OPERATING SYSTEMS

Threads and Concurrency

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

OPERATING SYSTEMS

Course Syllabus - Unit 2

12 Hours

Unit 2: Threads & 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.

OPERATING SYSTEMS

Course Outline

13	Introduction to Threads, types of threads, Multicore Programming, Multithreading Models	4.1 - 4.3	42.8
14	Thread creation, Thread Scheduling	5.4	
15	Pthreads and Windows Threads	4.4	
16	Mutual Exclusion and Synchronization: software approaches,	6.1-6.2	
17	principles of concurrency, hardware support	6.3-6.4	
18	Mutex Locks, Semaphores	6.5, 6.6	
19	Classic problems of Synchronization: Bounded-Buffer Problem, Readers-Writers problem	6.7-6.8	
20	Dining-Philosophers Problem	6.8	
21	Synchronization Examples: Synchronisation mechanisms provided by Linux/Windows/Pthreads.	6.9	
22	Deadlocks: principles of deadlock, Deadlock Characterization	7.1-7.3	
23	Deadlock Prevention, Deadlock example	7.4-7.5	
24	Deadlock Detection, Algorithm	7.6	

Topic Outline

- Threads and Concurrency
  - Introduction to Threads
  - Type of Threads
  - MultiCore Programming
  - Multithreading Models

Motivation

- Process creation is heavy-weight while thread creation is light-weight
- Most modern applications are multithreaded
- Threads run within application
- Multiple tasks in application can be implemented by threads
  - Update display
  - Fetch data
  - Spell checking
- Can simplify code, increase efficiency
- Kernels are generally multithreaded

