Chapter 3
Process Description and Control
Contents

• Process states
• Process description
• Process control
• Unix process management
Process

- From processor’s point of view
  - Execute instruction dictated by *program counter*
  - Interleave the execution of various processes

- From individual program’s point of view
  - Executes a sequence of instructions within that program
Snapshot of Example Execution
## Traces of Processes

<table>
<thead>
<tr>
<th>5000</th>
<th>8000</th>
<th>12000</th>
</tr>
</thead>
<tbody>
<tr>
<td>5001</td>
<td>8001</td>
<td>12001</td>
</tr>
<tr>
<td>5002</td>
<td>8002</td>
<td>12002</td>
</tr>
<tr>
<td>5003</td>
<td>8003</td>
<td>12003</td>
</tr>
<tr>
<td>5004</td>
<td></td>
<td>12004</td>
</tr>
<tr>
<td>5005</td>
<td></td>
<td>12005</td>
</tr>
<tr>
<td>5006</td>
<td></td>
<td>12006</td>
</tr>
<tr>
<td>5007</td>
<td></td>
<td>12007</td>
</tr>
<tr>
<td>5008</td>
<td></td>
<td>12008</td>
</tr>
<tr>
<td>5009</td>
<td></td>
<td>12009</td>
</tr>
<tr>
<td>5010</td>
<td></td>
<td>12010</td>
</tr>
<tr>
<td>5011</td>
<td></td>
<td>12011</td>
</tr>
</tbody>
</table>

(a) Trace of Process A  
(b) Trace of Process B  
(c) Trace of Process C

5000 = Starting address of program of Process A  
8000 = Starting address of program of Process B  
12000 = Starting address of program of Process C
# Combined Trace of Processes

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5000</td>
</tr>
<tr>
<td>2</td>
<td>5001</td>
</tr>
<tr>
<td>3</td>
<td>5002</td>
</tr>
<tr>
<td>4</td>
<td>5003</td>
</tr>
<tr>
<td>5</td>
<td>5004</td>
</tr>
<tr>
<td>6</td>
<td>5005</td>
</tr>
</tbody>
</table>
|   |      | I/O request
| 7 | 100  |
| 8 | 101  |
| 9 | 102  |
|10 | 103  |
|11 | 104  |
|12 | 105  |
|13 | 8000 |
|14 | 8001 |
|15 | 8002 |
|16 | 8003 |

|   |      | Time out
| 27| 12004|
| 28| 12005|
|   |      | Time out
| 29| 100  |
| 30| 101  |
| 31| 102  |
| 32| 103  |
| 33| 104  |
| 34| 105  |
| 35| 5006 |
| 36| 5007 |
| 37| 5008 |
| 38| 5009 |
| 39| 5010 |
| 40| 5011 |

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

|   |      | Time out
Two-State Process Model

- Process may be in one of two states
  - Running
  - Not-running

(a) State transition diagram
Not-Running Process in a Queue

(b) Queuing diagram
Dispatcher

- A program that moves the processor from one process to another
- Selects a process from the queue to execute after interrupt or process termination
- Prevents a single process from monopolizing the processor time
Process Creation

- Submission of a batch job
- User logs on
- Created by OS to provide a service
  - A process to control printing
  - A process to control network connection
- Spawned by an existing process
Process Spawning

- A process is created by OS at the explicit request of another process
  - `fork()`
- Parent process, child process
- Related processes need to communicate and cooperate with each other
Process Termination

- Batch job issues *Halt* instruction
- User logs off
- Quit an application
  - e.g., word processing
- Error and fault conditions
Reasons for Process Termination

- Normal completion
- Time limit exceeded
- Memory unavailable
- Bounds violation
- Protection error
  - Example: write to read-only file
- Arithmetic error
- Time overrun
  - Process waited longer than a specified maximum for an event
Reasons for Process Termination

● I/O failure
● Invalid instruction
  ✷ Happens when try to execute data
● Privileged instruction
● Data misuse
● Operating system intervention
  ✷ Such as when deadlock occurs
● Parent terminates so child processes terminate
● Parent request
A Five-State Model

