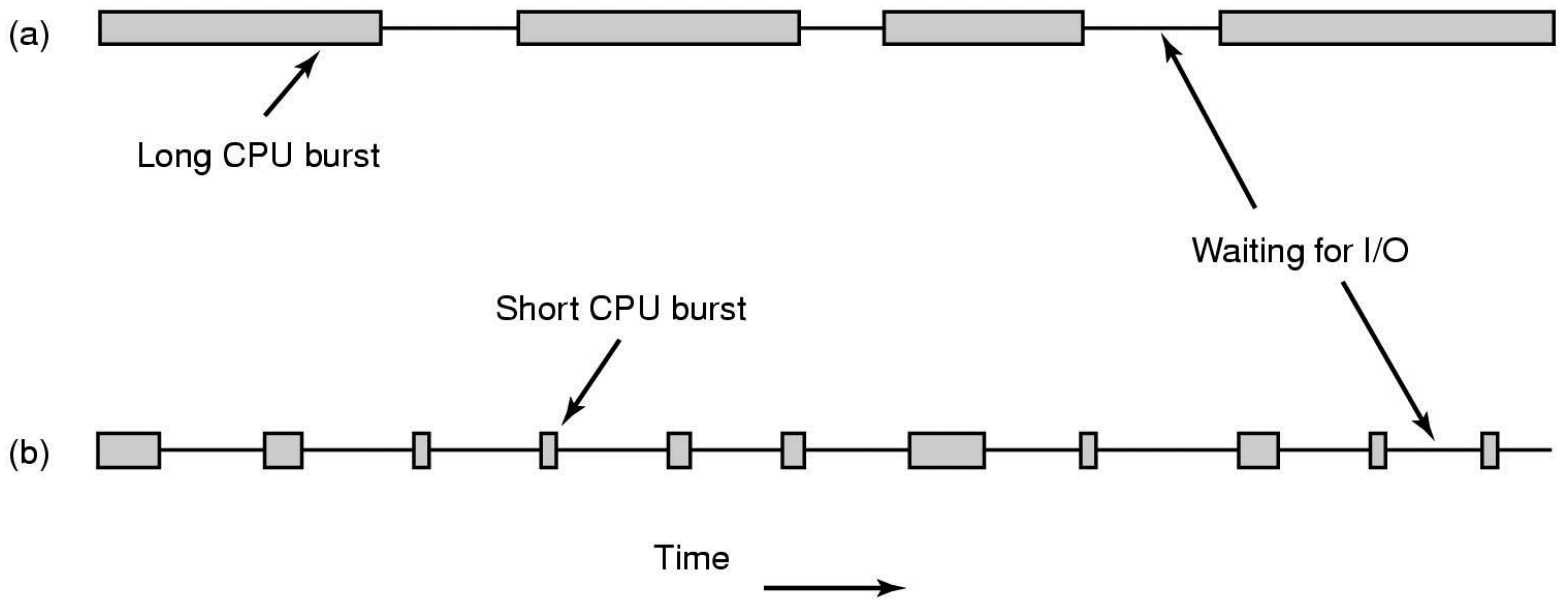
Introduction to Scheduling

- Old days of batch systems and timesharing systems
  - CPU time was a scarce resource on the old machines.
  - A great deal of work has gone into devising clever and efficient scheduling algorithms.
- Advent of Personal Computers
  - Most of the time, there is only one active process.
  - CPU has gotten so much faster that CPU is rarely a scarce resource any more.
  - Scheduling does not matter much on simple PCs.
- High-end networked workstations and servers
  - Process scheduling matters very much.
- Process switching is expensive.

Process Behavior



- Bursts of CPU usage alternate with periods of I/O wait
  - (a) a CPU-bound(or compute-bound) process
  - (b) an I/O-bound process
    - I/O: A process enters the blocked state waiting for an external device to complete its work

When to Schedule

- When a process created
  - Run the parent process or the child process
- When a process exits
  - Some other process must be chosen from the set of ready processes.
- When a process blocks
  - Another process has to be selected to run
- When an I/O interrupt occurs
  - Runs newly ready process, interrupted process, or third process
- Schedling algorithms can be divided into two categories with respect to how they deal with clock interrupts
  - Nonpreemptive
    - Picks a process to run and lets it run until it blocks or until it voluntarily releases the CPU
  - Preemptive
    - Picks a process and lets it run for a maximum of some fixed time. If it is still running at the end of the time interval, it is suspended and the scheduler picks another process to run

