Processes and Threads

Chapter 3 and 4

Operating Systems:
Internals and Design Principles, 6/E
William Stallings

Patricia Roy
Manatee Community College, Venice, FL
©2008, Prentice Hall
Process

• A program in execution
• An instance of a program running on a computer (cooking vs. recipe)
• The entity that can be assigned to and executed on a processor
• A unit of activity characterized by the execution of a sequence of instructions, a current state, and an associated set of system resources
Requirements of an Operating System

• Interleave the execution of multiple processes – multiprogramming -, to maximize processor utilization while providing reasonable response time

• Allocate resources to processes

• Support interprocess communication and user creation of processes
Example Execution

The figure illustrates the memory layout for three processes: Dispatcher, Process A, Process B, and Process C. The program counter is set to address 8000, indicating the current execution point.
Combined Trace of Processes

1  5000
2  5001
3  5002
4  5003
5  5004
6  5005

7  100
8  101
9  102
10 103
11 104
12 105
13 8000
14 8001
15 8002
16 8003

7  100
8  101
9  102
10 103
11 104
12 105
13 8000
14 8001
15 8002
16 8003

--- I/O Request

27 12004
28 12005

29 100
30 101
31 102
32 103
33 104
34 105
35 5006
36 5007
37 5008
38 5009
39 5010
40 5011

--- Timeout

41 100
42 101
43 102
44 103
45 104
46 105
47 12006
48 12007
49 12008
50 12009
51 12010
52 12011

--- Timeout
Figure 3.7  Process States for Trace of Figure 3.4
Five-State Process Model

States:
- New
- Ready
- Running
- Blocked
- Exit

Transitions:
- Admit: New to Ready
- Dispatch: Ready to Running
- Release: Running to Exit
- Timeout: Running to Ready
- Event Occurs: Exit to Blocked
- Event Wait: Blocked to Ready
- Event Occurs: Blocked to Running
- Event Wait: Running to Block
Process Creation

• New batch job
• Interactive logon
• Created by OS to provide a service
• Spawned by existing process – UNIX: fork + exec
Process Termination

• Normal completion
• Time limit exceeded
• Privileged instructions
• Parent termination
• Operator or OS intervention (e.g. deadlock)
• Errors
  – memory unavailable
  – bounds violation
  – protection error
  – arithmetic error
  – I/O failure
Multiple Blocked Queues

Admit → Ready Queue → Dispatch → Processor → Release

Event 1 Occurs → Event 1 Queue → Event 1 Wait

Event 2 Occurs → Event 2 Queue → Event 2 Wait

Event n Occurs → Event n Queue → Event n Wait
Suspended Processes

• Processor is faster than I/O so many processes could be waiting for I/O
• Swap these processes to disk to free up more memory
• Blocked state becomes suspend state when swapped to disk
• Two new states
  – Blocked/Suspend
  – Ready/Suspend
Two Suspend States

- New
- Ready/Suspend
- Ready
- Running
- Exit
- Blocked/Suspend
- Blocked

States and Transitions:
- Admit
- Activate
- Suspend
- Dispatch
- Release
- Event Occurs
- Event Wait
- Timeout
Operating System Control Structures

• Information about the current status of processes and resources
• Tables are constructed for each entity the operating system manages: memory table, I/O table, file table
Memory Tables

- Allocation of main memory to processes
- Allocation of secondary memory to processes
- Protection attributes for access to shared memory regions
- Information needed to manage virtual memory (page table)
I/O Tables

• I/O device is available or assigned
• Status of I/O operation
• Location in main memory being used as the source or destination of the I/O transfer
File Tables

- Existence of files
- Location on secondary memory
- Current Status
- Attributes
- Sometimes this information is maintained by a file management system
Figure 3.11 General Structure of Operating System Control Tables
Process Table – Process Control Block

- Identifiers: process id, parent process, user id
- User-visible registers
- Control and status registers: PC, PSW
- Stack pointer
- Scheduling and state info: process state, priority, used CPU time, event
- Process privileges
- Memory management: pointers to segments, page table
- Resource ownership and utilization
Modes of Execution

