

Solaris System Call Emulation in a Complete Machine Simulator

(Brief version)

Bill Clarke,
(with Peter Strazdins, Andrew Over, Adam Czezowski)

Department of Computer Science,
Australian National University,
and
The CC-NUMA Project (ANU/Sun/Gaussian)

<http://cs.anu.edu.au/~Bill.Clarke/seminars/Solemn>

18 August 2003



THE AUSTRALIAN NATIONAL UNIVERSITY

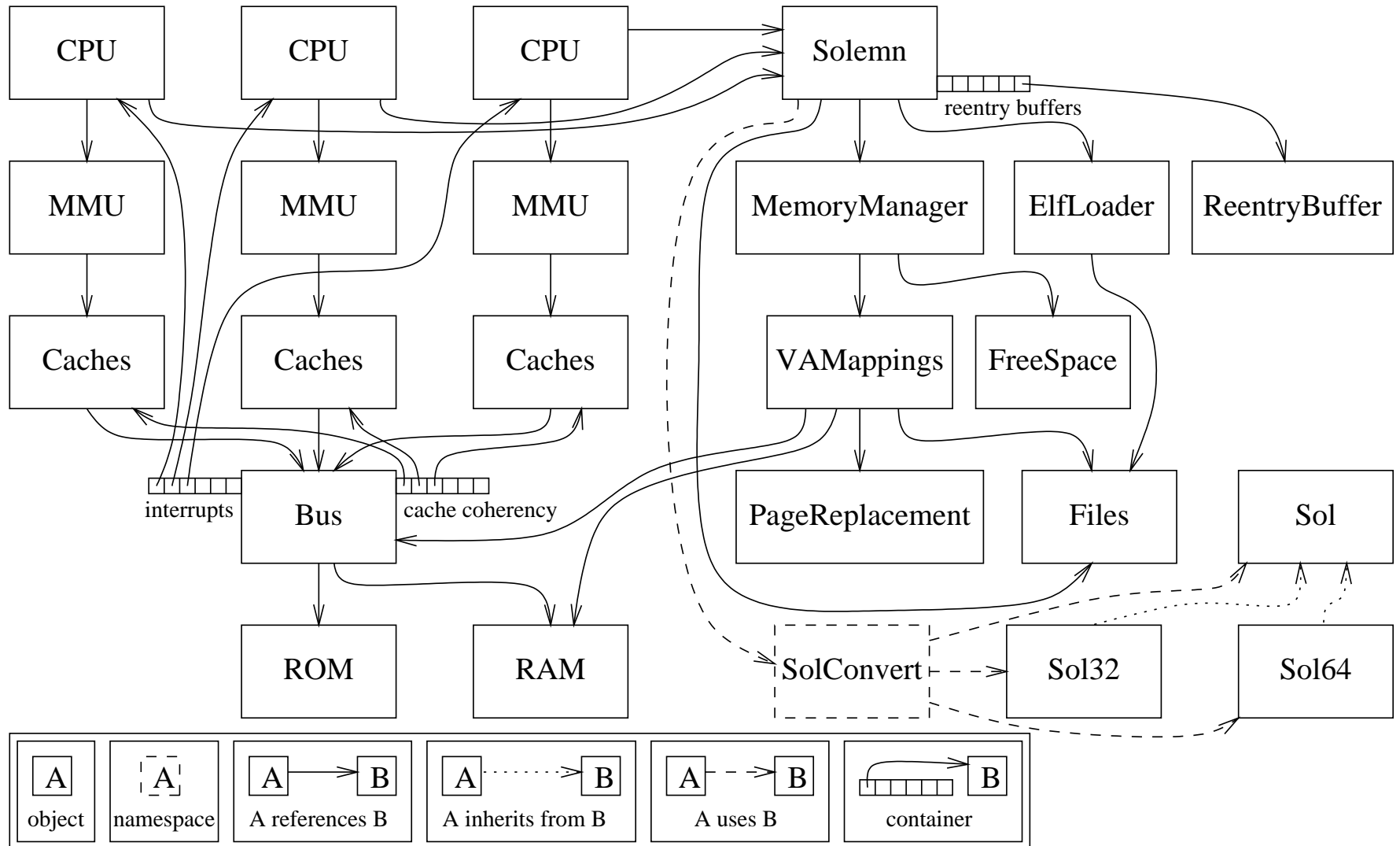
1 Why System Call Emulation?

