Processor Scheduling

Background

• The previous lecture introduced the basics of concurrency
  – Processes and threads
    – Definition, representation, management
• We now understand how a programmer can spawn concurrent computations
• The OS now needs to partition one of the central resources, the CPU, between these concurrent tasks

Scheduling

• The scheduler is the manager of the CPU resource
• It makes allocation decisions – it chooses to run certain processes over others from the ready queue
  – Zero threads: just loop in the idle loop
  – One thread: just execute that thread
  – More than one thread: now the scheduler has to make a resource allocation decision
• The scheduling algorithm determines how jobs are scheduled

Scheduling

• Threads alternate between performing I/O and performing computation
• In general, the scheduler runs:
  – when a process switches from running to waiting
  – when a process is created or terminated
  – when an interrupt occurs
• In a non-preemptive system, the scheduler waits for a running process to explicitly block, terminate or yield
• In a preemptive system, the scheduler can interrupt a process that is running.
Process States

Processes are **I/O-bound** when they spend most of their time in the waiting state.
Processes are **CPU-bound** when they spend their time in the ready and running states.

Time spent in each entry into the running state is called a **CPU burst**.

Scheduling Evaluation Metrics

- There are many possible quantitative criteria for evaluating a scheduling algorithm:
  - CPU utilization: percentage of time the CPU is not idle
  - Throughput: completed processes per time unit
  - Turnaround time: submission to completion
  - Waiting time: time spent on the ready queue
  - Response time: response latency
  - Predictability: variance in any of these measures

- The right metric depends on the context.

Scheduling Algorithms FCFS

- **First-come First-served** (FCFS) (FIFO)
  - Jobs are scheduled in order of arrival
  - Non-preemptive

- **Problem:**
  - Average waiting time can be large if small jobs wait behind long ones

  ![FCFS Diagram](image)

  - May lead to poor overlap of I/O and CPU and convoy effects.

Scheduling Algorithms LIFO

- **Last-In First-out** (LIFO)
  - Newly arrived jobs are placed at the head of the ready queue
  - Improves response time for newly created threads

- **Problem:**
  - May lead to starvation – early processes may never get the CPU
Problem

- You work as a short-order cook
  - A short order cook has to cook food for customers as they come in and specify which dish they want
  - Each dish takes a different amount of time to prepare
- You want to minimize the average amount of time the customers wait for their food
- What strategy would you use?
  - Note: most restaurants use FCFS.

Scheduling Algorithms SJF

- **Shortest Job First (SJF)**
  - Choose the job with the shortest next CPU burst
  - Provably optimal for minimizing average waiting time

<table>
<thead>
<tr>
<th></th>
<th>P_1</th>
<th>P_3</th>
<th>P_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>24</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Problem:
  - Impossible to know the length of the next CPU burst

Shortest Job First Prediction

- Approximate the duration of the next CPU-burst from the durations of the previous bursts
  - The past can be a good predictor of the future
- No need to remember entire past history
- Use exponential average:
  \[
  \hat{t}_{n+1} = \hat{t}_n + (1 - \hat{\theta}) \hat{t}_n
  \]
  where \(0 \leq \hat{\theta} \leq 1\)
  \(\hat{\theta}\) determines the weight placed on past behavior

Scheduling Algorithms SRTF

- SJF can be either preemptive or non-preemptive
  - The distinction occurs when a new, short job arrives while the currently process has a long time left to execute
- Preemptive SJF is called **shortest remaining time first**
Priority Scheduling

- **Priority Scheduling**
  - Choose next job based on priority
  - For SJF, priority = expected CPU burst
  - Can be either preemptive or non-preemptive
- **Problem:**
  - Starvation: jobs can wait indefinitely
- **Solution to starvation**
  - Age processes: increase priority as a function of waiting time

Round Robin

- **Round Robin (RR)**
  - Often used for timesharing
  - Ready queue is treated as a circular queue (FIFO)
  - Each process is given a time slice called a quantum
  - It is run for the quantum or until it blocks
  - RR allocates the CPU uniformly (fairly) across all participants.
  - If average queue length is n, each participant gets 1/n
  - As the time quantum grows, RR becomes FCFS
  - Smaller quanta are generally desirable, because they improve response time
- **Problem:**
  - Context switch overhead of frequent context switch

RR with Time Quantum = 20

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P₁</td>
<td>53</td>
</tr>
<tr>
<td>P₂</td>
<td>17</td>
</tr>
<tr>
<td>P₃</td>
<td>68</td>
</tr>
<tr>
<td>P₄</td>
<td>24</td>
</tr>
</tbody>
</table>

The Gantt chart is:

```
  P₁  P₂  P₃  P₄  P₁  P₂  P₃  P₄  P₁  P₂  P₃  P₄
0   20  37  57  77  97 117 134 154 162
```

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

Scheduling Algorithms

- **Multi-level Queue Scheduling**
  - Implement multiple ready queues based on job “type”
  - interactive processes
  - CPU-bound processes
  - batch jobs
  - system processes
  - student programs
- **Different queues may be scheduled using different algorithms**
- **Intra-queue CPU allocation can be either strict or proportional**
- **Problem: Classifying jobs into queues is difficult**
  - A process may have CPU-bound phases as well as interactive ones
### Multilevel Queue Scheduling

- **Highest priority**
  - System Processes
  - Interactive Processes
  - Batch Processes
  - Student Processes

- **Lowest priority**

### Scheduling Algorithms

- **Multi-level Feedback Queues**
  - Implement multiple ready queues
  - Different queues may be scheduled using different algorithms
  - Just like multilevel queue scheduling, but assignments are not static
  - Jobs move from queue to queue based on feedback
  - Feedback = The behavior of the job, e.g. does it require the full quantum for computation, or does it perform frequent I/O?

- **Very general algorithm**
- **Need to select parameters for:**
  - Number of queues
  - Scheduling algorithm within each queue
  - When to upgrade and downgrade a job

### Real-time Scheduling

- **Real-time processes have timing constraints**
  - Expressed as deadlines or rate requirements

- **Common RT scheduling policies**
  - **Rate monotonic**
    - Simple, just one scalar priority related to the periodicity of the job
    - Priority = 1/rate
    - Static
  - **Earliest deadline first (EDF)**
    - Dynamic but more complex
    - Priority = deadline

- **Both schemes require admission control to provide guarantees**
Scheduling on a Multiprocessor

- Two alternatives based on the total number of queues:
  - Each processor has its own separate queue
  - All processors share a common ready-queue, and autonomously pick threads to execute from this common queue whenever they are idle (work stealing)

- Scheduling locally on any single processor is mostly the same as scheduling on a uniprocessor

- Issues:
  - Want to keep threads local to a processor (processor affinity)
  - Want to keep related threads together (gang-scheduling)