User Threads

* Created by a threading library API

* Scheduling is managed by the threadin

* Creating a user thread is fast since

* Context switch is faster since kernel is not involved.

* The kernel is not aware about the user level threads.

* User level threads do not need any hardware support. Kernel threads require a CPU and registers.

* User threads cannot take advantage of the multiple processors.

* User threads can even work on systems that do not have support for OS threads. Procss

* Synchronization is much easier between user threads.

User > Kernel > Procss

* If one user thread blocks on I/O, all other user threads get blocked.

* user threads can scale to large numbers.

* user threads are lightweight than OS threads and are very fast - consumes less resources

* lacks Coordination with the kernel. Since kernel doesn't know about user level threads, it cannot take any scheduling decision on it.

Why creating a kernel thread is slow ?

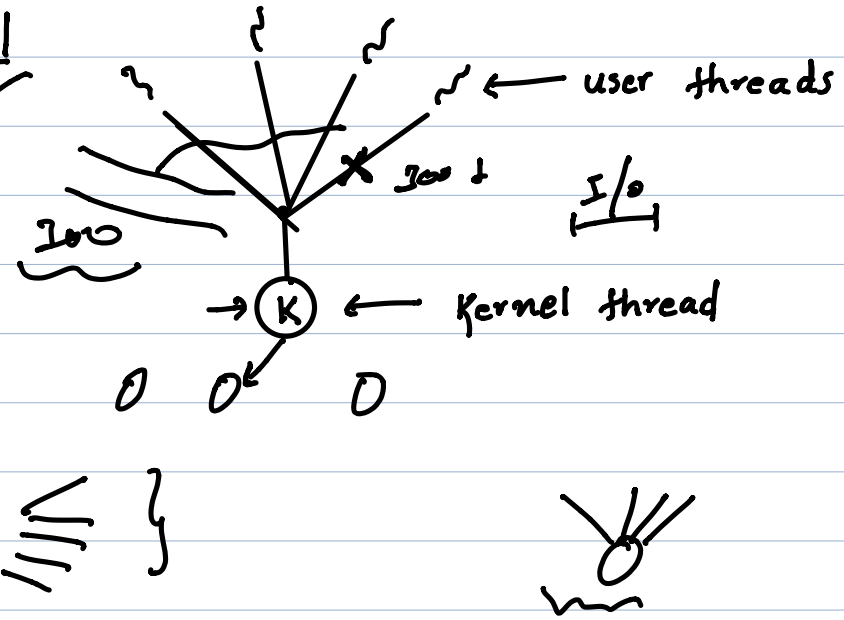
* Overhead of creating thread control block
* overhead of system calls

Why context switch is faster for user thread ?

* Context switch doesn't involve the kernel
* Involves only some function call and variables change
* Kernel thread requires copying of CPU register values

Multithreading Models

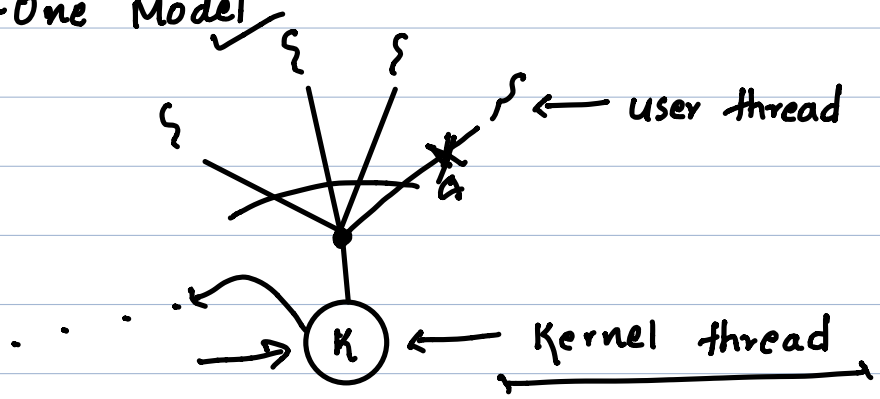
Many-to-One Model



* Ultimately, a relationship must exist between user threads and Kernel threads;

* User threads reside inside a program. Kernel threads can only be scheduled on a processor.

① Many-To-One Model



* Many user threads are mapped to a single Kernel thread.

