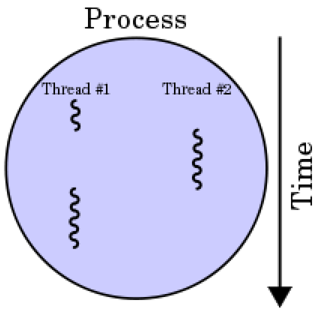
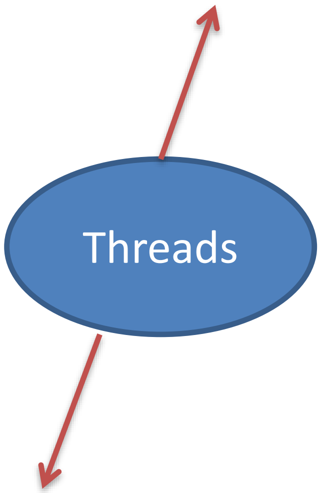
Thread

- Thread is **light weight process** created by a process.



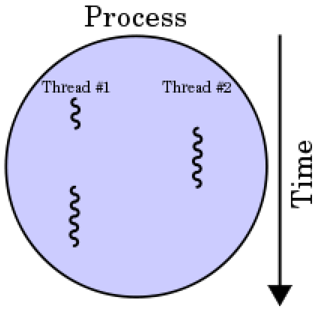
Processes are used to execute large, ‘heavyweight’ jobs such as working in word, while threads are used to carry out smaller or ‘lightweight’ jobs such as auto saving a word document.



Threads

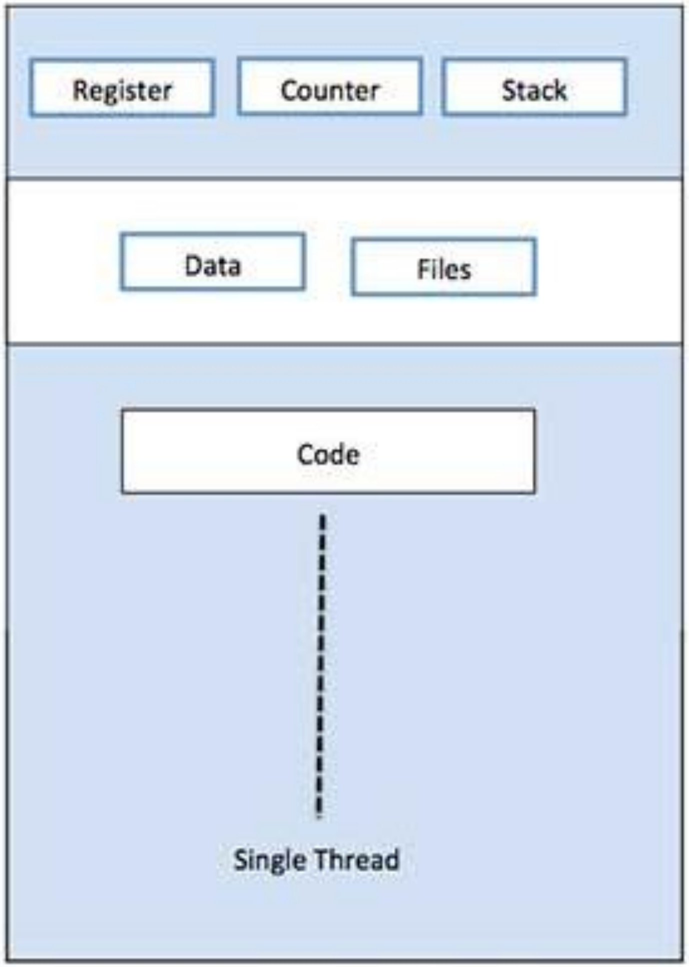
Thread

- Thread is **light weight process** created by a process.
- Thread is a **single sequence stream** within a process.
- Thread has it own

