Scheduling

There cannot be a crisis next week. My schedule is already full.

— *Henry Kissinger*

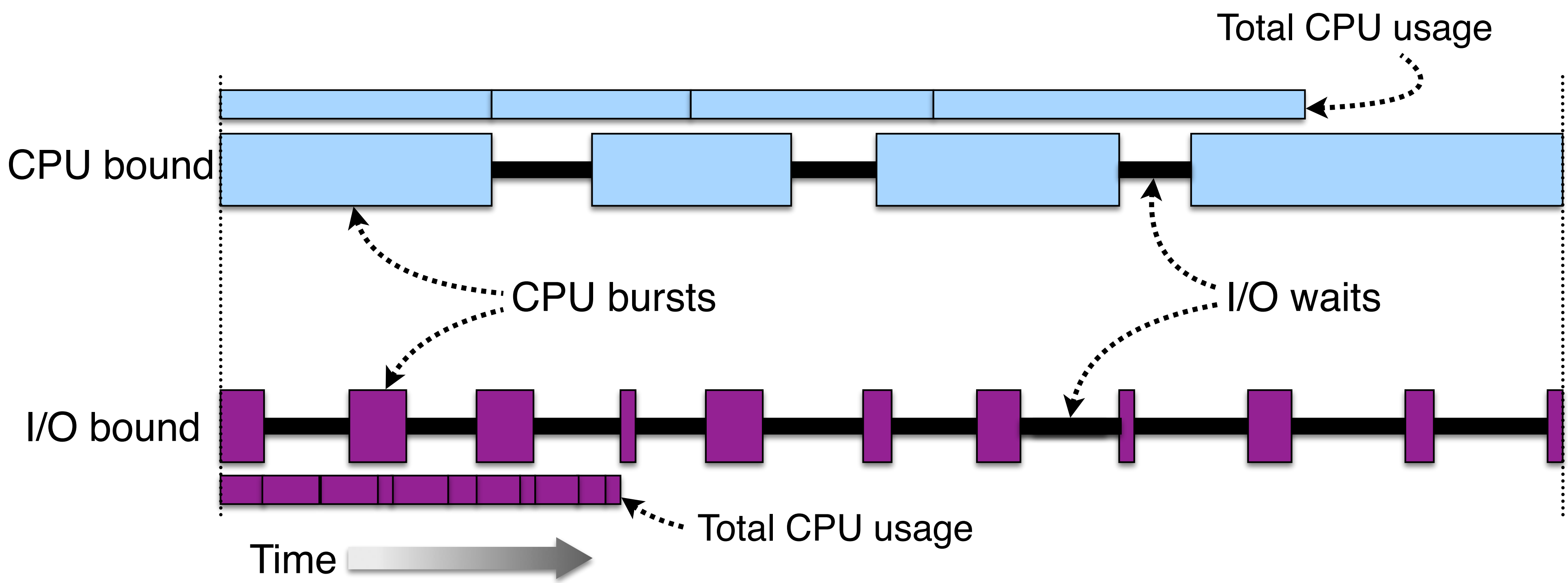
Scheduling

- What is scheduling?
  - Goals
  - Mechanisms
- Scheduling on batch systems
- Scheduling on interactive systems
- Other kinds of scheduling
  - Real-time scheduling

Or you could do this …

Baskin Engineering UC SANTA CRUZ

Why schedule processes?



- Bursts of CPU usage alternate with periods of I/O wait
- Some processes are CPU-bound: they don't many I/O requests
- Other processes are I/O-bound and make many kernel requests

When are processes scheduled?

- At the time they enter the system
  - Common in batch systems
  - Two types of batch scheduling
    - Submission of a new job causes the scheduler to run
    - Scheduling only done when a job voluntarily gives up the CPU (i.e., while waiting for an I/O request)
- At relatively fixed intervals (clock interrupts)
  - Necessary for interactive systems
  - May also be used for batch systems
  - Scheduling algorithms at each interrupt, and picks the next process from the pool of “ready” processes

Scheduling goals

- All systems
  - Fairness: give each process a fair share of the CPU
  - Enforcement: ensure that the stated policy is carried out
  - Balance: keep all parts of the system busy
- Batch systems
  - Throughput: maximize jobs per unit time (hour)
  - Turnaround time: minimize time users wait for jobs
  - CPU utilization: keep the CPU as busy as possible
- Interactive systems
  - Response time: respond quickly to users' requests
  - Proportionality: meet users' expectations
- Real-time systems
  - Meet deadlines: missing deadlines is a system failure!
  - Predictability: same type of behavior for each time slice

