Operating Systems W2L1 - Processes & Threads: Part 1

Class	Operating Systems
Created	@Sep 9, 2020 6:07 PM
Materials	02 - Processes and Threads Part 1.pdf
Reviewed	☐
Type	Lecture

All about the illusions given to a process by the OS for the amount of CU and memory available for a given program— process management and thread usage are the primary focuses of how the OS gives processes this illusion.

OS Management of Application Execution (Processes)

- All resources are made available to multiple programs running at once, otherwise programmers would have to account for multiple programs running
- Processor is switching between multiple applications, so all programs appear to be progressing to the human eye
    - "Multitasking" is just incredibly fast switching, done by the processor

▼ What is the main goal here?

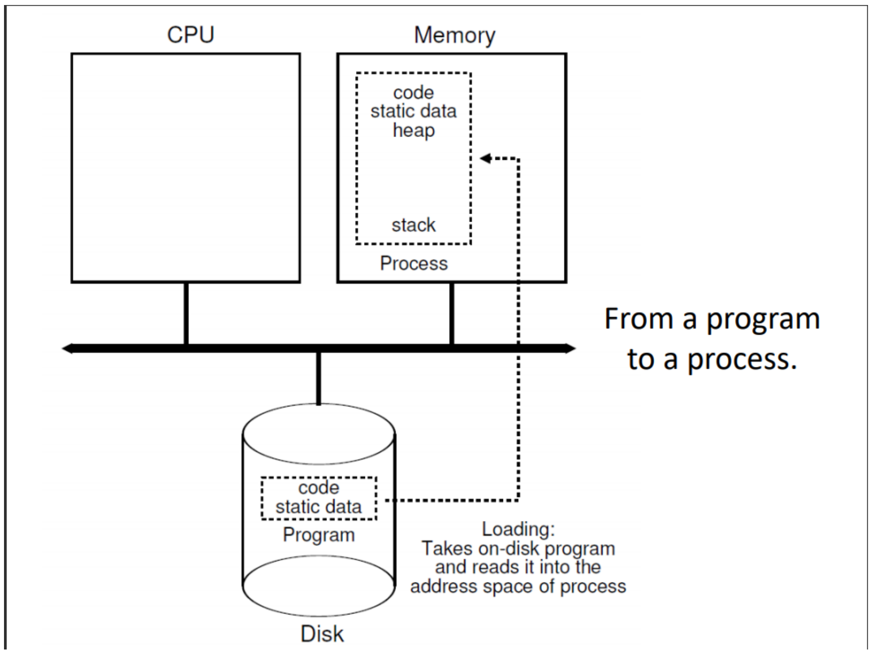
The processor, memory, and I/O devices need to be used efficiently. I/O is anything that isn't the processor or memory.

- Loader is the part of an OS that loads executable code into memory, addresses of instructions, etc.

▼ What defines a *process*?

A process is an *abstraction* of a *running program*

▼ Program → Process Diagram


CPU
Memory
code
static data
heap
stack
Process
From a program to a process.
code
static data
Program
Disk
Loading:
Takes on-disk program and reads it into the address space of process

- Abstraction is the illusion behind a process thinking it has all the resources it could ever need
- Nothing is sequential anymore; pretty much all computing is parallel

▼ What is a *program counter*?

The program counter, stored in register `%rip`, stores the address of the instruction currently being executed.

Process Model

▼ What four things does a process have?

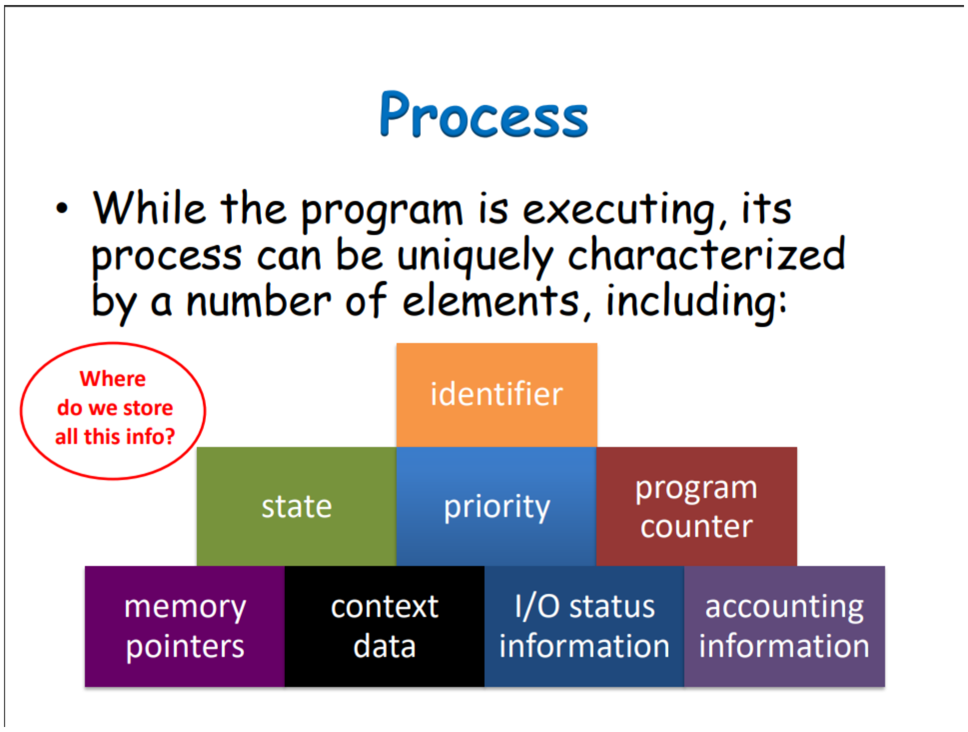
1. Program (is a *running* program)
2. Input

