Announcement

• Exercise #2 will be out today

• Due date is next Monday
Major OS Developments
Evolution of Operating Systems

- Generations include:
  - Serial Processing
  - Simple Batch Systems
  - Multiprogrammed Batch Systems
  - Time Sharing Systems
Three major lines of computer system development created problems in **timing** and **synchronization** that contributed to the development:

- **Multiprogramming batch operation**
  - Processor is switched among the various programs residing in main memory

- **Time sharing**
  - Be responsive to the individual user but be able to support many users simultaneously

- **Real-time transaction systems**
  - A number of users are entering queries or updates against a database
Major Advances

- Operating Systems are among the most complex pieces of software ever developed

Major advances in development include:

- Processes
- Memory management
- Scheduling and resource management
- System structure
- Information protection and security
Causes of Errors

- **Improper synchronization**
  - a program must wait until the data are available in a buffer
  - improper design of the signaling mechanism can result in loss or duplication

- **Failed mutual exclusion**
  - more than one user or program attempts to make use of a shared resource at the same time
  - only one routine at a time allowed to perform an update against the file

- **Nondeterminate program operation**
  - program execution is interleaved by the processor when memory is shared
  - the order in which programs are scheduled may affect their outcome

- **Deadlocks**
  - it is possible for two or more programs to be hung up waiting for each other
  - may depend on the chance timing of resource allocation and release
Components of a Process

- A process contains three components:
  - an executable program
  - the associated data needed by the program (variables, work space, buffers, etc.)
  - the execution context (or “process state”) of the program

- The execution context is essential:
  - it is the internal data by which the OS is able to control the process
  - includes the contents of the various process registers
  - includes information such as the priority of the process and whether the process is waiting for the completion of a particular I/O event
Chapter 3

Process, Description and Control
A computer platform consists of a collection of hardware resources.

Computer applications are developed to perform some task.

It is inefficient for applications to be written directly for a given hardware platform.

The OS was developed to provide a convenient, feature-rich, secure, and consistent interface for applications to use.

We can think of the OS as providing a uniform, abstract representation of resources that can be requested and accessed by applications.
Chapter 3: Process, Description and Control

Chapter 4: Threads, SMPs and Microkernels

Chapter 9: Processor Scheduling
Process Description and Control

Chapter 3
Outline of Chapter 3

Process creation, termination, state of processes

Process image, Process control block

User mode vs Privileged mode

Different OS design (self study)
Process

- Fundamental to the structure of operating systems
- Many definitions have been given including

A *process* can be defined as:

- a program in execution
- an instance of a running program
- the entity that can be assigned to, and executed on, a processor
- a unit of activity characterized by
  - a single *sequential thread* of execution,
  - a current state, and
  - an associated set of system resources
All multiprogramming systems are built based on the concepts of processes. Main issues about processes:

- different states of processes and transitions among them.
- process description (data structures maintained)
- process control (how process switches between processes)
Requirements of 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
Two essential elements of a process are:

**Program code**
which may be shared with other processes that are executing the same program

**A set of data associated with that code**

When the processor begins to execute the program code, we refer to this executing entity as a **process**
Process: program in execution

• If we have a single program running in the system, then the task of OS is easy:
  – load the program, start it and program runs in CPU
  – (from time to time it calls OS to get some service done)

• But if we want to start several processes, then the running program in CPU (current process) has to be stopped for a while and other program (process) has to run in CPU.
  – Process management becomes an important issue

  – To do process switch, we have to save the state/context (register values) of the CPU which belongs to the stopped program, so that later the stopped program can be re-started again as if nothing has happened.
Process: program in execution

(cpu)

Main
Memory
(RAM)

CPU state
of the process
(CPU context)

process address space
(currently used portion of the address space must be in memory)
While the program is executing, this process can be uniquely characterized by a number of elements, including:

- identifier
- state
- priority
- program counter
- accounting information
- I/O status information
- context data
- memory pointers
Process Control Block

- Contains the process elements
- Created and managed by the operating system
- It is possible to interrupt a running process and later resume execution as if the interruption had not occurred
- Key tool that allows support for multiple processes

**Figure 3.1 Simplified Process Control Block**

<table>
<thead>
<tr>
<th>Identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>State</td>
</tr>
<tr>
<td>Priority</td>
</tr>
<tr>
<td>Program counter</td>
</tr>
<tr>
<td>Memory pointers</td>
</tr>
<tr>
<td>Context data</td>
</tr>
<tr>
<td>I/O status information</td>
</tr>
<tr>
<td>Accounting information</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
Information associated with each process

