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

✗ 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 1. Thread Operation Latencies (μsec.)

<table>
<thead>
<tr>
<th>Operation</th>
<th>FastThreads</th>
<th>Topaz threads</th>
<th>Ulrix 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>414</td>
<td>1840</td>
</tr>
</tbody>
</table>

Hybrid Multithreading

✓ Can get real thread parallelism on multiprocessor
✗ Blocking can still be 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 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

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
Working principle
• Blocking syscall scenario on 2 processors

Process
1 2 3 4
A B

Preempt

Blocking syscall

Process
1 2 3 4
A B C
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 (usec.)

| Operation   | TopThreads | FastThreads on Scheduler Activation | TopThreads | Util Int
|-------------|-----------|-------------------------------------|-----------|--------
| Nuss Freq   | 34        | 37                                  | 948        | 11300  |
| Signal Wait | 37        | 42                                  | 441        | 1840   |

Performance (compute-bound)

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

- Zopas threads
- Orig FastThreading
- New FastThreading

% available memory

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