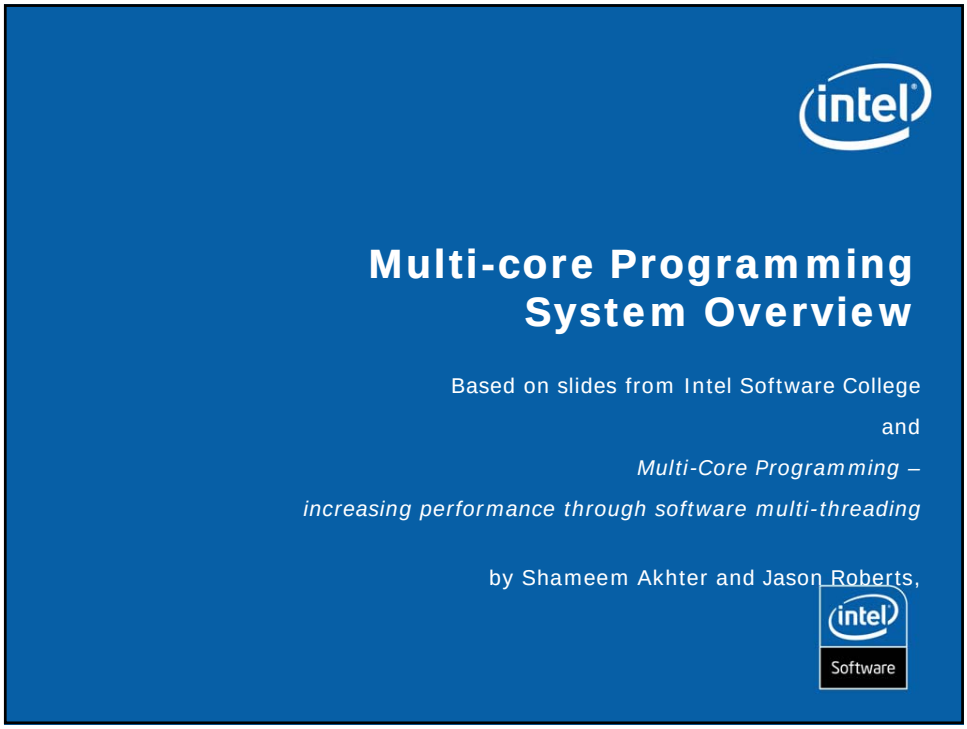
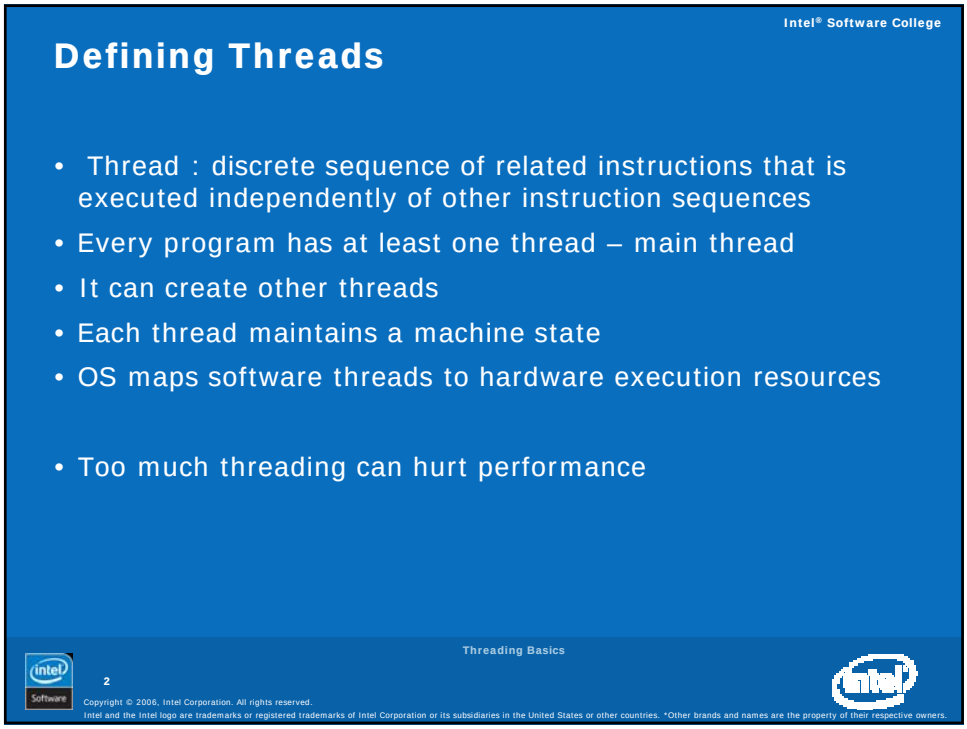
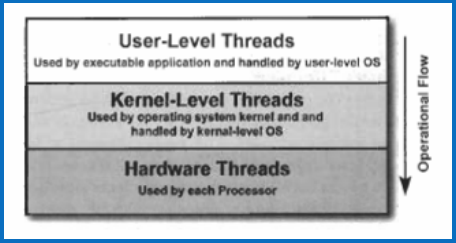

