Theory of Operating Systems

Week 3: Process Scheduling Algorithms

Faye (Chi Zhang)

CUNY Hunter

February 11, 2026

Today's Outline

Why Do We Need Scheduling?

The Fundamental Problem:

- Many processes want to run
- Limited number of CPUs (often just one per core)
- Who gets the CPU? For how long?

The Scheduler's Job:

- Decide which process runs next
- Decide how long it runs
- Balance competing goals (fairness, efficiency, etc.)

When Does Scheduling Occur?

Scheduling decisions happen when:

1. Process switches from Running to Waiting (I/O)
2. Process switches from Running to Ready (preemption)
3. Process switches from Waiting to Ready (I/O complete)
4. Process terminates
5. New process is created

Non-preemptive vs Preemptive

- **Non-preemptive**: Process runs until it blocks or exits
- **Preemptive**: OS can interrupt running process (timer)

The Dispatcher

Dispatcher: Module that gives CPU control to selected process

- Switching context
- Switching to user mode
- Jumping to proper location in program

Dispatch Latency: Time to stop one process and start another

- Should be as fast as possible
- Typically 1-10 microseconds on modern systems

Scheduling Metrics

How do we evaluate a scheduling algorithm?

Metric	**Definition**
Turnaround Time	Time from submission to completion $T_{turnaround} = T_{completion} - T_{arrival}$
Response Time	Time from submission to first response $T_{response} = T_{first\_run} - T_{arrival}$
Throughput	Number of processes completed per time unit
CPU Utilization	Percentage of time CPU is busy
Waiting Time	Time spent in ready queue

Turnaround Time

Definition: Total time from arrival to completion

$$T_{turnaround} = T_{completion} - T_{arrival}$$

Includes:

- Waiting time in ready queue
- Execution time on CPU
- Time waiting for I/O

Important for: Batch systems, background jobs

Goal

Minimize average turnaround time across all processes

Response Time

Definition: Time from arrival to first execution

$$T_{response} = T_{first\_run} - T_{arrival}$$

Why it matters:

- Users perceive system as “snappy” or “slow”
- Interactive applications need quick feedback
- Even if total work takes time, starting quickly matters

Important for: Interactive systems, user-facing applications

Trade-off

Optimizing for response time often hurts turnaround time!

Competing Goals

No scheduler can optimize everything:

Maximize:

- CPU utilization
- Throughput
- Fairness

Minimize:

- Turnaround time
- Response time
- Waiting time

Key Insight

Different workloads need different schedulers:

- Batch systems: Optimize throughput
- Interactive systems: Optimize response time
- Real-time systems: Meet deadlines

First-Come First-Served (FCFS)

Algorithm:

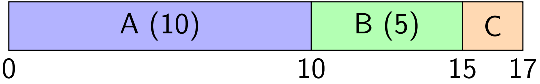
- Run processes in the order they arrive
- Non-preemptive: run until complete or blocked
- Simple FIFO queue

Example: Three jobs arrive at time 0

Process	Arrival	Burst Time
A	0	10
B	0	5
C	0	2

FCFS Example: Order A, B, C

Gantt Chart:

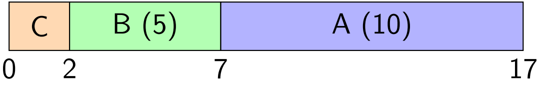


Turnaround Times:

- A: 10 - 0 = 10
- B: 15 - 0 = 15
- C: 17 - 0 = 17
- **Average**: $(10 + 15 + 17)/3 = 14$

FCFS: The Convoy Effect

What if C arrived first? (Order: C, B, A)



Turnaround Times:

- C: 2, B: 7, A: 17
- **Average**: $(2 + 7 + 17)/3 = 8.67$

Convoy Effect

Short processes stuck behind long process $\Rightarrow$ poor average turnaround time. Order matters dramatically!