Categories of Scheduling Algorithms

- Batch
  - Nonpreemptive algorithms or preemptive algorithms with long time periods for each process
  - Reduces process switches and improves performance
- Interactive
  - Preemptive algorithms to keep one process from hogging the CPU and denying service to the others
- Real time
  - Preemption is sometimes not needed because the processes know that they may not run for long periods of time and usually do their work and block quickly

Scheduling Algorithm Goals

All systems

- Fairness - giving each process a fair share of the CPU
- Policy enforcement - seeing that stated policy is carried out
- Balance - keeping all parts of the system busy

Batch systems

- Throughput - maximize jobs per hour
- Turnaround time - minimize time between submission and termination
- CPU utilization - keep the CPU busy all the time

Interactive systems

- Response time - respond to requests quickly
- Proportionality - meet users' expectations

Real-time systems

- Meeting deadlines - avoid losing data
- Predictability - avoid quality degradation in multimedia systems

Scheduling Algorithm Goals

Scheduling in Batch Systems (1)

- First-Come First-Served
  - Processes are assigned the CPU in the order they request it.
  - Single queue of ready processes
  - Nonpreemptive
  - When the running process blocks, the first process on the queue is run next. When a blocked process becomes ready, it is put on the end of the queue.
  - Easy to understand and easy to program
  - Disadvantage
    - One compute-bound process that runs for 1 sec at a time and reads a disk block.
    - Many I/O-bound processes that use little CPU time but each has to perform 1000 disk reads to complete
    - Result: Each I/O-bound process gets to read 1 block per second and will take 1000 sec to finish.

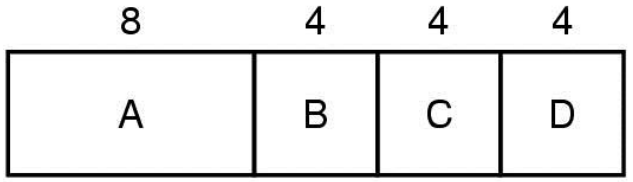
Scheduling in Batch Systems (2)

- Shortest Job First
  - Nonpreemptive
  - Run times are known in advance
  - Optimal average turnaround time

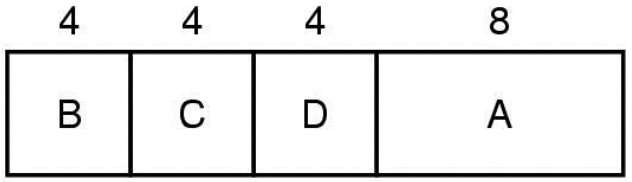
    (when all the jobs are available simultaneously)
  - Average turnaround time

    In (a), $(8+12+16+20)/4 = 14$ minutes

    In (b), $(4+8+12+20)/4 = 11$ minutes



(a)

