目录 Contents

Process Description and Control

2.1 What is a Process?

1 Why we need processes?

- The most central concept in any operating system is the process: an abstraction of a running program. Everything else hinges on this concept, and it is important that the OS designer understand this concept well.
- **Program**: A sequence of instructions, codes, and data that follow one another in strict order in time.
- Sequential execution of programs: the process by which a program monopolizes the processor until the final result is obtained is called sequential execution of programs.

Why we need processes.

- The program execution environment in a multiprogramming system(Basic characteristics of modern OS):
  - Concurrency(并发): multi-program runing in a same time interval
  - Share(共享): resources sharing with multi-user
- **Concurrency**: the overlap of execution time between logically independent programs or program segments, i.e. the execution of one segment has not yet finished while another segment has begun to run.
- The processor is switched among multiple applications so all will appear to be progressing.

Why we need processes..

- The principal tool available in developing the early multiprogramming and mutiuser interactive systems was the interrupt. But the design of the system software to coordinate these various activities of programs turned out to be remarkably difficult.
- With many jobs in progress at any one time, each of which involved numerous steps to be performed in sequence, it became impossible to analyze all of the possible combinations of sequences of events unless sufficient system control information is added to the program itself, that is, similar to jobs, relevant control information, is packaged into the program to track the execution, this is a process.

2.1 What is a Process?

2 What is a process?

- A **process** can be defined as:
  - a program in execution
  - an instance of a running program
  - the entity(实体) 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(当前状态), and an associated set of system resources

What is a process.

- A process contains **three** components:
  - an executable program
  - the associated data needed by the program (variables, work space, buffers, etc.)
  - the execution context (or "process state") of the program
- This last element is essential.
  - It is the interal data by which the operating system is able to supervise and control the process. This data is separated from the process, because all of the OS has information not permitted to the process
  - includes the contents of the various processor registers
  - includes information such as the priority of the process and whether the process is waiting for the completion of a particular I/O event

Typical Process Implementation

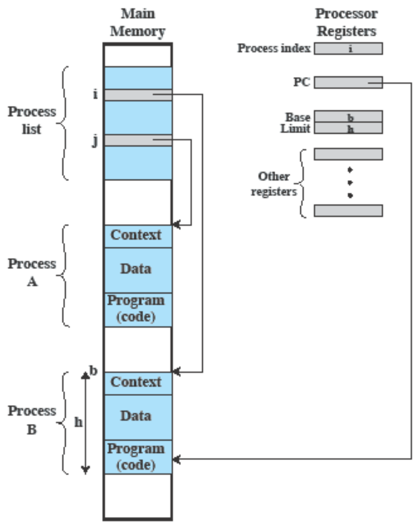
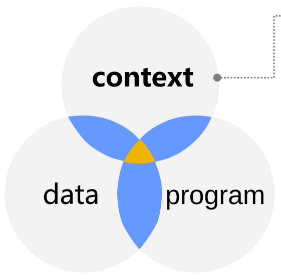


Figure 2.8 Typical Process Implementation

Process Control Blocks/Process Lists

- Two essential elements of a process are:
  - Program code: which may be shared with other processes that are executing the same program
  - A set of data associated with that code
- When the processor begins to execute the program code, we refer to this executing entity as a process.



1. Identifier
2. State
3. Priority
4. Program counter
5. Memory pointers
6. Context data(register data)
7. I/O status information
8. Accounting information

Process Control Blocks/Process Lists.

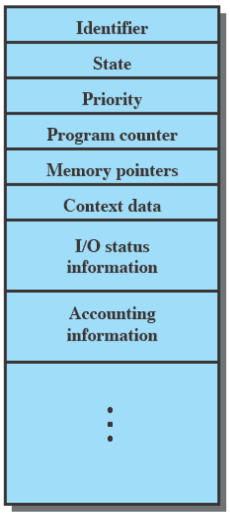


Figure 3.1 Simplified Process Control Block

- Process Control Block
  - Contains the process elements
  - It is possible to interrupt a running process and later resume execution as if the interruption had not occurred
  - Created and managed by the operating system
  - Key tool that allows support for multiple processes