- Inadequacy of two-state model
  - Some processes in Not-running state are ready to execute, whereas others are blocked
  - Dispatcher could not just select the process at the oldest end of the queue
  - Dispatcher would have to scan the list looking for the processes
  - Need to split the Not-running state into two states
    - Ready state and Blocked state
A Five-State Model

- Running
- Ready
- Blocked
- New
  - A process has just been created but has not yet been admitted to main memory
- Exit
  - A process has been released from the pool of executable processes by OS
Five-State Process Model
Process States for Trace

Process A

Process B

Process C

Dispatcher

= Running

= Ready

= Blocked
Using Blocked Queues
Suspended Processes

- Need for swapping
  - Processor is faster than I/O, so all processes could be waiting for I/O
  - Thus, even with multiprogramming, a processor could be idle most of the time
- Solution
  - Main memory could be expanded, and so be able to accommodate more processes
  - Swapping
Suspended Processes

● Swapping
  • Moving part or all of a process from main memory to disk
    – Swap in and swap out
  • Blocked state becomes suspend state when swapped to disk
  • Suspended queue: a queue of existing processes that have been temporarily kicked out of main memory, or suspended
One Suspend State

(a) With One Suspend State
Suspended Processes

- Problem of one suspended state
  - Swapped out processes could be ready in the meantime
  - Two new states are needed
    - Blocked, suspend
    - Ready, suspend
Two Suspend States
# Reasons for Process Suspension

<table>
<thead>
<tr>
<th>Reason</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Swapping</td>
<td>The operating system needs to release sufficient main memory to bring in a process that is ready to execute.</td>
</tr>
<tr>
<td>Other OS reason</td>
<td>The operating system may suspend a background or utility process or a process that is suspected of causing a problem.</td>
</tr>
<tr>
<td>Interactive user request</td>
<td>A user may wish to suspend execution of a program for purposes of debugging or in connection with the use of a resource.</td>
</tr>
<tr>
<td>Timing</td>
<td>A process may be executed periodically (e.g., an accounting or system monitoring process) and may be suspended while waiting for the next time interval.</td>
</tr>
<tr>
<td>Parent process request</td>
<td>A parent process may wish to suspend execution of a descendent to examine or modify the suspended process, or to coordinate the activity of various descendents.</td>
</tr>
</tbody>
</table>
What is the Role of OS?

- Controller of events within the computer
- Schedules and dispatches processes for execution by the processor
- Allocates resources to processes
- Responds to requests by user programs
- Entity that manages the use of system resources by processes
OS Control Structures

- Tables are constructed for each entity the operating system manages
  - Process tables
  - Memory tables
  - I/O tables
  - File tables
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
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
Process Table

- Where the process attributes are stored
  - Process ID, parent process ID
  - Process state
  - Execution time so far
  - Location in memory
  - ........
**Process Image**

- **User Data**
  - Modifiable user space (user data, user stack)

- **User Program**
  - The program to be executed

- **System Stack**
  - Store parameters of system calls

- **Process Control Block**
  - Data needed by OS to control the process
Process Control Block

- Process identification
- Processor state information
- Process control information
Process Control Block

● Process Identification
  ✷ Process identifier
    – Unique numeric identifier
    – May be an index into the primary process table
  ✷ User identifier
    – Who is responsible for the job
    – Real-user id, real-group id
    – Effective-user id, effective-group id
Process Control Block

- Processor State Information
  - User-Visible Registers
    - A user-visible register is one that may be referenced by means of the machine language that the processor executes. Typically, there are from 8 to 32 of these registers, although some RISC implementations have over 100.
Process Control Block

- Processor State Information
  - Control and Status Registers
    - These are processor registers that are employed to control the operation of the processor
    - Program counter: Contains the address of the next instruction to be fetched
    - Condition codes: Result of the most recent arithmetic or logical operation (e.g., sign, zero, carry, equal, overflow)
    - Status information: Includes interrupt enabled/disabled flags, execution mode
Process Control Block

