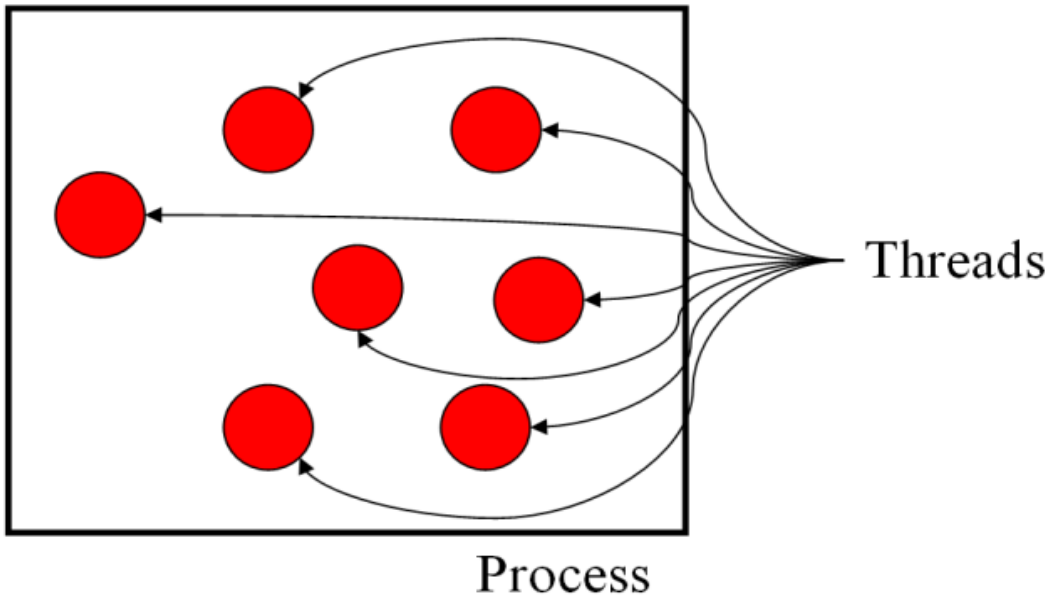
Operating System
Session 4

Hakim Sabzevari University
Dr.Malekzadeh

Threads

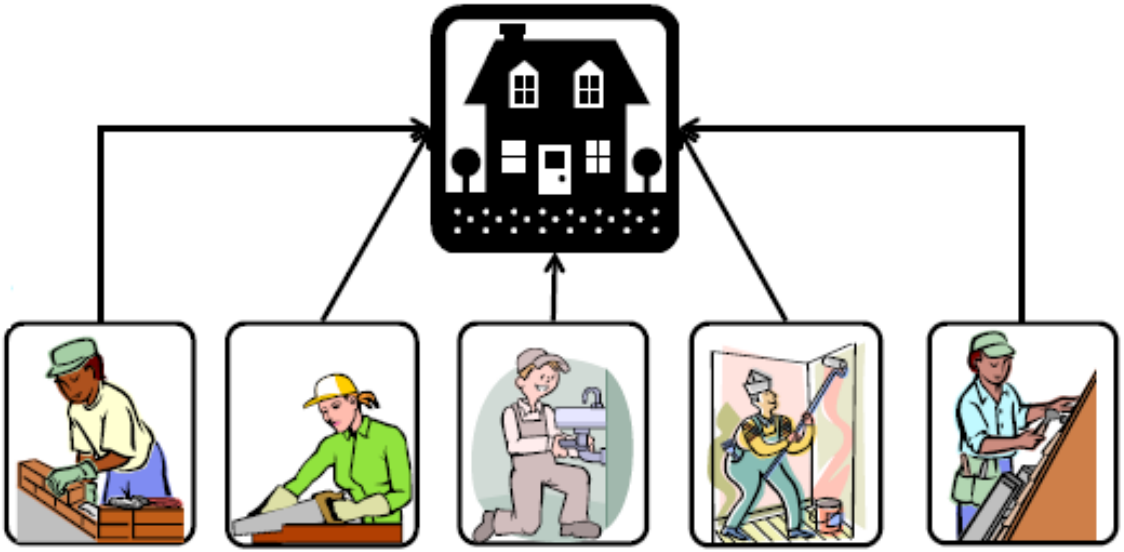
Thread

- Within a process, the thing, which is really executed, is the thread. A thread is a basic unit of CPU utilization within a process. Threads are a popular way to improve application performance through parallelism.
- The term "thread" refers to a "thread of control" or "thread of execution", meaning a sequence of instructions that are executed one after another.



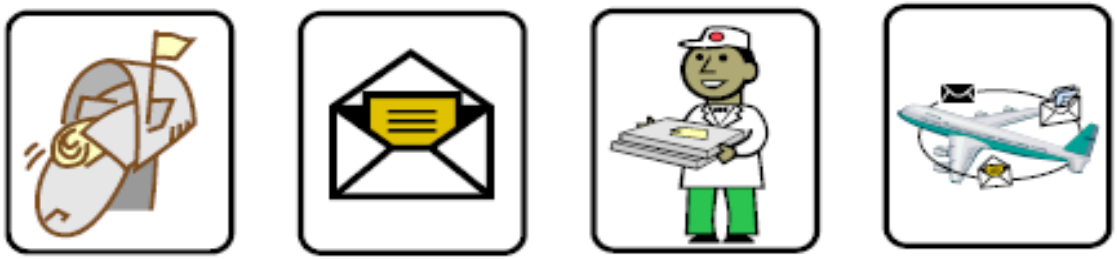
Threading for functionality

- Different people involved in building a house are:



Example: Building a House

Example: Setting Up Dinner Table



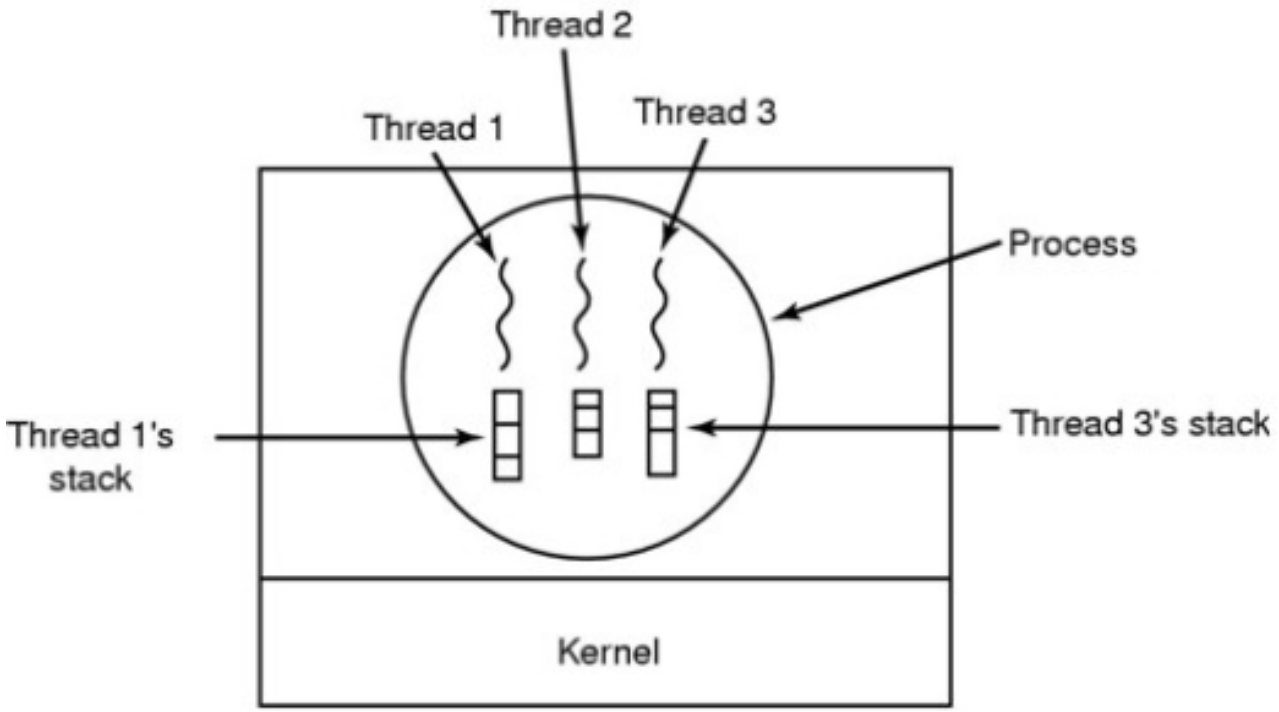
Example: Postal Service

Thread cont…

- Each thread has its own:
  - thread ID
  - program counter (PC)
  - register set
  - stack
- Thread shares the following with other threads within the same process:
  - code section
  - data section
  - heap (dynamically allocated memory)
  - open files and signals