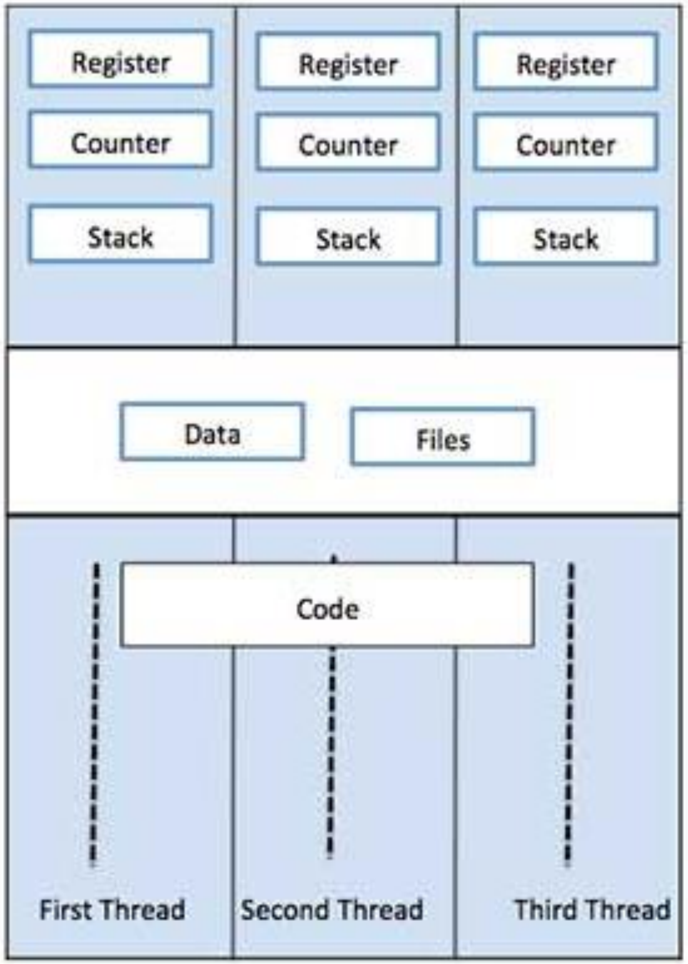


1. **program counter** that **keeps track** of **which instruction to execute next**.
2. **system registers** which **hold its current working variables**.
3. **stack** which **contains the execution history**.

Single Thread VS Multiple Thread



Single Process P with single thread



Single Process P with three threads

Similarities between Process & Thread

- Like processes threads **share CPU** and **only one thread is running at a time**.
- Like processes threads **within a process execute sequentially**.
- Like processes thread **can create childrens**.
- Like a traditional process, a thread **can be in any one of several states**: running, blocked, ready or terminated.
- Like process threads **have Program Counter**, **Stack**, **Registers** and **State**.

