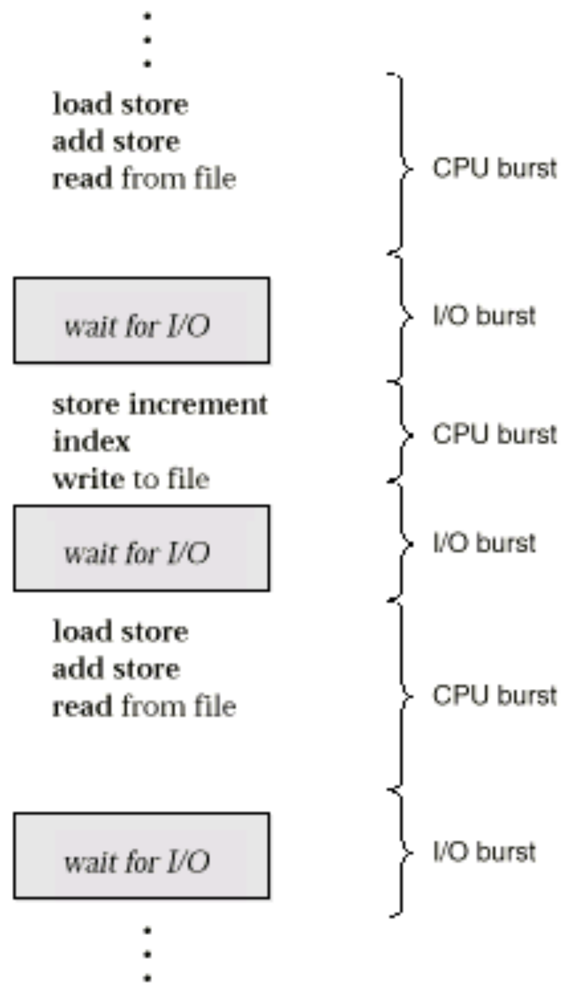
