Process Description and Control

Chapter 3
Process Related Requirements for an Operating System

• Interleave the execution of multiple processes to maximize processor utilization while providing reasonable response time
• Allocate resources to processes
• Support interprocess communication and user creation of processes
“Operational” Concepts

• Computer platform consists of a collection of hardware resources
• Computer applications are developed to perform some task
• *Inefficient for applications to be written directly for a given hardware platform*

• OS provides a convenient to use, feature rich, secure, and consistent *interface* for applications to use
• OS provides a uniform, *abstract representation* of resources that can be requested and accessed by application
OS Manages Execution of Applications

- Resources made available to multiple applications
- Processor is switched among multiple applications
- The processor and I/O devices can be used efficiently
Views of a “Process”

• A program in execution
• An instance of a program running on a computer
• 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 instructions
Process Elements

- Identifier – number
- State – run / blocked / eady
- Priority – high / low
- Program counter – next statement to execute
- Memory pointers
- Context data – registers, stack
- I/O status information
- Accounting information
Process Control Block

The OS data Structure defining a process

- Contains the process elements
  - id, state, priority, PC

- Created and manage by the operating system
- Allows support for multiple processes
  - One PCB / PD for each process
# Process Control Block

<table>
<thead>
<tr>
<th>Identifier</th>
<th>State</th>
<th>Priority</th>
<th>Program counter</th>
<th>Memory pointers</th>
<th>Context data</th>
<th>I/O status information</th>
<th>Accounting information</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Symbols: ¥ ¥ ¥ ¥
Example Execution
### Trace of Processes

<table>
<thead>
<tr>
<th>(a) Trace of Process A</th>
<th>(b) Trace of Process B</th>
<th>(c) Trace of Process C</th>
</tr>
</thead>
<tbody>
<tr>
<td>5000</td>
<td>8000</td>
<td>12000</td>
</tr>
<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>

5000 = Starting address of program of Process A  
8000 = Starting address of program of Process B  
12000 = Starting address of program of Process C
Processes Traces w/ Interleaving

Dispatcher
100 - 105

A

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

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

------------------- I/O request
17 100
18 101
19 102
20 103
21 104
22 105
23 12000
24 12001
25 12002
26 12003

B

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

------------------- I/O request
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

- In an FSM each state has a **unique** meaning
- Process may be in one of two states
  - Running
  - Not-running
Not-Running Processes in a Queue (Ready Queue)

Ready
But not running

Run

(b) Queuing diagram
## Reasons for Process Creation

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>New batch job</td>
<td>The operating system is provided with a batch job control stream, usually on tape or disk. When the operating system is prepared to take on new work, it will read the next sequence of job control commands.</td>
</tr>
<tr>
<td>Interactive logon</td>
<td>A user at a terminal logs on to the system.</td>
</tr>
<tr>
<td>Created by OS to provide a service</td>
<td>The operating system can create a process to perform a function on behalf of a user program, without the user having to wait (e.g., a process to control printing).</td>
</tr>
<tr>
<td>Spawned by existing process</td>
<td>For purposes of modularity or to exploit parallelism, a user program can dictate the creation of a number of processes.</td>
</tr>
</tbody>
</table>
# Reasons for Process Termination

<table>
<thead>
<tr>
<th>Reason</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal completion</td>
<td>The process executes an OS service call to indicate that it has completed running.</td>
</tr>
<tr>
<td>Time limit exceeded</td>
<td>The process has run longer than the specified total time limit. There are a number of possibilities for the type of time that is measured. These include total elapsed time (&quot;wall clock time&quot;), amount of time spent executing, and, in the case of an interactive process, the amount of time since the user last provided any input.</td>
</tr>
<tr>
<td>Memory unavailable</td>
<td>The process requires more memory than the system can provide.</td>
</tr>
<tr>
<td>Bounds violation</td>
<td>The process tries to access a memory location that it is not allowed to access.</td>
</tr>
<tr>
<td>Protection error</td>
<td>The process attempts to use a resource such as a file that it is not allowed to use, or it tries to use it in an improper fashion, such as writing to a read-only file.</td>
</tr>
<tr>
<td>Arithmetic error</td>
<td>The process tries a prohibited computation, such as division by zero, or tries to store numbers larger than the hardware can accommodate.</td>
</tr>
</tbody>
</table>
2 State Process Model Insufficient

- **Not-running**
  - ready to execute

- **Blocked**
  - waiting for I/O

Non-executing process can be in either state

A single queue for both is not sufficient

- Dispatcher cannot just select the process that has been in the queue the longest because it may be blocked
A Five-State Model

• Running
  – Memory + Processor
• Ready
  – Memory, *not* Blocked, not in Processor
• Blocked
  – Memory + Blocked
• New
  – Job arrival, PCB(?), *no memory allocated*
• Exit
Five-State Process Model
Process States