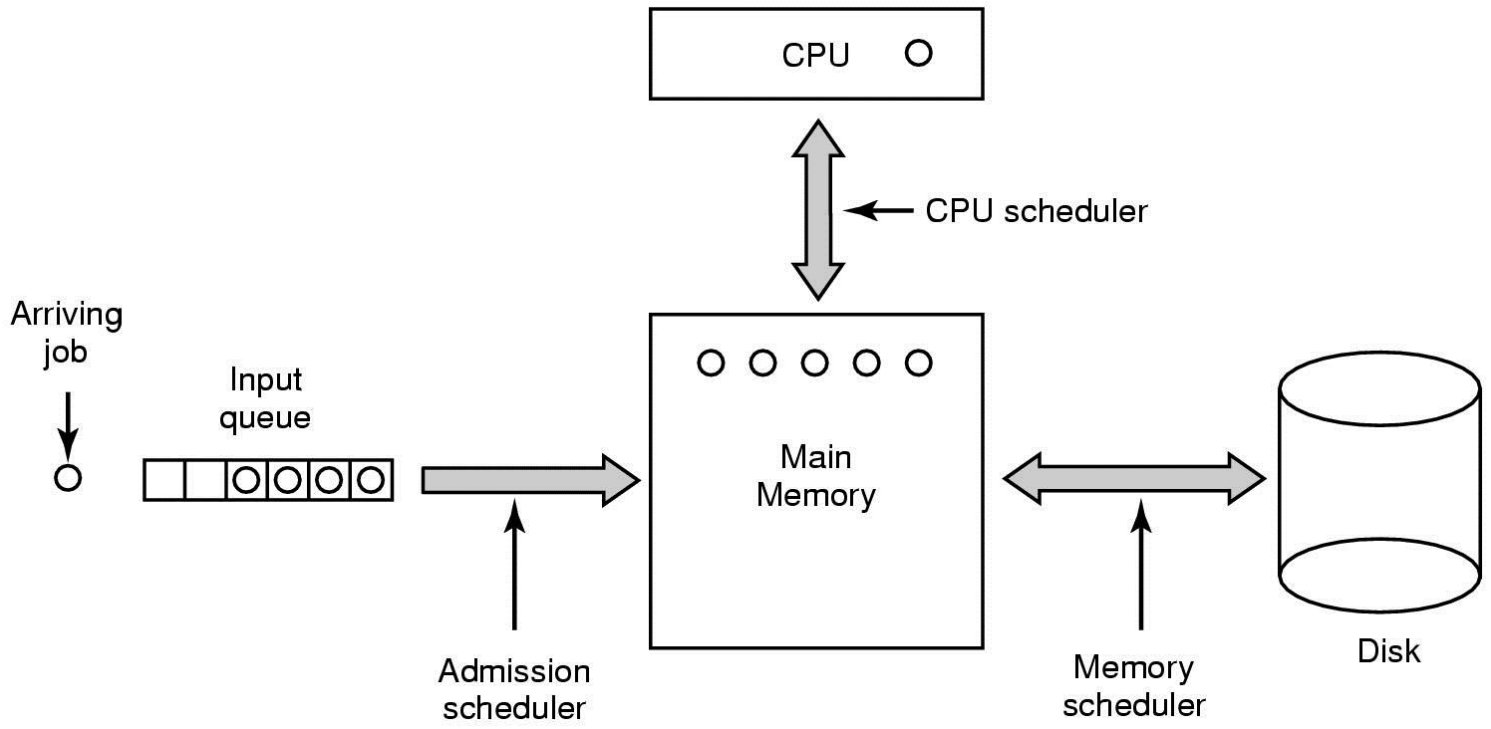


(b)

Scheduling in Batch Systems (3)

- Shortest Remaining Time Next
  - Preemptive version of shortest job first
  - Chooses the process whose remaining run time is the shortest
  - When a new job arrives, its total time is compared to the current process' remaining time. If the new job needs less time to finish than the current process, the current process is suspended and the new job is started.

Scheduling in Batch Systems (4)

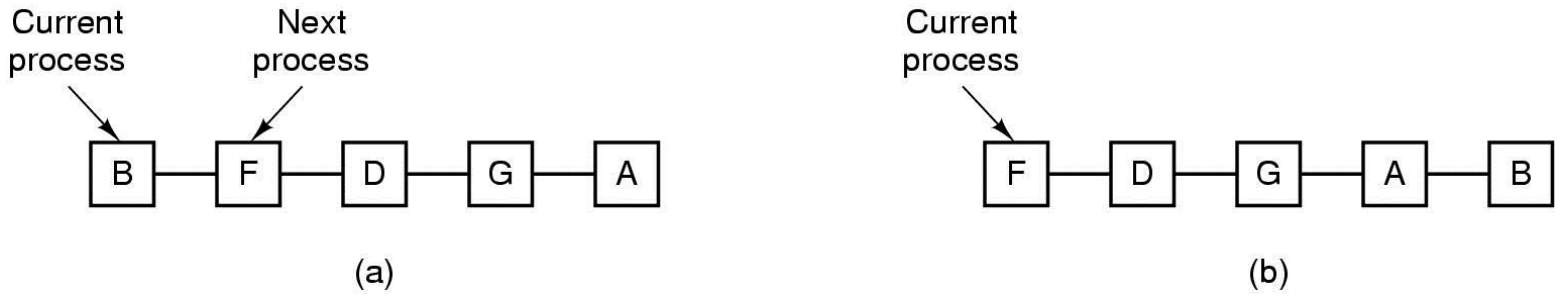


Three level scheduling

Scheduling in Batch Systems (5)

- Three-Level Scheduling
  - Admission Scheduler
    - Decides which jobs to admit to the system from the input queue
    - E.g. Looks for a mix of compute-bound jobs and I/O-bound jobs
  - Memory Scheduler
    - Decides which processes are kept in memory (degree of multiprogramming) and which on disk
    - Bringing a process in from disk is expensive
    - Periodically reviews each process on disk whether or not to bring it into memory based on the following criteria:
      - How long has it been since the process was swapped in or out?
      - How much CPU time has the process had recently ?
      - How big is the process?
      - How important is the process?
  - CPU scheduler
    - Picks one of the ready processes in main memory to run next

