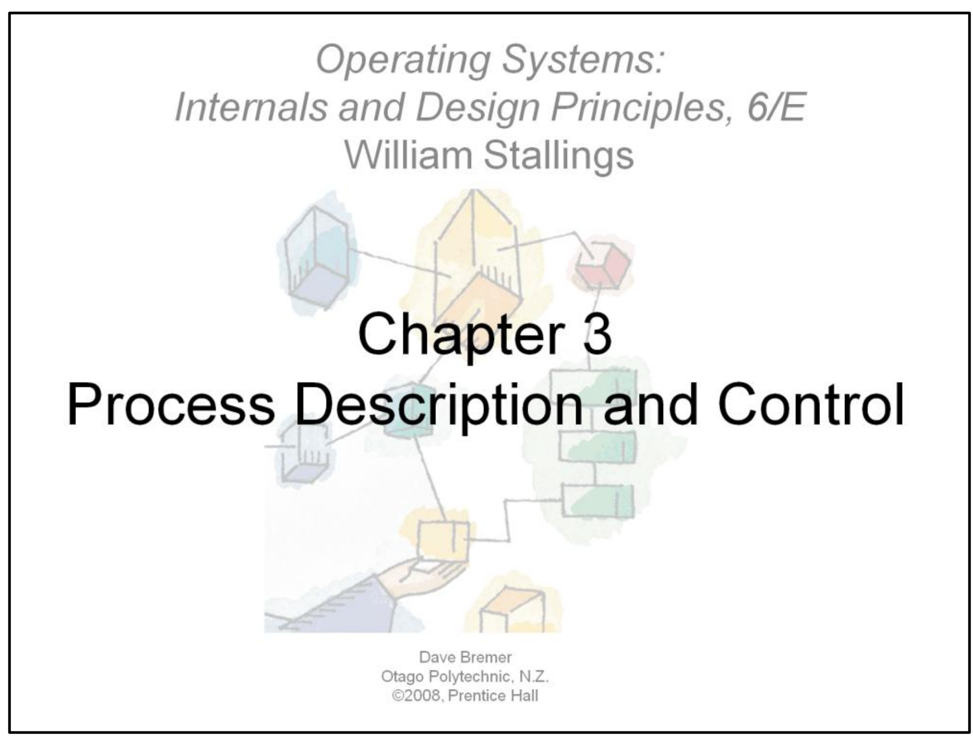
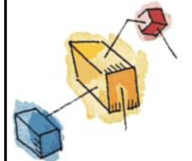


These slides are intended to help a teacher develop a presentation. This PowerPoint covers the entire chapter and includes too many slides for a single delivery. Professors are encouraged to adapt this presentation in ways which are best suited for their students and environment.



Roadmap

- 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.
  - Discuss process management in UNIX SVR4.

Note: In this chapter, reference is occasionally made to virtual memory. Much of the time, we can ignore this concept in dealing with processes, but at certain points in the discussion, virtual memory considerations are pertinent. Virtual memory is not discussed in detail until Chapter 8.

Refer students to a brief overview which is provided in Chapter 2.

Requirements of an Operating System

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

Explain that:

- on a multiprogramming uniprocessor, the execution of multiple processes can be interleaved in time.
- on a multiprocessor, not only may process execution be interleaved, but also multiple processes can execute simultaneously.

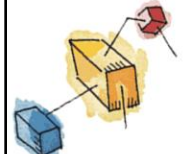
Explain that process management is compounded by the introduction of the concept of thread. In a multithreaded system,

- the ***process*** retains the attributes of resource ownership,
- The ***thread*** retains the attributes of multiple, concurrent execution streams running within a process.

Concepts

- From earlier chapters we saw:
  - Computer platforms consists of a collection of hardware resources
  - Computer applications are developed to perform some task
  - It is inefficient for applications to be written directly for a given hardware platform

Recapping chapters 1 & 2



Concepts cont…

– OS provides an interface for applications to use

– OS provides a representation of resources that can be requested and accessed by application

The OS Manages Execution of Applications

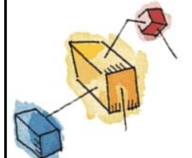
- Resources are made available to multiple applications
- The processor is switched among multiple application
- The processor and I/O devices can be used efficiently

What is a *"process"*?

- *A program in execution*
- An instance of a program running on a computer
- The entity that can be assigned to and executed on a processor
- A unit of activity characterized by the execution of a sequence of instructions, a current state, and an associated set of system instructions

Process Elements

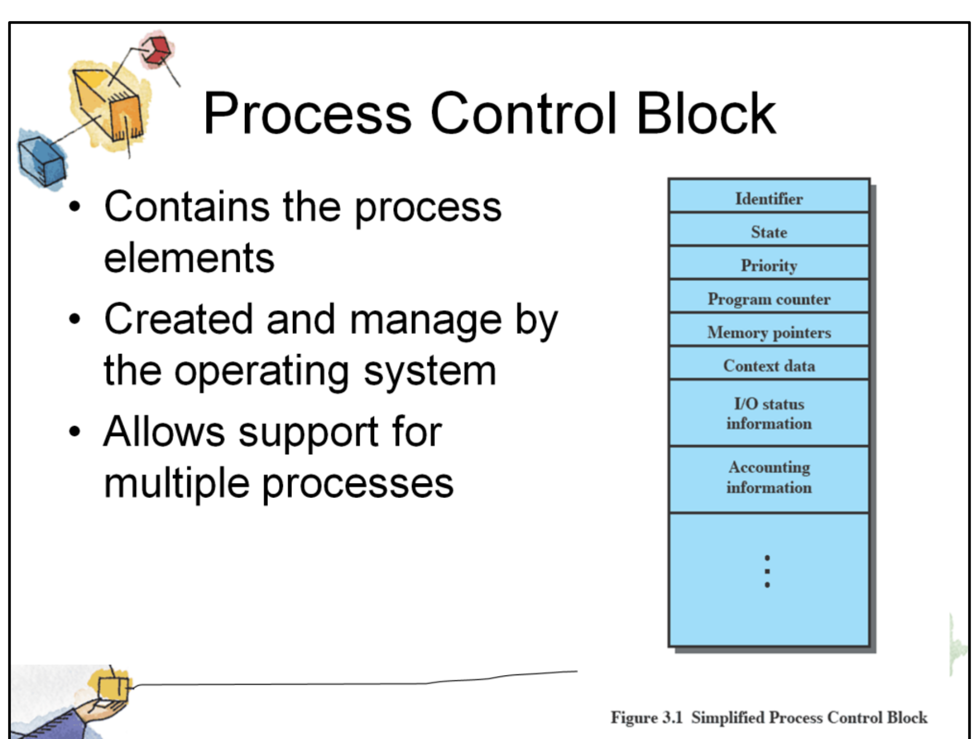
- A process is comprised of:
  - Program code (possibly shared)
  - A set of data
  - A number of attributes describing the state of the process



Process Elements

- While the process is running it has a number of elements including
  - Identifier
  - State
  - Priority
  - Program counter
  - Memory pointers
  - Context data
  - I/O status information
  - Accounting information

Mention that this list is not necessarily complete, and may include optional elements (e.g. not all OS's may need a process to have accounting information)

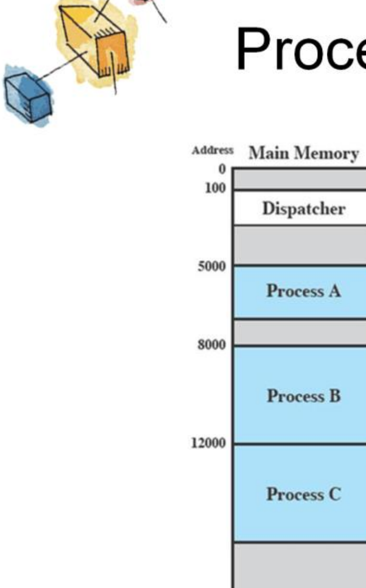


Emphasise that the Process Control Block contains sufficient information so that it is possible to interrupt a running process and later resume execution as if the interruption had not occurred.

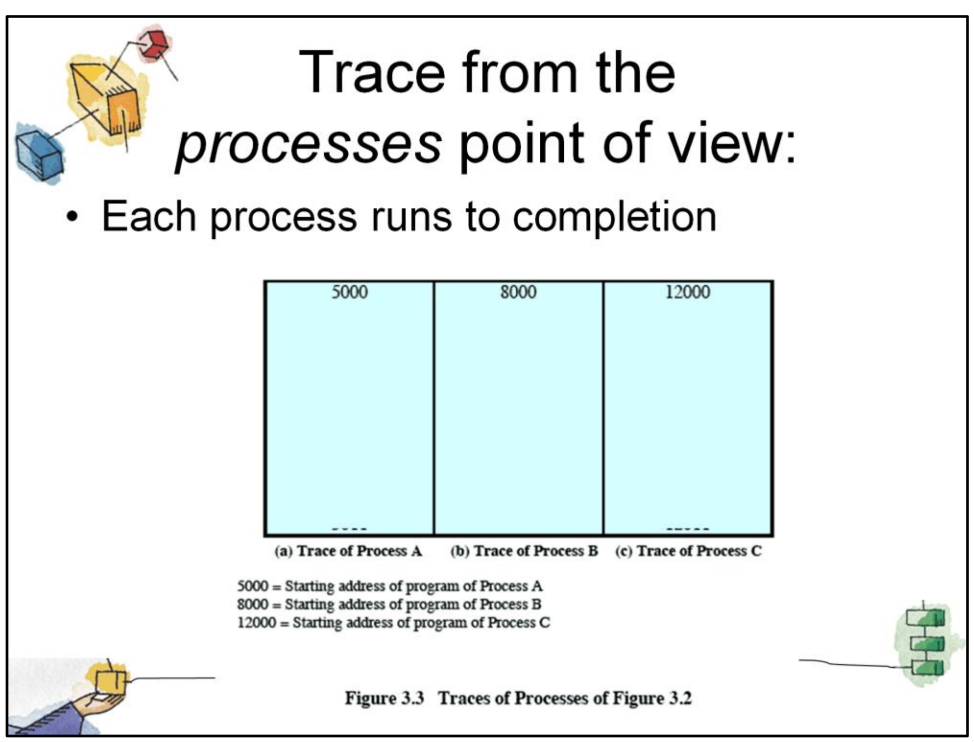
Trace of the Process

- The behavior of an individual process is shown by listing the sequence of instructions that are executed
- This list is called a ***Trace***
- ***Dispatcher*** is a small program which switches the processor from one process to another

