Scheduler Activations

Learning Outcomes

• An understanding of hybrid approaches to thread implementation
• A high-level understanding of scheduler activations, and how they overcome the limitations of user-level and kernel-level threads.

User-level Threads


User-level Threads

Kernel-Level Threads

✓ Fast thread management (creation, deletion, switching, synchronisation,...)
  ▶ Blocking blocks all threads in a process
    – Syscalls
    – Page faults
  ▶ No thread-level parallelism on multiprocessor
Kernel-level Threads
- Slow thread management (creation, deletion, switching, synchronisation…)
  - System calls
  - Blocking blocks only the appropriate thread in a process
  - Thread-level parallelism on multiprocessor

Hybrid Multithreading
- Can get real thread parallelism on multiprocessor
- Blocking still a problem!!

Scheduler Activations
- First proposed by [Anderson et al. 91]
- Idea: Both schedulers co-operate
  - User scheduler uses system calls
    - Kernel scheduler uses upcalls!
  - Two important concepts
    - Upcalls
    - Notify the user-level of kernel scheduling events
    - Activations
    - A new structure to support upcalls and execution approximately a kernel thread
    - As many running activations as (allocated) processors
    - Kernel controls activation creation and destruction

Performance

<table>
<thead>
<tr>
<th>Operation</th>
<th>FastThreads</th>
<th>Topaz threads</th>
<th>Ultrap processes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nil Fork</td>
<td>34</td>
<td>048</td>
<td>11300</td>
</tr>
<tr>
<td>Signal-Wait</td>
<td>37</td>
<td>441</td>
<td>1840</td>
</tr>
</tbody>
</table>
Upcalls to User-level scheduler

- **New**
  - Allocated a new virtual CPU
  - Can schedule a user-level thread
- **Preempted**
  - Deallocated a virtual CPU
  - Can schedule one less thread
- **Blocked**
  - Notifies thread has blocked
  - Can schedule another user-level thread
- **Unblocked**
  - Notifies a thread has become runnable
  - Must decided to continue current or unblocked thread

Working principle

- Blocking syscall scenario on 2 processors
Working principle
- Blocking syscall scenario on 2 processors

Blocking syscall

Working principle
- Blocking syscall scenario on 2 processors

I/O completion

Working principle
- Blocking syscall scenario on 2 processors

Working principle
- Blocking syscall scenario on 2 processors

Working principle
- Blocking syscall scenario on 2 processors

Working principle
- Blocking syscall scenario on 2 processors

Working principle
- Blocking syscall scenario on 2 processors
Scheduler Activations

- Thread management at user-level
  - Fast
- Real thread parallelism via activations
  - Number of activations (virtual CPU) can equal CPUs
- Blocking (syscall or page fault) creates new activation
  - User-level scheduler can pick new runnable thread.
- Fewer stacks in kernel
  - Blocked activations + number of virtual CPUs

Performance

<table>
<thead>
<tr>
<th>Operation</th>
<th>Exit Threads</th>
<th>Exit Threads</th>
<th>Exit Threads</th>
<th>Exit Threads</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>on Top Threa</td>
<td>on Top Threa</td>
<td>on Top Threa</td>
<td>on Top Threa</td>
</tr>
<tr>
<td>Null Fork</td>
<td>34</td>
<td>37</td>
<td>948</td>
<td>11330</td>
</tr>
<tr>
<td>Signal-Wait</td>
<td>37</td>
<td>42</td>
<td>441</td>
<td>1840</td>
</tr>
</tbody>
</table>

Adoption

- Adopters
  - BSD “Kernel Scheduled Entities”
    - Reverted back to kernel threads
  - Variants in Research OSs: K42, Barrelish
  - Digital UNIX
  - Solaris
  - Mach
  - Windows 7
- Linux -> kernel threads