Threads

Jonathan Geisler

February 16, 2007

What is a thread?

- Separate for each thread
  - PC
  - Registers
  - Stack
- Shared by entire process
  - Data
  - Code
  - OS resources

Why multithread?

- Handle multiple tasks within process concurrently (responsiveness)

Why multithread?

- Handle multiple tasks within process concurrently (responsiveness)
- Resource sharing

Why multithread?

- Handle multiple tasks within process concurrently (responsiveness)
- Resource sharing
- Less costly to create or context switch than processes (economy)

Why multithread?

- Handle multiple tasks within process concurrently (responsiveness)
- Resource sharing
- Less costly to create or context switch than processes (economy)
- Full use of multiprocessors (utilization)

User vs. Kernel threads

- User
  - Faster
  - Blocking single thread blocks all threads
- Kernel
  - Slower
  - Multiple can be scheduled
    - No blocking problems
    - Take full advantage of multiprocessor

Thread models

1. $M : 1$
   - How threads typically began on a system
   - User level threads
2. $1 : 1$
   - Easy to implement
   - Kernel level threads
3. $M : N$
   - Best performance supposedly
   - Most difficult to get "right"

Potential pitfalls

1. fork() & exec()
2. Cancellation
3. Signal handling

Unique threading issues

- Thread pools
  - Supported by Apache 2
  - Amortize costs across multiple uses to further reduce performance penalties
- Thread-specific data
  - Sounds useful, but often problem can be solved using some other technique
  - Still provided by most threading systems