Thread cont…

- Thread states: running, blocked, or ready



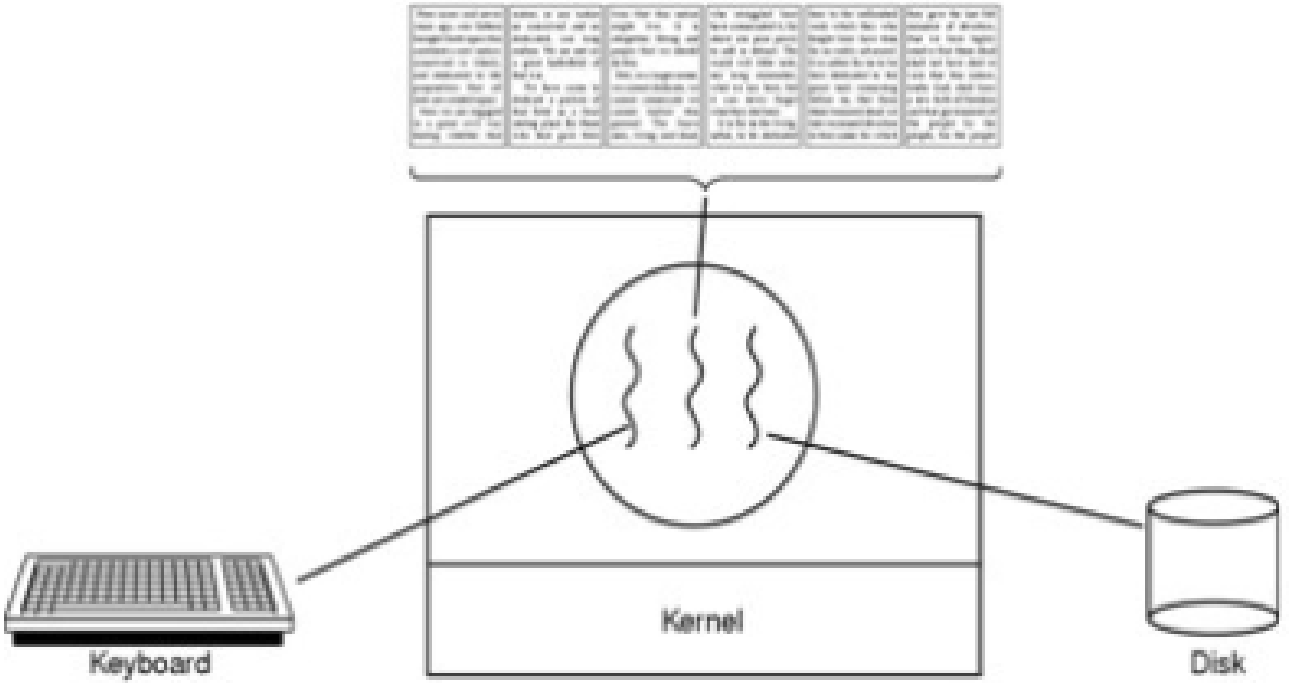
Why using threads?

- Threads are very useful in modern programming whenever a process has multiple tasks to perform independently of the others.

- For example in Microsoft word, a background thread may check spelling and grammar while a foreground thread processes user input (keystrokes), while yet a third thread loads images from the hard drive, and a fourth does periodic automatic backups of the file being edited.

Thread usage

- Example: word processor
  - Display contents
  - Edit (typing)
  - Reformats
  - Auto save
  - Printing
  - Etc.



Thread analogy

- To illustrate how threads and processes work, we establish an analogy.

- Let's base our analogy for processes and threads using a regular, object as a house. A house is really a container, with certain attributes (such as the amount of floor space, bedrooms, bathrooms, and so on). If you look at it that way, the house really doesn't actively do anything on its own. This is effectively what a process is.

- The people living in the house are the active objects -- they're the ones using the various resources including rooms, watching TV, cooking, taking showers, and so on. They are threads.

Thread analogy cont…

- Just as a house occupies an area of city, processes occupy memory.
- Just as people in house are free to go anywhere in the house, all threads of a process have common access to **that part of memory**.
- If a thread allocates something (mom goes out and buys something), all the other threads immediately have access to it (because it's present in the common address space -- it's in the house).
- Likewise, if a process allocates memory, this new memory is available to all its threads as well.

Threads are execution paths of the same process, and the heap actually belongs to the process (and as a result shared by the threads). Each threads just needs its own stack to function as a separate unit of work.

Single threading

- If you live alone in the house, you know that you can do anything you want in the house at any time, because there's nobody else in the house. If you want to turn on the stereo, use the washroom, have dinner -- whatever -- you just go ahead and do it.

- Some systems have only one CPU in the system. In this case, since there's only one CPU present, only one thread can run at any given point in time. The kernel decides (scheduling) which thread to run.

Multithreading cont…

- Things change dramatically when you add another person into the house. You can't just go into the washroom at any given point; you need to check first to make sure your friend isn't in there!

- If a process has multiple threads of control, it can perform more than one task at a time.

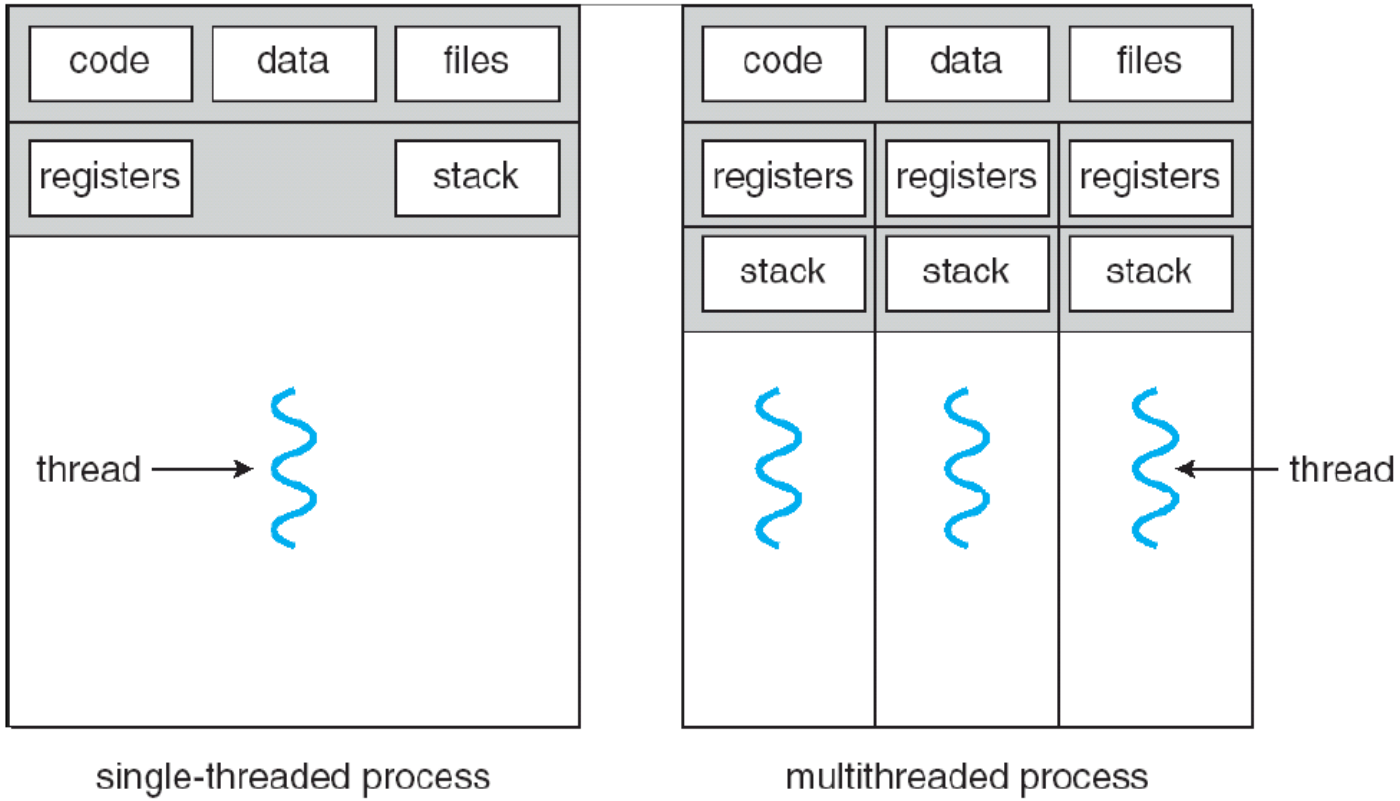
- Multithreading allows multiple threads to exist within a single process. These threads share the process's resources, but are able to execute independently.