FCFS: Pros and Cons

Advantages:

- Simple to implement
- No starvation
- Fair in arrival order sense
- Low overhead

Disadvantages:

- Convoy effect
- Poor average turnaround
- Not good for interactive
- Order-dependent performance

Use Case

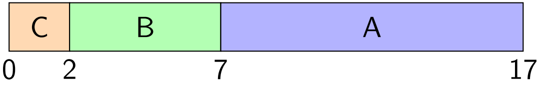
Batch processing systems where job order doesn't matter much

Shortest Job First (SJF)

Algorithm:

- Run the job with shortest burst time first
- Provably optimal for minimizing average turnaround time
- Non-preemptive version: once started, runs to completion

Same Example: Jobs A(10), B(5), C(2) arrive at time 0



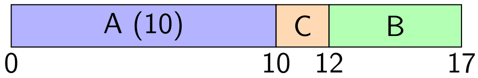
Average Turnaround: $(2 + 7 + 17)/3 = 8.67$ (optimal!)

SJF: The Arrival Problem

What if jobs arrive at different times?

Process	Arrival	Burst
A	0	10
B	1	5
C	2	2

Non-preemptive SJF: A starts at 0, runs to completion

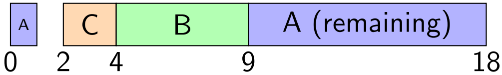


B and C arrive but must wait for A!

Shortest Time-to-Completion First (STCF)

Also called: Preemptive SJF (PSJF) **Algorithm**:

- When new job arrives, compare remaining times
- Preempt if new job is shorter
- Always run job with least remaining time



Optimal for average turnaround time!

SJF: The Knowledge Problem

Critical Issue: How do we know job length?

- **We don't!** Future is unpredictable
- User estimates are unreliable
- Historical data may not apply

Approaches:

- Exponential averaging: $\tau_{n+1} = \alpha \cdot t_n + (1 - \alpha) \cdot \tau_n$
  - t_n: actual length of last burst
  - τ_n: predicted length
  - α: weight (typically 0.5)
- Learn from past behavior

SJF: Starvation Problem

Scenario: Continuous stream of short jobs

- Short jobs keep arriving
- Long job keeps getting pushed back
- Long job may **never** run!

Starvation

A process waiting indefinitely because other processes always have higher priority.

Solution: Aging - gradually increase priority of waiting processes

10-Minute Break

We'll continue with Round Robin

Round Robin (RR)

Algorithm:

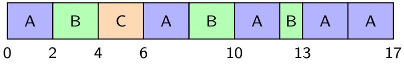
- Each process gets a small unit of CPU time (time quantum)
- After quantum expires, process is preempted
- Preempted process goes to end of ready queue
- Circular execution of ready processes

Key Parameter: Time quantum (time slice)

- Typically 10-100 milliseconds
- Critical design choice!

Round Robin Example

Jobs: A(10), B(5), C(2), all arrive at 0. **Quantum = 2**



Completion: C at 6, B at 13, A at 17
Response Times: A=0, B=2, C=4 ⇒ Avg = 2 (great!)
Turnaround: A=17, B=13, C=6 ⇒ Avg = 12 (worse than SJF)

Time Quantum: The Critical Trade-off

Quantum too small:

- Excellent response time
- Too many context switches
- High overhead
- CPU spends time switching, not working

Quantum too large:

- Fewer context switches
- Poor response time
- Degenerates to FCFS
- Users notice delays

Rule of Thumb

Quantum should be large enough that context switch overhead is $< 1\%$ of quantum. Typical: 10-100 ms.

Round Robin vs SJF

Aspect	SJF	RR
Turnaround time	Optimal	Worse
Response time	Can be bad	Good (bounded)
Starvation	Possible	No
Fairness	Unfair to long jobs	Fair
Knowledge needed	Job length	None
Preemptive	Optional	Yes

Key Insight

RR trades turnaround time for fairness and response time. If all jobs are same length, RR is worst for turnaround!

Priority Scheduling

Algorithm:

- Assign priority to each process
- Run highest priority process first
- Can be preemptive or non-preemptive

Priority can be:

- **External**: User/admin assigned
- **Internal**: Based on measurable attributes
  - Memory requirements
  - Time limits
  - I/O to CPU burst ratio

Priority Scheduling Example

Process	Burst	Priority	Arrival
A	10	3 (low)	0
B	1	1 (high)	0
C	2	4 (lowest)	0
D	1	2	0
E	5	2	0

Execution Order: B, D, E, A, C (by priority)

Priority Scheduling: Starvation and Aging

Problem: Low priority processes may starve

- High priority processes keep arriving
- Low priority never gets CPU

Solution: Aging

- Gradually increase priority of waiting processes
- Eventually, every process reaches high priority
- Guarantees all processes eventually run

Example

Increase priority by 1 every second of waiting. A priority-10 process will reach priority-1 after 9 seconds.

