CPU Scheduling

- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms
  - FCFS, SJF, RR
- Exponential Averaging
- Multi-level Queue Scheduling
- Performance Evaluation
- Unix Scheduler

Scheduling Terminology

- **Context switch (dispatch)** - the actions taken to make the selected process run on the CPU - restore CPU state to that of the selected process when it last ran (all stored in the PCB - registers, cc, PC)
- **Degree of multiprogramming** - the number of active processes (i.e. in the ready and running states.)
- **Swapping** - the actions taken to remove a process from memory to reduce the degree of multiprogramming.

Scheduling Terminology

- **Long term scheduling (job scheduling)** - decision as to which jobs to select for the pool of active processes.
- **Medium term scheduling** - decision to move some of the processes out of main memory.
- **Short term scheduling (process scheduling)** - decision as to which process to allocate to the CPU.

Scheduling/Process State Transition

Objectives for Schedulers

- **GOOD PERFORMANCE FOR THE USER** (based on some metric)
  - **high throughput** - number of jobs completed per time unit
  - **low turnaround time** - elapsed time from submission to completion, averaged over a set of jobs.
  - **low weighted avg. turnaround time** - same as above but certain jobs are given more weight.
  - **low response time** - elapsed time from submission to some response (timesharing).
Objectives for Schedulers (cont’d)

- GOOD PERFORMANCE FOR THE SYSTEM (based on some metric)
  - CPU Utilization - percent of time the CPU is engaged in useful work over some time frame.
- FAIRNESS
  - No process should get postponed indefinitely (priorities)
- STABILITY
  - predictability
  - graceful degradation

CPU burst and I/O Burst

- Maximum CPU utilization obtained with multiprogramming
- CPU–I/O Burst Cycle – Process execution consists of a cycle of CPU execution and I/O wait.
- CPU burst distribution

Alternating Sequence of CPU And I/O Bursts

Histogram of CPU-burst Times

Types of Schedulers

- PREEMPTIVE versus NON-PREEMPTIVE
  - Preemptive - can interrupt a process during its CPU burst and schedule a different process.
  - Non-preemptive - each process completes its CPU burst before next process is scheduled. (Can’t be used for timesharing.)
- PRIORITIES (STATIC vs. DYNAMIC)
  - Static - fixed for the life of the process; dynamic - changes over the life of the process.
  - Determined by characteristics of the process.
  - Aging - dynamically changing the priority over time (to give jobs that have waiting a long time higher priority).

CPU (Short Term) Scheduler

- Selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them.
- CPU scheduling decides which process to switch from ready state to running state.
CPU Scheduling Algorithms

- First Come First Served (FCFS)
- Shortest Job First (SJF)
- Round Robin (RR)
- Multi-level Queues
- Multi-level with Feedback
- (There are many others)

First-Come, First-Served (FCFS) Scheduling

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>24</td>
</tr>
<tr>
<td>$P_2$</td>
<td>3</td>
</tr>
<tr>
<td>$P_3$</td>
<td>3</td>
</tr>
</tbody>
</table>

Suppose processes arrive in the order $P_1, P_2, P_3$
The Gantt chart for the schedule is:

- Waiting time for $P_1 = 0$; $P_2 = 24$; $P_3 = 27$
- Average waiting time: $(0 + 24 + 27)/3 = 17$

FCFS Scheduling (Cont.)

Suppose that the processes arrive in the order $P_2, P_3, P_1$.
The Gantt chart for the schedule is:

- Waiting time for $P_1 = 6$; $P_2 = 0$; $P_3 = 3$
- Average waiting time: $(6 + 0 + 3)/3 = 3$
- Much better than previous case.
- Avoided convoy effect short process behind long process

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.
- Two schemes:
  - nonpreemptive – once CPU given to the process it cannot be preempted until 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 known as the Shortest-Remaining-Time-First (SRTF).
- SJF is optimal – gives minimum average waiting time for a given set of processes. It is an idealized algorithm because difficult to accurately know next CPU burst time.