- Process state (ready, running, waiting, etc)
- Program counter (PC)
- CPU registers
- CPU scheduling information
  - Priority of the process, etc.
- Memory-management information
  - text/data/stack section pointers, sizes, etc.
  - pointer to page table, etc.
- Accounting information
  - CPU usage, clock time so far, …
- I/O status information
  - List of I/O devices allocated to the process, a list of open files, etc.
Process Control Block (PCB)

Process management
- Registers
- Program Counter (PC)
- Program status word (PSW)
- Stack pointer
- Process state
- Priority
- Scheduling parameters
- Process ID
- Parent Process
- Time when process started
- CPU time used
- Children’s CPU time

Memory management
- Pointer to text segment info
- Pointer to data segment info
- Pointer to stack segment info

File management
- Root directory
- Working directory
- File descriptors
- User ID
- Group ID

......more

a PCB of a process may contain this information
Kernel mains a PCB for each process. They can be linked together in various queues.
CPU Switch from Process to Process

- **Process $P_0$**
  - Executing
  - Interrupt or system call
    - Save state into PCB$_0$
      - ...
      - ...
    - Reload state from PCB$_1$
- **Operating System**
  - Idle
- **Process $P_1$**
  - Executing
  - Interrupt or system call
    - Save state into PCB$_1$
      - ...
      - ...
    - Reload state from PCB$_0$
the behavior of an individual process by listing the sequence of instructions that execute for that process

the behavior of the processor can be characterized by showing how the traces of the various processes are interleaved

small program that switches the processor from one process to another
Example Process Execution

![Diagram showing process memory allocation with addresses 0, 100, 5000, 8000, and 12000. Main Memory is divided into sections for Dispatcher, Process A, Process B, and Process C. The Program Counter is at address 8000.]
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

Figure 3.3
**Combined Trace of Processes**

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>5001</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>5002</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>5003</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5004</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>5005</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>101</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>102</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>103</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>104</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>105</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>8000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>8001</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>8002</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>8003</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>27</td>
<td>12004</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>12005</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>29</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>101</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>102</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>103</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>104</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>105</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>5006</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>5007</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>5008</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>5009</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>5010</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>5011</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>41</td>
<td>100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>101</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>102</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>103</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>104</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>105</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>47</td>
<td>12006</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>12007</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>12008</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>12009</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>51</td>
<td>12010</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>12011</td>
<td></td>
<td></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 3.4**
Process in Memory

A process needs this memory content to run (called address space; memory image)

- **Text segment** (code segment) (instructions are here)
- **Data segment** (includes global variables, arrays, etc., you use)
- **Stack segment** (holds the called function parameters, local variables, return values)
- **Heap** (storage for dynamically created variables)
Process Address Space

• A process can only access its address space
• Each process has its own address space
• Kernel can access everything
A process may be in one of two states:
- Running
- Not-running

Transitions between these states is performed by **dispatcher**.
As a process executes, it changes state

- **New**: The process is being created
- **Running**: Instructions are being executed
- **Exit**: The process has finished execution
Queuing Diagram

(b) Queuing diagram
Announcement

- Homework #2 will be on the course Webpage today.
- Due date is next Wednesday
- Please read chapters about Linux Programming (processes) uploaded in Piazza and do Exercise #2
Process States
Recap: Two-State Process Model

- Necessity of processes and Process Control Block (PCM)
- Previously we discussed the two-state model
- A process may be in one of two states:
  - Running
  - Not-running

Transitions between these states is performed by dispatcher
Process Creation and Termination
Reasons for Process Creation

• Submission of a batch job
• User logs on
• Created to provide a service
  – E.g., for printing, network management, etc
• One process creates another process
  – For modularity
  – This is something you would do in your projects as well
## Reasons for Process Creation

<table>
<thead>
<tr>
<th>Reason</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>New batch job</td>
<td>The OS is provided with a batch job control stream, usually on tape or disk. When the OS is prepared to take on new work, it will read the next sequence of job control commands.</td>
</tr>
<tr>
<td>Interactive log-on</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 OS 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>
There must be some way that a process can indicate completion

This indication may be:

- A **HALT** instruction generating an interrupt alert to the OS: A batch job should include a HALT instruction or an explicit OS service call for termination (Batch job terminate)
- A **user action** (e.g. log off, quitting an application) : For an interactive application, the action of the user will indicate when the process is completed
- A **fault** or error
- **Parent process terminating**
Reasons for Process Termination (I)