• User mode
  – Less-privileged mode
  – User programs typically execute in this mode

• System mode, control mode, or kernel mode
  – More-privileged mode
  – Kernel of the operating system

• Mode switch
Process Creation

• Assign a unique process identifier
• Allocate memory space for the process
• Initialize process control block
• Set up appropriate linkages (e.g. put process in scheduling queue)
• Create or expand other data structures (e.g. CPU time, page table)
When to Switch Process

Clock interrupt: process has executed for the maximum allowable time slice

I/O interrupt

Memory fault: memory address is in virtual memory so it must be brought into main memory – requires I/O

Trap: error or exception occurred; may cause process to be moved to Exit state

Supervisor/system call, e.g., such as file open
Change of Process State

1. Save context of processor including program counter and other registers
2. Update the process control blocks
3. Move process into appropriate queue – ready; blocked; ready/suspend
4. Run the scheduler to select another process for execution
5. Update the process control block
6. Restore context of the selected process
Execution of the Operating System

• Non-process Kernel
  – Execute kernel outside of any process
  – Operating system code is executed as a separate entity that operates in privileged mode – monolithic OS

• Execution within user processes
  – Operating system software within context of a user process, e.g. scheduler

• Process-based operating system
  • Implement the OS as a collection of system processes – modular OS
Execution of the Operating System

(a) Separate kernel

(b) OS functions execute within user processes

(c) OS functions execute as separate processes
Processes and Threads

• 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 operating system
Threads

Process = resource grouping (code, data, open files, etc.) + execution (program counter, registers, stack)

Multithreading:
• multiple execution takes place in the same process environment
• co-operation by sharing resources (address space, open files, etc.)
# The Thread Model

<table>
<thead>
<tr>
<th>Per process items</th>
<th>Per thread items</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address space</td>
<td>Program counter</td>
</tr>
<tr>
<td>Global variables</td>
<td>Registers</td>
</tr>
<tr>
<td>Open files</td>
<td>Stack</td>
</tr>
<tr>
<td>Child processes</td>
<td>State</td>
</tr>
<tr>
<td>Pending alarms</td>
<td></td>
</tr>
<tr>
<td>Signals and signal handlers</td>
<td></td>
</tr>
<tr>
<td>Accounting information</td>
<td></td>
</tr>
</tbody>
</table>

Left: Items shared by all threads in a process
Right: Items private to each thread
Threads
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
Uses of Threads in a Single-User Multiprocessing System

• Foreground and background work
• Speed of execution, e.g. blocked and running threads in one process
• Modular program structure
• Specific scheduling algorithms
A word processor with three threads
A multithreaded Web server

Diagram showing:
- Web server process
- Dispatcher thread
- Worker thread
- Web page cache
- Network connection
- User space
- Kernel space
User-Level Threads

(a) Pure user-level
User-Level Threads

• All thread management is done by the application
• The kernel is not aware of the existence of threads
• Blocking system call!
Kernel-Level Threads

- 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

• Example is Solaris
• Thread creation done in the user space
• Bulk of scheduling and synchronization of threads within application
Combined Approach
Solaris

- Process includes the user’s address space, stack, and process control block
- User-level threads
- Lightweight processes (LWP)
- Kernel threads
Processes and Threads in Solaris

Figure 4.15  Processes and Threads in Solaris [MCD07]
LWP 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
Process Structure

UNIX Process Structure

- Process ID
- User IDs
  - Signal Dispatch Table
  - Memory Map
  - Priority
  - Signal Mask
  - Registers
  - STACK
  - Processor State
  - File Descriptors

Solaris Process Structure

- Process ID
- User IDs
  - Signal Dispatch Table
  - Memory Map
  - File Descriptors

LWP 1
- LWP ID
- Priority
- Signal Mask
- Registers
- STACK

LWP 2
- LWP ID
- Priority
- Signal Mask
- Registers
- STACK
Windows Processes

• Implemented as objects
• An executable process may contain one or more threads
• Both processes and thread objects have built-in synchronization capabilities
Windows 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