OPERATING SYSTEMS

Threads and Concurrency 02

Nitin V Pujari
Faculty, Computer Science
Dean - IQAC, PES University

Course Syllabus - Unit 2

UNIT 2: Threads and Concurrency

Introduction to Threads, types of threads, Multicore Programming, Multithreading Models, Thread creation, Thread Scheduling, PThreads and Windows Threads, Mutual Exclusion and Synchronization: software approaches, principles of concurrency, hardware support, Mutex Locks, Semaphores. Classic problems of Synchronization: Bounded-Buffer Problem, Readers -Writers problem, Dining Philosophers Problem concepts. Synchronization Examples - Synchronisation mechanisms provided by Linux/Windows/Pthreads. Deadlocks: principles of deadlock, tools for detection and Prevention.

Course Outline - Unit 2

13	4.1,4.2	Introduction to Threads, types of threads, Multicore Programming.	4	42.8
14	4.3,5.4	Multithreading Models, Thread creation, Thread Scheduling	4	
15	4.4	Pthreads and Windows Threads	4	
16	6.1-6.3	Mutual Exclusion and Synchronization: software approaches	6	
17	6.3-6.4	principles of concurrency, hardware support	6	
18	6.5,6.6	Mutex Locks, Semaphores	6	
19	6.7.1-6.7.3	Classic problems of Synchronization: Bounded-Buffer Problem, Readers -Writers problem, Dining Philosophers Problem concepts	6	
20	6.9	Synchronization Examples - Synchronisation mechanisms provided by Linux/Windows/Pthreads.	6	
21	Handouts	Demonstration of programming examples on process synchronization		
22	7.1-7.3	Deadlocks: principles of deadlock, Deadlock Characterization.	7	
23	7.4	Deadlock Prevention, Deadlock example	7	
24	7.6	Deadlock Detection	7	

Topics Outline

- Threads and Concurrency
  - Type of Threads
  - Multithreading Models

Types of Threads

There are two types of threads:

1. User Threads

2. Kernel Threads

User Threads

- **User threads**, are above the kernel and without kernel support. These are the threads that application programmers use in their programs.

- Three primary thread libraries:
    - POSIX Pthreads
    - Windows threads
    - Java threads

Kernel Threads

- **Kernel threads** are supported within the kernel of the OS itself.

- All modern OS support kernel level threads, allowing the kernel to perform multiple simultaneous tasks and/or to service multiple kernel system calls simultaneously.

- Examples – virtually all general purpose operating systems, including:

  - Windows
  - Solaris
  - Linux Tru64 UNIX
  - Mac OS X

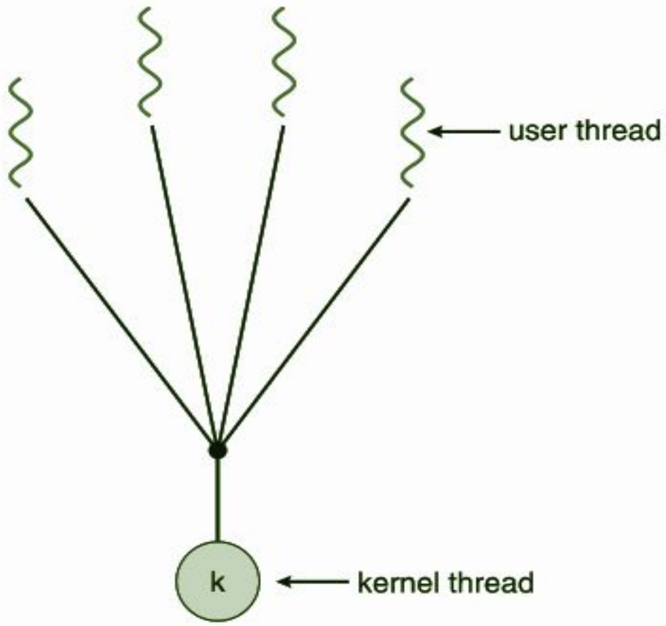
Kernel Threads

Note:

- Linux has a unique implementation of threads.
- To the Linux kernel, there is no concept of a thread.
- Linux implements all threads as standard processes.
- The Linux kernel does not provide any special scheduling semantics or data structures to represent threads.
- Instead, a thread is merely a process that shares certain resources with other processes.
- Each thread has a unique task_struct and appears to the kernel as a normal process which just happens to share resources, such as an address space, with other processes.

