Distributed Shared Memory

Motivation

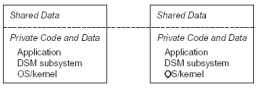
- Distributed memory hardware is cheaper
- Shared memory is “easier” to program
- Implement shared memory in software on top of distributed memory hardware

DSM

- Provided virtual address space that is accessible to every process
- Typically organized by pages
  - Also organized by Objects
- Managed similarly to VM
- Page consistency protocol manages movement of pages

Implementation overview

- DSM is a software layer between application and OS
- Each process has shared and private sections



Implementation overview

- Each shared variable is mapped into shared section
  - Has same address on all processes
- What happens on variable access?

Question

- We have been talking about "interleaving" of instructions
- What does that mean when multiple processors are doing the execution of different threads/processes?

Consistency Models

- Strict
- Sequential
- Processor
- Weak
- Release

Why is this important?

Uniprocessor Consistency

- Memory operations occur one at a time
- Operations occur (appear to occur) in program order
  - Some reordering of instructions allowed

Strict Consistency

- The value returned by a read operation must always be the value written by the most recent write
- Requires global timing

Example: Strict Consistency

P1: W(x)1

P2: R(x)1 R(x) 1

P1: W(x)1

P2: R(x) 0 R(x) 1

Example: Strict Consistency

P1: W(x)1

P2: R(x)0 R(x) 1

Sequential Consistency

- Individual processors maintain *program order*
- All processes "see" a single (the same) order of memory operations
  - atomicity
- Weaker than strict consistency
  - No global time

Example: Sequential Consistency

P1: W(x)1

P2: R(x)1 R(x) 1

P3: R(x)1 R(x) 1

P4: W(x)2

Example: Sequential Consistency

P1: W(x)1

P2: R(x)1 R(x) 2

P3: R(x)1 R(x) 2

P4: W(x)2

Example: Sequential Consistency

P1: W(x)1

P2: R(x)1 R(x) 2

P3: R(x)1 R(x) 1

P4: W(x)2

Example: Sequential Consistency

P1: W(x)1

P2: R(x) 1 R(x) 2

P3: R(x) 2 R(x) 1

P4: W(x)2

Processor Consistency

- All write operations performed by a single process be viewed by others in the order they were performed
- Write operations performed by different processors may be viewed by others in different order

Example: Processor Consistency

P1: W(x)1

P2: R(x) 1 R(x) 2

P3: R(x) 2 R(x) 1

P4: W(x)2

Example: Processor Consistency

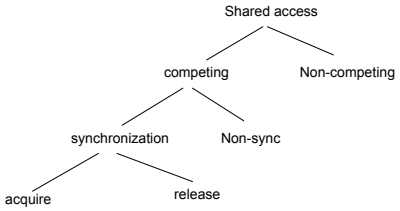
P1: W(x)1 W(x) 2

P2: R(x) 1 R(x) 2 R(x) 2

P3: R(x) 2 R(x) 1 R(x) 1

Weak Consistency

- Not all memory accesses are created equal



Weak Consistency

- Shared Access
  - Consistency issues are relevant only for shared variables
- Competing *vs.* Non-Competing
  - Result depends on which access occurs first
- Synchronizing *vs.* Non-Synchroning
  - Accesses used in synchronizing processes. Ordinary competing accesses (e.g., variable accesses) are non-synchronizing accesses
- Acquire *vs.* Release
  - Synchronization accesses divided into accesses that acquire locks, and accesses that release locks.

It's the synchronization, stupid!

- Accesses to synchronization variables are sequentially consistent
- No access to a synchronization variable is allowed to be performed until all previous writes have completed everywhere
- No data access (read or write) is allowed to be performed until all previous accesses to synchronization variables have been performed

Example: Sequential Consistency

P1: W(x)1 W(x)2 S

P2: R(x)0 R(x) 2 S R(x) 2

P3: R(x) 1 S R(x) 2

P4:

Release consistency model

- Before an ordinary access to a shared variable is performed, all previous acquires done by the process must have completed successfully
- Before a release is allowed to be performed, all previous reads and writes done by the process must have completed
- The acquire and release accesses must be processor consistent

Specifying Synchronization

- < await (B) S; >

Example

- Explain

```
int x = 0, y = 0;

process P1 {
   x = 1;
}

process P2 {
   y = 1;
}
```

Page fault handling

	Node 1	Node 2
initial state	`x = 0, y = 0`	
1.	write `x`, no fault	
2.		write `y`, page fault
3.		send request to Node 1
4.	send page to Node 2	
5.		receive page
6.		write y, no fault
final state		x = 1, y = 1

Page consistency protocols

- Migration protocol
  - Exactly one copy of a page anywhere
- Write invalidate protocol
  - Replicated pages for reading
- Write shared protocol
  - Allows multiple concurrent writers to a page
  - Merged at synchronization points

History

- Li & Hudak (1989)
- Munin (Carter, Bennett, Zwaenepoel 1991)
- Treadmarks (Amza et al, 1996)