Measuring scheduling performance

- Throughput
  - Amount of work completed per second (minute, hour)
  - Higher throughput usually means better utilized system
- Response time
  - Response time is time from when a command is submitted until results are returned
  - Can measure average, variance, minimum, maximum, …
  - May be more useful to measure time spent waiting
  - Can also measure how often response time is faster than a given time (e.g., 100 ms): useful for real-time systems and servers
- Turnaround time
  - Like response time, but for batch jobs (response is the completion of the process)
- Usually not possible to optimize for all metrics with a single scheduling algorithm

Interactive vs. batch scheduling

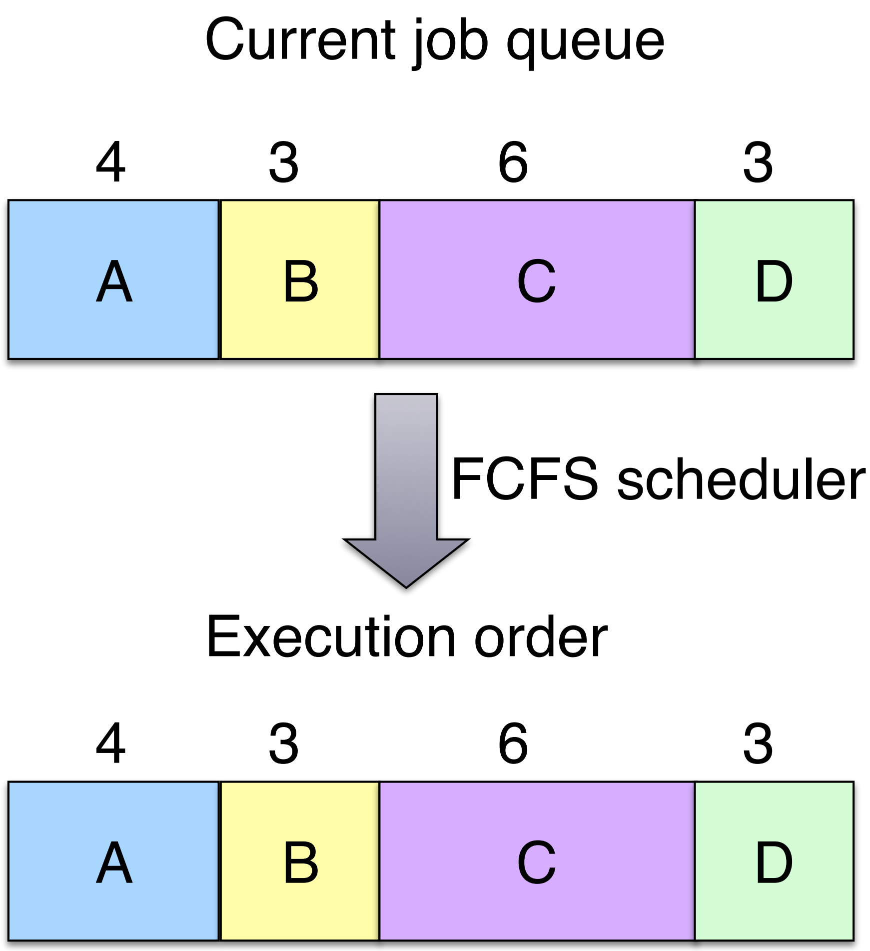
Batch

First-Come-First-Served (FCFS)
Shortest Job First (SJF)
Shortest Remaining Time First (SRTF)
Priority (non-preemptive)

