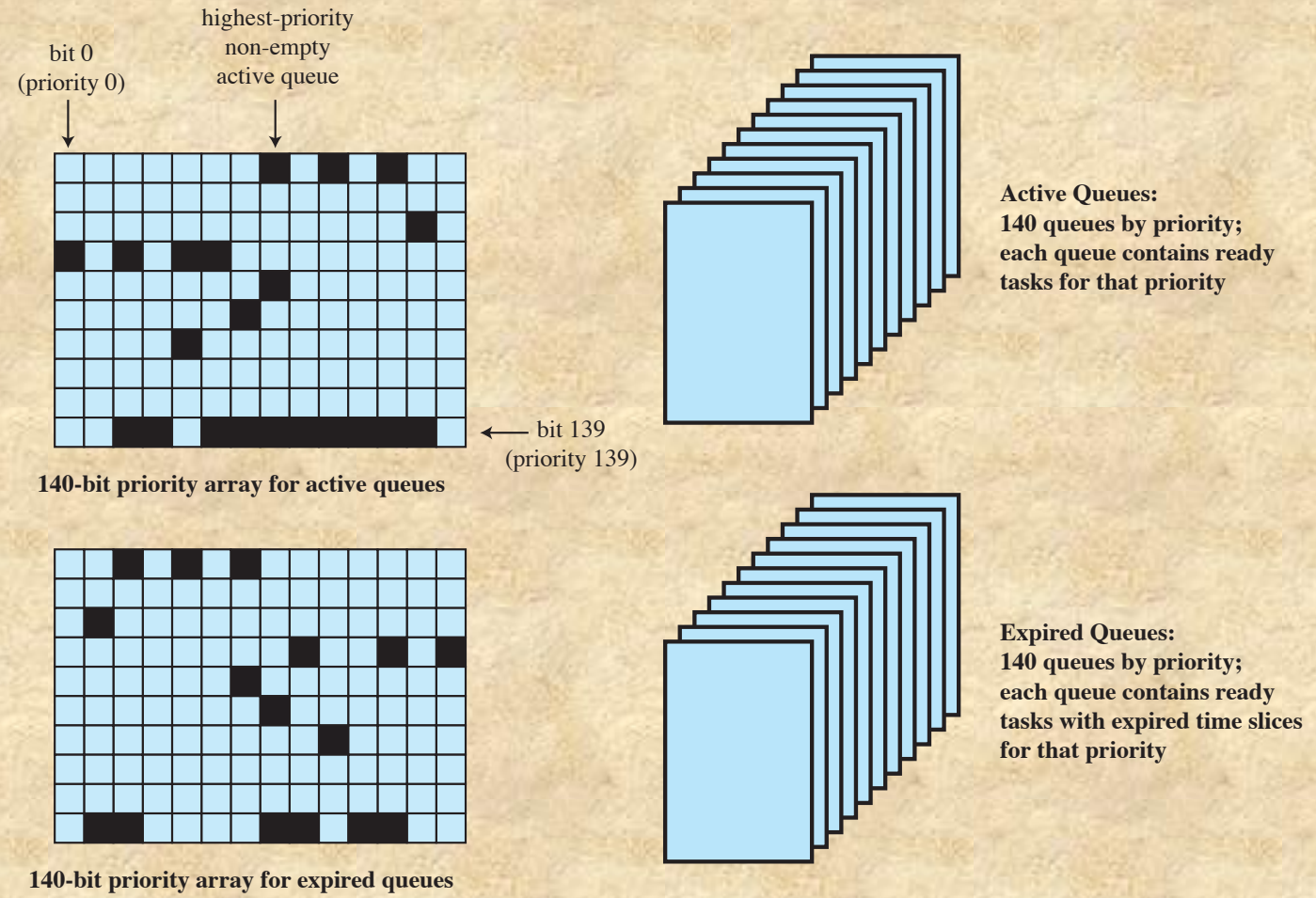


Figure 10.11 Linux Scheduling Data Structures for Each Processor

UNIX SVR4 Scheduling

- A complete overhaul of the scheduling algorithm used in earlier UNIX systems

The new algorithm is designed to give:

- highest preference to real-time processes
- next-highest preference to kernel-mode processes
- lowest preference to other user-mode processes

Major modifications:

- addition of a preemptable static priority scheduler and the introduction of a set of 160 priority levels divided into three priority classes
- insertion of preemption points