Process Execution

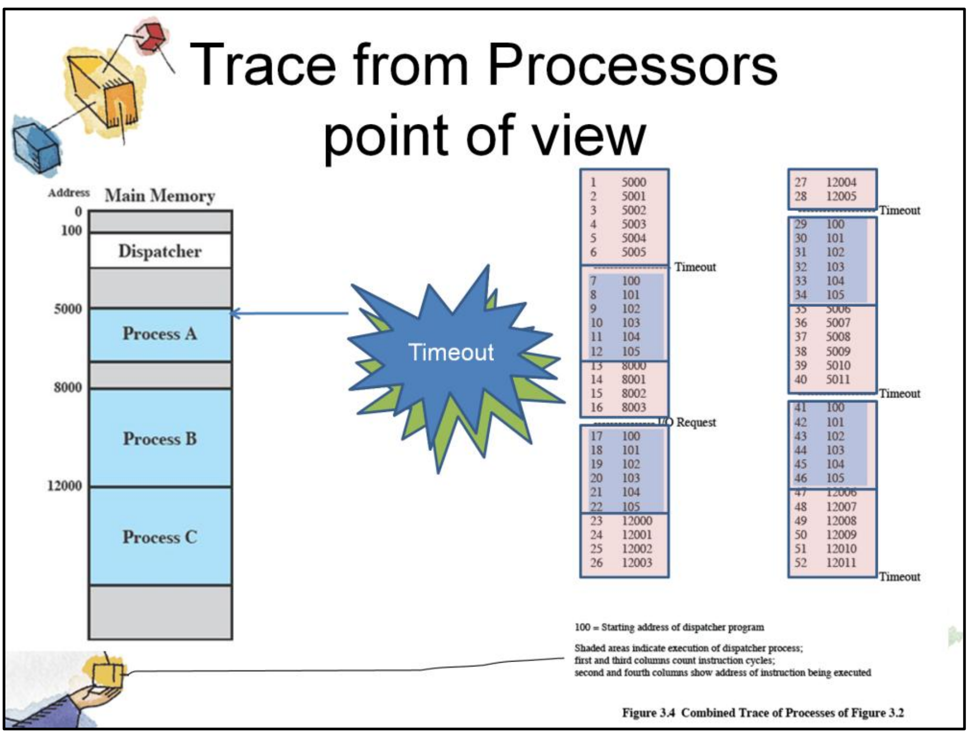


- Consider three processes being executed
- All are in memory (plus the dispatcher)
- Lets ignore virtual memory for this.



Animated slide

Wipes down to give impression of process progression



Animated slide

Explain the scenario then click for the animation

The shaded areas represent code executed by the dispatcher.

The same sequence of instructions is executed by the dispatcher in each instance because the same functionality of the dispatcher is being executed.

We assume that the OS only allows a process to continue execution for a maximum of six instruction cycles, after which it is interrupted; this prevents any single process from monopolizing processor time.

Animate here

The first six instructions of process A are executed, followed by a time-out and the execution of some code in the dispatcher, which executes six instructions before turning control to process B2.

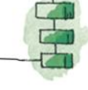
After four instructions are executed, process B requests an I/O action for which it must wait. Therefore, the processor stops executing process B and moves on, via the dispatcher, to process C.

After a time-out, the processor moves back to process A. When this process times out, process B is still waiting for the

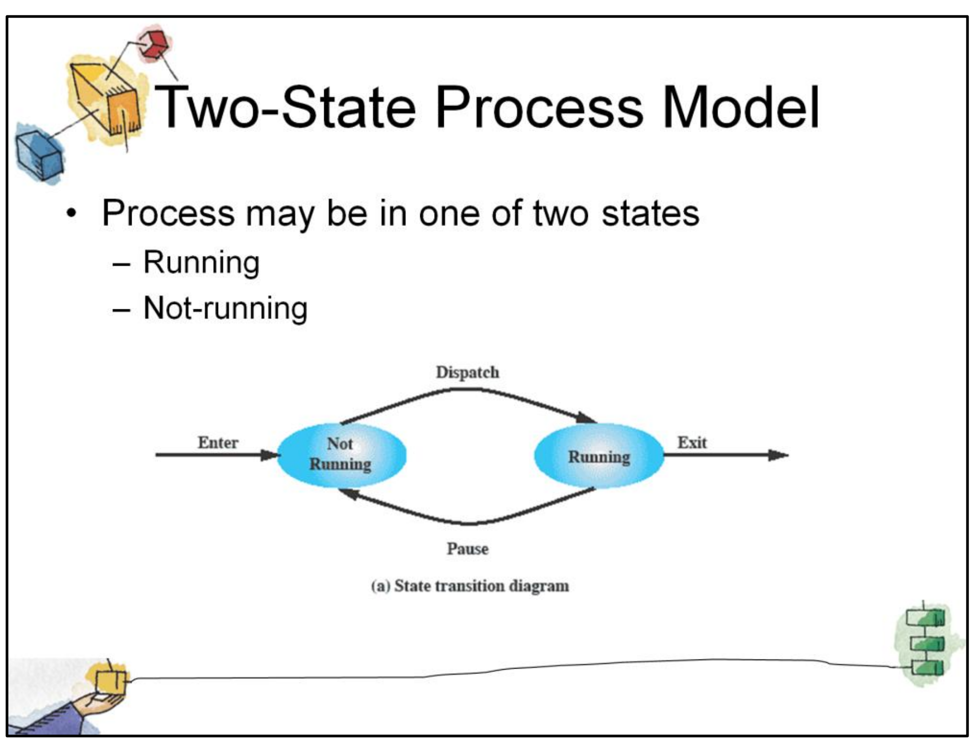
I/O operation to complete, so the dispatcher moves on to process C again.

Roadmap

- 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.
- Discuss process management in UNIX SVR4.



Progress indicator

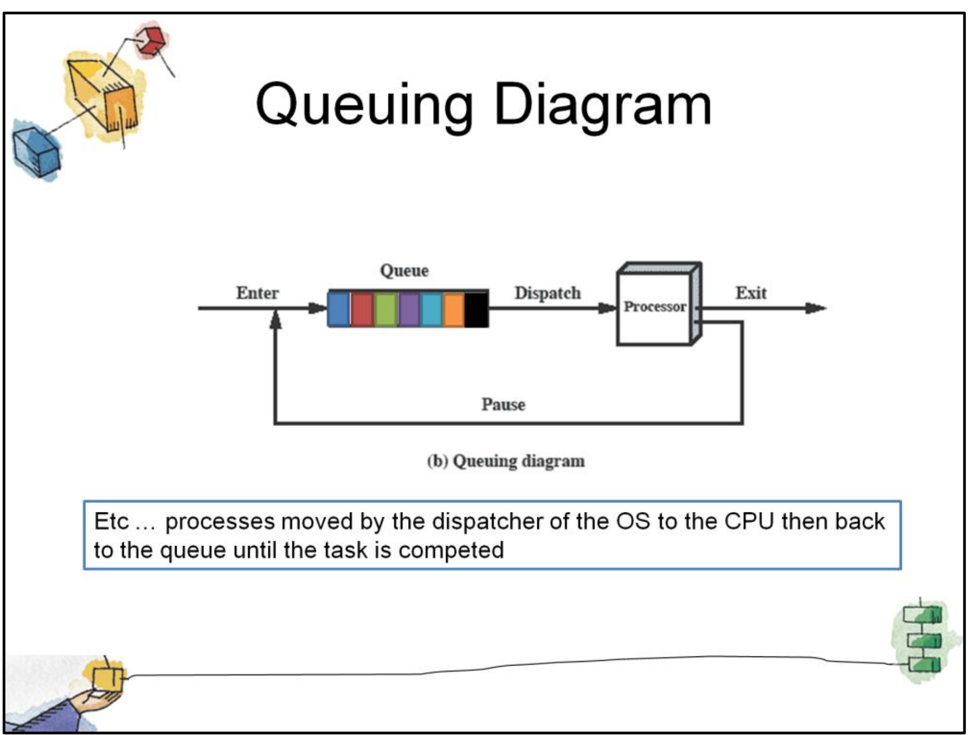


Emphasize that you are introducing a ***Simple*** model and that this will be expanded.

The operating system's principal responsibility is controlling the execution of processes;

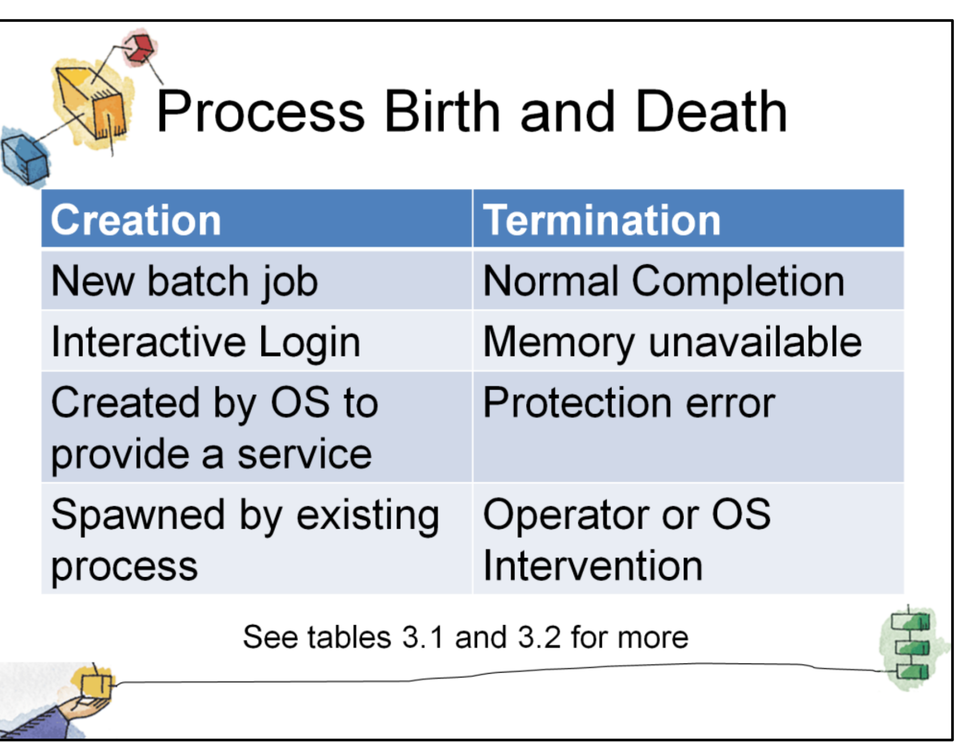
this includes determining the interleaving pattern for execution and allocating resources to processes.

The first step in designing an OS to control processes is to describe the behaviour that we would like the processes to exhibit. In the most simple model, a process is either running, or it is not.



Animated slide The animation only shows one process moving.

There needs to be some structure so that the OS can keep track of the processes. This could be a simple queue which is managed by the dispatcher routine of the OS.



There are many reasons why a process may be created or terminated.