User Threads - Kernel Threads: Many to One Model

- Many user-level threads mapped to single kernel thread
- One thread blocking causes all to block
- Multiple threads may not run in parallel on muticore system because only one may be in kernel at a time
- Few systems currently use this model

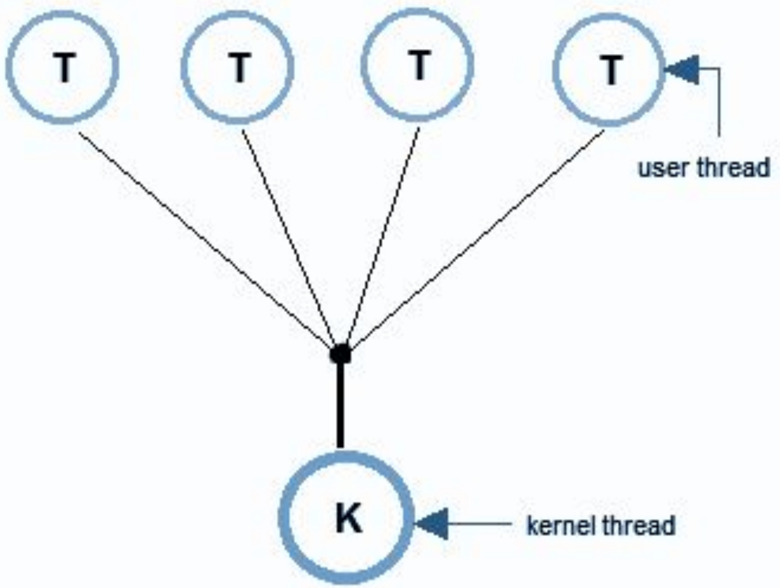


- Examples:
  - **Solaris Green Threads**
  - **GNU Portable Threads**

User Threads - Kernel Threads: Many to One Model

Many to One Model

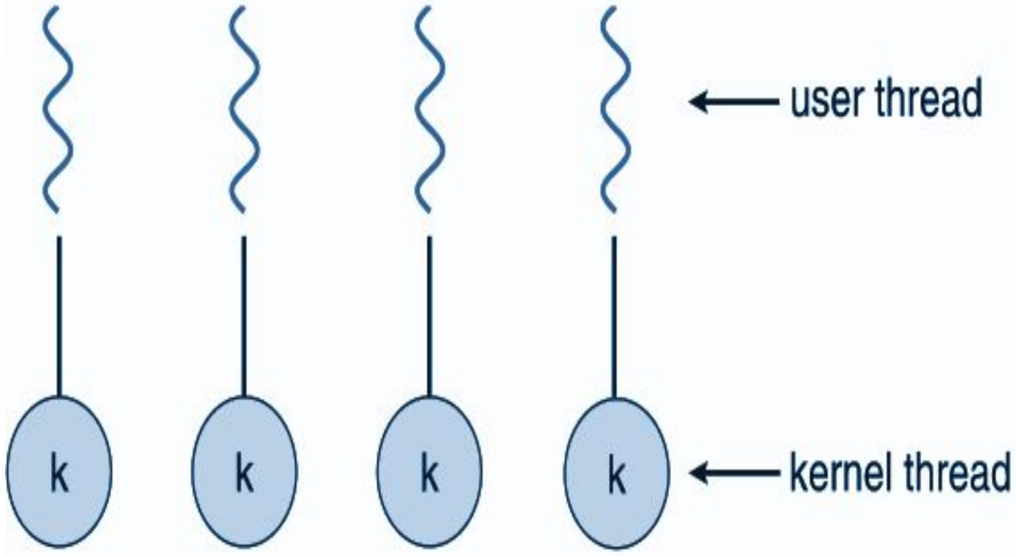
- In the **many to one** model, many user-level threads are all mapped onto a single kernel thread.
- Thread management is handled by the thread library in user space, which is efficient in nature.



