Chapter 4 Lectures

Stallings - 9ed
Process

• Two characteristics:
  – Resource ownership
    • process includes a virtual address space to hold the process image
  – Scheduling/execution
    • follows an execution path that may be interleaved with other processes
  – These two characteristics are treated independently by the OS
Process

• **process**
  – sometimes referred to as *task* or *job*
  – refers to resource of ownership
  – (addresses the 1st characteristic)

• **thread** or **lightweight process**
  – this is the unit of dispatching
  – (addresses the 2nd characteristic)
Multithreading

- Operating system supports multiple threads of execution within a single process
  - MS-DOS supports a single thread
  - UNIX supports multiple user processes but only supports one thread per process
  - Windows, Solaris, Linux, Mach, OS X, and OS/2 support multiple threads
    - e.g. OS X 10.6 (snow leopard) offers POSIX threads (or pthreads, POSIX 1003.1c standard), and Cocoa threads
Figure 4.1  Threads and Processes

- One process, one thread
- One process, multiple threads
- Multiple processes, one thread per process
- Multiple processes, multiple threads per process

\{ \} = instruction trace
Process

• In multithreaded environment a **process** is the unit of resource allocation and a unit of protection

• Processes
  – Have a virtual address space which holds the process image
  – Protected access to processors, other processes, files, and I/O resources
Thread

- Within a process there are one or more threads, each with the following:
  - an execution state (running, ready, etc.)
  - a saved thread context when not running
    - may view a thread as an independent program counter operating within a process
  - an execution stack
  - some per-thread static storage for local variables
  - access to the memory & resources of its process
    - all threads of a process share this
Figure 4.2  Single Threaded and Multithreaded Process Models
Benefits of Threads

- Takes less time to create a new thread than a process
- Less time to terminate a thread than a process
- Less time to switch between two threads within the same process
- Since threads within the same process share memory and files, they can communicate with each other without invoking the kernel
Threads in a Single-User Multiprocessing System

• Foreground to background work
  – e.g. spreadsheet, multiple threads display menues, read user input, update spreadsheet etc.

• Asynchronous processing
  – e.g. thead in word processor to periodically flush RAM to disk
Threads in a Single-User Multiprocessing System

• Speed of execution
  – e.g. a process may compute one batch of data while reading in the next.
  – in multiprocessor: true parallel execution of threads in a process

• Modular program structure
  – thread model can be used to “group” activities of process
Threads

• Suspending a process
  – suspends all threads of the process since all threads share the same address space

• Termination of a process
  – terminates all threads within the process
Thread States

• States of a thread
  – Spawn
    • when process is spawned
    • thread may spawn other threads
    • each thread has its own:
      – register context, state space, and place in ready queue
  – Block
    • when thread waits for event
      – saves user registers, PC and stack pointer
Thread States

• States of a thread
  – Unblock
    • when blocking event occurs
    • thread is moved to ready queue
  – Finish
    • register context and stack is deallocated
Remote Procedure Call Using Single Thread

What is a RPC?
Remote Procedure Call Using Threads

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

- Blocked, waiting for response to RPC
- Blocked, waiting for processor, which is in use by Thread B
- Running

Figure 4.3 Remote Procedure Call (RPC) Using Threads
Figure 4.4    Multithreading Example on a Uniprocessor
Basic questions

• What is the difference between this and multiprocessing?
  – kind of looks the same, or...?

• Is there a need to synchronize threads?
  – e.g. two threads insert an element into a linked structure
User-Level Threads (ULT)

• All thread management is done by the application
  – e.g. using threads library
• The kernel is not aware of the existence of threads
User-Level Threads
Figure 4.6 Examples of the Relationships Between User-Level Thread States and Process States
Kernel-Level Threads (KLT)

- Often called *lightweight processes*
- Windows is an example of this approach
- Kernel maintains context information for the process and the threads
- Scheduling is done on a thread basis
Kernel-Level Threads

(b) Pure kernel-level
Combined Approaches

• Thread creation is done in user space
• Bulk of scheduling and synchronization of threads done within application

• Example is Solaris
Combined Approaches
## Relationship Between Threads and Processes

### Table 4.2  Relationship Between Threads and Processes

<table>
<thead>
<tr>
<th>Threads:Processes</th>
<th>Description</th>
<th>Example Systems</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:1</td>
<td>Each thread of execution is a unique process with its own address space and resources.</td>
<td>Traditional UNIX implementations</td>
</tr>
<tr>
<td>M:1</td>
<td>A process defines an address space and dynamic resource ownership. Multiple threads may be created and executed within that process.</td>
<td>Windows NT, Solaris, Linux OS/2, OS/390, MACH</td>
</tr>
<tr>
<td>1:M</td>
<td>A thread may migrate from one process environment to another. This allows a thread to be easily moved among distinct systems.</td>
<td>Ra (Clouds), Emerald</td>
</tr>
<tr>
<td>M:N</td>
<td>Combines attributes of M:1 and 1:M cases.</td>
<td>TRIX</td>
</tr>
</tbody>
</table>
Advantages of ULT over KLT

• thread switching does not require kernel mode privileges
  – saves two mode switches (user-to-kernel and kernel-to-user)

• application specific scheduling
  – applications may prefer their own specific scheduling algorithm

• ULT can run on any OS
Disadvant. of ULT vs KLT

• Many OS system calls are blocking.
  – so if ULT executes such call all threads within its process are blocked

• In pure ULT strategy a multithreaded application cannot take advantage of multiprocessing
  – no concurrency
Figure 4.7 Performance Effect of Multiple Cores

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

(b) Speedup with overheads
Figure 4.8 Scaling of Database Workloads on Multiple-Processor Hardware
Figure 4.9 Hybrid Threading for Rendering Module
Windows Processes

• Implemented as objects
• An executable process may contain one or more threads
• Both processes and thread objects have built-in synchronization capabilities
Figure 4.10  A Windows Process and Its Resources
Windows Process Object

Object Type

- Process ID
- Security Descriptor
- Base priority
- Default processor affinity
- Quota limits
- Execution time
- I/O counters
- VM operation counters
- Exception/debugging ports
- Exit status

Object Body Attributes

- Create process
- Open process
- Query process information
- Set process information
- Current process
- Terminate process

(a) Process object
Windows 2000
Thread States

- Ready
- Standby
- Running
- Waiting
- Transition
- Terminated
Figure 4.11  Windows Thread States
Solaris

- Process includes the user’s address space, stack, and process control block
- User-level threads
- Lightweight processes (LWP)
- Kernel threads
Figure 4.12 Processes and Threads in Solaris
Figure 4.15 Solaris Multithreaded Architecture Example
Figure 4.13  Process Structure in Traditional UNIX and Solaris [LEWI96]
Solaris Lightweight Data Structure

- Identifier
- Priority
- Signal mask
- Saved values of user-level registers
- Kernel stack
- Resource usage and profiling data
- Pointer to the corresponding kernel thread
- Pointer to the process structure
Figure 4.14  Solaris Thread States
Linux Task Data Structure

- State
- Scheduling information
  - normal or real-time, priorities
- Identifiers
- Interprocess communication
- Links
- Times and timers
- File system
- Address space
- Processor-specific context
Linux States of a Process

- Running
- Interruptable
- Uninterruptable
- Stopped
- Zombie
Figure 4.15    Linux Process/Thread Model