Operating Systems
ECE344

Kernel and User Threads

Ashvin Goel
ECE
University of Toronto
Overview

- **Thread scheduler**
  - Scheduler API
  - Scheduler implementation

- **Kernel threads vs. user threads**
Kernel vs. User Threads

- Thread scheduler functions implemented in OS kernel are called **kernel threads**
  - Kernel threads virtualize CPU
    - Allow arbitrary number of threads to run on one or more CPUs
  - OS provides system calls for scheduler API
    - e.g., `pthread_create()` is a system call that calls the kernel’s `thread_create`

- Thread scheduler functions implemented in a user program are called **user threads**
  - User threads virtualize a kernel thread
    - Allow arbitrary number of threads to utilize kernel threads
Kernel vs. User Threads

Kernel Threads

Process 1 kernel thread Process 2

Kernel-level scheduler

Kernel thread table

User Threads

Process 1 user thread Process 2

User space

Kernel space

Kernel

User-level scheduler

user thread table

user thread table

kernel thread table

kernel thread
# Kernel vs. User Threads

<table>
<thead>
<tr>
<th></th>
<th>Kernel Threads</th>
<th>User Threads</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Switching cost</strong></td>
<td>Kernel switches threads, requiring running kernel code, more expensive</td>
<td>Program switches threads, time closer to procedure call, less expensive</td>
</tr>
<tr>
<td><strong>Scheduling policy</strong></td>
<td>System has fixed policies</td>
<td>User can define custom policy</td>
</tr>
<tr>
<td><strong>Blocking system calls</strong></td>
<td>When system call blocks (in kernel), kernel switches to another thread, so overlap of IO and computable is possible</td>
<td>When system call blocks, all user threads (associated with the corresponding kernel thread) block, so overlap of I/O and computation is not possible</td>
</tr>
<tr>
<td><strong>Multiprocessors</strong></td>
<td>Different kernel threads can use multiple CPUs in parallel</td>
<td>Different user threads (associated with a given kernel thread) cannot use multiple CPUs concurrently</td>
</tr>
</tbody>
</table>
Summary

- A thread scheduler allows running one or more threads
  - Implements scheduler functions define the thread API
- Kernel and user threads
  - Thread scheduler in kernel implements kernel threads
  - User program can implement user threads
- Next lecture discusses how to write programs using threads, and ensure they work correctly when thread run concurrently
Think Time

- We have described the threads API. Do programs invoke this API by using system calls or function calls?
- Why is preferable to use kernel threads instead of user threads in more programs?
- When would you definitely consider using user threads?
Think Time Answers

- We have described the threads API. Do programs invoke this API by using system calls or function calls?
  - For kernel threads, programs use system calls to invoke the kernel’s scheduler functions. For user threads, programs use function calls to invoke the program’s scheduler functions.

- Why is preferable to use kernel threads instead of user threads in more programs?
  - See table in previous slides
When would you definitely consider using user threads?

- When your program is dealing with lots (say 100-10000) of threads, and the program needs precise control over which thread runs next, and which thread runs on which core, and when IO is performed, then user threads are beneficial. In a sense, the program is really implementing OS functionality, and so needs to schedule threads itself. Typical examples of such programs are large, complex programs such as web browsers/servers, in-memory databases, that need to handle highly concurrent operations, etc.