- Processor State Information
  - Stack Pointers
    - Each process has one or more last-in-first-out (LIFO) system stacks associated with it. A stack is used to store parameters and calling addresses for procedure and system calls. The stack pointer points to the top of the stack.
# Pentium II EFLAGS Register

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>31</td>
<td>ID</td>
</tr>
<tr>
<td>30-29</td>
<td>VIP, VIP</td>
</tr>
<tr>
<td>28-27</td>
<td>AV, CM, RF</td>
</tr>
<tr>
<td>26-22</td>
<td>NT, IO, OF, ODF, OFF, IFF</td>
</tr>
<tr>
<td>21-16</td>
<td>TS, ZF, AF, PF, CF</td>
</tr>
<tr>
<td>15</td>
<td></td>
</tr>
</tbody>
</table>

- **ID** = Identification flag
- **VIP** = Virtual interrupt pending
- **VIF** = Virtual interrupt flag
- **AC** = Alignment check
- **VM** = Virtual 8086 mode
- **RF** = Resume flag
- **NT** = Nested task flag
- **IOPL** = I/O privilege level
- **OF** = Overflow flag
- **DF** = Direction flag
- **IF** = Interrupt enable flag
- **TF** = Trap flag
- **SF** = Sign flag
- **ZF** = Zero flag
- **AF** = Auxiliary carry flag
- **PF** = Parity flag
- **CF** = Carry flag
Process Control Block

- Process Control Information
  - Scheduling and state information
  - Data structuring
  - Interprocess communication
  - Process privileges
  - Memory management
  - Resource ownership and utilization
User Processes in Virtual Memory

Process Identification

Processor State Information

Process Control Information

User Stack

Private User Address Space (Programs, Data)

Shared Address Space

Process 1

Process Identification

Processor State Information

Process Control Information

User Stack

Private User Address Space (Programs, Data)

Shared Address Space

Process 2

Process Identification

Processor State Information

Process Control Information

User Stack

Private User Address Space (Programs, Data)

Shared Address Space

Process n

Process Control Block
Process List Structures
Process Control Block

- Scheduling and State Information

This is the formation that is needed by the operating system to perform its scheduling functions. Typical items of information:

- Process state: defines the readiness of the process to be scheduled or execution (e.g., running, ready, waiting, halted).

- Priority: One or more fields may be used to describe the scheduling priority of the process. In some systems, several values are required (e.g., default, current, highest, lowest, etc.).

- Scheduling-related information: This will depend on the scheduling algorithm used. Examples are the amount of time that a process has been waiting and the amount of time that the process executed the last time it was running.

- Event: Identity of event the process is awaiting before it can be resumed.
Process Control Block

- Data Structuring

  - A process may be linked to other process in a queue, ring, or some other structure. For example, all processes in a waiting state for a particular priority level may be linked in a queue. A process may exhibit a parent-child (creator-created) relationship with another process. The process control block may contain pointers to other processes to support these structures.
Process Control Block

- Interprocess Communication
  - Various flags, signals, and messages may be associated with communication between two independent processes. Some or all of this information may be maintained in the process control block.

- Process Privileges
  - Processes are granted privileges in terms of the memory that may be accessed and the types of instructions that may be executed. In addition, privileges may apply to the use of system utilities and services.
Process Control Block

- Memory Management
  - This section may include pointers to segment and/or page tables that describe the virtual memory assigned to this process.

- Resource Ownership and Utilization
  - Resources controlled by the process may be indicated, such as opened files. A history of utilization of the processor or other resources may also be included; this information may be needed by the scheduler.
Modes of Execution

- **User Mode**
  - Less privileged mode
  - User program typically execute in this mode

- **Kernel Mode**
  - More privileged mode
  - Has complete control of the processor and all its instructions, registers, and memory
  - Not desirable for user programs
Typical Functions of an OS Kernel

- Process Management
- Memory Management
- I/O Management
- Support Functions
Typical Functions of an OS Kernel

- Process Management
  - Process creation and termination
  - Process scheduling and dispatching
  - Process switching
  - Process synchronization and support for inter-process communication
  - Management of process control blocks
Typical Functions of an OS Kernel

