Operating Systems

Jinghui Zhong （钟竞辉）
Office： B3-515
Email： jinghuizhong@scut.edu.cn

Process Behavior

- Nearly all processes alternate bursts of computing with I/O requests.
- CPU-bound: Spend most of the time computing.
- IO-bound: Spend most of the time waiting for I/O.

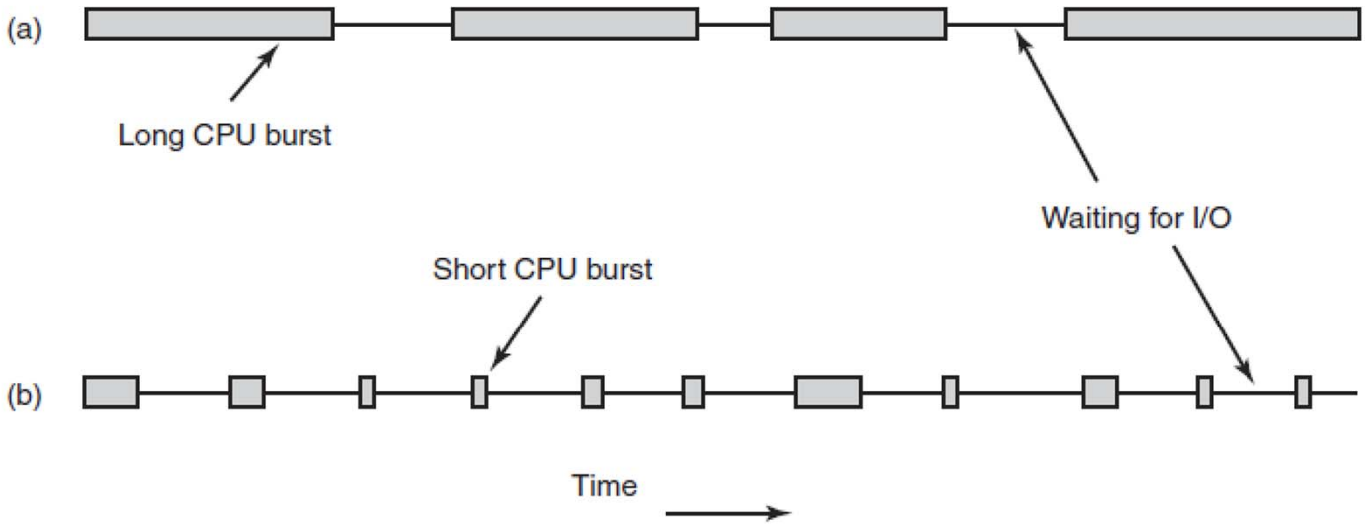


Figure 2-39. Bursts of CPU usage alternate with periods of waiting for I/O. (a) A CPU-bound process. (b) An I/O-bound process.

Process Scheduling

- **Scheduler:** A part of the operating system that decides which process is to be run next.
- **Scheduling Algorithm:** a policy used by the scheduler to make that decision.
- To make sure that no process runs too long, a clock is used to cause a periodic interrupt (usually around 50-60 Hz); that is, about every 20 msec.
- **Preemptive Scheduling:** allows processes that are runnable to be temporarily suspended so that other processes can have a chance to use the CPU.

When to Schedule

- When a new process is created;
- When a process exist;
- When a process blocks on I/O;
- When an I/O interrupt occurs (e.g., clock interrupt).

Scheduling Algorithm Goals

1. **Fairness** - each process gets its fair share of time with the CPU.
2. **Efficiency** - keep the CPU busy doing productive work.
3. **Response Time** - minimize the response time for interactive users.
4. **Turnaround Time** - minimize the average time from a batch job being submitted until it is completed.
5. **Throughput** - maximize the number of jobs processed per hour.

