CPU Scheduling

CSE 2431: Introduction to Operating Systems
Reading: Chap. 6, [OSC]
Contents

• Why Scheduling?
• Basic Concepts of Scheduling
• Scheduling Criteria
• A Basic Scheduling Algorithm (FCFS)
• Scheduling Algorithms (SJF, RR, etc.)
• Thread Scheduling
Why Scheduling?

- Deciding which process/thread should occupy the resource (CPU, disk, etc.)

(CPU (horsepower))

I want to ride it

Whose turn is it?

Process 1  Process 2  Process 3
Contents

• Why Scheduling?
• Basic Concepts of Scheduling
• Scheduling Criteria
• A Basic Scheduling Algorithm (FCFS)
• Scheduling Algorithms (SJF, RR, etc.)
• Thread Scheduling
When to Schedule?

- ready queue
- CPU
- I/O
- I/O queue
- I/O request
- time slice expired
- child executes
- fork a child
- interrupt occurs
- wait for an interrupt
Contents

• Why Scheduling?
• Basic Concepts of Scheduling
• Scheduling Criteria
• Basic Scheduling Algorithm (FCFS)
• Scheduling Algorithms (SJF, RR, etc.)
• Thread Scheduling
Scheduling Objectives

• **Fairness** (nobody cries)
• **Priority** (ladies first)
• **Efficiency** (make best use of equipment)
• **Encourage good behavior** (good boy/girl)
• **Support heavy loads** (degrade gracefully)
• **Adapt to different environments** (interactive, real-time, multimedia)
Performance Criteria

- **Fairness**: No starvation
- **Efficiency** keep resources as busy as possible
- **Throughput**: # of processes completed in unit time
- **Turnaround Time** (also called *elapsed time*): Amount of time to complete a particular process from its beginning
- **Waiting Time**: Amount of time process has been waiting in ready queue
- **Response Time**: Amount of time from when a request was first submitted until first response is produced.
- **Policy Enforcement**: Enforcing that stated policy is carried out
- **Proportionality**: Meet users' expectations
- **Meeting Deadlines**: Avoid losing data
Different Systems, Different Foci

• For all
  – Fairness, policy enforcement, resource balance

• Batch Systems
  – Max throughput, min turnaround time, max CPU utilization

• Interactive Systems
  – Min Response time, best proportionality

• Real-Time Systems
  – predictability, meeting deadlines
Preemptive vs. Non-preemptive

- **Non-preemptive scheduling:**
  - The running process keeps the CPU until it *voluntarily* gives up the CPU
    - Process exits
    - Switches to waiting state
    - 1 and 4 only (no 3)

- **Preemptive scheduling:**
  - The running process can be interrupted and must release the CPU (be *forced* to give up CPU)
Process Behavior

• I/O-Bound
  – Does too much I/O to keep CPU busy

• CPU-Bound
  – Does too much computation to keep I/O busy

• Process Mix
  – Scheduling should load-balance between I/O bound and CPU-bound processes
  – Ideal would be to run all equipment at 100% utilization, but that would not necessarily be good for response time
Program Characteristics Considered in Scheduling

- Is it I/O bound?
- Is it CPU bound?
- Batch or interactive environment
- Urgency
- Priority
- Frequency of page faults
- Frequency of preemption
- How much execution time it has already received
- How much execution time it needs to complete
CPU Scheduler

• Proc 1: 14 time units
• Proc 2: 8 time units
• Proc 3: 8 time units
• Dispatcher
• Preemptive vs. non-preemptive
Dispatcher

• Gives the control of the CPU to the process, scheduled by the short-term scheduler.

• Functions:
  – Switching context
  – Switching to user mode
  – Jumping to the proper location in the user program.

• *Dispatch Latency*: time to stop a process and start another one.
  – Pure overhead
  – Needs to be fast
Single Processor Scheduling Algorithms

• Batch systems
  – First Come First Serve (FCFS)
  – Shortest Job First

• Interactive Systems
  – Round Robin
  – Priority Scheduling
  – Multi Queue & Multi-level Feedback
  – Shortest process time
  – Guaranteed Scheduling
  – Lottery Scheduling
  – Fair Sharing Scheduling
First Come First Serve (FCFS)

- Process that requests the CPU FIRST is allocated the CPU FIRST.
- Also called FIFO
- Preemptive or non-preemptive?
- Used in batch systems
- Real life analogy?
  - Buying tickets?
- Implementation
  - FIFO queues
    - A new process enters the tail of the queue
    - The schedule selects from the head of the queue.
- Performance Metric: **Average Waiting Time**.
- Given Parameters:
  - Burst Time (in ms), Arrival Time and Order
FCFS Example