3. Output

4. State

▼ Processes are uniquely characterzed by a number of elements

All of this information is needed by the OS, not hte process, so these are stored in the **process control block**



Process

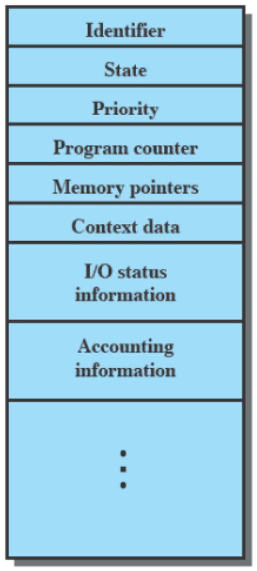
- While the program is executing, its process can be uniquely characterized by a number of elements, including:

Where do we store all this info?
identifier
state
priority
program counter
memory pointers
context data
I/O status information
accounting information

- A program running twice is two instances- two difference processes

▼ Process Control Block

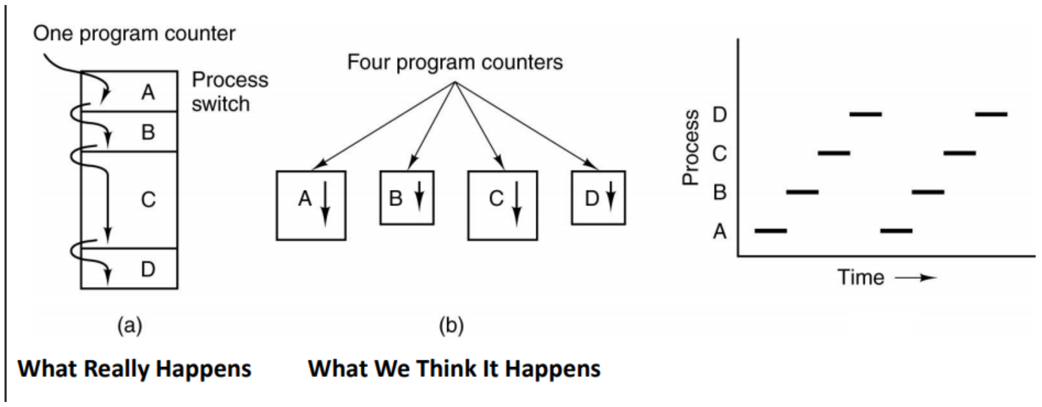
It is a data structure that the OS has for each process that is running. It is what allows support for multiple processes running at once, and also what makes it possible for a process to be *interrupted* and resumed at a later time.