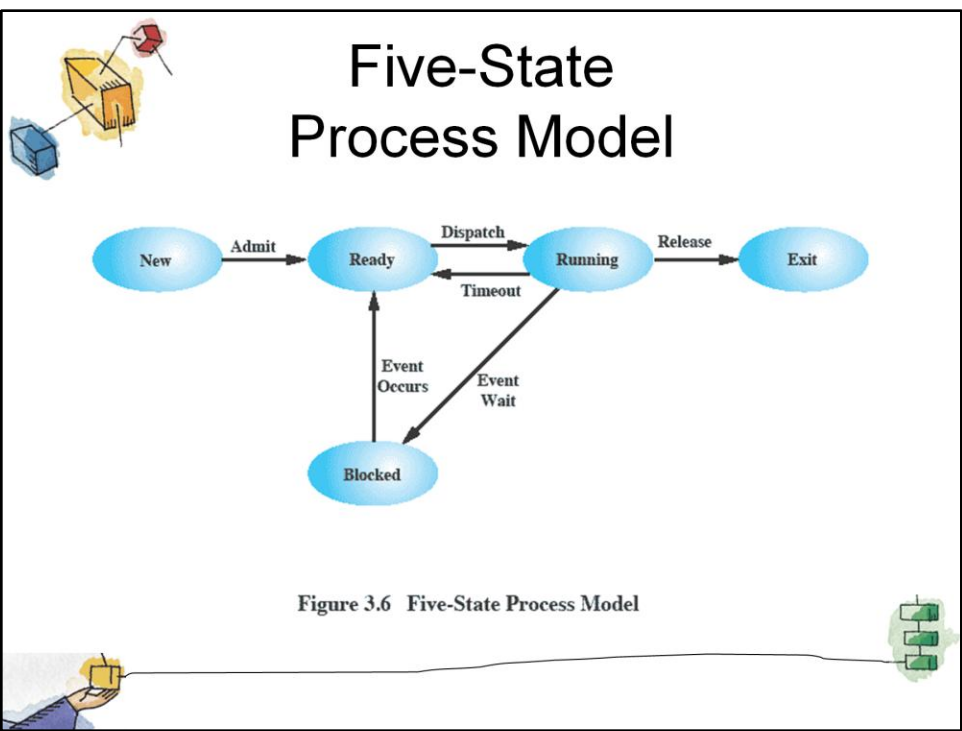
Process Creation

- The OS builds a data structure to manage the process
- 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

Typically, the "related" processes need to communicate and cooperate with each other. Achieving this cooperation is a difficult task for the programmer; discussed further in Chapter 5.

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



If all processes were always ready to execute, then the simple FIFO queuing model would suffice.

However, even with simple examples, this implementation is inadequate:

- some processes in the Not Running state are ready to execute,
- while others are blocked, waiting for an I/O operation to complete.

In a two state model, the dispatcher would have to scan the list looking for the process that is not blocked and that has been in the queue the longest.

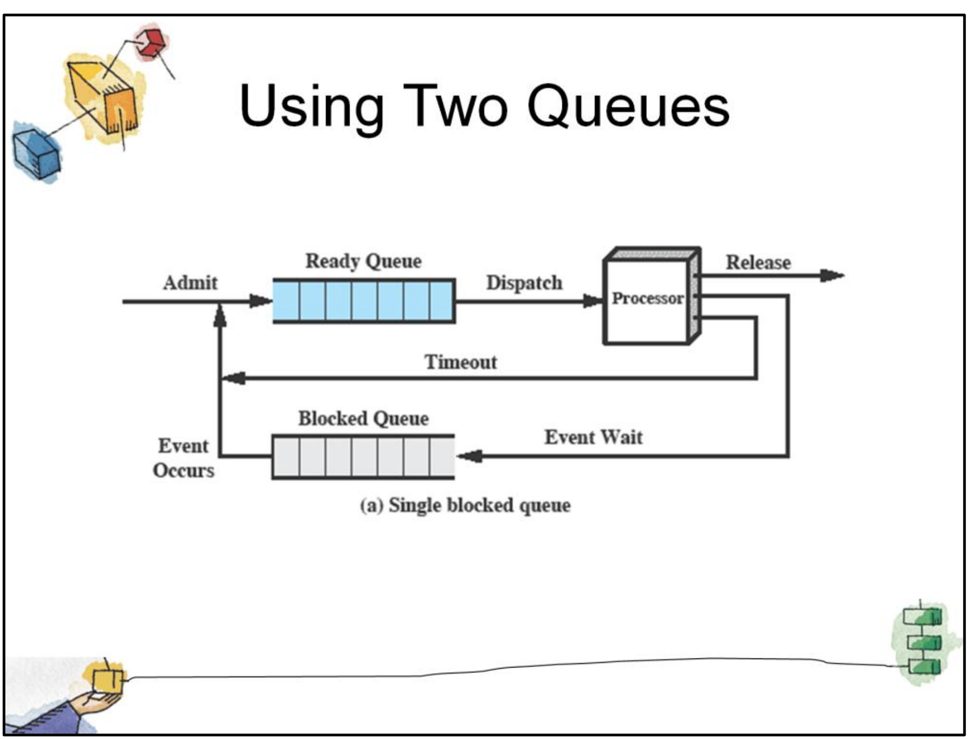
But we could split the ***Not Running*** state into two states:

- Ready and
- Blocked.

For good measure, we have added two additional states that will prove useful (new and exit)

Suggestion:

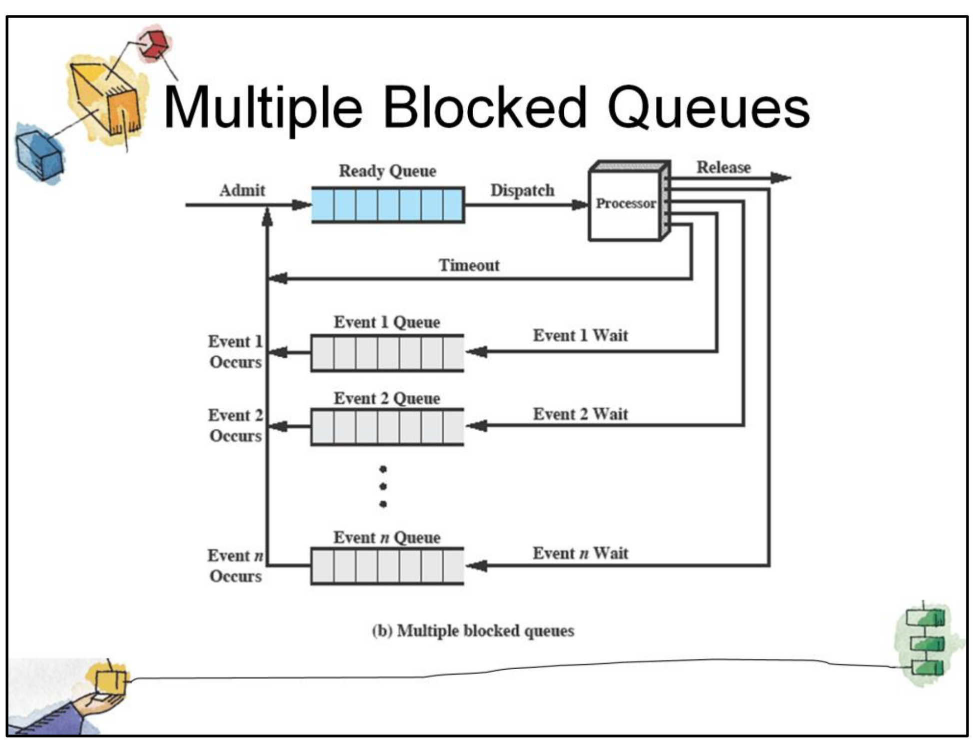
- Mention that “Wait” and “Blocked” have the same meaning here.
- Talk through the state transitions.



In the simplest solution, this model would require an additional queue for the blocked processes.

But when an event occurs the dispatcher would have to cycle through the entire queue to see which process is waiting for the event.

This can cause huge overhead when their may be 100's or 1000's of processes



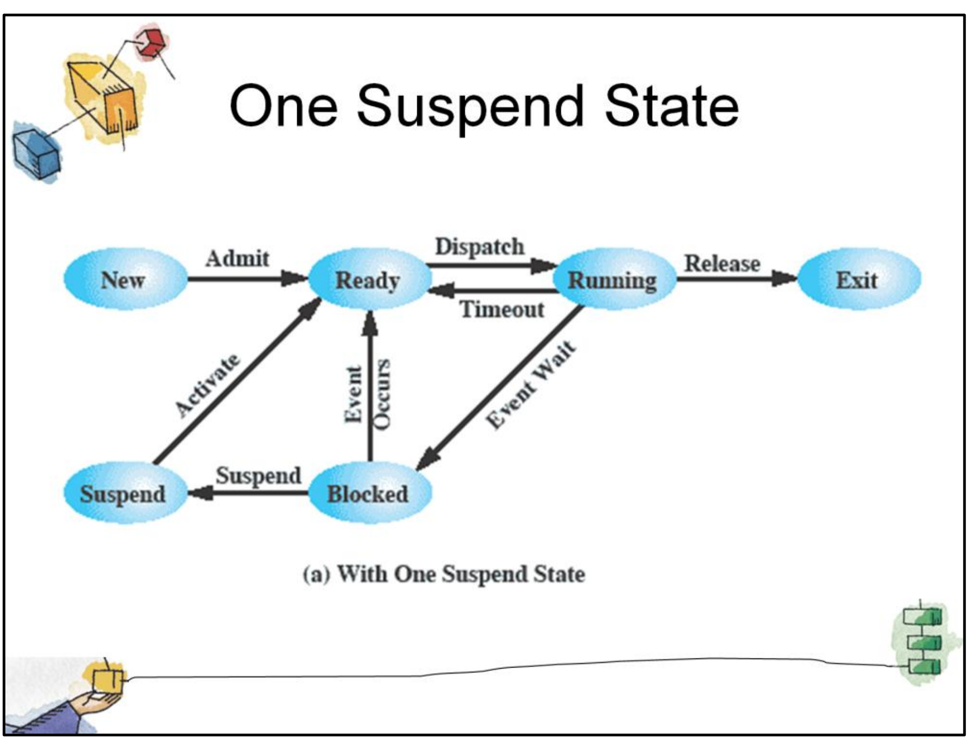
(b) Multiple blocked queues

More efficient to have a separate ‘blocked’ queue for each type of event.

