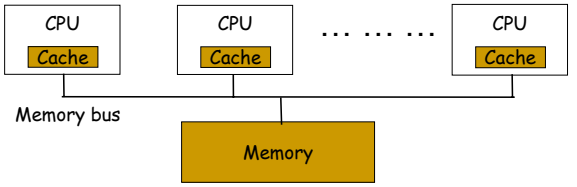
Multiprocessor OS

CS 256/456

Dept. of Computer Science, University of Rochester

Multiprocessor Hardware

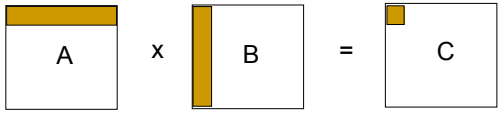
- A computer system in which two or more CPUs share full access to the main memory
- Each CPU might have its own cache and the coherence among multiple cache is maintained
  - write operation by a CPU is visible to all other CPUs
  - writes to the same location is seen in the same order by all CPUs (also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



  - bus snooping and cache invalidation

Multiprocessor Applications

- Multiprogramming
  - Multiple regular applications running concurrently
- Concurrent servers
  - Web servers, … …
- Parallel programs
  - Utilizing multiple processors to complete one task (parallel matrix multiplication, Gaussian elimination)

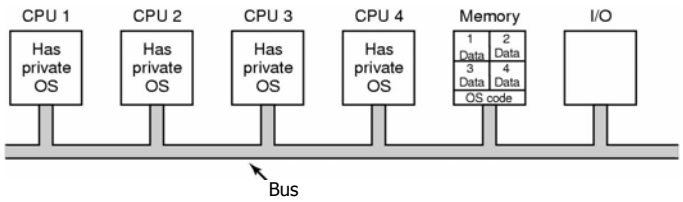


  - Strong synchronization

Single-processor OS vs. Multi-processor OS

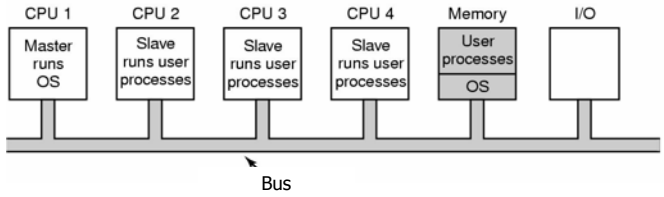
- Single-processor OS
  - easier to support kernel synchronization
    - fine-grained locking vs. coarse-grained locking
    - disabling interrupts to prevent concurrent executions
  - easier to perform scheduling
    - which to run, not where to run
- Multi-processor OS
  - evolution of OS structure
  - synchronization
  - scheduling

Multiprocessor OS



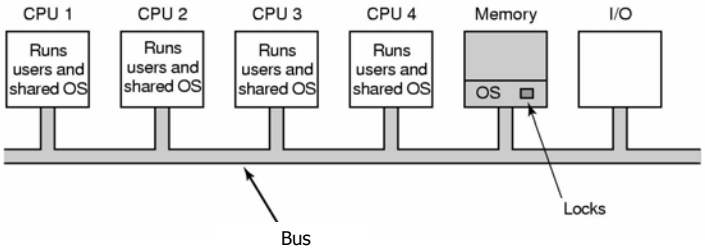
- Each CPU has its own operating system
  - quick to port from a single-processor OS
- Disadvantages
  - difficult to share things (processing cycles, memory, buffer cache)

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



- A single OS instance may run on all CPUs
- The OS itself must handle multiprocessor synchronization
  - multiple OS instances from multiple CPU may access shared data structure

Synchronization (Fine/Coarse-Grain Locking)

- Fine-grain locking – only locking necessary critical section
- Coarse-grain locking – locking large piece of code, much of which is unnecessary
  - simplicity, robustness
  - prevent simultaneous execution

Simultaneous execution is not possible anyway on uniprocessor anyway.