Scheduling in Interactive Systems (1)



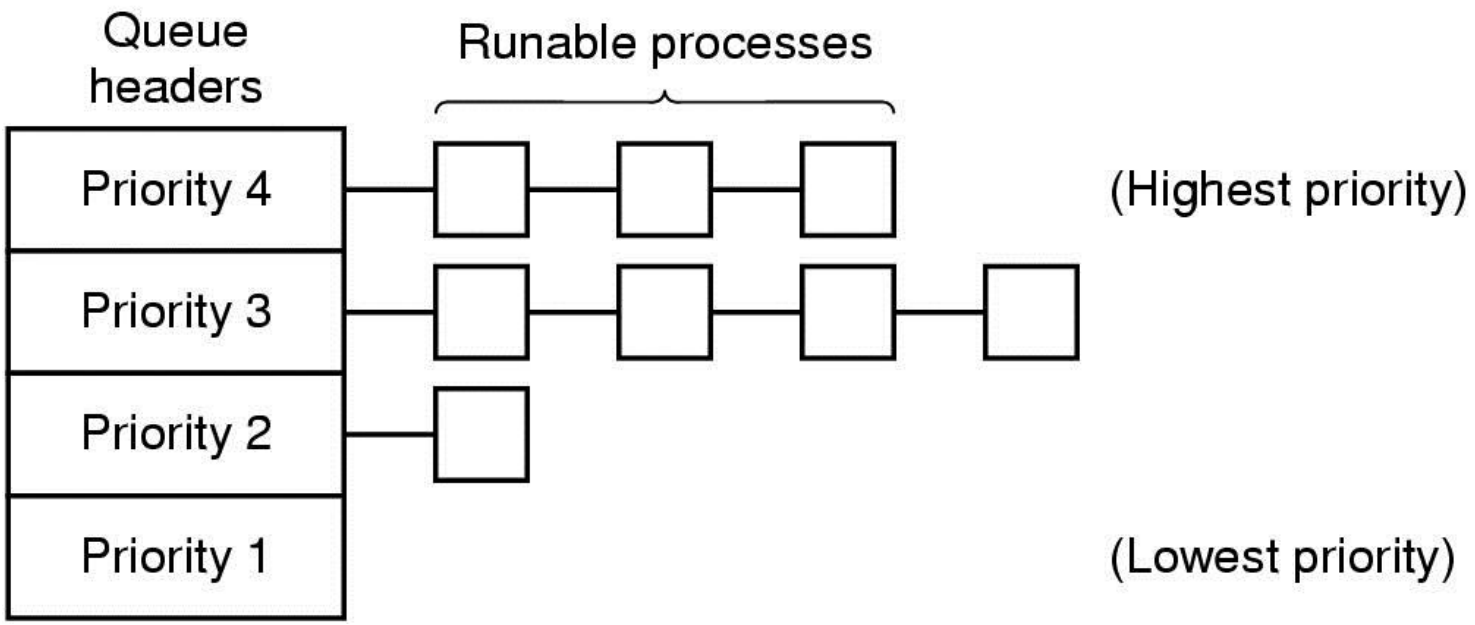
- Round Robin Scheduling
  - (a) list of runnable processes
  - (b) list of runnable processes after B uses up its quantum
  - Length of the quantum
    - Too short: causes too many process switches (context switches) and lowers the CPU efficiency.
    - Too long: causes poor response to short interactive requests.

Scheduling in Interactive Systems (2)

- Priority Scheduling
  - Each process is assigned a priority, and the runnable process with the highest priority is allowed to run.
  - To prevent high-priority processes from running indefinitely,
    - decreases the priority of the currently running process at each clock tick. If the priority drops below that of the next highest process, a process switch occurs.
    - Assigns each process a maximum time quantum that it is allowed to run.
  - Priority assignment
    - Static
      - Based on the user, price, etc. (cf. nice)
    - Dynamic
      - Giving good service to I/O-bound processes:

        Sets the priority to 1/f (f: fraction of the last quantum that a process used.)

Scheduling in Interactive Systems (2)



- A scheduling algorithm with four priority classes
  - Use priority scheduling among the classes
  - Use round-robin scheduling within each class.
  - As long as there are runnable processes in the higher priority class, just run each one for one quantum, round-robin fashion.
  - If priorities are not adjusted occasionally, lower priority classes can all starve to death.