Multi-core Programming System Overview
Based on slides from Intel Software College
and
Multi-Core Programming –
increasing performance through software multi-threading
by Shameem Akhter and Jason Roberts,
Software


Intel® Software College
Defining Threads
• Thread : discrete sequence of related instructions that is executed independently of other instruction sequences
• Every program has at least one thread – main thread
• It can create other threads
• Each thread maintains a machine state
• OS maps software threads to hardware execution resources
• Too much threading can hurt performance
Threading Basics
2
Copyright © 2006, Intel Corporation. All rights reserved.
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. *Other brands and names are the property of their respective owners.

System View of Threads

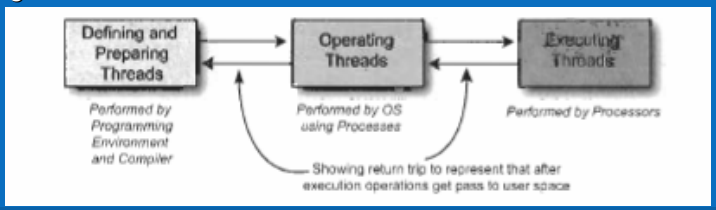
- User-level Threads
  - created in application software
- Kernel-level Threads
  - way OS implements threads
- Hardware Threads
  - how threads appear to execution resources



- A program thread is implemented by OS as a kernel-level thread and executed as a hardware thread

Threading above OS

- Can rely on run-time framework
  - Known as *managed code*
    - *managed environments* include Java Virtual Machine (JVM) and Microsoft's Common Language Runtime (CLR)
    - Do not provide scheduling – rely on OS
- If don't then thread creation is call to system API
  - executed as call to OS kernel to create thread
  - figure shows thread flow in this case



APIs for Threading

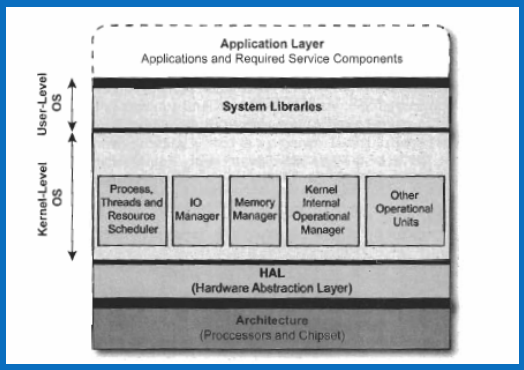
- Common APIs
- OpenMP
  - Ease of use
  - More developer friendly implementation
  - Requires compiler which supports – C/C++ or Fortran
- Explicit low-level threads such as pThreads and Windows threads
  - Requires significantly more code
  - Give fine-grained control
  - Only requires access to OS's libraries

Threading in OpenMP and pThreads

- Hello World
- OpenMP
  - No explicit thread creation
- pThreads
  - Thread created using pthread_create()

Threads inside OS

- Kernel maintains tables for processes and threads
- OpenMP and pThreads use Kernel level threads
- Windows supports both kernel and user level threads
  - Kernel level – more overhead but better performance
  - Windows user-level threads (fibers) – programmer must create management infrastructure and schedule threads manually