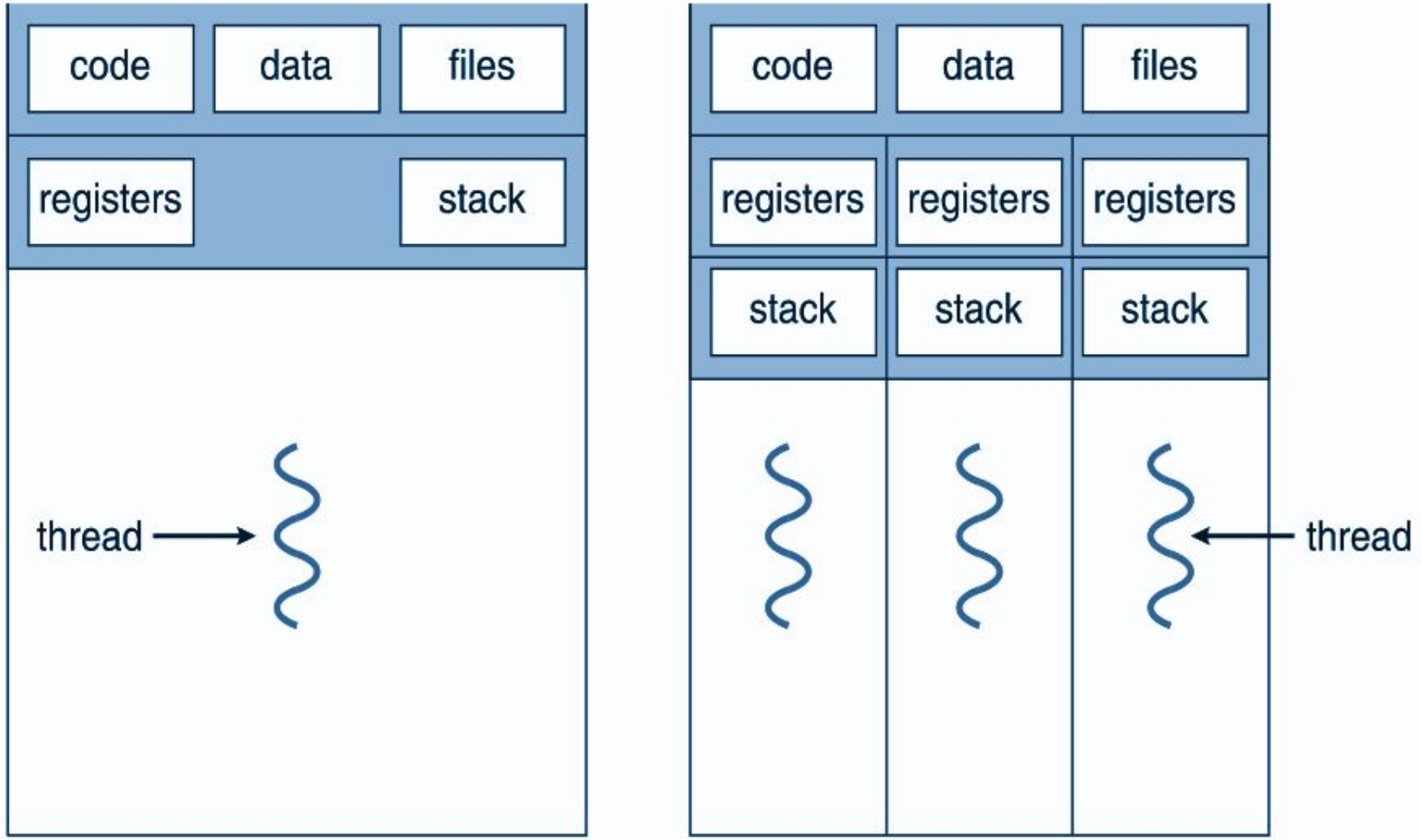
Motivation

Comparison Basis	Process	Thread
Definition	A process is a program under execution i.e an active program.	A thread is a lightweight process that can be managed independently by a scheduler.
Context switching time	Processes require more time for context switching as they are more heavy.	Threads require less time for context switching as they are lighter than processes.
Memory Sharing	Processes are totally independent and don’t share memory.	A thread may share some memory with its peer threads.
Communication	Communication between processes requires more time than between threads.	Communication between threads requires less time than between processes .
Blocked	If a process gets blocked, remaining processes can continue execution.	If a user level thread gets blocked, all of its peer threads also get blocked.
Resource Consumption	Processes require more resources than threads.	Threads generally need less resources than processes.