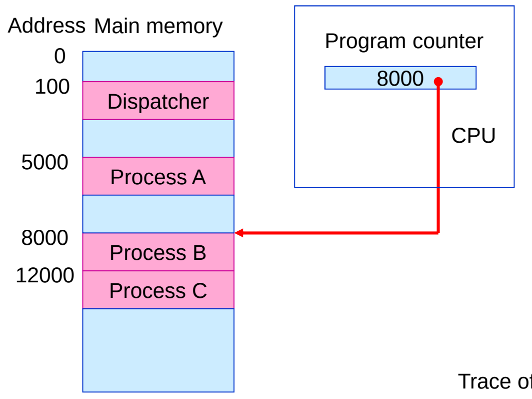
2.2 Process States

- 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.
- Process is introduced to describe the execution process of the program, from the system's point of view, it is to describe the behavior of the program in the execution.
- **Process life cycle**: The life cycle of a process is the entire process from creation to termination.
- **The state of the process** refers to the different stages in the process life cycle.
  - new state: for a program, to be executed, a process, or task, is created for that program.

Trace and Dispatcher

- **Trace(轨迹)**: We can characterize the behavior of an individual process by listing the sequence of instructions that execute for that process. Such a listing is referred to as a trace of the process.
  - the behavior of the processor can be characterized by showing how the traces of the various processes are interleaved
- **Dispatcher(分派器)**: small program that switches the processor from one process to another

Trace of Processes

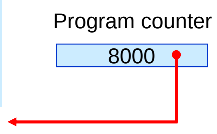


Trace of Process A	B trace	C trace
5000	8000	12000
5001	8001	12001
5002	8002	12002
5003	8003	12003
5004		12004
5005		12005
5006		12006
5007		12007
5008		12008
5009		12009
5010		12010
5011		12011

5000 = starting address of program of Process A
8000 = starting address of program of Process B
12000 = starting address of program of Process C

Trace of Processes.

1	5000	Process A
2	5001	
3	5002	
4	5003	
5	5004	
6	5005	
- - - - -	- - - - -	Timeout
7	100	Dispatcher
8	101	
9	102	
10	103	
11	104	
12	105	
13	8000	Process B
14	8001	
15	8002	
16	8003	
- - - - -	- - - - -	I/O request
17	100	
18	101	
19	102	
20	103	
21	104	
22	105	
23	12000	Process C
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	Process A
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	Process C
48	12007	
49	12008	
50	12009	
51	12010	
52	12011	
- - - - -	- - - - -	Timeout

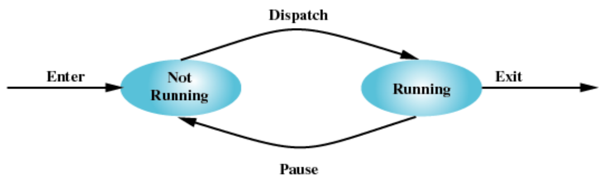


100 = starting address of dispatcher program
Shaded areas indicate execution of dispatcher process;
first and third columns count instruction cycles;
second and fourth columns show address of instruction being executed

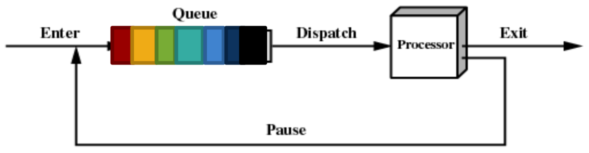
2.2 Process States

1 A Two-State Process Model

- A process may be in one of two states:
  - running
  - not-running



(a) State transition diagram



(b) Queuing diagram

2.2 Process States

2 The Creation and Termination of Processes

- The **Creation** of processes: When a new process is to be added to those currently being managed, the OS builds the data structures that are used to manage the process and allocates address space in main memory to the process. These actions constitute the creation of a new process.
- The **Termination** of processes: There must be a means for a process to indicate its completion.
  - A batch job should include a HALT instruction or an explicit OS service call for termination
  - For an interactive application, the action of the user will indicate when the process is completed (e.g. log off, quitting an application)

