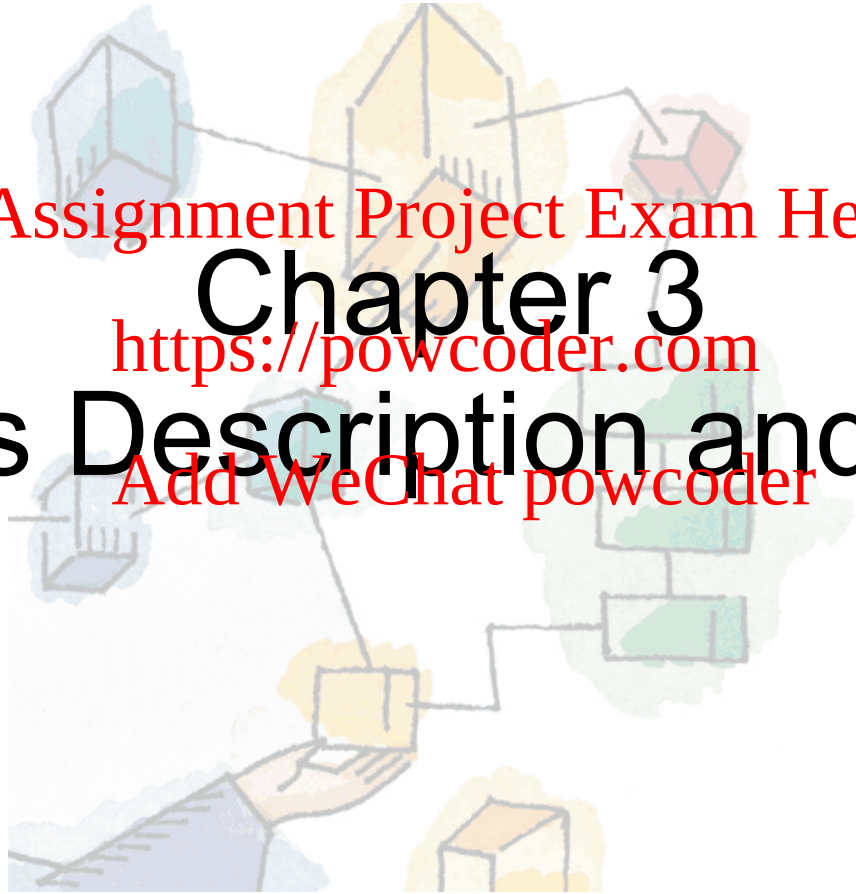
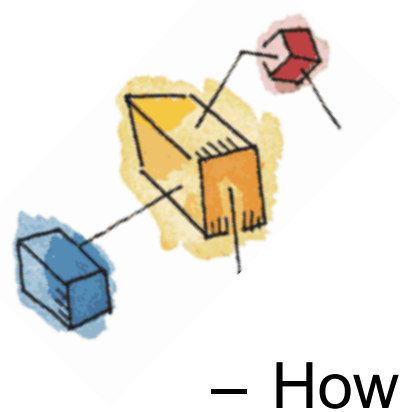
Operating Systems:
Internals and Design Principles
William Stallings

Chapter 3

Process Description and Control





Objectives

- How are processes represented and controlled by the OS.
- ***Process states*** which characterize the behaviour of processes.
- ***Data structures*** used to manage processes.
- Ways in which the OS uses these data structures to control process execution.

Time Sharing Systems

- Using multiprogramming to handle multiple interactive jobs
- Processor's time is shared among multiple users
- Multiple users simultaneously access the system through terminals

Process

- Fundamental to the structure of operating systems

A *process* can be defined as:

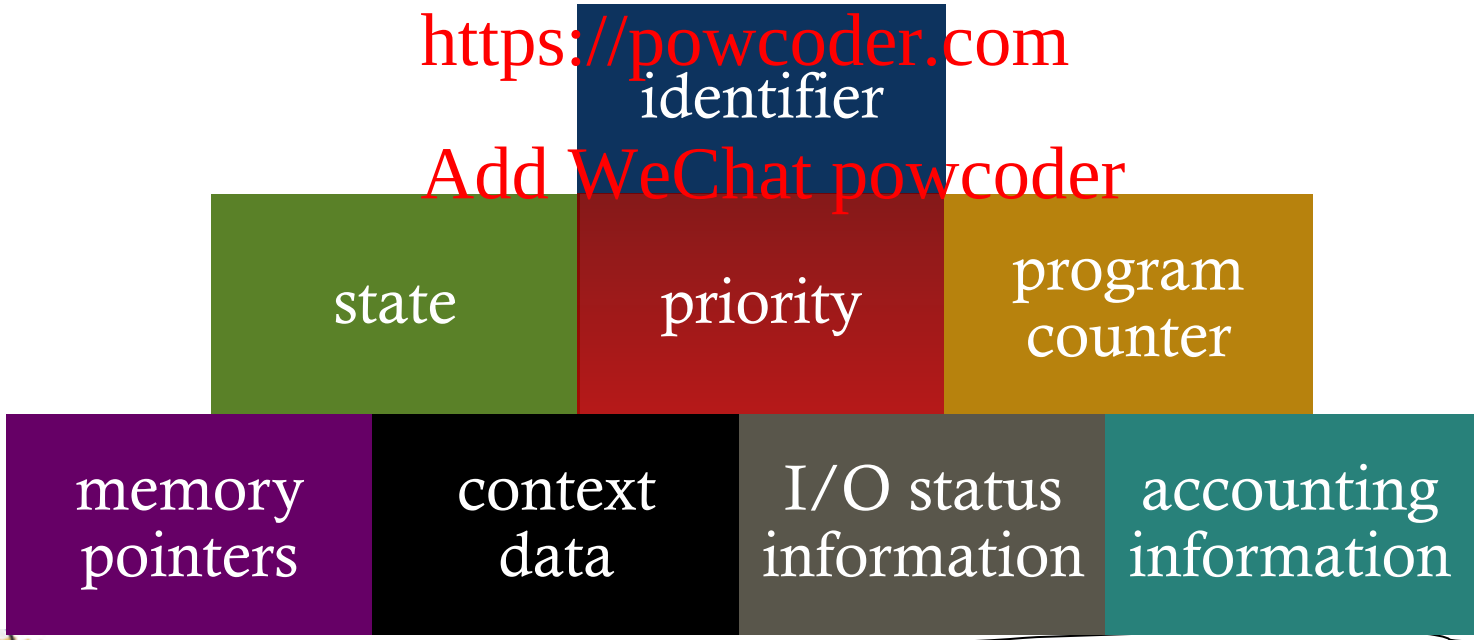
- a program in execution
- an instance of a running program
- the entity that can be assigned to, and executed on, a processor
- a unit of activity characterized by a single sequential thread of execution, a current state, and an associated set of system resources

Process Management

- Is the *Fundamental Task*
- The Operating System must
  - Allocate resources to processes, and protect the resources of each process from other processes,
  - Interleave the execution of multiple processes
  - Enable processes to share and exchange information,
  - Enable synchronization among processes.

