Lesson 6

Objectives

- Process management
- Process Creation, Execution and Termination
- Process Hierarchies
- Process Control Block

PROCESS MANAGEMENT

1. A process is a program in execution. It is a unit of work within the system. Program is a *passive entity*, process is an *active entity*.
2. Process needs resources to accomplish its task
    a. CPU, memory, I/O, files
    b. Initialization data
3. Process termination requires reclaim of any reusable resources
4. Single-threaded process has one program counter specifying location of next instruction to execute
    a. Process executes instructions sequentially, one at a time, until completion
5. Multi-threaded process has one program counter per thread
6. Typically system has many processes, some user, some operating system running concurrently on one or more CPUs
    a. Concurrency by multiplexing the CPUs among the processes / threads

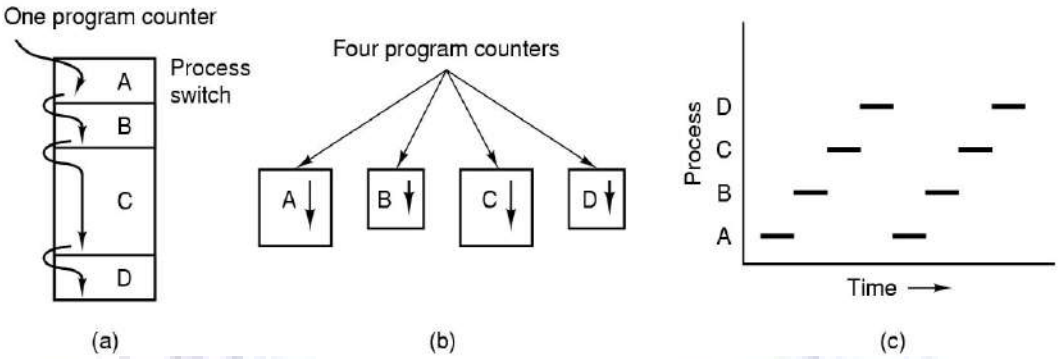
Process management activities

The operating system is responsible for the following activities in connection with process management:

- Creating and deleting both user and system processes
- Suspending and resuming processes
- Providing mechanisms for process synchronization
- Providing mechanisms for process communication
- Providing mechanisms for deadlock handling

For example we have 4 processes namely A, B, C and D. Only one program active at any instant, in a multiprogramming environment the execution can be demonstrated as below.

Here (a) shows process queue, with initially single program counter, in (b) each process is distinguished by different program counters, while (c) shows the time-division of the four on time-lines.

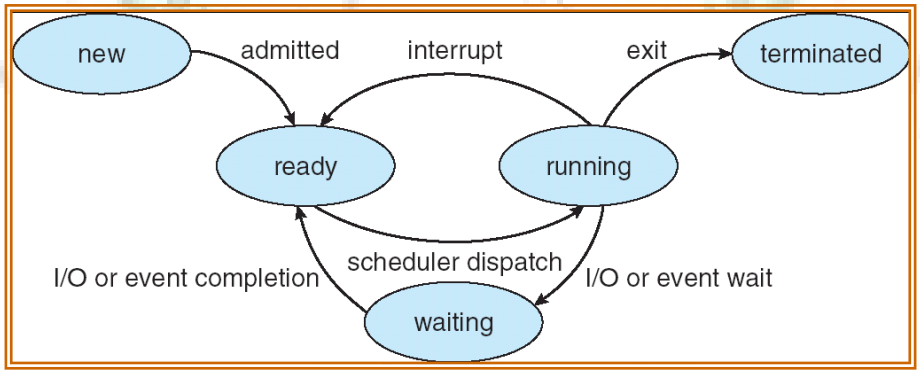


PROCESS HIERARCHIES

1. Parent creates a child process, child processes can create their own processes
2. Forms a hierarchy
   - UNIX calls this a "process group"
3. Windows has no concept of process hierarchy
   - all processes are created equal

PROCESS STATES

Generally a process can be divided into five states, from its creation to termination, as shown below:



New: The process is being created

Running: Instructions are being executed

Waiting: The process is waiting for some event to occur (such as an I/O completion or reception of a signal)

Ready: The process is waiting to be assigned to a processor

Terminated: The process have finished execution

PROCESS CONTROL BLOCK

Each process is represented in the operating system by a *process control block* (PCB)—also called a task control block. It guarantees the process implementation. Following information and steps are involved. It contains entire process status and hardware status being involved in process. These are stored in process tables. Here are given the Fields of a process table entry.

Process management	**Memory management**	**File management**
Registers	Pointer to text segment	Root directory
Program counter	Pointer to data segment	Working directory
Program status word	Pointer to stack segment	File descriptors
Stack pointer		User ID
Process state		Group ID
Priority		
Scheduling parameters		
Process ID		
Parent process		
Process group		
Signals		
Time when process started		
CPU time used		
Children's CPU time		
Time of next alarm		

- The *code segment*, also known as *text segment* contains the machine instructions of the program. The code can be thought of like the text of a novel: It tells the story of what the program does.
- The *data segment* contains the *static* data of the program, i.e. the variables that exist throughout program execution. Global variables in a C or C++ program are static, as are variables declared as static in C, C++, or Java.
- The *stack segment* contains the system stack, which is used as temporary storage. The stack is a simple data structure with a *LIFO (last-in first-out)* access policy. Items are only added to or removed from the "top" of the stack. Implementing a stack requires only a block of memory (e.g. an array in a HLL) and a *stack pointer* which tells us where the top of the stack is.

Also given a skeleton of what lowest level of OS does when an interrupt occurs.

1. Hardware stacks program counter, etc.
2. Hardware loads new program counter from interrupt vector.
3. Assembly language procedure saves registers.
4. Assembly language procedure sets up new stack.
5. C interrupt service runs (typically reads and buffers input).
6. Scheduler decides which process is to run next.
7. C procedure returns to the assembly code.
8. Assembly language procedure starts up new current process.

It is a common operating system practice that simultaneously several processes are being executed in a time division fashion. So CPU switches among them in a sophisticated fashion so that each process thinks that he is utilizing the CPU alone. Here is a diagram showing this fact.