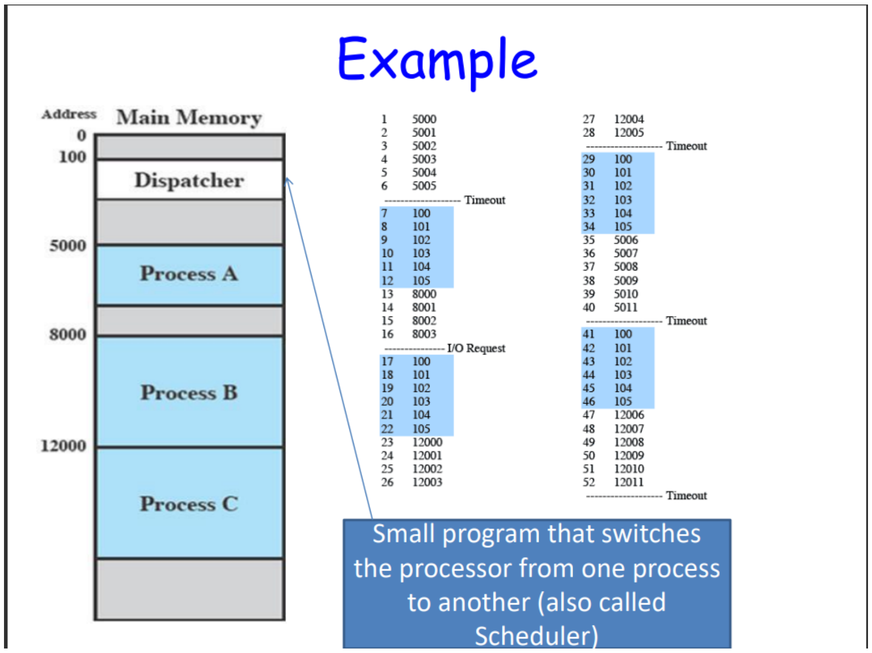


Multiprogramming

- WLOG, one CPU and several processes have the ability to swap back and forth

▼ The **scheduler** in an OS tells the CU when to switch and what address to execute next





Example

```
1    5000                        27   12004
2    5001                        28   12005
3    5002                        ------------------- Timeout
4    5003                        29   100
5    5004                        30   101
6    5005                        31   102
------------------- Timeout      32   103
7    100                         33   104
8    101                         34   105
9    102                         35   5006
10   103                         36   5007
11   104                         37   5008
12   105                         38   5009
13   8000                        39   5010
14   8001                        40   5011
15   8002                        ------------------- Timeout
16   8003                        41   100
--------------- I/O Request      42   101
17   100                         43   102
18   101                         44   103
19   102                         45   104
20   103                         46   105
21   104                         47   12006
22   105                         48   12007
23   12000                       49   12008
24   12001                       50   12009
25   12002                       51   12010
26   12003                       52   12011
                                 ------------------- Timeout
```

Small program that switches the processor from one process to another (also called Scheduler)

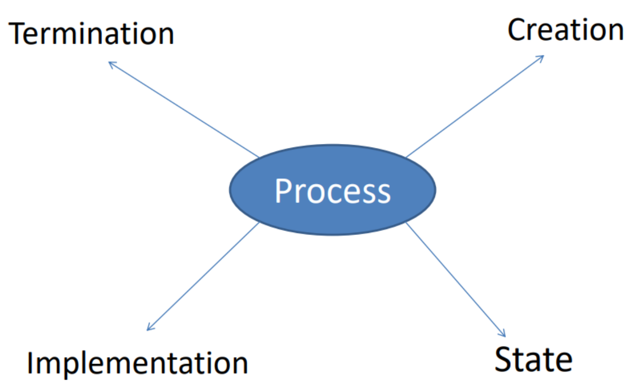
- The number of processes in memory depends on the size of memory
- *Dispatcher* is a piece of an OS like the loader is — Dispatcher is synonymous with scheduler
  - A dispatcher is always called from the very beginning of its set of instructions, whereas other processes can pick up where they left off

▼ Which process element stores last memory location?

The program counter — See the diagram under the scheduler (above) and note the memory address in blue and how others resume midway through

- It takes a while to switch processes because the OS has to decide where to go, find the address, start up, etc.
- A process can *NOT* resume by itself; cannot ask the OS for things, where the OS can resume and terminate a process

▼ Process web diagram



Process Creation and Termination

When Does Creation Happen?