Dissimilarities between Process & Thread

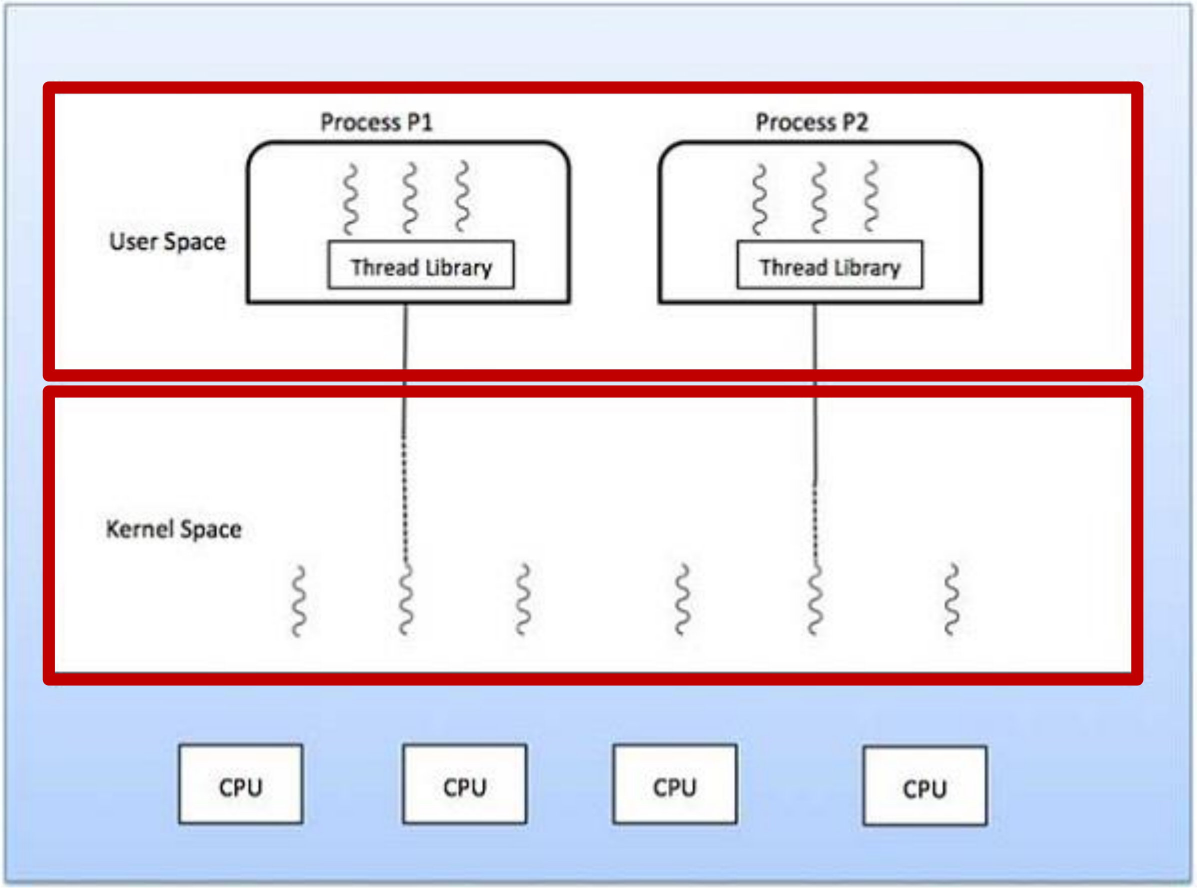
- Unlike processes threads are **not independent** of one another.
- Threads within the same process **share an address space**.
- Unlike processes all threads can **access every address** in the task.
- Unlike processes threads are **design to assist one other**. Note that processes might or might not assist one another because processes may be originated from different users.

Advantages of Threads

- Threads **minimize** the **context switching time**.
- Use of threads **provides concurrency** within a process.
- **Efficient communication**.
- It is more **easy to create** and **context switch** threads.
- Threads can **execute** in **parallel** on multiprocessors.
- With threads, an application can **avoid per-process overheads**
  - Thread creation, deletion, switching easier than processes.
- Threads have **full access** to **address space** (easy sharing).

Types of Threads

1. Kernel Level Thread
2. User Level Thread



User Level Threads

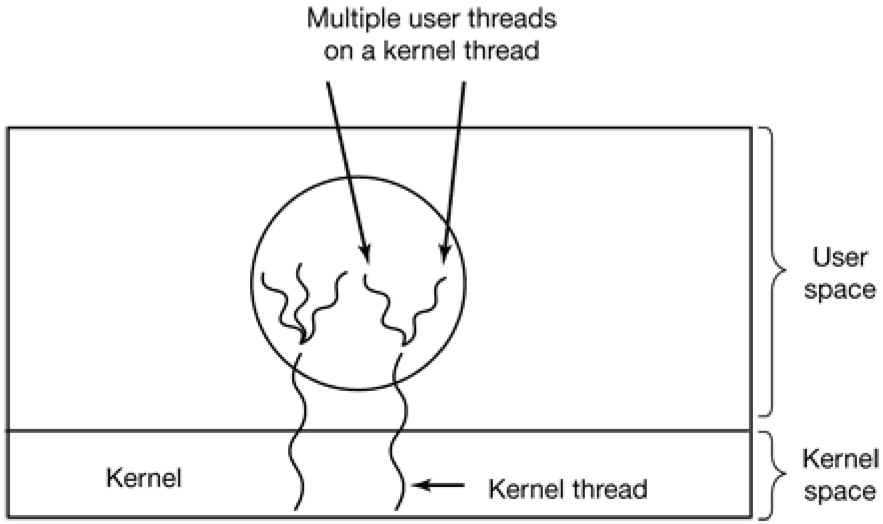
Kernel Level Threads

Types of Threads (Cont…)