- Normal termination
- Time limit exceeded
- Memory limitation
- Illegal memory access
- Protection errors; Performing operations with resources that are not permitted
- Arithmetic error
- Process waited longer than a specified maximum for an event
Reasons for Process Termination (II)

- I/O failure
- Invalid instruction
- When process tries to execute data
- If return address from a procedure is corrupted
- Privileged instruction execution
- Intervention by operating system (e.g., Deadlock resolution)
- Signals (/usr/include/bits/signum.h @pacific)
- Termination of parent terminates the child process
Other States

• New state
  – OS has performed the necessary actions to create the process
    • assigning a process identifier
    • created data structures to manage the process
  – the process execution has not begun
    • e.g. if resources are limited

• Exit state
  – Process has terminated
  – Data structure about the process is preserved
  – E.g., if parent is allowed to check the status of its children
Problems of 2 state model:

- **Running**
- **Not-running**
  - Ready to execute
  - Blocked (e.g., waiting for I/O)

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

It is not fair!!
Problems of 2 state model:

- **Running**
- **Not-running**
  - Ready to execute
  - Blocked (e.g., waiting for I/O)

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

It is not fair!!
Five-State Process Model

- Running: The process that is currently being executed.
- Ready: A process that is prepared to execute when given opportunity.
- Blocked: A process can not execute until some event occurs (completion of an I/O operation)
- New
- Exit
Process State

- As a process executes, it changes state
  - **New**: The process is being created
  - **Running**: Instructions are being executed
  - **Blocked**: The process is waiting for some event to occur
  - **Ready**: The process is waiting to be assigned to a CPU
  - **Exit**: The process has finished execution

In a single-CPU system, only **ONE** process may be in running state; many processes may be in ready and waiting states.
Process States for Trace of Figure 3.4

Figure 3.7 Process States for Trace of Figure 3.4
Refinement for Processor Organization Using Two Queues

Can we make the queueing system more efficient?
Refinement for Process Organization: Multiple Blocked Queues

(b) Multiple blocked queues
Swapping

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

- Most of processes usually require I/O

- All of the processes in memory to be waiting for I/O

- There is a limit on memory to bring in multiple processes for execution
  \[\Rightarrow\text{Degrades the utilization of CPU (CPU could be idle most of the time)}\]

- OS may swap some processes to disk in order to improve performance
Suspended Processes

- **Swapping**: moving part or all of a process from main memory to disk to free up more memory
- when none of the processes in main memory is in the Ready state, the OS swaps one of the processes out on to disk into a suspend queue.

- Blocked state becomes suspend state when swapped to disk

- One new state
  - Blocked/Suspend state

- Which processes should be selected by OS?
One Suspend State

(a) With One Suspend State
Two Suspend States

1) Some times the process in Ready state is the only way to free up a sufficiently large block of MM.

2) There is a higher priority blocked process which will be ready soon and lower priority process in ready state

One new state
- Ready/Suspend state

- swapped process may be either blocked or ready
Two Suspend States

(b) With Two Suspend States
## Reasons for Process Suspension

<table>
<thead>
<tr>
<th>Reason</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Swapping</td>
<td>Due to lack of available physical memory</td>
</tr>
<tr>
<td>Other OS Reason</td>
<td>OS suspects process of causing a problem.</td>
</tr>
<tr>
<td>Interactive User Request</td>
<td>e.g. 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. Periodically executing processes</td>
</tr>
<tr>
<td>Parent Process Request</td>
<td>Under certain conditions, a parent can suspend a child process</td>
</tr>
</tbody>
</table>
Characteristic of a Suspended Process

- The process is not immediately available for execution
- The process was placed in a suspended state by an agent: either itself, a parent process, or the OS, for the purpose of preventing its execution
- The process may or may not be waiting on an event
- The process may not be removed from this state until the agent explicitly orders the removal
Operating System Control Structure for Process Management
Operating System Control Structures

- Tables are constructed for each entity the OS manages
- OS controls tables
- Information about the current status of each process and resource

Figure 3.11 General Structure of Operating System Control Tables
Memory Tables

- Used to **keep track** of both main (real) and secondary (virtual) memory

- Processes are maintained on secondary memory using some sort of virtual memory or simple swapping mechanism

**Must include:**

- **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 (Chapter 7 & 8)
I/O Tables

- Used by the OS to manage the I/O devices and channels of the computer system
- At any given time, an I/O device may be available or assigned to a particular process

If an I/O operation is in progress, the OS needs to know:
- I/O device is available or assigned
- the status of the I/O operation
- the location in main memory being used as the source or destination of the I/O transfer
These tables provide information about:

- existence of files
- location on secondary memory
- current status
- other attributes

- Information may be maintained and used by a file management system
- OS needs to implement an integrated file management system
Process Tables

- To manage processes the OS must know details of the processes
  - Current state (e.g., sleeping, running, zombie,)
  - PID
  - Location in memory
  - Priority

*Process image* is the collection of program, data, stack, and the *process control block* (process attributes)
Typical Elements of a Process Image

**User Data**
The modifiable part of the user space. May include program data, a user stack area, and programs that may be modified.

