Scheduler Activations

Including some slides modified from Raymond Namyst, U. Bordeaux
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 Mode

Kernel Mode

Scheduler

Scheduler

Scheduler

Scheduler

Process A

Process B

Process C
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

Scheduler
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 I: Thread Operation Latencies (μsec.)

<table>
<thead>
<tr>
<th>Operation</th>
<th>FastThreads</th>
<th>Topaz threads</th>
<th>Ultrix processes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Null Fork</td>
<td>34</td>
<td>948</td>
<td>11300</td>
</tr>
<tr>
<td>Signal-Wait</td>
<td>37</td>
<td>441</td>
<td>1840</td>
</tr>
</tbody>
</table>

User-level threads
Kernel-level threads
Hybrid Multithreading

User Mode

Kernel Mode
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
Scheduler
Scheduler Activations

• Instead of

User Space

Kernel Space

Hardware

syscall

I/O request

interrupt

CPU time wasted

• …rather use the following scheme:

User Space

Kernel Space

Hardware

CPU used

upcall

upcall
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 decided to continue current or unblocked thread
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
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

![Diagram showing processes and blocking scenario]
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 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

### Table IV. Thread Operation Latencies (μsec.)

<table>
<thead>
<tr>
<th>Operation</th>
<th>FastThreads on Topaz Threads</th>
<th>FastThreads on Scheduler Activations</th>
<th>Topaz threads</th>
<th>Ultrix processes</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>37</td>
<td>42</td>
<td>441</td>
<td>1840</td>
</tr>
</tbody>
</table>
Performance (compute-bound)

Fig. 2. Speedup of N-Body application versus number of processors, 100% of memory available.
Performance (I/O Bound)

Fig. 3. Execution time of N-Body application versus amount of available memory, 6 processors.
Adoption

• Adopters
  – BSD “Kernel Scheduled Entities”
    • Reverted back to kernel threads
  – Variants in Research OSs: K42, Barreelfish
  – Digital UNIX
  – Solaris
  – Mach
  – Windows 7 64-bit *User Mode Scheduling*

• Linux, MacOS(?) -> kernel threads
Fig. 1. Example: I/O request/completion.