- Memory Management
  - Allocation of address space to processes
  - Swapping
  - Page and segment management
Typical Functions of an OS Kernel

- I/O Management
  - Buffer management
  - Allocation of I/O channels and devices to processes

- Support Functions
  - Interrupt handling
  - Accounting
  - Monitoring
Process Creation

- Assign a unique process identifier
- Allocate space for the process
- Initialize process control block
- Set up appropriate linkages
  - Ex: add new process to linked list used for scheduling queue
- Other
  - Maintain an accounting file
When to Switch a Process

- Interrupts
  - 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
When to Switch a Process

• Trap
  • Error occurred during program execution
    – Division by zero
  • May cause process to be moved to Exit state

• Supervisor call
  • System call
    – Such as file open
Change of Process State

- Save context of processor including program counter and other registers
- Update the process control block with the new state and any accounting information
- Move process control block to appropriate queue - ready, blocked
- Select another process for execution
Change of Process State

● Update the process control block of the process selected
● Update memory-management data structures
● Restore context of the selected process
Execution of the Operating System

● Nonprocess Kernel
  - Execute kernel outside of any process
  - Operating system code is executed as a separate entity that operates in privileged mode

● Execution Within User Processes
  - Operating system software within the context of a user process
    - A process switch is not performed, just a mode switch within the same process
  - Process executes in privileged mode when executing operating system code
(a) Separate kernel
(b) OS functions execute within user processes
Process Image: OS executes within User Space
Execution of the Operating System

- Process-Based Operating System
  - major kernel functions are separate user processes
    - modular design and clean interfaces
  - useful in multi-processor or multi-computer environment
    - naturally implements client-server computing
(c) OS functions execute as separate processes
# UNIX Process States

<table>
<thead>
<tr>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Running</td>
<td>Executing in user mode.</td>
</tr>
<tr>
<td>Kernel Running</td>
<td>Executing in kernel mode.</td>
</tr>
<tr>
<td>Ready to Run, in Memory</td>
<td>Ready to run as soon as the kernel schedules it.</td>
</tr>
<tr>
<td>Asleep in Memory</td>
<td>Unable to execute until an event occurs; process is in main memory (a blocked state).</td>
</tr>
<tr>
<td>Ready to Run, Swapped</td>
<td>Process is ready to run, but the swapper must swap the process into main memory before the kernel can schedule it to execute.</td>
</tr>
<tr>
<td>Sleeping, Swapped</td>
<td>The process is awaiting an event and has been swapped to secondary storage (a blocked state).</td>
</tr>
<tr>
<td>Preempted</td>
<td>Process is returning from kernel to user mode, but the kernel preempts it and does a process switch to schedule another process.</td>
</tr>
<tr>
<td>Created</td>
<td>Process is newly created and not yet ready to run.</td>
</tr>
<tr>
<td>Zombie</td>
<td>Process no longer exists, but it leaves a record for its parent process to collect.</td>
</tr>
</tbody>
</table>
Figure 3.16  UNIX Process State Transition Diagram
<table>
<thead>
<tr>
<th>User-Level Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process Text</td>
</tr>
<tr>
<td>Process Data</td>
</tr>
<tr>
<td>User Stack</td>
</tr>
<tr>
<td>Shared Memory</td>
</tr>
</tbody>
</table>
## Register Context