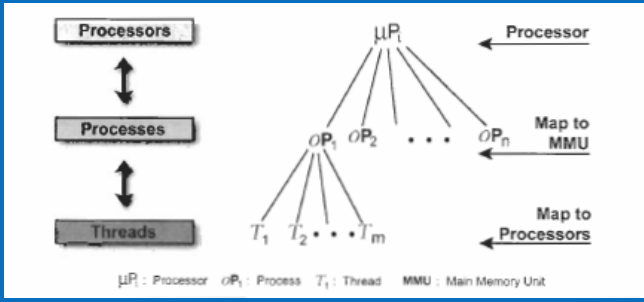


Processes

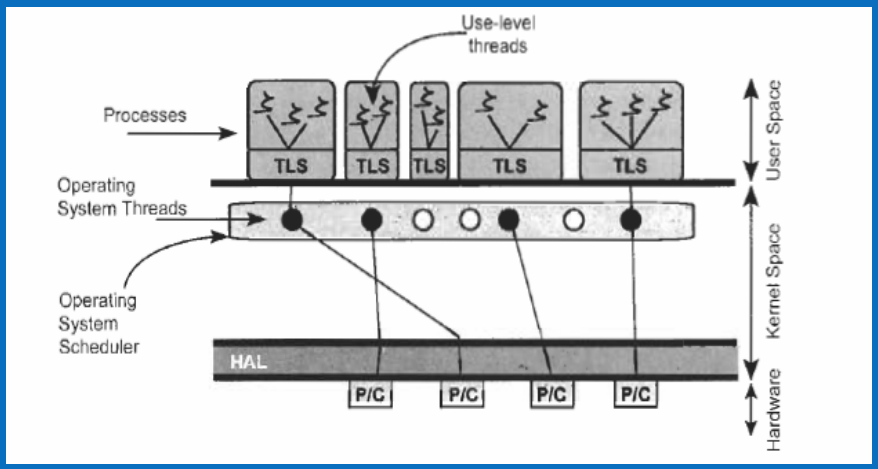
- Processes – course level execution units
  - Have own address space
- Processes can have multiple threads
  - Threads share address space
    - See same functions, data
    - If one thread changes data, all see
    - If one thread opens file, all can read from it
  - Each thread has registers, stack, some other data
  - Have simple inter-thread communication
- OS maintains
  - a Process Control Block (PCB) for each process
    - process identity, machine state, priority, address of virtual memory, file descriptors, user ID
    - in kernel space – requires system call (trap to OS) to access
  - A thread table for all threads (not normally per process table)

Processes and Threads

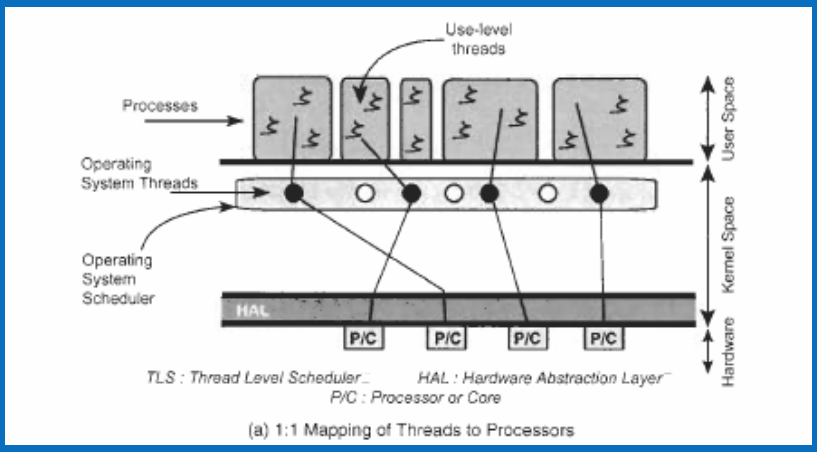
- Memory is allocated to processes by the Memory Management Unit (MMU)
- Threads are mapped to CPUs by scheduler
  - *processor affinity* – enables user to request specific CPU for thread. Not guaranteed that OS will grant.



Thread to Processor Mapping – 1:1

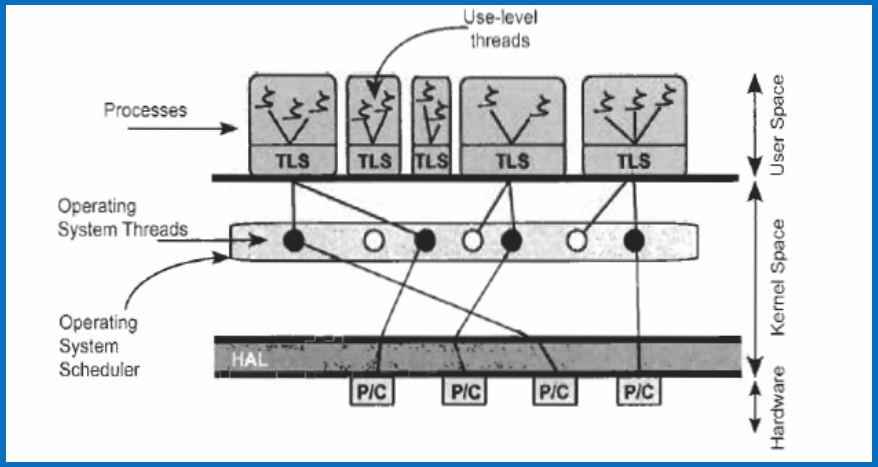


Thread to Processor Mapping – M:1



(a) 1:1 Mapping of Threads to Processors

Thread to Processor Mapping – M:N

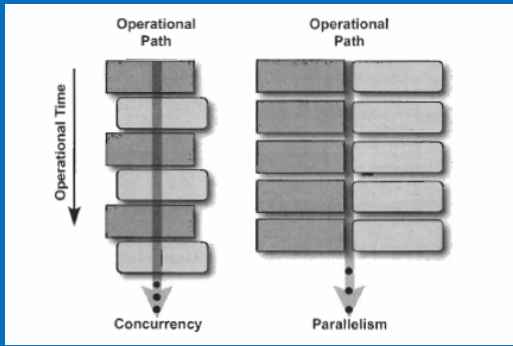


Thread to Processor Mapping

- 1:1
  - No thread-library scheduler overhead
  - OS handles scheduling
  - Called *preemptive multi-threading*
  - Linux, Windows 2000, Windows XP use this
  - Enables stronger handling of threads by OS
  - We will concentrate on this model
- M:1 many to one
  - thread-library scheduler decides which thread gets priority
  - Called *cooperative multi-threading*