Multithreading: multiple threads in the same process

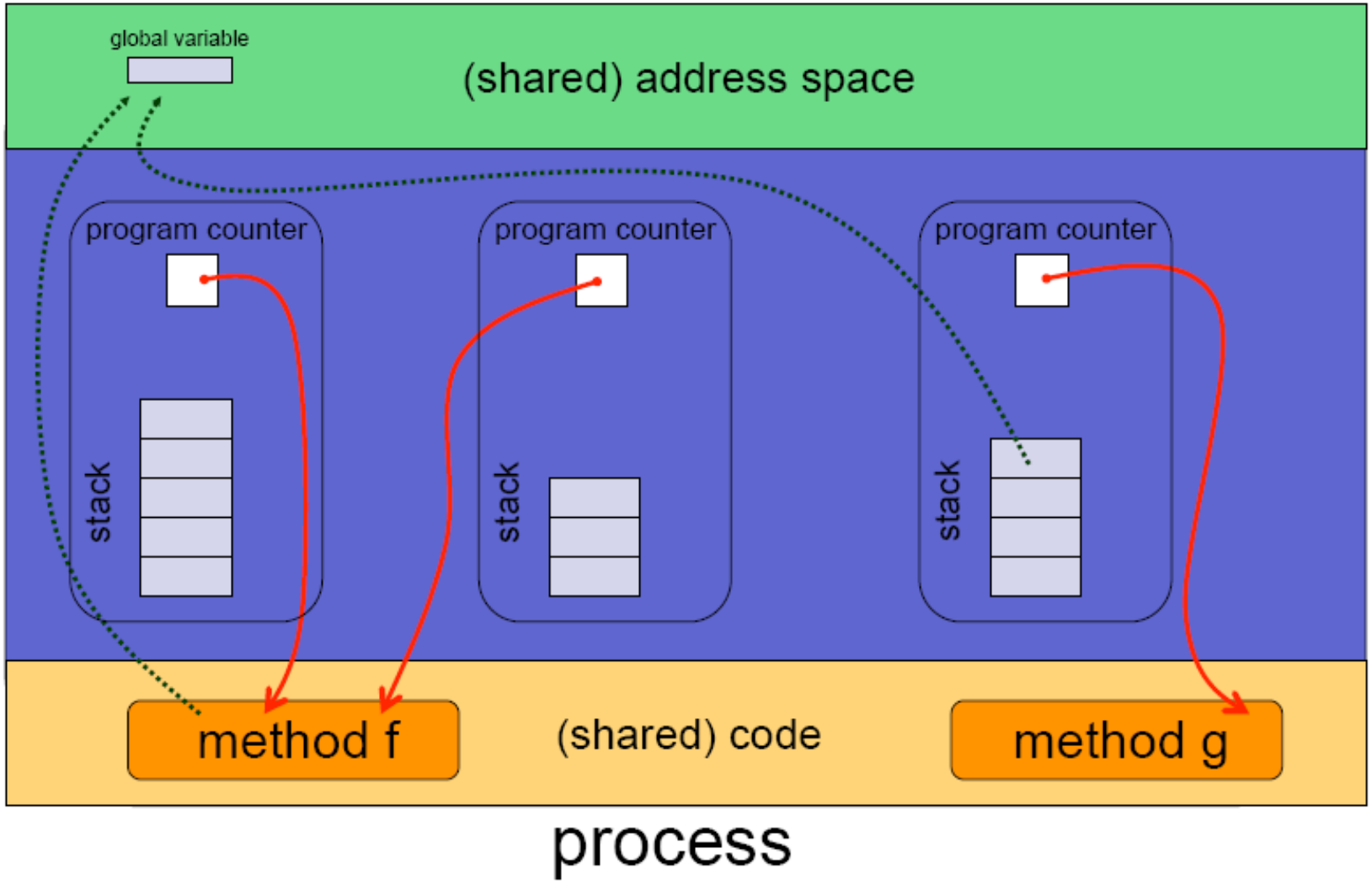
Multithreading cont…

- In Multi threading, some information cannot be shared, such as the program counter, stack (stack pointer to a different memory area per thread), registers and thread-specific data. This information allow threads to be scheduled independently.

difference between a traditional single-threaded process and a multithreaded process

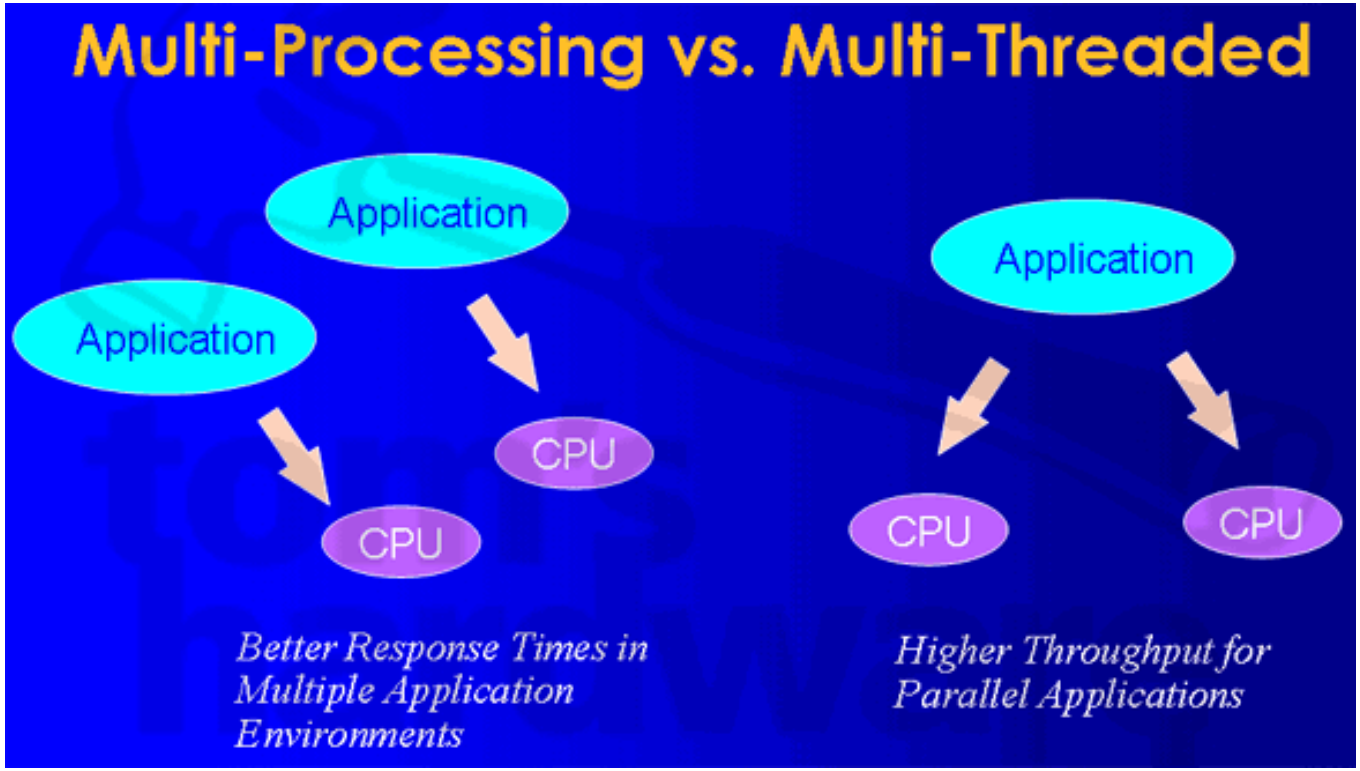


Multi threading cont…



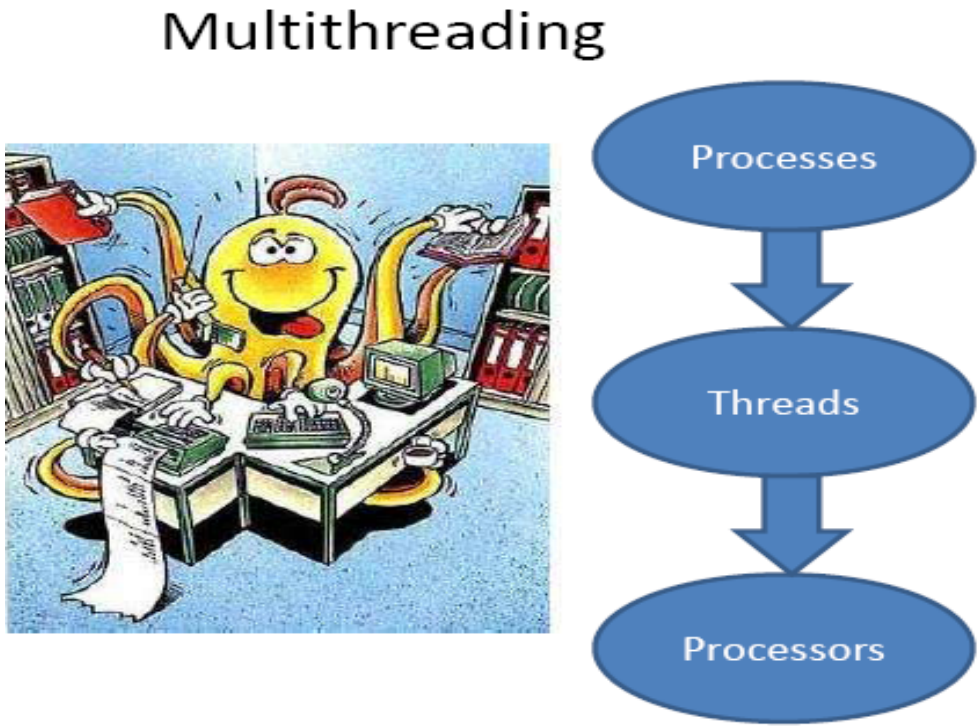
Multi threading cont…

- Communication between processes (IPC) is resource-intensive.
- Intra-process thread communication is less expensive than inter-process communication.



Multi threading cont…

- The multi-processor architectures can benefit from thread more than single CPU systems.
- Since each CPU can execute only one thread at a time, with multiple CPUs, multiple threads can execute simultaneously. In this case, the number of threads that can run simultaneously are limited by the number of CPUs.



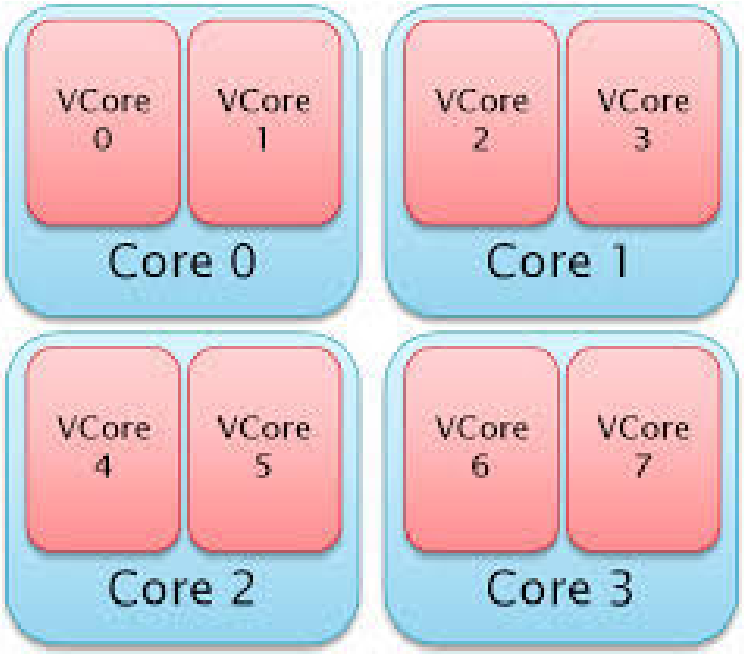
Multi threading cont…

- Any sequential process that cannot be divided into parallel task will not benefit from thread, as they would block until the previous one completes. Example is like a program that reads input, process it, and outputs.