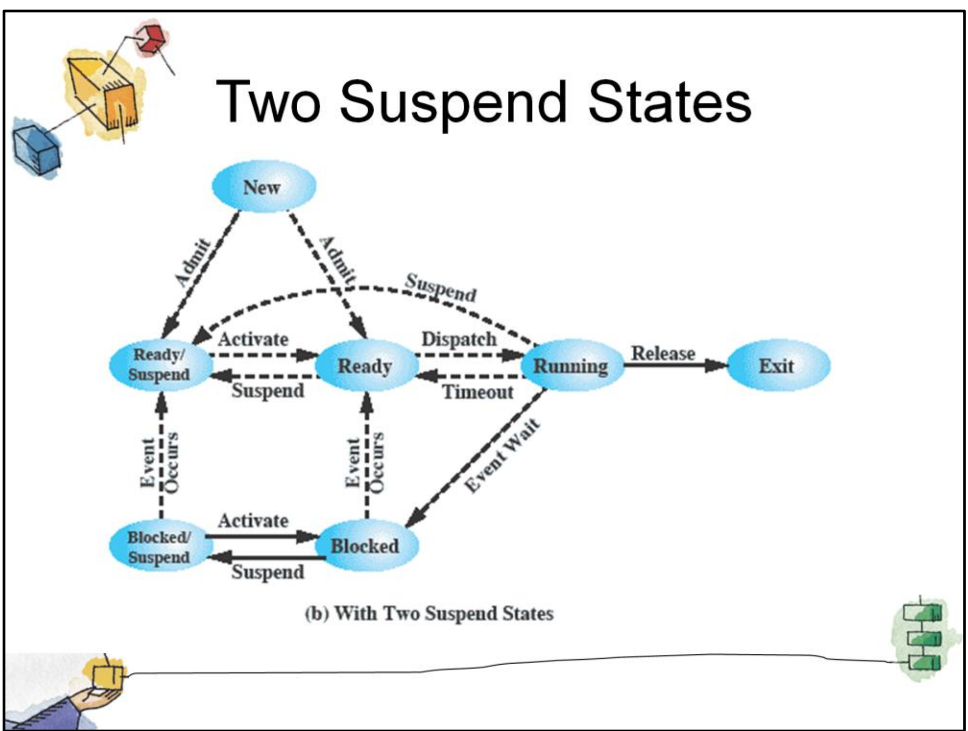
Suspended Processes

- Processor is faster than I/O so all processes could be waiting for I/O
  - Swap these processes to disk to free up more memory and use processor on more processes
- Blocked state becomes ***suspend*** state when swapped to disk
- Two new states
  - Blocked/Suspend
  - Ready/Suspend

Mention that the aim is to fully utilize the processor.



Again, the simple solution is to add a single state – but this only allows processes which are blocked to be swapped out.



(b) With Two Suspend States

Two suspend states allow all processes which are not actually running to be swapped.

Run through the four states:

• **Ready:** The process is in main memory and available for execution.

• **Blocked:** The process is in main memory and awaiting an event.

• **Blocked/Suspend:** The process is in secondary memory and awaiting an event.

• **Ready/Suspend:** The process is in secondary memory but is available for execution as soon as it is loaded into main memory.

It would be useful to also summarise the main transitions (abridged)

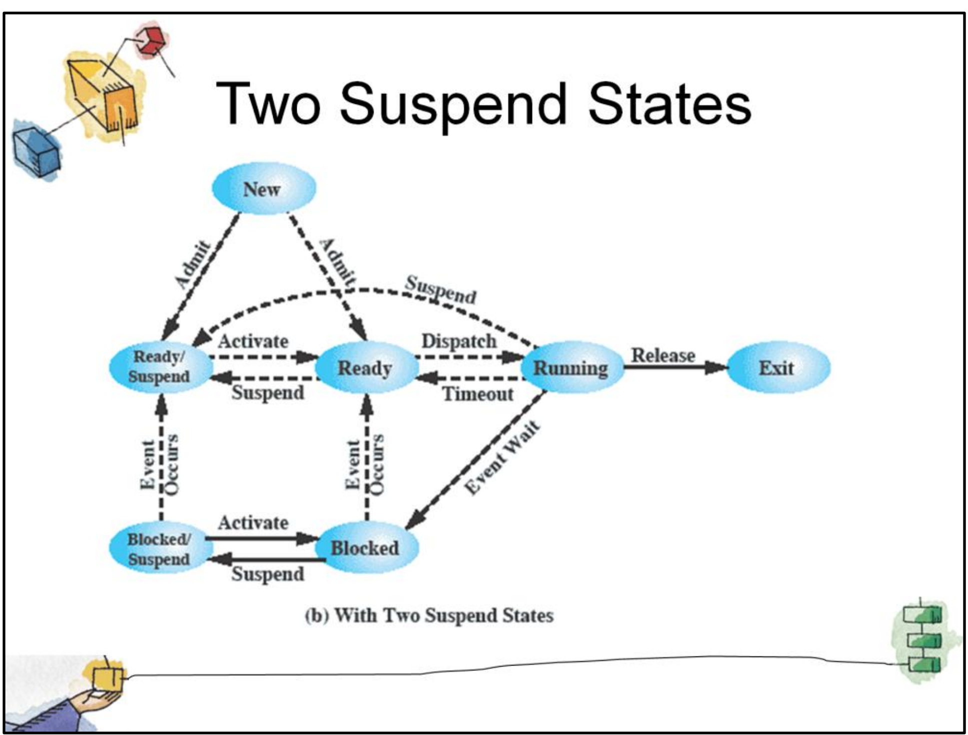
Blocked → Blocked/Suspend: If there are no ready processes, then at least one blocked process is swapped out to make room for another process that is not blocked. This transition can be made even if there are ready processes available, if the OS determines that the currently running process or a ready process that it would like to dispatch requires more main memory to maintain adequate performance.

Blocked/Suspend → Ready/Suspend: A process in the Blocked/Suspend state is moved to the Ready/Suspend state when the event for which it has been waiting occurs.

Ready/Suspend S → Ready: When there are no ready processes in main memory, or if a suspended process has a higher priority, the OS will need to bring one in to continue execution.

Ready → Ready/Suspend: Normally, the OS would prefer to suspend a blocked process rather than a ready one, because the ready process can now be executed, whereas the blocked process is taking up main memory space and cannot be executed. However, it may be necessary to suspend a ready process if that is the only way to free up a sufficiently large block of main memory.

Next picture...



Several other transitions that are worth considering are the following:

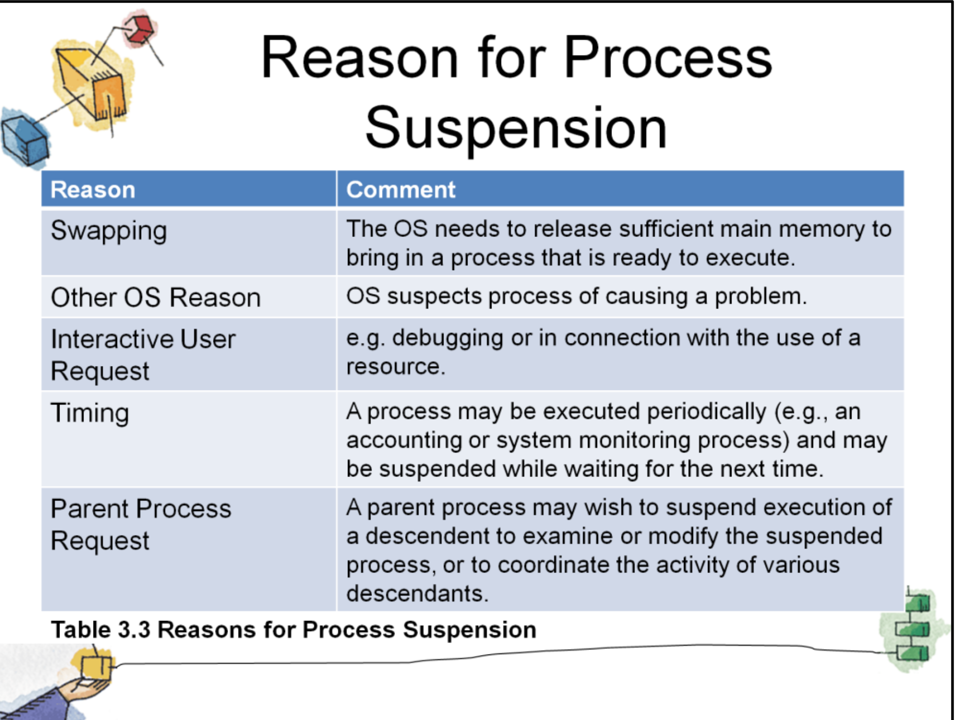
New → Ready/Suspend and New →Ready: When a new process is created, it can either be added to the Ready queue or the Ready/Suspend queue. In either case, the OS must create a process control block and allocate an address space to the process. It might be preferable for the OS to perform these housekeeping duties at an early time, so that it can maintain a large pool of processes that are not blocked. With this strategy, there would often be insufficient room in main memory for a new process; hence the use of the (New → Ready/Suspend) transition.

Blocked/Suspend → Blocked: Inclusion of this transition may seem to be poor design. After all, if a process is not ready to execute and is not already in main memory, what is the point of bringing it in? But consider the following scenario:

- A process terminates, freeing up some main memory.
- There is a process in the (Blocked/Suspend) queue with a higher priority than any of the processes in the (Ready/Suspend) queue and
- the OS has reason to believe that the blocking event for that process will occur soon.
- Under these circumstances, it would seem reasonable to bring a blocked process into main memory in preference to a ready process.

Running → Ready/Suspend: Normally, a running process is moved to the Ready state when its time allocation expires. If, however, the OS is pre-empting the process because a higher-priority process on the Blocked/Suspend queue has just become unblocked, the OS could move the running process directly to the (Ready/Suspend) queue and free some main memory.

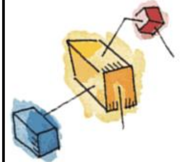
Any State → Exit: Typically, a process terminates while it is running, either because it has completed or because of some fatal fault condition. However, in some operating systems, a process may be terminated by the process that created it or when the parent process is itself terminated. If this is allowed, then a process in any state can be moved to the Exit state.



Reason	Comment
Swapping	The OS needs to release sufficient main memory to bring in a process that is ready to execute.
Other OS Reason	OS suspects process of causing a problem.
Interactive User Request	e.g. debugging or in connection with the use of a resource.
Timing	A process may be executed periodically (e.g., an accounting or system monitoring process) and may be suspended while waiting for the next time.
Parent Process Request	A parent process may wish to suspend execution of a descendent to examine or modify the suspended process, or to coordinate the activity of various descendants.

Table 3.3 Reasons for Process Suspension

The comments are summarized due to space on slide



Roadmap

- 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.
- Discuss process management in UNIX SVR4.

