Process Description and Control

Chapter 3
Processes

- Working definition: An instance of a program
- Processes are among the most important abstractions in an OS
  - all the running software on a computer, including the OS itself, is organized as a set of processes
  - As a unit of execution, they enable multiprogramming (the execution of multiple programs on a system at the same time)
- Processes are a fundamental OS concept
OS Requirements for Processes

- OS must **interleave the execution** of several processes to maximize CPU usage while providing reasonable response time.
- OS must **allocate resources** to processes while avoiding deadlock.
- OS must **support inter process communication** and user creation of processes.
Process Image

- Process image: the collection of code, data, stack, and attributes for a single process

- Program code:
  - the instructions that the program to execute
  - may be shared with other processes (e.g., dynamically linked libraries)

- Data:
  - the data on which the program operates

- Stack:
  - one or more last-in-first-out (LIFO) stacks
  - used to store parameters and calling addresses for procedure and system calls
Process Image

- **Stack segment:**
  - used for function and system calls

- **Data segment:**
  - contains variables, both static and dynamically allocated on the heap

- **Text segment:**
  - the program code (read-only)
Process Control Block

- **Process Control Block (PCB)**
  - A data structure containing meta-data of each process (to save the attributes)
  - Used by the operating system to manage the execution of the process

- **Includes information like:**
  - Process Identification
  - Processor State Information
  - Process Control Information
Process Control Block

- Process Control Block (PCB)
- Process Identification

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

- Process Control Block (PCB)
- Processor State Information

**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 while in user mode. Typically, there are from 8 to 32 of these registers, although some RISC implementations have over 100.

**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

**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.
Process Control Block

- Process Control Block (PCB)
- Process Control Information

Scheduling and State Information
This is information 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.

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

- Process Control Block (PCB)
- Process Control Information

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.

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.
Process Image

- Process image: the collection of program, data, stack, and PCB (to store attributes)

![Diagram of Process Image]

- Process identification
- Processor state information
- Process control information
- User stack
- Private user address space (programs, data)
- Shared address space
Dispatcher
(short-term scheduler)

- It is an OS program that moves the processor from one process to another.
- It prevents a single process from monopolizing processor time.
- It decides who goes next according to a scheduling algorithm (Chap. 9).
- The CPU will always execute instructions from the dispatcher while switching from process A to process B.
Multiprogramming Example
### Multiprogramming Example

<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
We assume the unit of execution is 6 (the CPU runs 6 instructions before timeout)
Two-Stage Process Model

(a) State transition diagram

(b) Queueing diagram
Two-Stage Process Model

- Over-simplified model
  - a process may be in one of two states
- Running:
  - the process is currently executing on a processor
- Not-running:
  - the processor is not currently executing on a processor
Process Creation

- **System boot**
  - when a system is initialized, several processes are started

- **One process spawning another**
  - a web server process might spawn a child process for each request handled

- **User requests the creation of a process**
  - E.g., the user might type a command into a shell, the result is a new process, with the shell as the parent process

- **Batch jobs**: a batch system begins processing the next job in the queue
Process Creation

- To create a process, several steps are required
  - the process must be assigned a unique identifier
  - space must be allocated in memory for the process image (space for code, data, stack, PCB)
  - the process must be loaded from secondary storage
  - the process control block must be initialized
  - the process must be added to a ready queue
Process Termination

- Processes must be able to indicate their termination
- Common reasons for process termination
  - normal completion
  - time limit exceeded (diff from a time-out)
  - insufficient memory available
  - memory bound violation
  - illegal operation (e.g. division by zero, attempt to use a privileged instruction)
  - parent request
Two-Stage Process Model

What is the problem with the two-stage process model?
ANDROID

- Cupcake: 04/2009
- Donut: 09/2009
- Eclair: 10/2009
- Froyo: 05/2010
- Gingerbread: 12/2010
- Honeycomb: 02/2011
- Ice Cream Sandwich: 10/2011
- Jelly Bean: 08/2012
- KitKat: 09/2013
A Two-Stage Process Model

- Two-stage model cannot deal with processes with I/O requests (i.e., blocked)
Process States

Let us start with these states:

- The **Running** state
  - The process that gets executed (single CPU)
- The **Ready** state
  - any process that is ready to be executed
- The **Blocked** state
  - when a process cannot execute until some event occurs (e.g., the completion of an I/O)
Three-state Process Model

- Two-stage model cannot deal with processes with I/O requests (i.e., blocked)
Three-state Process Model
Five-State Process Model
Other Useful States

- **The New state**
  - OS has performed the necessary actions to create the process
    - has created a process identifier
    - has created tables needed to manage the process (e.g., to schedule the processes)
  - but has not yet committed to execute the process (not yet admitted)
    - because resources are limited
    - typically, very limited resources are allocated
Other Useful States

- **The Exit state**
  - Termination moves the process to this state
  - It is no longer eligible for execution
  - Tables and other info are temporarily preserved for auxiliary program
    - E.g., accounting program that cumulates resource usage for billing the users
  - The process (and its tables) gets deleted when the data is no more needed
Process Transitions

- **Ready --> Running**
  - When it is time, the dispatcher selects an appropriate process to run

- **Running --> Ready**
  - the running process has expired its time slot
  - the running process gets interrupted because a higher priority process is in the ready state
Process Transitions

- **Running --> Blocked**
  - When a process requests something for which it must wait
    - an access to a resource not yet available (resource)
    - initiates I/O and must wait for the result
    - a service that the OS is not ready to perform
    - waiting for a process to provide input (synchronization)
Process Transitions

- Blocked $\rightarrow$ Ready
  - When the event for which it was waiting occurs
- Ready (Blocked) $\rightarrow$ Exit \textit{(not drawn)}
  - A parent may terminate a child process
The need for swapping