- In general, any program that can do more than one task at a time could benefit from multi-threading.

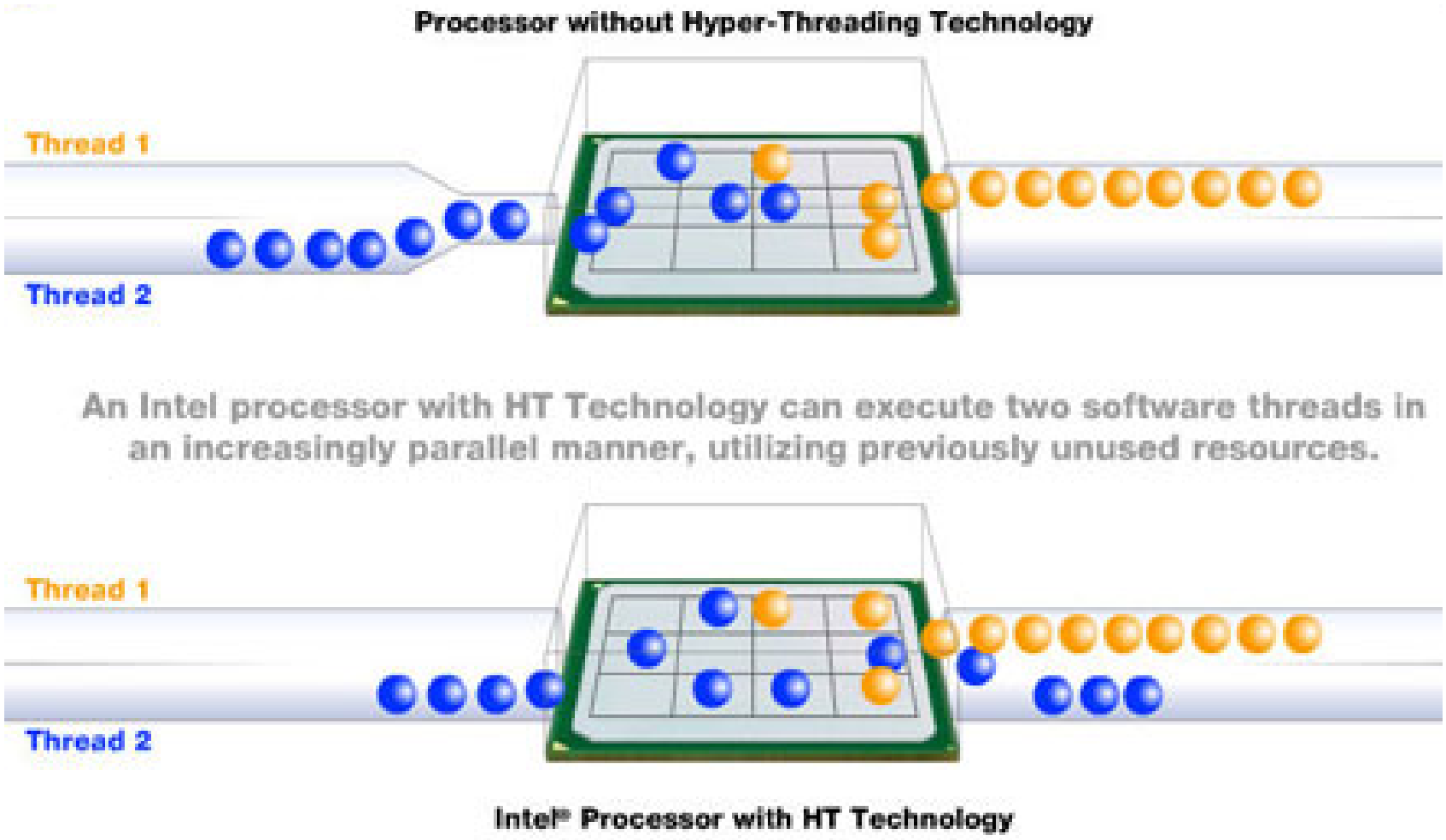
Hyper-Threading Technology

- Hyper-Threading Technology (HTT) is Intel's implementation of simultaneous multi-threading technology (SMT).
- HTT presents each physical core as two logical/virtual cores to OS.
- With HTT, now each core of the processor can work on two tasks at the same time.



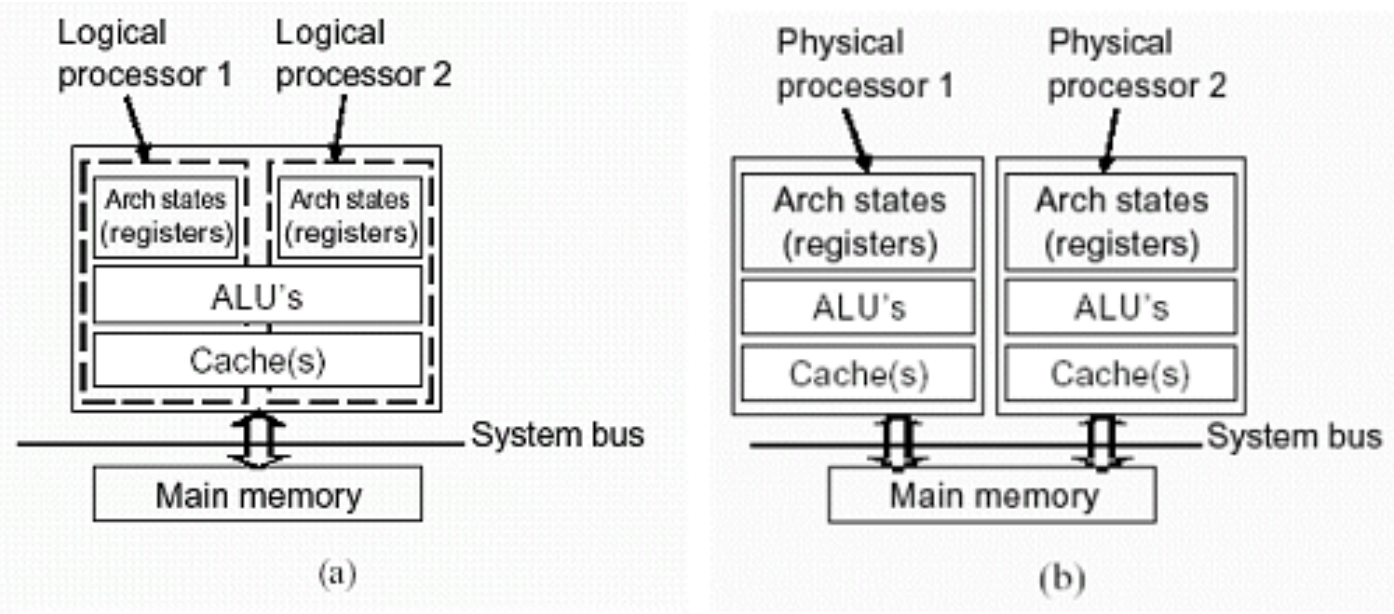

```
Logical Cores = # of Physical Core *  Thread on each core


Therefore, Logical Cores = 4 * 2 = 8 cores
```

HTT cont…

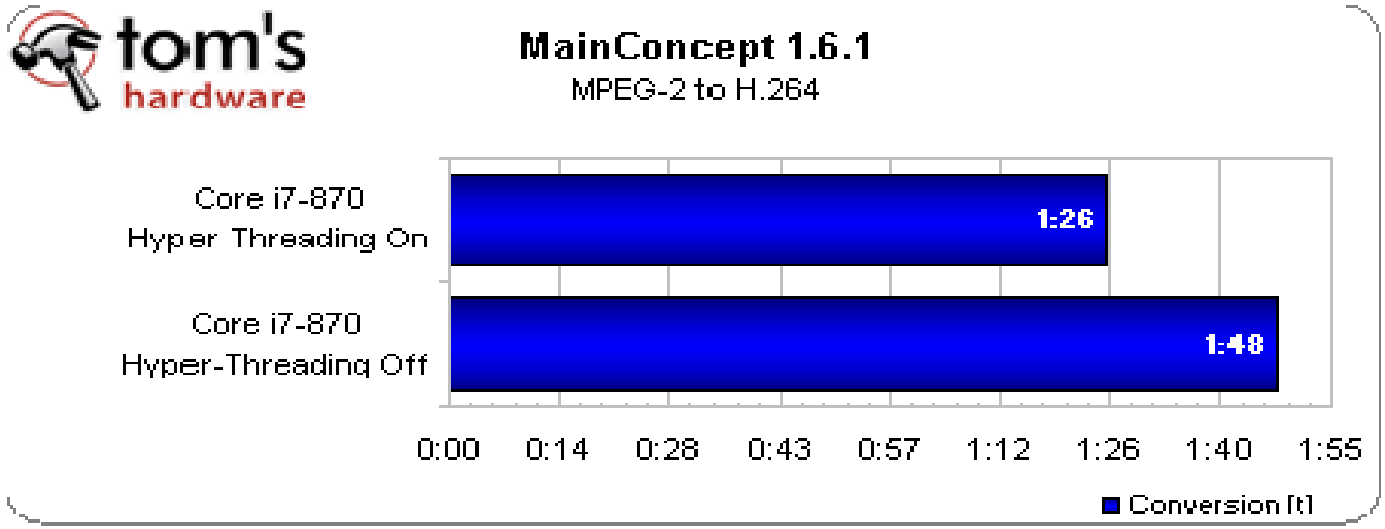

Processor without Hyper-Threading Technology
Thread 1
Thread 2
An Intel processor with HT Technology can execute two software threads in an increasingly parallel manner, utilizing previously unused resources.
Thread 1
Thread 2
Intel® Processor with HT Technology