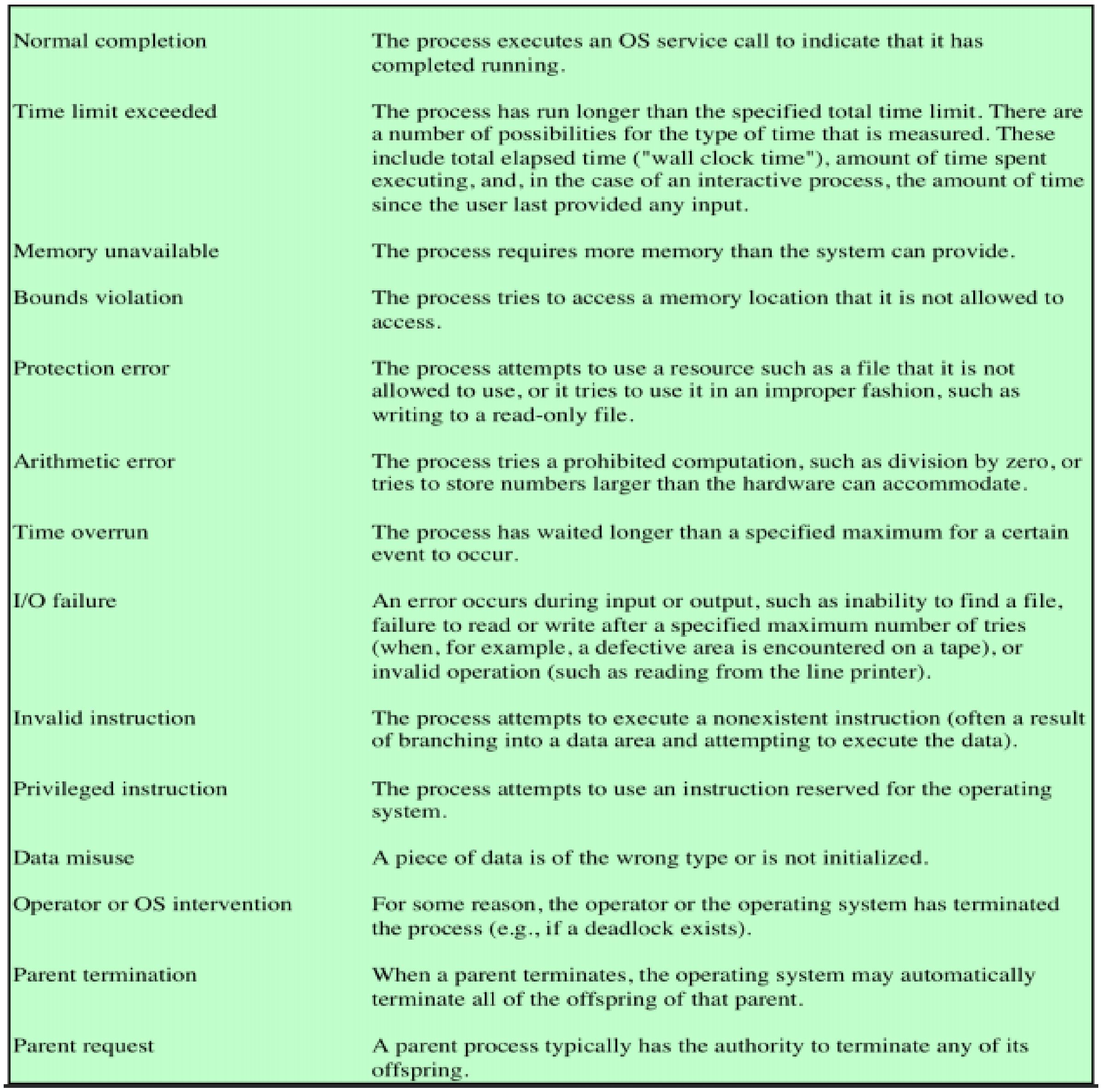
- At system initialization (boot time, foreground, background)
    - Background initializations/processes are referred to as *daemons*
- Execution of a process creation system called by a running process
    - i.e. a parent process calls a child process — forking processes
- At a user request, i.e. double clicking on an executable file
- A batch job → created by sysadmin or user, not OS
- Created by OS to provide a service
- Interactive logon, i.e. Linux terminal waiting for a command

When Does Termination Happen?

- Voluntary [by programmer]
    - Normal exit
    - Error exit → Conditional in code that is intentional, i.e. wrong password
- Involuntary [by programmer]

- Fata error
- Killed by another process

▼ Several termination scenarios



Normal completion	The process executes an OS service call to indicate that it has completed running.
Time limit exceeded	The process has run longer than the specified total time limit. There are a number of possibilities for the type of time that is measured. These include total elapsed time ("wall clock time"), amount of time spent executing, and, in the case of an interactive process, the amount of time since the user last provided any input.
Memory unavailable	The process requires more memory than the system can provide.
Bounds violation	The process tries to access a memory location that it is not allowed to access.
Protection error	The process attempts to use a resource such as a file that it is not allowed to use, or it tries to use it in an improper fashion, such as writing to a read-only file.
Arithmetic error	The process tries a prohibited computation, such as division by zero, or tries to store numbers larger than the hardware can accommodate.
Time overrun	The process has waited longer than a specified maximum for a certain event to occur.
I/O failure	An error occurs during input or output, such as inability to find a file, failure to read or write after a specified maximum number of tries (when, for example, a defective area is encountered on a tape), or invalid operation (such as reading from the line printer).
Invalid instruction	The process attempts to execute a nonexistent instruction (often a result of branching into a data area and attempting to execute the data).
Privileged instruction	The process attempts to use an instruction reserved for the operating system.
Data misuse	A piece of data is of the wrong type or is not initialized.
Operator or OS intervention	For some reason, the operator or the operating system has terminated the process (e.g., if a deadlock exists).
Parent termination	When a parent terminates, the operating system may automatically terminate all of the offspring of that parent.
Parent request	A parent process typically has the authority to terminate any of its offspring.