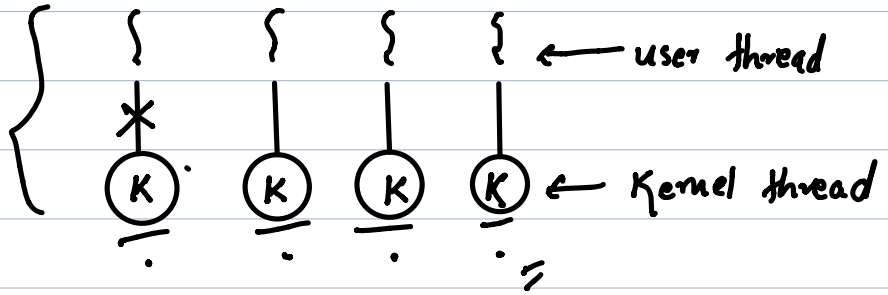
* Thread management and scheduling is by the thread library.

* If one user thread blocks, all others also get blocked.

* Cannot take advantage of the multiple processing cores.

* Green thread - a library for Solaris system.
Adopted by early versions of Java.

(2) One-To-One Model



* Each user thread is mapped to a kernel thread.

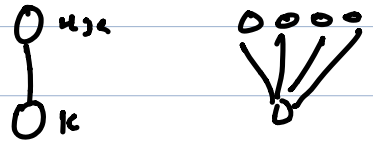
* Can utilize multiple cores, higher concurrency.

* If one thread blocks, others can still continue running.

Limitation :

* Restriction on the number of user threads that can be created.
* Creating a lot of Kernel threads can put burden on the application.

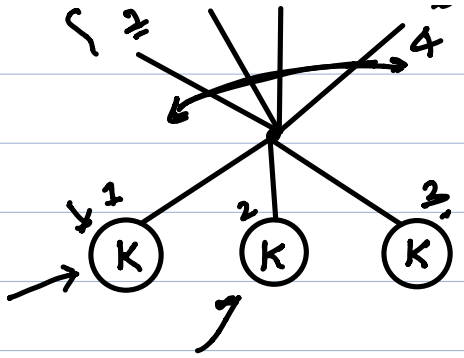
Many-To-Many-Model



* Multiple user threads are mapped to a smaller or equal number of Kernel threads.
* Can utilize multiple processors, higher concurrency.
* Number of Kernel threads can be adjusted depending upon machine or application.
* If one the threads get blocked, the Kernel can schedule another thread for execution.

Two-level model

* A variation of many-to-many model
* A user thread can be bounded to a Kernel thread



M : N

M > N

← Kernel thread

* Decreases context switch
* Perform Some specific tasks with bounded threads.

Thread creation and scheduling

M : 1 → Thread library

1 : 1 → Kernel involvement

M : M → Thread library + Kernel involvement

Ease of context switch

M : 1 > M : M > 1 : 1

Hardware Support

M : 1 → No support

{ 1 : 1 > M : M

Concurrency / advantage of multi processors

1 : 1 > M : M > M : 1

Working on a Single processor system

1:1 → Cannot work

M:M and M:1 → Can work

System call blocking

M:1 → all others block

M:M and 1:1 → no effect on others

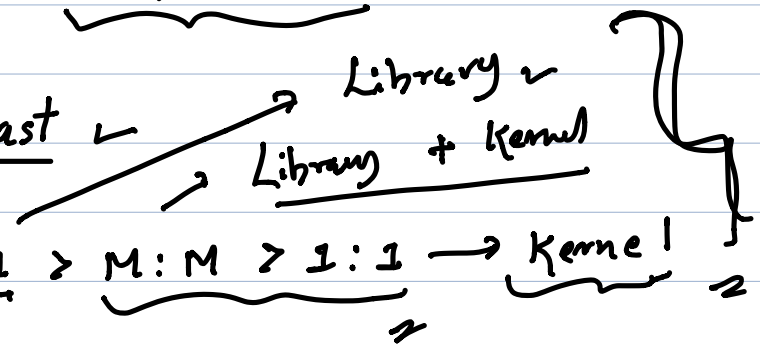
Ease of synchronization

M:1 > M:M > 1:1

Scaling / no of user threads you can create

M:1 > M:M > 1:1

Lightweightness / Fast



Coordination with Kernel

1:1 > M:M > M:1