Chapter 6
Process Management

Introduction
- Scenario
  - One process running
  - One/more process performing I/O
  - One/more process waiting on resources
- Most of the complexity stems from the need to manage multiple processes

Introduction
- Process Manager
  - CPU sharing
  - Process synchronization
  - Deadlock prevention
- Each process has a Process Descriptor
  - Describes complete environment for a process

Process Address Space
- Defines all aspects of process computation
  - Program
  - Variables
  - ...
- Address space is generated/defined by translation

Creating an executable program
- Separate objects each relative to 0
- One large program
- Y = (X+Y)
- Maps relative address space to physical memory location
- Generates separate object code modules
- Relocates modules one behind other
- Relocates addresses of all but first
- Resolves external reference to library calls and external modules
Basic Memory Hierarchy

- Fastest
- Access Speed
- Slowest

At any point in the same program, element can be in
- Secondary memory $M_S$
- Primary memory $M_P$
- Registers $M_R$

Consistency is a Problem
- $M_S = M_P = M_R$ (code vs data)
- When does one make them consistent?
- How?

Consistency Problem
- Scheduler switching out processes - Context Switch
- Is Instruction a Problem???
  - NO
  - Instructions are never modified
  - Separate Instruction and Data space
  - Therefore, $M_{Ri} = M_{Pi} = M_{Si}$

How can an instruction be in a register?

Consistency Problem...
- Is Data a Problem???
  - YES
  - Variable temporarily stored in register has value added to it
  - Therefore, $M_{Ri} = M_{Si}$

On context switch, all registers are saved
- Therefore, current state is saved

Sample Scenario...
- Suppose ‘MOV X Y’ instruction is executed
  - $M_{Si} = M_{Yi}$
- On context switch, is all of a process’ memory flushed to $M_S$?
  - No, only on page swap
- Hence, $env_{process} = (M_R + M_S) + \{\ldots\}$

Note:
- Flushing of memory frees it up for incoming process
  $\Rightarrow$ Page Swap

Process States
- Focus on Resource Management & Process Management
- Recall also that part of the process environment is its state

State Transition Diagram
Process States...
1. When process enters 'Ready' state, it must compete for CPU. Memory has already been allocated.
2. Process has CPU
3. Process requests resource that is immediately available → NO blocking
4. Process requests resource that is NOT yet available
5. Resource allocated, memory re-allocated?

State Transition Diagram

Resources & Resource Manager
- 2 types of Resources
  - Reusable (Memory)
  - Consumable (Input/Time slice)

Units of Resource R
- Process requests resource unit(s)
  - Get it, or
  - Block → Stay in Queue

Resources & Resource Manager

Process Hierarchy
- Conceptually, this is the way in which we would like to view it
- Root controls all processes i.e. Parent

Creating Processes
- Parent Process needs ability to
  - Block child
  - Activate child
  - Destroy child
  - Allocate resources to child

- True for User processes spawning child
- True for OS spawning init, getty, etc.
- Process hierarchy a natural, if fork/exec commands exist

UNIX fork command
- Fork
  - Shares text
  - Shares memory
  - Has its own address space
  - Cannot communicate with parent by referring variable stored in code

- Earlier definition: ForkConway
  - Shares text
  - Shares resources
  - Shares address space
  - Process can communicate thru variables declared in code
Cooperating Processes

Program

\[ \text{x, y : int} \]

Proc A
ref x & y
Proc B
ref x & y
Fork "A"
Fork "B"

Now processes A & B, share address space & can communicate thru declared variables

Problem ??

A can write 2 times before B reads

Synchronizing Access to Shared Variables

- Shared address space allows communication through declared variables automatically
- How then, can we synchronize access to them?
- Need Synchronization Primitives

=> JOIN & QUIT

Fork, Join & Quit - Conway

- In addition to the "Fork(proc)" command, Conway also defined system calls to support process synchronization
  - Join (count)
    - Un-interruptible
      - Decrement count;
      - if count ≠ 0 then Quit, else Continue
  - Quit
    - Terminate process

Fork, Join, Quit example

Spawning A Child Different From Parent

- Suppose we wish to spawn a child that is different from the parent

\[ \text{fork } \]
\[ \text{execve(…)} \]

OS
\[ \text{init} \]
\[ \text{getty} \]
\[ \text{shell} \]
Factoring in additional Control Complexities

- Recall:
  - A parent process can suspend a child process.

- Therefore, if a child is in run state and goes to ready (time slice up), and the parent runs and decides to suspend the child, then how do we reflect this in the process state diagram???

- We need 2 more states
  - Ready suspended
  - Blocked suspended

Process State diagram reflecting Control

Give it a thought...

Why can a process NOT go from 'Ready Active' to 'Blocked Active' or 'Blocked Suspended'?