- Process A
- Process B
- Process C
- Dispatcher

- Running
- Ready
- Blocked
5-State Process Model
(An Implementation Perspective)
Multiple Blocked Queues

More Efficient

Check queue associated with event occurrence
Suspended Processes

• Processor is faster than I/O so **all processes** could be waiting for I/O

• Swap one or more processes to disk to *free up more memory*
  – Swap out process in Blocked or Ready
    • Memory taken away
  – Bring in a **NEW** process
## Reasons for Process Suspension

<table>
<thead>
<tr>
<th>Reason</th>
<th>Explanation</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>
Suspended Processes

• Blocked state becomes suspend state when swapped to disk
• Can suspend process from either the Block or Ready state

• Two new states
  – Blocked/Suspend
  – Ready/Suspend
Modeling Process Suspension

Suspend ONLY Blocked Processes
Modeling Process Suspension

Suspend **Ready** or **Blocked** Processes
Processes and Resources

- Processes P1 and P2 are in Memory
- Process P2 is blocked waiting for I/O resource held by P1
- Process Pn is awaiting memory allocation
  - Suspended or New Job Arrival
Operating System
Control Structures

Contain information about the current status of each process and resource

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

Tables $\equiv$ (linked) Data structures in the OS
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

Tables $\equiv$ (linked) Data structures in the OS
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

Tables ≡ (linked) Data structures in the OS
File Tables

- Existence of files
- Location on secondary memory
- Current Status
- Attributes
- Sometimes this information is maintained by a file management system

Tables $\equiv$ (linked) Data structures in the OS
Process Table

- Where process is located
- Attributes in the process control block
  - Program
  - Data
  - Stack

Tables $\equiv$ (linked) Data structures in the OS
<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>User Data</strong></td>
<td>The modifiable part of the user space. May include program data,</td>
</tr>
<tr>
<td></td>
<td>a user stack area, and programs that may be modified.</td>
</tr>
<tr>
<td><strong>User Program</strong></td>
<td>The program to be executed.</td>
</tr>
<tr>
<td><strong>System Stack</strong></td>
<td>Each process has one or more last-in-first-out (LIFO) system</td>
</tr>
<tr>
<td></td>
<td>stacks associated with it. A stack is used to store parameters</td>
</tr>
<tr>
<td></td>
<td>and calling addresses for procedure and system calls.</td>
</tr>
<tr>
<td><strong>Process Control Block</strong></td>
<td>Data needed by the operating system to control the process (see</td>
</tr>
</tbody>
</table>
Control Tables, Processes and Process Images

Note:
LINKAGES pervade OS Control Structure
Process Control Block: Categories of Information

• Process Identification
  – Process Id…..

• Processor State Information
  – Registers, Stack pointers…

• Process Control Information
  – State Information, Resource ownership…
Process Control Block

• Process identification
  – Identifiers
    • Numeric identifiers that may be stored with the process control block include
      – Identifier of this process
      – Identifier of the process that created this process (parent process)
      – User identifier
Process Control Block

• Processor State Information
  – User-Visible Registers
  – Control and Status Registers
    • Program counter, condition codes, status information
  – Stack Pointers
    • Each process has an associated system/runtime stack
  – Program Status Word (PSW)
    • Example: the EFLAGS register on Pentium machines
Process Control Block

• Process Control Information
  – Scheduling and State Information
    • Process state (ready, running…)
    • Priority
    • Scheduling info (time used, waiting…)
  – Granted Privileges
    • Shared memory, system utility access
  – VM Page Map Tables
  – Resource Ownership and Utilization
  – Data Structuring
    • Data structures indicating relationships
      – parent/child, threads, shared resources
  – IPC Information
When to Switch a Process
(Context Switch)

• 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
(Context Switch)

• Trap
  – error or exception occurred
  – may cause process to be moved to Exit state

• Supervisor call
  – such as file open
Change of Process State: Performing the Context Switch

• Save context of processor including program counter and other registers
• Update the process control block of the process that is currently in the Running state
• Move process control block to appropriate queue – ready; blocked; ready/suspend
• Select another process for execution
Change of Process State: Performing the Context Switch

- Update the process control block of the process selected
- Update memory-management data structures
- Restore context of the selected process
OS Design

OS can be integrated into the execution framework in 3 distinct ways

• Executing as a Non-Process Kernel
• Execution within User Processes
• Process-Based execution
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
Execution of the Operating System

• Execution Within User Processes
  – Operating system software within context of a user process
  – Process executes in privileged mode when executing operating system code

OS is a collection of routines LINKED to user processes

Minimal CTX time!
Execution of the Operating System

• Process-Based Operating System
  – Implement operating system as a collection of system processes
  – Useful in multi-processor or multi-computer environment
UNIX SVR4 Process Management

- Most of the operating system executes within the environment of a user process
## UNIX Process States

Table 3.9 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 3.17 UNIX Process State Transition Diagram
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
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
- Create or expand other data structures
  - Ex: maintain an accounting file