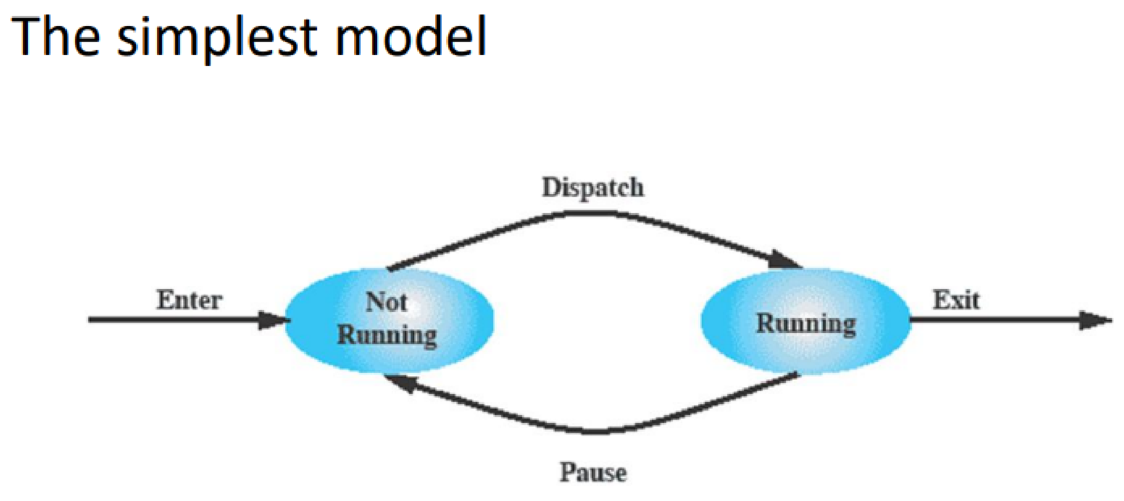
Process State

- Can be several possible *state models*
  - ▼ What are the various states found in various models?
    1. New

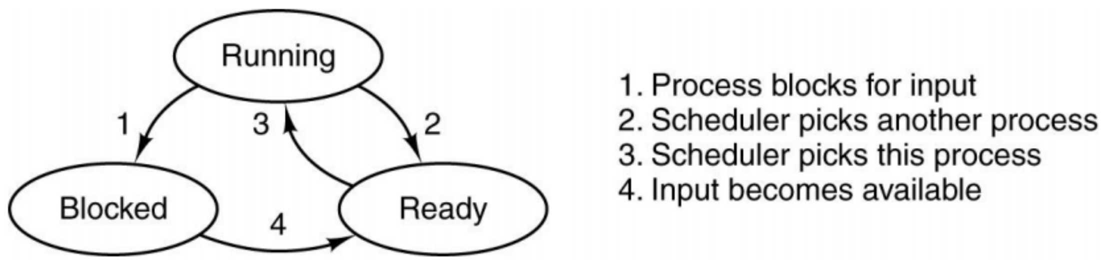
    2. Ready
    3. Running
    4. Blocked
    5. Exit

- Each state model shows…
  - A number of states
  - The type of each state
  - When to move from one state to another