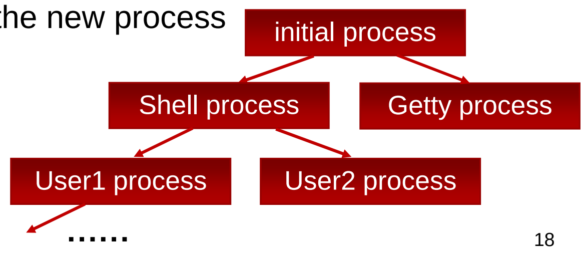
(1) Reasons for Process Creation

New batch job	The operating system is provided with a batch job control stream, usually on tape or disk. When the operating system is prepared to take on new work, it will read the next sequence of job control commands.
Interactive logon	A user at a terminal logs on to the system.
Created by OS to provide a service	The operating system can create a process to perform a function on behalf of a user program, without the user having to wait (e.g., a process to control printing).
Spawned by existing process	For purposes of modularity or to exploit parallelism, a user program can dictate the creation of a number of processes.

Table 3.1 Reasons for Process Creation

Reasons for Process Creation.

- Traditionally, the OS created all processes in a way that was transparent to the user or application program.
- It can be useful to allow one process to cause the creation of another.
- Process spawning: when the OS creates a process at the explicit request of another process
- **Parent process**: is the original, creating, process
- **Child process**: is the new process



(2) Reasons for Process Temination

Table 3.2 Reasons for Process Termination

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.

Reasons for Process Temination.

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 (for example, 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.

2.2 Process States

3 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/Waiting**: A process that cannot execute until some event occurs, such as the completion of an I/O operation.
- **New**: A process that has just been created but has not yet been admitted to the pool of executable processes by the OS. Typically, a new process has not yet been loaded into main memory, although its process control block has been created.
- **Exit**: A process that has been released from the pool of executable processes by the OS, either because it halted or because it aborted for some reason.

New and Exit

- **New**: information concerning the process that is needed by the OS is maintained in control tables in main memory. However, the process itself is not in main memory, and no space has been allocated for the data associated with that program.
  - the OS performs the necessary housekeeping chores.An identifier is associated with the process. Any tables that will be needed to manage the process are allocated and built.
- **Exit**: a process exits a system in two stages
  - Termination moves the process to the exit state. The process is no longer eligible for execution. The tables and other information associated with the job are temporarily preserved by the OS, which provides time for auxiliary or support programs extract any needed information.
  - Once these programs have extracted the needed information, the OS no longer needs to maintain any data relating to the process and the process is deleted from the system.

Five-State Process and Transition Model

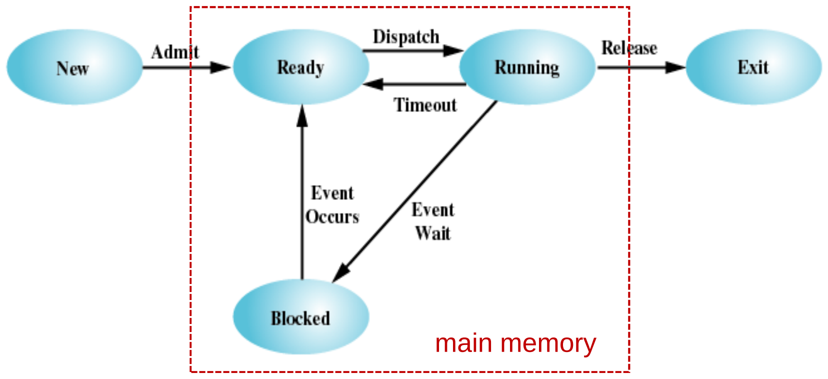


Figure 3.6 Five-State Process Model

Possible Transitions

- **Null→New:** A new process is created to execute a program. This event occurs for any of the reasons listed in Table 3.1.
- **New→Ready**: The OS will move a process from the New state to the Ready state when it is prepared to take on an additional process.
- **Running→Exit:** The currently running process is terminated by the OS if the process indicates that it has completed, or if it aborts. See Table 3.2.
- **Ready→Exit and Blocked→Exit:** For clarity, this transition is not shown on the state diagram. In some systems, a parent may terminate a child process at any time. Also, if a parent terminates, all child processes associated with that parent may be terminated.