Process Elements

- While the program is executing, this process can be uniquely characterized by a number of *attributes*, including:



Process Control Block

- The most important data structure in an OS
- Contains the process attributes
- Created and managed by the operating system
- Key tool that allows support for multiple processes
- Attributes in general categories:
  - Process identification
  - Processor state information
  - Process control information

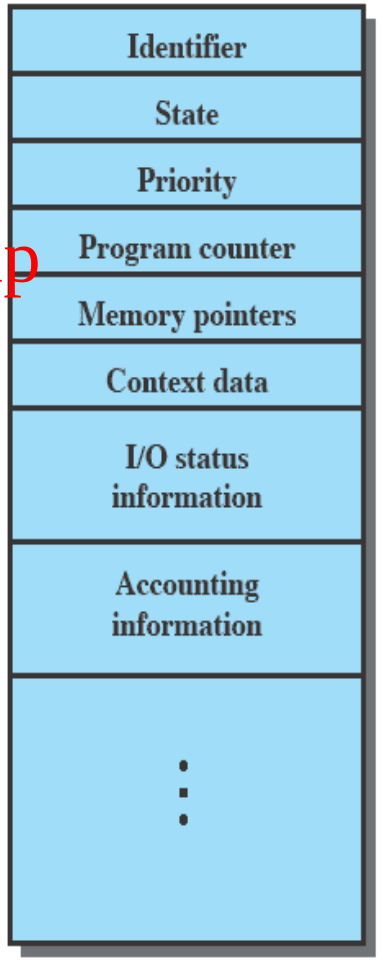


Figure 3.1 Simplified Process Control Block

Process Attributes

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

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 Attributes

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.

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

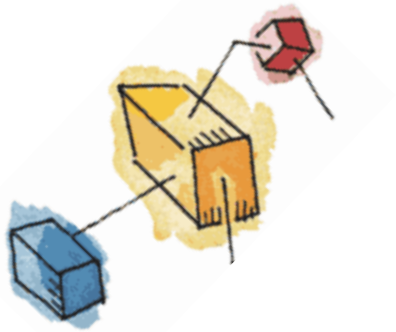
Processes are granted privileges in terms of the memory that may be accessed and the types of instructions that may be executed. In addition, privileges may apply to the use of system utilities and services.

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

- Typical elements:

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

- Process image location will depend on the memory management scheme being used

Structure of Process Images in Virtual Memory

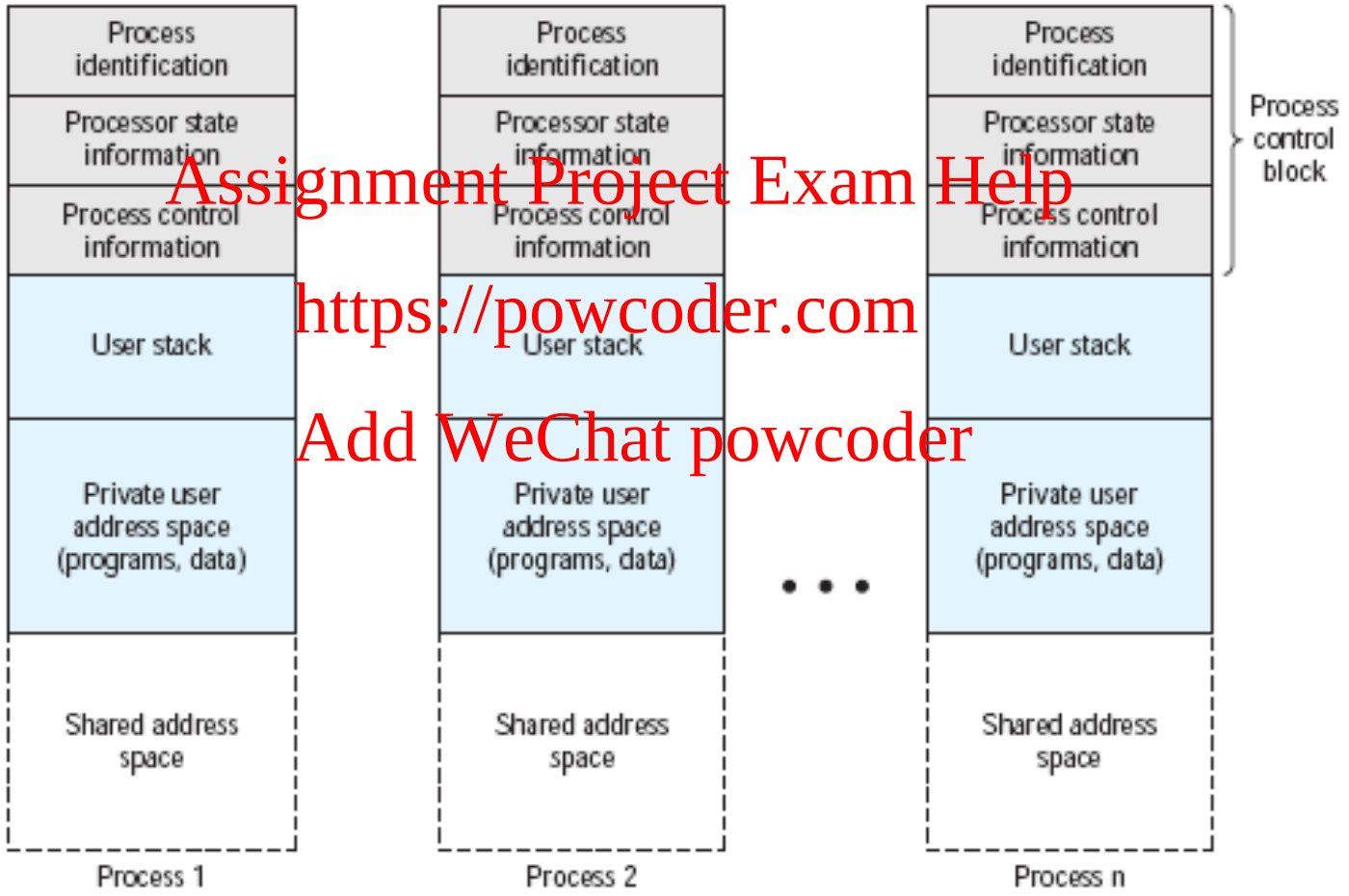


Figure 3.13 User Processes in Virtual Memory

Operating System Control Structures

- For the OS to manage processes and resources, it must have information about the current status of each process and resource
- Tables are constructed for each entity (memory, I/O and files) the operating system manages