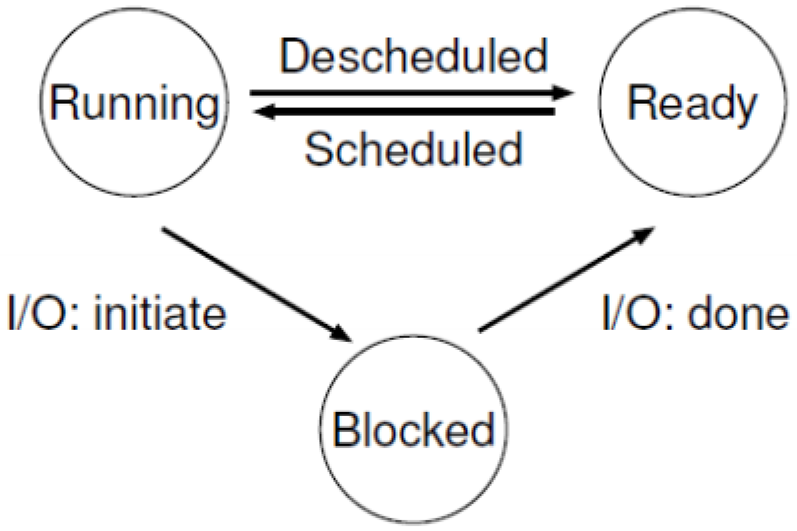
▼ Two-state model



▼ Three-state model

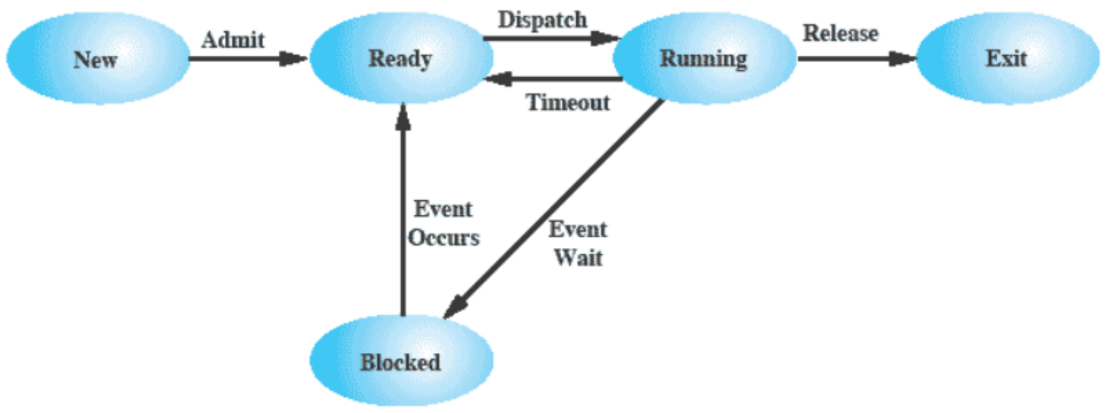


1. Process blocks for input
2. Scheduler picks another process
3. Scheduler picks this process
4. Input becomes available



▼ Five-state model

Often beneficial if you need to track, on the exit step, that other processes are waiting on a parent process to do the exiting. The new state is useful for knowing that a process is not yet loaded into memory.

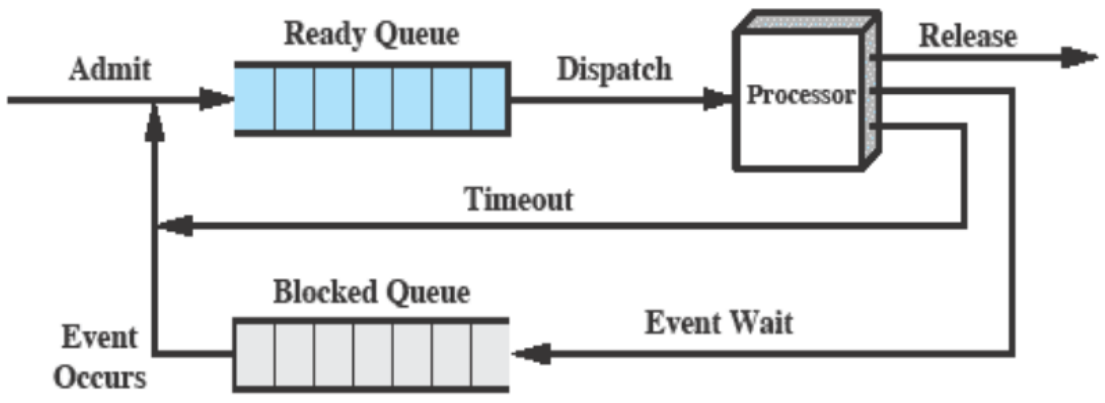


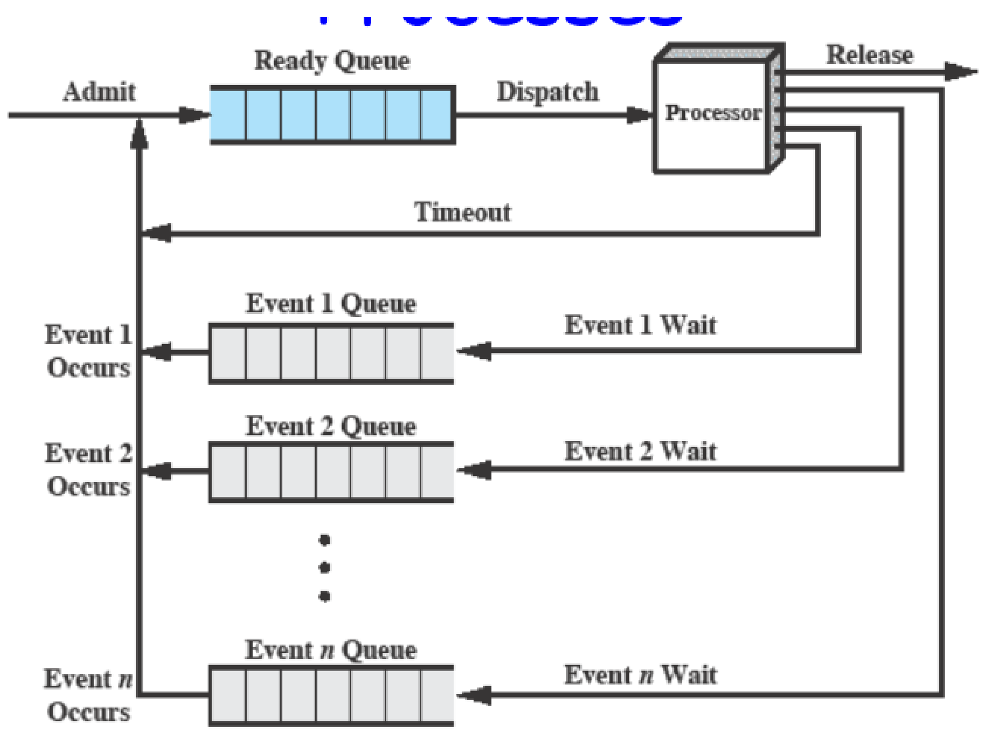
💡 The advantage to having more states is that you can switch between processes more efficiently. You have more information about the give status- or state- of a process.