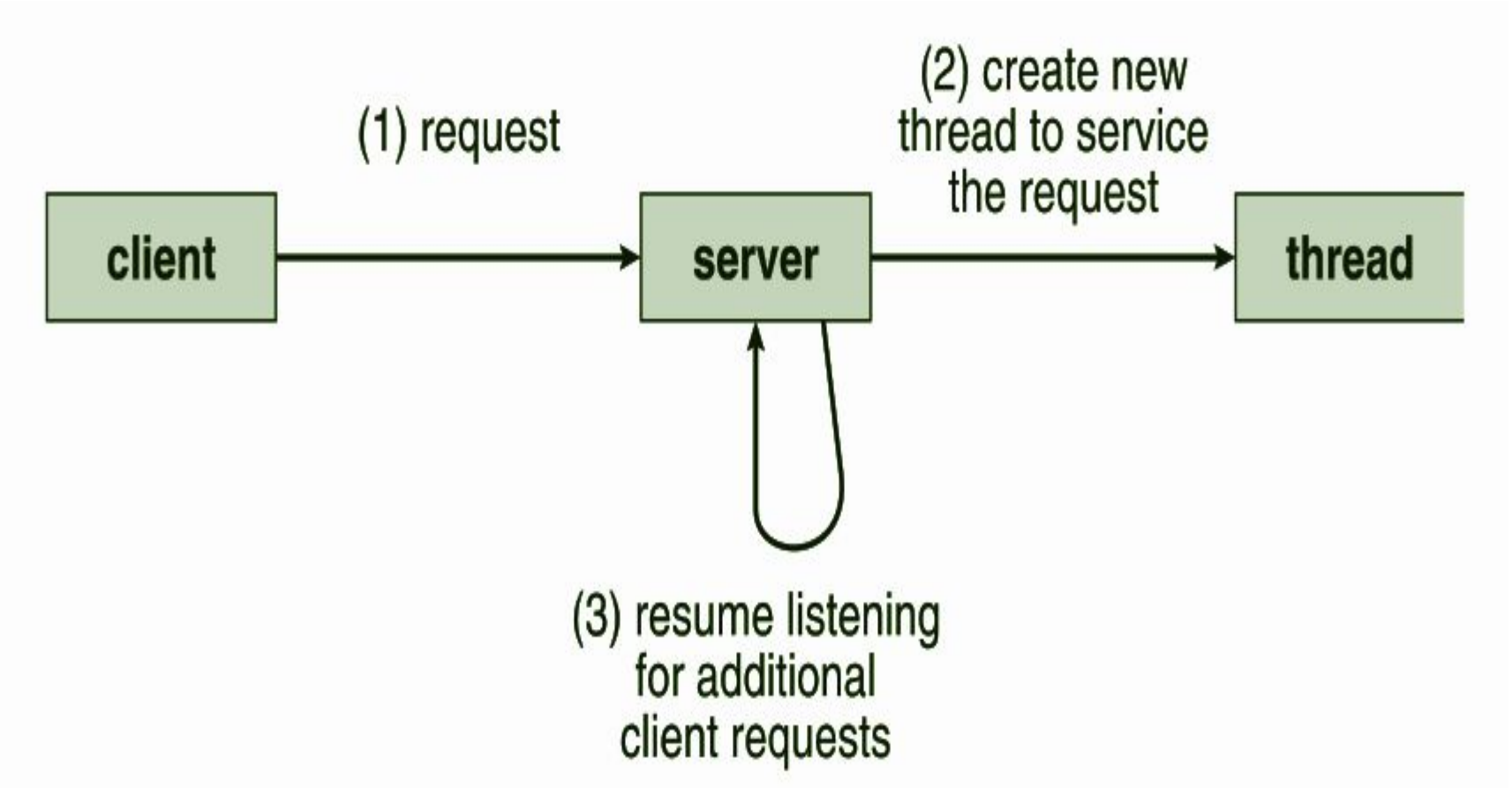
Motivation

Comparison Basis	Process	Thread
Dependency	Individual processes are independent of each other.	Threads are parts of a process and so are dependent.
Data and Code sharing	Processes have independent data and code segments.	A thread shares the data segment, code segment, files etc. with its peer threads.
Treatment by OS	All the different processes are treated separately by the operating system.	All user level peer threads are treated as a single task by the operating system.
Time for creation	Processes require more time for creation.	Threads require less time for creation.
Time for termination	Processes require more time for termination.	Threads require less time for termination.

Single and Multithreaded Processes



Multithreaded Server Architecture



Benefits of Threads

Responsiveness – may allow continued execution if part of process is blocked, especially important for user interfaces

Resource Sharing – threads share resources of process, easier than shared memory or message passing

Economy – cheaper than process creation, thread switching lower overhead than context switching

Scalability – process can take advantage of multiprocessor architectures

MultiCore Programming

- Multicore or multiprocessor systems put pressure on programmers, by throwing challenges like
  - Dividing activities
  - Balance
  - Data splitting
  - Data dependency
  - Testing and debugging
- Parallelism implies a system can perform more than one task simultaneously
- Concurrency supports more than one task making progress
  Single processor / core, scheduler providing concurrency

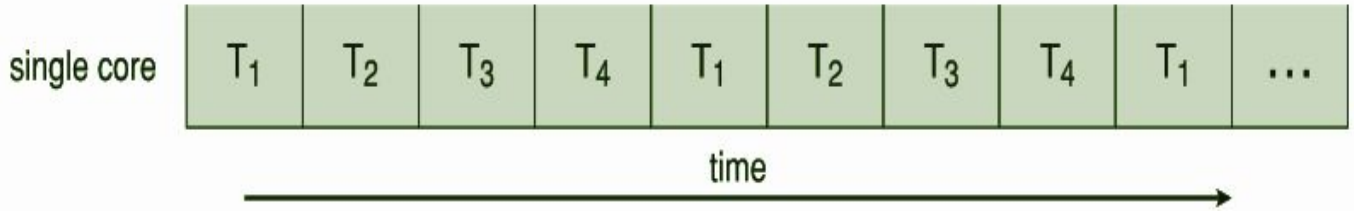
- Types of parallelism
- Data parallelism – distributes subsets of the same data across multiple cores, same operation on each
- Task parallelism – distributes threads across cores, each thread performing unique operation
- As # of threads grows, so does architectural support for threading
- CPUs have cores as well as hardware threads
- Consider Oracle SPARC T4 with 8 cores, and 8 hardware threads per core

