Operating System Transactions

Donald E. Porter, Owen S. Hofmann, Christopher J. Rossbach, Alexander Benn, and Emmett Witchel
The University of Texas at Austin
{porterde, osh, rossbach, abenn1, witchel}@cs.utexas.edu

An OS Concurrency Crisis
The POSIX API is not designed for concurrency
- Shift from time-sharing uniprocessor machines to multi-core
  - 12 core AMD chip due in January 2010
- OS state may change between any two system calls
- API race conditions are problematic for complex operations
  - Distill to single system call in simple cases (e.g. rename())
  - Some operations cannot be distilled to a single system call
- Proliferation of ad hoc solutions to race conditions
  - New file system extensions: openat, close_on_exec
  - New signal handling API: sigaction, pselect, etc.
- Developers need transactions to ensure consistency from OS

Example API Race Condition
Time-of-check-to-time-of-use (TOCTTOU) Attacks
- Attacker exploits race condition to trick a setuid program
- Changes a symbolic link between check and use

<table>
<thead>
<tr>
<th>Victim</th>
<th>Attacker</th>
</tr>
</thead>
</table>
| if(access('foo')) {
  symlink('secret','foo');
  ...
} | symlink('secret','foo');
| sys_xbegin(); | symlink('secret','foo');
| sys_xend(); | symlink('secret','foo');

- No deterministic solution without changing API
- 600+ hits in National Vulnerability DB for "symlink attack"
- Solved deterministically with transactions:

Developers Need Transactions
System transactions synchronize access to system resources
- Simple API: sys_xbegin, sys_xend, sys_xabort
- Transaction wraps a group of system calls
  - Results isolated from system until commit
  - Interfering operations automatically serialized
  - Atomic and isolated access to local resources
  - Support for files, memory allocation, process creation, etc.
  - Network, graphics, etc. left for future work
  - Previous systems hit implementation challenges, compromised isolation

Implementation Overview
TxOS: System transactions in Linux 2.6.22.6
- How are updates isolated?
  - Previous systems used in-place updates, undo log
  - Issues with priority inversion waiting for long aborts
  - TxOS operates on private copies of objects
  - Avoids priority inversion; keeps data structures consistent
  - Split objects into header and data component
  - Commit updates with a single pointer swap per object
- How are updates isolated?
  - Previous systems use two-phase locking (2PL)
    - 2PL is deadlock prone; can't order lock acquisition
  - TxOS updates private copies, eliminating deadlock
- Locks only held to make copies and commit

Useful Applications
Transactional Software Install
- sys_xbegin();
- sys_xbegin();
- open -i openssh;
- make install svn;
- sys_xend();
- sys_xend();
- 10% overhead
- 70% overhead

- With no code changes to installer:
  - A failed install is rolled back
  - If the system crashes, reboot to entire install or none
  - Concurrent applications see consistent libraries, config files

Lightweight Database Alternative
- Rename insufficient for middle ground, databases are overkill
- Case study: OpenLDAP directory server:
  - Replaced BDB backend with TxOS + flat files

Reasonable Overheads
- Overhead of using transactions ranges from 1-2.4x
  - 1.7-20x speedups for write-intensive workloads
- Non-transactional Linux compile: <2% overhead
- Individual, non-transactional system calls: 42% mean overhead
  - Can be reduced to 14% with better compilation support