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

**Stack**
Each process has one or more last-in-first-out (LIFO) stacks associated with it. A stack is used to store parameters and calling addresses for procedure and system calls.

**Process Control Block**
Data needed by the OS to control the process (see Table 3.5).
### Process Control Structures

#### Process Location

- A process must include a **program** or set of programs to be executed.

- A process will consist of at least **sufficient memory** to hold the programs and **data** of that process.

- The execution of a program typically involves a **stack** that is used to keep track of procedure calls and parameter passing between procedures.

#### Process Attributes

- Each process has associated with it a **number of attributes** that are used by the OS for process control.

- The collection of **program**, **data**, **stack**, and **attributes** is referred to as the process image.

- Process image location will depend on the memory management scheme being used.
Example PCB Contents (I)

- Control bits
- Process state information
- Process control information
Example PCB Contents (II)

• Process 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

• Links to other processes
  – Parent, child, …

• Processor State Information
  – User-Visible Registers
  – Stack Pointers
• Process context

  – 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
In Summary: Organization of Processes in Different States

Figure 3.14 Process List Structures
Role of the Process Control Block

- The most important data structure in an OS
  - contains all of the information about a process that is needed by the OS
  - blocks are read and/or modified by virtually every module in the OS
  - defines the state of the OS

- Difficulty is not access, but protection
  - a bug in a single routine could damage process control blocks, which could destroy the system’s ability to manage the affected processes
  - a design change in the structure or semantics of the process control block could affect a number of modules in the OS
Outline

• Process creation, termination, states of Processes
• Process image, process control block
• User mode vs privileged mode
• Different OS designs
Modes of Execution

Most processors support at least two modes of execution

**User Mode**
- less-privileged mode
- user programs typically execute in this mode

**System Mode**
- more-privileged mode
- also referred to as control mode or kernel mode
- kernel of the operating system
- Once the OS decides to create a new process it:

  - Assigns a unique process identifier to the new process
  - Allocates space for the process
  - Initializes the process control block
  - Creates or expands other data structures
  - Sets the appropriate linkages
After creating the process the Kernel can do one of the following, as part of the dispatcher routine:

- **stay in the parent process:**
  - Control returns to user mode at the point of the fork call of the parent.

- **transfer control to the child process:**
  - The child process begins executing at the same point in the code as the parent, namely at the return from the fork call.

- **transfer control to another process:**
  - Both parent and child are left in the Ready to Run state.
A process switch may occur any time that the OS has gained control from the currently running process. Possible events giving OS control are:

<table>
<thead>
<tr>
<th>Mechanism</th>
<th>Cause</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt</td>
<td>External to the execution of the current instruction</td>
<td>Reaction to an asynchronous external event</td>
</tr>
<tr>
<td>Trap</td>
<td>Associated with the execution of the current instruction</td>
<td>Handling of an error or an exception condition</td>
</tr>
<tr>
<td>Supervisor call</td>
<td>Explicit request</td>
<td>Call to an operating system function</td>
</tr>
</tbody>
</table>
System Interrupts

Interrupt

- Due to some sort of event that is external to and independent of the currently running process
  - clock interrupt
  - I/O interrupt
  - memory fault
- Time slice
  - the maximum amount of time that a process can execute before being interrupted

Trap

- An error or exception condition generated within the currently running process
- OS determines if the condition is fatal
  - moved to the Exit state and a process switch occurs
  - action will depend on the nature of the error
The steps in a full process switch are:

1. Save the context of the processor
2. Update the process control block of the process currently in the Running state
3. Move the process control block of this process to the appropriate queue
4. Select another process for execution
5. Update memory management data structures
6. Restore the context of the processor to that which existed at the time the selected process was last switched out
Summary