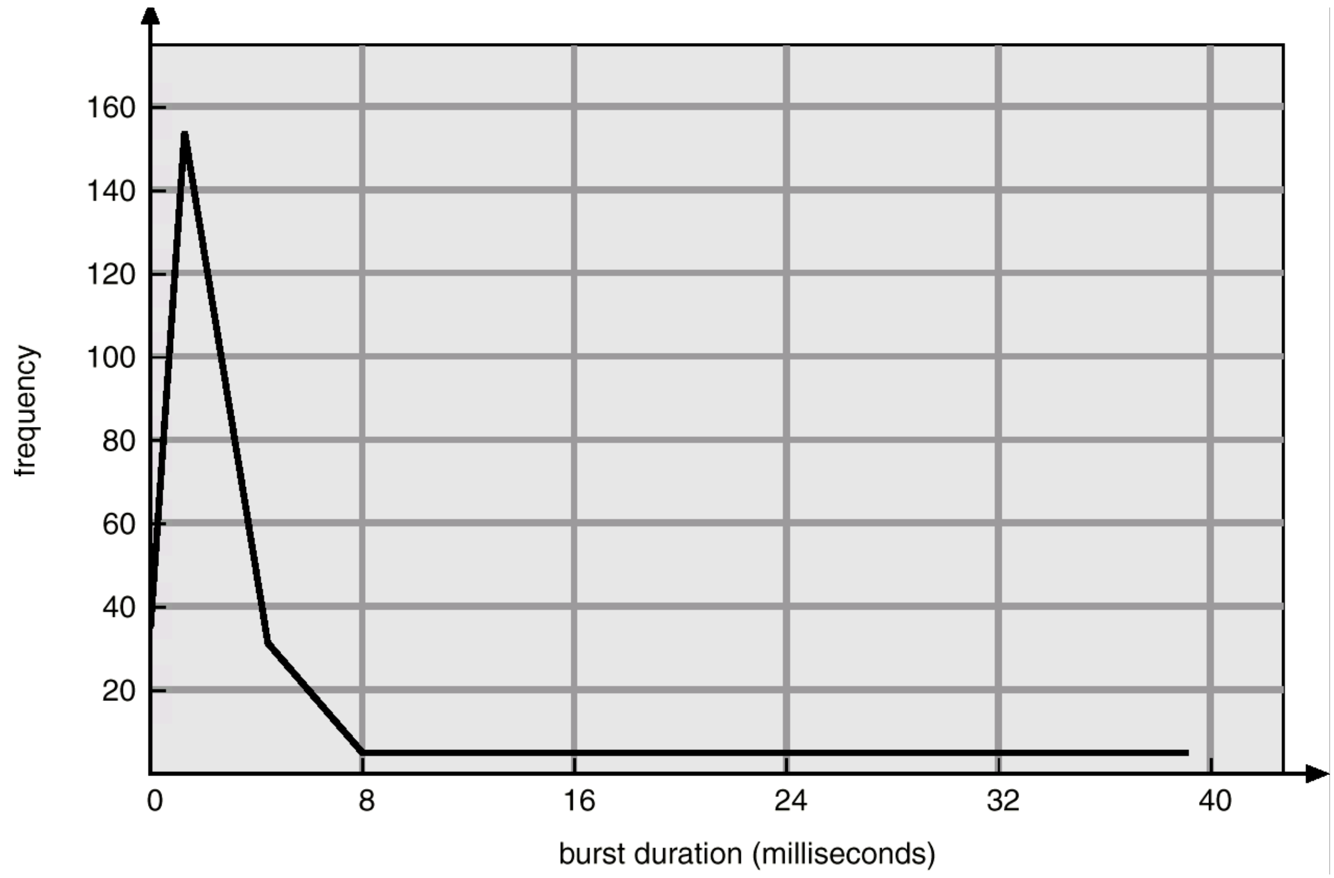
CPU Scheduling