First-Come, First-Served (FCFS) Scheduling

Process	Burst Time
P_1	24
P_2	3
P_3	3

- Suppose that the processes arrive in the order: P_1 , P_2 , P_3
- Waiting time for P_1, P_2, and P_3 = 0， 24， 27
- Average waiting time = (0+24+27)/3 = 17

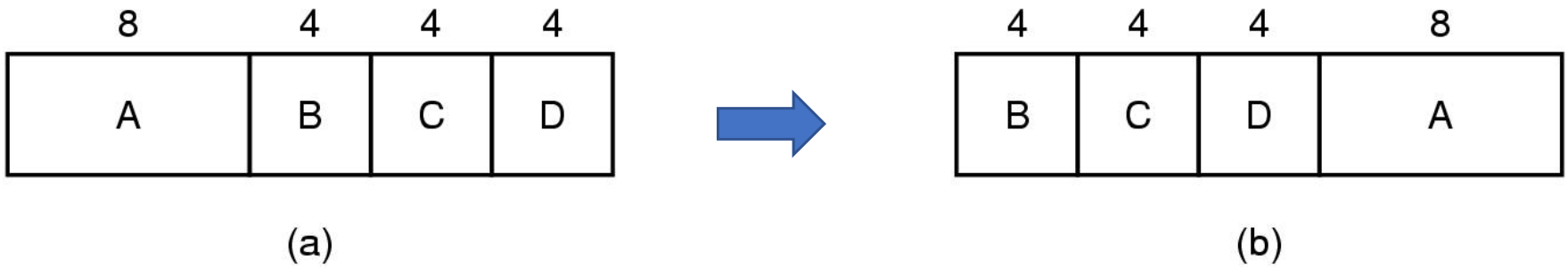
FCFS Scheduling

Suppose that the processes arrive in the order P_2 , P_3 , P_1 .

- Waiting time: $P_1 = 6$; $P_2 = 0$; $P_3 = 3$
- Average waiting time: $(6 + 0 + 3)/3 = 3$
- Much better than previous case.

Shortest-Job-First (SJF) Scheduling

- Associate with each process the length of its next CPU burst. Use these lengths to schedule the process with the shortest time.



An example of shortest job first scheduling

- Drawback?

The real difficulty with the SJF algorithm is knowing the length of the next CPU request.

Shortest-Job-First (SJR) Scheduling

- Two schemes:
  - nonpreemptive – once CPU given to the process it cannot be preempted until it completes its CPU burst.
  - preemptive – if a new process arrives with CPU burst length less than remaining time of current executing process, preempt. This scheme is know as the Shortest-Remaining-Time-First (SRTF).
- SJF is optimal – gives minimum average waiting time for a given set of processes.

Example of Non-Preemptive SJF

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

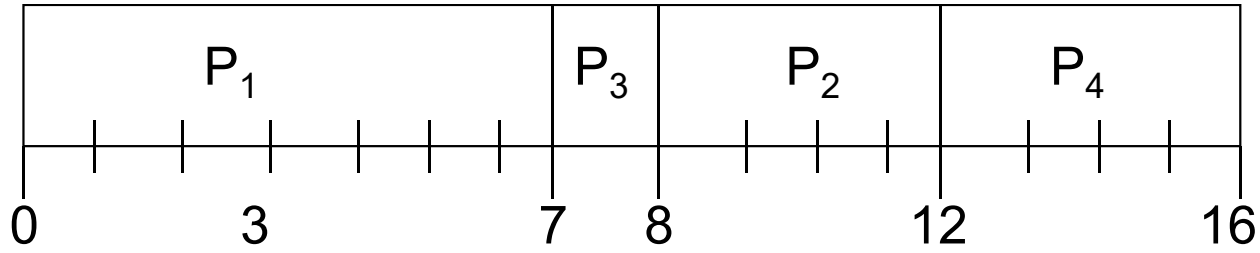
- SJF (non-preemptive)

Average waiting time = ?

