**Heavyweight-processes**

The cooperation of traditional processes also known as heavyweight processes which have the following characteristics:

* They require space in main memory where they reside during their execution.
* They pass through several states from their initial entry into the computer system to their completion: ready, running, waiting, new, etc.
* These processes are often swapped between main memory and secondary memory during their execution to accommodate multiprogramming and to take advantage of virtual memory.

**Every** time a swap occurs **overhead increase because of all the information must be saved**. To minimize this **overhead time** the operating system support the implementation of threads or lightweight processes. Threads are supports at both kernel and user level and can be managed by either **operating system** or the **application** that created them.

The aims of having multiple threads of execution are the following:

* + - To maximize degree of **concurrent** execution between operations
    - To enable the **overlap** of computation with input and output
    - To enable concurrent processing on multiprocessors.

**Concurrent Execution on a Single-core System**

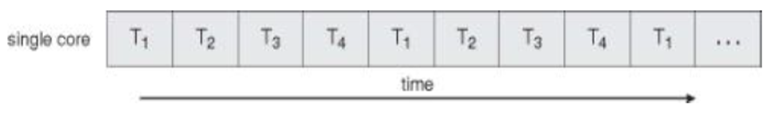
****

Figure (1): Concurrent Execution on a Single-core System

**Parallel Execution on a Multi-core System**

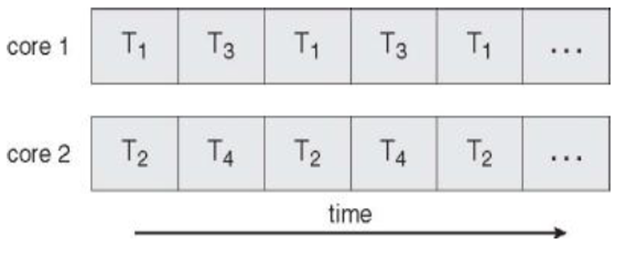


Figure (2): Parallel Execution on a Multi-core System

**Thread stats:**

* **New**: create new thread.
* **Runnable**:
* Ready: Ready to execute
* Running: Thread in execution.
* **Waiting:** Thread Wait until a specific event has occurred.
* **Timed Waiting:** Thread putting for specified amount of time.
* **Terminated:** Terminate execution and release resources.
* **Blocked:** Waiting for I/O to complete.

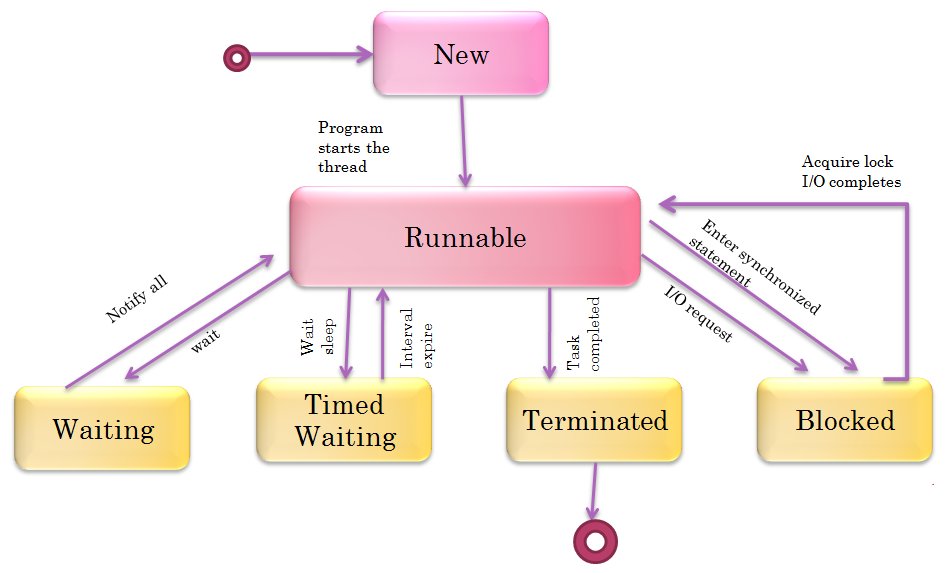


Figure (3): Threads States

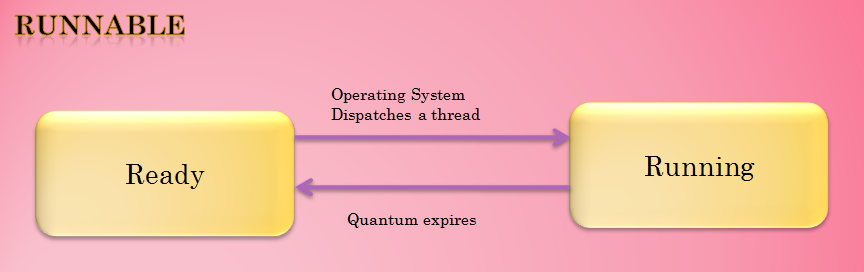


Figure (4): Operating system’s internal view of java’s Runnable state

**Thread Control Block (TCB)**

It contains basic information about a thread such as:

* + Thread ID.
  + Thread state.
  + CPU information.
  + Thread priority.
  + A pointer to the process that created the thread.