Possible Transitions.

- **Ready→Running:** When it is time to select a process to run, the OS chooses one of the processes in the Ready state. This is the job of the scheduler or dispatcher.
- **Running→Ready:** The most common reason for this transition is that the running process has reached the maximum allowable time for uninterrupted execution; Another importance reason is the case in which the OS assigns different levels of priority to different processes. Thus a process running at a higher priority level may interrupt process at a lower priority. Or a process may voluntarily release control of the processor.
- **Running→Blocked:** A process is put in the Blocked state if it requests something for which it must wait.
- **Blocked→Ready:** A process in the Blocked state is moved to the Ready state when the event for which it has been waiting occurs.

States of the Three Processes

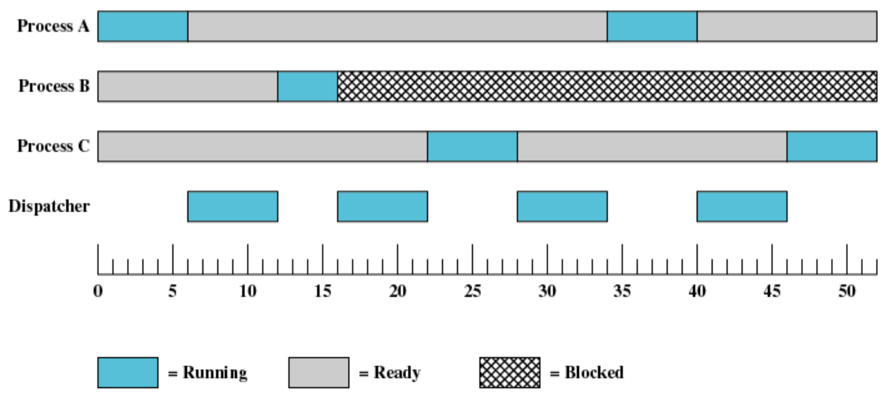
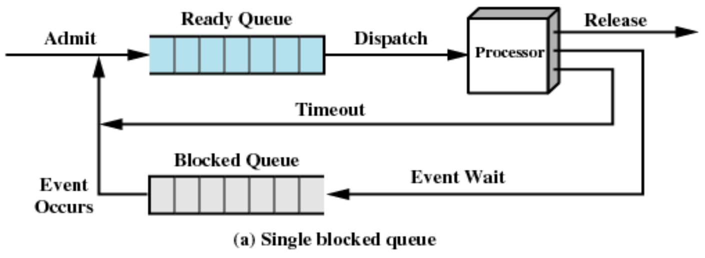


Figure 3.7 Process States for Trace of Figure 3.4

Queuing Model — Single Blocked Queues



- The reason for this elaborate machinery is that I/O activities are much slower than computation and therefore the processor in a uniprogramming system is idle most of the time.

- In this case, memory holds multiple processes and that the processor can move to another ready-process when one process is blocked.