Interactive

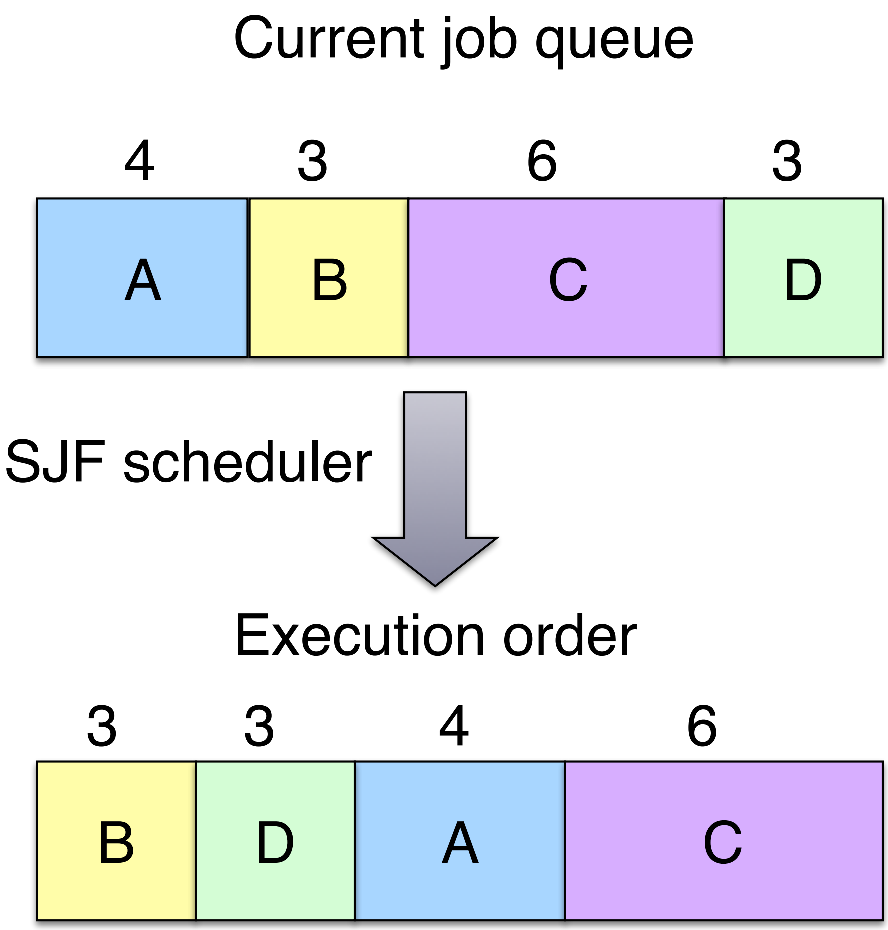
Round-Robin (RR)
Priority (preemptive)
Multi-level feedback queue
Lottery scheduling

First Come, First Served (FCFS)



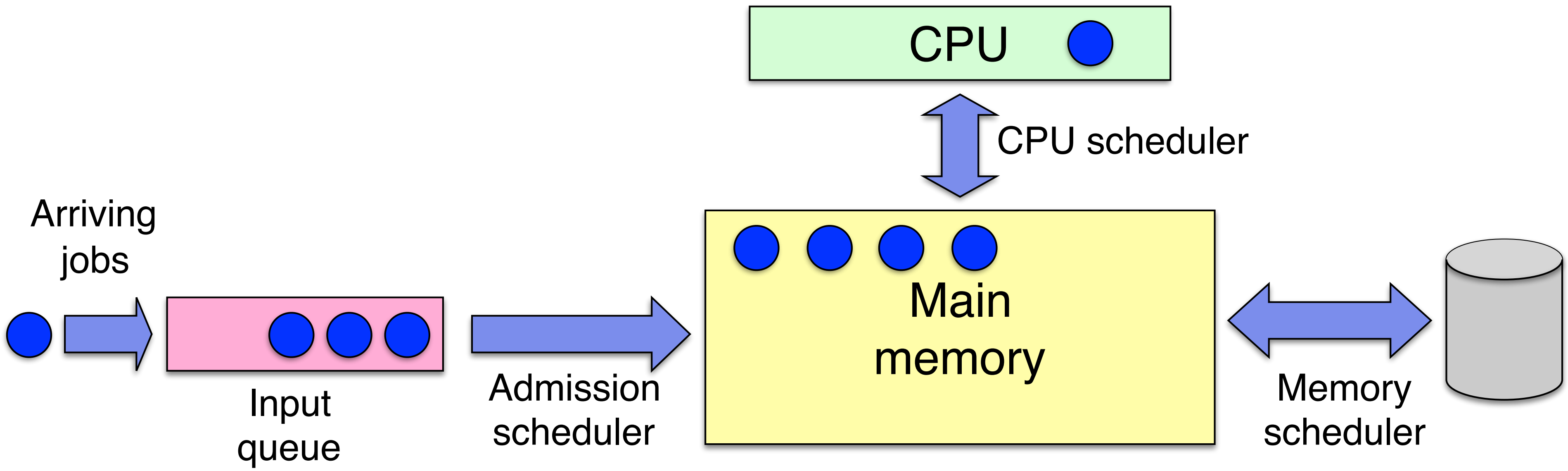
- Goal: do jobs in the order they arrive
  - Fair in the same way a bank teller line is fair
- Simple algorithm!
- Problem: long jobs delay every job after them
  - Many processes may wait for a single long job

