SCHEDULER ACTIVATIONS

Effective Kernel Support for the User–level Management of Parallelism

Thomas E. Anderson, Brian N. Bershad, Edward D. Lazowska, Henry M. Levy

February 1992

Presenter – Anitha Suryanarayan
### Thread System – User Level

- Thread library’s code + data structures in user space
- Invocation of library function = local function call. Not system call

<table>
<thead>
<tr>
<th>PROS</th>
<th>CONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Good performance</td>
<td></td>
</tr>
<tr>
<td>- fast thread switches</td>
<td></td>
</tr>
<tr>
<td>- Scale better</td>
<td></td>
</tr>
<tr>
<td>Highly flexible</td>
<td></td>
</tr>
<tr>
<td>- Custom scheduling algorithm</td>
<td></td>
</tr>
<tr>
<td>Poor concurrency support</td>
<td></td>
</tr>
<tr>
<td>- blocking system calls starve sibling threads</td>
<td></td>
</tr>
</tbody>
</table>
### THREAD SYSTEM - KERNEL LEVEL

- Thread library’s code + data structures in kernel space
- Invocation of library function = system call

<table>
<thead>
<tr>
<th>PROS</th>
<th>CONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Good concurrency support</td>
<td>Poor performance</td>
</tr>
<tr>
<td>- Blocking system calls do not starve</td>
<td>- Operations involve system calls</td>
</tr>
<tr>
<td>sibling threads</td>
<td>- Full context switch to perform thread</td>
</tr>
<tr>
<td></td>
<td>switches</td>
</tr>
<tr>
<td></td>
<td><strong>Less Flexible</strong></td>
</tr>
<tr>
<td></td>
<td>- Generic scheduling Algorithm</td>
</tr>
</tbody>
</table>
**User Threads**

1. Excellent performance
   - No system calls to perform thread operations

2. More flexible => Can use domain specific scheduling algorithm (‘customized’ thread library)

3. Blocking system calls such as I/O problematic; Starvation of sibling threads

**Kernel Threads**

1. Bad performance
   - System calls needed to perform thread operations

2. Generic scheduling algorithm (scheduled by kernel)

3. Good integration with system services – blocking calls do not prevent other user threads from being scheduled. Less likelihood of starvation
## COST OF THREADS

<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>

Scheduler Activations: Effective Kernel Support for the User-Level Management of Parallelism, Anderson et al
N:1 MODEL  1:1 MODEL  N:M MODEL
MOTIVATION

- A mechanism through which we can obtain Performance & flexibility of user-level threads + Functionality of kernel threads
SCHEDULER ACTIVATIONS

- Communication scheme between the user-thread library and the kernel
- Upcalls
- Data Structures
  - Scheduler activation stacks
  - Activation control block
- Punish greedy apps
Each process is allocated a number of virtual processors instead of kernel threads.
Each process can decide what to do with the CPUs it has been allocated.
This abstraction, presented in the form of scheduler activations allows:
- The kernel to have control over processor allocations
- The user process to have control over its share of processors
UPCALLS

Source - https://www.cs.columbia.edu/~smb/classes/s06-4118/l05.pdf
Add this processor (processor #)
  ◦ *Execute a runnable user-level thread.*

Processor has been preempted (preempted activation # and its machine state)
  ◦ *Return to the ready list the user-level thread that was executing in the context of the preempted scheduler activation.*

Scheduler activation has blocked (blocked activation #)
  ◦ *The blocked scheduler activation is no longer using its processor.*

Scheduler activation has unblocked (unblocked activation # and its machine state)
  ◦ *Return to the ready list the user-level thread that was executing in the context*
Communication : User Level to Kernel

- **#Runnable Threads != # Processors**
- **Add more processors** (additional # of processors needed)
  - *Allocate more processors to this address space and start them running scheduler activations.*
- **This processor is idle ()**
  - *Preempt this processor if another address space needs it.*

Scheduler Activations: Effective Kernel Support for the User-Level Management of Parallelism, Anderson et al
I/O request/completion

Fig. 1. Example: I/O request/completion.
CRITICAL SECTIONS

- Preempting a thread that is running in its critical section
  - Why is this a problem?
  - Does it exist with kernel threads?

- How is it managed through scheduler activation?
The Topaz OS and FastThreads thread library was modified to implement scheduler activations

Performance Optimizations

- No lock latency because threads running their critical sections are not preempted
- Recycle scheduler activations
## Cost of Thread Operations

<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 Evaluation

Fig. 2. Speedup of N-Body application versus number of processors, 100% of memory available.

Fig. 3. Execution time of N-Body application versus amount of available memory, 6 processors.
To gain performance and improve flexibility, export some functionalities out of the kernel and maintain just enough communication to ensure that the kernel can do its job.

Implementations
- TAOS
- MACH 3.0
- BSD/OS
- Digital Unix
REFERENCES

- Operating System Concepts, 7th edition, Silberschatz, Galvin and Gagne


- An Implementation of Scheduler Activations on the NetBSD Operating System – Nathan J. Williams