Scheduling

- On a multiprocessing system, more than one process may be available to run.
- The task of deciding which process to run next is called *scheduling*, and is performed by the *scheduler*.
  - Different types of system have different goals, and so run different *scheduling algorithms*.

- Thread scheduling and process scheduling are similar, but not identical.
Different Process Requirements

CPU-Bound Process
Running on CPU
Makes I/O request
Long CPU burst

I/O Bound Process
Waiting on I/O
Short CPU burst

Generally, if an I/O bound process wants to run, it should be allowed to run, so as to keep the I/O device busy.

When to Schedule?

A process is started.
☐ Run child first, or continue with parent?

A process terminates.
☐ Need to schedule a different process.

A process blocks (due to I/O, semaphore, etc)
☐ Need to schedule a different process.

An I/O interrupt occurs.
☐ May choose to schedule another process.
☐ Interrupt may be a clock interrupt - can periodically reschedule processes.
When to Schedule?

Two types scheduler, according to how they deal with clock interrupts:

**Non-preemptive**: a process is allowed to run until it blocks on I/O (however long that takes).

**Preemptive**: a process is only run for a certain length of time. After that another process is selected to run.

Goals of Scheduling Algorithms

**All systems**
- Fairness - giving each process a fair share of the CPU
- Policy enforcement - seeing that stated policy is carried out
- Balance - keeping all parts of the system busy

**Batch systems**
- Throughput - maximize jobs per hour
- Turnaround time - minimize time between submission and termination
- CPU utilization - keep the CPU busy all the time

**Interactive systems**
- Response time - respond to requests quickly
- Proportionality - meet users’ expectations

**Real-time systems**
- Meeting deadlines - avoid losing data
- Predictability - avoid quality degradation in multimedia systems
Many different possible scheduling algorithms are possible. A few are:

- First-Come First-Served
- Shortest Job First
- Shortest Remaining Time Next
- Three-Level Scheduling

Batch System Scheduling: First-Come First-Served

- Simple FIFO job queue.
  - Process at head of queue runs until it blocks.
  - When it unblocks, rejoins tail of queue.

- Advantage:
  - Very simple

- Disadvantage:
  - Mix of CPU and I/O bound processing causes I/O bound job to be interspersed with long CPU-bound bursts.
  - During these bursts, I/O device is mostly idle.
Batch System Scheduling: Shortest Job First

- Non-preemptive algorithm for when run-times are known in advance.
- Run job with shortest run time first.

<table>
<thead>
<tr>
<th>4</th>
<th>8</th>
<th>4</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>C</td>
<td>D</td>
</tr>
</tbody>
</table>

FCFS: mean delay until completion is 
\[(4+12+16+20)/4 = 13\]

<table>
<thead>
<tr>
<th>4</th>
<th>4</th>
<th>4</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>C</td>
<td>D</td>
<td>B</td>
</tr>
</tbody>
</table>

SJF: mean delay until completion is 
\[(4+8+12+20)/4 = 11\]

- This is provably optimal (in terms of delay), so long as all jobs are available at the start.

Batch System Scheduling: Shortest Remaining Time Next

- Preemptive version of Shortest Job First.
  - Always run the job whose remaining run time is least.

- Works well when processes arrive incrementally, although their run time still needs to be known in advance.
  - Gives new short jobs good service.
Batch System Scheduling:

Three Level Scheduling

Admission Scheduler: takes jobs from the input queue. Goal: good balance of CPU and I/O bound jobs. Long timescale.

Memory Scheduler: manages swapping from disk, so as to effectively use CPU, memory and I/O. Medium Timescale (once per second)

CPU Scheduler: decides which job in memory to run. Short Timescale.

Scheduling in Interactive Systems

Again many many algorithms in the literature.

Some examples:
- Round-Robin Scheduling
- Priority Scheduling
- Multiple Queues
- Shortest Process Next
- Guaranteed Scheduling
- Lottery Scheduling
Interactive System Scheduling: 
Round Robin

- Maintain a queue of runnable processes.
- Assign each process a time interval or *quantum*.
- Run process from head of queue.
- If process is still running at the end of its quantum, de-schedule it, and move it to the end of the queue.

- Quantum of 20ms - 50ms is common.

Interactive System Scheduling: 
Priority Scheduling

Maintain a process queue for each priority.
Runnable process with highest priority is always run.

- Can be adaptive:
  - Decrease priority of a process that’s run for a long time.
  - Increase priority of I/O bound processes.
Interactive Systems Scheduling:

Shortest Process Next

Adaptation of Shortest Job First for Interactive Systems.

- Run job with shortest expected run time for the next user “command”.
- Estimate run time based on new measurement and aged past history:

\[
T_n = at_n + (1 - a)T_{n-1}
\]

\[T: \text{estimated time}\]

\[t: \text{measured time}\]

Interactive Systems Scheduling:

Lottery Scheduling

- Attempt to provide statistical fairness with a simple implementation.
  - Give processes “lottery tickets” for resources.
  - When a process needs to be scheduled, choose a lottery ticket at random, and run that job that owns that ticket.

- Advantages:
  - Responsive
  - Cooperative processes may trade tickets:
    - Client can loan tickets to a server so the server can process the client’s request.
  - Can achieve proportional fairness.
Real-Time Systems

- Hard Real-Time:
  - there are guarantees that MUST be met.
- Soft Real-Time:
  - deadlines should be met, but no hard guarantee.

- Hard real-time processes generally short, predictable, and run to completion quickly.
- Scheduler handles external events so as to ensure that all guarantees are met.

Scheduling in Real-Time Systems

- Given a system with $m$ periodic events, and where event $i$ occurs with a period of $P_i$ seconds and requires $C_i$ seconds to process.
- The load can only be handled if:
  $$\sum_{i=1}^{m} \frac{C_i}{P_i} \leq 1$$
- Such a system is said to be schedulable.
Policy versus Mechanism

- Separate what is **allowed** to be done with **how** it is done
  - a process knows which of its child threads are important and need priority

- Scheduling algorithm parameterized
  - mechanism in the kernel

- Parameters filled in by user processes
  - policy set by user process

Real Systems: Solaris 2

Priority-based preemptive scheduling of threads, with 4 classes:

**real-time**
- Highest priority - can provide guarantees.

**system**
- Used for kernel service threads.

**time-sharing**
- Dynamically allocates priority and time-slice length - higher priority gets shorter time-slice. Interactive processes get higher priority.

**interactive**
- Same algorithm as time-sharing, but used to give windowing applications higher performance.
Real Systems: Windows 2000

- Priority-based, preemptive scheduling of threads.
- 32-level priority scheme determines execution order.
  - Highest priority runnable task is run first.
  - Two classes:
    - **real-time**: priorities 16-32
    - **variable**: priorities 1-15
  - For variable class processes:
    - Priority is reduced when a quantum expires.
    - When unblocked, priority is boosted depending on why it was blocked. Eg: keyboard event gives high boost when gives good interactivity.
    - Foreground process on screen gets higher priority.