Shortest Job First (SJF)



- Goal: do the shortest job first
  - Short jobs complete first
  - Long jobs delay every job after them
- Jobs sorted in increasing order of execution time
  - Ordering of ties doesn't matter
- Shortest Remaining Time First (SRTF): preemptive form of SJF
  - Re-evaluate when a new job is submitted
- Problem: how does the scheduler know how long a job will take?

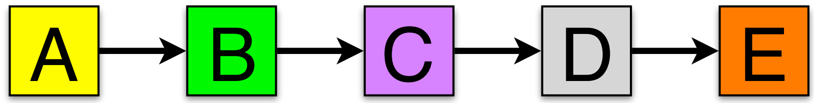
Three-level scheduling

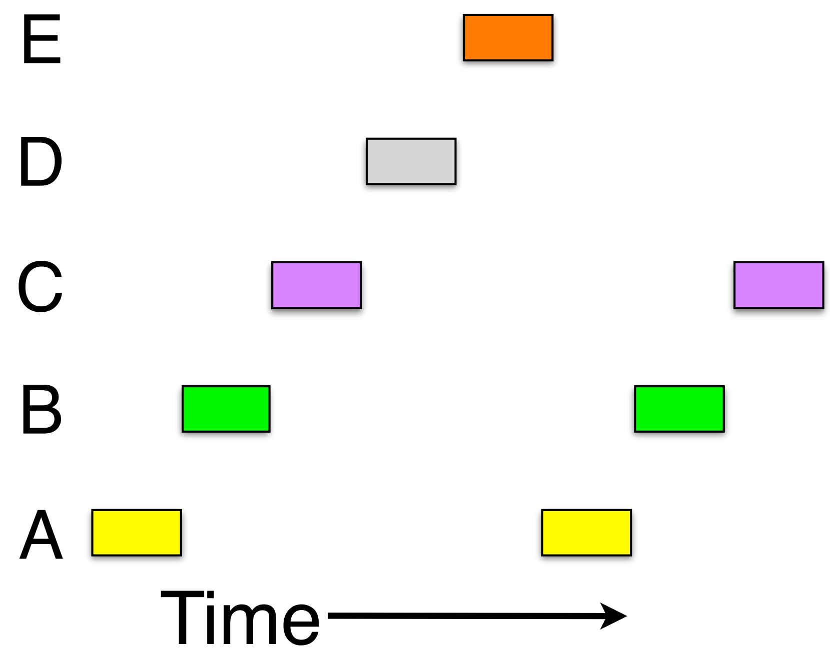


- Jobs held in input queue until moved into memory
  - Pick "complementary jobs": small & large, CPU- & I/O-intensive
  - Jobs move into memory when admitted
- CPU scheduler picks next job to run
- Memory scheduler picks some jobs from main memory and moves them to disk if insufficient memory space

Round Robin (RR) scheduling

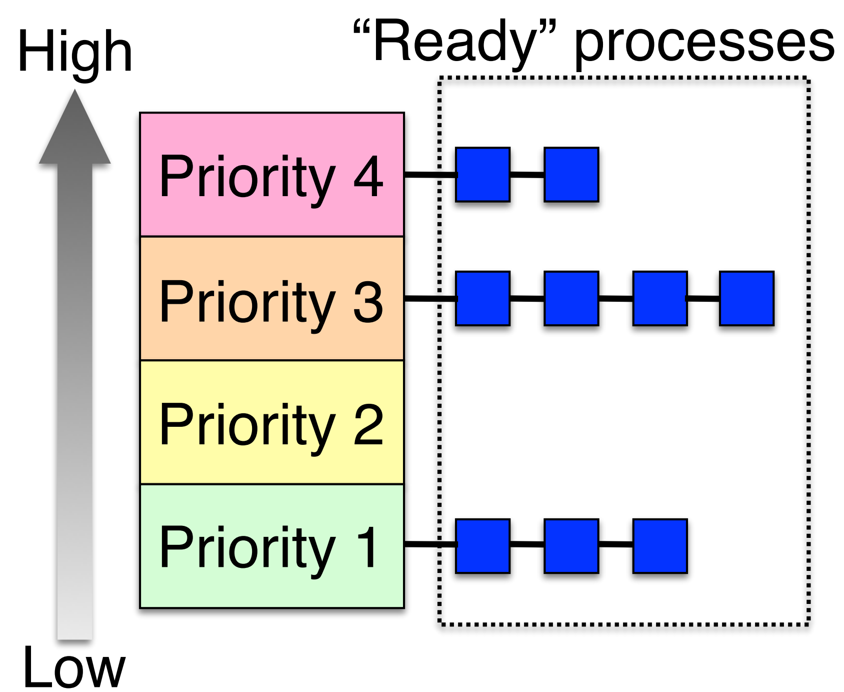
- Scheduling interactive processes
  - Give each process a fixed time slot (quantum)
  - Rotate through "ready" processes
  - Each process makes some progress
- What's a good quantum?
  - Too short: many process switches hurt efficiency
  - Too long: poor response to interactive requests
  - Typical length: 10–100 ms
- "Strict" rotation: round robin