Process Execution: Alternating Sequence of CPU And I/O Bursts



Histogram of CPU-burst Times



CPU Scheduler

- Selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them
- CPU scheduling decisions may take place when a process:
  1. Switches from running to waiting state
  2. Terminates
  3. Switches from running to ready state
  4. Switches from waiting to ready
- Scheduling under 1,2 is *nonpreemptive*
- Scheduling under 3,4 is *preemptive*

Dispatcher

- Dispatcher module gives control of the CPU to the process selected by the short-term scheduler; this involves:
  - switching context
  - switching to user mode
  - jumping to the proper location in the user program to restart that program
- *Dispatch latency* – time it takes for the dispatcher to stop one process and start another running

Scheduling Criteria

- CPU utilization – keep the CPU as busy as possible
- Throughput – # of processes that complete their execution per time unit
- Turnaround time – amount of time to execute a particular process
- Waiting time – amount of time a process has been waiting in the ready queue
- Response time – amount of time it takes from when a request was submitted until the first response is produced, **not** output (for time-sharing environment)

Optimization Criteria

- Max CPU utilization
- Max throughput
- Min turnaround time
- Min waiting time
- Min response time

Conflicting criteria

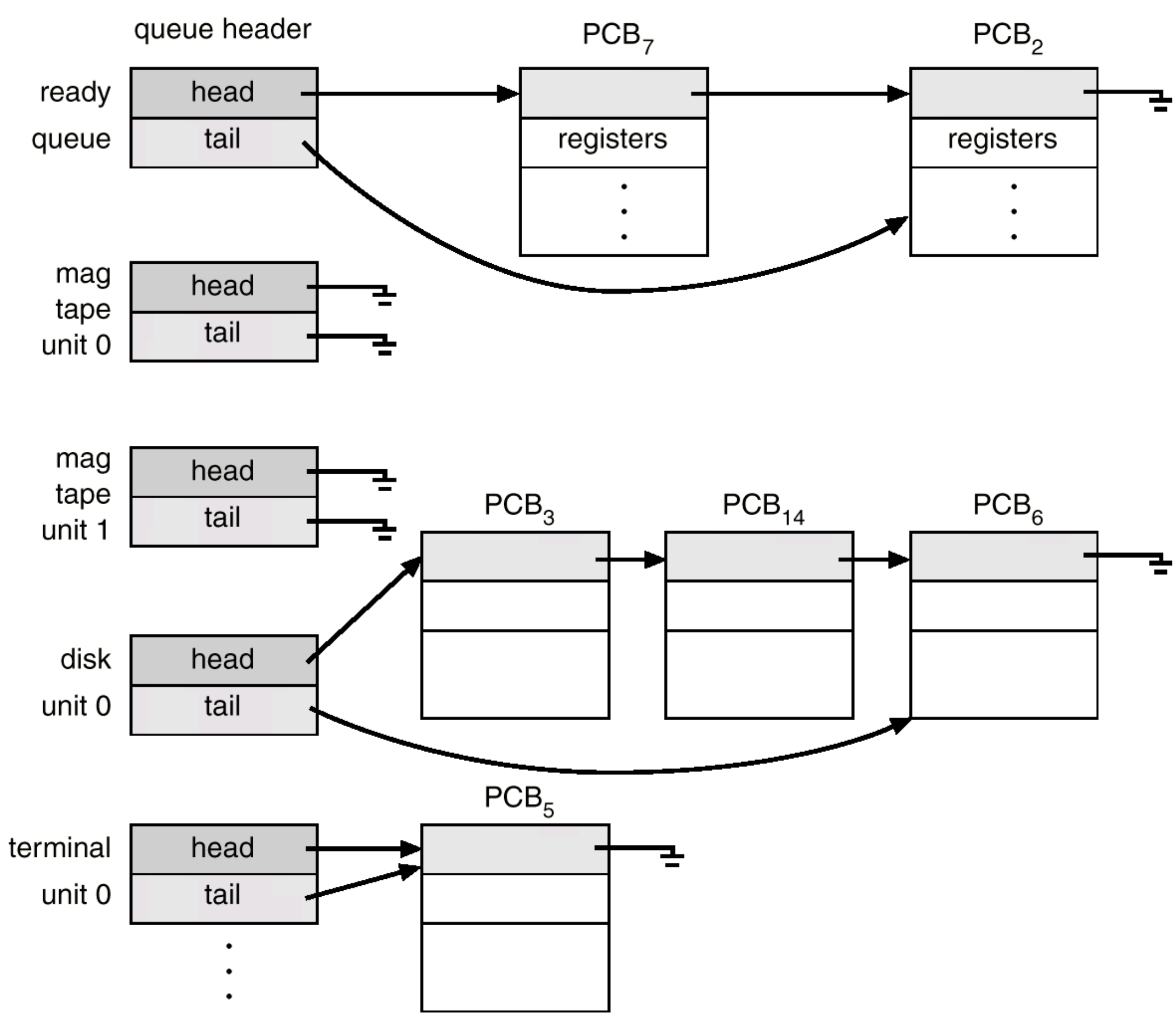
- Minimizing response time requires more context switches for many processes
- → incur more scheduling overhead
- → decrease system throughput

- Scheduling algorithm depends on nature of system
  - Batch vs. interactive
  - Designing a generic AND efficient scheduler is difficult