User Threads - Kernel Threads: One to One Model

- Each user-level thread maps to kernel thread
- Creating a user-level thread creates a kernel thread
- More concurrency than many-to-one
- Number of threads per process sometimes restricted due to overhead

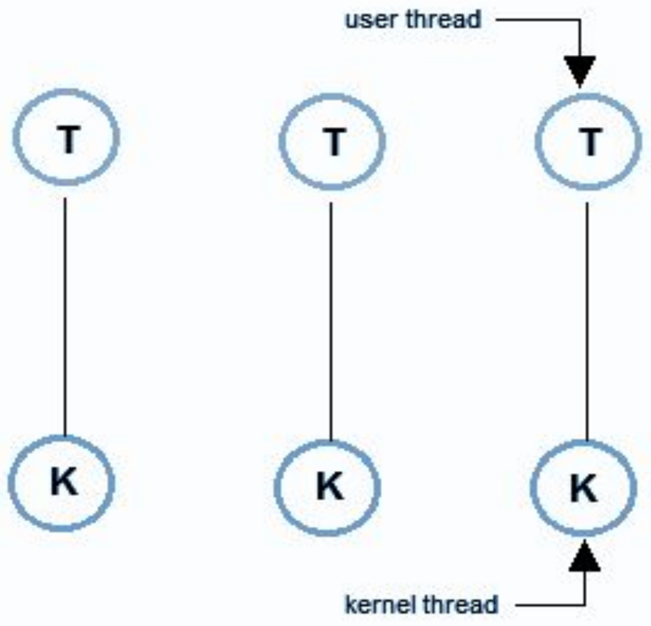
- Examples
  - Windows
  - Linux
  - Solaris 9 and later



User Threads - Kernel Threads: One to One Model

One to One Model

- The **one to one** model creates a separate kernel thread to handle each and every user thread.
- Most implementations of this model place a limit on how many threads can be created.
- Linux and Windows from 95 to XP implement the one-to-one model for threads.



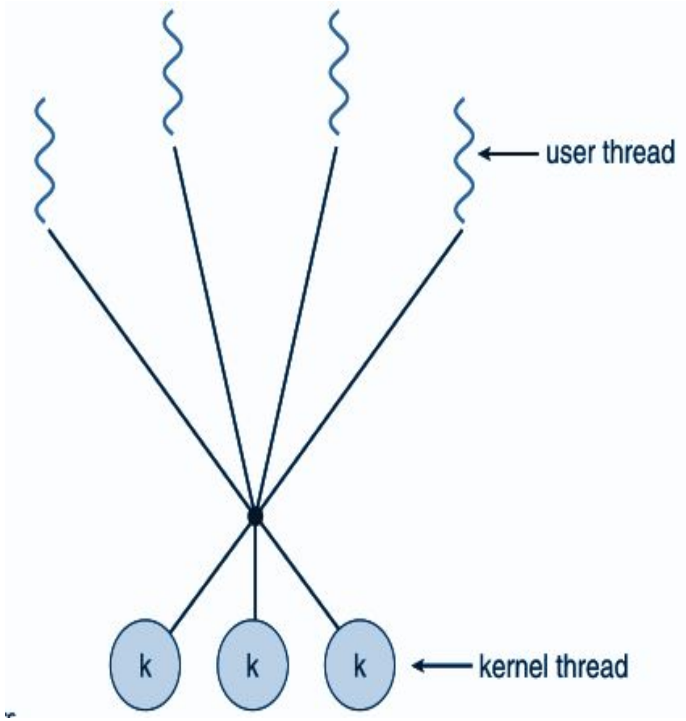
User Threads - Kernel Threads: Many to Many Model