Difference between Concurrency and Parallelism

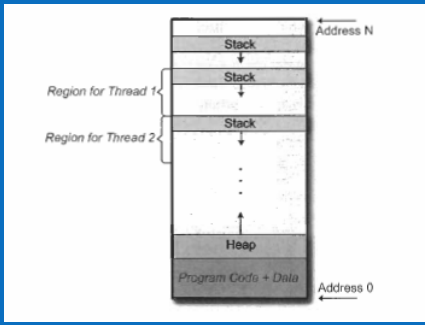
- Parallelism – Chip Multithreading (CMT)
  - multiprocessor, multicore
- Concurrency - Simultaneous Multithreading (SMT)
  - hyperthreading
- For true parallelism – program threads <= hardware threads



- Too many threads can slow performance

Thread Creation

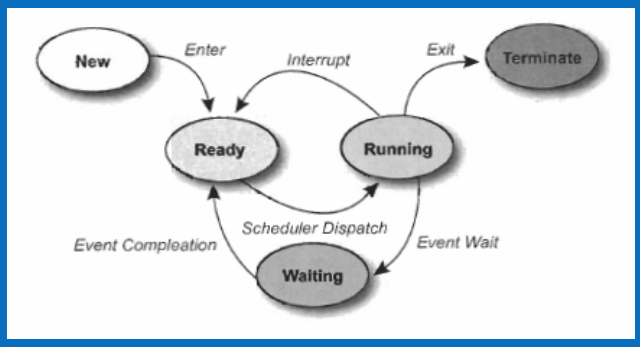
- Each thread has own stack space
- Default thread stack size varies with OS
- Need to be aware of this



Thread States

- Ready
- Running
- Waiting (Blocked)
- Terminated

•Substates give reason for entering e.g. blocked on I/O etc



Virtual Environments

- Virtualization : creating the appearance of a different set of resources
- Runtime virtualization e.g. Java VM (JVM) or MS CLR
  - Creates appearance Java appl running in private environment
  - JVM is container and executor appl
  - Create at least 3 threads for execution, garbage collection, just-in-time (JIT) compilation
  - Generally more threads for internal tasks
- System virtualization
  - Creates appearance of virtual machine with own independent OS

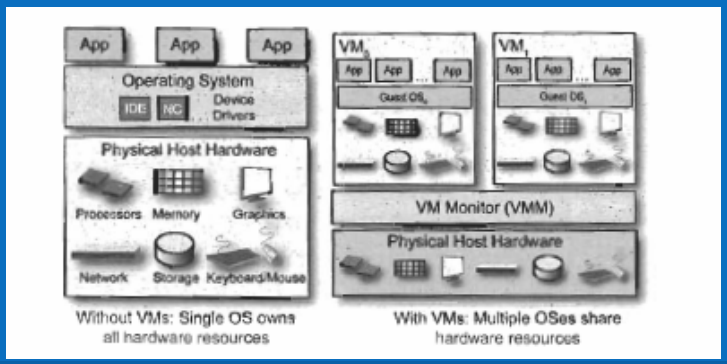
System virtualization

Complete virtual execution context

- virtualized NIC, disks, OS
- can be several VM on hardware

Virtual Machine Monitor (VMM) or hypervisor

- intermediate layer



VMM role

- VMM gives each VM illusion owns hardware
- presents virtual processors (VP) to guest OS on VM
  - Number equal to number of cores
- Creates isolation of instruction set architecture (ISA)
- What of privileged instruction ?
  - VMM and VM are applications – cannot execute
  - Priviliged insts are trapped by VM and call made to VMM
  - VMM may be able to handle or
  - may pass on to host OS, wait for response, emulate response in VM
  - Performance cost
    - to minimize Intel has provided extensions to ISA to allow VMM to efficiently execute priviliged insts
    - Part of Intel Virtualization Technology

Mapping Application threads

- VMM does very little
- When VM creates threads, handled by guest OS
- When guest OS schedules thread, virtual processor executes
- VMM does not match appl threads to particular cores
- Only time VMM is involved is when swaps out a VM or performs internal tasks
- Can be issue when thread is locked and waiting for thread on virtual processor that is swapped out
  - Thread has substantial delay – problem known as *lock-holder preemption*
  - One of problems that can arise in virtualization