MultiCore Programming

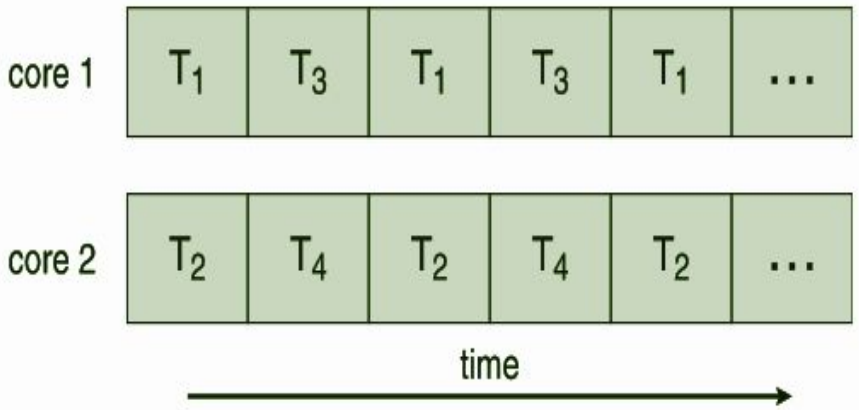
- **What is the Difference Between a Core and a Thread ?**
  - The thread helps deliver the workload to the CPU more efficiently.
  - More threads translates into a better-organized work queue, hence more efficiency in processing the information.
  - CPU cores refer to the actual hardware component.
  - Threads refer to the virtual component that manages the tasks.
  - There are a lot of different variations regarding how the CPU interacts with multiple threads.
  - Basically, the CPU is fed tasks from a thread. It only accesses the second thread when the information sent by the first thread is slow or unreliable (like a cache miss)

Concurrency versus Parallelism

- **Concurrent execution on single-core system:**



- **Parallelism on a multi-core system:**



Types of Thread

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 OSs 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,

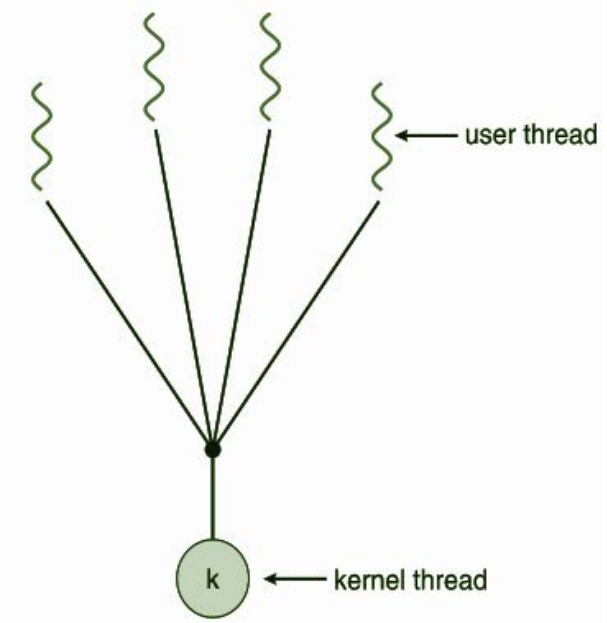
  including:

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

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

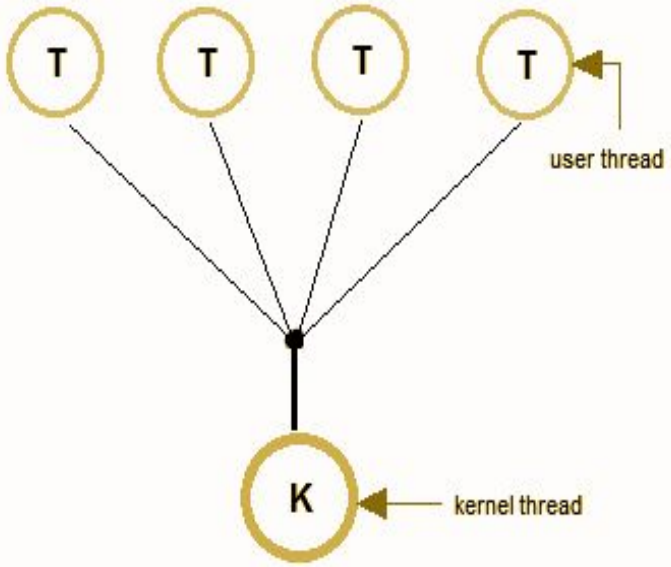
- Examples:
  - **Solaris Green Threads**
  - **GNU Portable Threads**