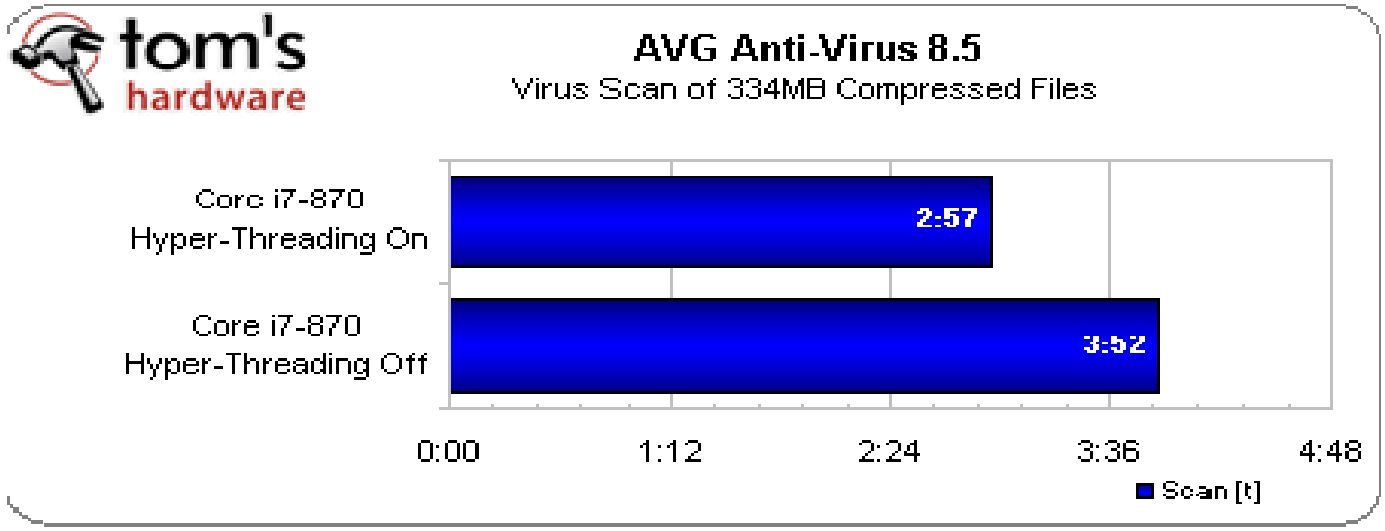
HTT cont…

- The basic idea is that a "multi-threaded" core contains multiple copies of some—but not all—of the hardware necessary to execute a program. The extra hardware allows the CPU core to execute multiple threads or processes more efficiently, but without the expense of adding a whole extra CPU core.
- So, from an OS perspective, it looks like the CPU has twice as much virtual cores as it has physical cores.
- This doesn't mean you suddenly have the equivalent of an eight-core processor. Instead, Intel duplicates certain resources in each physical core to make the technology available;

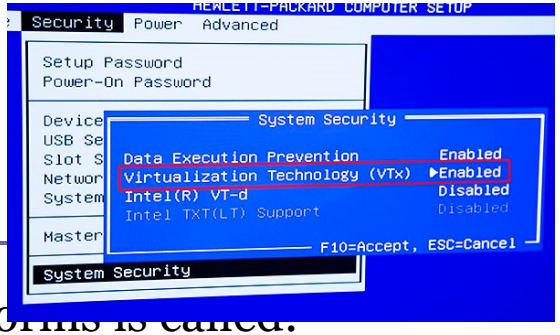


HTT advantage





Virtualization Technology



- Virtualization Technology for two different CPU platforms is called:
  - Intel → Intel Virtualization Technology (VT, VT-x, VT-i ,vmx)
  - AMD → AMD Secure Virtual Machine (SVM, AMD-V)

- If your CPU does not support one of these virtualization technologies, then you cannot use virtual machine acceleration. Also because an emulator (like Android emulator) runs inside an accelerated virtual machine, so you cannot run emulator on that system.

Overclocking

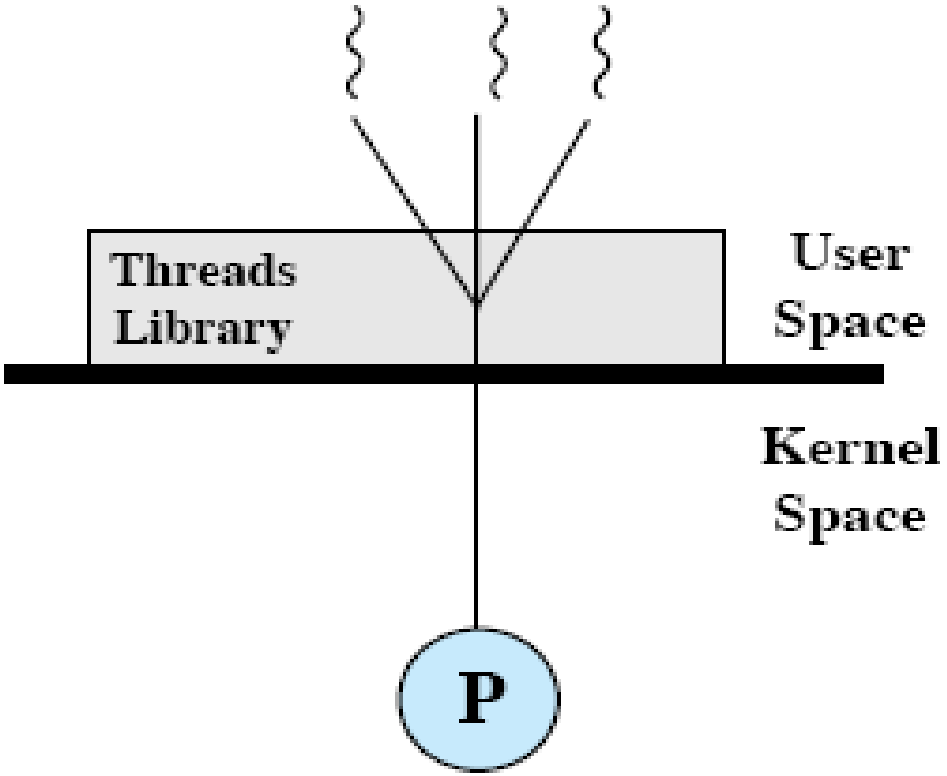
- Over clocking means increasing clock rate of a component (CPU, GPU, etc), running it at a higher speed than they were designed for. The over-clocking for Intel is called turbo boost.
- CPUs can be **overclocked by increasing the clock multiplier.**
- The phrase *clock doubling* implies a clock multiplier of two.
- Keep in mind modern computers and laptops have become so advanced and powerful that overclocking may not even produce noticeable results.

Types of Threads

- Threads can be implemented in two ways:
  - User Level Threads (ULT): their management is done by user-level threads library (Pthreads, Win32 threads, Java threads)
  - Kernel Level Threads (KLT): their management is done by the kernel/OS

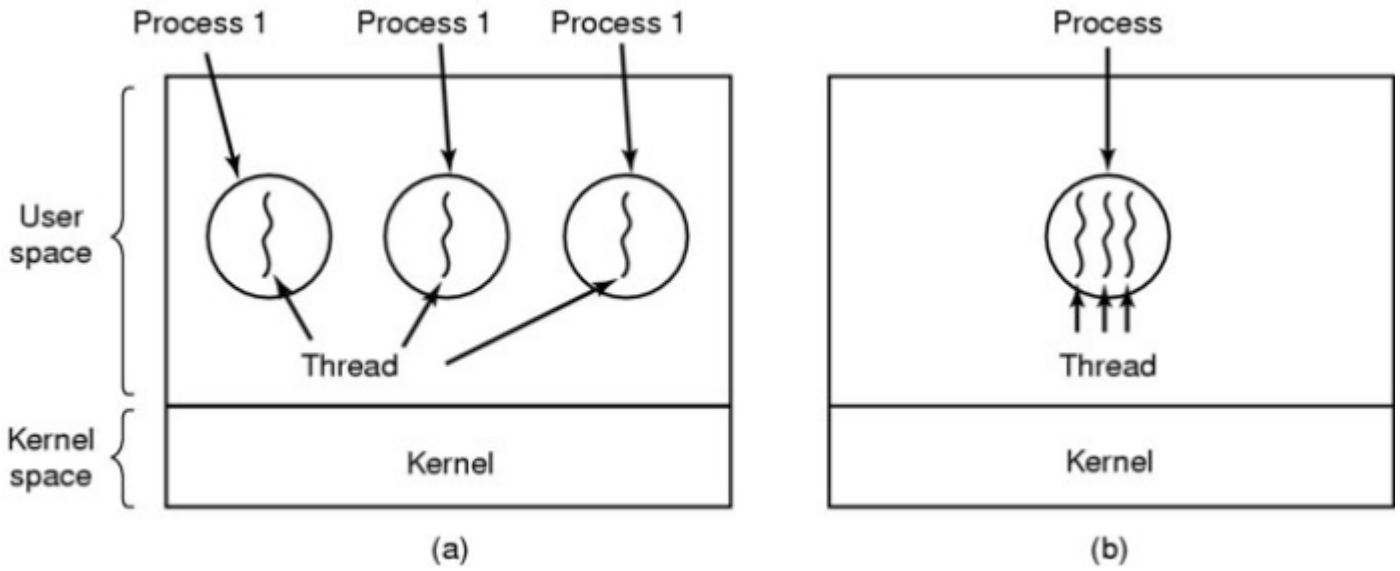
User level thread

- The kernel is not aware of the existence of threads and manages them as if they were single-threaded processes. It means kernel only sees the parent process (the process containing the threads).