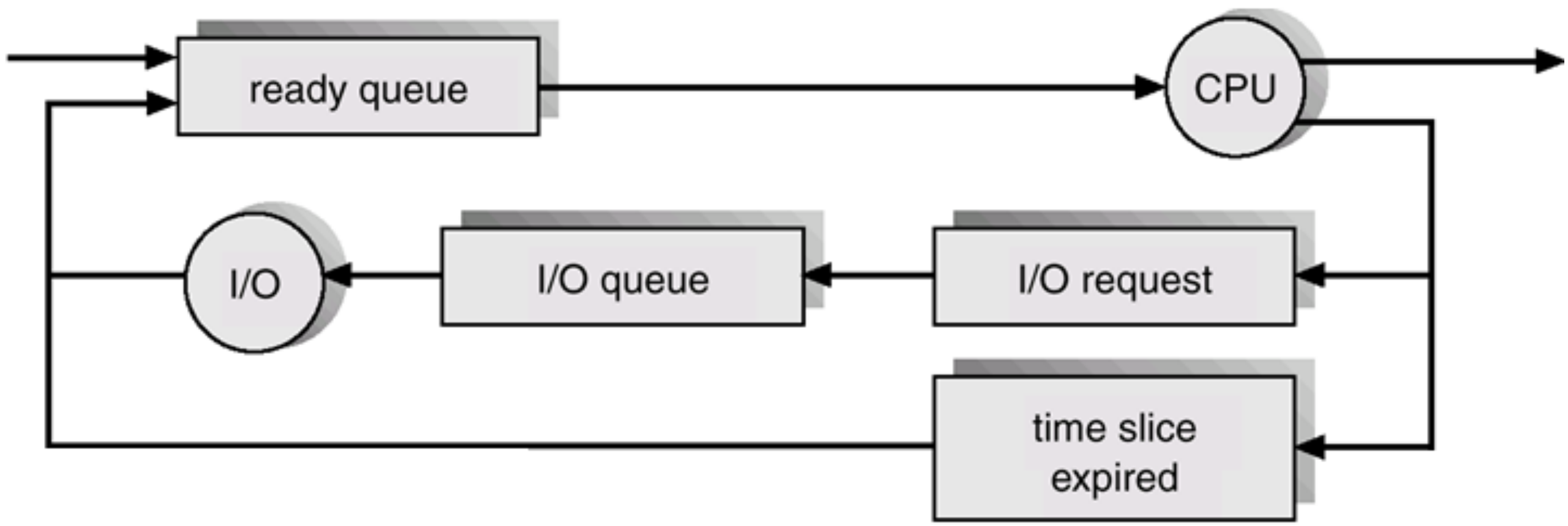
Process Scheduling Queues

- Ready queue
  - Set of processes residing in main memory, ready, and waiting to execute
- Job queue
  - Set of all processes in the system
- Device queues
  - Set of processes waiting for an I/O device
- Process migration between the various queues

Ready Queue And Various I/O Device Queues



Representation of Process Scheduling



Schedulers

- Long-term scheduler (or job scheduler)
  - Which processes should be brought into the ready queue
  - Invoked very infrequently (seconds, minutes) $\Rightarrow$ (may be slow)
  - Controls the **degree of multiprogramming**
- Short-term scheduler (or CPU scheduler)
  - Which process should execute next (allocates CPU)
  - Invoked very frequently (milliseconds) $\Rightarrow$ (must be fast)

Schedulers (Cont.)

- Processes can be described as either:
  - I/O-*bound process*
    - spends more time doing I/O than computations
    - many short CPU bursts
  - *CPU-bound process*
    - spends more time doing computations
    - few very long CPU bursts

Scheduling Algorithms

- Make scheduling decisions
  - Which process(es) to run
  - For how long
  - When to swap running process out

- Examples
  - First come first serve (FCFS), round robin, shortest job first (SJF)
  - Modern OS: priority-based algorithms

Example Scheduling in XINU

- Each process assigned a *priority*
  - Non-negative integer value
  - Initialized when process created
  - Can be changed
- Scheduler chooses process with the highest priority
  - Processes with the same priority are scheduled in a round-robin fashion
- Policy enforced as a system-wide invariant

Implementation of Scheduling

- Process eligible if state is
  - *ready or current*
- To avoid searching process table
  - Keep ready processes on linked list called *ready list*
  - Order ready list by priority
  - Selection in constant time

Example Scheduler Code

```
int resched()
{
        register struct pentry  *optr;  /* pointer to old process entry */
        register struct pentry  *nptr;  /* pointer to new process entry */

        /* no switch needed if current process priority higher than next*/
        if ( ( (optr= &proctab[currpid])->pstate == PRCURR) &&
           (lastkey(rdytail)<optr->pprio)) {
                return(OK);
        }

        /* force context switch */
        if (optr->pstate == PRCURR) {
                optr->pstate = PRREADY;
                insert(currpid,rdyhead,optr->pprio);
        }

        /* remove highest priority process at end of ready list */
        nptr = &proctab[ (currpid = getlast(rdytail)) ];
        nptr->pstate = PRCURR;          /* mark it currently running    */
#ifdef  RTCLOCK
        preempt = QUANTUM;              /* reset preemption counter     */
#endif

        ctxsw((int)&optr->pesp, (int)optr->pirmask, (int)&nptr->pesp, (int)nptr-
   >pirmask);

        /* The OLD process returns here when resumed. */
        return OK;
}
```

Puzzle #1

- Invariant says that at any time, one process must be executing
- Context switch code moves from one process to another
- Question: which process executes the context switch code?