User Threads - Kernel Threads: Many to One Model

Many to One Model

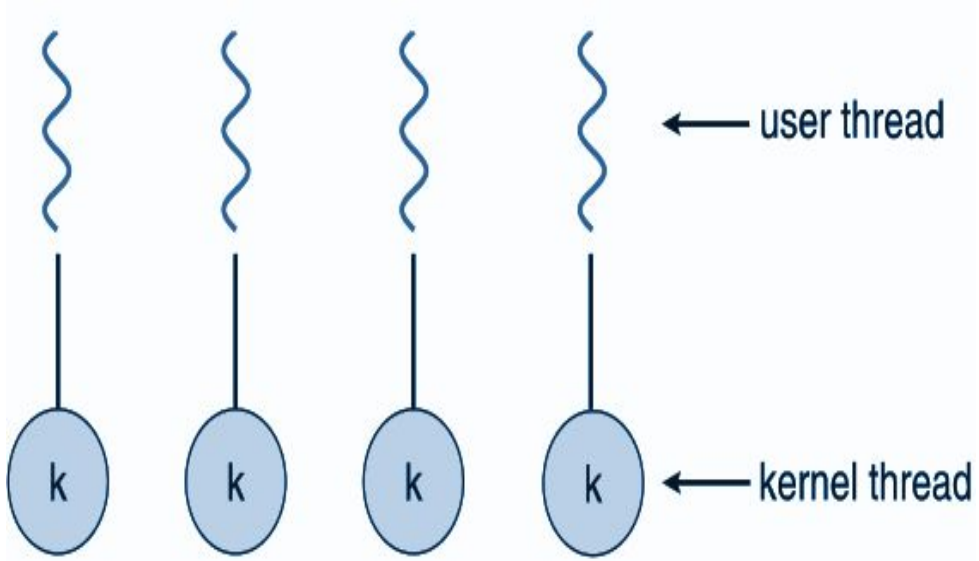
- In the **many to one** model, many user-level threads are all mapped onto a single kernel thread.
- Thread management is handled by the thread library in user space, which is efficient in nature.



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

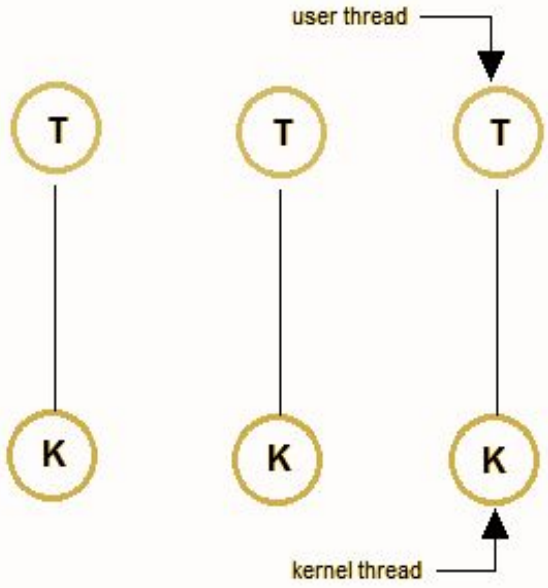
- Examples
  - Windows
  - Linux
  - Solaris 9 and later



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.



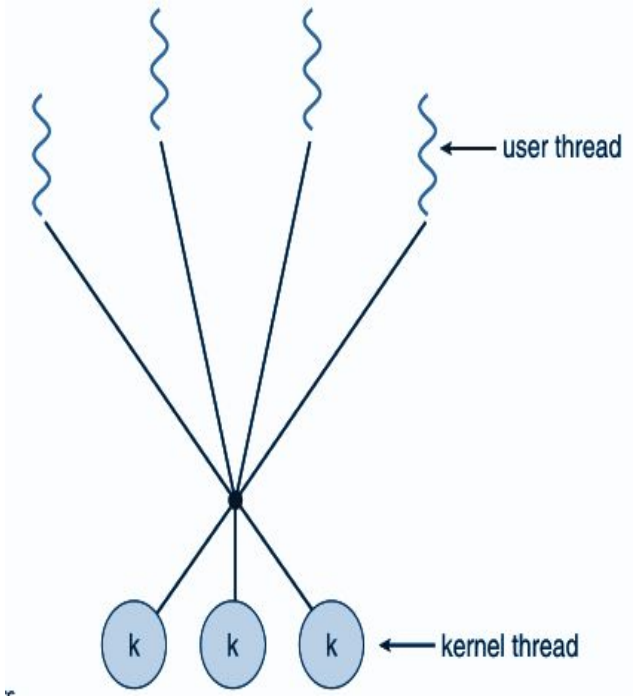
User Threads - Kernel Threads: Many to Many Model