OS Control Tables

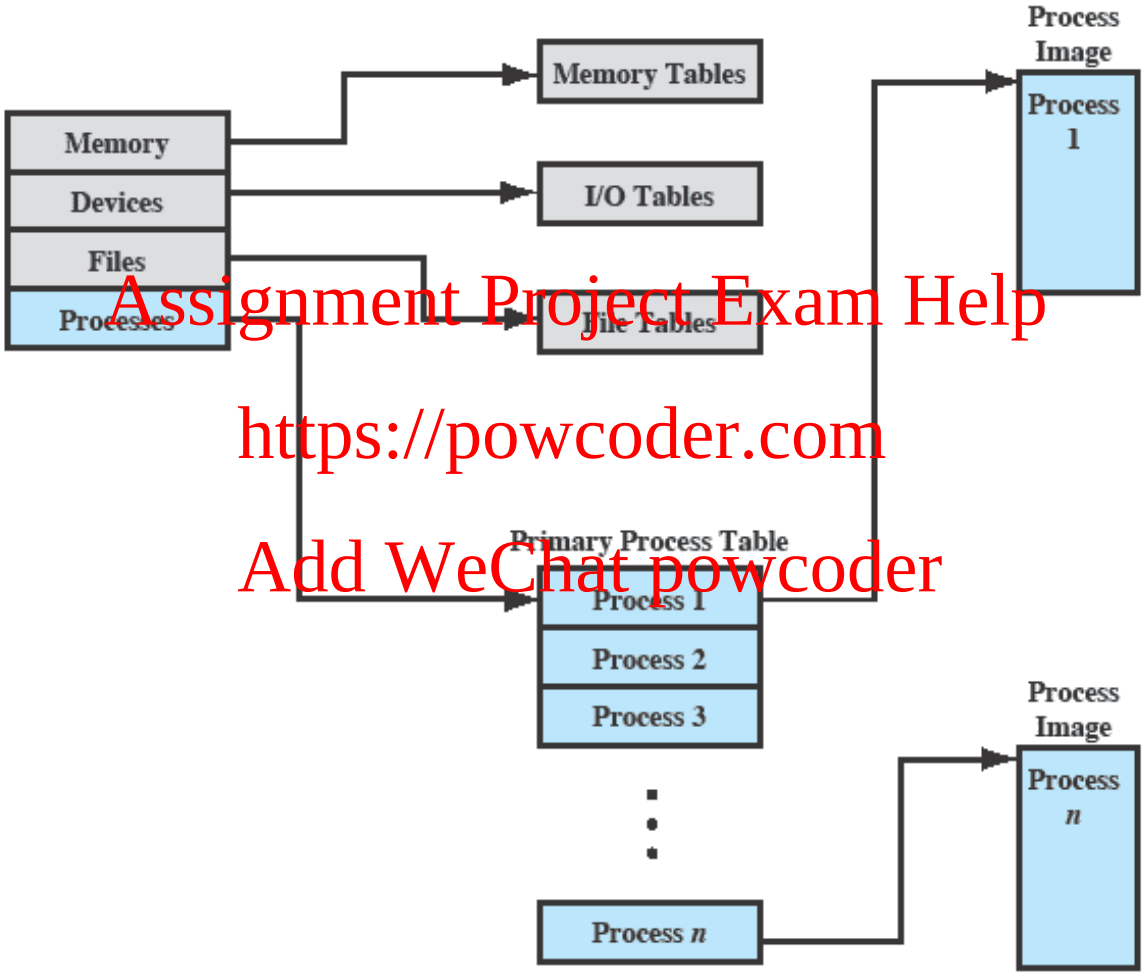
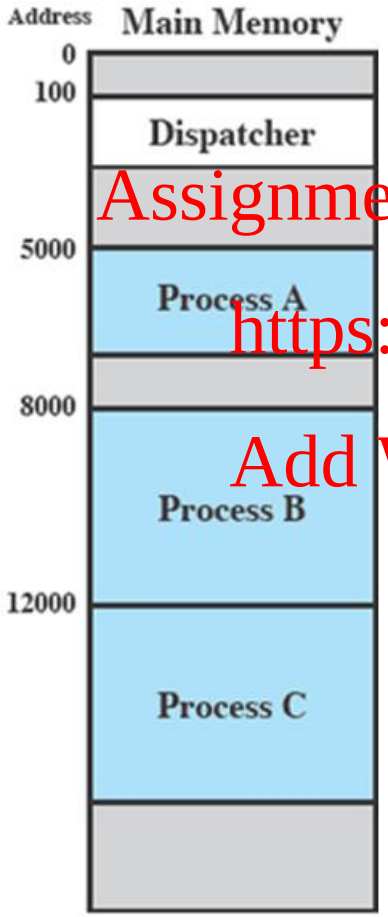


Figure 3.11 General Structure of Operating System Control Tables

Process Tables

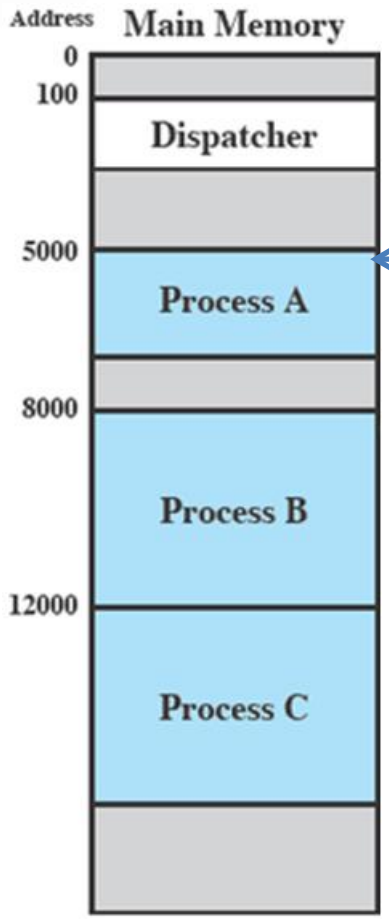
- Must be maintained to manage processes
- The OS tables must be linked or cross-referenced
  - Memory, I/O and files are managed on behalf of processes, there are some references to these resources, directly or indirectly , in the process tables.

Process Execution



- ***Dispatcher*** is a small program which switches the processor from one process to another
- Consider three processes being executed
- All are in memory (plus the dispatcher)

Trace from Processors point of view



1	5000
2	5001
3	5002
4	5003
5	5004
6	5005
	Timeout
7	100
8	101
9	102
10	103
11	104
12	105
13	8000
14	8001
15	8002
16	8003
	I/O Request
17	100
18	101
19	102
20	103
21	104
22	105
23	12000
24	12001
25	12002
26	12003

27	12004
28	12005
	Timeout
29	100
30	101
31	102
32	103
33	104
34	105
35	5006
36	5007
37	5008
38	5009
39	5010
40	5011
	Timeout
41	100
42	101
43	102
44	103
45	104
46	105
47	12006
48	12007
49	12008
50	12009
51	12010
52	12011
	Timeout

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

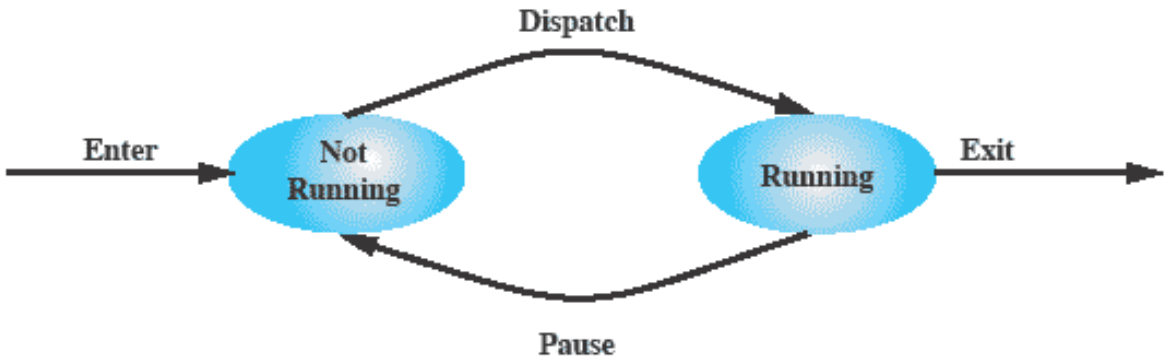
Figure 3.4 Combined Trace of Processes of Figure 3.2