Solution to Puzzle #1

- “Old” process
  - Executes first half of context switch
  - Is suspended
- “New” process
  - Continues executing where previously suspended
  - Usually runs second half of context switch

Puzzle #2

- Invariant says that at any time, one process must be executing
- All user processes may be idle (e.g., applications all wait for input)
- Which process executes?

Solution to Puzzle #2

- OS needs an extra process
  - Called *NULL process*
  - Never terminates
  - Cannot make a system call that takes it out of ready or current state
  - Typically an infinite loop

Lab 1 – Process Scheduling

- *Revisit Xinu scheduling invariant:*

  At any time, the CPU must run the highest priority eligible process. Among processes with equal priority, scheduling is round robin

- *Question: what is a potential problem here?*

PA 1 – Process Scheduling

- The process scheduling policy has a limitation, namely process starvation
- You are asked to implement two different policies
  - Random scheduler
  - Linux Scheduling

Random Scheduler

- Total N processes Pi (i=0..N-1) in the ready queue
- Each Pi
  - Priority: PRIOi
  - Generate a random number *x* between [0, sum(PRIOi)-1]
  - If x < the highest priority, pick the first process in the queue
  - Otherwise, x = x - the highest priority, and check whether x < the second highest priority and so on.

Linux Like Scheduling

- Epoch based scheduling
- Dynamically adjust per-process quantum at the beginning of epoch
  - Quantum defines how many CPU ticks are allocated to the process
  - Consider both priority and past usage
    - E.g. quantum = priority + unused CPU ticks in previous epoch/2
- Select the process with the highest goodness
  - Goodness = 0 if the process uses up its quantum
  - Goodness = priority + unused CPU ticks
  - Round robin among equal goodness

Unix Scheduling – 4.3 BSD

- Multilevel feedback
- For process j at the beginning of timer interval i
  - $P_j(i)=Base_j+CPU_j(i)+Nice_j$
- CPU(i): an estimate of *recent* CPU usage
  - More CPU use leads to lower priority
  - Uses current load (number of ready processes)
  - Aging: CPU(i) = CPU(i-1)/2 + load(i)/2
- $P_j(i)$: priority of j at beginning of interval i; lower value → higher priority
- $Base_j$: base priority of Process j
  - Divided into bands: Swapper, Block I/O device, File, I/O, User process
- $Nice_j$: user controllable factor

Linux Scheduling

- Two algorithms: time-sharing and real-time
- Time-sharing
  - Prioritized credit-based – process with most credits is scheduled next
  - Credit subtracted when timer interrupt occurs
  - When credit = 0, another process chosen
  - When all processes have credit = 0, recrediting occurs
    - Based on factors including priority and past CPU usage
- Real-time
  - Soft real-time
    - Rate monotonic scheduling (not covered)
    - End deadline first scheduling (not covered)

PA 1 – Process Scheduling

- Read relevant source code in Xinu
  - Process queue management
    - h/q.h sys/queue.c sys/insert.c, …
  - Proc. creation/suspension/resumption/termination:
    - sys/create.c, sys/suspend.c sys/resume.c, sys/kill.c
  - Priority change
    - sys/chprio.c
  - Process scheduling
    - sys/resched.c
  - Other initialization code
    - sys/initialize.c

Dynamic Priority-based Scheduling

- In Xinu: Timer interrupt handler
  - Related files: sys/clkint.S sys/clkinit.c
  - Interrupt rate – based on clock timer
    - ctr1000: 1ms
  - Scheduling rate:
    - Interrupt rate * QUANTUM
  - Others
    - preempt: preemption counter

Next Lecture

- Process Synchronization