# CHAPTER 5

# Process Description and Control

# Process Description and Control

- All multiprogramming OS are build around the concept of processes
- A process is sometimes called a task

# Major Requirements of an OS

- OS must interleave the execution of several processes to maximize processor usage while providing reasonable response time.
- OS must allocate resources to processes while avoiding deadlock.- when a process A enters a waiting state because a resource requested is being held by Process B, which in turn is waiting resource held by process A.
- OS must support inter process communication and user creation of processes.

# Process

- A process is created for a program to be executed.
- Also called a task
- Execution of an individual program Involves a sequence of instruction within that program.
- Can characterize the behavior of a process by listing the sequence (trace of the process) of instructions that execute for that process.

# Process Trace

- Figure 3.1 shows a memory layout of three processes.
- Assume no used of virtual memory.
- All of the process are fully loaded in the main memory.
- There is dispatcher program that switches the processor from one process to another.

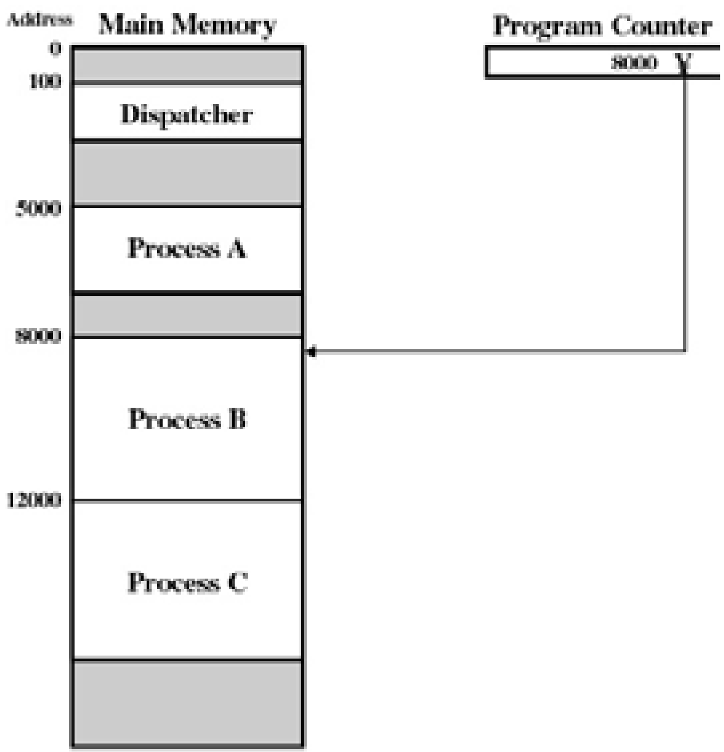



Figure 3.1 Snapshot of Example Execution (Figure 3.3) at Instruction Cycle 13

# Process Trace..

- Figure 3.2 shows the trace of three processes during early part of their execution.
- First 12 instructions executed in processes A and C are shown.
- Process B executes four instructions and assume that the fourth instruction invokes and I/O operation for which the process must wait.

| | | |
|---|---|---|
| 5000 | 8000 | 12000 |
| 5001 | 8001 | 12001 |
| 5002 | 8002 | 12002 |
| 5003 | 8003 | 12003 |
| 5004 | | 12004 |
| 5005 | | 12005 |
| 5006 | | 12006 |
| 5007 | | 12007 |
| 5008 | | 12008 |
| 5009 | | 12009 |
| 5010 | | 12010 |
| 5011 | | 12011 |
| **(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 3.2 Traces of Processes of Figure 3.1**

# Process Trace..

- Figure 3.3 shows traces from processor's point of view.
- It shows the interleaved traces resulting from the 52 instruction cycles.
- Assume OS allowed a process to continue execution for a maximum of 6 instruction cycles, then the process will be interrupted, thus its prevent single process from monopolizing processor

# Process Trace..

- The first 6 instructions of process A are executed, followed by a time-out and execution of some code in the dispatcher, which execute 6 instructions before turning control to process B
- After 4 instructions are executed, process B request and I/O action for which it must wait.

# Process Trace..

- The processor stops executing process B and moves on to process C.
- After time-out, the processor moves back to process A.
- When the process times out, process B is still waiting for I/O operation to complete, so the dispatcher moves on to process C again.

| | |
|---|---|
| 1 | 5000 |
| 2 | 5001 |
| 3 | 5002 |
| 4 | 5003 |
| 5 | 5004 |
| 6 | 5005 |
| ------------- Time out | |
| 7 | 100 |
| 8 | 101 |
| 9 | 102 |
| 10 | 103 |
| 11 | 104 |
| 18 | 101 |
| 19 | 102 |
| 20 | 103 |
| 21 | 104 |
| 22 | 105 |
| 23 | 12000 |
| 24 | 12001 |
| 25 | 12002 |
| 26 | 12003 |

| | |
|---|---|
| 27 | 12004 |
| 28 | 12005 |
| ------------- Time out | |
| 29 | 100 |
| 30 | 101 |
| 31 | 102 |
| 32 | 103 |
| 33 | 104 |
| 34 | 105 |
| 35 | 5006 |
| 36 | 5007 |
| 37 | 5008 |
| 44 | 103 |
| 45 | 104 |
| 46 | 105 |
| 47 | 12006 |
| 48 | 12007 |
| 49 | 12008 |
| 50 | 12009 |
| 51 | 12010 |
| 52 | 12011 |
| ------------- Time out | |