Progress indicator

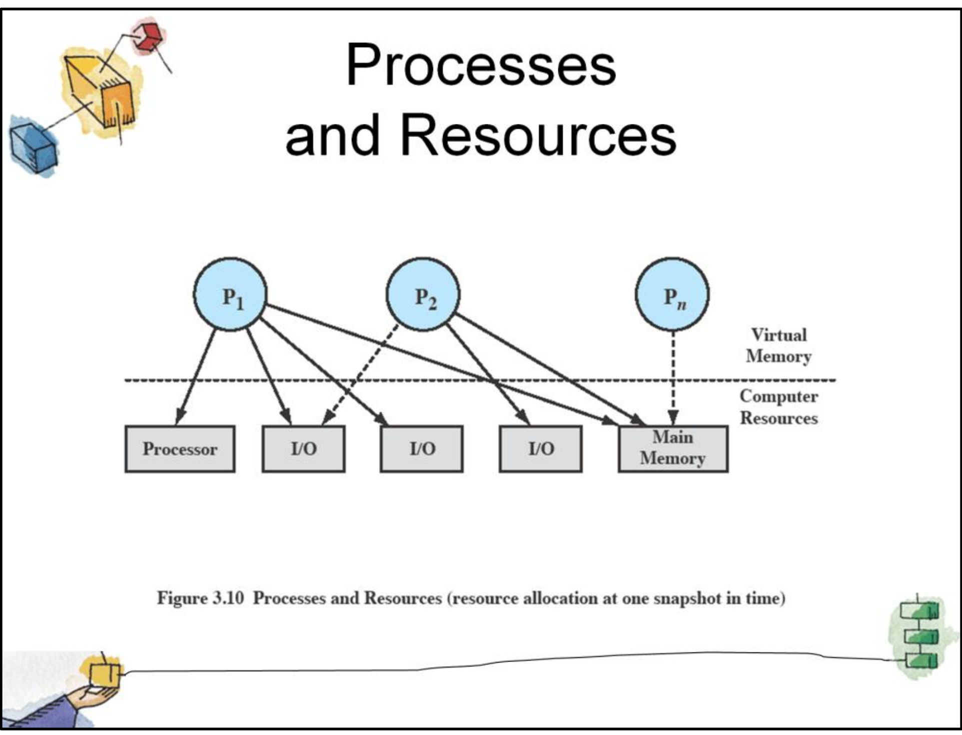


Figure 3.10 Processes and Resources (resource allocation at one snapshot in time)

The OS controls events within the computer system. It schedules and dispatches processes for execution by the processor, allocates resources to processes, and responds

to requests by user processes for basic services.

Fundamentally, we can think of the OS as that entity that manages the use of system resources by processes.

In this slide –

- There are a number of processes (P1, . . ., Pn,) that have been created and exist in virtual memory.
- Each process, during the course of its execution, needs access to certain system resources, including the processor, I/O devices, and main memory.
- In the figure, process P1 is running; at least part of the process is in main memory, and it has control of two I/O devices.
- Process P2 is also in main memory but is blocked waiting for an I/O device allocated to P1.
- Process Pn has been swapped out and is therefore suspended.

Operating System Control Structures

- For the OS is to manage processes and resources, it must have information about the current status of each process and resource.
- Tables are constructed for each entity the operating system manages

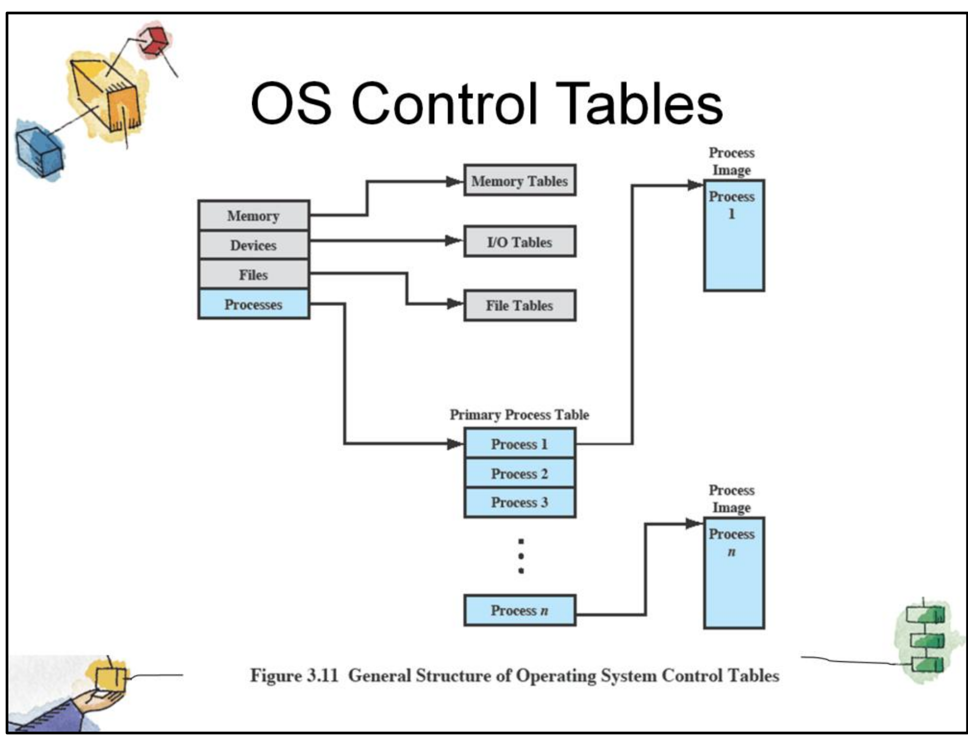


Figure 3.11 General Structure of Operating System Control Tables

A general idea of the scope of the tables is in Figure 3.11, which shows four different types of tables maintained by the OS:

- memory,
- I/O,
- file,
- process.

Although the details will differ from one OS to another, fundamentally, all operating systems maintain information in these four categories.

The next few slides mention details of these four tables.

You may prefer to stick with this diagram and discuss each from here

Memory Tables

- Memory tables are used to keep track of both main and secondary memory.
- Must include this information:
  - Allocation of main memory to processes
  - Allocation of secondary memory to processes
  - Protection attributes for access to shared memory regions
  - Information needed to manage virtual memory

Memory tables are used to keep track of both main (real) and secondary (virtual) memory.

Some of main memory is reserved for use by the OS; the remainder is available for use by processes.

Processes are maintained on secondary memory using some sort of virtual memory or simple swapping mechanism.

I/O Tables

- Used by the OS to manage the I/O devices and channels of the computer.
- The OS needs to know
  - Whether the I/O device is available or assigned
  - The status of I/O operation
  - The location in main memory being used as the source or destination of the I/O transfer

I/O tables are used by the OS to manage the I/O devices and channels of the computer system.

At any given time, an I/O device may be available or assigned to a particular process.

If an I/O operation is in progress, the OS needs to know the status of the I/O operation and the location in main memory being used as the source or destination of the I/O transfer.

File Tables

- These tables provide information about:
  - Existence of files
  - Location on secondary memory
  - Current Status
  - other attributes.
- Sometimes this information is maintained by a file management system

The OS may also maintain file tables.

Much, if not all, of this information may be maintained and used by a file management system, in which case the OS has little or no knowledge of files.

In other operating systems, much of the detail of file management is managed by the OS itself.

Process Tables

- To manage processes the OS needs to know details of the processes
  - Current state
  - Process ID
  - Location in memory
  - etc
- Process control block
  - ***Process image*** is the collection of program. Data, stack, and attributes

Finally, the OS must maintain process tables to manage processes.

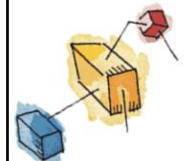
The remainder of this section is devoted to an examination of the required process tables.

Note: Although Figure 3.11 (earlier slide) shows four distinct sets of tables, it should be clear that these tables must be linked or cross-referenced in some fashion.

- Memory, I/O, and files are managed on behalf of processes, so there must be some reference to these resources, directly or indirectly, in the process tables.

The files referred to in the file tables are accessible via an I/O device and will, at some times, be in main or virtual memory.

The tables themselves must be accessible by the OS and therefore are subject to memory management.



Process Attributes

- We can group the process control block information into three general categories:
  - Process identification
  - Processor state information
  - Process control information

A sophisticated multiprogramming system requires a great deal of information about each process.

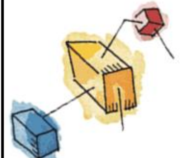
Different systems will organize this information in different ways

For now, let us simply explore the type of information that might be of use to an OS without considering in any detail how that information is organized.

Table 3.5 in the book (not in this slide set) lists the typical categories of information required by the OS for each process.

We can group the process control block information into three general categories:

- Process identification
- Processor state information
- Process control information



Process Identification

- Each process is assigned a unique numeric identifier.
- Many of the other tables controlled by the OS may use process identifiers to cross-reference process tables

In virtually all operating systems, each process is assigned a unique numeric identifier,

- which may simply be an index into the primary process table (see Figure 3.11);
- otherwise there must be a mapping that allows the OS to locate the appropriate tables based on the process identifier.

This identifier is useful in several ways.

Many of the other tables controlled by the OS may use process identifiers to cross-reference process tables.

For example,

- The memory tables may be organized so as to provide a map of main memory with an indication of which process is assigned to each region.
- Similar references will appear in I/O and file tables.
- When processes communicate with one another, the process identifier informs the OS of the destination of a particular communication.
- When processes are allowed to create other processes, identifiers indicate the parent and descendents of each process.

In addition to these process identifiers, a process may be assigned a user identifier that indicates the user responsible for the job.

Processor State Information

- This consists of the contents of processor registers.
  - User-visible registers
  - Control and status registers
  - Stack pointers
- Program status word (PSW)
  - contains status information
  - Example: the EFLAGS register on Pentium processors

Processor state information consists of the contents of processor registers.

While a process is running the information is in the registers.

When a process is interrupted, all of this register information must be saved so that it can be restored when the process resumes execution. Nature and number of processes vary by processor design

Typically, the register set will include user-visible registers, control and status registers, and stack pointers.

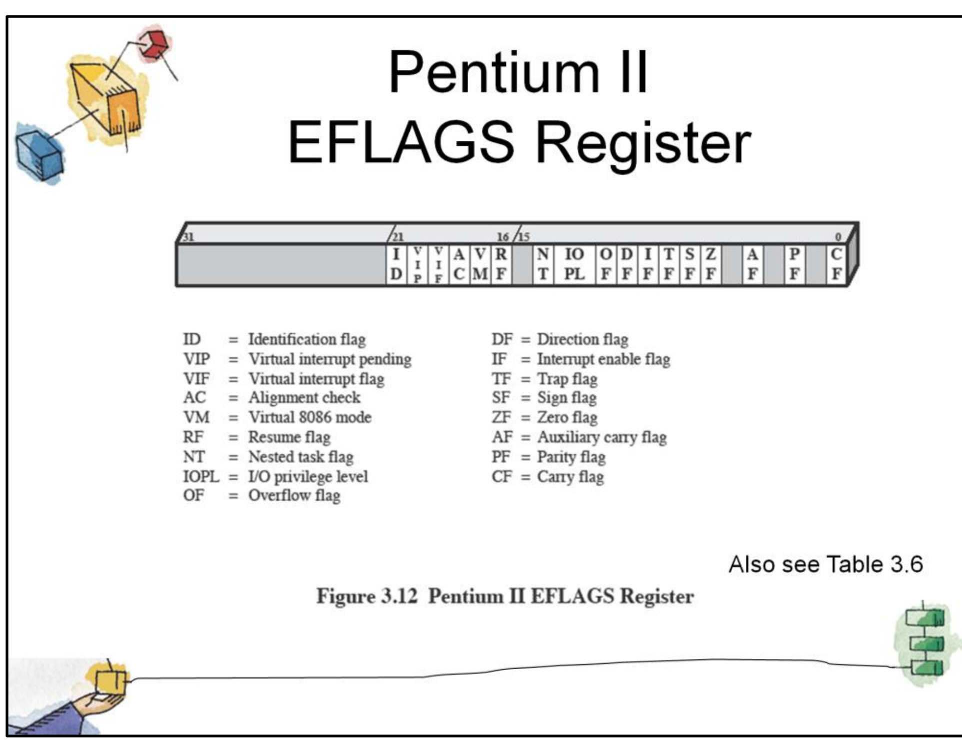


Figure 3.12 Pentium II EFLAGS Register

All processor designs include a register or set of registers, often known as the program status word (PSW), that contains status information.

