DTHREADS: Efficient Deterministic Parallelism
Tongping Liu, Charlie Curtsinger, Emery Berger

Insanity: Doing the same thing over and over again and getting different results.

DTHREADS Enables...
- Deterministic executions
- Replay w/o logging
- Replicate applications on different machines

DTHREADS is the new basis of Deterministic Multithreading
Impact: Graduate Course Project: UWisc, Qualify Examination: U of Virginia, VT...

Citation: 216

What Causes Non-determinism?
- Race conditions
- Atomicity violations
- Deadlock
- Order violations
- Robust determinism

<table>
<thead>
<tr>
<th></th>
<th>Thread 1</th>
<th>Thread 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>a = 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b = 0</td>
<td>T1-1: if (b == 0), T1-2: a = 1;</td>
<td>T2-1: if (a == 0), T2-2: b = 1;</td>
</tr>
<tr>
<td>a = 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b = 1</td>
<td>T1-1: if (b == 0), T1-2: a = 1;</td>
<td>T2-1: if (a == 0), T2-2: b = 1;</td>
</tr>
</tbody>
</table>

- 99.43% 0.56% 0.01%

1. Non-ordered Shared Accesses
2. Non-ordered Synchronizations
3. Non-deterministic Memory Alloca...
Previous Approach (CoreDet)

- Check every memory access and every synchronization
  - If an access/sync is owned by me, then execute
  - Else, wait until next serial phase

Performance
Unstable
Incomplete

Isolation to Avoid Non-ordered Accesses

Threads
shared address space

Processes
disjoint address spaces

Processes-as-Threads

- Performance Difference
  - Creations: rare event
  - Context Switches: unnecessary if we match processes number to cores number

- Benefit to use process
  - "Per-thread" isolation
  - "Per-thread" protection/fault handler

DTHREADS Overview

1. Non-ordered Shared Accesses
2. Non-ordered Synchronizations
3. Non-deterministic Memory Allocs

- Isolated memory accesses
- Deterministic memory commits/synchronizations
- Deterministic memory allocs
**DTHREADS Overview**

<table>
<thead>
<tr>
<th>Parallel</th>
<th>Serial</th>
<th>Parallel</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Thread&quot; 1</td>
<td>mutex lock</td>
<td>parallel</td>
</tr>
<tr>
<td>&quot;Thread&quot; 2</td>
<td>cond wait</td>
<td>parallel</td>
</tr>
<tr>
<td>&quot;Thread&quot; 3</td>
<td>pthread create</td>
<td>parallel</td>
</tr>
</tbody>
</table>

Isolated Accesses | Deterministic memory commits & syncs | Isolated Accesses

**Page Snapshot to Support Commits**

Snapshot pages before modifications

**Page Snapshot to Support Commits**

Write back diffs

Global State

"Thread" Local State

Begin

Time
Deterministic Memory Allocs

- “Per-thread” heap
- Deterministically fetch blocks (through a global lock)
- Per-thread heaps never return blocks
- Inter-threads frees only return to the current heap

DTHREADS: Efficient Determinism

Usually faster than the state of the art
Generally as fast or faster than pthreads

Data is fetched in the unit of cache line (32/64 bytes)

Interleaved writes cause cache invalidations

Isolation Prevents False Sharing
DTHREADS: Easy To Use

% g++ myprog.cpp -lp threads

Project2B: implement processes-as-threads

• Target:
  – Run multithreaded programs successfully
• Approach:
  – Make everything shared across threads, including stack, heap, or globals of application and libraries.
  – This can be done with clone() system call.

Project2B: How

• How to share: creating mappings before clone()
  – Stack: mmap(), and pass to clone()
  – Globals:
    • Identify the range of globals
    • Mmap() to change the mapping: maintain the mapping
    • Problems: some initialized variables may gone. Thus, keep a copy at first
  – Heap:
    • Custom heap; intercepting all allocations and deallocations
    • All objects will be allocated from the same shared space

Project2B: How

• How to handle synchronization like mutexes, conditional variables, barriers?
  – pthread_mutexattr_getpshared first
  – Pass the attribute to pthread_mutex_init()
Evaluation & Reference

- You can confirm your evaluation using https://github.com/UTSASRG/multithreadingtests/tree/master/parsec
- You can reference: https://github.com/UTSASRG/Sheriff

Outline of Lecture-04

- Motivation and thread basics
  - Resources requirements: thread vs. process
- Thread implementations
  - User threads: e.g., Pthreads and Java threads
  - Kernel threads: e.g., Linux tasks
  - Map user- and kernel-level threads
  - Lightweight process and scheduler activation
- Other issues with threads: process creation and signals etc.
  - Threaded programs
    - Thread pool
    - Performance vs. number of threads vs. CPUs and I/Os

Outline of Lecture-05

- Problems with concurrent access to shared data
  - Race condition and critical section
  - General structure for enforce critical section
- Synchronization mechanism
  - Hardware supported instructions: e.g., TestAndSet
  - Software solution: e.g., semaphore
- Classical Synchronization Problems
- High-level synchronization structure: Monitor
- Case study for synchronization
  - Pthread library: mutex and conditional variables
  - Java inherit monitor and conditional variable