Modes of Execution

- All modern processors support at least two modes of execution
- User mode
  - Less-privileged mode
  - User programs typically execute in this mode
- System (or kernel) mode
  - More-privileged mode
  - Kernel of the operating system

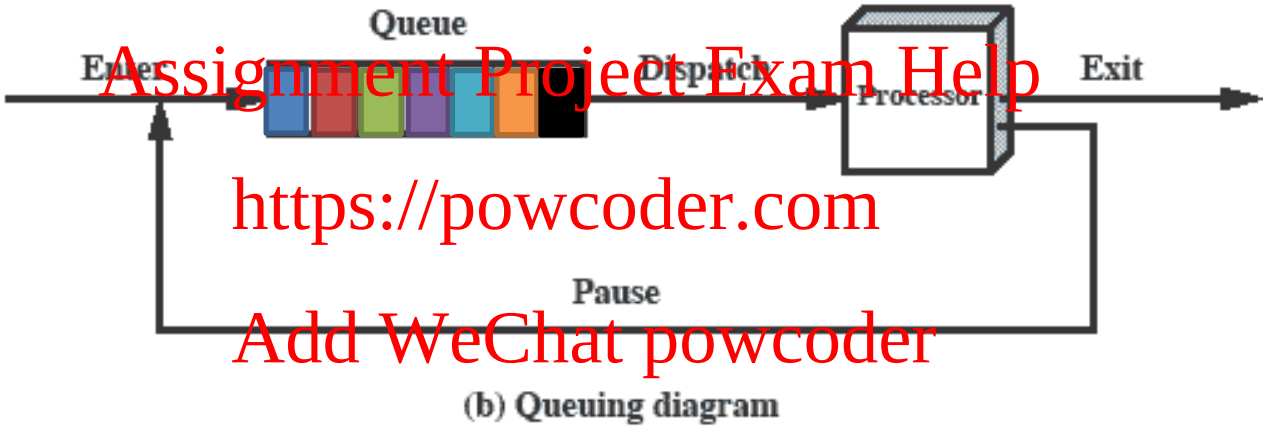
Two-State Process Model

- The state of a process may be defined by the current activity of that process
  - Used to describe the behaviour that we would like each process to exhibit
- Process may be in one of two states
  - Running
  - Not-running



(a) State transition diagram

Queuing Diagram



(b) Queuing diagram

Etc … processes moved by the dispatcher of the OS to the CPU then back to the queue until the task is completed

Five-State Process Model

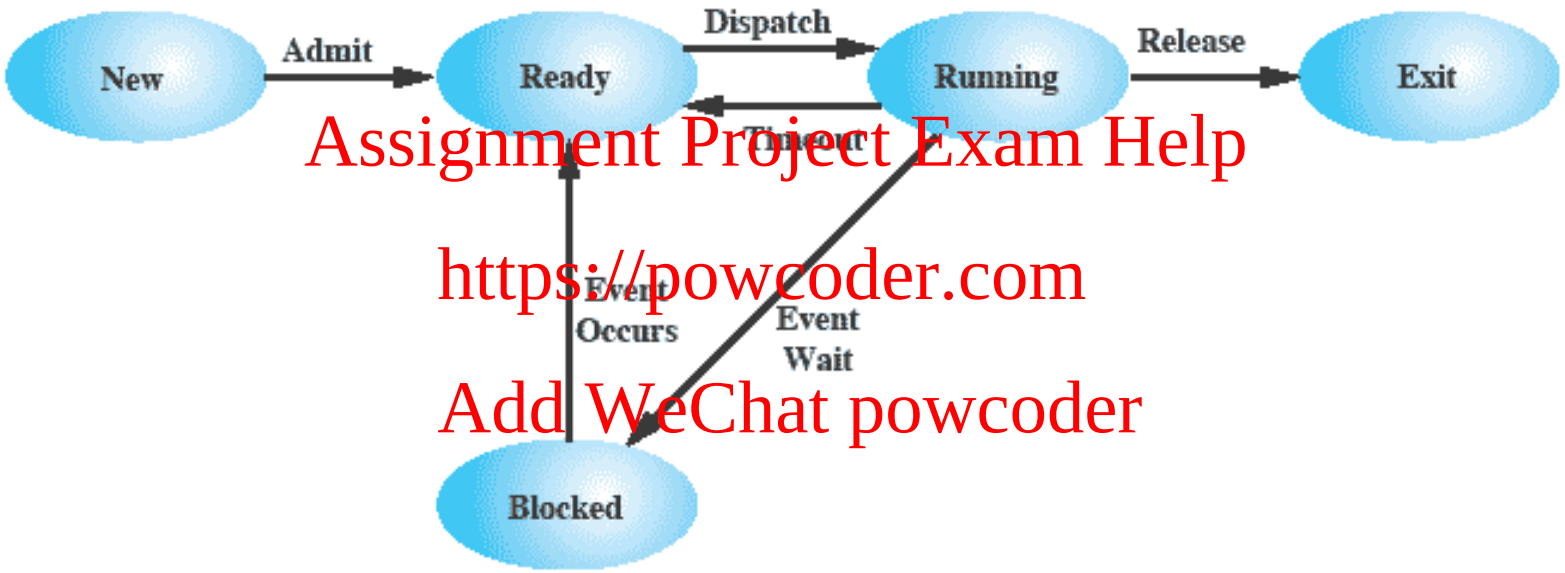
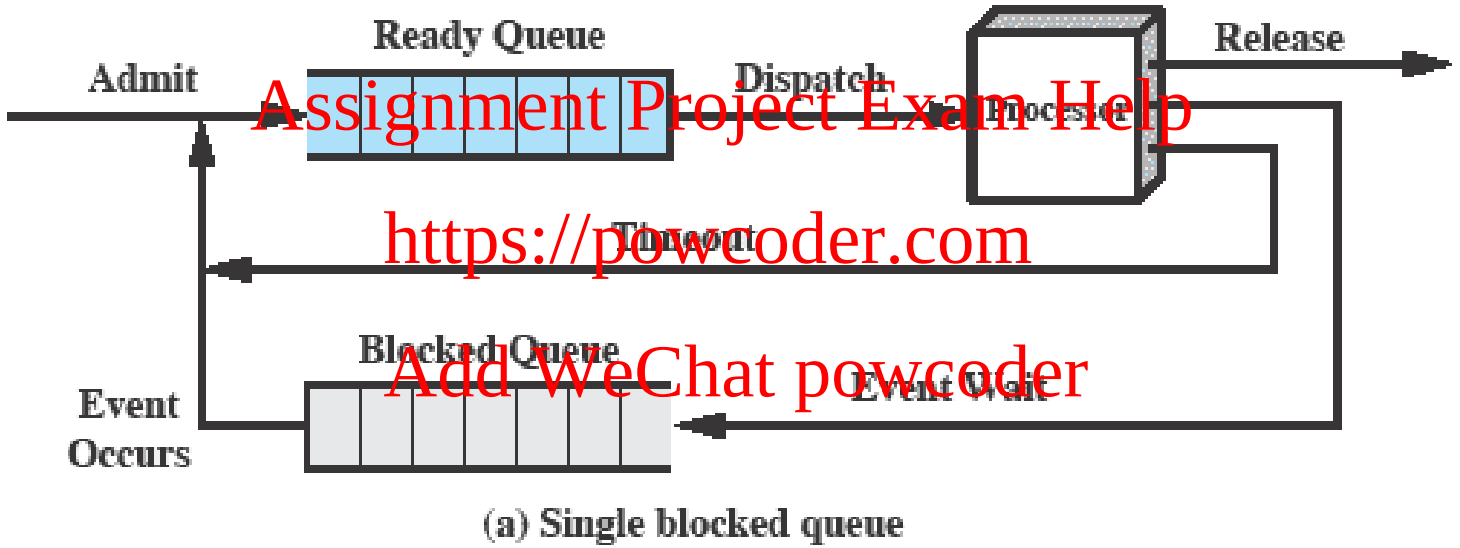