Priority scheduling

- Assign a priority to each process
  - "Ready" process with highest priority allowed to run
  - Running process may be interrupted after its quantum expires
- Priorities may be assigned dynamically
  - Reduced when a process uses CPU time
  - Increased when a process waits for I/O
- Often, processes grouped into multiple queues based on priority, and run round-robin per queue



Shortest process next

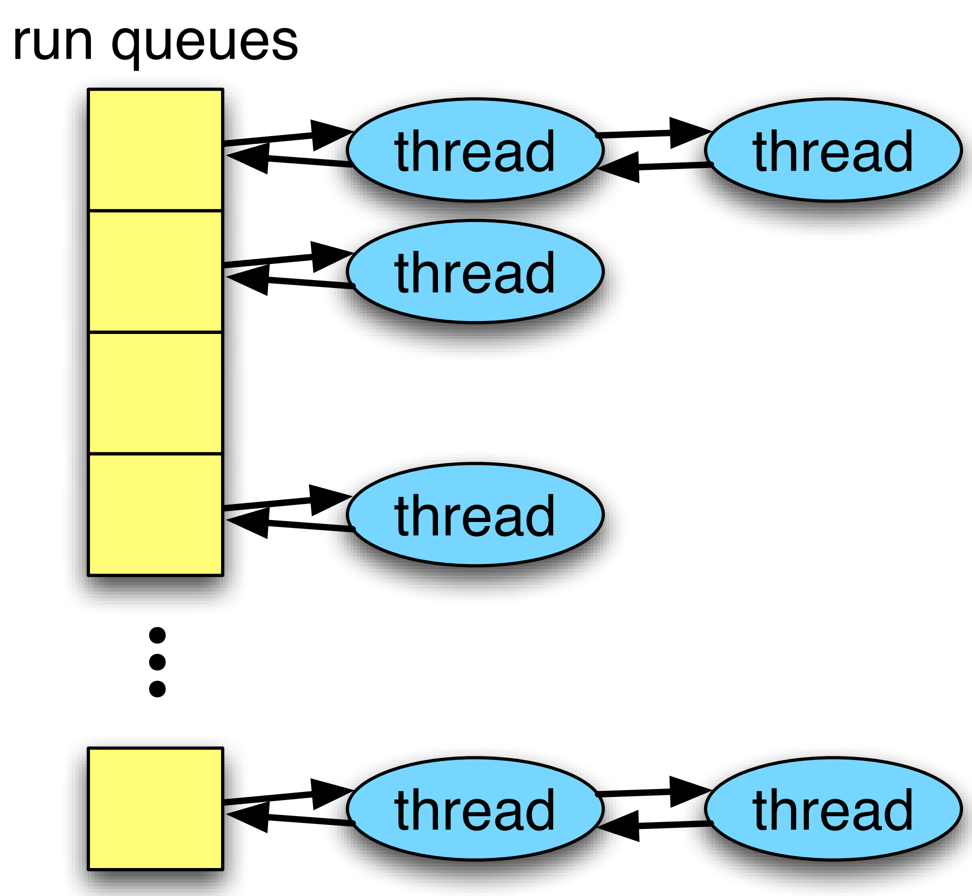
- Run the process that will finish the soonest
  - In interactive systems, job completion time is unknown!
- Guess at completion time based on previous runs
  - Update estimate each time the job is run
  - Estimate is a combination of previous estimate and most recent run time
- Not often used because round robin with priority works so well!