- The most fundamental concept in a modern OS is the process
- The principal function of the OS is to create, manage, and terminate processes
- Process control block contains all of the information that is required for the OS to manage the process, including its current state, resources allocated to it, priority, and other relevant data
- The most important states are Ready, Running and Blocked
- The running process is the one that is currently being executed by the processor
- A blocked process is waiting for the completion of some event
- A running process is interrupted either by an interrupt or by executing a supervisor call to the OS
What is the following pseudocode doing?

```
main()
{
    pid_t n=0;

    for (i=0; i<10; ++i) {
        n = fork();
        if (n==0) {
            print ("hello");
            exit (0);
        }
    }

    for (i=0; i<10; ++i)
        wait();
}
```

What is the following pseudocode doing?

```
main()
{
    pid_t n=0;

    for (i=0; i<2; ++i) {
        print (i);
        n = fork();
        print ("hello");
    }
}
```
What is the following pseudocode doing?

```c
main()
{
    pid_t n, m;

    n = fork();
    if (n == 0) {
        m = fork();
        if (m == 0) {
            print(“hello”);
        }
    }
}
```
What is the following pseudocode doing?

```c
main() {
    pid_t n,m;

    n = fork();
    if (n == 0) {
        m = fork();
        if (m == 0) {
            print ("hello");
            exit(0);
        }
        exit(0);
    }
    waitpid(n);
}
```
Example: fork() Exercise 2.1

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int A = 47;

int main()
{
    int B = 23;
    pid_t pid;
    printf( "\nPID: %d  A: %d  B: %d\n", getpid(), A, B );
    pid = fork();
    printf( "\nPID: %d  A: %d  B: %d\n", getpid(), A, B );
    // sleep(20);
    if (pid < 0)
    {
        printf( "\n*** Error: fork failed ***\n" );
        exit( 1 );
    }
    else if (pid == 0)
    {
        A = A + 10;
        B = B + 10;
    }
    else
    {
        A = A + 25;
        B = B + 25;
    }
    printf( "\nPID: %d  A: %d  B: %d\n", getpid(), A, B );
    exit( 0 );
}
```
Example: execv()

```c
using namespace std;
#include <iostream>
#include <unistd.h>

int main()
{
    int retval;
    retval = fork();

    char *argv[10];
    argv[0] = "ls";
    argv[1] = "-l";
    argv[2] = NULL;

    execvp("ls", argv);
    cout << " test" << endl;

    return 0;
}
```
Execution of the Operating System

(a) Separate kernel

(b) OS functions execute within user processes

(c) OS functions execute as separate processes

Figure 3.15 Relationship Between Operating System and User Processes
Execution *Within* User Processes

![Diagram of process image](image_url)

**Figure 3.16** Process Image: Operating System Executes within User Space
Security Issues

- An OS associates a set of privileges with each process.
- Typically a process that executes on behalf of a user has the privileges that the OS recognizes for that user.
- Highest level of privilege is referred to as administrator, supervisor, or root access.
- A key security issue in the design of any OS is prevent, or at least detect, attempts by a user or malware from gaining unauthorized privileges and from gaining root access.
- Uses the model where most of the OS executes within the environment of a user process

- System processes run in kernel mode
  - executes operating system code to perform administrative and housekeeping functions

- User Processes
  - operate in user mode to execute user programs and utilities
  - operate in kernel mode to execute instructions that belong to the kernel
  - enter kernel mode by issuing a system call, when an exception is generated, or when an interrupt occurs
## 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>
UNIX Process State Transition Diagram

Figure 3.17 UNIX Process State Transition Diagram
# A Unix Process

<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>

<table>
<thead>
<tr>
<th>Register Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program counter</td>
</tr>
<tr>
<td>Processor status register</td>
</tr>
<tr>
<td>Stack pointer</td>
</tr>
<tr>
<td>General-purpose registers</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>System-Level Context</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process table entry</td>
</tr>
<tr>
<td>U (user) area</td>
</tr>
<tr>
<td>Per process region table</td>
</tr>
<tr>
<td>Kernel stack</td>
</tr>
</tbody>
</table>
### Table 3.11

**UNIX Process Table Entry**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process status</td>
<td>Current state of process.</td>
</tr>
<tr>
<td>Pointers</td>
<td>To U area and process memory area (text, data, stack).</td>
</tr>
<tr>
<td>Process size</td>
<td>Enables the operating system to know how much space to allocate the process.</td>
</tr>
<tr>
<td>User identifiers</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>Process identifiers</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>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>Process table pointer</td>
<td>Indicates entry that corresponds to the U area.</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-----------------------------------------------</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>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 opened.</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>