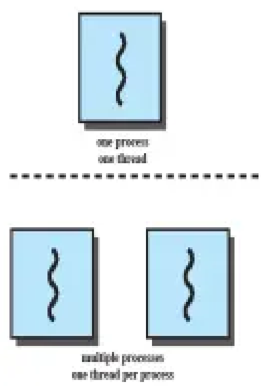
Chapter 4

Processes and Threads

- **Resource Ownership**
  - Includes a virtual address space that hold the process image(PCB)
  - OS performs a protection function to prevent interference between processes resources
- **Scheduling/Execution**
  - Execution path that maybe interleaved with other processes
  - Process has an execution state and a dispatching priority and that is controlled by the OS
- **Thread or Lightweight Process**: Unit of dispatching
- **Process or Task**: The unit of resource ownership
- **Multi-threading**: Ability of the OS to support multiple, concurrent paths of execution within a single process

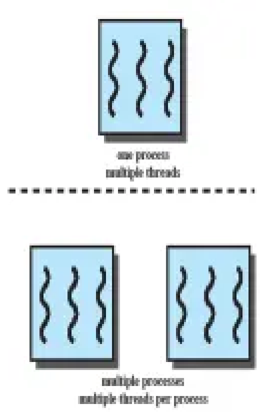
Single Threaded Approaches

- Single thread of execution per process is referred to as a single threaded approach
- MS-DOS is an example



Multi-threaded Approach

- A Java run-time environment is an example of a system of one process with multiple threads



Process

- The unit of *resource allocation* and a unit of *protection*
- A virtual address space that holds the process image
- Protected access to:
  - Processors

- Other processes
- files
- I/O resources

One of More Threads in a Process

- Each Thread has:
  - execution state
  - saved thread context when not running
  - execution stack
  - per-thread static storage for local variables
  - access to the memory and resources of its process (shared amongst threads)

Threads vs Processes

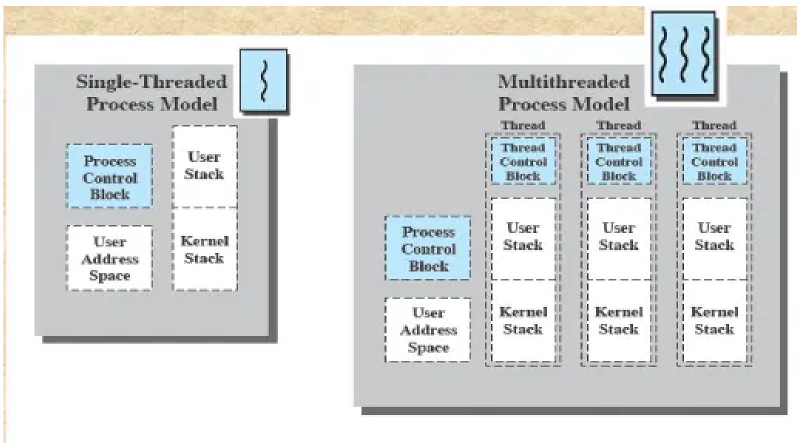


Figure 4.2 Single Threaded and Multithreaded Process Models

Benefits of Threads

- Takes less time to create a new thread
- Less time to terminate a thread
- Switching between 2 threads takes less time
- Threads enhance efficiency in communication between programs

Thread use in a single-user system

- Foreground and Background work
- Asynchronous processing
- Speed of execution
- Modular program structure

Threads

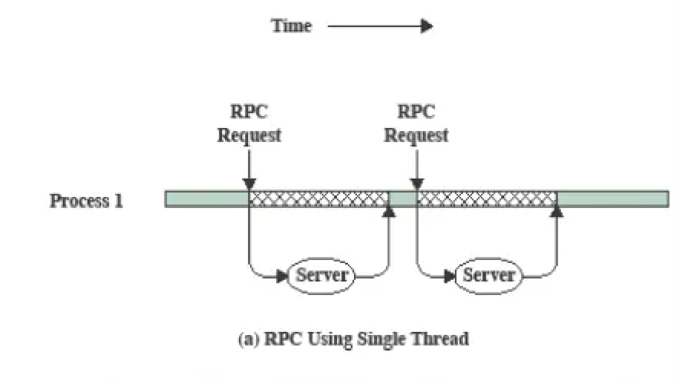
- Scheduling and dispatching is done on a thread basic on the OS that supports threads
- State information dealing with execution is stored in a thread level data structure
  - *suspending* a process involves suspending all threads of the process
  - *terminating* of a process terminates all threads within the process

