Basic Concepts

- 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
CPU Scheduler

- Selects from among the processes in ready queue, and allocates the CPU to one of them
  - Queue may be ordered in various ways
- CPU scheduling decisions may take place when a process:
  1. Switches from running to waiting state
  2. Switches from running to ready state
  3. Switches from waiting to ready
  4. Terminates
- Scheduling under 1 and 4 is nonpreemptive
- All other scheduling is preemptive
  - Consider access to shared data
  - Consider preemption while in kernel mode
  - Consider interrupts occurring during crucial OS activities

Dispatcher

- Dispatcher module gives control of the CPU to the process selected by the short-term scheduler; this involves:
  - switching context
  - switching to user mode
  - jumping to the proper location in the user program to restart that program
- Dispatch latency – time it takes for the dispatcher to stop one process and start another running
Scheduling Criteria

- **CPU utilization** – keep the CPU as busy as possible
- **Throughput** – # of processes that complete their execution per time unit
- **Turnaround time** – amount of time to execute a particular process
- **Waiting time** – amount of time a process has been waiting in the ready queue
- **Response time** – amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment)

Scheduling Algorithm Optimization Criteria

- Max CPU utilization
- Max throughput
- Min turnaround time
- Min waiting time
- Min response time

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 that the 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:

\[
\begin{array}{ccc}
\text{Process} & \text{Arrival Time} & \text{Burst Time} \\
\hline
P_2 & 0.0 & 6 \\
P_3 & 2.0 & 8 \\
P_1 & 4.0 & 7 \\
P_4 & 5.0 & 3 \\
\end{array}
\]
- Average waiting time: \( (3 + 16 + 9 + 0) / 4 = 7 \)
- Much better than previous case
  - Convoy effect - short process behind long process
- Consider one CPU-bound and many I/O-bound processes

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
- SJF is optimal – gives minimum average waiting time for a given set of processes
  - The difficulty is knowing the length of the next CPU request
  - Could ask the user

Example of SJF
Determining Length of Next CPU Burst

- Can only estimate the length – should be similar to the previous one
- Then pick process with shortest predicted next CPU burst

- Can be done by using the length of previous CPU bursts, using exponential averaging

1. \( t_p \) = actual length of \( n^{th} \) CPU burst
2. \( \tau_{pred} \) = predicted value for the next CPU burst
3. \( \alpha, 0 < \alpha < 1 \)
4. Define:
   \[ \tau_{new} = \alpha t_p + (1-\alpha) \tau_{pred} \]

- Commonly, \( \alpha \) set to \( \frac{1}{2} \)
- Preemptive version called shortest-remaining-time-first

Prediction of the Length of the Next CPU Burst

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 priority scheduling where priority is the inverse of predicted next CPU burst time
- Problem = Starvation – low priority processes may never execute
- Solution = Aging – as time progresses increase the priority of the process
Example of Priority Scheduling

<table>
<thead>
<tr>
<th>Process</th>
<th>Burst Time</th>
<th>Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>P_1</td>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>P_2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>P_3</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>P_4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>P_5</td>
<td>5</td>
<td>2</td>
</tr>
</tbody>
</table>

- Priority scheduling Gantt Chart

- Average waiting time = 8.2 msec

Round Robin (RR)
- Each process gets a small unit of CPU time (time quantum \( q \)), 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.
- Timer interrupts every quantum to schedule next process
- 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 = 4

<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>

- The Gantt chart is:

- Typically, higher average turnaround than SJF, but better response
- \( q \) should be large compared to context switch time
- \( q \) usually 10ms to 100ms, context switch < 10 usec
Time Quantum and Context Switch Time

Turnaround Time Varies With The Time Quantum

Multilevel Queue

- Ready queue is partitioned into separate queues, eg:
  - foreground (interactive)
  - background (batch)
- Process permanently in a given queue
- 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

- MQ requires process to be assigned a priori
- 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

Example of Multilevel Feedback Queue

- Three queues:
  - \( Q_0 \) – RR with time quantum 8 milliseconds
  - \( Q_1 \) – RR 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, job is moved to queue \( Q_1 \)
  - At \( Q_1 \) job is again served FCFS and receives 16 additional milliseconds
  - If it still does not complete, it is preempted and moved to queue \( Q_2 \)
Multilevel Feedback Queues

Thread Scheduling

- Distinction between user-level and kernel-level threads
- When threads supported, threads scheduled, not processes
- Many-to-one and many-to-many models, thread library schedules user-level threads to run on LWP
  - Known as process-contention scope (PCS) since scheduling competition is within the process
  - Typically done via priority set by programmer
- Kernel thread scheduled onto available CPU is system-contention scope (SCS) – competition among all threads in system

Multiple-Processor Scheduling

- CPU scheduling more complex when multiple CPUs are available
  - Homogeneous processors within a multiprocessor
  - Asymmetric multiprocessing – only one processor accesses the system data structures, alleviating the need for data sharing
  - Symmetric multiprocessing (SMP) – each processor is self-scheduling, all processes in common ready queue, or each has its own private queue of ready processes
  - Processor affinity – process has affinity for processor on which it is currently running
    - soft affinity
    - hard affinity
  - Variations including processor sets
Numerous and CPU Scheduling

Note that memory-placement algorithms can also consider affinity.

Multicore Processors

- Recent trend to place multiple processor cores on the same physical chip
- Faster and consumes less power
- Multiple threads per core also growing
  - Takes advantage of memory stall to make progress on another thread while memory retrieval happens

Multithreaded Multicore System
Virtualization and Scheduling
- Virtualization software schedules multiple guests onto CPU(s)
- Each guest doing its own scheduling
  - Not knowing it doesn't own the CPUs
  - Can result in poor response time
  - Can effect time-of-day clocks in guests
- Can undo good scheduling algorithm efforts of guests
- Hyperthreading is an example

Algorithm Evaluation
- How to select CPU-scheduling algorithm for an OS?
- Determine criteria, then evaluate algorithms
- Deterministic modeling
  - Type of analytic evaluation
    - Takes a particular predetermined workload and defines the performance of each algorithm for that workload

Queueing Models
- Describes the arrival of processes, and CPU and I/O bursts probabilistically
  - Commonly exponential, and described by mean
  - Computes average throughput, utilization, waiting time, etc
  - Computer system described as network of servers, each with queue of waiting processes
  - Knowing arrival rates and service rates
  - Computes utilization, average queue length, average wait time, etc
Little’s Formula

- \( n \) = average queue length
- \( W \) = average waiting time in queue
- \( \lambda \) = average arrival rate into queue
- Little's law – in steady state, processes leaving queue must equal processes arriving, thus
  \( n = \lambda \times W \)
  - Valid for any scheduling algorithm and arrival distribution

- For example, if on average 7 processes arrive per second, and normally 14 processes in queue, then average wait time per process = 2 seconds

Simulations

- Queueing models limited
- **Simulations** more accurate
  - Programmed model of computer system
  - Clock is a variable
  - Gather statistics indicating algorithm performance
  - Data to drive simulation gathered via
    - Random number generator according to probabilities
    - Distributions defined mathematically or empirically
    - Trace tapes record sequences of real events in real systems

Evaluation of CPU Schedulers by Simulation

- Simulation statistics for FCFS
- Simulation statistics for SJF
- Simulation statistics for RR (\( q = 14 \))