- So far, all the processes had to be (at least partly) in main memory.
- Even with virtual memory, keeping too many processes in main memory will deteriorate the system’s performance.
- The OS may need to suspend some processes, i.e., to swap them out to disk.
The need for swapping

- Redefinitions of previous states:
  - ready: the process is in main memory and available for execution
  - blocked: the process is in main memory and awaiting the completion of an event

- We add two new states:
  - Ready Suspend
    - ready processes which have been swapped out to disk
    - available for execution as soon as it is loaded into main memory
  - Blocked Suspend
    - blocked processes which are swapped out to disk
    - awaiting the completion of an event
A Seven-state Process Model
New State Transitions (mid-term scheduling)

- **Blocked --→ Blocked Suspend**
  - When all processes are blocked, the OS will make room to bring a ready process in memory

- **Blocked Suspend --→ Ready Suspend**
  - When the event for which it has been waiting occurs (state info is available to OS)

- **Ready Suspend --→ Ready**
  - When no more ready process in main memory

- **Ready --→ Ready Suspend (less likely)**
  - When there are no blocked processes and must free memory for adequate performance
A Review of Stage Models
An OS maintains several data tables for managing processes and resources:
- Memory tables
- I/O tables
- File tables
- Process tables
OS Control Structures

- **Memory tables:**
  - tracks the allocation of main memory to processes
  - additional information needed to manage virtual memory
  - various other attributes (dirty, protected, etc.)

- **I/O tables:**
  - used to manage and track state of all the various I/O devices in a system
  - each I/O device may be assigned to process
  - OS must know source or destination in memory for results
OS Control Structures

- **File tables:**
  - provides information about the existence of files, their location in secondary storage, and various attributes

- **Process tables:**
  - list of active processes, resources they’re using, etc.
What the Tech Industry Has Learned from Linus Torvalds
Modes of Execution

- **Kernel mode**
  - allows execution of privileged instructions
  - allows access to restricted memory locations
  - operating system typically runs in this mode

- **User mode**
  - less privileged mode
  - user programs run in this mode and should be prevented from attaining kernel mode
Mode Switching

- When the switch happens
  - on detecting an interrupt
    - the processor will switch to kernel mode and execute the appropriate interrupt handler
    - the OS must reset to user mode afterwards
  - on system calls
    - a special instruction denotes that a system call has been requested
    - this instructions causes the processor to switch into kernel mode
Process Switching

- Process Switch (or context switch)
- Process switches happen for a number of reasons
  - Exceptions
    - internal, synchronous events
    - due to invalid instructions (division by zero, attempt to execute a privileged instruction)
Process Switching

- Process Switch (or context switch)
- Process switches happen for a number of reasons
  - Exceptions
  - Interrupts
    - external asynchronous events
    - beyond the control of the currently executing process
  - examples include:
    - timer interrupt: your turn is up
    - I/O interrupt: I/O request has finished or needs attention
    - memory fault: must wait for missing page to be loaded from memory
Process Switching

- Process Switch (or context switch)
- Process switches happen for a number of reasons
  - Exceptions
  - Interrupts
  - System calls

Application developers often do not have direct access to the system calls, but can access them through an application programming interface (API).
Process Switching

- Steps involved:
  - Save the current CPU context (e.g., program counter and other registers)
  - Update the process state information in the process’ PCB (e.g., running -> ready)
  - Move PCB to appropriate queue (e.g., ready, blocked)
  - Select another process for execution (by a scheduling algorithm in Chap 9)
  - Update PCB of the selected process (e.g., ready-> running)
  - Restore CPU context from that of the selected process
  - Begin executing the new process
Process Switching

- Compared to a mode switch:
  - a mode switch is simply a change in the current privilege level (e.g. user to kernel mode)
  - a process switch is significantly more expensive and likely involves multiple mode switches
OS Execution

- OS commonly runs inside a user process
  - mode switch happens (interrupt, system call)
  - OS saves current process context (registers, etc.)
  - performs requested operation

- The OS may choose
  - to return control to the user process
    - in this case, only a mode switch back to user mode is required
  - to **preempt** the process and begin executing another
    - in this case, a full process switch must occur
Virtually all OS code gets executed within the context of a user process.
Academic Integrity: All students are advised to be familiar with university policy on Academic Integrity. In addition, the following Collaboration Policy exists for all CS@Mines courses.

1. If the project is an individual effort project, you are not allowed to give code you have developed to another student or use code provided by another student. If the project is a group project, you are only allowed to share code with your group members.

2. You are encouraged to discuss homework and final project assignments with other students in the class, as long as the following rules are followed:
   a. You view another student's code only for the purpose of offering/receiving debugging assistance. Students can only give advice on what problems to look for; they cannot debug your code for you. All changes to your code must be made by you.
   b. Your discussion is subject to the empty hands policy, which means you leave the discussion without any record [electronic, mechanical or otherwise] of the discussion.

3. Any material from any outside source such as books, projects, and in particular, from the Web, should be properly referenced and should only be used if specifically allowed for the assignment.

4. To prevent unintended sharing, any code stored in a hosted repository (e.g., on github) must be private. For group projects, your team members may, of course, be collaborators.

5. If you are aware of students violating this policy, you are encouraged to inform the professor of the course. Violating this policy will be treated as an academic misconduct for all students involved. See the Student Handbook for details on academic dishonesty.

Violations of this policy result in one of a range of punitive measures, from a zero score for an assignment, up to and including a course letter grade drop for all students involved. All issues of misconduct are reported to the Dean of Students. Academic misconduct associated with an exam grade will likely result in course failure.

Academic Misconduct will be dealt with harshly!