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



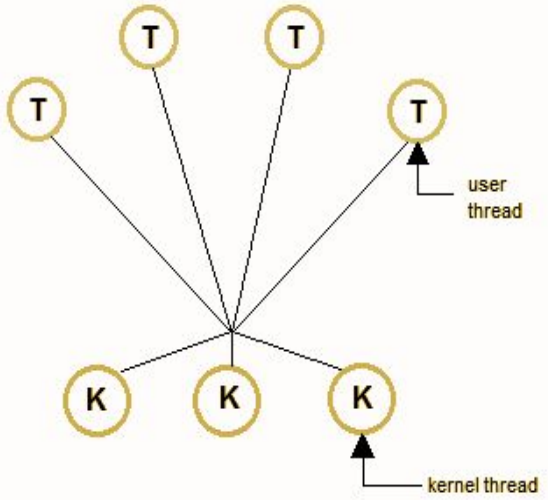
Examples

- 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.

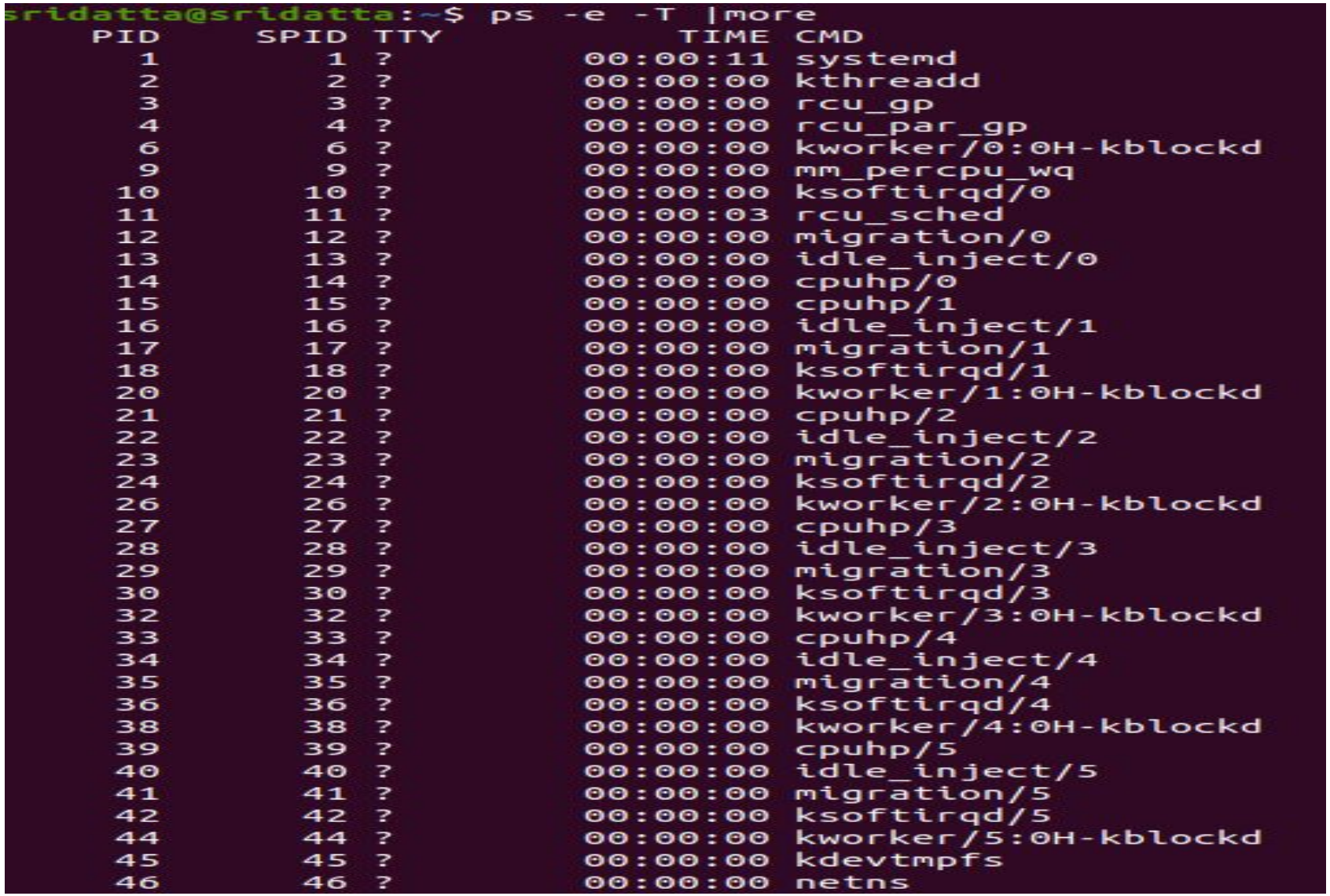
Your max thread count is, 4 CPU x 12 cores x 2 threads per core, so 12 x 4 x 2 is 96. Therefore the max thread count is 96 and max core count is 48.