Multi threading

- Multithreading: multiple threads in the same process.
- using System.Threading.Tasks;



User level thread cont…

- The OS only schedules the process, not the threads within the process. Thus, the thread management is done by the application using a thread library. There is a thread table in each process.

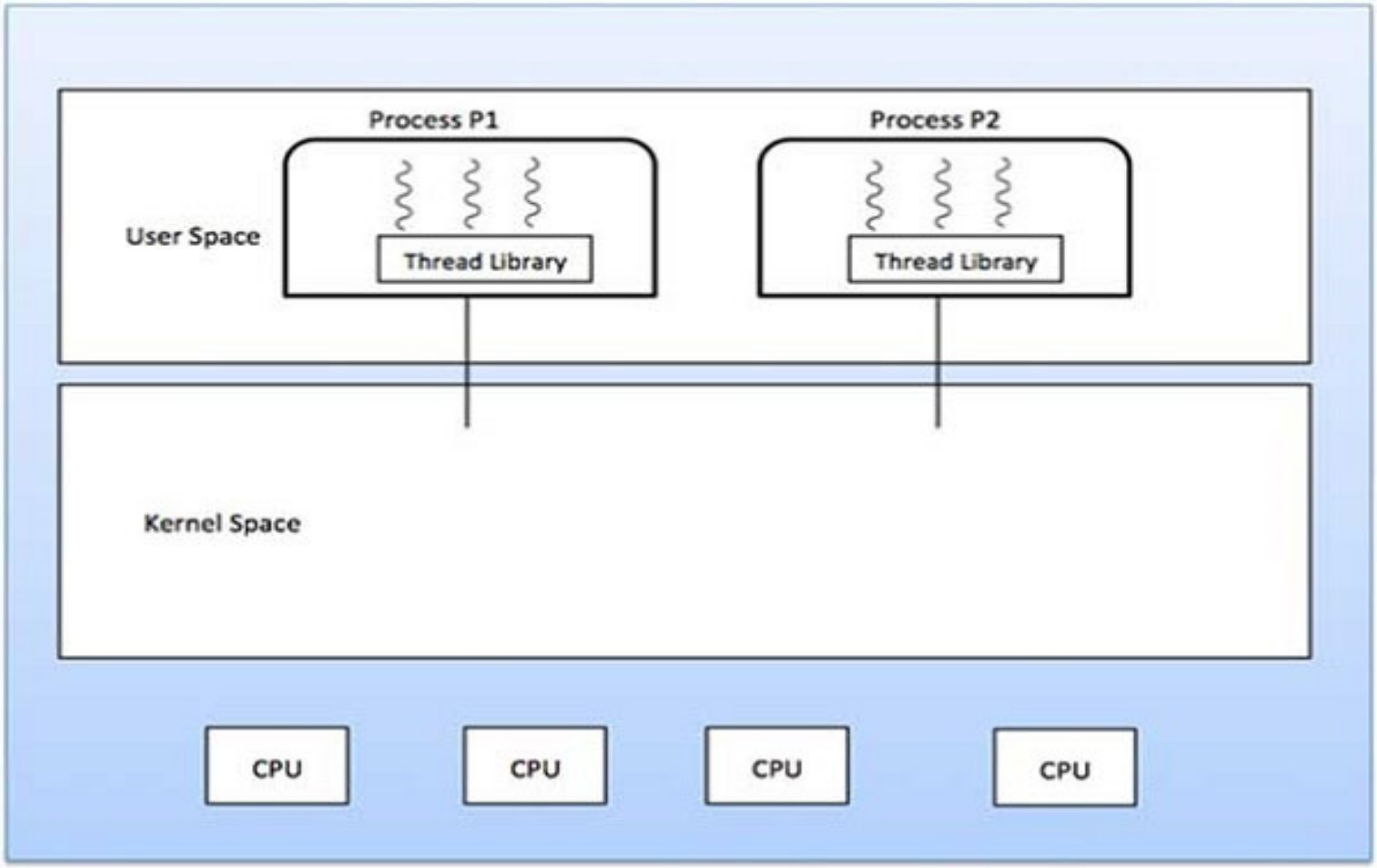
- The thread library creates and manages user-level threads in a process and associates a thread control block (TCB) with each user-level thread.

- The thread library contains code for:
  - Creating, managing, and destroying threads
  - scheduling thread execution
  - saving and restoring thread contexts
  - passing message and data between threads

user-level threads Libraries

- It implement a library entirely in user space with no kernel support. All code and data structures for the library exist in user space. This means that invoking a function in the library results in a local function call in user space and not a system call.

user-level threads Libraries cont…



user-level threads Libraries cont…

- Three main user-level threads Libraries are in use today:
  - Pthreads: is common in UNIX (Solaris, Linux, Mac OS X)
  - Win32: the Win32 thread library is available on Windows systems.
  - Java: The Java thread API allows java threads be managed by JVM in Java programs. However, because in most instances the JVM is running on top of a host OS, the Java thread API is typically implemented using the threads model provided by underlying OS.

Advantages of ULT

- ULT has a simple representation. Each thread is represented simply by a PC, registers, stack and a small thread control block, all stored in the user process address space.
- ULT has a simple Management: Creating a new thread and Thread switching do not require Kernel mode privileges.
- ULT is flexible. Each process can use a different scheduling algorithm for its own threads.

Advantages of ULT cont…

- ULT is Fast and Efficient. **Thread switching** (switching between threads of the same process) is like a procedure call. It does not involve kernel mode, i.e., no mode switching because *kernel* is not needed.

- Because the kernel knows nothing about the threads, the ULT runs on any OS (even ones not supporting threads).

Context Switch

- Switching the CPU from one process or thread to another is called context switch. It requires saving the state of the old process or thread and loading the state of the new one. Context switching between threads in the same process is typically faster than context switching between two separate processes.

  - Process switching: is context switching from one process to a different process.
  - Thread switching: is context switching from one thread to another in the same process.
  - Mode switching: switching between user mode and kernel mode within a given process or thread using a system call.

A context is the contents of a CPU's registers and program counter at any point in time.

Disadvantages of ULT

- Since the OS does not know about the existence of the user-level threads, it may make poor scheduling decisions:
  - It might run a process that only has idle threads.
  - It schedules the process the same way as other processes, regardless of the number of threads.

- Since the kernel assigns only one CPU to the process, a **multithreaded application cannot take advantage of multiprocessors**. The reason is that, if a thread starts running, other threads in that process can not run unless the first thread gives up the CPU.

Threading Models

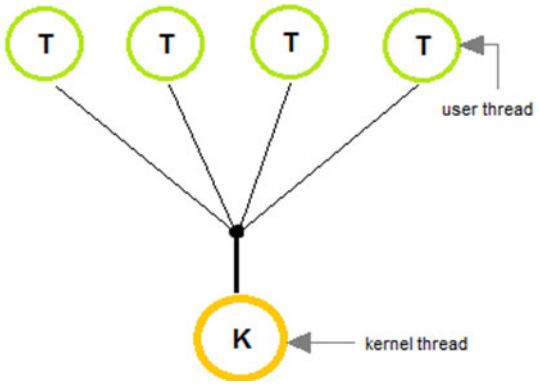
- The user level threads must be mapped to kernel threads. In general, user-level threads can be implemented using one of four models:

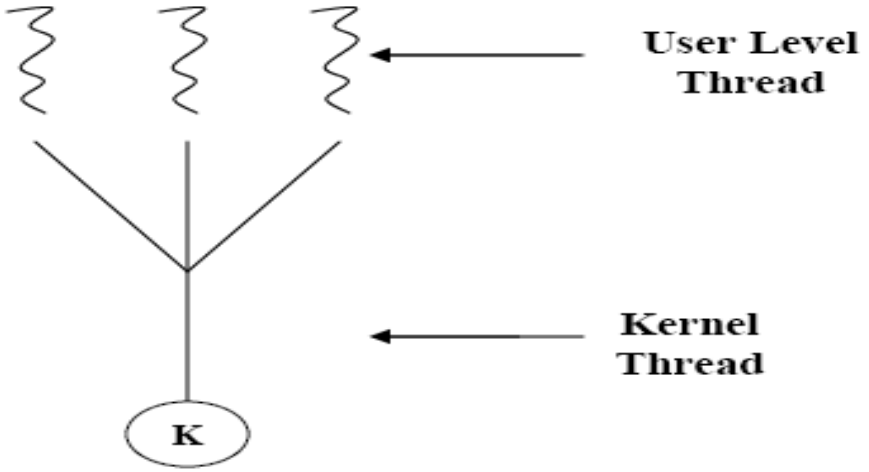
Threads:Processes	Description
1:1	Each thread of execution is a unique process with its own address space and resources.
M:1	A process defines an address space and dynamic resource ownership. Multiple threads may be created and executed within that process.
1:M	A thread may migrate from one process environment to another. This allows a thread to be easily moved among distinct systems.
M:N	Combines attributes of M:1 and 1:M cases.

