Threads, SMP, and Microkernels

Chapter 4
Contents

- Processes and threads
- Symmetric multiprocessing
- Microkernels
- Window 2000 thread and SMP management
- Solaris thread and SMP management
- Linux process and thread management
Processes of yesterday

- Unit of resource ownership
  - process is allocated a virtual address space to hold the process image

- Unit of dispatching
  - scheduled and dispatched by the OS
  - execution may be interleaved with other processes

- These two characteristics are treated independently by today’s operating system
Processes of today

- Process or task
  - unit of resource ownership
  - virtual address space, main memory, I/O devices, and files

- Thread or light weight process
  - unit of dispatching
  - scheduled and dispatched by the OS
Multithreading

- Refers to the ability of an OS to support multiple threads of execution within a single process.
- MS-DOS supports a single user process and a single thread.
- UNIX supports multiple user processes but only supports one thread per process.
- Windows 2000, Solaris, Linux, Mach, and OS/2 support multiple threads.
Figure 4.1  Threads and Processes [ANDE97]
Processes in Multithreaded environment

▸ A process is the unit of resource allocation and a unit of protection
  ◀ have a virtual address space that holds the process image
  ◀ protected access to processors, other processes, files, and I/O resources

▸ Per Process Items
  ◀ address space, global variable, open files, child processes, timers, signals, semaphores, account
Threads

- Within a process, there may be one or more threads, each with the followings:
  - thread execution state (running, ready,...)
  - saved thread context
    - program counter, stack, register set, child threads, memory for local variables
  - access to the memory and 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
  - no need to allocate a virtual address space
- 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
Uses of Threads in a Single-User Multiprocessing System

- Foreground and background work
  - one thread displays menus and read user input
  - another thread executes user commands
- Asynchronous elements in the program can be implemented as threads
- Speed execution
  - on a multiprocessor system, multiple threads from the same process may be able to execute simultaneously
- Modular program structure
Threads

Actions that affect all of the threads in a process:

- 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 that process.
Thread States

Operations associated with a change in thread state

- Spawn
  - A thread within a process may spawn another thread within the same process
- Block
- Unblock
- Finish
  - Deallocate register context and stacks
Remote Procedure Call Using Threads

(a) RPC Using Single Thread

- 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
Remote Procedure Call Using Threads

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

- User-level threads (ULTs)
- Kernel-level threads (KLTs)
  - also referred to as kernel-supported threads or lightweight processes
- Combined approaches
Figure 4.6  User-Level and Kernel-Level Threads
User-Level Threads

- All thread management is done by the application
  - kernel is not aware of the existence of threads

- Advantages of ULTs
  - thread switching does not require kernel mode privileges
  - scheduling is application specific
  - ULTs can run on any OS
Figure 4.7 Examples of the relationship between User-level thread states and process states
User-Level Threads

Disadvantages of ULTs

- when a thread is blocked, all of the threads within that process are blocked
- cannot take advantage of multiprocessing
  - kernel assigns one process to only one processor at a time
Kernel-Level Threads

- Windows 2000 and OS/2 are examples of this approach
- Kernel maintains the context information for the process and the threads
- Scheduling by the kernel is done on a thread basis
Combined Approaches for Threads

- Example is Solaris
- Thread creation done in the user space as is bulk of scheduling and synchronization
- Multiple ULTs from a single application are mapped onto some (smaller or equal) number of KLTs
# 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>
</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 (by Flynn)

- **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 a different set of data by the different processors
  - vector and array 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
Figure 4.7 Parallel Processor Architectures

- Parallel Processor
  - SIMD (single instruction multiple data stream)
    - Shared-Memory (tightly coupled)
      - Master/Slave
    - Symmetric Multiprocessors (SMP)
  - MIMD (multiple instruction multiple data stream)
    - Distributed-Memory (loosely coupled)
      - Clusters
Symmetric Multiprocessing