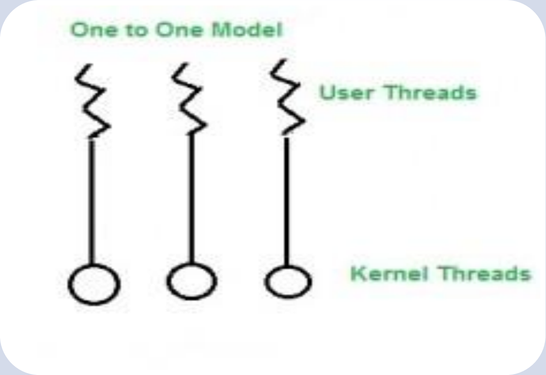
USER LEVEL THREAD	KERNEL LEVEL THREAD
User thread are implemented by users.	Kernel threads are implemented by OS.
OS doesn’t recognize user level threads.	Kernel threads are recognized by OS.
Implementation of User threads is easy.	Implementation of Kernel thread is complex.
Context switch time is less.	Context switch time is more.
Context switch requires no hardware support.	Context switch requires hardware support.
If one user level thread perform blocking operation then entire process will be blocked.	If one kernel thread perform blocking operation then another thread with in same process can continue execution.
Example : Java thread, POSIX threads.	Example : Window Solaris

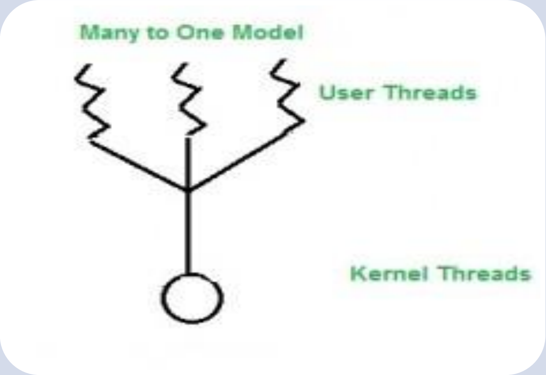
Hybrid Thread

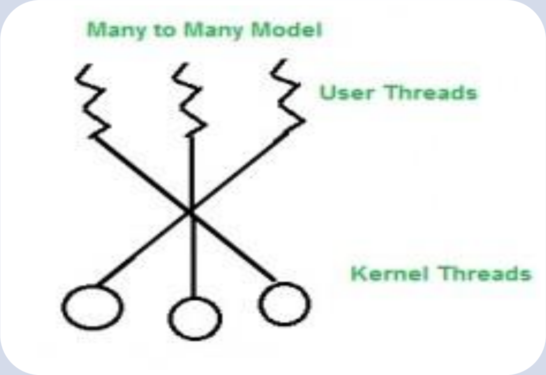
- Combines the advantages of user level and kernel level thread.
- It **uses kernel level thread** and then **multiplex user level thread on to** some or all of **kernel threads**.
- **Gives flexibility** to programmer that how many kernel level threads to use and how many user level thread to multiplex on each one.
- Kernel is aware of only kernel level threads and schedule it.



Multi threading models







One to One Model

Each user threads mapped to one kernel thread.

Problem with this model is that creating a user thread requires the corresponding kernel thread.

Many to One Model

Multiple user threads mapped to one kernel thread.

Problem with this model is that a user thread can block entire process because we have only one kernel thread.

Many to Many Model

Multiple user threads multiplex to more than one kernel threads.

Advantage with this model is that a user thread can not block entire process because we have multiple kernel thread.

Pthread function calls

1. Pthread_create:- Create a new thread
2. Pthread_exit:- Terminate the calling thread
3. Pthread_join:- Wait for a specific thread to exit
4. Pthread_yield:- Release the CPU to let another thread run
5. Pthread_attr_init:- Create and initialize a thread’s attribute structure
6. Pthread_destroy:- Remove a thread’s attribute structure