1:M is not Multithreading

Many to One Model (M:1)

- This is done in cases where OS does not support multithreading.
- Because only one thread can access the kernel at a time, multiple threads are not able to run in parallel on multiprocessors **when kernel mode is required.**
- Applications get some user-level benefit from multithreading but when they interact with the kernel there is no additional benefit.
- the entire process will block if a thread makes a blocking system call.
- If multiple threads need the kernel, only one gets through and the rest block.



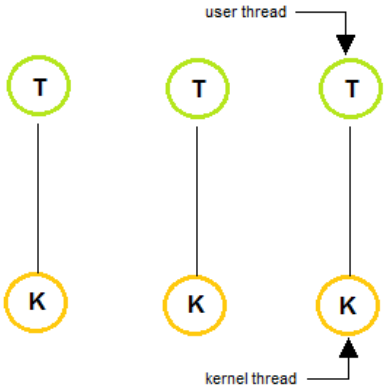


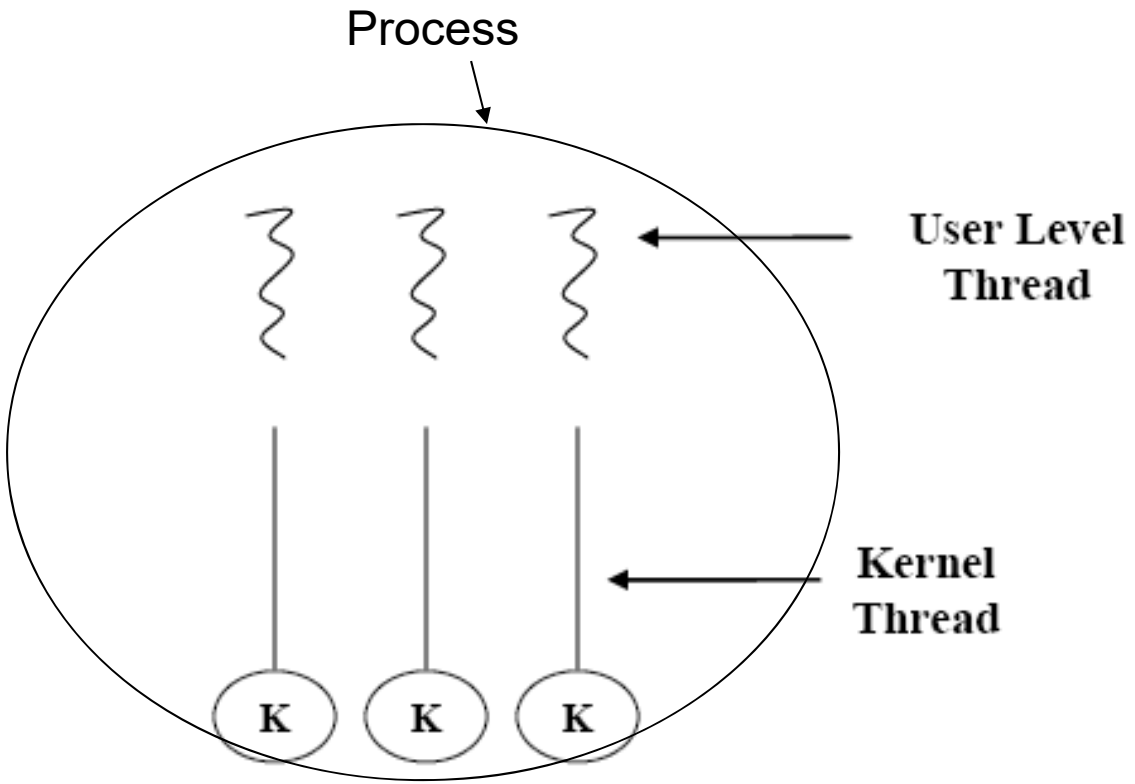
One to One Model (1:1)

- One user thread always corresponds to a kernel thread. This model for each thread allocates one lightweight processes (LWP), thus supports multiple threads to run in parallel on multiprocessors.

- The OS is aware of each thread and can schedule another if a particular thread blocks.

If an application requires many kernel services, it also bears the burden of creation and destruction of kernel threads.





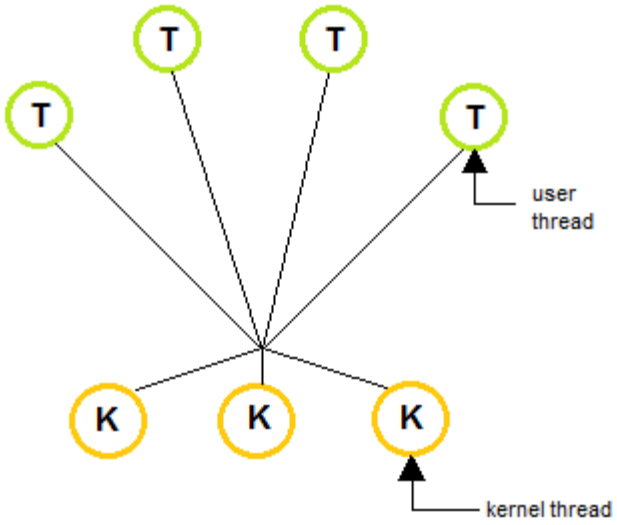
One to One Model cont…

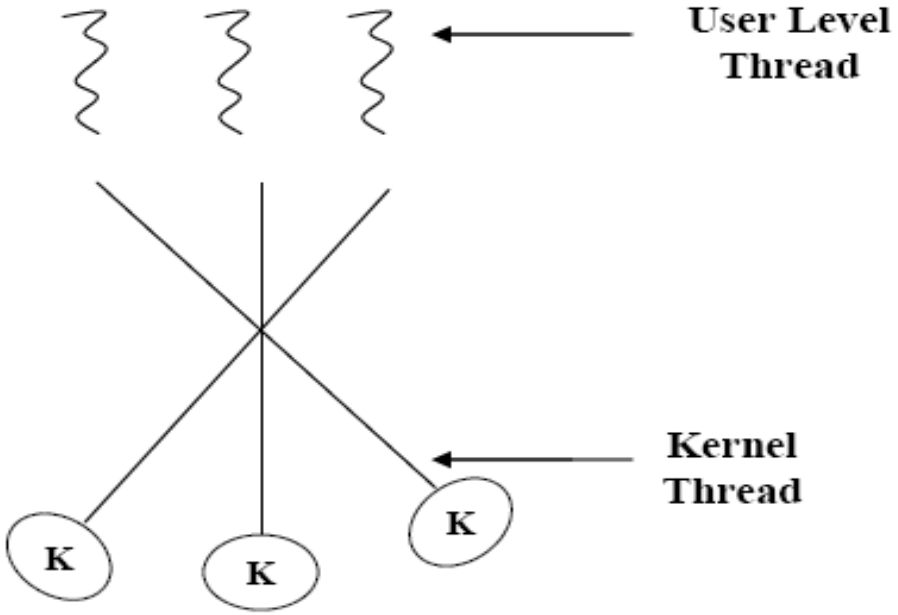
- It also allows one or more threads to issue blocking system calls as the other threads continue to run even on uniprocessor.

- This model has the drawback that thread creation involves LWP creation; hence it requires a system call. In addition, each LWP takes up additional kernel resources, so you are limited in the total number of threads you can create. Win32 and OS/2 use this model.