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 3.3 Combined Trace of Processes of Figure 3.1**

# Dispatcher

- 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.
- The CPU will always execute instructions from the dispatcher while switching from process A to process B

# Process Creation

- Reasons for process creation:
  - Submission of a batch job
  - User logs on
  - Created to provide a service such as printing (ex: printing a file).
  - Process creates another process (Process Spawning)

# Process Termination

- General reasons for process termination:
  - Batch job issues *Halt* instruction
  - User logs off
  - Process executes a service request to terminate (Quit an application)
  - Parent process terminate
  - Parent ask to terminate the child
  - Error and fault conditions

# Process Termination..

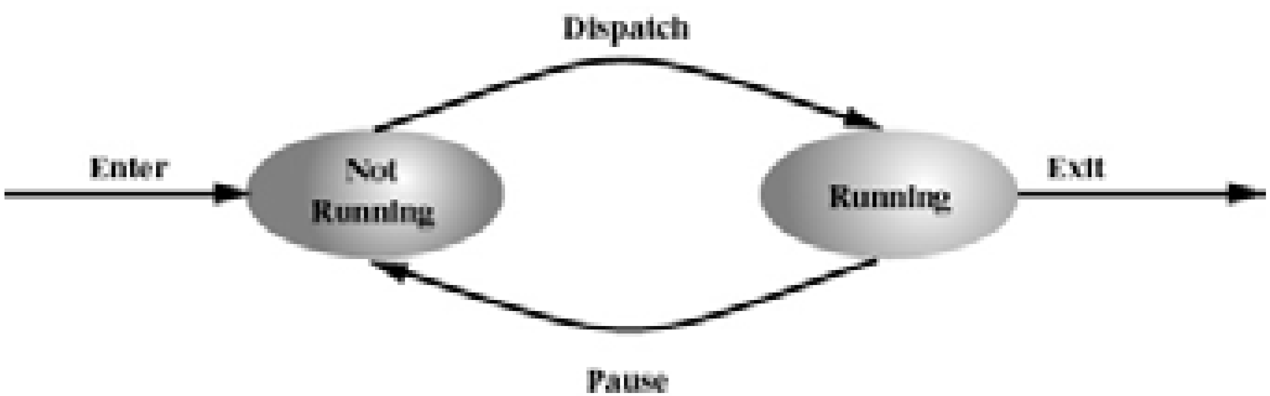
- Reasons for error and fault condition :
  - Time limit exceeded
  - Memory unavailable
  - Bounds violation -attempted access of (non-existent) 11th element of a 10-element array
  - Protection error
    - example write to read-only file
  - Arithmetic error - attempted division by zero

# Process Termination..

- Time overrun -process waited longer than a specified maximum for an event
- I/O failure
- Invalid instruction - when a process tries to execute data (text)
- Privileged instruction - defined as the delegation of authority over a computer system. A privilege is a permission to perform an action. Examples of various privileges include the ability to create a file in a directory, or to read or delete a file etc.
- Data misuse
- Operating system intervention - to resolve a deadlock

# Two-State Process Model

- Process may be in one of two states
  - Running
  - Not-running




(a) State transition diagram

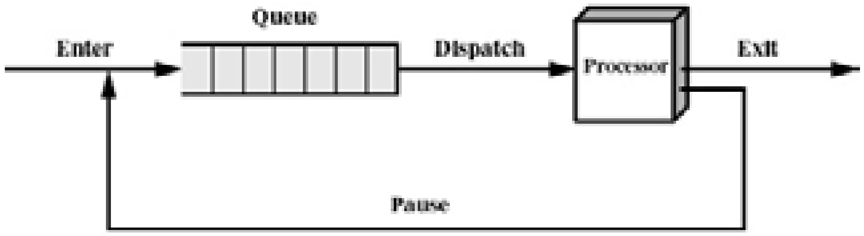
# Two-State Process Model..

- When OS creates a new process, it enters that process into Not Running State.
- The existence of the process is known by the OS and is waiting for an opportunity to execute.
- Running process will be interrupted from time to time and dispatcher will select a new process to run.
- Process will moves from the Running state to Not Running state, another process will moves to the Running state.

# Two-State Process Model..

- Each process must be represented in some way so that OS can keep track of it.
  - There must be some information relating to each process, including current state and location of main memory.
- Process that are not running must be kept in some sort of queue, waiting their turn to be execute.
- Figure 3.4b suggests a structure to deploy two state process model.

# Not-Running Process in a Queue




(b) Queuing diagram

# Two-State Process Model..

- There is a single queue in which each entry is a pointer to a particular process.
- The queue must consist of linked list of data blocks, in which each block represents one process.
- The queue is first in first out (FIFO) list and the processor operates in Round robin.