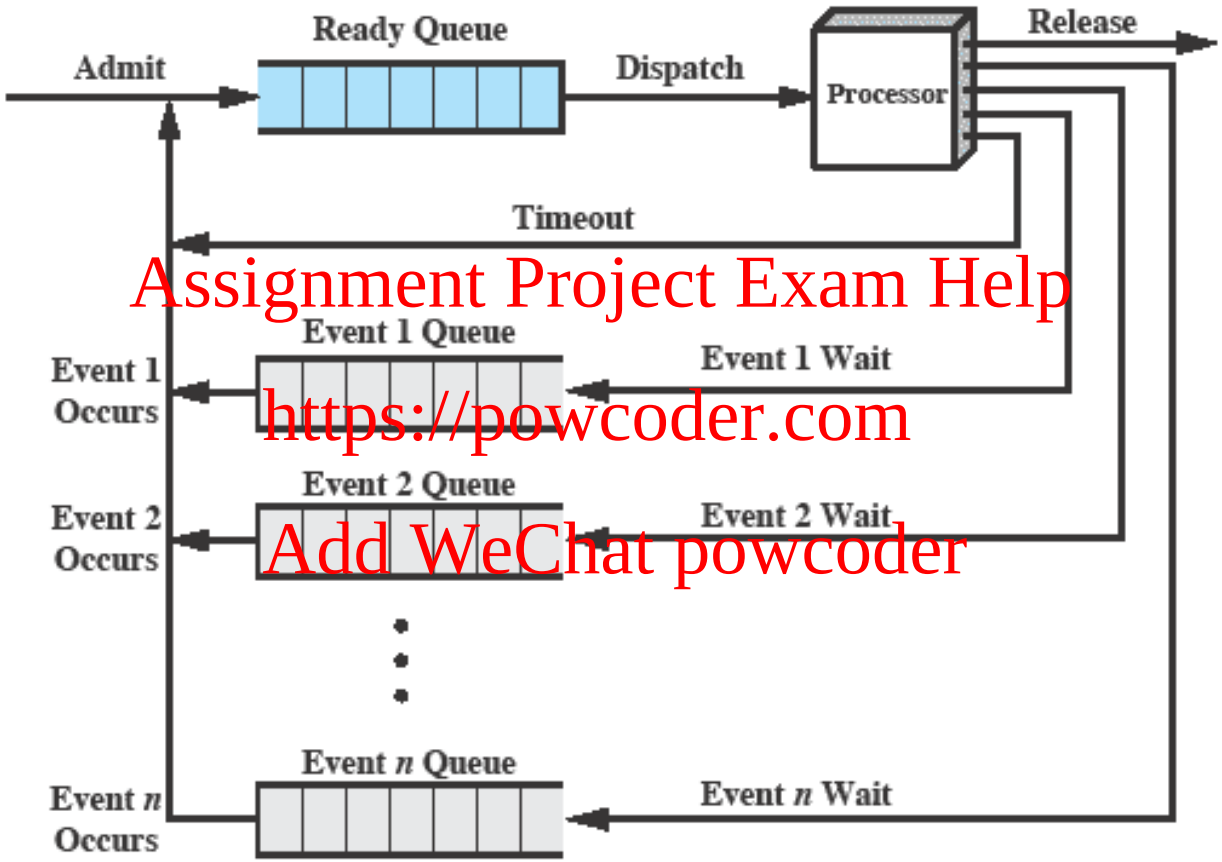
Figure 3.6 Five-State Process Model

Using Two Queues



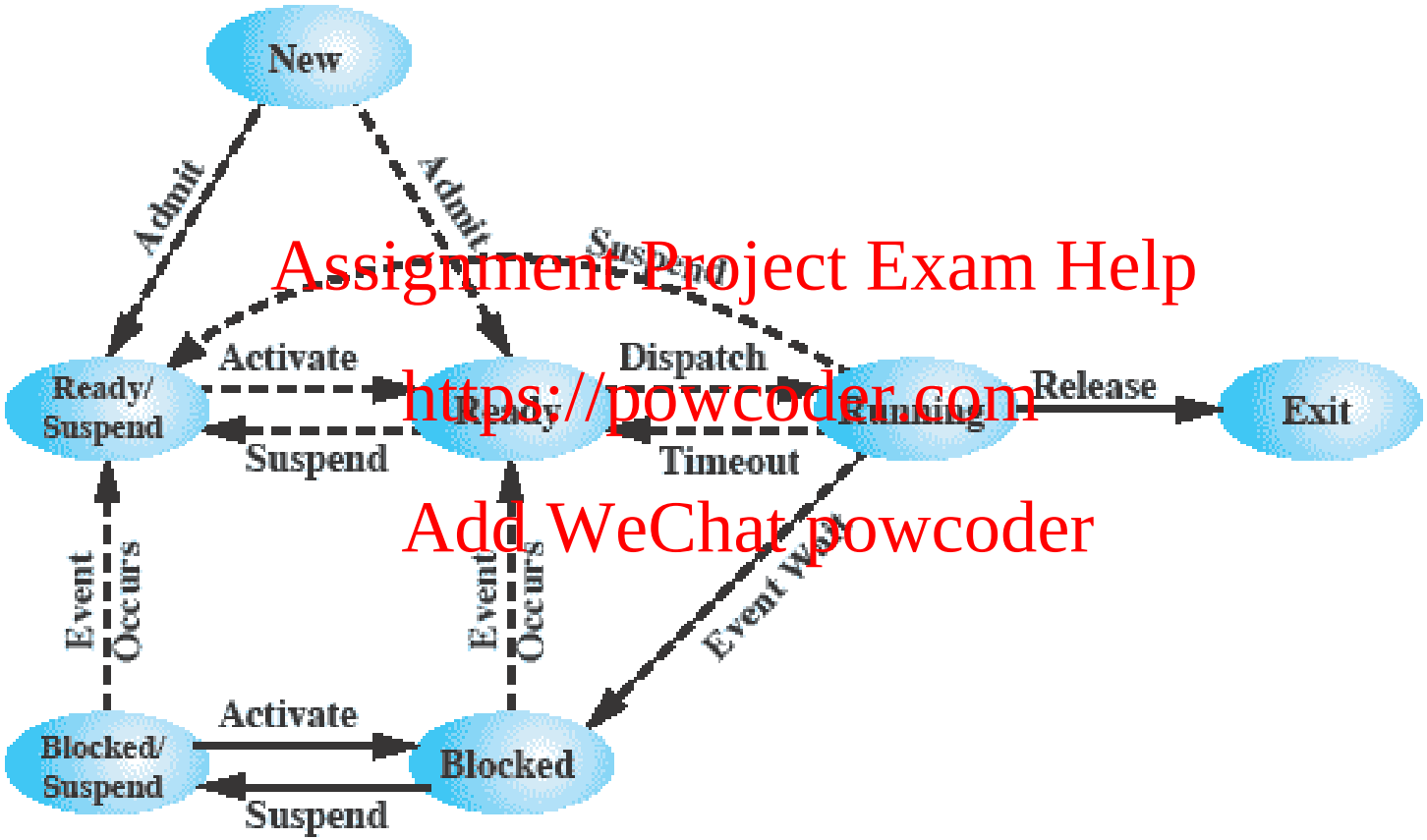
(a) Single blocked queue

Multiple Blocked Queues



(b) Multiple blocked queues

Adding Suspend States



(b) With Two Suspend States

Process List Structures

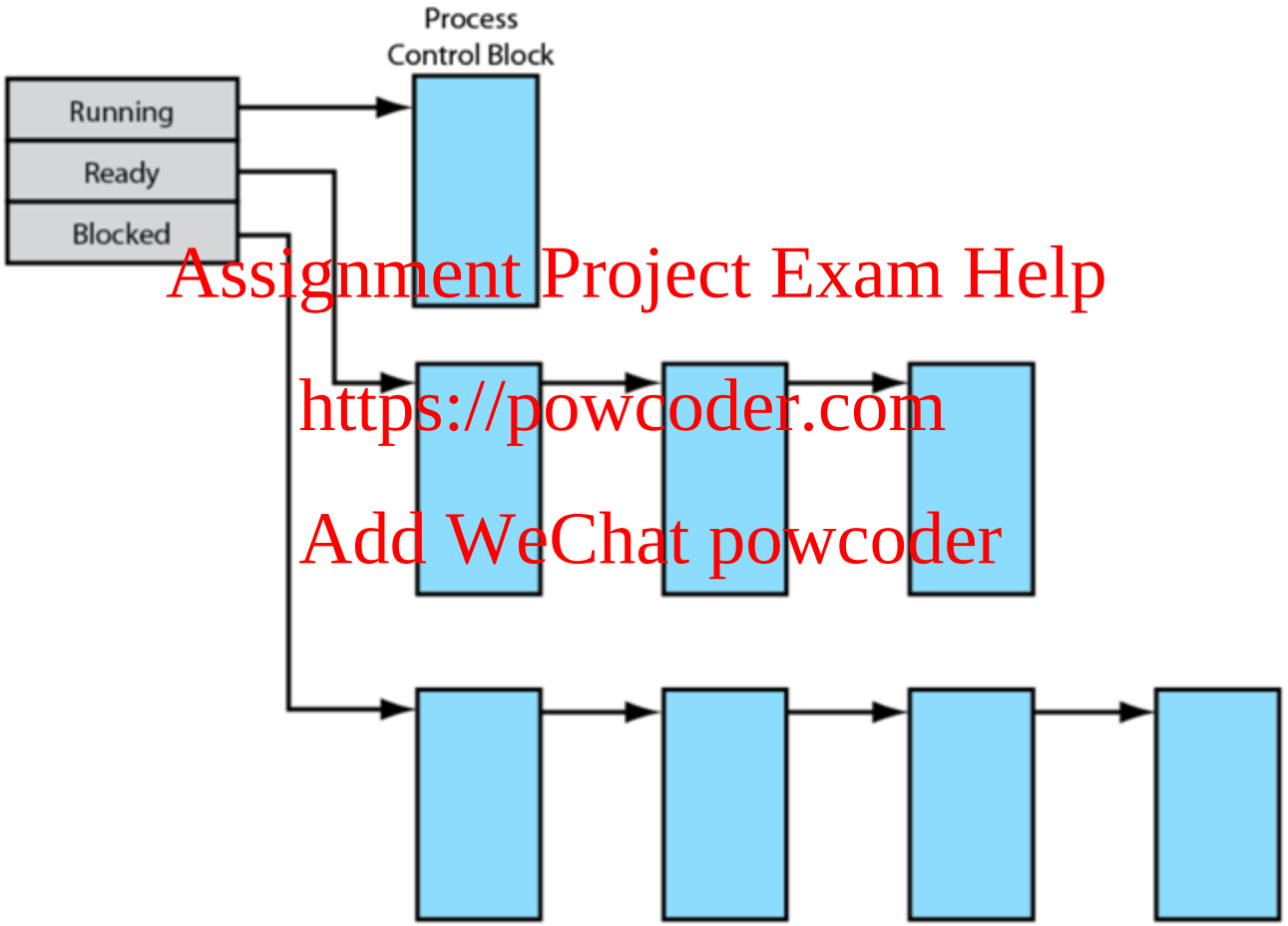


Figure 3.14 Process List Structures

Switching Processes

- Several design issues are raised regarding process switching
  - What events trigger a process switch?
  - What must the OS do to the various data structures under its control to achieve a process switch?

When to switch processes

A process switch may occur any time that the OS has gained control from the currently running process. Possible events giving OS control are:

Mechanism	Cause	Use
Interrupt	External to the execution of the current instruction	Reaction to an asynchronous external event
Trap	Associated with the execution of the current instruction	Handling of an error or an exception condition
Supervisor call	Explicit request	Call to an operating system function