Queuing Model — Multiple Blocked Queues

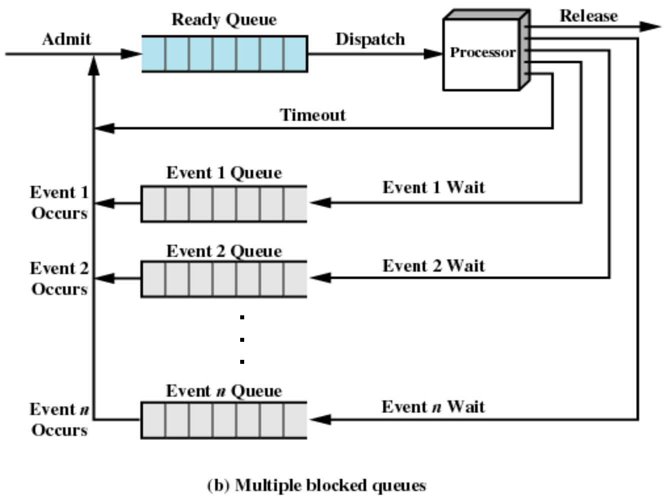


Figure 3.8 Queuing Model for Figure 3.6

2.2 Process States

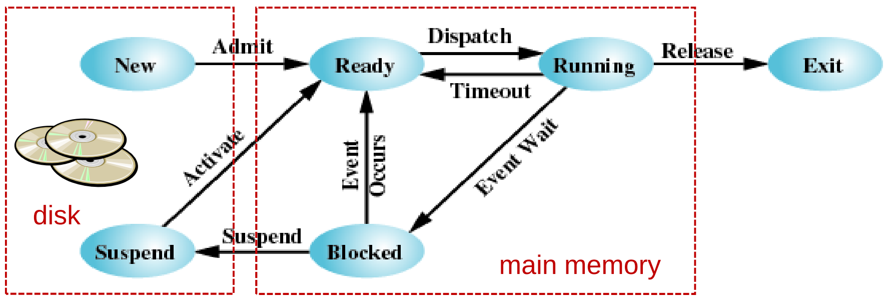
4 Suspended Process

- Processor is much faster than I/O so all processes in main memory could be waiting for I/O, that is, there are no enough memory to hold more ready-processes in main memory.

- when none of the processes in main memory is in the Ready state, the OS swaps one of the blocked processes out on to disk into a **suspend** queue.

- **Swapping(交换)**: to free up more main memory, move part or all of a process from main memory to disk.
- **Suspend**: Blocked state becomes suspend state(挂起) when swapped to disk.

One Suspend State



(a) With One Suspend State

- Two new **Suspend** states:
  - **Blocked/Suspend**(阻塞/挂起);
  - **Ready/Suspend**(就绪/挂起)

Two Suspend States

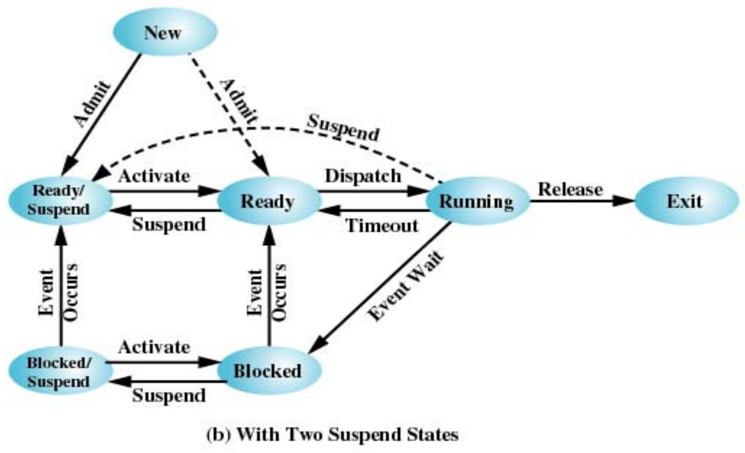


Figure 3.9 Process State Transition Diagram with Suspend States

Characteristics of a Suspended Process

- A suspended process can be defined as having the following characteristics:
  - The process is not immediately available for execution.
  - The process was placed in a suspended state by an agent: either itself, a parent process, or the OS, for the purpose of preventing its execution.
  - The process may or may not be waiting on an event.
  - The process may not be removed from this state until the agent explicitly orders the removal.

Reasons for Process Suspension

Table 3.3 Reasons for Process Suspension

Swapping	The operating system needs to release sufficient main memory to bring in a process that is ready to execute.
Other OS reason	The operating system may suspend a background or utility process or a process that is suspected of causing a problem.
Interactive user request	A user may wish to suspend execution of a program for purposes of 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 interval.
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 descendents.

