Process Description and Control

Chapter 3
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
Figure 3.1 Snapshot of Example Execution (Figure 3.3) at Instruction Cycle 13
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></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>

(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

**Figure 3.2** Traces of Processes of Figure 3.1
<table>
<thead>
<tr>
<th></th>
<th>5000</th>
<th></th>
<th>27</th>
<th>12004</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>5001</td>
<td></td>
<td>28</td>
<td>12005</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>5002</td>
<td></td>
<td>29</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5003</td>
<td></td>
<td>30</td>
<td>101</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5004</td>
<td></td>
<td>31</td>
<td>102</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>5005</td>
<td></td>
<td>32</td>
<td>103</td>
<td></td>
</tr>
<tr>
<td></td>
<td>100</td>
<td></td>
<td>33</td>
<td>104</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>101</td>
<td></td>
<td>34</td>
<td>105</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>102</td>
<td></td>
<td>35</td>
<td>5006</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>103</td>
<td></td>
<td>36</td>
<td>5007</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>104</td>
<td></td>
<td>37</td>
<td>5008</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>105</td>
<td></td>
<td>38</td>
<td>5009</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>8000</td>
<td></td>
<td>39</td>
<td>5010</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>8001</td>
<td></td>
<td>40</td>
<td>5011</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>8002</td>
<td></td>
<td>41</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>8003</td>
<td></td>
<td>42</td>
<td>101</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>I/O</td>
<td></td>
<td>43</td>
<td>102</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>100</td>
<td></td>
<td>44</td>
<td>103</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>101</td>
<td></td>
<td>45</td>
<td>104</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>102</td>
<td></td>
<td>46</td>
<td>105</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>103</td>
<td></td>
<td>47</td>
<td>12006</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>104</td>
<td></td>
<td>48</td>
<td>12007</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>105</td>
<td></td>
<td>49</td>
<td>12008</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>12000</td>
<td></td>
<td>50</td>
<td>12009</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>12001</td>
<td></td>
<td>51</td>
<td>12010</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>12002</td>
<td></td>
<td>52</td>
<td>12011</td>
<td></td>
</tr>
</tbody>
</table>

100 = Starting address of dispatcher program

shaded areas indicate execution of dispatcher process;
first and third columns count instruction cycles;
second and fourth columns show address of instruction being executed

**Figure 3.3 Combined Trace of Processes of Figure 3.1**
Two-State Process Model

Process may be in one of two states
- Running
- Not-running
Not-Running Process in a Queue
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
Figure 3.5  Five-State Process Model
Figure 3.6  Process States for Trace of Figure 3.3
Using Blocked Queues

(a) Single blocked queue
(b) Multiple blocked queues
Suspended Processes

The 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
Suspended Processes

- Problem of one suspended state
  - swapped out processes could be ready in the mean time
- two new states are needed
  - Blocked, suspend
  - Ready, suspend
# 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
Operating System 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 a variety of processor registers that are employed to control the operation of the processor. These include:

- 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

Figure 3.11 Pentium II EFLAGS Register
Process Control Block

- Process Control Information
  - scheduling and state information
  - data structuring
  - interprocess communication
  - process privileges
  - memory management
  - resource ownership and utilization
Scheduling and State Information

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

- Process state: defines the readiness of the process to be scheduled for 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-allowable)
- Scheduling-related information: This will depend on the scheduling algorithm used. Examples are the amount of time that the 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 processes 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 Operating System Kernel

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

- Memory Management
  - Allocation of address space to processes
  - Swapping
  - Page and segment management
Typical Functions of an Operating-System 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
Figure 3.15 Process Image: Operating System 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
## 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