Thread Execution State

- Key states
  - *Running*
  - *Ready*
  - *Blocked*

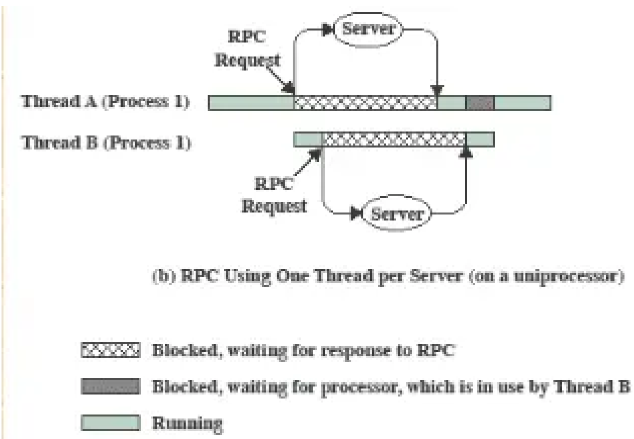
- Thread operations associated with a change in thread state are
  - *Spawn*
  - *Block*
  - *Unblock*
  - *Finish*

RPC Using Single Thread



(a) RPC Using Single Thread

RPC Using One Thread per Server



(b) RPC Using One Thread per Server (on a uniprocessor)

Multi-threading on a Uni-processor

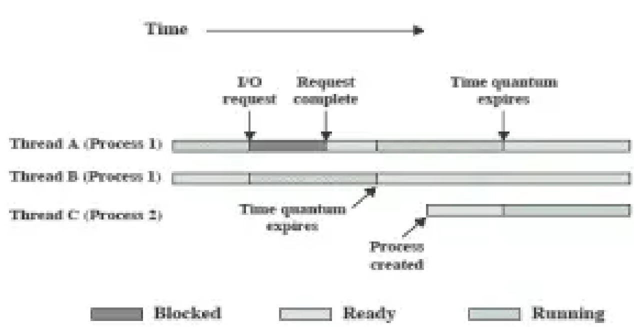


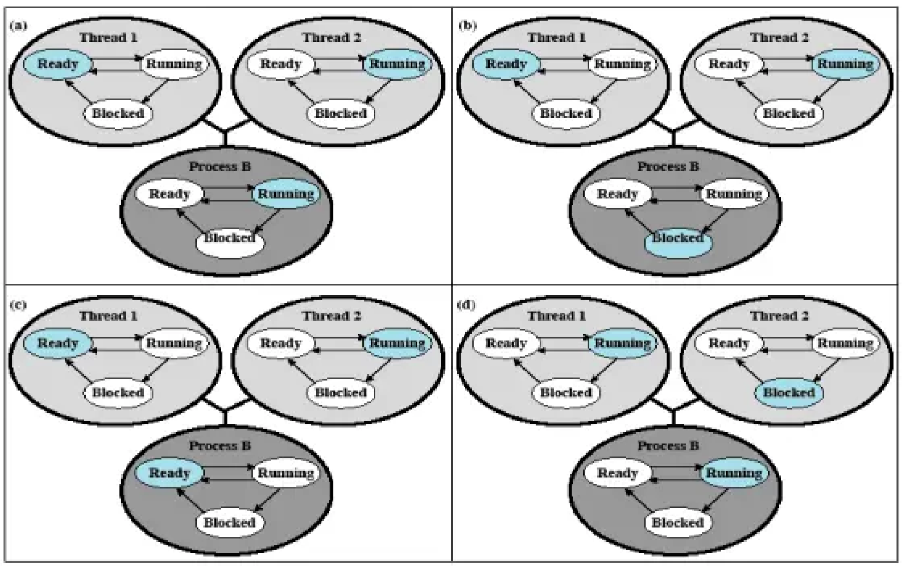
Figure 4.4 Multithreading Example on a Uniprocessor

Thread Synchronization

- Necessary to sync activities of the various threads
  - all threads of a process share the address space and other resources
  - any alteration will affect the other threads in the same process

Types of Threads

- **User Level Thread**
  - All thread management is done by the application
  - Kernel not aware of its existence
  - **Advantages**
  - **Thread switching does not require kernel mode privileges**
  - **Scheduling can be application specific**
  - **ULT's can run on any OS**
  - Disadvantages*
  - *System calls are blocked which in turn also blocks all other threads within the process*
  - *A multi-threading application cannot take advantage of multiprocessing if it is purely ULT*
  - Overcoming ULT Disadvantages
  - *converts a blocking system call into a non-blocking system call* - jacketing
  - *writing an application as multiple processes rather than multiple threads*
  - Relation between ULT states and Process States*



- **Kernel Level Thread**
  - Thread management is done by the kernel
    - no thread management is done by the application i.e. windows
  - **Advantages of KTLs**
    - Kernel can simultaneously schedule multiple threads from the same process on multiple processors
    - If one thread is blocked, the kernel reschedules another thread of the same process
    - routines can be multithreaded
  - **Disadvantage**
    - Transfer of control from one thread to another within the same process requires mode switch to the kernel

Operation	User-Level Threads	Kernel-Level Threads	Processes
Null Fork	34	948	11,300
Signal Wait	37	441	1,840

Combined Approaches (i.e. Solaris)

- Thread creation is done in the user space
- Bulk of scheduling and syncing is done by application