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).

OPERATING SYSTEMS

Screen Shot



```
sridatta@sridatta:~$ ps -e -T |more
    PID    SPID TTY          TIME CMD
      1       1 ?        00:00:11 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
      9       9 ?        00:00:00 mm_percpu_wq
     10      10 ?        00:00:00 ksoftirqd/0
     11      11 ?        00:00:03 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
     23      23 ?        00:00:00 migration/2
     24      24 ?        00:00:00 ksoftirqd/2
     26      26 ?        00:00:00 kworker/2:0H-kblockd
     27      27 ?        00:00:00 cpuhp/3
     28      28 ?        00:00:00 idle_inject/3
     29      29 ?        00:00:00 migration/3
     30      30 ?        00:00:00 ksoftirqd/3
     32      32 ?        00:00:00 kworker/3:0H-kblockd
     33      33 ?        00:00:00 cpuhp/4
     34      34 ?        00:00:00 idle_inject/4
     35      35 ?        00:00:00 migration/4
     36      36 ?        00:00:00 ksoftirqd/4
     38      38 ?        00:00:00 kworker/4:0H-kblockd
     39      39 ?        00:00:00 cpuhp/5
     40      40 ?        00:00:00 idle_inject/5
     41      41 ?        00:00:00 migration/5
     42      42 ?        00:00:00 ksoftirqd/5
     44      44 ?        00:00:00 kworker/5:0H-kblockd
     45      45 ?        00:00:00 kdevtmpfs
     46      46 ?        00:00:00 netns
```