The PSW typically contain condition codes plus other status information.

A good example of a processor status word is that on Pentium processors, referred to as the EFLAGS register

This structure is used by any OS (including UNIX and Windows) running on a Pentium processor.

Process Control Information

- This is the additional information needed by the OS to control and coordinate the various active processes.
  - See table 3.5 for scope of information

This is the additional information needed by the OS to control and coordinate the various active processes.

The last part of Table 3.5 indicates the scope of this information.

As we examine the details of operating system functionality in succeeding chapters, the need for the various items on this list should become clear.

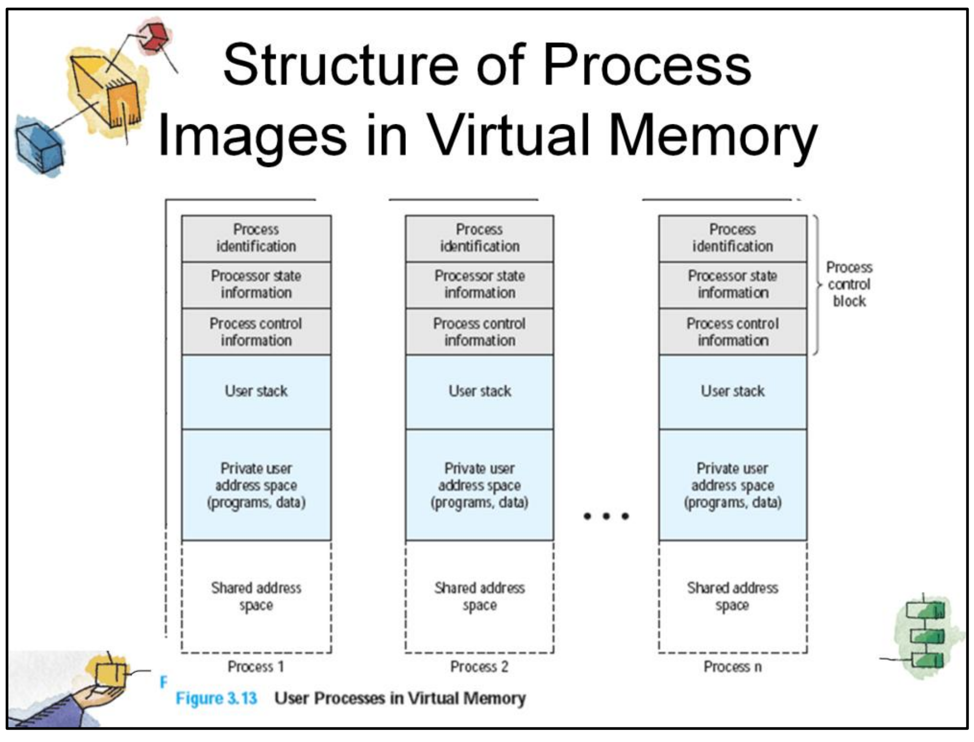


Figure 3.13 suggests the structure of process images in virtual memory.

Each process image consists of

- a process control block,
- a user stack,
- the private address space of the process, and
- any other address space that the process shares with other processes.

In the figure, each process image appears as a contiguous range of addresses. In an actual implementation, this may not be the case; it will depend on the memory management scheme and the way in which control structures are organized by the OS.

The process control block may contain structuring information, including pointers that allow the linking of process control blocks.

Thus, the queues that were described in the preceding section could be implemented as linked lists of process control blocks.

Role of the Process Control Block

- The most important data structure in an OS
  - It defines the state of the OS
- Process Control Block requires protection
  - A faulty routine could cause damage to the block destroying the OS's ability to manage the process
  - Any design change to the block could affect many modules of the OS

The process control block is the most important data structure in an OS.

- Each process control block contains all of the information about a process that is needed by the OS.
- The blocks are read and/or modified by virtually every module in the OS, including those involved with scheduling, resource allocation, interrupt processing, and performance monitoring and analysis.

One can say that the set of process control blocks defines the state of the OS.

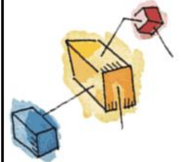
Important design issue.

A number of routines within the OS will need access to information in process control blocks.

The difficulty is not access but rather protection.

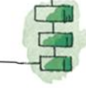
Two problems present themselves:

- A bug in a single routine, such as an interrupt handler, could damage process control blocks, which could destroy the system's ability to manage the affected processes.
- A design change in the structure or semantics of the process control block could affect a number of modules in the OS.

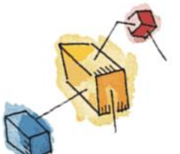


Roadmap

- 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.
- Discuss process management in UNIX SVR4.



Progress indicator



Modes of Execution

- Most processors support at least two modes of execution
- User mode
  - Less-privileged mode
  - User programs typically execute in this mode
- System mode
  - More-privileged mode
  - Kernel of the operating system

The less-privileged mode is often referred to as the user mode, because user programs typically would execute in this mode.

Mention that System Mode is also known as

- Control Mode
- Kernel Mode
- Protected Mode

Certain instructions can only be executed in the more-privileged mode.

- Including reading or altering a control register, such as the program status word;
- primitive I/O instructions;
- Instructions that relate to memory management.

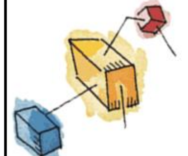
In addition, certain regions of memory can only be accessed in the more-privileged mode.

Table 3.7 lists the functions typically found in the kernel of an OS.

Questions: How does the processor know in which mode it is to be executing? And how does it change

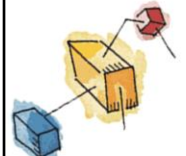
Answer:

- Typically a flag (single bit) in the program status word (PSW). This bit is changed in response to certain events.
- Typically, when a user makes a call to an operating system service or when an interrupt triggers execution of an operating system routine, the mode is set to the kernel mode and, upon return from the service to the user process, the mode is set to user mode.



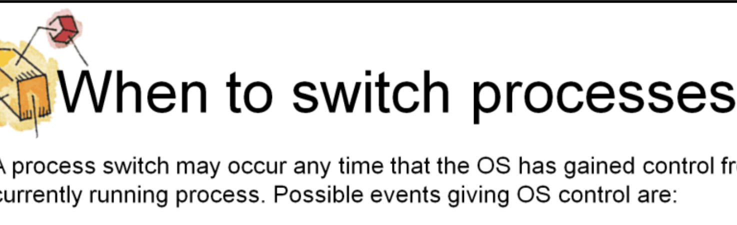
Process Creation

- Once the OS decides to create a new process it:
  - Assigns a unique process identifier
  - Allocates space for the process
  - Initializes process control block
  - Sets up appropriate linkages
  - Creates or expand other data structures



Switching Processes

- Several design issues are raised regarding process switching
  - What events trigger a process switch?
  - We must distinguish between mode switching and process switching.
  - 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

A process switch may occur any time that the OS has gained control from the currently running process.

Table 3.8 suggests the possible events that may give control to the OS.

Two kinds of system interrupts,

- one is simply called an **interrupt**,
- and the other called a **trap**.

"interrupts" are due to some sort of event that is external to and independent of the currently running process, such as the completion of an I/O operation.

With an ordinary interrupt, control is first transferred to an interrupt handler, which does some basic housekeeping and then branches to an OS routine that is concerned with

the particular type of interrupt that has occurred.

"Traps" relate to an error or exception condition generated within the currently running process, such as an illegal file access attempt.

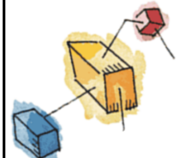
With traps, the OS determines if the error or exception condition is fatal.

- If so, then the currently running process is moved to the Exit state and a process switch occurs.
- If not, then the action of the OS will depend on the nature of the error and the design of the OS.
    - It may attempt some recovery procedure or simply notify the user.
    - It may do a process switch or resume the currently running process.

Finally, the OS may be activated by a **supervisor call** from the program being executed.

For example, a user process is running and an instruction is executed that requests an I/O operation, such as a file open.

This call results in a transfer to a routine that is part of the operating system code. The use of a system call may place the user process in the Blocked state.



Change of Process State …

- The steps in a process switch are:
  1. Save context of processor including program counter and other registers
  2. Update the process control block of the process that is currently in the Running state
  3. Move process control block to appropriate queue – ready; blocked; ready/suspend

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.

The steps involved in a full process switch are as follows:

1. Save the context of the processor, including program counter and other registers.
2. Update the process control block of the process that is currently in the Running state.

   This includes changing the state of the process to one of the other states (Ready; Blocked; Ready/Suspend; or Exit). Other relevant fields must also be updated, including the reason for leaving the Running state and accounting information.

3. Move the process control block of this process to the appropriate queue (Ready;Blocked on Event*i*; Ready/Suspend).

More next slide

Change of Process State cont…

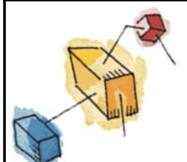
4. Select another process for execution
5. Update the process control block of the process selected
6. Update memory-management data structures
7. Restore context of the selected process

4. Select another process for execution;

5. Update the process control block of the process selected. This includes changing the state of this process to Running.

6. Update memory management data structures. This may be required, depending on how address translation is managed;

7. Restore the context of the processor to that which existed at the time the selected process was last switched out of the Running state, by loading in the previous values of the program counter and other registers.

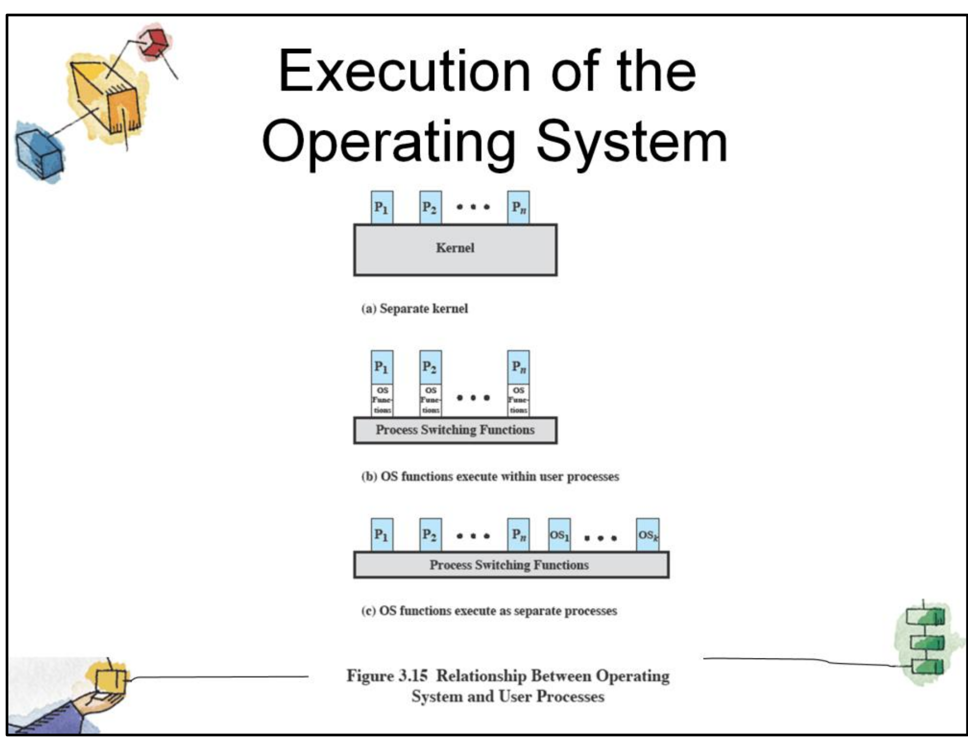


Is the OS a Process?