Example of Non-Preemptive SJF

<table>
<thead>
<tr>
<th>Arrival Time</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>0.0</td>
</tr>
<tr>
<td>$P_2$</td>
<td>2.0</td>
</tr>
<tr>
<td>$P_3$</td>
<td>4.0</td>
</tr>
<tr>
<td>$P_4$</td>
<td>5.0</td>
</tr>
</tbody>
</table>

- SJF (non-preemptive)
- Average waiting time = $(0 + 6 + 3 + 7)/4 = 4$

Example of Preemptive SJF

<table>
<thead>
<tr>
<th>Arrival Time</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>0.0</td>
</tr>
<tr>
<td>$P_2$</td>
<td>2.0</td>
</tr>
<tr>
<td>$P_3$</td>
<td>4.0</td>
</tr>
<tr>
<td>$P_4$</td>
<td>5.0</td>
</tr>
</tbody>
</table>

- SJF (preemptive)
- Average waiting time = $(9 + 1 + 0 + 2)/4 = 3$
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).
- 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.

Determining Length of Next CPU Burst

- Can only estimate the length.
- Can be done by using the length of previous CPU bursts, using **exponential averaging**.
  1. \( r_n \) = actual length of \( n \)-th CPU burst
  2. \( r_{n+1} \) = predicted value for the next CPU burst
  3. \( 0 < \alpha < 1 \)
  4. Define:
     \[
     r_{n+1} = \alpha r_n + (1-\alpha) t_n
     \]

Predicting Length of Next CPU Burst

- \( \alpha = 0 \)
  - \( r_{n+1} = T_b \)
  - Recent history does not count.

- \( \alpha = 1 \)
  - \( r_{n+1} = t_n \)
  - Only the actual last CPU burst counts.

Exponential Averaging and the Value of \( \alpha \)

- If we expand the formula, we get:
  \[
  r_{n+1} = \alpha t_{n+1} + (1-\alpha) r_{n+1} + \ldots\]
- Since both \( \alpha \) and \( 1-\alpha \) are less than or equal to 1, each successive term has less weight than its predecessor.

Exponential Averaging

Round Robin (RR)

- Each process gets a small unit of CPU time (time quantum), usually 10-200 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

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>P_1</td>
<td>53</td>
</tr>
<tr>
<td>P_2</td>
<td>17</td>
</tr>
<tr>
<td>P_3</td>
<td>68</td>
</tr>
<tr>
<td>P_4</td>
<td>24</td>
</tr>
</tbody>
</table>

* The Gantt chart is:

```
0  20  37  77  97  117  134  162
```

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

Turnaround Time Varies With Time Quantum

Multilevel Queue

* Ready queue is partitioned into separate queues: foreground (interactive) background (batch)
* Each queue has its own scheduling algorithm, foreground – RR, background – FCFS
* Scheduling must be done between the queues.
  - Fixed priority scheduling; (i.e., serve all from foreground then from background). Possibility of starvation.
  - Time slice – each queue gets a certain amount of CPU time which it can schedule amongst its processes; i.e., 80% to foreground in RR
  - 20% to background in FCFS

Multilevel Queue Scheduling

Multilevel Feedback Queue

* A process can move between the various queues; aging can be implemented this way.
* Multilevel-feedback-queue scheduler defined by the following parameters:
  - number of queues
  - scheduling algorithms for each queue
  - method used to determine when to upgrade a process
  - method used to determine when to demote a process
  - method used to determine which queue a process will enter when that process needs service

Introduction to Scheduling
Example of Multilevel Feedback Queue

- Three queues:
  - $Q_0$ - time quantum 8 milliseconds
  - $Q_1$ - time quantum 16 milliseconds
  - $Q_2$ - FCFS

- Scheduling:
  - A new job enters queue $Q_0$, which is served FCFS. When it gains CPU, job receives 8 milliseconds. If it does not finish in 8 milliseconds, the job is moved to queue $Q_1$.
  - At $Q_1$, the job is again served FCFS and receives 16 additional milliseconds. If it still does not complete, it is preempted and moved to queue $Q_2$. 

Performance for Schedulers

- **Queueing Theory Analysis** - uses well-established mathematical models and techniques.
- **Simulation** - create a model of the system and simulate its performance using simulation software.
- **Empirical Experiments** - implement and test the algorithms in a real system.

