Multiprocessor OS

CS 256/456

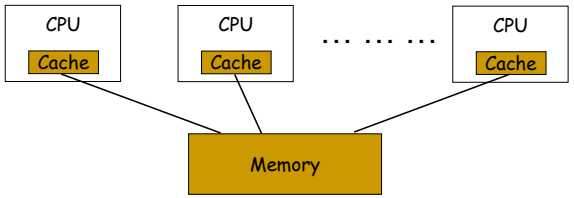
Dept. of Computer Science, University of Rochester

Comparisons on OS Extension/Customization

- Approaches
  - Exokernel
  - Downloading code into the kernel
  - The graybox approach
  - OS configuration
  - Virtual machine
- Comparison on
  - flexibility
  - how much changes in the original OS?
  - overall simplicity
  - overhead of each customization/extension

Multiprocessor Hardware

- A computer system in which two or more CPUs share full access to the main memory
- Each CPU might have its own cache and the coherence among multiple cache is maintained
  - Write operation by a CPU is visible to all other CPUs
  - writes to the same location is seen in the same order by all CPUs (also called write serialization)



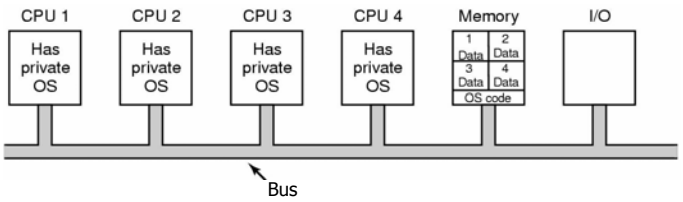
Multiprocessor Applications

- Multiprogramming
  - Multiple regular applications running concurrently
- Concurrent servers
  - Web servers, … …
- Parallel programs
  - Utilizing multiple processors to complete one task
  - Strong synchronization

Single-processor OS vs. Multi-processor OS

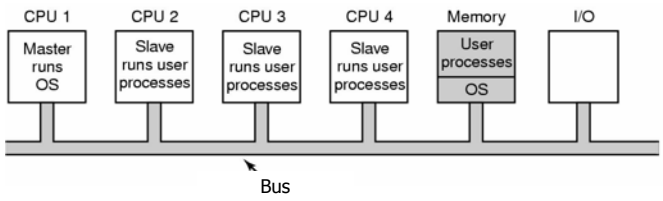
- Single-processor OS
  - easier to support kernel synchronization – why?
  - easier to perform scheduling – less complex.
- Multi-processor OS
  - OS structure
  - synchronization
  - scheduling

Multiprocessor OS



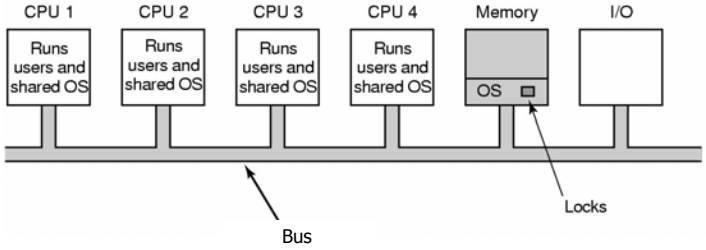
- Each CPU has its own operating system
  - quick to port from a single-processor OS
- Disadvantages
  - difficult to share things (processing cycles, memory, buffer cache)

Multiprocessor OS – Master/Slave



- All operating system functionality goes to one CPU
  - no multiprocessor concurrency in the kernel
- Disadvantage
  - OS CPU consumption may be large so the OS CPU becomes the bottleneck (especially in a machine with many CPUs)

Multiprocessor OS – Shared OS



- All CPUs run a single OS instance
- The OS itself must handle multiprocessor synchronization
  - have a big kernel lock – only one processor can execute in the kernel at a time
  - support fine-grain synchronization

Multiprocessor Kernel Synchronization

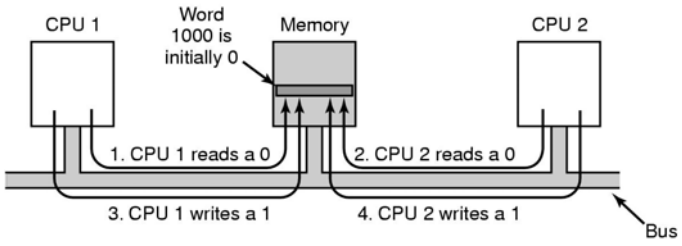
Protecting short critical region – busy waiting is OK

- Disabling interrupts does not work
- Software spin locks
- Hardware spin locks
  - using TSL

```
entry_section:
   TSL R1, LOCK        | copy lock to R1 and set lock to 1
   CMP R1, #0          | was lock zero?
   JNE entry_section   | if it wasn't zero, lock was set, so loop
   RET                 | return; critical section entered

exit_section:
   MOV LOCK, #0        | store 0 into lock
   RET                 | return; out of critical section
```

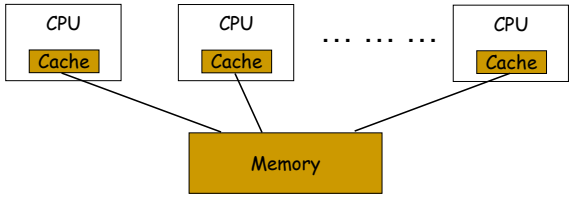

TSL on Multiprocessor



On multiprocessor, the TSL implementation is more complex, usually it has to lock the memory bus

More on TSL Locks

- Every TSL is a read/write, image multiple CPUs are busy waiting on one block, there will be a lot of traffic on the bus



- Precede each TSL lock will a trylock (basically a simple read)
  - only when trylock shows the lock is not locked, a TSL lock will be applied

Spin or Yield?

- Multi-processor synchronization.
  - A process is waiting for an event, triggered by another process.
  - Spinning wastes CPU cycles
  - Switching also uses up CPU cycles
  - Should it spin wait or yield the processor?

Multiprocessor Scheduling

- Timesharing
  - using a single wait queue (protected by synchronization) for scheduling
- cache affinity
  - affinity-based scheduling
- synchronization of parallel programs
  - gang scheduling

Disclaimer

- Parts of the lecture slides contain original work by Andrew S. Tanenbaum. The slides are intended for the sole purpose of instruction of operating systems at the University of Rochester. All copyrighted materials belong to their original owner(s).