2.3 Process Description

Processes and Resources

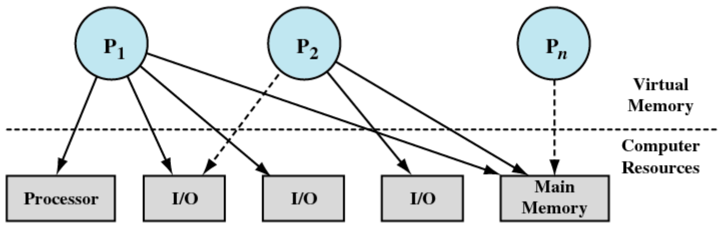


Figure 3.10 Processes and Resources (resource allocation at one snapshot in time)

- What information does the operating system need to control processes and manage resources for them?

2.3 Process Description

1 Operating System Control Structures

- OS must have information about the current status of each process and resource-need

- Universal approach: OS manages the tables that are constructed for each entity.

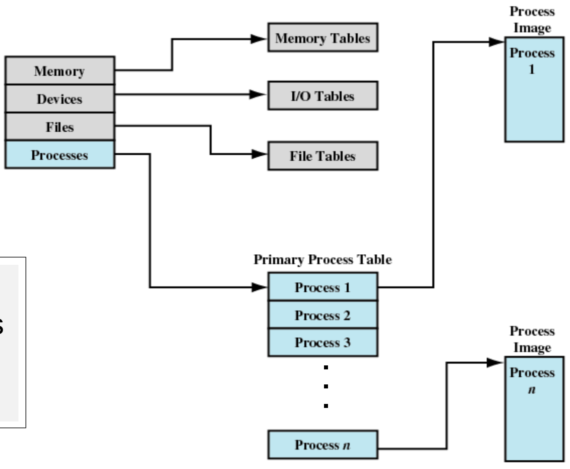


Figure 3.11 General Structure of Operating System Control Tables

(1) Memory Tables

- Used to keep track of both main (real) and secondary (virtual) memory.
- Processes are maintained on secondary memory using some sort of virtual memory or simple swapping mechanism.
- Must include:
  - 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

(2) I/O Tables

- 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:
  - I/O device is available or assigned
  - Status of I/O operation
  - Location in main memory being used as the source or destination of the I/O transfer

(3) File Tables

- 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.
- These tables provide information about:
  - Existence of files
  - Location on secondary memory
  - Current status
  - Other attributes

(4) Process Tables

- Must be maintained to manage processes.
- There must be some reference to memory, I/O, and files, directly or indirectly.
- The tables themselves must be accessible by the OS and therefore are subject to memory management.
- These tables provide information about:
  - Where process is located
  - **Attributes** in the process control block
    - Program(Code/Text)
    - Data
    - Stack

2.3 Process Description

2 Process Control Structures

- To manage and control a process the OS must know:
  - where the process is located
  - the attributes of the process that are necessary for its management

- What is the physical manifestation(表示) of a process in a computer?

  - **sufficient memory** to hold the programs and data
  - **a stack** used to keep track of procedure calls and parameter passing between procedures
  - **process control block**, a collection of attributes used by the OS for process control

(1) Process Image

- **Process Image(进程映像)**: a collection of program, data, stack and attributes.

Table 3.4 Typical Elements of a Process Image

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.
System Stack	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.
Process Control Block	Data needed by the operating system to control the process (see Table 3.5).

(2) Process Location

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

- PI usually in disk
- management: at least a portion in main memory
- execution: the entire PI must in main memory.

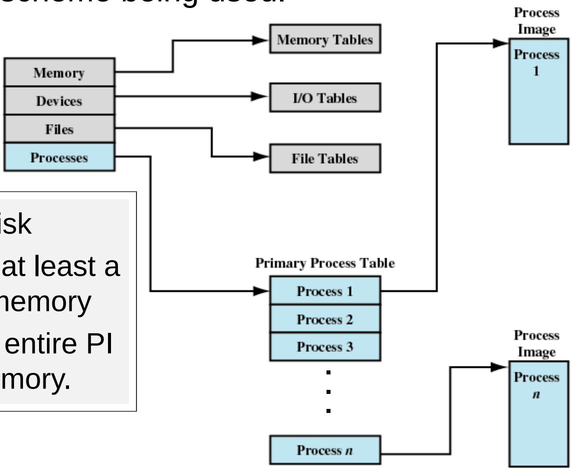
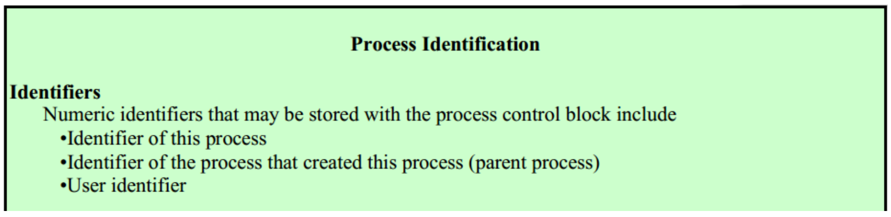