<table>
<thead>
<tr>
<th>Register Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program Counter</td>
<td>Address of next instruction to be executed; may be in kernel or user memory space of this process</td>
</tr>
<tr>
<td>Processor Status Register</td>
<td>Contains the hardware status at the time of preemption; contents and format are hardware dependent</td>
</tr>
<tr>
<td>Stack Pointer</td>
<td>Points to the top of the kernel or user stack, depending on the mode of operation at the time or preemption</td>
</tr>
<tr>
<td>General-Purpose Registers</td>
<td>Hardware dependent</td>
</tr>
<tr>
<td><strong>System-Level Context</strong></td>
<td></td>
</tr>
<tr>
<td>--------------------------</td>
<td></td>
</tr>
<tr>
<td><strong>Process Table Entry</strong></td>
<td>Defines state of a process; this information is always accessible to the operating system</td>
</tr>
<tr>
<td><strong>U (user) Area</strong></td>
<td>Process control information that needs to be accessed only in the context of the process</td>
</tr>
<tr>
<td><strong>Per Process Region Table</strong></td>
<td>Defines the mapping from virtual to physical addresses; also contains a permission field that indicates the type of access allowed the process: read-only, read-write, or read-execute</td>
</tr>
<tr>
<td><strong>Kernel Stack</strong></td>
<td>Contains the stack frame of kernel procedures as the process executes in kernel mode</td>
</tr>
<tr>
<td>Table 3.11 UNIX Process Table Entry</td>
<td></td>
</tr>
<tr>
<td>------------------------------------</td>
<td></td>
</tr>
<tr>
<td><strong>Process Status</strong></td>
<td>Current state of process.</td>
</tr>
<tr>
<td><strong>Pointers</strong></td>
<td>To U area and process memory area (text, data, stack).</td>
</tr>
<tr>
<td><strong>Process Size</strong></td>
<td>Enables the operating system to know how much space to allocate the process.</td>
</tr>
<tr>
<td><strong>User Identifiers</strong></td>
<td>The <strong>real user ID</strong> identifies the user who is responsible for the running process. The <strong>effective user ID</strong> may be used by a process to gain temporary privileges associated with a particular program; while that program is being executed as part of the process, the process operates with the effective user ID.</td>
</tr>
<tr>
<td><strong>Process Identifiers</strong></td>
<td>ID of this process; ID of parent process. These are set up when the process enters the Created state during the fork system call.</td>
</tr>
<tr>
<td>Description</td>
<td>Details</td>
</tr>
<tr>
<td>---------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Event Descriptor</td>
<td>Valid when a process is in a sleeping state; when the event occurs, the process is transferred to a ready-to-run state.</td>
</tr>
<tr>
<td>Priority</td>
<td>Used for process scheduling.</td>
</tr>
<tr>
<td>Signal</td>
<td>Enumerates signals sent to a process but not yet handled.</td>
</tr>
<tr>
<td>Timers</td>
<td>Include process execution time, kernel resource utilization, and user-set timer used to send alarm signal to a process.</td>
</tr>
<tr>
<td>P_link</td>
<td>Pointer to the next link in the ready queue (valid if process is ready to execute).</td>
</tr>
<tr>
<td>Memory Status</td>
<td>Indicates whether process image is in main memory or swapped out. If it is in memory, this field also indicates whether it may be swapped out or is temporarily locked into main memory.</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>----------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Process Table Pointer</td>
<td>Indicates entry that corresponds to the U area.</td>
</tr>
<tr>
<td>User Identifiers</td>
<td>Real and effective user IDs. Used to determine user privileges.</td>
</tr>
<tr>
<td>Timers</td>
<td>Record time that the process (and its descendants) spent executing in user mode and in kernel mode.</td>
</tr>
<tr>
<td>Signal-Handler Array</td>
<td>For each type of signal defined in the system, indicates how the process will react to receipt of that signal (exit, ignore, execute specified user function).</td>
</tr>
<tr>
<td>Control Terminal</td>
<td>Indicates login terminal for this process, if one exists.</td>
</tr>
<tr>
<td>Error Field</td>
<td>Records errors encountered during a system call.</td>
</tr>
<tr>
<td>Return Value</td>
<td>Contains the result of system calls.</td>
</tr>
<tr>
<td>Category</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>I/O Parameters</td>
<td>Describe the amount of data to transfer, the address of the source (or target) data array in user space, and file offsets for I/O.</td>
</tr>
<tr>
<td>File Parameters</td>
<td>Current directory and current root describe the file system environment of the process.</td>
</tr>
<tr>
<td>User File Descriptor Table</td>
<td>Records the files the process has open.</td>
</tr>
<tr>
<td>Limit Fields</td>
<td>Restrict the size of the process and the size of a file it can write.</td>
</tr>
<tr>
<td>Permission Modes Fields</td>
<td>Mask mode settings on files the process creates.</td>
</tr>
</tbody>
</table>