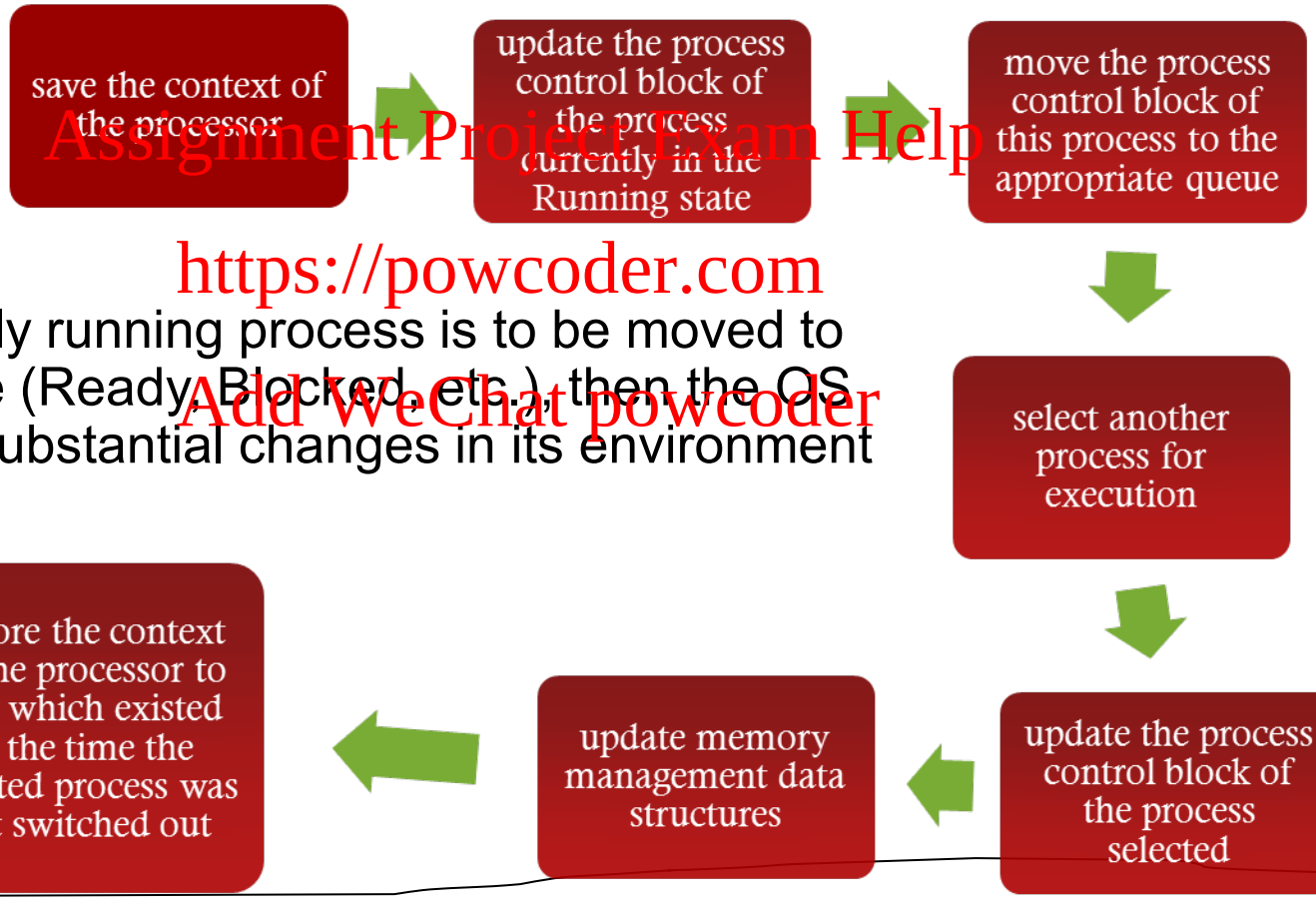
Table 3.8 Mechanisms for Interrupting the Execution of a Process

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
  - 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

Change of Process State

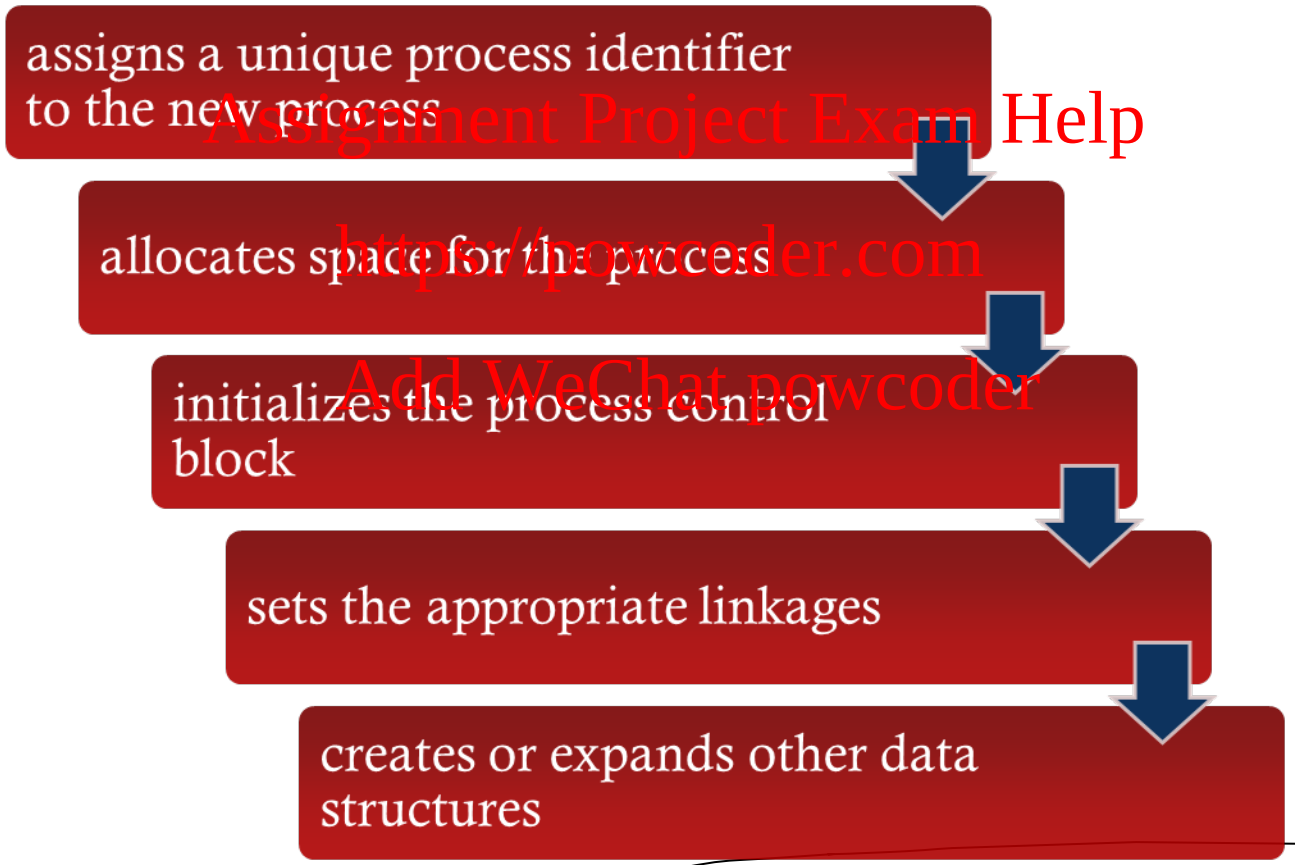
- The steps in a process switch are:



If the currently running process is to be moved to another state (Ready, Blocked, etc.), then the OS must make substantial changes in its environment

Process Creation

- Once the OS decides to create a new process it:

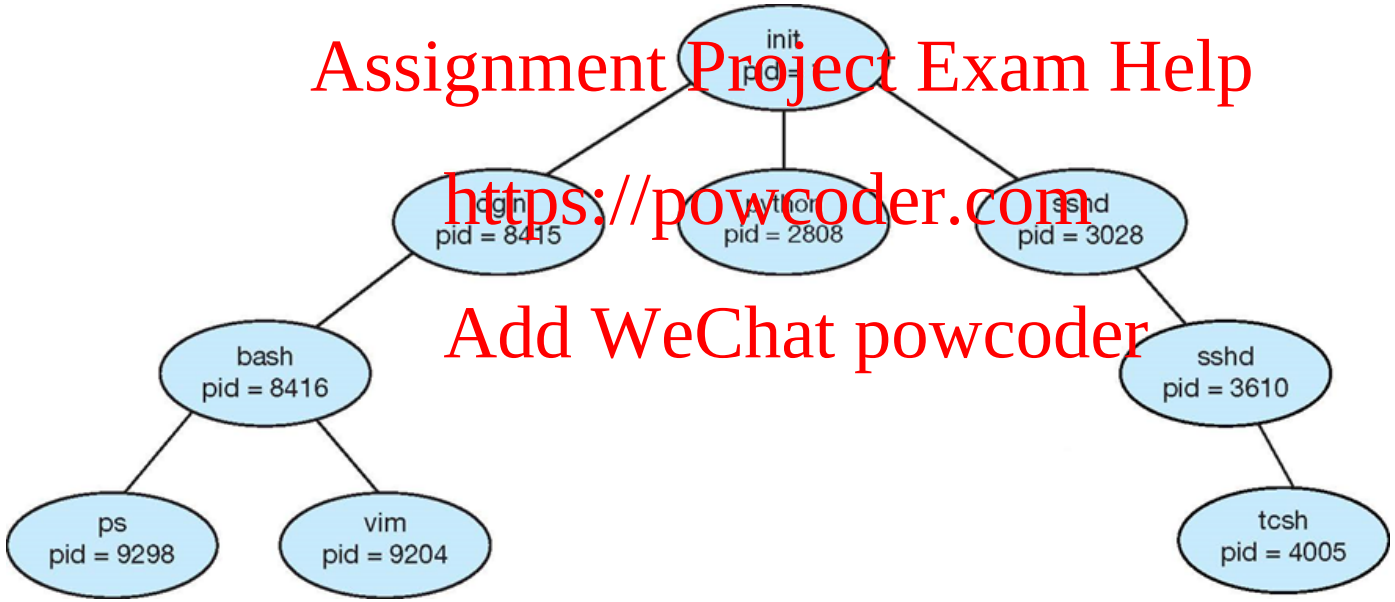


Process Creation (cont.)

- Traditionally, the OS created all processes
  - But it can be useful to let a running process create another
- This action is called ***process spawning***
  - ***Parent Process*** is the original, creating, process
  - ***Child Process*** is the new process
- Parent process create children processes, which, in turn create other processes, forming a tree of processes

Process Creation (cont.)

- A Tree of Processes in UNIX/Linux:



Process Creation (cont.)

- Generally, process identified and managed via **a process identifier** (**pid**)
- Resource sharing
  - Parent and children share all resources
  - Children share subset of parent's resources
  - Parent and child share no resources
- Execution
  - Parent and children execute concurrently
  - Parent waits until children terminate

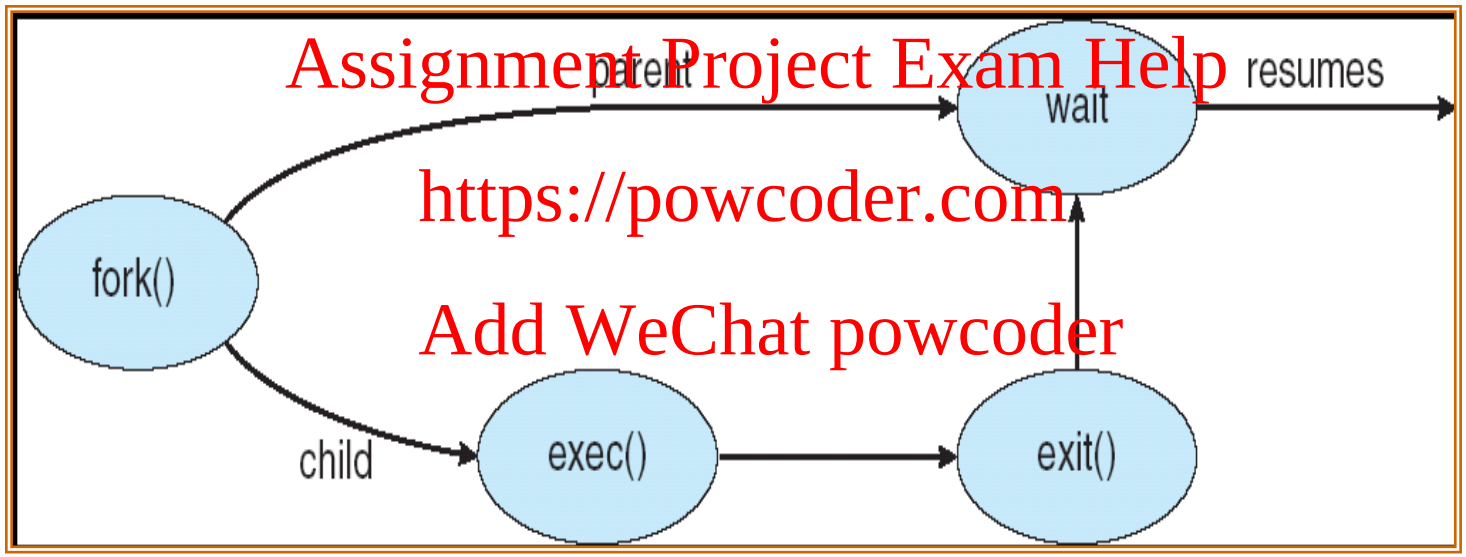
UNIX Process Creation

- Process creation is by means of the kernel system call, fork( ).
- This causes the OS, in Kernel Mode, to:
  1. Allocate a slot in the process table for the new process.
  2. Assign a unique process ID to the child process.
  3. Copy of process image of the parent, with the exception of any shared memory.
  4. Increment the counters for any files owned by the parent, to reflect that an additional process now also owns those files.
  5. Assign the child process to the Ready state.
  6. Returns the ID number of the child to the parent process, and a 0 value to the child process.

UNIX Process Creation (cont.)

- After creating the process the Kernel can do one of the following, as part of the dispatcher routine:
  - Stay in the parent process.
  - Transfer control to the child process
  - Transfer control to another process.

UNIX Process Creation (cont.)



Process Creation (cont.)

```
int main()
{
pid_t  pid;
    /* fork another process */
    pid = fork();
    if (pid < 0) { /* error occurred */
        fprintf(stderr, "Fork Failed");
        exit(-1);
    }
    else if (pid == 0) { /* child process */
        execlp("/bin/ls", "ls", NULL);
    }
    else { /* parent process */
        /* parent will wait for the child to complete */
        wait (NULL);
        printf ("Child Complete");
        exit(0);
    }
}
```


Process Termination

- 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 user action (e.g. log off, quitting an application)
  - A fault or error
  - Parent process terminating

Security Issues

- An OS associates a set of privileges with each process.
  - Highest level being administrator, supervisor, or root, access.
- A key security issue in the design of any OS is to prevent anything (user or process) from gaining unauthorized privileges on the system.
  - Especially - from gaining root access.

Summary

- The principal function of the OS is to create, manage, and terminate processes
- The most fundamental concept in a modern OS is the process
- 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