**THREAD-PROCESSOR MAPPING**

- T1
- T2
- T3

<table>
<thead>
<tr>
<th>CPU</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 CPUs</td>
<td></td>
</tr>
</tbody>
</table>

**DATABASE SERVER EXAMPLE**

- Worker Threads
- Task List
- Database Cache
- User Space
- Kernel Space

** BENEFITS OF THREADS**

- **Equivalent to multiple processes that can share one address space**
  - But thread creation is much faster (100X?) than process creation
- **Increased Throughput** (work per unit time):Overlap processing and I/O
- **Use Multiple Processors:** Spread load across multiple processors
  - e.g., simulation of a large circuit
- **User Interface Responsiveness**
  - One thread handles GUI while other threads perform operations
- **Server Responsiveness**
  - Adapt to variations in user load and reduce blocking
- **Improved Program Structure in Some Cases**

**THREAD PRIMITIVES**

- **thread_create:** Create a thread
- **thread_exit:** Exit a thread (and destroy thread)
- **thread_yield:** Yield the CPU to another thread and go to sleep
- Get thread identity, get/set thread attributes
- Synchronize between threads
**THREADS STANDARDS**

- Defines an API and behavior of a threads paradigm
  - About 50 function calls
- **POSIX Threads**
  - IEEE 1003.1c (Pthreads)
  - Portable
  - Implementations on almost all Unix systems
  - Not adopted by Microsoft
- **Win32 and OS/2 Threads**
  - Not compatible with Pthreads
  - Proprietary (vendor-specific)
- **Solaris Threads (UI Threads)**
  - Used in Solaris 2 and developed before Pthreads standard was finalized
  - Virtually the same as Pthreads

**THREAD LIBRARY IMPLEMENTATIONS**

- **User-Space**
  - Self-contained user-level library
  - All code and structures are in user-space
  - Depends on a small number of OS system calls
- **Kernel-Space**
  - Thin user-space layer
  - Substantial amount of kernel code and structures

**THREAD ARCHITECTURE**

**MODELS OF KERNEL SCHEDULING**

- **N:1 (Many Threads on One LWP)**
  - Coroutine style; Fast, but no speed-up on a multiprocessor
  - Thread creation, scheduling, and synchronization all done in user space
- **1:1**
  - Many threads can run simultaneously on different CPUs
  - Allows 1 or more threads to issue blocking system calls while others run, even on a uniprocessor
  - Thread creation requires LWP creation (and a system call)
  - Each LWP takes up kernel resources ⇒ Limited total number of threads
MODELS OF KERNEL SCHEDULING

- **Strict M:N (M ≥ N)**
  - Thread creation, scheduling, and some synchronization done in user space
- **M:N + 1:1**
  - Combines the best of M:N and 1:1
  - Used in Solaris, Digital Unix, IRIX, HP-UX
  - Win32 fibers is a rough approximation

![Diagram showing threads and kernel space](image)

THREAD SCHEDULING

- **Local Scheduling (Process Contention Scope (PCS))**
  - Scheduling done by the threads library
    - Very fast except for preemption (requires system call)
  - Scheduling of the LWP is global, but is independent of the local scheduling
  - Scheduling is by thread priority
    - Set by programmer; not adjusted by threads library

- **Global Scheduling (System Contention Scope (SCS))**
  - Scheduling done by the OS kernel
  - Thread blocks ⇒ LWP goes to sleep

THREAD STATE TRANSITIONS

![Diagram showing thread state transitions](image)