**Types of Threads:**

1. User Threads: Thread management done by user-level threads library, Three primary thread libraries:
   * POSIX Pthreads.
   * Win32 threads.
   * Java threads.
2. Kernel Thread: Supported by the Kernel, the examples as the following:
   * Windows XP/2000.
   * Solaris.
   * Linux.
   * Tru64 UNIX.
   * Mac OSX.

**Multithreading Models:**

1. **Many-to-One Model:** maps many user-level threads to one kernel thread. Examples: lSolaris Green Threads, lGNU Portable Threads.
2. **One-to-One Model:** maps each user thread to a kernel thread. Examples: Windows NT/XP/2000, Linux, Solaris 9 and later
3. **Many-to-Many Model:** multiplexes many user-level threads to a smaller or equal number of kernel threads. Examples: Solaris prior to version 9, Windows NT/2000 with the *ThreadFiberpackage.*

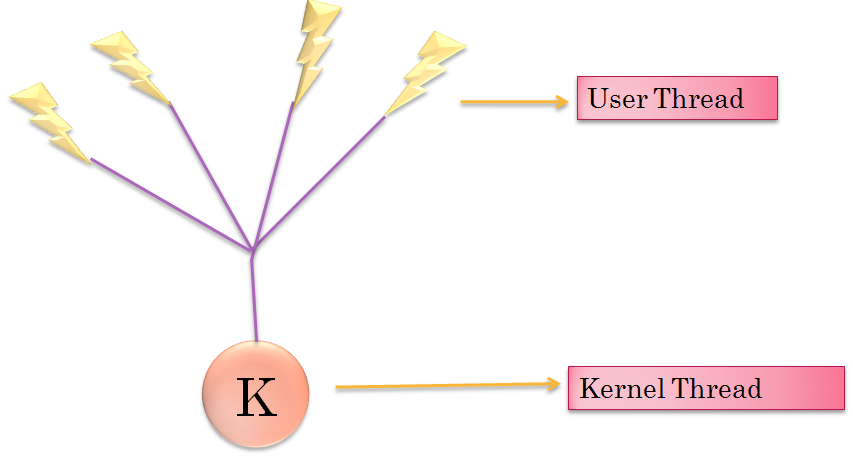


Figure (5): Multithreading Models (Many-to-One):

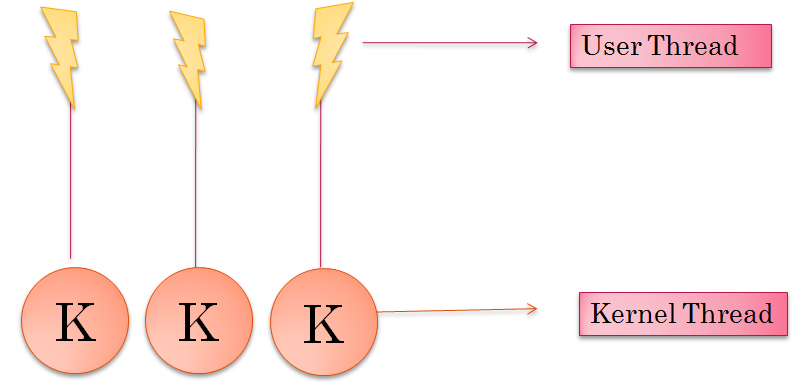
****

Figure (6): Multithreading Models (One-to-One):

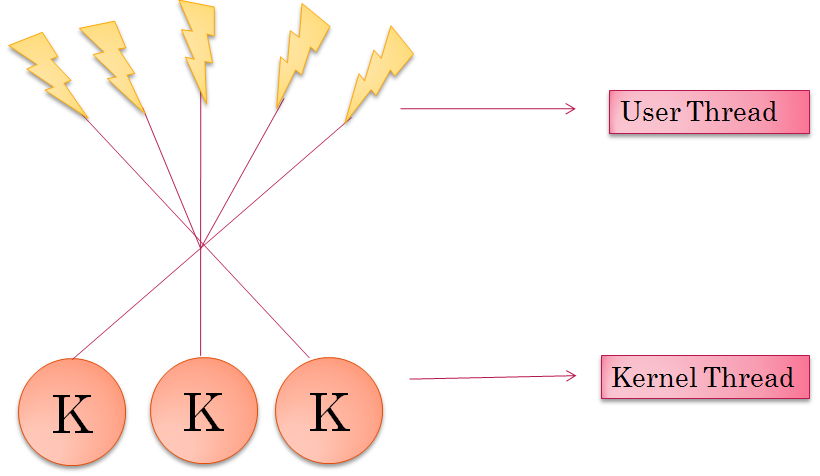


Figure (7): Multithreading Models (Many-to-Many)