Lottery scheduling

- Give processes "tickets" for CPU time
  - More tickets ➡ higher share of CPU
- Each quantum, pick a ticket at random
  - If there are *n* tickets, pick a number from 1 to *n*
    - Pseudo-random number is OK if it's a good RNG
  - Process holding the ticket gets to run for a quantum
  - This can be implemented efficiently without "real" tickets
    - Track range of tickets belonging to each process
- Over the long run, each process gets the CPU *m*/*n* of the time if the process has *m* of the *n* existing tickets
- Tickets can be transferred
  - Cooperating processes can exchange tickets
  - Clients can transfer tickets to a server so it can have a higher priority
  - Parent (shell) can transfer tickets to a child process

Scheduling in BSD4

- Quantum is 100 ms: longest that's OK for interactive scheduling
- Scheduler is based on multi-level feedback queues
  - Priority is based on two things
    - Resource requirements: blocked threads have higher priority when rescheduled
    - Previous CPU usage: CPU hogs have lower priority
- Each thread is placed into a run queue for its priority
  - Head of highest-priority run queue with a ready thread runs next

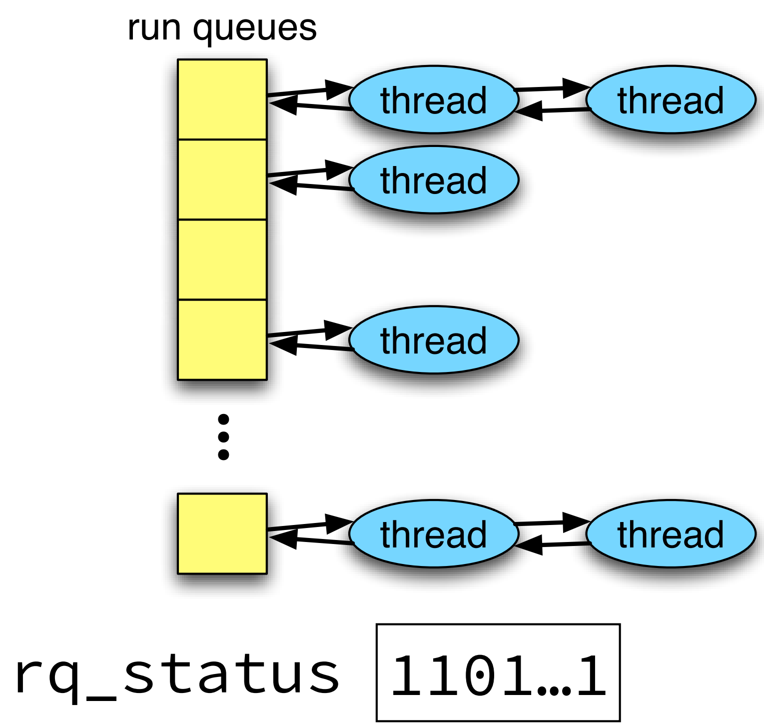


Calculating priority in BSD4

- Thread priority is set by:
  $pri = MIN + [estcpu/4] + 2 \times nice$
  - Values above *MAX* are set to *MAX*
  - *MIN*=160, *MAX*=223
  - *nice* is set by the user to manually lower thread priority
  - *estcpu* is an estimate of the number of “ready” processes in the CPU when the calculation is made
    - Has a bit of “memory” so it doesn’t change too quickly
    - *estcpu* is updated each clock tick
  - Higher numbers indicate lower priority: threads with lowest priority values are scheduled first
- Thread priority is set every 40 ms
- Scheduling is more complex for multiprocessors...

FreeBSD ULE scheduler

- Not an acronym: scheduler code is in `sched_ule.c`
- Divided into two parts
  - Low-level scheduler: runs on every context switch
    - Selects next runnable thread from the run queues
    - Uses bitmask to quickly find first non-empty queue
  - High-level scheduler: runs a few times per second
    - Sets thread priorities
    - Selects a processor on which to run a particular thread
- Run queues only contain runnable threads
  - Blocked threads placed on
    - Turnstile: blocked on short-term lock
    - Sleepqueue: blocked on medium/long-term lock
  - Runnable thread that consumes its quantum placed at the end of its run queue

