Threads

Pradipta De
pradipta.de@sunykorea.ac.kr
Today’s Topic

- Design choices to implement Threads
- Threads in Linux
Need for Threads

• Allow parallelism among activities within a process for speedup in performance
  – Multiple activities where some may block on I/O
  – What if all activities are CPU bound?
• Easier and faster to create / destroy
• Exploits multiple CPUs more effectively

• Threads are common in software design
  – Word processor, Web Servers, etc.
Classical Thread Model

• A process model represents
  – Grouping of resources
  – Collection of execution sequences

• Threads represent the “execution sequences”, or entities scheduled for execution
  – Resources are shared
Classical Thread Model

<table>
<thead>
<tr>
<th>Per-process Items</th>
<th>Per-thread Items</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address space</td>
<td>Program counter</td>
</tr>
<tr>
<td>Global variables</td>
<td>Registers</td>
</tr>
<tr>
<td>Open files</td>
<td>Stack</td>
</tr>
<tr>
<td>Child processes</td>
<td>State</td>
</tr>
<tr>
<td>Pending alarms</td>
<td></td>
</tr>
<tr>
<td>Signals and signal handlers</td>
<td></td>
</tr>
<tr>
<td>Accounting information</td>
<td></td>
</tr>
</tbody>
</table>

Threads maintain individual items to manage independent execution, and has states to denote current execution state (running, blocked, ready, terminated)

Since threads share the address space, a thread shares global variables, and can even read, write and wipe out other threads’ stack
- What if a thread requires any thread-specific global variables?
  Thread Local Storage
Classical Thread Model

• Thread Design Issues
  – How to schedule threads
    • Thread scheduling is cooperative → a thread should explicitly yield processor (thread_yield)
  – If parent process has multiple threads, then will child process inherit all parent threads?
  – How should signals be delivered to threads?
    • Should the entire process group receive it, or individual thread
Threads Library

• Early OS did not have support for Threads
• User space library to provide support programmer APIs for creating/managing threads
• Early implementation in user space, but can be in kernel space
• 3 main thread libraries now:
  – POSIX Pthreads
  – Win32 threads
  – Java Threads
User Space Implementation

- Runs on top of a run-time system ➔ no OS dependence
- Each process maintains a thread table to manage threads
- Thread switching faster
  - No need to trap into the kernel
- Different scheduling policies
What if a thread makes a blocking system call?

- OS is unaware of multiple threads, and will block the process → all threads are blocked

- Thread makes a blocking read request
- Thread encounters a page fault
Kernel Space Implementation

- Kernel manages the thread table
- To create/destroy a thread, a system call is invoked
  - This can be slower than user space implementation
  - Thread pool: create a set of threads on initialization, and keep recycling them
Hybrid Implementation

- Multiplex user threads on kernel threads
- How does kernel notify if a thread blocks?
  - Scheduler activations: technique to make an upcall to user level runtime system that a thread has blocked ... runtime system schedules another thread
    - Low context switch overhead
    - No need to ensure non-blocking system calls from user threads

Three possible mappings of user thread to kernel threads
- M : N (where M > N)
- M : 1
- 1 : 1
Many to 1 Model

- Many user-level threads are all mapped onto a single kernel thread
- Thread management completely at user space
  - Fast context switching among threads
- Blocking system call by a thread blocks entire process
- Does not exploit multiple CPUs since only one kernel thread
- SunOS had similar design … Solaris allowed for new kernel threads when a user thread blocks
Many to Many model

- Many user threads onto many kernel threads
  - Best of both sides
- No restriction on number of threads
- Blocking calls do not block process
- Can exploit multiple processors
- Next Generation POSIX Threads implemented M:N model
1:1 model

- A separate kernel thread to handle each user thread
- No problem with blocking calls
- Management overhead can be high
  - Kernel has to create a thread control block for each user thread
- Often a limit on how many user threads can be created
  - Linux Threading Library (earlier Linux versions)
Native POSIX Thread Library (NPTL)

• Each thread has a user and kernel mode stack, and program counter
  – What if a kernel thread blocks?
    • Interrupted while executing a system call → save the execution context in the TCB

• Kernel manages processes and threads in the same way
  – Single threaded process has one task_struct
  – Multithreaded process has task_structs for each user level thread

• Introduced PID for process, and TID for thread
  – Signal can be delivered to each thread individually or to the process group

• Limits on number of threads removed
• What about context switching overhead with large number of threads?
Thread Library APIs

• Creating threads
  – fork
  – clone

• Better to use pthreads API to create and manage threads
  – Retains program portability across different OS platforms
Putting It Together

- Different Threading mechanisms
- Specific choices in Linux