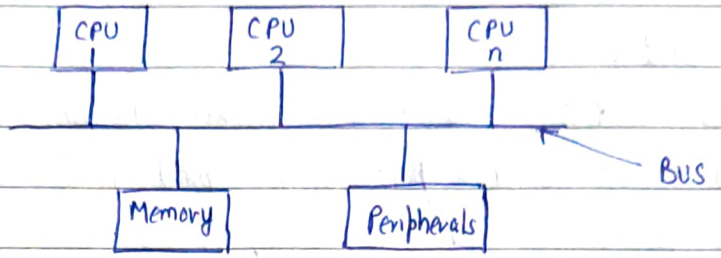
WRITEWELL
Date
Page

Multiprocessor Systems —:



- Multiple CPUs within single PC, sharing System Bus, Memory & I/O devices
- CPUs execute independently, but exec. one copy of kernel.
- Processes behave as on uniprocessor system — semantics of syscalls are same, but processes allowed migration b/w processors.
- Allows processes to work in parallel, increasing throughput, reliability & cost-saving behavior (less memory cost).

× Problems :
- Inconsistency due to simultaneous execution in different processors.
- Solutions :
  - Execute critical actions on one processor (master & slave arch.)
  - Serialize access to critical regions w/ lock. primitives (semaphores)
  - Redesign algorithms to avoid critical sections.

× Master & Slave Processors :
- Master processor handles syscalls & interrupts.
- Slave processors execute in user mode & notifies master of syscalls.
- Process table now has field to indicate processor ID to exec. on (eg. master / slave).
- Scheduling algorithm now checks if process may be run on slave for slave processors & assigns highest priority process to each processor based on status (master or slave)
- Syscall invocation now marks processor ID to that of master if executed on slave & replace process in ready queue.

WRITEWELL
Date
Page

- Once process finishes syscall, reset processor ID to any (slave or master). Also, if master processor is required for another process, perform context switch to assign this process to a slave.
- Clock intr handler periodically reschedules processes to prevent monopolization.
- Challenges —:
  - Scheduler algorithm does not protect against same process being selected for execution on multiple processors.
  - Load Balancing : One processor may have high # of processes assigned, while others are idle.
- Solutions —:
  - Master processor may specify slave processor for process execution, thereby acting as scheduler.
  - Further, master permits more than one process to be assigned to a processor.
  - Master kernel distributes process load b/w processors.

x Semaphores w/ Multiprocessors :
- Partition kernel into critical regions such that atmost one processor executes in critical region at a time.
- Issues : Implementation of semaphores
  Definition of critical regions.
- Sleep-Lock mechanism :

```
while lock set :                    free lock
    sleep (until lock is free);     wakeup processes sleeping on
set lock;                               condition lock is set.
```

lock ——→ unlock ——→

- Does not work on multiprocessor systems, as lock checked simultaneously by processors imply multiple processors lock a limited resource, leading to corruption due to incorrect lock state.

- Semaphore Init +ve value
  P : decrement (wait if value < 0)
  V : increment (signal to waiting processes)
  CP : Conditional P (decrement, returns true if value > 0.
  If value ≤ 0, unchange value, sleep until > 0)

- PPrim, VPrim : Processor locking primitives.
- Implementable using :
  - Software instructions (array of semaphore values)
  - HW instructions : read & clear. (atomic)

  read & clear : read value at location, clear value & set condition code as per original value.
- Lock : Control access to semaphore
- Value : Determine if process has access to critical region.
- Queue : Processes sleeping on semaphore.
- Semaphore now contains components : lock, value & queue.
- Problems : Infinite loops prevent preemption.

  Solutions : Updated algorithms for P & V using PPrim & VPrim.

× P : lock semaphore.lock using PPrim., next decrement semaphore value. If value is still >= 0, release lock (VPrim) & return. Else, attempt sleep : check signals, if signal to interrupt sleep, increment semaphore value & if signal caught in kernel, release lock in VPrim & return −1 (abnormal wakeup in kernel mode). Else release lock & longjmp for context switch. If no signals, enqueue process in semaphore queue, release lock (VPrim) perform context switch & check signals before return.

× V : lock semaphore (PPrim), increment value. If value is still 0 or < 0, dequeue from semaphore queue & make it ready to run (wakeup process). Lastly, release lock (VPrim).

WRITEWELL
Date
Page

while (value (semaphore) < 0) V (semaphore) ;

- Not equivalent to wakeup, as in wakeup, signals are sent to indicate to processes that resource is available, but V for multiprocessor directly assigns resource, leading to inconsistency. (non atomic)
- Conditional P : provide solution to possible deadlock situations.
  If CP is false, release held resources to prevent deadlock.

× Types of Multiprocessing —:

- Asymmetric (Master & Slave)
- Symmetric (Peer-Peer Relation)

- ASMP Challenges :
  - Performance Degradation (Master becomes Bottleneck)
- SMP :
  - Every processor has shared copy of OS.
  - On syscall, CPU receiving syscall traps to kernel to execute.
  - May pick same process for execution, preventable using mutexes.
  - Allows modularity in OS. (each processor runs non interacting part of OS).
  - Challenges :
    - Complex OS
    - Large main memory requirement.