OPERATING SYSTEMS

Screen Shot

```
pg@chairperson-HP:~$ pidstat -t -p 13325
Linux 4.15.0-112-generic (chairperson-HP)       Friday 28 August 2020   _x86_64_        (4 CPU)

12:02:11  IST   UID      TGID       TID    %usr %system  %guest   %wait    %CPU   CPU  Command
12:02:11  IST  1000     13325         -    2.33    0.00    0.00    0.00    2.33     0  a.out
12:02:11  IST  1000         -     13325    0.00    0.00    0.00    0.00    0.00     0  |__a.out
12:02:11  IST  1000         -     13329    0.16    0.00    0.00    0.46    0.16     1  |__a.out
12:02:11  IST  1000         -     13330    0.15    0.00    0.00    0.47    0.15     2  |__a.out
12:02:11  IST  1000         -     13331    0.14    0.00    0.00    0.47    0.14     3  |__a.out
12:02:11  IST  1000         -     13333    0.12    0.00    0.00    0.46    0.12     0  |__a.out
12:02:11  IST  1000         -     13334    0.11    0.00    0.00    0.47    0.11     0  |__a.out
12:02:11  IST  1000         -     13335    0.11    0.00    0.00    0.47    0.11     0  |__a.out
12:02:11  IST  1000         -     13336    0.09    0.00    0.00    0.46    0.09     2  |__a.out
12:02:11  IST  1000         -     13337    0.09    0.00    0.00    0.46    0.09     1  |__a.out
12:02:11  IST  1000         -     13338    0.09    0.00    0.00    0.46    0.09     1  |__a.out
12:02:11  IST  1000         -     13341    0.08    0.00    0.00    0.43    0.08     2  |__a.out
12:02:11  IST  1000         -     13342    0.08    0.00    0.00    0.43    0.08     2  |__a.out
12:02:11  IST  1000         -     13343    0.08    0.00    0.00    0.43    0.08     2  |__a.out
12:02:11  IST  1000         -     13345    0.06    0.00    0.00    0.41    0.06     0  |__a.out
12:02:11  IST  1000         -     13346    0.06    0.00    0.00    0.41    0.06     3  |__a.out
12:02:11  IST  1000         -     13347    0.06    0.00    0.00    0.41    0.06     3  |__a.out
12:02:11  IST  1000         -     13348    0.05    0.00    0.00    0.38    0.05     0  |__a.out
12:02:11  IST  1000         -     13349    0.05    0.00    0.00    0.38    0.05     3  |__a.out
12:02:11  IST  1000         -     13350    0.05    0.00    0.00    0.38    0.05     3  |__a.out
12:02:11  IST  1000         -     13367    0.04    0.00    0.00    0.35    0.04     0  |__a.out
12:02:11  IST  1000         -     13368    0.04    0.00    0.00    0.35    0.04     1  |__a.out
12:02:11  IST  1000         -     13369    0.05    0.00    0.00    0.35    0.05     3  |__a.out
12:02:11  IST  1000         -     13371    0.04    0.00    0.00    0.32    0.04     0  |__a.out
12:02:11  IST  1000         -     13372    0.04    0.00    0.00    0.32    0.04     0  |__a.out
12:02:11  IST  1000         -     13373    0.04    0.00    0.00    0.32    0.04     2  |__a.out
12:02:11  IST  1000         -     13378    0.03    0.00    0.00    0.29    0.03     3  |__a.out
12:02:11  IST  1000         -     13379    0.03    0.00    0.00    0.29    0.03     2  |__a.out
12:02:11  IST  1000         -     13380    0.03    0.00    0.00    0.29    0.03     1  |__a.out
12:02:11  IST  1000         -     13389    0.03    0.00    0.00    0.26    0.03     0  |__a.out
12:02:11  IST  1000         -     13390    0.03    0.00    0.00    0.26    0.03     3  |__a.out
12:02:11  IST  1000         -     13391    0.03    0.00    0.00    0.26    0.03     0  |__a.out
12:02:11  IST  1000         -     13392    0.02    0.00    0.00    0.23    0.02     0  |__a.out
12:02:11  IST  1000         -     13393    0.02    0.00    0.00    0.23    0.02     3  |__a.out
12:02:11  IST  1000         -     13394    0.02    0.00    0.00    0.23    0.02     1  |__a.out
12:02:11  IST  1000         -     13395    0.02    0.00    0.00    0.20    0.02     3  |__a.out
12:02:11  IST  1000         -     13396    0.02    0.00    0.00    0.20    0.02     0  |__a.out
12:02:11  IST  1000         -     13397    0.02    0.00    0.00    0.20    0.02     1  |__a.out
12:02:11  IST  1000         -     13399    0.01    0.00    0.00    0.16    0.01     3  |__a.out
12:02:11  IST  1000         -     13400    0.01    0.00    0.00    0.16    0.01     2  |__a.out
12:02:11  IST  1000         -     13401    0.01    0.00    0.00    0.16    0.01     1  |__a.out
```

```
File  Edit  View  Search  Terminal  Help
Before creating the threads
 Thread ID 1=>0x7f49577b6700 with process Id=>13325
 Thread ID 2=>0x7f4956fb5700 with process Id=>13325
 Thread ID 3=>0x7f49567b4700 with process Id=>13325
Before creating the threads
 Thread ID 1=>0x7f4955fb3700 with process Id=>13325
 Thread ID 2=>0x7f49557b2700 with process Id=>13325
 Thread ID 3=>0x7f4954fb1700 with process Id=>13325
Before creating the threads
 Thread ID 1=>0x7f49547b0700 with process Id=>13325
 Thread ID 2=>0x7f4953faf700 with process Id=>13325
 Thread ID 3=>0x7f49537ae700 with process Id=>13325
Before creating the threads
 Thread ID 1=>0x7f4952fad700 with process Id=>13325
 Thread ID 2=>0x7f49527ac700 with process Id=>13325
 Thread ID 3=>0x7f4951fab700 with process Id=>13325
Before creating the threads
 Thread ID 1=>0x7f49517aa700 with process Id=>13325
 Thread ID 2=>0x7f4950fa9700 with process Id=>13325
 Thread ID 3=>0x7f49507a8700 with process Id=>13325
Before creating the threads
 Thread ID 1=>0x7f494ffa7700 with process Id=>13325
 Thread ID 2=>0x7f494f7a6700 with process Id=>13325
```

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