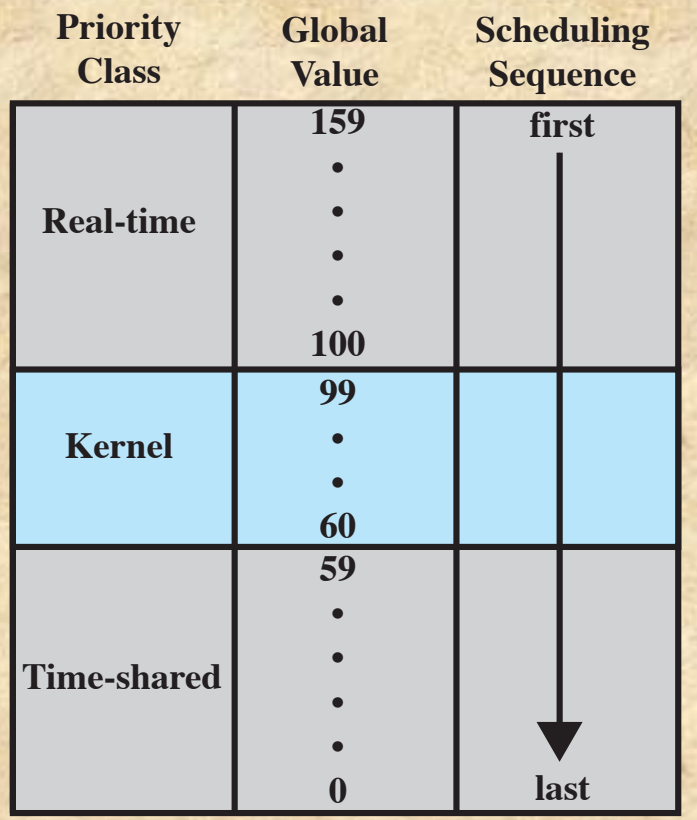


Figure 10.12 SVR4 Priority Classes

SVR Priority Classes

Real time (159 – 100)

- guaranteed to be selected to run before any kernel or time-sharing process
- can preempt kernel and user processes

Kernel (99 – 60)

- guaranteed to be selected to run before any time-sharing process, but must defer to real-time processes

Time-shared (59-0)

- lowest-priority processes, intended for user applications other than real-time applications

Table 10.6
FreeBSD Thread Scheduling Classes

Priority Class	Thread Type	Description
0 - 63	Bottom-half kernel	Scheduled by interrupts. Can block to await a resource.
64 - 127	Top-half kernel	Runs until blocked or done. Can block to await a resource.
128 - 159	Real-time user	Allowed to run until blocked or until a higher priority thread becomes available. Preemptive scheduling.
160 - 223	Time-sharing user	Adjusts priorities based on processor usage.
224 - 255	Idle user	Only run when there are no time sharing or real-time threads to run.

Note: Lower number corresponds to higher priority

SMP and Multicore Support

- FreeBSD scheduler was designed to provide effective scheduling for a SMP or multicore system
- Design goals:
  - address the need for processor affinity in SMP and multicore systems
    - *processor affinity* – a scheduler that only migrates a thread when necessary to avoid having an idle processor
  - provide better support for multithreading on multicore systems
  - improve the performance of the scheduling algorithm so that it is no longer a function of the number of threads in the system

