Threads, SMP, and Microkernels

Chapter 4
Processes

✓ Resource ownership - process is allocated a virtual address space to hold the process image

✓ Dispatched - process is an execution path through one or more programs
  ▪ execution may be interleaved with other processes

✓ These two characteristics are treated independently by the operating system
Processes

✔ Dispatching is referred to as a *thread*
✔ Resource of ownership is referred to as a *process* or *task*
Multithreading

 ✓ Operating system supports multiple threads of execution within a single process
 ✓ **MS-DOS** supports just one process and a single thread
 ✓ **Traditional UNIX** supports multiple user processes but only one thread per process
 ✓ **Modern Unix** (**Solaris, Linux, AIX**) and **Windows** (**2000/XP**) support multiple threads per process
Threads and Processes

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

- Have a virtual address space which holds the process image
- Protected access to processors, communication lines to other processes, files, and I/O resources (devices, channels)
Thread Resources

- An execution state (running, ready, etc.)
- Saved thread context when not running
- An execution stack
- Per-thread static storage for local variables
- Access to the memory and other resources of the owner-process
  - all threads of a process share the resources/memory of the owner-process
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
Suspension and Termination of Threads

- Suspending a process involves suspending all threads of the process since all threads share the same address space.
- Termination of a process, terminates all threads within the process.
User-Level Threads

✓ All thread management is done by the application
✓ The kernel is not aware of the existence of threads
✓ Thread switching does not require kernel mode privileges
✓ Scheduling is application specific
Kernel-Level Threads

- Windows 2000/XP, Modern UNIXes are examples of this approach
- Kernel maintains context information for the process and the threads
- Switching between threads requires the kernel
Combined Approaches for Threads

✓ Example is Solaris (Sun’s Unix)
✓ Thread creation is done in the user space
✓ Bulk of scheduling and synchronization of threads is done in the user space
# Relationship Between Threads and Processes

<table>
<thead>
<tr>
<th>Threads:Process</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>Linux, Windows XP, Solaris, OS/2, OS/390, MACH</td>
</tr>
</tbody>
</table>
## Relationship Between Threads and Processes

<table>
<thead>
<tr>
<th>Threads:Process</th>
<th>Description</th>
<th>Example Systems</th>
</tr>
</thead>
<tbody>
<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:M</td>
<td>Combines attributes of M:1 and 1:M cases</td>
<td>TRIX</td>
</tr>
</tbody>
</table>
Categories of Computer Systems

✓ Single Instruction Single Data (SISD)
  - single processor executes a single instruction stream to operate on data stored in a single memory

✓ Single Instruction Multiple Data (SIMD)
  - one instruction is executed on different sets of data by the different processors
Categories of Computer Systems

✓ Multiple Instruction Single Data (MISD)
  - a sequence of data is transmitted to a set of processors, each of which executes a different instruction sequence. Never implemented

✓ Multiple Instruction Multiple Data (MIMD)
  - a set of processors simultaneously execute different instruction sequences on different data sets
Symmetric Multiprocessing

✓ Kernel can execute on any processor
✓ Typically each processor does self-scheduling from the pool of available processes or threads
Symmetric Multiprocessor Organization
Microkernel

✓ Small operating system core
✓ Contains only essential operating systems functions
✓ Many services traditionally included in the operating system are now external subsystems
  ▪ device drivers
  ▪ file systems
  ▪ virtual memory manager
  ▪ windowing system and security services
Benefits of a Microkernel Organization

✓ Uniform interface to requests made by a process
  ▪ all services are provided by means of message passing

✓ Extensibility
  ▪ allows the addition of new services

✓ Flexibility
  ▪ existing features can be subtracted
Benefits of a Microkernel Organization

✓ Portability
  ▪ changes needed to port the system to a new processor can be limited to the microkernel - not to the other services

✓ Reliability
  ▪ modular design
  ▪ small microkernel can be rigorously tested
Benefits of Microkernel Organization

✓ Distributed system support
  ▪ messages are sent without knowing what the target machine is

✓ Object-oriented operating system
  ▪ components are objects with clearly defined interfaces that can be interconnected to form software
Microkernel Design

- Primitive memory management
  - mapping each virtual page to a physical page frame
- Inter-process communication
- I/O and interrupt management
MS Windows Processes

✓ Implemented as objects
✓ An executable process may contain one or more threads
✓ Both process and thread objects have built-in synchronization capabilities
Windows Process Object Attributes

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

- Thread ID
- Thread context
- Dynamic priority
- Base priority
- Thread processor affinity
- Thread execution time
- Alert status
- Suspension count
- Impersonation token
- Termination port
- Thread exit status
Windows Thread States

Runnable

Pick to Run

Preempted

Switch

Ready

Runnable

Resource Available

Unblock/Resume

Resource Available

Block/Suspend

Terminate

Not Runnable

Unblock, Resource Not Available

Transition

Waiting

Terminate

Running
Solaris

✓ Process includes the user’s address space, stack, and process control block
✓ User-level threads
✓ Lightweight processes
✓ Kernel threads
Solaris User Level Threads

- Stopped
- Runnable
- Active
- Sleeping

Actions:
- Stop
- Continue
- Preempt
- Wakeup
- Dispatch
- Sleep

States:
- Stopped
- Runnable
- Active
- Sleeping
Solaris Lightweight Processes

Diagram:
- **Timeslice or Preempt** directly to **Running**
- **Dispatch** from **Running** to **Runnable**
- **Wakeup** from **Runnable** to **Active**
- **Blocking System Call** from **Active** to **Stopped**
- **Continue** from **Stopped** to **Active**
- **Stop** from **Active** to **Stopped**
- **Stop** from **Stopped** to **Running**
- **Wakeup** from **Stopped** to **Runnable**
Linux Threads

✓ Linux threads appear as processes to the kernel: it doesn’t distinguish that much among them

✓ But processes can share the same process group ID
  - Processes with the same group ID share resources
    - Memory
    - files