Synchronization (Spin Locking)

Protecting short critical region – busy waiting is OK

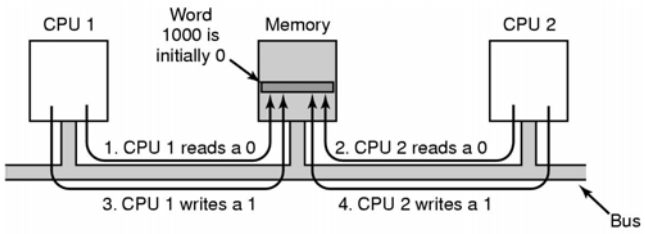
- Disabling interrupts does not work
- Software spin locks
- Hardware spin locks
  - using TSL

```
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
```

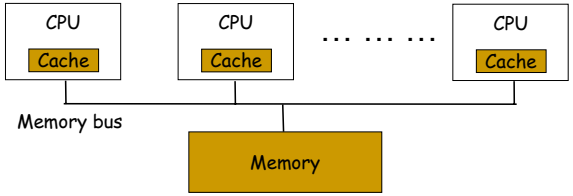

TSL on Multiprocessor

- TSL involves two actions (reading the value and writing 1) and it needs to be atomic
- On uniprocessor, we simply need to make the two actions unbreakable
- On multiprocessor, the TSL implementation is more complex, usually it has to monopolize the memory bus



More on TSL Locks

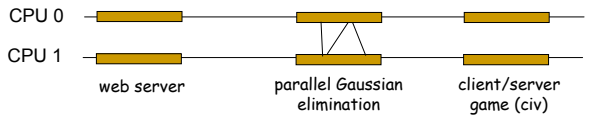
- TSL is a heavy operation, imagine multiple CPUs are busy waiting on one block, there will be a lot of load on the bus



- 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

Multiprocessor Scheduling

- Timesharing
  - similar to uni-processor scheduling – one queue of ready tasks (protected by synchronization), a task is dequeue and executed when a processor is available
- cache affinity
  - affinity-based scheduling – try to run each process on the processor that it last ran on
- caching sharing and synchronization of parallel/concurrent applications
  - gang/cohort scheduling – utilize all CPUs for one parallel/concurrent application at a time

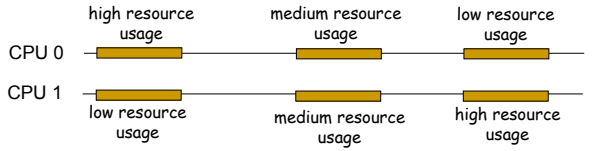


Resource Contention-Aware Scheduling I

- Hardware resource sharing/contention in multi-processors
  - SMP processors share memory bus bandwidths
  - Multi-core processors share L2 cache
  - SMT processors share a lot more stuff
- An example: on an SMP machine
  - a web server benchmark delivers around 6300 reqs/sec on one processor, but only around 9500 reqs/sec
- Contention-reduction scheduling
  - co-scheduling tasks with complementing resource needs (a computation-heavy task and a memory access-heavy task)
  - In [Fedorova et al. USENIX2005], IPC is used to distinguish computation-heavy task and memory access-heavy task

Resource Contention-Aware Scheduling II

- What if contention on a resource is unavoidable?
- Two evils of contention
  - high contention ⇒ performance slowdown
  - fluctuating contention ⇒ uneven application progress over the same amount of time ⇒ poor fairness
- [Zhang et al. HotOS2007] Scheduling so that:
  - very high contention is avoided
  - the resource contention is kept stable



Multiprocessor Scheduling in Linux 2.6

- One ready task queue per processor
  - scheduling within a processor and its ready task queue is similar to single-processor scheduling
- One task tends to stay in one queue
  - for cache affinity
- Tasks move around when load is unbalanced
  - e.g., when the length of one queue is less than one quarter of the other
  - which one to pick?
- No native support for gang/cohort scheduling or resource-contention-aware 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).