Example of Non-Preemptive SJF

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (non-preemptive)



- Average waiting time = (0 + 6 + 3 + 7)/4 = 4

Example of Preemptive SJF

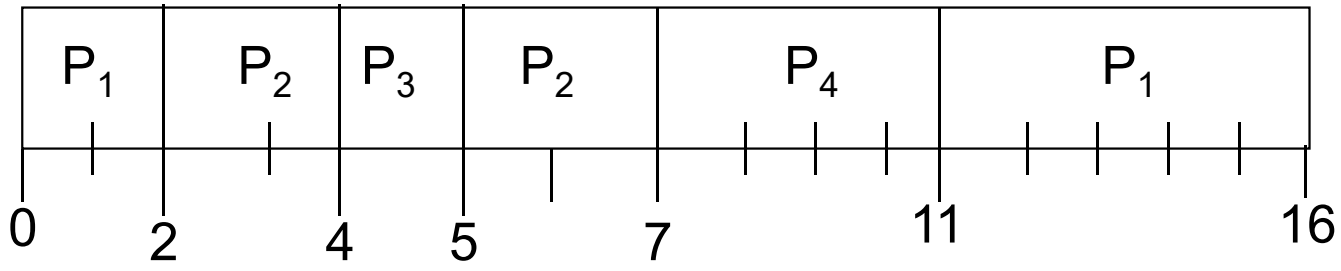
Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (preemptive)

Example of Preemptive SJF

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (preemptive)



- Average waiting time = (9 + 1 + 0 +2)/4 = 3

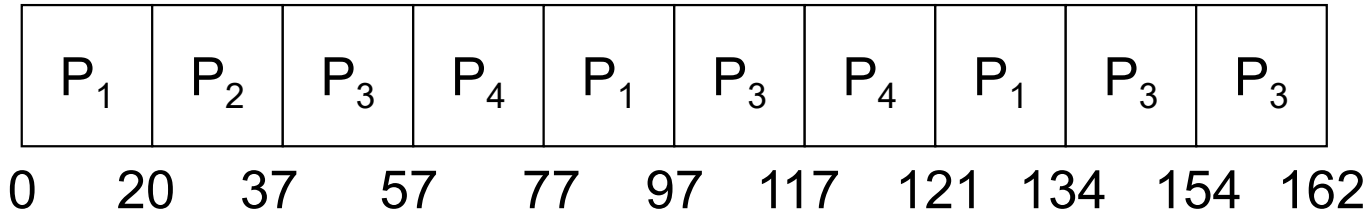
Round Robin (RR) Scheduling

- Each process gets a small unit of CPU time (*time quantum*), usually 10-100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.
- If there are n processes in the ready queue and the time quantum is q, then each process gets $1/n$ of the CPU time in chunks of at most q time units at once. No process waits more than $(n-1)q$ time units.
- Performance
  - q large $\Rightarrow$ FIFO
  - q small $\Rightarrow$ q must be large with respect to context switch, otherwise overhead is too high.

Example of RR with Time Quantum = 20

Process	Burst Time
P_1	53
P_2	17
P_3	68
P_4	24

- The Gantt chart is:



- Typically, higher average turnaround than SJF, but better *response*.

Round Robin (RR) Scheduling

Explain how time quantum value and context switching time affect each other, in a round-robin scheduling algorithm.

Measurements of a certain system have shown that the average process runs for a time T before blocking on I/O. A process switch requires a time S, which is effectively wasted (overhead). For round-robin scheduling with quantum Q, give a formula for the CPU efficiency for each of the following:

(a) $Q = \infty$
(b) $Q > T$
(c) $S < Q < T$
(d) $Q = S$
(e) Q nearly 0

Priority Scheduling

- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer ≡ highest priority).
  - Preemptive
  - nonpreemptive
- SJF is a priority scheduling where priority is the predicted next CPU burst time.
- Problem: Starvation – low priority processes may never execute.
- Solution: Aging – as time progresses increase the priority of the process.