Queuing Theory Analysis (cont’d)
Queuing Theory Analysis (cont’d)

- Inputs:
  - arrival rate - from a probability distribution (usually Poisson which implies random arrivals)
  - service time - from a probability distribution (often exponential)
  - scheduling discipline/algorithm

Queuing Theory Analysis (cont’d)

- Outputs:
  - Items waiting
  - Waiting time
  - Items queued
  - Queuing time

Simulation Analysis

- Discrete-event Simulation
  - Often uses models similar to queuing analysis
  - More detailed or more realistic parameters (e.g., trace driven)
  - Simulates events step by step and gathers statistics rather than using mathematical formulas

Evaluation of CPU Schedulers by Simulation

Performance metrics

- Turnaround time = time from when job is submitted to the system until when it is completed
- Normalized turnaround time = Turnaround time / job runtime (also called slowdown)
- Sometime ‘response time’ is used interchangeably. Strictly speaking, response time is used in interactive systems to mean the time from when the user submits a request to systems’ first reaction

Simulation Analysis (cont’d)

Figure 5.15: Simulation Results for Waiting Time
Empirical Experiments

- Run experiments on live system
- Properties:
  - Costly and time-consuming
  - Sometimes not possible
  - More realistic

Traditional UNIX Scheduling

- Multilevel feedback queues
- 128 priorities possible (-64 to -63)
- 1 Round Robin queue per priority
- Every scheduling event the scheduler picks the highest priority (lowest number) non-empty queue and runs jobs in round-robin

Traditional UNIX Scheduling

- All processes assigned a baseline priority based on the type and current execution status (for convenience here renumber from 0 to 127):
  - swapper 0
  - waiting for disk 20
  - waiting for lock 35
  - user-mode execution 50
- At scheduling events, all process’s priorities are adjusted based on the amount of CPU used, the current load, and how long the process has been waiting.
- Most processes are not running, so lots of computing shortcuts are used when computing new priorities.

UNIX Scheduler

Process Scheduling in Unix

- Negative numbers reserved for processes waiting in kernel mode (just woken up by interrupt handlers) (why do they have a higher priority?)
- Time quantum = 1/10 sec (empirically found to be the longest quantum that could be used without loss of the desired response for interactive jobs such as editors)
- short time quantum means better interactive response
- long time quantum means higher overall system throughput since less context switch overhead and less processor cache flush.
- Priority dynamically adjusted to reflect
  - resource requirement (e.g., blocked awaiting an event)
  - resource consumption (e.g., CPU time)

UNIX Priority Calculation

- Every 4 clock ticks a processes priority is updated:
  \[ P = BASELINE + \left[ \text{utilization} \right] + 2 \text{NiceFactor} \]
- The utilization is incremented by 1 on every clock tick when the process is running; it is also decayed every second (next slide)
- The niceFactor allows user some control of job priority. It can be set by user from –20 to 20.
- Jobs using a lot of CPU increase the priority value. Interactive jobs not using much CPU will return to the baseline.
UNIX Priority Calculation

• Without decay, long running CPU bound jobs will get “stuck” at the highest numbered priority.
• Decay function used to weight utilization to recent CPU usage.
• A process’s utilization at time \( t \) is decayed every second:
  \[
  \text{util}_t = \left( \frac{2 \times \text{load}}{2 \times \text{load} + 1} \right) \times \text{util}_{t-1} + \text{niceFactor}
  \]
• The system load is the system average number of runnable jobs during last 1 minute

UNIX Priority Decay

• 1 job on CPU: load will thus be 1. Assume niceFactor is 0.
• Compute utilization at time \( N \):
  \[
  U_1 = \frac{2}{3} U_0
  \]
  \[
  U_2 = \frac{2}{3} \left( U_1 + \frac{2}{3} U_0 \right) = \frac{2}{3} U_1 + \left( \frac{2}{3} \right)^2 U_0
  \]
  \[
  U_n = \frac{2}{3} U_{n-1} + \left( \frac{2}{3} \right)^n U_0
  \]

Summary of Unix Scheduler

• Commonly used implementation with multiple priority queues
• Priority computed using 3 factors
  - PUSER used as a base (changed dynamically)
  - CPU utilization (time decayed)
  - Value specified at process creation (nice)
• Processes with short CPU bursts are favored
• Processes just woken up from blocked states are favored even more
• Weighted averaging of CPU utilization
• Details vary in different versions of Unix

Introduction to Scheduling