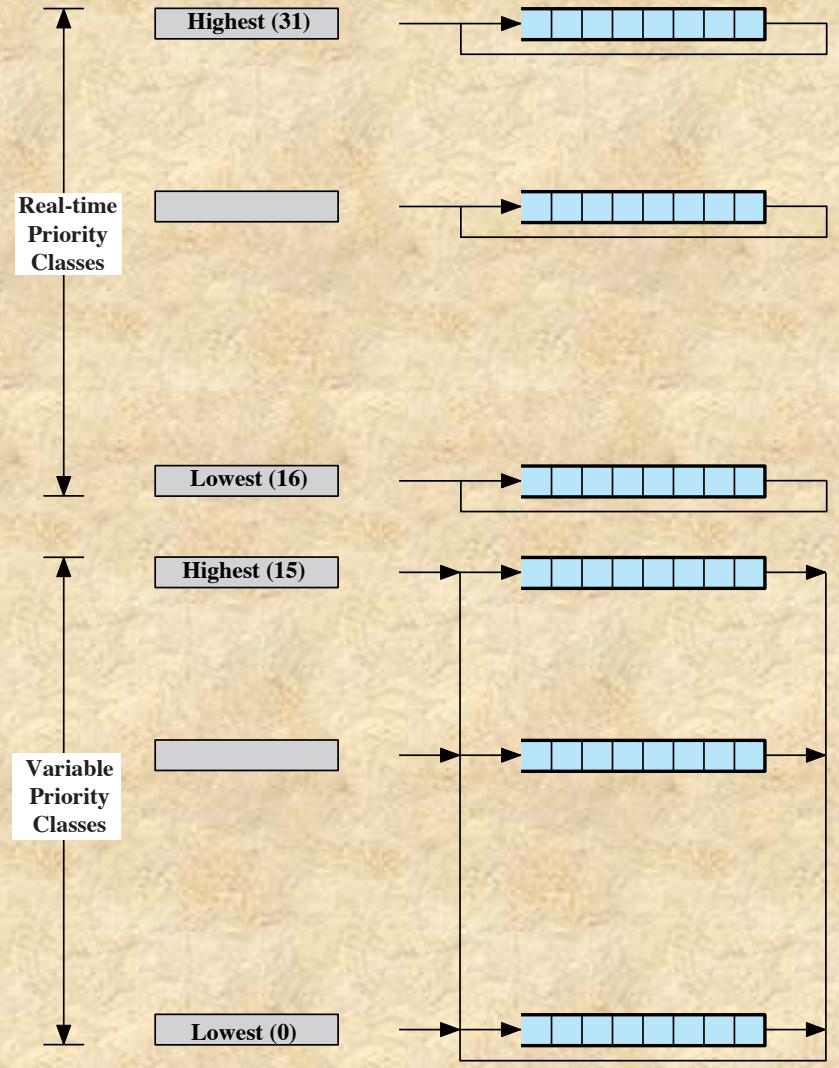


Figure 10.14 Windows Thread Dispatching Priorities

Interactivity Scoring

- A thread is considered to be *interactive* if the ratio of its voluntary sleep time versus its runtime is below a certain threshold
- Interactivity threshold is defined in the scheduler code and is not configurable
- Threads whose sleep time exceeds their run time score in the lower half of the range of interactivity scores
- Threads whose run time exceeds their sleep time score in the upper half of the range of interactivity scores

Windows Scheduling

- Priorities in Windows are organized into two bands or classes:

real time priority class

- all threads have a fixed priority that never changes
- all of the active threads at a given priority level are in a round-robin queue

- Each band consists of 16 priority levels
- Threads requiring immediate attention are in the real-time class
  - include functions such as communications and real-time tasks

variable priority class

- a thread's priority begins an initial priority value and then may be temporarily boosted during the thread's lifetime

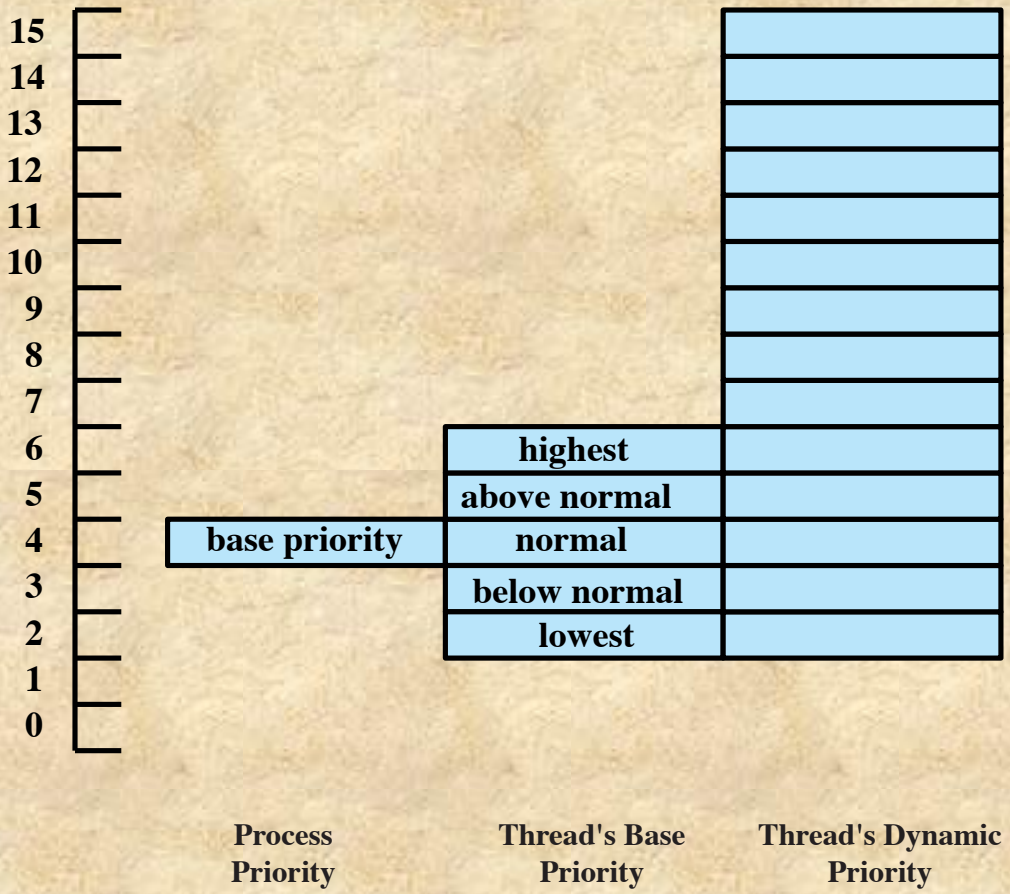


Figure 10.15 Example of Windows Priority Relationship