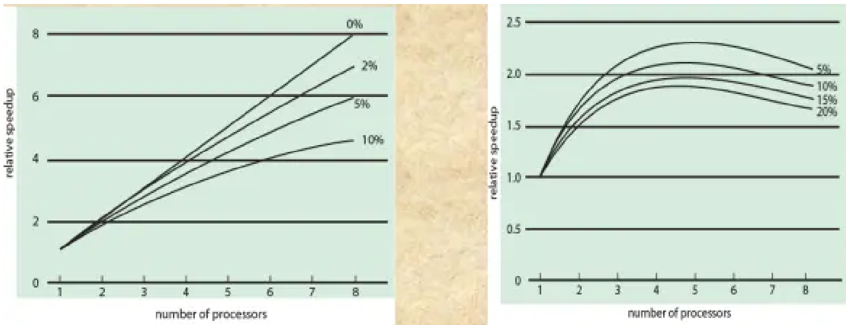
Relation between threads and processes

- *1:1* : Each thread of execution is a unique process with its own address space and resources. Unix

- *M:1*: Defined address space and dynamic resource ownership. Multiple threads created and executed. Windows, Linux
- *1:M*: Migrated thread from one process to another. Allows a thread to be easily moved among different systems. Ra, Emerald
- *M:N*: Combination of *M:1 and 1:M*. TRIX

Multi-threading and Multicore

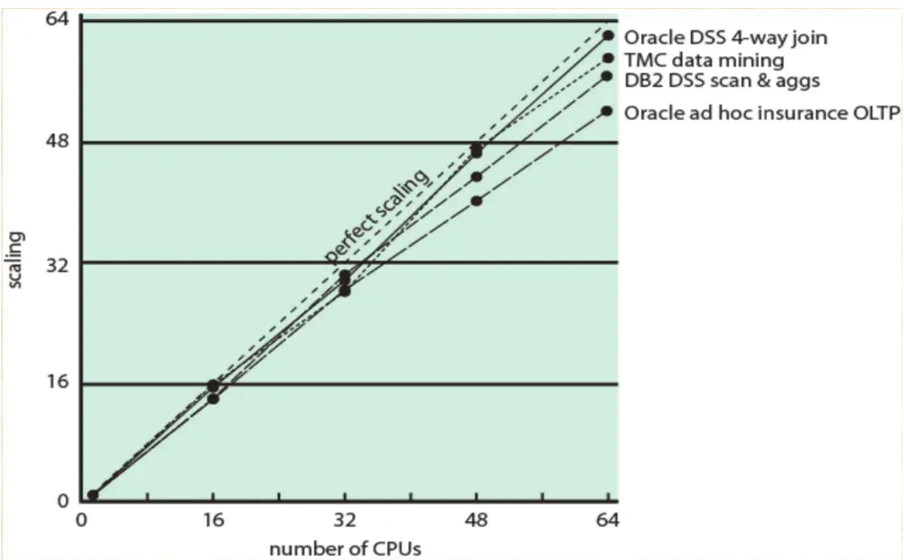
$$\text{Speedup} = \frac{\textit{time to execute on 1 processor}}{\textit{time to execute on N parallel processors}} = \frac{1}{(1-f)+\frac{f}{N}}$$



(a) Speedup with 0%, 2%, 5%, and 10% sequential portions

(b) Speedup with overheads

Database Workloads on Multiple-Processor Hardware



Linux Tasks

- A process or task in Linux is represented by a task_struct data structure. It contains
  - *State* => executing, ready, blocked
  - *Scheduling Info* => priority, time slice allowed
  - *Identifiers* => PID, userID, groupID
  - *IPC*
  - *Links* => to parent, sibling, children
  - *Times/Timers* => processor time used so far, interval timer
  - *File system* => pointers to opened files, current directory of process
  - *Address Space* => program, data

- *Context* => registers, stack

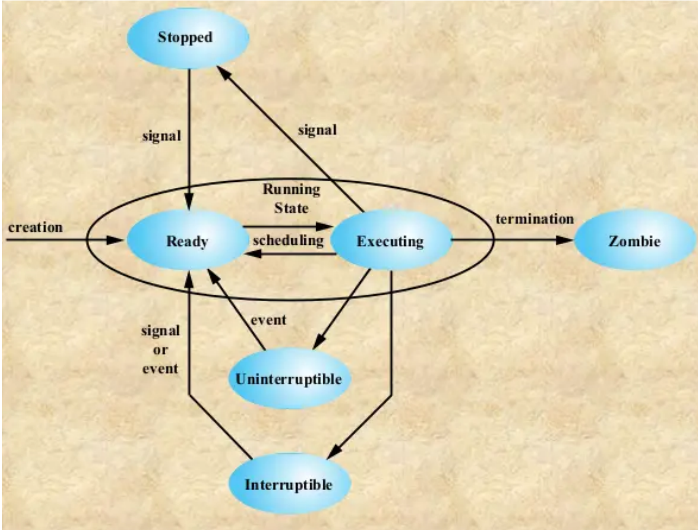


Figure 4.15 Linux Process/Thread Model

Linux Threads

- Linux does not recognise a distinction between threads and processes
- New process is created by copying the attributes of the current process (fork or clone)
- new process can be cloned so that the it shares resources (address space (VM), signal handlers, files, IO etc)
- Processes sharing same VM operate as threads within a single process
- Both clone & pthreads make what text calls KLTs