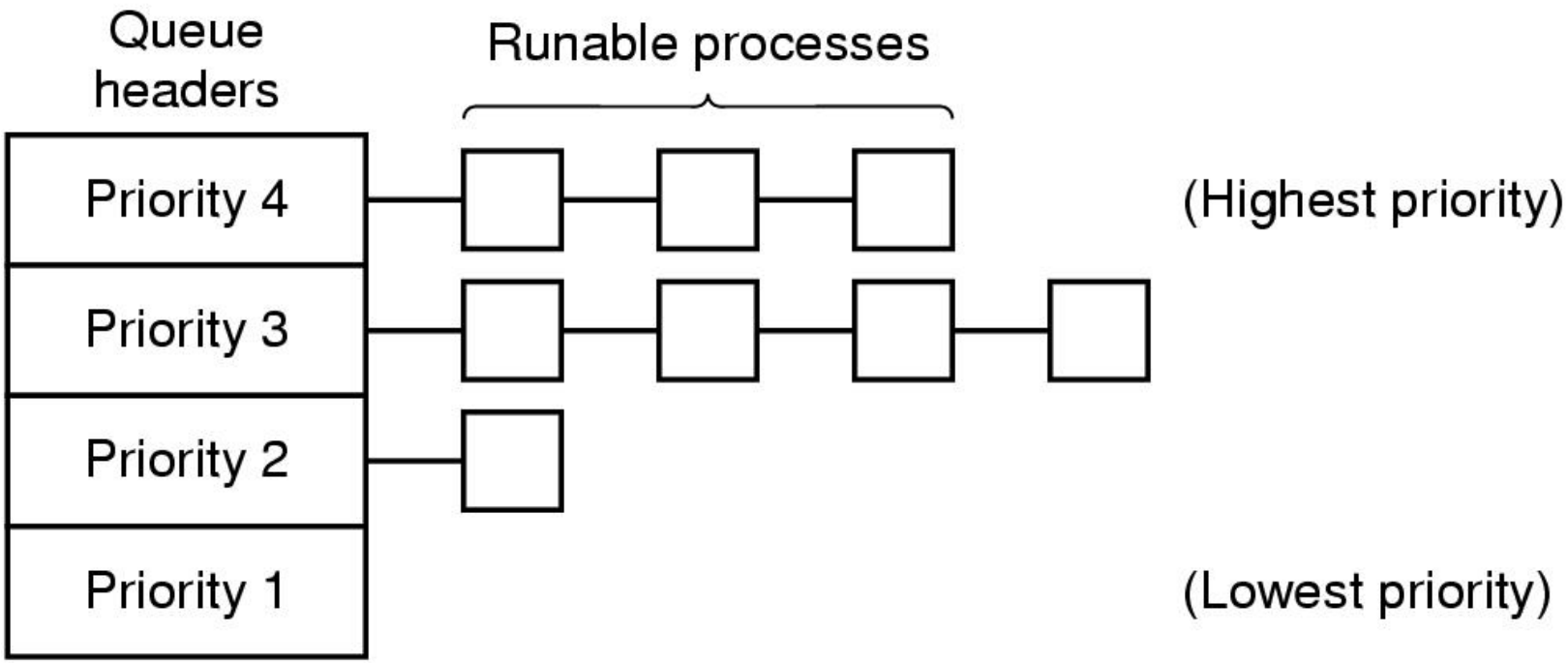
Priority Scheduling

Five batch jobs. *A* through *E*, arrive at a computer center at almost the same time. They have estimated running times of 10, 6, 2, 4, and 8 minutes. Their (externally determined) priorities are 3, 5, 2, 1, and 4, respectively, with 5 being the highest priority. For each of the following scheduling algorithms, determine the mean process turnaround time. Ignore process switching overhead.

(a) Round robin.
(b) Priority scheduling.
(c) First-come, first-served (run in order 10, 6, 2, 4, 8).
(d) Shortest job first.

For (a), assume that the system is multiprogrammed, and that each job gets its fair share of the CPU. For (b) through (d), assume that only one job at a time runs, until it finishes. All jobs are completely CPU bound.

