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

- 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

User-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:

- User Mode
- Kernel Mode

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

Performance

<table>
<thead>
<tr>
<th>Operation</th>
<th>FastTrack</th>
<th>Topaz</th>
<th>Ultrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Null 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>

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
  - 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

Upcalls
Scheduler Activations

- Instead of
  - User Space
  - Kernel Space
  - Hardware

- ...rather use the following scheme:
  - User Space
  - Kernel Space
  - Hardware

Upcalls to User-level scheduler

- New (processor #)
  - Allocated a new virtual CPU
  - Can schedule a user-level thread

- Preempted (activation # and its machine state)
  - Deallocated a virtual CPU
  - Can schedule one less thread

- Blocked (activation #)
  - Notifies thread has blocked
  - Can schedule another user-level thread

- Unblocked (activation # and its machine state)
  - Notifies a thread has become runnable
  - Must decide to continue current or unblocked thread

Working principle

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

Process

Preempt

Blocking syscall

New C + blocked B
Working principle

- Blocking syscall scenario on 2 processors

Process

1 2 3 4
A B C

1/0 completion

Scheduler Activations

- Thread management at user-level
  - Fast
- Real thread parallelism via activations
  - Number of activations (virtual CPUs) 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 (compute-bound)

Table IV: Thread Operation Latencies (microsec)

<table>
<thead>
<tr>
<th>Operation</th>
<th>FastThreads on TopThreads</th>
<th>FastThreads on Scheduler Activations</th>
<th>TopThreads</th>
<th>Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Null Fork</td>
<td>34</td>
<td>37</td>
<td>948</td>
<td>11300</td>
</tr>
<tr>
<td>Signal Wait</td>
<td>35</td>
<td>42</td>
<td>441</td>
<td>1040</td>
</tr>
</tbody>
</table>

Fig. 3: Speedup of N-body application versus number of processors, 100% of memory available.
Performance (I/O Bound)

Adoption

- **Adopters**
  - BSD "Kernel Scheduled Entities"
  - Reverted back to kernel threads
  - Variants in Research OSs: K42, Barrelfish
  - Digital UNIX
  - Solaris
  - Mach
  - Windows 64-bit *User Mode Scheduling*
  - Linux -> kernel threads