Scheduling in Interactive Systems (3)

- Multiple Queues
  - CTSS (1962)
    - 1. It's more efficient to give CPU-bound processes a large quantum.
    - 2. But, giving all processes a large quantum would mean poor response time.
    - Sets up priority classes
    - Processes in the highest class were run for one quantum. (next highest class: two quanta, next: four quanta, etc.) Whenever a process used up all the quanta allocated to it, it was moved down one class.
    - E.g. a process needed to compute for 100 quanta
      - It will get 1, 2, 4, 8, 16, 32 and 64(37) quanta in each run.
      - 7 switches needed instead of 100 with a pure round-robin algorithm(cf. 1)
    - As the process sank deeper and deeper into the priority queues, it would run less and less frequently, saving the CPU for short, interactive processes.(cf. 2)
  - XDS 940 (1968)
    - Four priority classes
      - Terminal: when a process waiting for terminal input was awakened
      - I/O: when a process waiting for a disk block became ready
      - Short quantum: when a process was still running when its quantum ran out
      - Long quantum: when a process used up its quantum too many times in a row without blocking for terminal or other I/O

Scheduling in Interactive Systems (4)

- Shortest Process Next
  - Minimum average response time
  - Make estimates based on past behavior and run the process with the shortest estimated running time
    - Aging
      - Estimate: aT0+(1-a)T1
        - » T0: estimated time per command
        - » T1: next run measured
      - With a=1/2, we get successive estimates of
        - » T0, T0/2+T1/2, T0/4+T1/4+T2/2, T0/8+T1/8+T2/4+T3/2

Scheduling in Interactive Systems (5)

- Guaranteed Scheduling
  - With n processes running, each one should get 1/n of the CPU cycles.
    - Keeps track of how much CPU each process has had since its creation
    - Computes

$$\text{ratio} = \frac{\text{actual CPU time consumed}}{\text{CPU time entitled}}$$

      - E.g. 0.5 : a process has only had half of what it should have had
    - Runs the process with the lowest ratio until its ratio has moved above its closest competitor

Scheduling in Interactive Systems (6)

- Lottery Scheduling
  - Gives processes lottery tickets for various system resources, such as CPU time.
  - Whenever a scheduling decision has to be made, a lottery ticket is chosen at random, and the process holding that ticket gets the resource.
  - A process holding a fraction f of the tickets will get about a fraction f of the resource.
  - Highly responsive
  - Cooperating processes can exchange tickets if they wish.
  - Video server in which several processes are feeding video streams to their clients at different frame rates.

Scheduling in Interactive Systems (7)

- Fair-Share Scheduling
  - Previous scheduling
    - Owner of the process not considered
    - E.g. in round-robin
      - User 1 with 9 processes
      - User 2 with 1 process gets only 10% of the CPU
  - Takes into account who owns a process before scheduling it
  - E.g.
    - User 1 with A,B,C,D : 50%
    - User 2 with E: 50%
    - AEBECEDEAEBE…

Scheduling in Real-Time Systems

- A real-time system must react appropriately to external events by the deadline.
- E.g.
  - Compact disk player, patient monitoring in a hospital intensive-care unit, autopilot in an aircraft, robot control in a factory
- Having the right answer too late is often just as bad as not having it at all.
- Hard real time
  - Absolute deadlines must be met
- Soft real time
  - Missing an occasional deadline is undesirable, but nevertheless tolerable.
- Real-time behavior
  - Achieved by dividing the program into a number of processes, each of whose behavior is predictable and known in advance.

Scheduling in Real-Time Systems

- Events
  - Periodic: occuring at regular intervals
  - Aperiodic: occuring unpredictably
- Schedulable real-time system
  - Given
    - *m* periodic events
    - event *i* occurs with period P_i and requires C_i seconds to handle
  - Then the load can only be handled if the sum of the fraction of the CPU being used by each process <= 1.