- Already have OS-Emulation mode (UserSim):
 - has its own C-library (from RSIM, based on glibc)
 - quite limited in what system calls are provided (no mmap!)
 - results not identical to host executables (e.g., floating point printf)
- Complete machine boot so far unsuccessful:
 - not enough experience in low level boot sequence
 - we are still working on it, new student is keen and experienced
- Gain experience in internals of Solaris: useful for future full machine boot
- Implementing our own memory manager was an interesting challenge
- Easier to change architecture (e.g., MMU, cache sizes) than with complete machine simulation (would likely require OS modifications)

2 Traps and System calls

- Solaris user programs communicate with the operating system call via system calls; these are implemented using the `TCC` instruction
- A `TCC` instruction contains a trap number (0 to 127)
- Solaris defines operating system actions given different trap numbers:
 - SunOS 4.x, Solaris 32-bit and Solaris 64-bit system calls
 - `%g1` determines the actual system call
 - `%o0 ... %o5` are the parameters
 - `%o0` (+ possibly `%o1`) are the return values
 - return value is an error number if the condition code register is set
 - flush/clean register windows, 32-bit get/set CCR, fast system calls (e.g., `gethrtime`, `gettimeofday`)
- Solaris defines 231 system calls (32 and 64-bit)

3 Structure of Sparc-Sulima with Solemn



4 System call emulation

- Solemn intercepts `TCC` instructions and emulates the effect
- Solemn uses 3 trap numbers to communicate with the nucleus
- Solemn currently implements 24 of the 231 Solaris system calls. e.g.,
 - process related: `exit`, `getpid`, `getuid`, `getgid`
 - file related: `read`, `write`, `open`, `close`, `stat`, `lseek`
 - memory related: `brk`, `mmap`, `munmap`, `memcntl`
- Some system calls are trivial to implement: `getpid`, `close`, `lseek`
- Many are a little harder, requiring re-entrant system call handlers: `read`, `write`, `open`, `stat`
- Some are much harder, requiring additional infrastructure: threads, signals
- Some do not make sense to implement for Solemn: `fork`

5 Nucleus, reentrant system calls, files, memory management

- Nucleus:
 - written completely in SPARC assembler, resides in ROM at RSTVaddr
 - contains power-on reset (boot) handler, and standard trap handlers
- Reentrant system call handlers:
 - Solemn communicates with the simulation's memory via the MMU
 - this can cause exceptions: data page miss or protection fault
 - host buffer and state stored in a ReentryBuffer until sys-call complete
- Files: wrapper around host standard IO, file descriptors
- Memory management: responsible for all handling memory-related
 - traps: page misses and protection faults; and
 - system calls: `brk`, `mmap`, `munmap`, ...
 - contains `FreeSpace` and `VAMappings` managers
- Need to convert to/from host and simulated types, structures and values
- Executable loading: uses memory manager to `mmap` parts of the file

6 Benchmarks, bugs, problems, and todo

- Matrix factorisation (Sun Blade 1000: 2×750 MHz US-III, 2 GB RAM, 8 MB E\$, Solaris 9)

options	500×500, bf=1				1000×1000, bf=1				1000×1000, bf=64	
	static		dynamic		static		dynamic		dynamic	
exe build	native	g++64	native	g++64	native	g++64	native	g++64	native	g++64
sim build	native	g++64	native	g++64	native	g++64	native	g++64	native	g++64
time (secs)	0.74	182	0.80	185	5.25	1364	5.37	1382	1.73	733
slowdown	1	246	1	231	1	260	1	257	1	424
MFLOPs	125	0.5	119	0.5	137	0.5	136	0.5	534	1.1

- Bugs:
 - 64-bit dynamically linked executables die partway through linking
- Problems:
 - Debugging: Sparc-Sulima is large: over 40000 lines of code, of which 6000 are Solemn
- Todo:
 - more system calls!
 - threads!
 - Gaussian kernels!