Allows many user level threads to be mapped to many kernel threads

Allows the operating system to create a sufficient number of kernel threads

Solaris prior to version 9

Windows with the *ThreadFiber* package



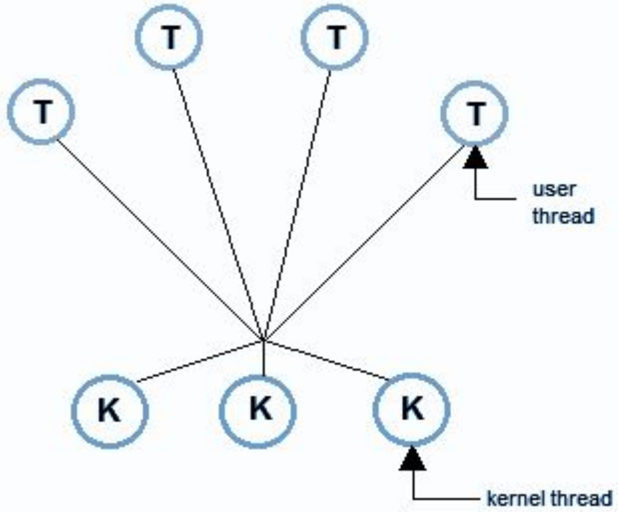
Examples

- IRIX
- HP-UX
- Tru64 UNIX
- Solaris 8 and earlier

User Threads - Kernel Threads: Many to Many Model

Many to Many Model

- The **many to many** model multiplexes any number of user threads onto an equal or smaller number of kernel threads, combining the best features of the one-to-one and many-to-one models.
- Users can create any number of the threads.
- Blocking the kernel system calls does not block the entire process.
- Processes can be split across multiple processors.



OS Kernel Threads, Hardware Threads

- When multi-core microprocessors came into existence, it had multiple executions of instruction streams going on at the same-exact time.
- The need to distinguish between threads arise. Hence, CPU Threads and OS Threads.
- A particular task with OS Thread #29 could be executing in the 4th Core which could be thought of as executing CPU Thread #4.
- A particular process such as OS Process #17 could consist of multiple tasks each with its unique OS Thread, which could, at any one moment, be executing CPU Thread #4.
- Thus, Process #17 OS Thread #29 is executing CPU Thread #4.
- One would also view that as Process #17 Thread #29 is executing in Core #4.
- In summary: using the terms: “CPU Thread” and “OS Thread” as defined above reduces the ambiguity of the term “Thread.”

Screen Shot

```
Thread(s) per core:    2
Core(s) per socket:    12
Socket(s):             4
```

You have 4 CPU sockets, each CPU can have, up to, 12 cores and each core can have two threads.

Your max thread count is, 4 CPU x 12 cores x 2 threads per core, so 12 x 4 x 2 is 96. Therefore the max thread count is 96 and max core count is 48.

What is better ?

That depends on what you want to do, more threads means less frequency (ie a 3ghz becomes split in two) but better multi-tasking (more threads) and using full cores (no hyper-threading) is better for high CPU usage tasks (ie games).

Screen Shot

```
sridatta@sridatta:~$ ps -e -T |more
    PID    SPID TTY          TIME CMD
      1       1 ?        00:00:04 systemd
      2       2 ?        00:00:00 kthreadd
      3       3 ?        00:00:00 rcu_gp
      4       4 ?        00:00:00 rcu_par_gp
      6       6 ?        00:00:00 kworker/0:0H-kblockd
      8       8 ?        00:00:20 kworker/u12:0-events_unbound
      9       9 ?        00:00:00 mm_percpu_wq
     10      10 ?        00:00:00 ksoftirqd/0
     11      11 ?        00:00:01 rcu_sched
     12      12 ?        00:00:00 migration/0
     13      13 ?        00:00:00 idle_inject/0
     14      14 ?        00:00:00 cpuhp/0
     15      15 ?        00:00:00 cpuhp/1
     16      16 ?        00:00:00 idle_inject/1
     17      17 ?        00:00:00 migration/1
     18      18 ?        00:00:00 ksoftirqd/1
     20      20 ?        00:00:00 kworker/1:0H-kblockd
     21      21 ?        00:00:00 cpuhp/2
     22      22 ?        00:00:00 idle_inject/2
```