Multi-Level Feedback Queue (MLFQ)

Goal: Best of both worlds

- Good response time for interactive jobs (like RR)
- Good turnaround for batch jobs (like SJF)
- Without knowing job length in advance!

Key Idea: Learn from past behavior

- Jobs that use lots of CPU $\Rightarrow$ probably batch (lower priority)
- Jobs that quickly give up CPU $\Rightarrow$ probably interactive (higher priority)

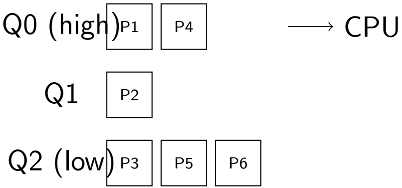
MLFQ Structure

Multiple Queues:

- Queue 0: Highest priority
- Queue 1: Lower priority
- ...
- Queue n: Lowest priority

Each queue may have different time quantum

- High priority: short quantum
- Low priority: long quantum



MLFQ: The Rules

1. If Priority(A) $>$ Priority(B), A runs
2. If Priority(A) $=$ Priority(B), run in Round Robin
3. New jobs enter at highest priority (top queue)
4. If a job uses its entire time slice, move down one queue
5. If a job gives up CPU before slice ends, stays in current queue

Intuition

- Interactive jobs: Give up CPU quickly $\Rightarrow$ stay at top
- Batch/CPU-bound jobs: Use full slice $\Rightarrow$ sink to bottom

MLFQ: Problems and Solutions

Problem 1: Gaming the system

- Job issues I/O just before quantum ends
- Stays at high priority unfairly
- **Solution**: Account for total CPU time, not per-slice

Problem 2: Starvation of long-running jobs

- Too many interactive jobs $\Rightarrow$ batch jobs starve
- **Solution**: Priority boost - periodically move all jobs to top queue

Problem 3: Changed behavior

- CPU-bound job becomes interactive
- Stuck at low priority
- **Solution**: Priority boost helps here too

MLFQ: Refined Rules

1. If Priority(A) > Priority(B), A runs
2. If Priority(A) = Priority(B), run in Round Robin
3. New jobs start at highest priority
4. Once a job uses its time allotment at a given level (regardless of how many times it gave up CPU), move down
5. After time period S, boost all jobs to top queue

Parameters to tune:

- Number of queues
- Time quantum per queue
- Boost period S
- Time allotment per level

Linux: Completely Fair Scheduler (CFS)

Goal: Fair CPU time for all processes

- Tracks "virtual runtime" for each process
- Always runs process with lowest virtual runtime
- Weighted by nice value (-20 to +19)
- Uses red-black tree for efficient selection

Nice Values:

- Lower nice = higher priority (gets more CPU)
- `nice -n 10 command` runs with lower priority
- Default is 0

Scheduling Algorithm Comparison

Algorithm	Preemptive	Starvation	Optimal	Complexity
FCFS	No	No	No	O(1)
SJF	No	Yes	Turnaround	O(n)
STCF	Yes	Yes	Turnaround	O(n)
Round Robin	Yes	No	No	O(1)
Priority	Both	Yes	No	O(n)
MLFQ	Yes	No*	No	O(1)
CFS	Yes	No	Fairness	O(log n)

*With priority boost

Key Takeaways

1. **Scheduling goals conflict**: Can't optimize everything
2. **FCFS**: Simple but convoy effect
3. **SJF**: Optimal turnaround but needs oracle
4. **Round Robin**: Fair but worse turnaround
5. **Priority**: Flexible but can starve
6. **MLFQ**: Learns from behavior, best of both worlds
7. **Quantum choice**: Critical trade-off

This Week's Tasks

- **Quiz 3**: Scheduling algorithms, Gantt charts
- **Reading**: Textbook Chapters 5-7
- **Assignment 1**: Implement FCFS, SJF, RR
  - Calculate turnaround and response times
  - Due: February 18
- **Next Week**: Inter-Process Communication (IPC)
  - Pipes, message queues, shared memory
  - Producer-consumer problem

Questions?