Figure 3.11 General Structure of Operating System Control Tables

(3) Process Attributes

Process identification

- Each process is assigned a unique numeric **identifier** otherwise there must be a mapping that allows the OS to locate the appropriate tables based on the process identifier
- Many of the tables controlled by the OS may use process identifiers to cross-reference process tables



Process Identification

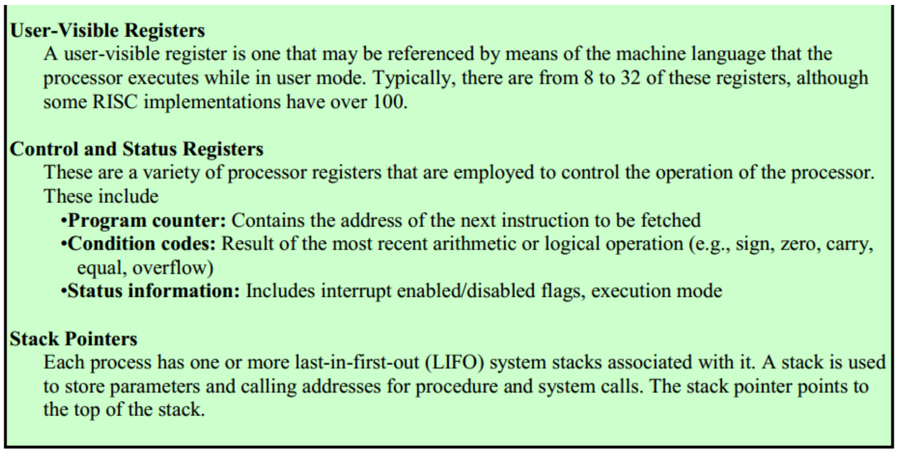
Identifiers

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

- Consists of the contents of processor registers.



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

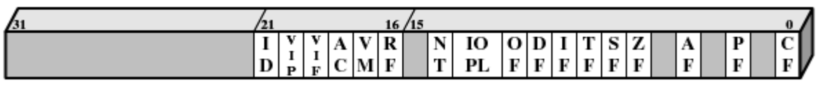
- •**Program counter:** Contains the address of the next instruction to be fetched
- •**Condition codes:** Result of the most recent arithmetic or logical operation (e.g., sign, zero, carry, equal, overflow)
- •**Status information:** Includes interrupt enabled/disabled flags, execution mode

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.

Processor State Information.

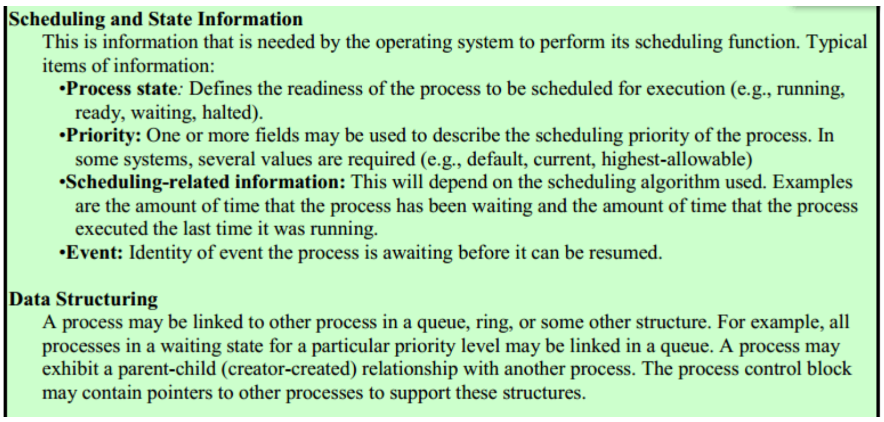
- Program status word (PSW) contains condition codes plus other status information.