Scheduling in Interactive Systems



A scheduling algorithm with four priority classes

More Scheduling

- Shortest Process Next
  - ✓SJF can be used in an interactive environment by estimating the runtime based on past behavior. **Aging** is a method used to estimate runtime by taking a weighted average of the current runtime and the previous estimate.
  - ✓Example: Let a = estimate weight, then the current estimate is: $a * T_0 + (1-a) * T_1$

    where T_0 is the previous estimate and T_1 is the current runtime.
- Guaranteed Scheduling
  - ✓Suppose $1/n$ of the CPU cycles.
  - ✓Compute ratio = actual CPU time consumed / CPU time entitled
  - ✓Run the process with the lowest ratio

More Scheduling

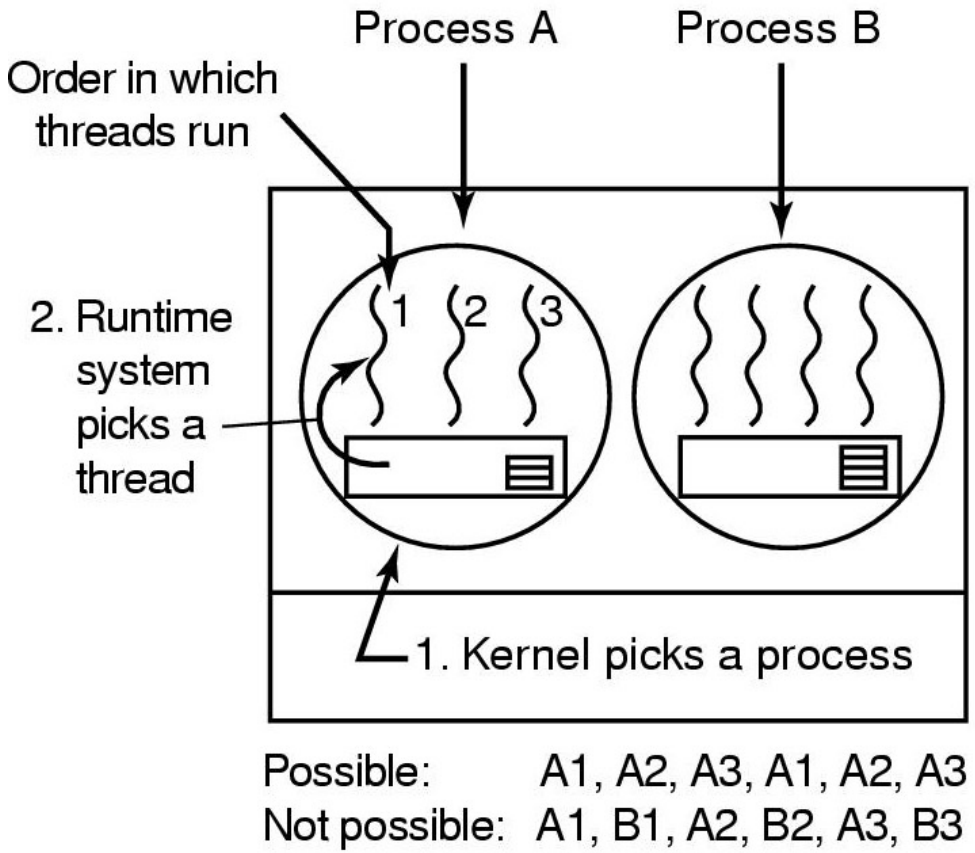
- Lottery Scheduling
  - Give processes lottery tickets for various system resources
  - When a scheduling decision is made, a lottery ticket is chosen, and the process holding that ticket gets the resource.
- Fair-Share Scheduling
  - Take into account how many processes a user owns.
  - Example: User 1 – A, B, C, D and Use 2 – E
  - Round-robin: ABCDEABCDE...
  - Fair-Share: if use 1 is entitled to the same CPU time as user 2

    AEBECEDEAEBECEDE….

Thread Scheduling

- The process scheduling algorithms can also be used in thread scheduling. In practice, round-robin and priority scheduling are commonly used.
- User-level and kernel-level threads
  - ✓A major difference between user-level threads and kernel-level threads is the performance.
  - ✓User-level threads can employ an application-specific thread scheduler.