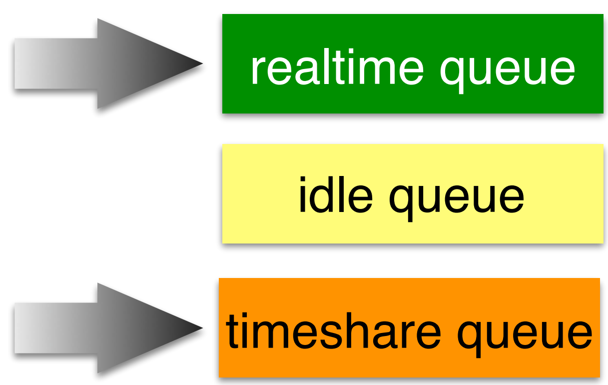


Baskin Engineering UC SANTA CRUZ

FreeBSD ULE high-level scheduler

- High-level scheduler runs infrequently
  - Adjusts length of quantum
  - Adjusts process priority
- Prioritize interactive processes: use calculated “interactivity” score
- Three sets of queues per CPU
  - Idle queue: idle threads
  - Realtime queue: real-time & interactive threads
  - Timeshare queue: long-running (batch) threads
    - Uses “calendar queue” to handle lots of batch-type threads efficiently
    - Lower priority processes inserted further back in queue: take longer to bubble to front

$$\text{interactivity} = \begin{cases} \dfrac{scaling\_factor}{sleep/run} & \text{Yes!} \\ \text{sleep} > \text{run?} & \\ \dfrac{scaling\_factor}{run/sleep} + scaling\_factor & \text{No} \end{cases}$$



Baskin Engineering UC SANTA CRUZ

Scheduling in Linux

- Three classes of processes
  - Conventional
  - Real-time (round-robin)
  - Real-time (FIFO)
- Queue structure similar to BSD
  - Two sets of queues (0–99 real-time, 100–139 conventional): active and expired
  - Scheduler runs process in lowest-valued active queue
  - Conventional threads placed in expired queue when their quantum is up
    - May be scheduled out before quantum expires: put back into active queue
  - Real-time threads placed back into active queue

Linux: rescheduling processes

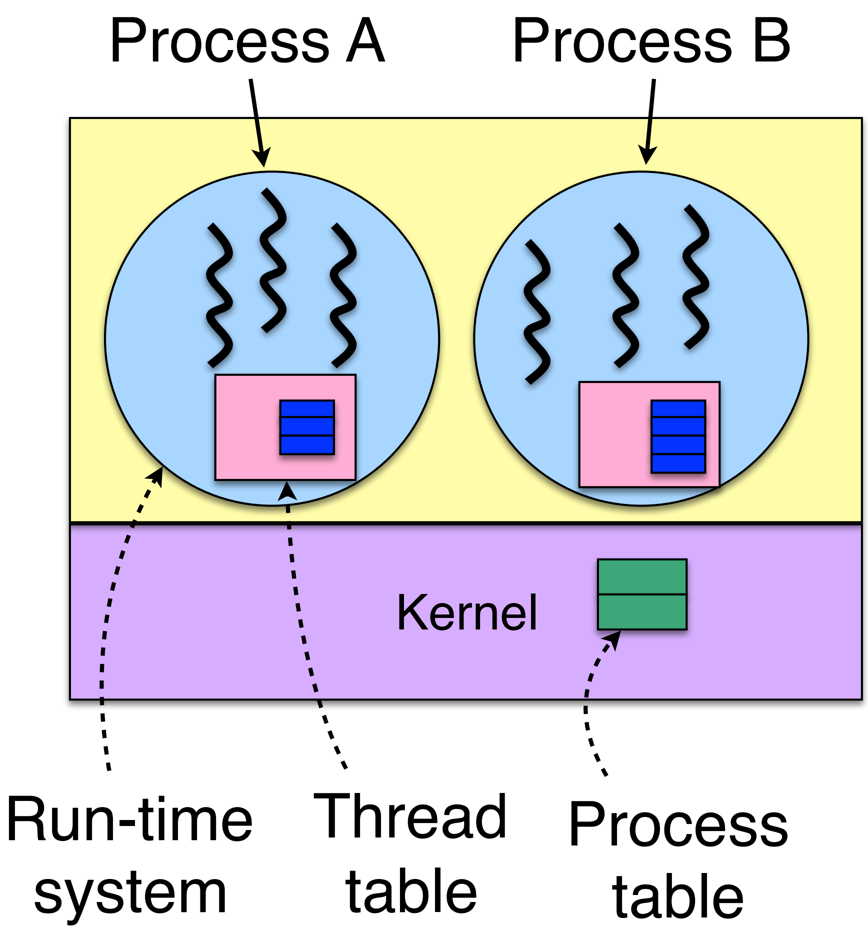
- Re-evaluate priorities when there are no threads on the active queue
- Thread priority is re-evaluated based on previous run time and sleep time
  - Threads that ran more and slept less are penalized with a worse priority
- Quantum is regenerated based on original (static) priority
  - Higher-priority threads are given longer quanta
  - Process may lose the CPU before quantum is up (if a higher-priority thread becomes available)