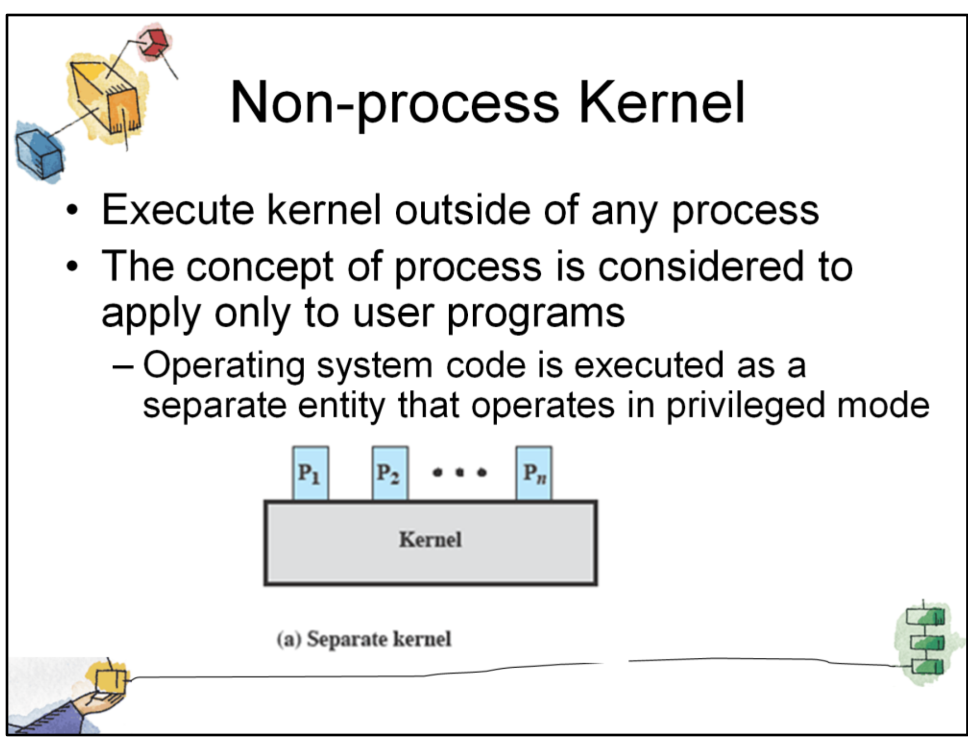
- If the OS is just a collection of programs and if it is executed by the processor just like any other program, is the OS a process?
- If so, how is it controlled?
  - Who (what) controls it?

If the OS is just a collection of programs and if it is executed by the processor just like any other program, is the OS a process?

If so, how is it controlled? These interesting questions have inspired a number of design approaches.



This figure is explained in the following slides



The key point is that the concept of process is considered to apply only to user programs. The operating system code is executed as a separate entity that operates in privileged mode.

Common on many older operating systems,

Execute the kernel of the OS outside of any process.

When the currently running process is interrupted or issues a supervisor call,

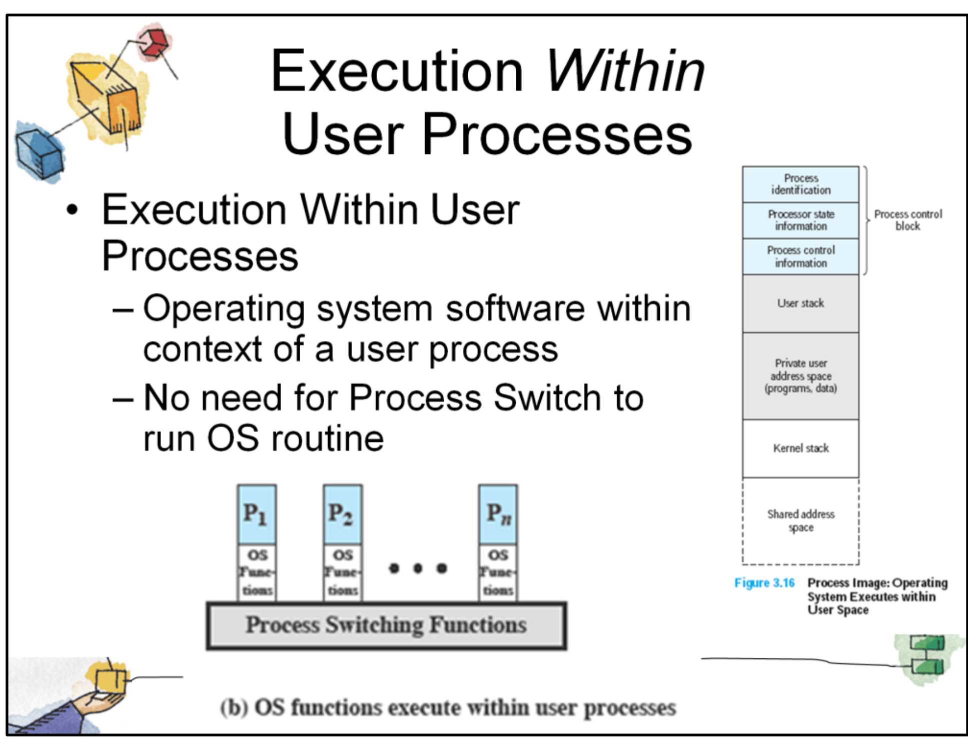
the mode context of this process is saved and control is passed to the kernel.

The OS has its own region of memory to use and its own system stack for controlling procedure calls and returns.

The OS can perform any desired functions and restore the context of the interrupted process, which causes execution to resume in the interrupted user process.

Or, the OS can complete the function of saving the environment of the process and proceed to schedule and dispatch another process. Whether this happens depends on the reason for the interruption and the circumstances at the time.

The key point is that the concept of process is considered to apply only to user programs. The operating system code is executed as a separate entity that operates in privileged mode.



Common with operating systems on smaller computers (PCs, workstations)

Execute virtually all OS software in the context of a user process

The view is that the OS is primarily a collection of routines that the user calls to perform various functions, executed within the environment of the user's process.

At any given point, the OS is managing ***n*** process images.

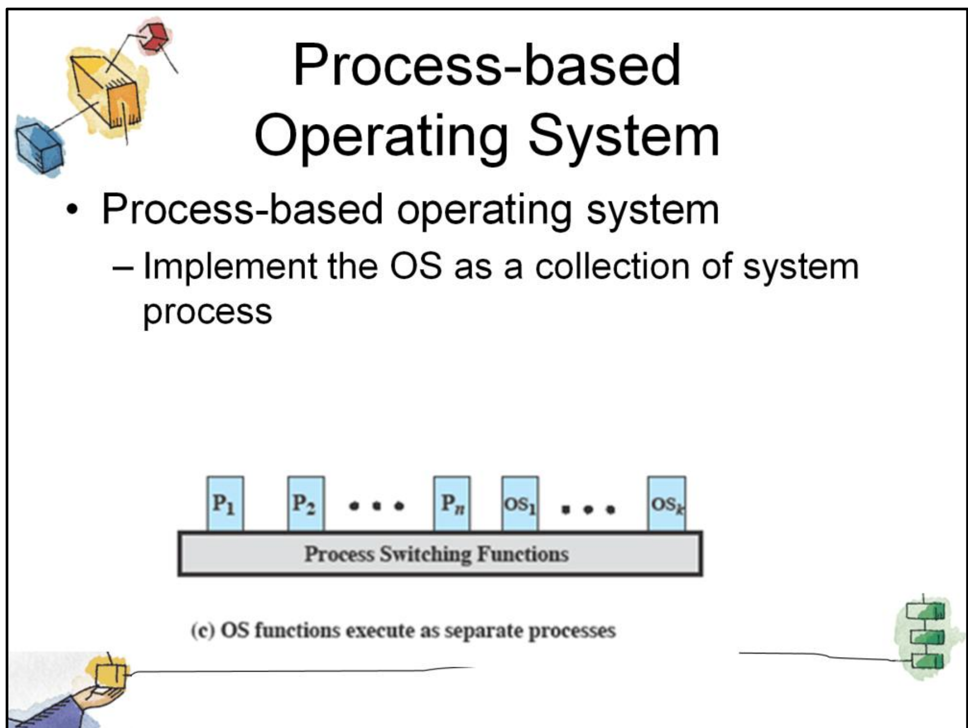
Each image includes not only the regions illustrated, but also program, data, and stack areas for kernel programs.

Figure 3.16 suggests a typical process image structure for this strategy.

- A separate kernel stack is used to manage calls/returns while the process is in kernel mode.
- Operating system code and data are in the shared address space and are shared by all user processes.
- When an interrupt, trap, or supervisor call occurs, the processor is placed in kernel mode and control is passed to the OS.
- To pass control from a user program to the OS, the mode context is saved and a mode switch takes place to an operating system routine.
- However, execution continues within the current user process. Thus, a process switch is not performed, just a mode switch within the same process.
- If the OS, upon completion of its work, determines that the current process should continue to run, then a mode switch resumes the interrupted program within the current process.

This is one of the key advantages of this approach:

A user program has been interrupted to employ some operating system routine, and then resumed, and all of this has occurred without incurring the penalty of two process switches.



Another alternative is to implement the OS as a collection of system processes.

As in the other options, the software that is part of the kernel executes in a kernel mode.

In this case, however, major kernel functions are organized as separate processes.

Again, there may be a small amount of process switching code that is executed outside of any process.

This approach has several advantages.

- It imposes a program design discipline that encourages the use of a modular OS with minimal, clean interfaces between the modules.
- Some noncritical operating system functions are conveniently implemented as separate processes.
- Implementing the OS as a set of processes is useful in a multiprocessor or multicomputer environment, in which some of the operating system services can be shipped out to dedicated processors, improving performance.

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.

An OS associates a set of privileges with each process.

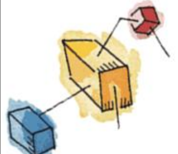
These privileges dictate what resources the process may access, including regions of memory, files, privileged system instructions, and so on.