Screen Shot

09:20:41 PM IST	UID	TGID	TID	%usr	%system	%guest	%wait	%CPU	CPU	Command
09:20:41 PM IST	1000	-	3427	0.00	0.00	0.00	0.00	0.00	3	|__ThreadPoolForeg
09:20:41 PM IST	1000	-	3428	0.01	0.01	0.00	0.00	0.02	3	|__Chrome_ChildIOT
09:20:41 PM IST	1000	-	3431	0.00	0.00	0.00	0.00	0.00	2	|__Compositor
09:20:41 PM IST	1000	3664	-	1.55	0.13	0.00	0.01	1.68	5	chrome
09:20:41 PM IST	1000	-	3664	0.86	0.04	0.00	0.01	0.90	5	|__chrome
09:20:41 PM IST	1000	-	3667	0.05	0.04	0.00	0.02	0.08	3	|__Chrome_ChildIOT
09:20:41 PM IST	1000	-	3668	0.00	0.00	0.00	0.00	0.00	5	|__GpuMemoryThread
09:20:41 PM IST	1000	-	3670	0.10	0.02	0.00	0.01	0.12	0	|__Compositor
09:20:41 PM IST	1000	-	3672	0.02	0.00	0.00	0.00	0.02	5	|__CompositorTileW
09:20:41 PM IST	1000	-	3673	0.01	0.00	0.00	0.00	0.02	1	|__CompositorTileW
09:20:41 PM IST	1000	-	3674	0.02	0.00	0.00	0.00	0.02	0	|__CompositorTileW
09:20:41 PM IST	1000	-	3715	0.01	0.00	0.00	0.00	0.01	4	|__SharedWorker th
09:20:41 PM IST	1000	-	5394	0.00	0.00	0.00	0.00	0.00	3	|__ThreadPoolForeg
09:20:41 PM IST	1000	-	6462	0.00	0.00	0.00	0.00	0.00	4	|__ThreadPoolForeg
09:20:41 PM IST	1000	3703	-	0.00	0.00	0.00	0.00	0.00	3	chrome
09:20:41 PM IST	1000	-	3703	0.00	0.00	0.00	0.00	0.00	3	|__chrome
09:20:41 PM IST	1000	3774	-	0.01	0.00	0.00	0.00	0.01	4	update-notifier
09:20:41 PM IST	1000	-	3774	0.01	0.00	0.00	0.00	0.01	4	|__update-notifier
09:20:41 PM IST	1000	-	3777	0.00	0.00	0.00	0.00	0.00	5	|__gmain
09:20:41 PM IST	1000	4413	-	10.52	1.23	0.00	0.02	11.74	5	chrome
09:20:41 PM IST	1000	-	4413	5.93	0.28	0.00	0.02	6.21	5	|__chrome
09:20:41 PM IST	1000	-	4418	0.39	0.35	0.00	0.05	0.74	3	|__Chrome_ChildIOT
09:20:41 PM IST	1000	-	4419	0.03	0.02	0.00	0.01	0.05	5	|__GpuMemoryThread
09:20:41 PM IST	1000	-	4427	1.26	0.42	0.00	0.06	1.67	5	|__Compositor
09:20:41 PM IST	1000	-	4429	0.52	0.07	0.00	0.02	0.59	3	|__CompositorTileW
09:20:41 PM IST	1000	-	4430	0.53	0.08	0.00	0.02	0.61	4	|__CompositorTileW
09:20:41 PM IST	1000	-	4431	0.53	0.07	0.00	0.02	0.60	5	|__CompositorTileW
09:20:41 PM IST	1000	-	4565	0.23	0.01	0.00	0.01	0.24	1	|__ThreadPoolForeg
09:20:41 PM IST	1000	-	6461	0.03	0.00	0.00	0.00	0.03	1	|__ThreadPoolForeg
09:20:41 PM IST	1000	4566	-	0.00	0.00	0.00	0.00	0.00	5	chrome
09:20:41 PM IST	1000	4570	-	2.93	0.30	0.00	0.01	3.23	2	chrome

Topics Uncovered in this Session

- Threads and Concurrency
  - Type of Threads
  - Multithreading Models

THANK YOU

Nitin V Pujari
Faculty, Computer Science
Dean - IQAC, PES University

nitin.pujari@pes.edu

For Course Deliverables by the Anchor Faculty click on www.pesuacademy.com