- There's no direct advantage of five-state over three-state per se— it is largely context dependent.

Using Queues to Manage Processes

▼ Far-out view





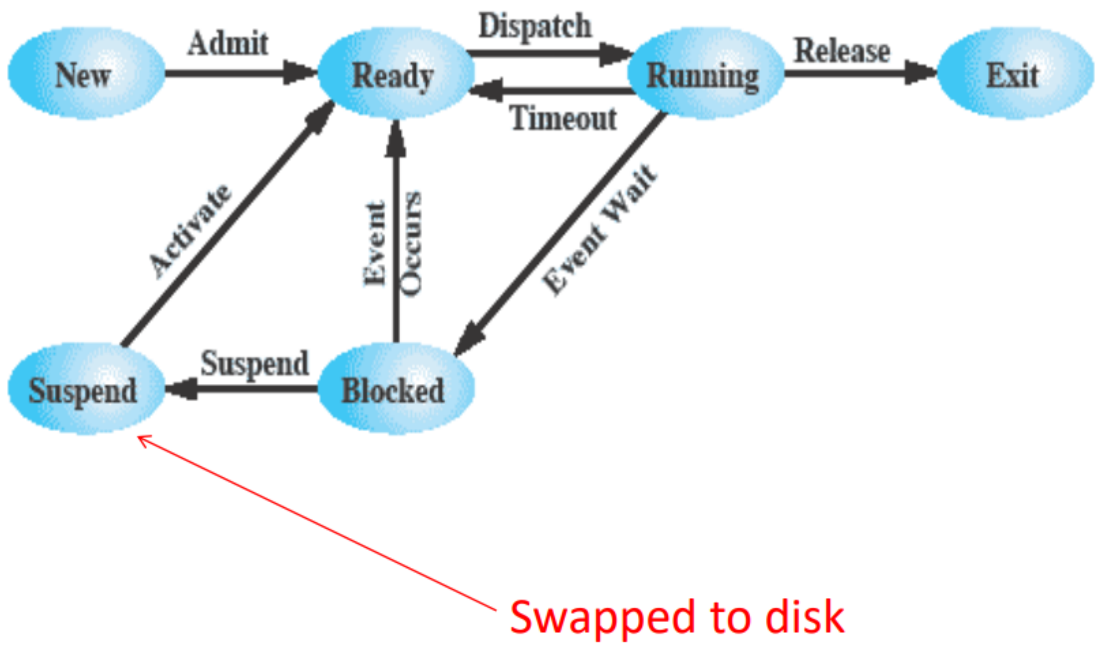
- OS can change the order of processes in a queue

  ▼ Why can an OS do this (hint: it's relevant to the data structure used for the queue)

    The process queue is often a LinkedList, so switching the overall order or just the organization of the processes is fairly straightforward (but not particularly easy)