- Kernel can execute on any processor
- Typically each processor does self-scheduling from the pool of available processes or threads
Figure 4.9 Symmetric Multiprocessor Organization
Multiprocessor Operating System Design Considerations

- Simultaneous concurrent processes or threads
  - Kernel routines need to be reentrant
- Scheduling
  - May be performed by any processor
- Synchronization
- Memory Management
- Reliability and Fault Tolerance
Microkernel

- Small operating system core
- Contains only essential operating system functions
- Many services traditionally included in the operating system are now external subsystems
  - device drivers
  - file systems
  - virtual memory manager
  - windowing system
  - security services
Figure 4.10  Kernel Architecture
Benefits of a Microkernel Organization

- **Uniform interface on request made by a process**
  - all services are provided by means of message passing
- **Extensibility**
  - allows the addition of new services
- **Flexibility**
  - not only can new features be added to OS, but existing features can be subtracted
Benefits of a Microkernel Organization

- **Portability**
  - almost all processor-specific code is in the microkernel
  - changes needed to port the system to a new processor are fewer and tend to be arranged in logical groupings

- **Reliability**
  - modular design
  - small microkernel can be rigorously tested
Benefits of Microkernel Organization

- Distributed system support
  - message can be 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

- Functions that must be included
  - Low-level memory management
    - mapping each virtual page to a physical page frame
  - Inter-process communication
    - message is the basic form
    - message passing between separate processes involves memory-to-memory copying
      - current research on thread-based IPC and memory sharing scheme
  - I/O and interrupt management
Windows 2000 Processes

- Implemented as objects
- An executable process may contain one or more threads
- Both process and thread objects have built-in synchronization capabilities
Process and its Resources

- Security access token
  - used to validate the user’s ability to access secured objects
- Virtual address space
  - a series of blocks
- Object table
  - have handles to other objects known to this process
  - one handle exists for each thread contained in this object
Figure 4.12 Windows 2000 Process and Its Resources
Windows 2000
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

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

(a) Process object
Windows 2000
Thread Object

Object Type
- 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

Object Body Attributes

Services
- Create thread
- Open thread
- Query thread information
- Set thread information
- Current thread
- Terminate thread
- Get context
- Set context
- Suspend
- Resume
- Alert thread
- Test thread alert
- Register termination port
Multithreading

- Threads in different processes may execute concurrently.
- Multiple threads within the same process may be allocated to separate processors and execute concurrently.
- Threads in different processes can exchange information through shared memory that has been set up between the two processes.
Windows 2000
Thread States

- Ready
- Standby: selected to run next
- Running
- Waiting: blocked
- Transition: ready to run but the resources are not available (e.g., stack may be paged out of memory)
- Terminated
Figure 4.14  Windows 2000 Thread States
**Solaris**

- Thread-related concepts used by Solaris
  - **Process**
    - includes the user’s address space, stack, and process control block
  - **User-level threads**
    - implemented through a threads library
  - **Lightweight processes**
    - can be viewed as a mapping between ULTs and kernel threads
  - **Kernel threads**
Figure 4.15  Solaris Multithreaded Architecture Example
Figure 4.16  Process Structure in Traditional UNIX and Solaris [LEWI96]
Figure 4.17 Solaris User-Level Thread and LWP States
Solaris Thread Execution

Event of some transitions
- Synchronization
  - invoke a concurrency action and go into the sleeping state
- Suspension
  - go into the stopped state
- Preemption
  - go into the runnable state
- Yielding
  - yield to another runnable thread
Linux Process

- State
- Scheduling information
- Identifiers
- Interprocess communication: SVR4 is supported
- Links
- Times and timers
- File system
- Virtual memory
- Processor-specific context
Linux States of a Process

- Running
- Interruptable
  - blocked state
- Uninterruptable
  - another blocked state, but will not accept any signals
- Stopped
  - halted, and can only resume by positive action from another process
- Zombie
Figure 4.18 Linux Process/Thread Model