Scheduling Wrap-up & Threads

Round Robin: FIFO w/ time quantum, preemptive

→ unfair to tasks that didn't use the full time quantum
  blocks after 1 ms, other tasks run
  10ms time slice  I/O bound / interactive ⇒ have to wait a long time
  10ms time slice  CPU bound

→ Option 1: jobs w/ less time on the CPU are strictly prioritized
  CFS: Linux default scheduler, time-ordered red black tree,
  schedules the task that spent least time on CPU, helps
  it catch up to its fair share of CPU time

→ Option 2: reduce wait time for I/O bound tasks
  MLFD: a number of RR queues w/ different time quantums
  5ms queue 1
  10ms queue 2
  20ms queue 3
  keep the I/O tasks in the top queue, shorter wait time
**Multilevel Feedback Queue (MLFQ)**

- Shorter time slice ⇒ shorter wait time, good for interactive jobs
- Longer time slice ⇒ less context switch for longer tasks, good for CPU-bound tasks

<table>
<thead>
<tr>
<th>Time Quantum</th>
<th>Tasks</th>
<th>Higher Priority</th>
<th>Lower Priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>5ms</td>
<td>task A</td>
<td>task B</td>
<td>...</td>
</tr>
<tr>
<td>10ms</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20ms</td>
<td>task C</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>40ms</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Scheduler runs tasks from the higher priority queue, if empty, goes to the next queue...
- Current task will be preemption if there are new tasks in higher priority queues.
- RR within a queue

**How do we know which tasks go to which Q?**

- Assumes all new tasks are short (top Q)
- If blocks before time slice ⇒ same Q
- If uses up the full time ⇒ down a Q
- Starvation for long tasks?

- Can this be gamed by injecting sleep?
- Set max time for a task per queue
Threads

- unit of execution / task
  - execution states: PC, SP, registers

- multithreaded program (concurrent)
  - divide program into tasks (threads)

- Concurrency vs. parallelism -> execute simultaneously
  - Program is structured into tasks; tasks take turn making progress (concurrently)
  - Concurrency can happen on single core; Kernel is always concurrent

Process = Address Space + OS Resources + 1+ Threads

- Threads share code, heap, data, but have their own stack & execution state (PC, regs)
- Managed & scheduled by the Kernel

Kernel stack

Thread creation

Stack for t1
Stack for t2

Heap
Data
Code

Single threaded process

Multithreaded (2) process

Each has its own
User stack &
Kernel stack

<multithreading diagram>
Switching between threads = context switch

- Save current thread’s context onto its kernel stack
- Switch to the next thread’s kernel stack & pop the saved context
- If the next thread is from a different process, load new address space & flush the TLB

xK: current thread -> scheduler -> next thread
   (pick new thread to run)
Pthreads API

- pthread_create (thread_func, args)

- pthread_join (tid)  wait for tid to exit, any thread can join another

- pthread_exit (exit-status)  terminate the calling thread.

- pthread_detach  upon exit, clean up resources (stack) automatically
  (does not require join)
Threads Execution

<table>
<thead>
<tr>
<th>Programmer's View</th>
<th>Possible Execution #1</th>
<th>Possible Execution #2</th>
<th>Possible Execution #3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$x = x + 1;$</td>
<td>$x = x + 1;$</td>
<td>$x = x + 1;$</td>
<td>$x = x + 1;$</td>
</tr>
<tr>
<td>$y = y + x;$</td>
<td>$y = y + x;$</td>
<td>$y = y + x;$</td>
<td>$y = y + x;$</td>
</tr>
<tr>
<td>$z = x + 5y;$</td>
<td>$z = x + 5y;$</td>
<td>$z = x + 5y;$</td>
<td>$z = x + 5y;$</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Thread is suspended.
Value of $x$ may change in a multi-threaded process.
Other thread(s) run.
Thread is resumed.

Timer Interrupt

<table>
<thead>
<tr>
<th>Thread 1</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Thread 3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>