<table>
<thead>
<tr>
<th>Process</th>
<th>Duration</th>
<th>Order</th>
<th>Arrival Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>24</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>P2</td>
<td>3</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>P3</td>
<td>4</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

The final schedule (Gantt chart):

P1 waiting time: 0
P2 waiting time: 24
P3 waiting time: 27

The average waiting time: 
\[(0+24+27)/3 = 17\]

What if P1 arrives at time 2
Problems with FCFS

• Non-preemptive
• Not optimal AWT
• Cannot utilize resources in parallel:
  – Assume 1 process CPU bounded and many I/O bounded processes
  – Result: *Convoy effect*, low CPU and I/O device utilization
  – Why?
Why Convoy Effects?

• Consider 100 I/O-bound processes and 1 CPU-bound job in the system.
• I/O-bound processes pass quickly through the ready queue and suspend themselves waiting for I/O.
• The CPU-bound process arrives at head of queue and executes the program until completion.
• I/O bound processes rejoin the ready queue and wait for the CPU-bound process releasing the CPU.
• **I/O devices idle until the CPU-bound process completes.**
• In general, a convoy effect happens when a set of processes need to use a resource for a short time, and one process holds the resource for a long time, blocking all of the other processes. Essentially, it causes poor utilization of resources in the system.
Shortest Job First (SJF)

- Schedule the job with the shortest duration time first
- Used in batch systems
- Two types:
  - Non-preemptive
  - Preemptive
- Requirement: the duration time needs to be known in advance
- **Optimal** if all jobs are available simultaneously (provable)
  - Gives the best possible AWT (average waiting time)
Non-preemptive SJF: Example

<table>
<thead>
<tr>
<th>Process</th>
<th>Duration</th>
<th>Order</th>
<th>Arrival Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>6</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>P2</td>
<td>8</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>P3</td>
<td>7</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>P4</td>
<td>3</td>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>

Do it yourself

P4 waiting time: 0
P1 waiting time: 3
P3 waiting time: 9
P2 waiting time: 16

The total time is: 24
The average waiting time (AWT): 
\[(0+3+9+16)/4 = 7\]
## Comparing to FCFS

<table>
<thead>
<tr>
<th>Process</th>
<th>Duration</th>
<th>Order</th>
<th>Arrival Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>6</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>P2</td>
<td>8</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>P3</td>
<td>7</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>P4</td>
<td>3</td>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>

Do it yourself

P1 waiting time: 0
P2 waiting time: 6
P3 waiting time: 14
P4 waiting time: 21

The total time is the same.
The average waiting time (AWT):

\[
\frac{0 + 6 + 14 + 21}{4} = 10.25
\]

(compared to 7)
SJF Is Not Always Optimal

Is SJF optimal if all the jobs are not available simultaneously?

<table>
<thead>
<tr>
<th>Process</th>
<th>Duration</th>
<th>Order</th>
<th>Arrival Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>10</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>P2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

P1 waiting time: 0
P2 waiting time: 8

The average waiting time (AWT): 
\[
\frac{(0+8)}{2} = 4
\]
Preemptive SJF

• Also called **Shortest Remaining Time First**
  – Schedule the job with the shortest remaining time required to complete

• Requirement: **the duration time needs to be known in advance**
**Preemptive SJF: Same Example**

<table>
<thead>
<tr>
<th>Process</th>
<th>Duration</th>
<th>Order</th>
<th>Arrival Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>10</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>P2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

P1 waiting time: 4-2 = 2  
P2 waiting time: 0  
The average waiting time (AWT):  
\[(0+2)/2 = 1\]
A Problem with SJF

• Starvation
  – In some scenarios, a job may wait forever
  – Example: SJF
    • Process A with duration time of 1 hour arrives at time 0
    • But every 1 minute, a shorter process with duration time of 2 minutes arrive
    • Result of SJF: process A never gets to run

• What’s the difference between starvation and a deadlock?
Interactive Scheduling Algorithms

• Usually preemptive
  – Time is **sliced** into quantum (time intervals)
  – Scheduling decision is also made at the beginning of each quantum

• Performance Criteria
  – Min Response time
  – best proportionality

• Representative algorithms:
  – Priority-based
  – Round-robin
  – Multi Queue & Multi-level Feedback
  – Shortest process time
  – Guaranteed Scheduling
  – Lottery Scheduling
  – Fair Sharing Scheduling
Priority Scheduling

• Each job is assigned a priority.
• FCFS within each priority level.
• Select highest priority job over lower ones.
• Rationale: higher priority jobs are more mission-critical
  – Example: DVD movie player vs. send email