- Very unlikely for priority to suddenly/dynamically switch

▼ One extra state — Blocked and suspended state

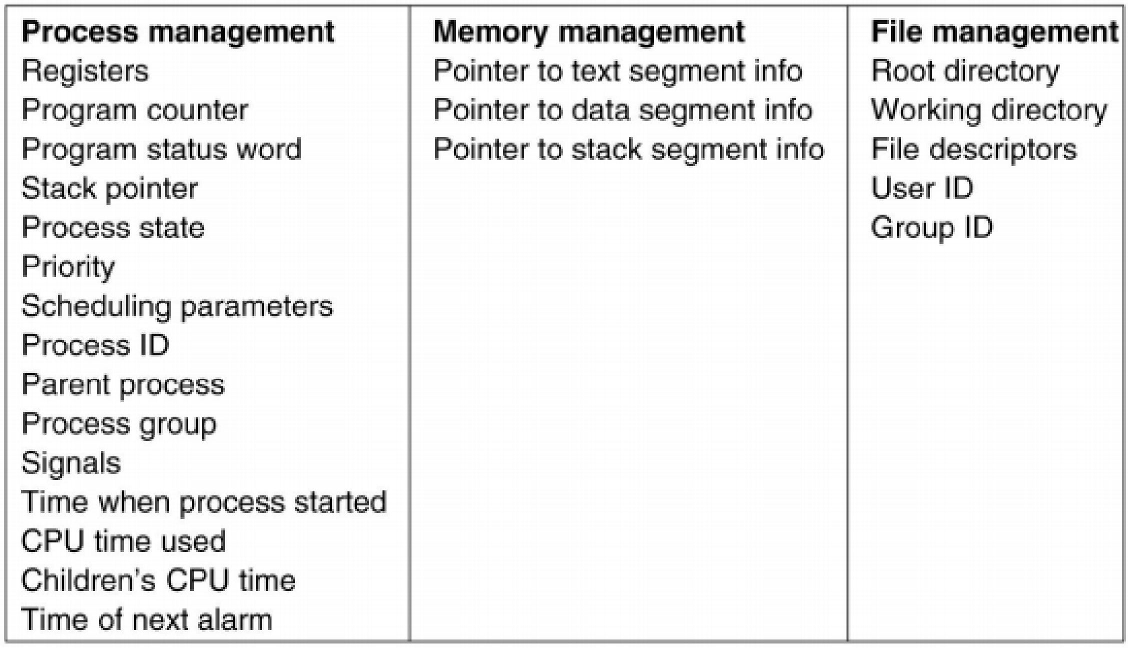


- When blocked you can't run anyway, so it's useful to know if a process is ready to resume

Implementation of Processes

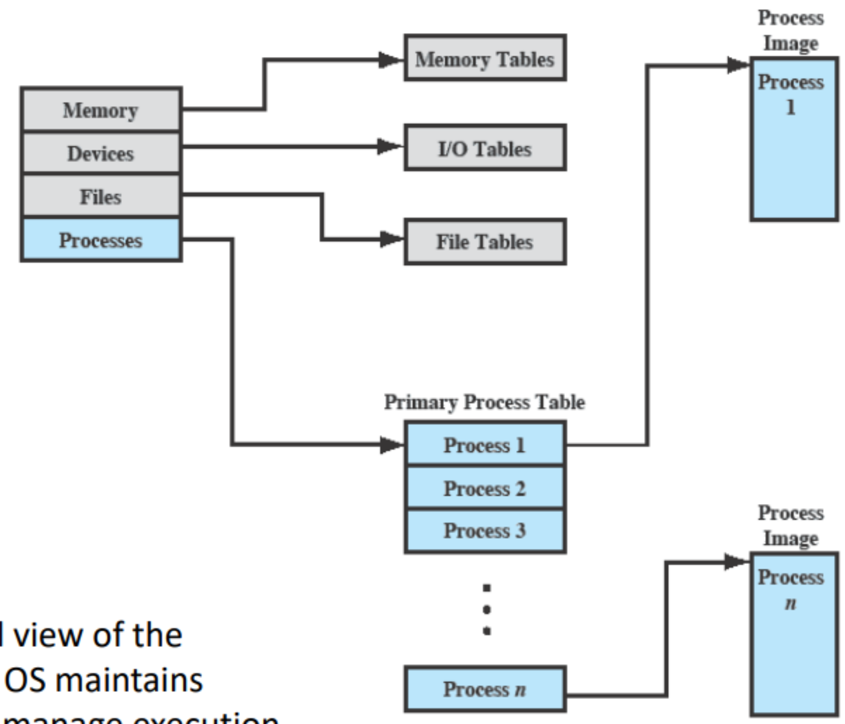
- OS maintains a *control table* (aka process table) whcih is an array of structures → one entry per process

  ▼ Some examples



Process management	Memory management	File management
Registers	Pointer to text segment info	Root directory
Program counter	Pointer to data segment info	Working directory
Program status word	Pointer to stack segment info	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		

▼ Conceptual view of tables maintained by the OS



Conceptual view of the tables that OS maintains in order to manage execution of processes on resources.

- Exact content and data structures vary from one OS to another, including different versions of the same OS
    - Generally unique to a given system

Ended on **A Bit About Interrupts** on the slides, presumably to be picked up next lecture.