# Process

- Queuing suggested in Figure 3.4b will be effective if all processes were always ready to execute.
- **BUT** it is inadequate because some process that are in Not Running state either:
  - ready to execute
  - blocked because of waiting for I/O operation complete.

# Process..

- By using queuing on fig. 3.4b, the dispatcher has to scan the queue looking for the process that is not blocked and has been in queue the longest.
- A way to tackle this situation is to split the Not Running state into two different states which are:
  - **Ready** state: Ready to execute
  - **Blocked** state: waiting for I/O
- Now, instead of two states we have three states

  →**Ready, Running, Blocked**

# Process..

- For a good measure, there are another two additional states that will be useful for process management:
- **New** state:
- OS performed the necessary actions to create the process
  - Process ID
  - Tables needed to manage the process
- but has not yet committed to execute the process (not yet admitted)
  - because resources are limited

# Process..

- **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
  - Ex: accounting program that cumulates resource usage for billing the users
- The process (and its tables) gets deleted when the data is no more needed.

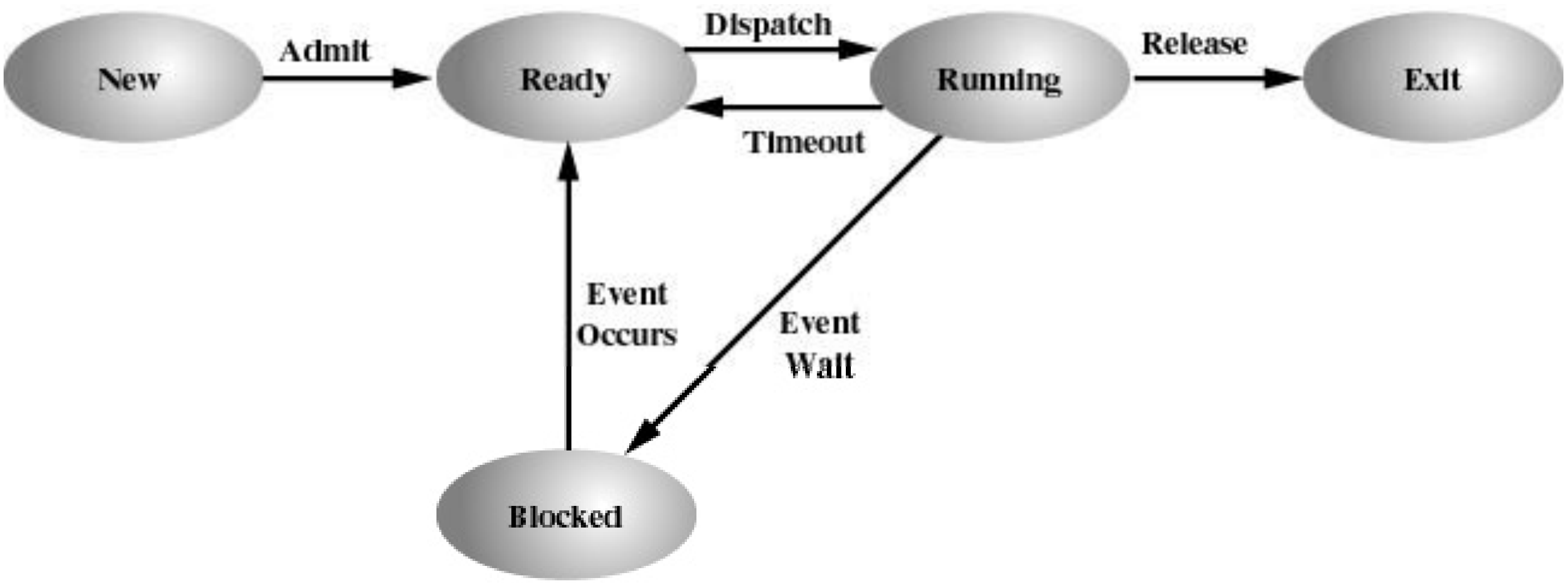



Figure 3.5 Five-State Process Model

http://courses.cs.vt.edu/csonline/OS/Lessons/Processes/index.html

# A Five-State Model

- Running → the process that is currently being executed.
- Ready → a process that is prepared to execute when given the opportunity.
- Blocked → a process that cannot execute until some event occurs, such as the completion of I/O operation.
- New → a process that has just been created but not being admitted to the pool of executable process by the OS (not being loaded in the main memory).
- Exit → a process that has been released from the pool of executable processes by the OS, either because it halted or aborted for some reason.

# Process Transitions

- Figure 3.5 indicates the possible state transition as follows:
- Null → New
  - A new process is created to execute a program.
- New → Ready
  - OS will move the process from New to Ready state when it is prepared to take an additional process.
- Ready → Running
  - When it is time, the dispatcher selects a new process to run

# Process Transitions..

- Running → Exit
  - The currently Running process is terminated by the OS if the process indicates that it has completed or if it aborts.
- Running → Ready
  - the running process has expired his time slot
  - the running process gets interrupted because a higher priority process is in the ready state