• Problems:
  – May not give the best AWT
  – Starvation
Set Priority

• Two approaches
  – Static (for system with well known and regular application behaviors)
  – Dynamic (otherwise)

• Priority may be based on:
  – Cost to user.
  – Importance of user.
  – Aging
  – Percentage of CPU time used in last X hours.
# Priority Scheduling: Example

<table>
<thead>
<tr>
<th>Process</th>
<th>Duration</th>
<th>Priority</th>
<th>Arrival Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>6</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>P2</td>
<td>8</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>P3</td>
<td>7</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>P4</td>
<td>3</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Process</th>
<th>Duration</th>
<th>Priority</th>
<th>Arrival Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P2</td>
<td>8</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>P3</td>
<td>7</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>P4</td>
<td>3</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

**Do it yourself**

P2 (8) P4 (3) P3 (7) P1 (6)

0 8 11 18 24

P2 waiting time: 0
P4 waiting time: 8
P3 waiting time: 11
P1 waiting time: 18

The average waiting time (AWT): 
\[(0+8+11+18)/4 = 9.25\]  
(worse than SJF)
Round-Robin (RR)

• One of the oldest, simple, commonly used scheduling algorithms

• Select process/thread from ready queue in a round-robin fashion (take turns)

• Problems:
  – Do not consider priority
  – More context switch overhead
# Round-Robin: Example

<table>
<thead>
<tr>
<th>Process</th>
<th>Duration</th>
<th>Order</th>
<th>Arrival Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>P2</td>
<td>4</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>P3</td>
<td>3</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>

Suppose time quantum is: 1 unit, P1, P2 & P3 never block

**Do it yourself**

- **P1 waiting time:** 4
- **P2 waiting time:** 6
- **P3 waiting time:** 6

The average waiting time (AWT): 

\[
\frac{4+6+6}{3} = 5.33
\]
Time Quantum

• Time slice too large
  – FIFO behavior
  – Poor response time
• Time slice too small
  – Too many context switches (overheads)
  – Inefficient CPU utilization
• Heuristic: (Eliminating preemption)
  – 70–80% of processes block within time-slice
• Typical time slice
  – 10 to 100 ms
• Time spent in system depends on size of job
Multi-Queue Scheduling

- Hybrid between priority and round-robin
- Processes assigned to one queue
- Scheduling between queues
  - Fixed Priorities
  - Dynamic priorities based on CPU % spent on queue
- Example
  - System processes
  - Interactive programs
  - Background processes
- Address the starvation problem
Multi-Queue Scheduling: Example

- System processes
- Interactive processes
- Interactive editing processes
- Batch processes
- Student processes
Multi-Processor Scheduling: Load Sharing

- Decides
  - Which process to run?
  - How long does it run?
  - Where to run it?

I want to ride it

Process 1

Process 2

Process n

(CPU (horsepower))
Multi-Processor Scheduling Choices

• Self-Scheduled
  – Each CPU dispatches a job from the ready queue

• Master-Slave
  – One CPU schedules the other CPUs

• Asymmetric
  – One CPU runs the kernel and the others run the user applications.
  – One CPU handles network and the others handle applications
Gang Scheduling for Multi-Processors

• A collection of processes belonging to one job
• All the processes are running at the same time
  – If one process is preempted, all the processes of
    the gang are preempted. Why?
• Helps to eliminate the time a process spends
  on waiting for other processes in its parallel
  computation.
Priority Inversion and Inheritance

• Priority inversion problem
  – When a higher priority process needs to read or modify kernel data that are currently being locked by a lower priority process:
  – The higher priority process must wait!
  – But the lower priority cannot proceed due to scheduling.

• Solution: Priority inheritance
  – When a lower-priority process accesses a resource, it inherits high priority until it is done with the resource in question. Then its priority reverts to its natural value.
User-Level Thread Scheduling

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

Possible: A1, A2, A3, A1, A2, A3
Not possible: A1, B1, A2, B2, A3, B3
Kernel-Level Thread Scheduling

Possible Scheduling
- 50-msec process quantum
- Threads run 5 msec/CPU burst
Summary (I)

• Why Scheduling?
• Basic Concepts of Scheduling
• Scheduling Criteria
• Basic Scheduling Algorithm (FCFS)
• Convoy Effects
Summary (II)

• Scheduling algorithms
  – Shortest job first (SJF)
  – Round-robin (RR)
  – Priority scheduling
  – Multi Queue
  – Multi-Processor Scheduling

• Priority Inversion

• Thread Scheduling