- Typically, a process that executes on behalf of a user has the privileges that the OS recognizes for that user.
- A system or utility process may have privileges assigned at configuration time.

Typically the highest level of privilege is referred to as administrator, supervisor, or root, access. Root access provides access to all the functions and

services of the operating system.

- With root access, a process has complete control of the system and can add or changes programs and files, monitor other processes, send and receive network traffic, and alter privileges.
- A key security issue in the design of any OS is to prevent, or at least detect, attempts by a user or a piece of malicious software (malware) from gaining unauthorized privileges on the system and, in particular, from gaining root access.



System access threats

- Intruders
  - Masquerader (outsider)
  - Misfeasor (insider)
  - Clandestine user (outside or insider)
- Malicious software (malware)

Intruder:

- **Masquerader**: An individual who is not authorized to use the computer and who penetrates a system's access controls to exploit a legitimate user's account
- **Misfeasor:** A legitimate user who accesses data, programs, or resources for which such access is not authorized, or who is authorized for such access but misuses his or her privileges
- **Clandestine user:** An individual who seizes supervisory control of the system and uses this control to evade auditing and access controls or to suppress audit collection

Countermeasures: Intrusion Detection

- Intrusion detection systems are typically designed to detect human intruder and malicious software behaviour.
- May be host or network based
- Intrusion detection systems (IDS) typically comprise
  - Sensors
  - Analyzers
  - User Interface

RFC 2828 defines an IDS as

> A security service that monitors and analyzes system events for the purpose of finding, and providing real-time or near-real-time warning of, attempts to access system resources in an unauthorized manner.

- **Host-based IDS:** Monitors the characteristics of a single host and the events occurring within that host for suspicious activity
- **Network-based IDS:** Monitors network traffic for particular network segments or devices and analyzes network, transport, and application protocols to identify suspicious activity

An IDS comprises three logical components:

- **Sensors:** Sensors are responsible for collecting data. The input for a sensor may be any part of a system that could contain evidence of an intrusion. Types of input to a sensor include network packets, log files, and system call traces. Sensors collect and forward this information to the analyzer.
- **Analyzers:** Analyzers receive input from one or more sensors or from other analyzers. The analyzer is responsible for determining if an intrusion has occurred. The output of this component is an indication that an intrusion has occurred. The output may include evidence supporting the conclusion that an intrusion occurred. The analyzer may provide guidance about what actions to take as a result of the intrusion.
- **User interface:** The user interface to an IDS enables a user to view output from the system or control the behaviour of the system. In some systems, the user interface may equate to a manager, director, or console component.

Countermeasures: Authentication

- Two Stages:
  - Identification
  - Verification
- Four Factors:
  - Something the individual ***knows***
  - Something the individual ***possesses***
  - Something the individual ***is*** (static biometrics)
  - Something the individual ***does*** (dynamic biometrics)

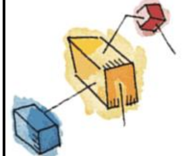
RFC 2828 defines user authentication as follows:

> The process of verifying an identity claimed by or for a system entity. An authentication process consists of two steps:
>
> •**Identification step:** Presenting an identifier to the security system. (Identifiers should be assigned carefully, because authenticated identities are the basis for other security services, such as access control service.)
>
> •**Verification step**: Presenting or generating authentication information that corroborates the binding between the entity and the identifier.

Be prepared to elaborate on each factor with examples. Point out that each has weaknesses, but strength comes with multiple factors (e.g. password and token)

Four Factors:

- Something the individual **knows:** Examples include a password, a personal identification number (PIN), or answers to a prearranged set of questions.
- Something the individual **possesses**: Examples include electronic keycards, smart cards, and physical keys. This type of authenticator is referred to as a token.
- Something the individual **is** (static biometrics): Examples include recognition by fingerprint, retina, and face.
- Something the individual **does** (dynamic biometrics): Examples include recognition by voice pattern, handwriting characteristics, and typing rhythm.



Countermeasures: Access Control

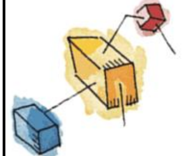
- A policy governing access to resources
- A security administrator maintains an authorization database
  - The access control function consults this to determine whether to grant access.
- An auditing function monitors and keeps a record of user accesses to system resources.

Access control implements a security policy that specifies who or what (e.g., in the case of a process) may have access to each specific system resource and the type of access that is permitted in each instance.

A security administrator maintains an authorization database that specifies what type of access to which resources is allowed for this user.

The access control function consults this database to determine whether to grant access.

An auditing function monitors and keeps a record of user accesses to system resources.



Countermeasures: Firewalls

- Traditionally, a firewall is a dedicated computer that:
  - interfaces with computers outside a network
  - has special security precautions built into it to protect sensitive files on computers within the network.

Firewalls can be an effective means of protecting a local system or network of systems from network-based security threats while at the same time affording access to the outside world via wide area networks and the Internet.

Traditionally, a firewall is a dedicated computer that interfaces with computers outside a network and has special security precautions built into it in order to protect sensitive files on computers within the network.

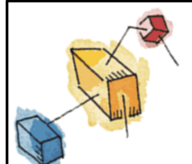
It is used to service outside network, especially Internet, connections and dial-in lines. Personal firewalls that are implemented in hardware or software, and associated with a single workstation or PC, are also common.

Roadmap

– 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.

Discuss process management in UNIX SVR4.

Progress indicator



Unix SVR4

System V Release 4

- Uses the model of fig3.15b where most of the OS executes in the user process
- System Processes - Kernel mode only
- User Processes
  - User mode to execute user programs and utilities
  - Kernel mode to execute instructions that belong to the kernel.

(b) OS functions execute within user processes

UNIX System V makes use of a simple but powerful process facility that is highly visible to the user.

System processes run in kernel mode and execute operating system code to perform administrative and housekeeping functions, such as allocation of memory and process swapping.

User processes operate in:

- user mode to execute user programs and utilities
- in kernel mode to execute instructions that belong to the kernel.

A user process enters kernel mode by issuing a system call, when an exception (fault) is generated, or when an interrupt occurs.

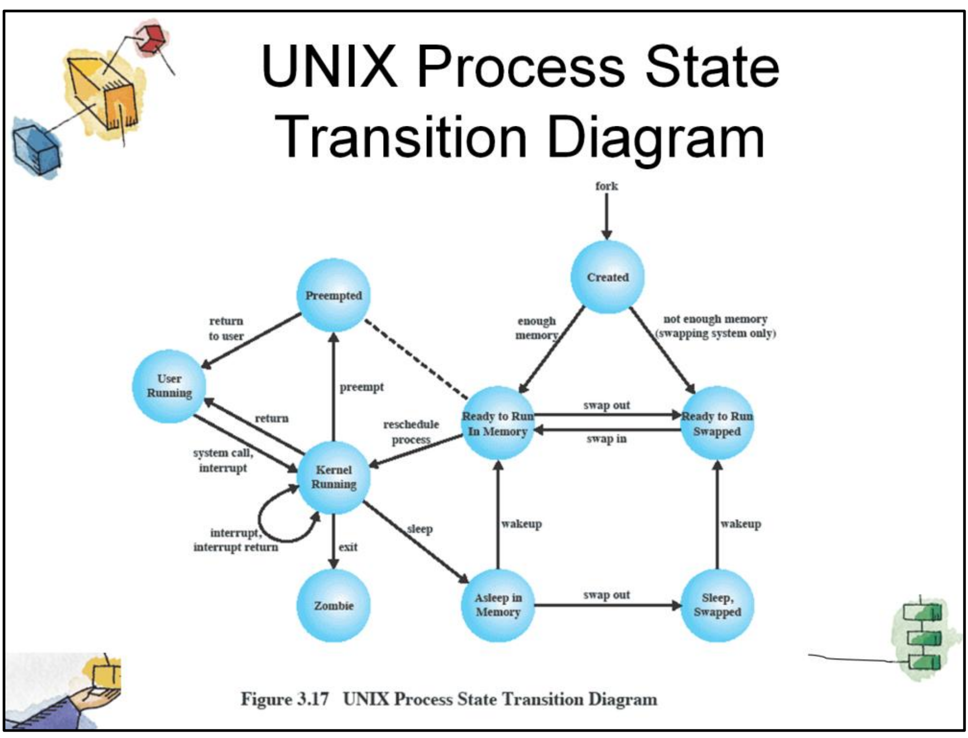


Figure 3.17 UNIX Process State Transition Diagram

A total of nine process states are recognized by the UNIX SVR4 operating system;

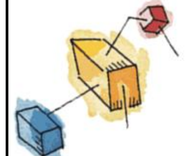
This figure is similar to Figure 3.9b, with the two UNIX sleeping states corresponding to the two blocked states.

Point out that:

- UNIX employs two Running states to indicate whether the process is executing in user mode or kernel mode.
- Two states are essentially the same: (Ready to Run, in Memory) and (Pre-empted) as indicated by the dotted line. The distinction is made to emphasize the way in which the pre-empted state is entered. When a process is running in kernel mode (as a result of a supervisor call, clock interrupt, or I/O interrupt), there will come a time when the kernel has completed its work and is ready to return control to the user program. At this point, the kernel may decide to pre-empt the current process in favour of one that is ready and of higher priority. In that case, the current process moves to the pre-empted state. However, for purposes of dispatching, those processes in the pre-empted state and those in the Ready to Run, in Memory state form one queue.

UNIX Process States

User Running	Executing in user mode.
Kernel Running	Executing in kernel mode.
Ready to Run, in Memory	Ready to run as soon as the kernel schedules it.
Asleep in Memory	Unable to execute until an event occurs; process is in main memory (a blocked state).
Ready to Run, Swapped	Process is ready to run, but the swapper must swap the process into main memory before the kernel can schedule it to execute.
Sleeping, Swapped	The process is awaiting an event and has been swapped to secondary storage (a blocked state).
Preempted	Process is returning from kernel to user mode, but the kernel preempts it and does a process switch to schedule another process.
Created	Process is newly created and not yet ready to run.
Zombie	Process no longer exists, but it leaves a record for its parent process to collect.



A Unix Process

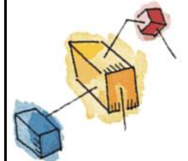
- A process in UNIX is a set of data structures that provide the OS with all of the information necessary to manage and dispatch processes.
- See Table 3.10 which organizes the elements into three parts:
  - user-level context,
  - register context, and
  - system-level context.

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.

Process Creation cont…

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 to Run state.
6. Returns the ID number of the child to the parent process, and a 0 value to the child process.



After Creation

- 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.

When the kernel has completed these functions it can do one of the following, as part of

the dispatcher routine:

- Stay in the parent process. Control returns to user mode at the point of the fork call of the parent.
- Transfer control to the child process. The child process begins executing at the same point in the code as the parent, namely at the return from the fork call.
- Transfer control to another process. Both parent and child are left in the Ready to Run state.