Adding Software Transactions to the Kernel

By Michael J. Schultz with advising from Dr. Dennis Brylow

Transactional Memory

Transactional memory is intended to provide programmers with a lock-free mechanism that allows data reads and writes to occur atomically.

If we take the optimistic assumption that a set of reads and writes will succeed, then it is only when a conflict occurs that steps must be taken to correct the mistake.

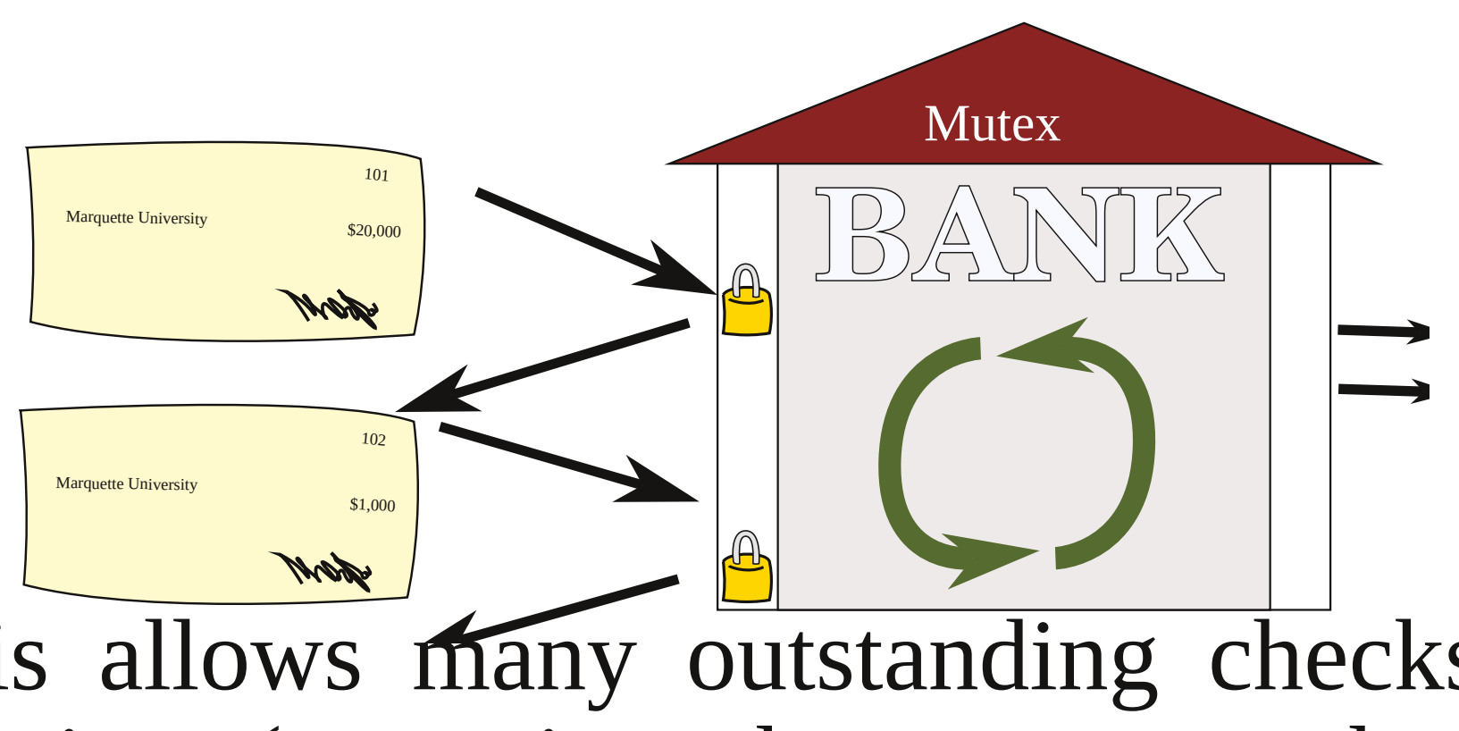
This approach makes thinking about program flow easier.

A simple analogy is issuing checks to a bank. Under mutual exclusion the following steps must occur:

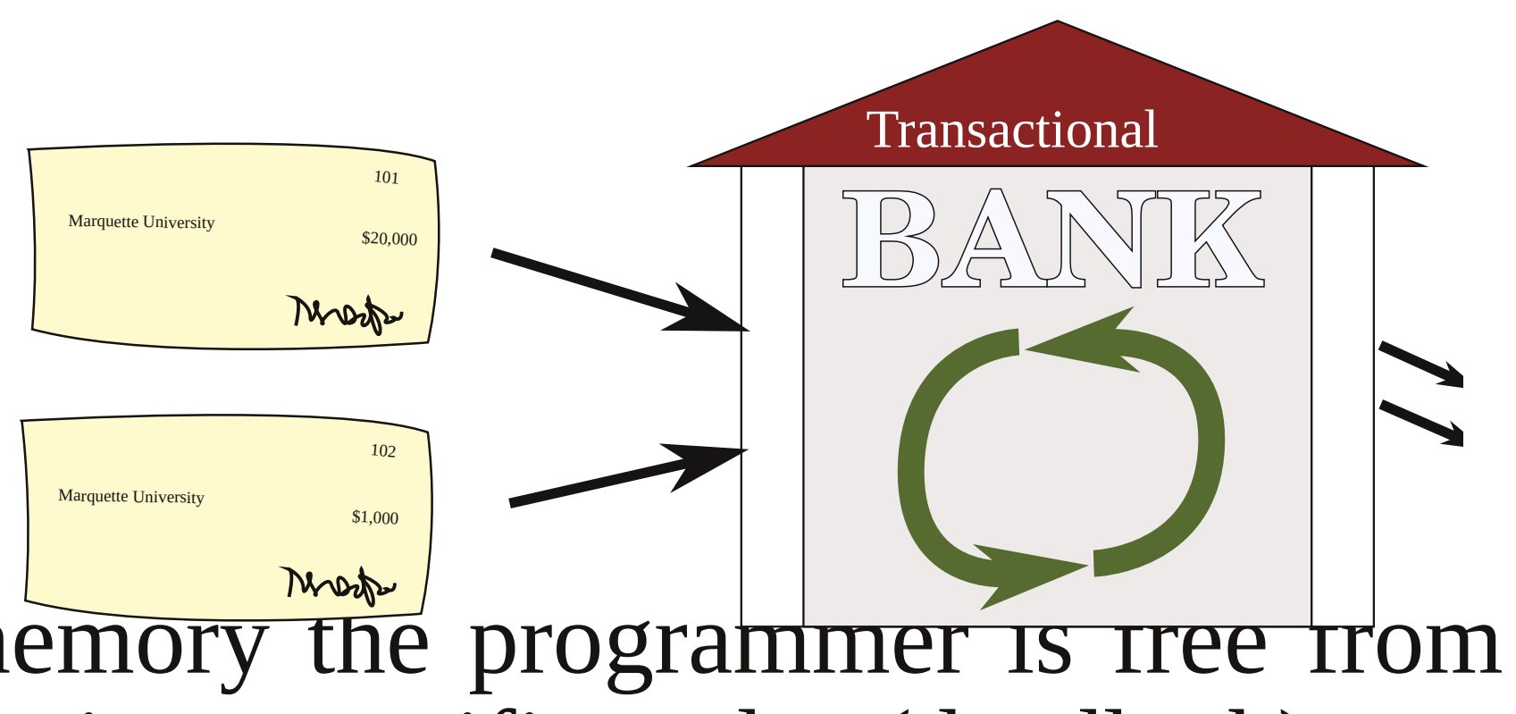
- issue a check;

- wait for the bank to process;

- receive confirmation (repeat).



Luckily, banks use transactions. This allows many outstanding checks while not preventing any from clearing (assuming the customer has sufficient money).



With transactional memory the programmer is free from having to worry about acquiring locks in a specific order (deadlock) or causing important components run incorrectly (priority inversion).

Transactional memory can be provided through hardware support or software libraries. Hardware systems place limits on the size and number of transactions. While software systems take more overhead and run slower.

Prototype C Compiler and Library

Recently Intel research has developed and released a prototype C compiler and library with support for software transactional memory.

Their compiler adds keywords to the C language that delineate atomic sections of code.

```
wait(a_sem);
wait(b_sem);
global_a++;
global_b++;
signal(b_sem);
signal(a_sem);
```

```
__tm_atomic {
    global_a++;
    global_b++;
}
```

To make sure reads get the most recent data, the transactions keep a log of old values to roll back to if a conflict occurs.

Naturally, these simplifications do not come for free. There are two upfront costs:

1. The binary image must now contain and support the runtime library that the compiler depends on, and

2. At compile time, the code generated for `__tm_atomic` sections is larger in size than the corresponding lock based code.

However, this is a prototype compiler so these size considerations may be reduced in the future.

Embedded Xinu Kernel

Xinu was developed by Dr. Douglas Comer a simple and elegant tool that can be used as a teaching tool for students and as a launching point for research in operating systems.

At its heart, Xinu is an interrupt-driven, multi-threaded kernel optimized for use on an embedded system. This enables fine tuning of the system to demonstrate various concepts without being buried under millions of lines of code.

For this project we are targeting the x86 architecture. By choosing this architecture we can take advantage of the prototype Intel compiler.

Currently we also have an Embedded Xinu kernel that runs on Linksys Wireless Routers using the MIPS architecture.

In the future, we would like to write a simple software transactional memory library that uses the load-linked/store-conditional opcodes.

STM for the Kernel

By using an small operating system at the core we are able to explore how STM will work at the kernel level.

Our intentions is to apply software transactional memory to key portions of the Xinu kernel.

Device drivers and interprocess communication use typical locking mechanisms to ensure only one thread is communicating with a device or process. However these operations may also delay the arrival of interrupts in the system, creating jitter.

By using transactional memory we can be sure interrupts will not be delayed and can safely read or write shared memory segments. Less critical processes will then have to retry their write before continuing.

While we acknowledge that transactional memory will not solve all the problems of system development, we are interested in where it can help. We seek to discover:

- how transactional memory can improve responsiveness;

- under what conditions will transactional memory fail, and why;

- is the additional overhead (in performance and code size) worth it?

References

Maurice Herlihy and J. Eliot B. Moss. ``Transactional Memory: Architectural Support for Lock-Free Data Structures." From the 1993 Proceedings of the International Symposium on Computer Architecture.

Christopher J. Rossbach, Owen S. Hofmann, Donald E. Porter, Hany E. Ramadan, Aditya Bhandari, and Emmett Witchel. ``TxLinux: Using and Managing Hardware Transactional Memory in an Operating System." From the 2007 Symposium on Operating System Principles.

Bratin Saha, Ali-Reza Adl-Tabatabai, Richard L. Hudson, Chi Cao Minh, and Benjamin Hertzberg. ``McRT-STM: A High Performance Software Transactional Memory System for a Multi-Core Runtime." From the 2006 Symposium on Prinicples and Practice of Parallel Programming.

This work is supported by the Wehr Foundation, Cisco Systems, the National Science Foundation, and viewers like you.