Many to Many Model (M:N)/Two-level thread model

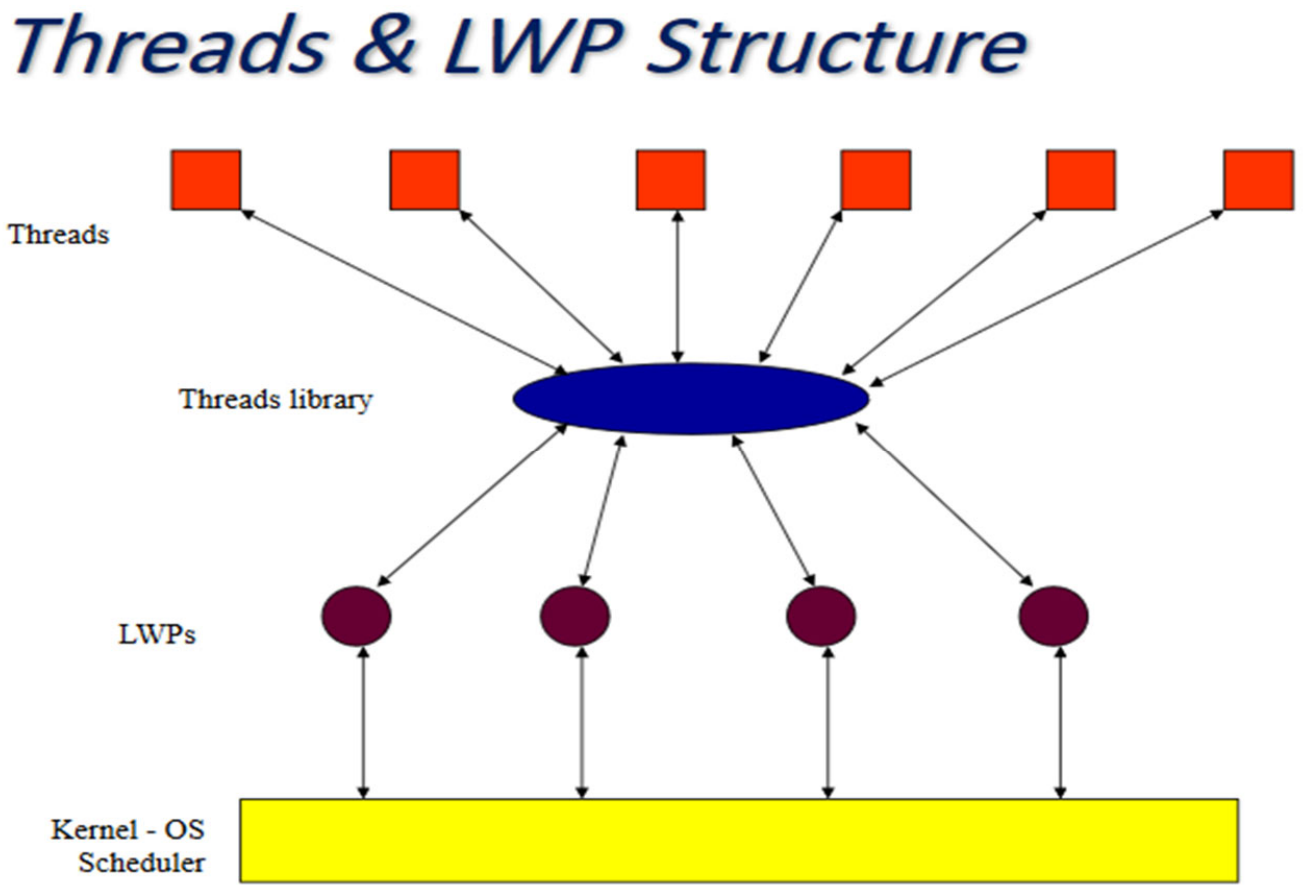
- This is known as hybrid threading and many user level threads are mapped to the Kernel thread of smaller or equal numbers.
- In this model, developers can create as many user threads as necessary and the corresponding Kernel threads can run in parallels on a multiprocessor.
- When a thread performs a blocking system call, the kernel can schedule another thread for execution.





Many to Many Model cont…

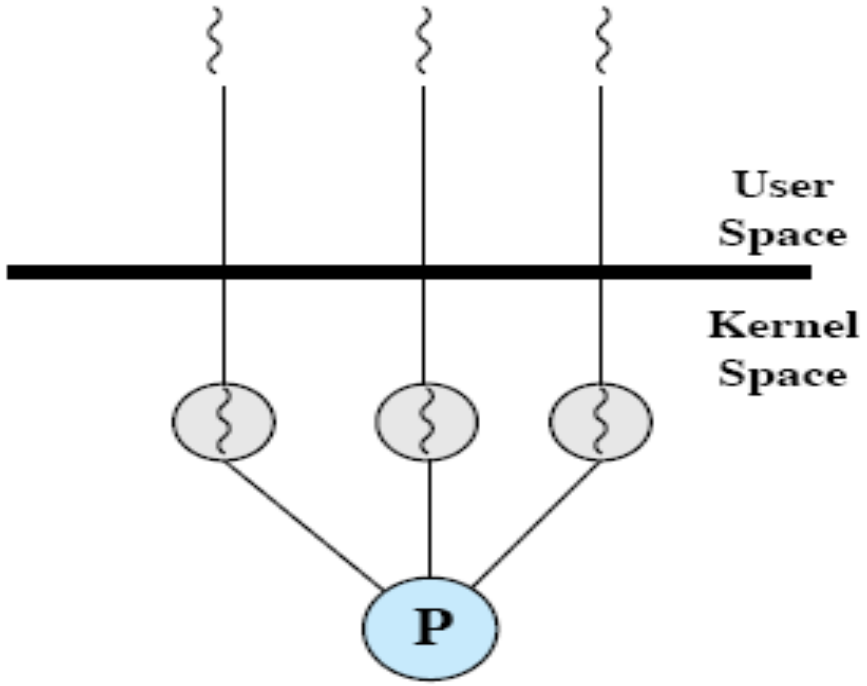
- In this model the only limit on the number of threads is the size of memory.



Kernel Level Thread

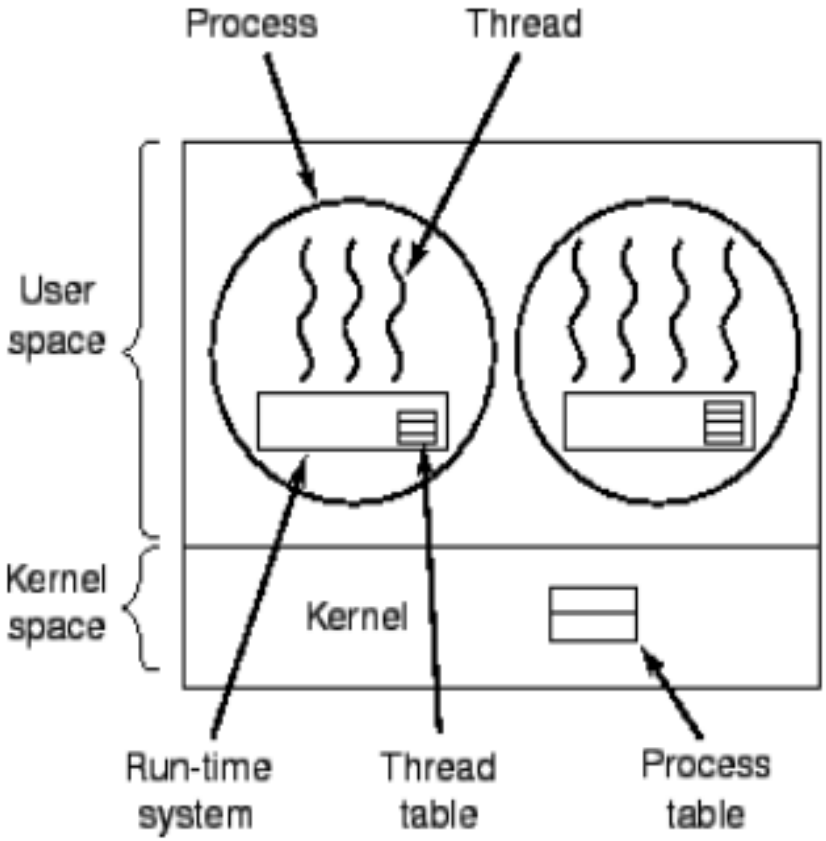
- A kernel thread is created and managed by the kernel (not by application). Example is Windows.

- Kernel threads are scheduled by the OS's scheduling algorithm. Since the kernel must manage and schedule threads as well as processes, the kernel maintains context information for the process as a whole and for individual threads within the process.

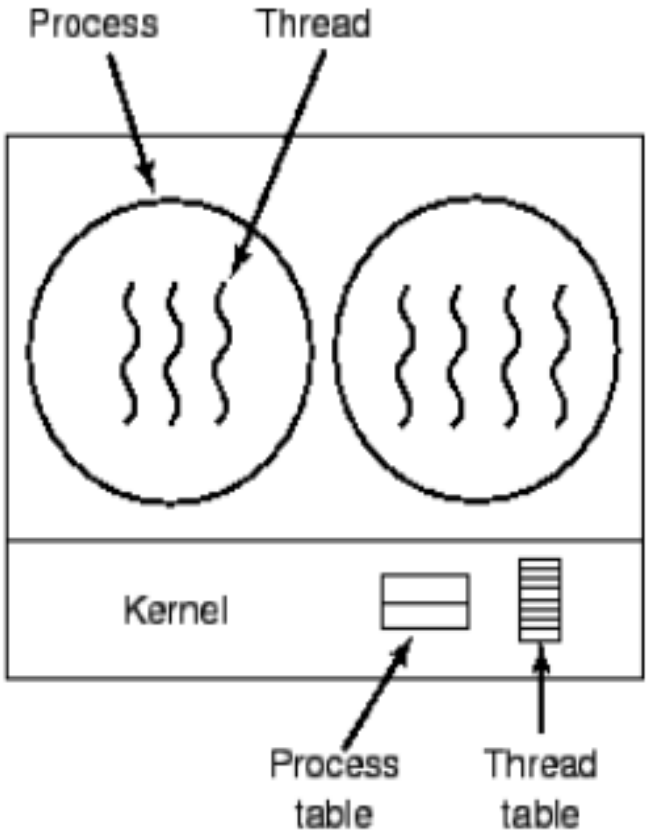


ULT vs. KLT

There is no thread table in each process.



(a) A user-level threads package



(b) A threads package managed by the kernel

Advantages of KLT

- Since the kernel assigns multiple CPUs to different threads of the same process, a **multithreaded application can fully take advantage of multiprocessing**.

Advantages of KLT cont…

- There is no thread table in each process. **Instead**, the kernel has a thread table that keeps track of all the threads in the system. If one thread in a process is blocked, the kernel can schedule another thread of the same process. Thus, Kernel-level threads are suitable for applications that frequently block.

- Because kernel has full knowledge of all threads, Scheduler may decide properly to give more time to a process having large number of threads than process having small number of threads.

Disadvantages of KLT

- The kernel-level threads are much slower than user threads. The thread switching involves kernel mode, i.e., requires mode switching which transfers control to the OS (user->kernel-> user).

- The kernel must manage and schedule threads as well as processes. Thus, it requires a full thread control block (TCB) for each thread to maintain information about threads. As a result there is significant overhead and increased in kernel complexity.

- Thread management is a system call to the kernel. The system calls to create kernel threads are specific to the OS unlike User level thread that can run on any operating system.