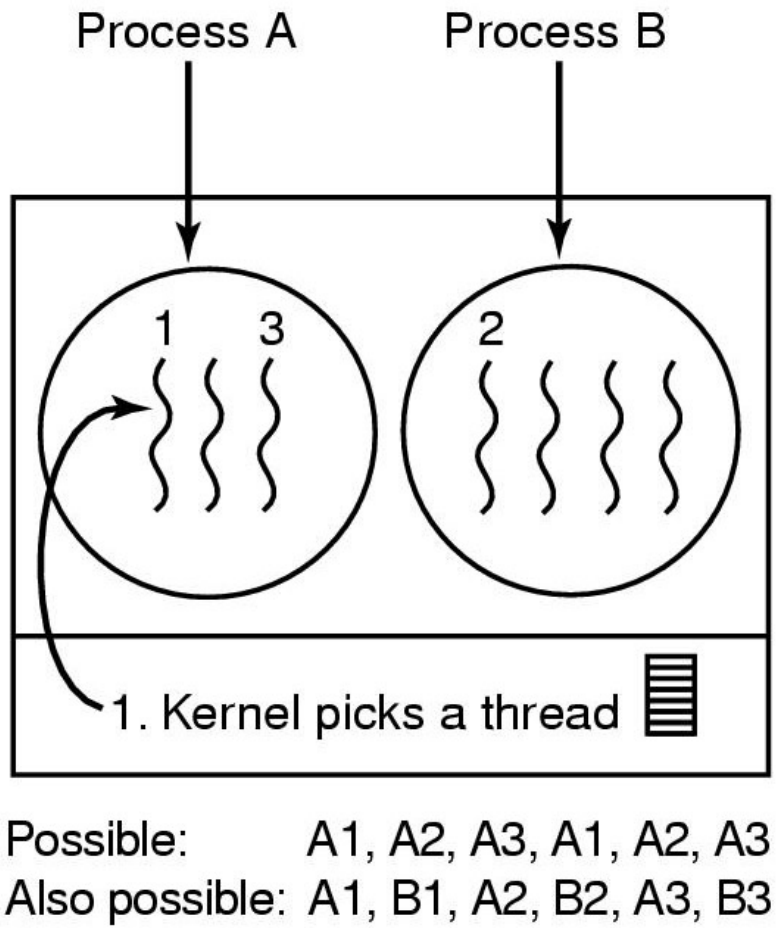
Thread Scheduling



Possible scheduling of user-level threads

- 50-msec process quantum
- Threads run 5 msec/CPU burst

Thread Scheduling



Possible scheduling of kernel-level threads

- 50-msec process quantum
- Threads run 5 msec/CPU burst

Semaphores [E.W. Dijkstra, 1965]

- **A SEMAPHORE, S, is a structure consisting of two parts:**
  - **(a) an integer counter, COUNT**
  - **(b) a queue of pids of blocked processes, Q**

- That is,

```
struct sem_struct {
  int count;
  queue Q;
} semaphore;

Semaphore S;
```

Semaphores [E.W. Dijkstra, 1965]

- There are two operations on semaphores, **UP** and **DOWN** (PV). These operations must be **executed atomically** (that is in mutual exclusion). Suppose that P is the process making the system call. The operations are defined as follows:

```
DOWN(S):

        if (S.count > 0)
                S.count = S.count - 1;
        else
                block(P); that is,
              (a) enqueue the pid of P in S.Q,
              (b) block process P (remove the pid from the ready queue)
              (c) pass control to the scheduler.
```

Semaphores [E.W. Dijkstra, 1965]

```
UP(S):

    if (S.Q is nonempty)
        wakeup(P) for some process P in S.Q; that is,
            (a) remove a pid from S.Q (the pid of P),
            (b) put the pid in the ready queue, and
            (c) pass control to the scheduler.
    else
        S.count = S.count + 1;
```

An Example

Semaphore a = 2;

Process A: down(a); sleep(a long time); up(a);

Process B: down(a);

Process C: down(a); up(a);

Process D: down(a); sleep(a long time); up(a);

Coming order: A→ B→C→D

What will happen?

Check points

1. What is IO-bound process?
2. When to schedule processes?
3. What are the goals of Scheduling algorithms?
4. What is the drawback of the SJF algorithm?
5. What is RR scheduling?