Further details in *Understanding Linux Kernel Internals* (3rd edition), Bovet & Cesati

Policy versus mechanism

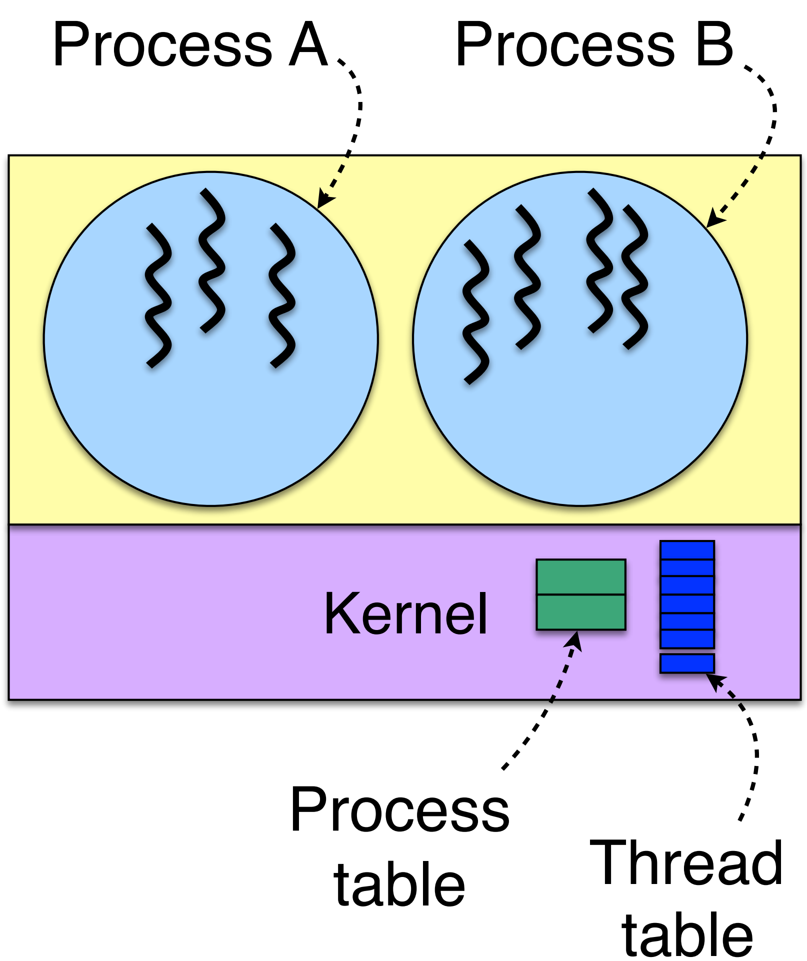
- Separate what may be done from how it is done
  - Mechanism allows
    - Priorities to be assigned to processes
    - CPU to select processes with high priorities
  - Policy set by what priorities are assigned to processes
- Scheduling algorithm parameterized
  - Mechanism in the kernel
  - Priorities assigned in the kernel or by users
- Parameters may be set by user processes
  - Don’t allow a user process to take over the system!
  - Allow a user process to voluntarily lower its own priority
  - Allow a user process to assign priority to its threads

Scheduling user-level threads



- Kernel picks a process to run next
- Run-time system (at user level) schedules threads
  - Run each thread for less than process quantum
  - Example
    - Processes get 40ms each
    - Threads get 10ms each
- Example schedule: A1,A2,A3,A1,B1,B3,B2,B3
- Not possible: A1,A2,B1,B2,A3,B3,A2,B1

Scheduling kernel-level threads



- ❖ Kernel schedules each thread
  - No restrictions on ordering
  - May be more difficult for each process to specify priorities
- ❖ Example schedule:
  A1, A2, A3, A1,
  B1, B3, B2, B3
- ❖ Also possible:
  A1, A2, B1, B2,
  A3, B3, A2, B1

Baskin Engineering UC SANTA CRUZ