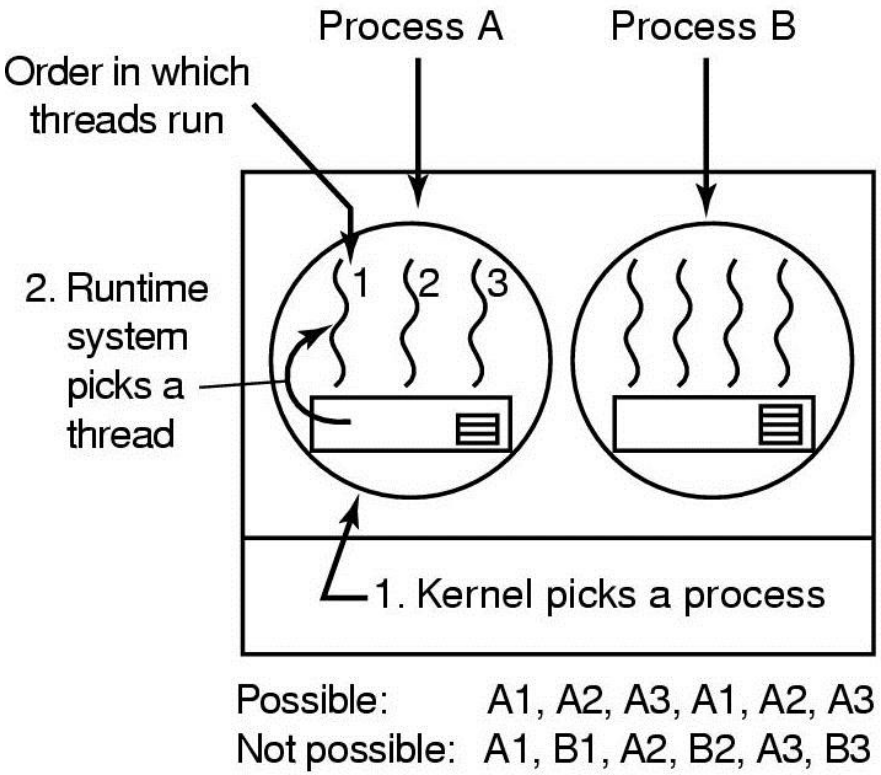
$$\sum_{i=1}^{m} \frac{C_i}{P_i} \leq 1$$

    - E.g. periods 100ms(CPU time 50ms), 200(30), 500(100)
      - To add a fourth event with a period of 1 sec,
        - » CPU time <= 150ms
- Real-time scheduling algorithms
  - Static : make scheduling decisions before the system starts running.
  - Dynamic : make scheduling decisions at run time

Policy versus Mechanism

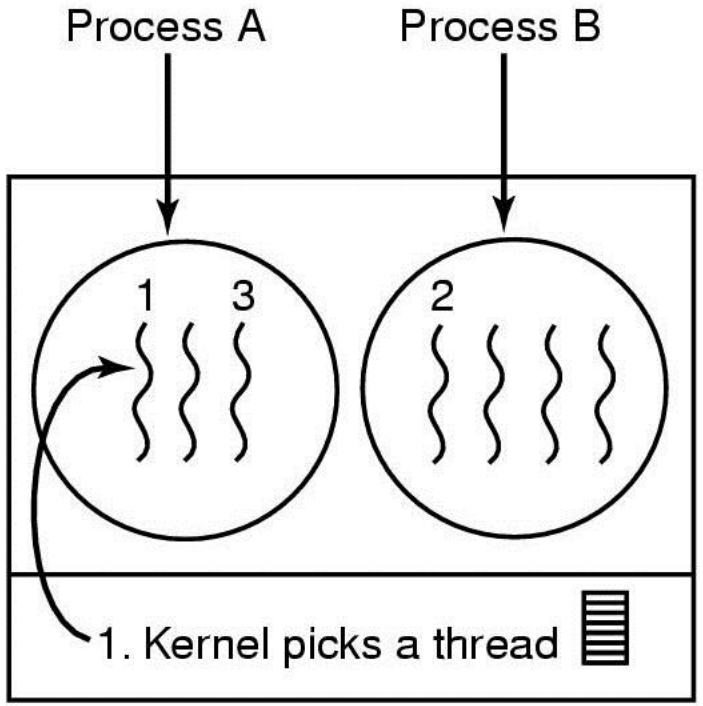
- Separate what is allowed to be done with how it is done
  - a process knows which of its children are the most important
  - E.g. database management system process
- Scheduling algorithm parameterized
  - mechanism in the kernel
- Parameters filled in by user processes
  - policy set by a user process

Thread Scheduling (1)



- Possible scheduling of user-level threads
  - 50-msec process quantum
  - threads run 5 msec/CPU burst
- High performance

Thread Scheduling (2)



- Possible scheduling of kernel-level threads
  - 50-msec process quantum
  - threads run 5 msec/CPU burst
- A thread block on I/O doesn't suspend the entire process.