ID	=	Identification flag		DF	=	Direction flag	方向
VIP	=	Virtual interrupt pending		IF	=	Interrupt enable flag	
VIF	=	Virtual interrupt flag		TF	=	Trap flag	
AC	=	Alignment check		SF	=	Sign flag	符号
VM	=	Virtual 8086 mode		ZF	=	Zero flag	
RF	=	Resume flag	恢复	AF	=	Auxiliary carry flag	辅助进位
NT	=	Nested task flag	嵌套任务	PF	=	Parity flag	奇偶
IOPL	=	I/O privilege level		CF	=	Carry flag	进位
OF	=	Overflow flag					

Figure 3.12 Pentium II EFLAGS Register

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.

Process Control Information.

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 特权
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 Images in Virtual Memory

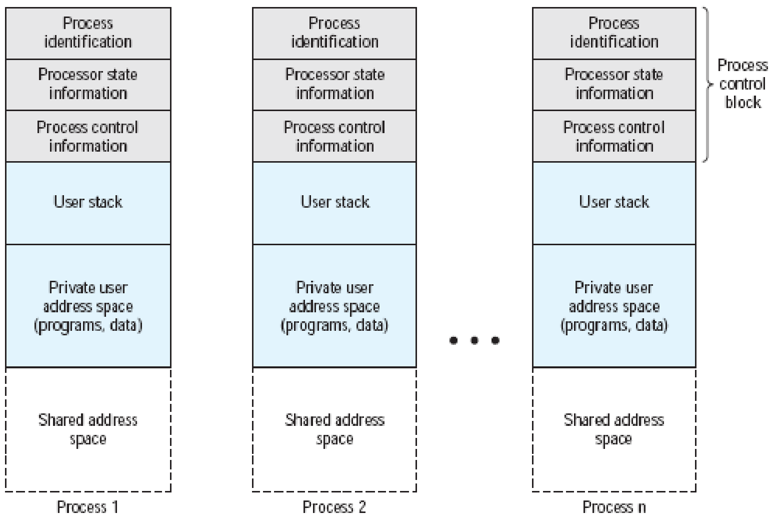
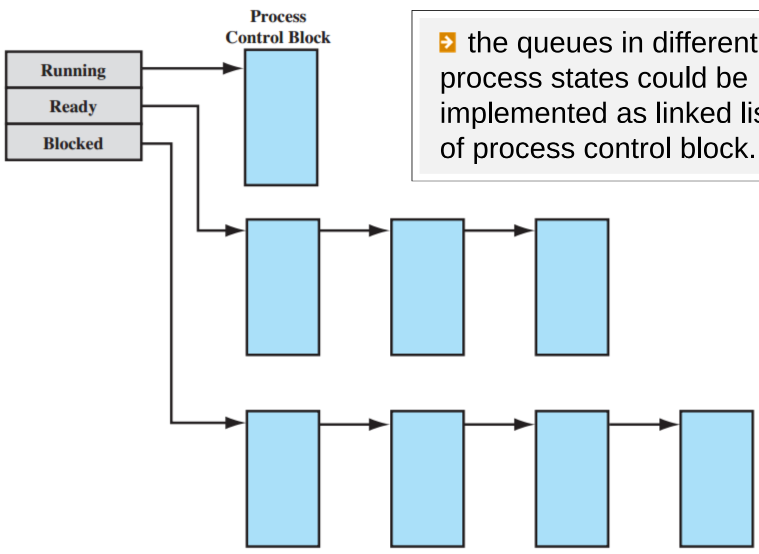


Figure 3.13 User Processes in Virtual Memory

Process List Structures



- the queues in different process states could be implemented as linked lists of process control block.

Role of the Process Control Block

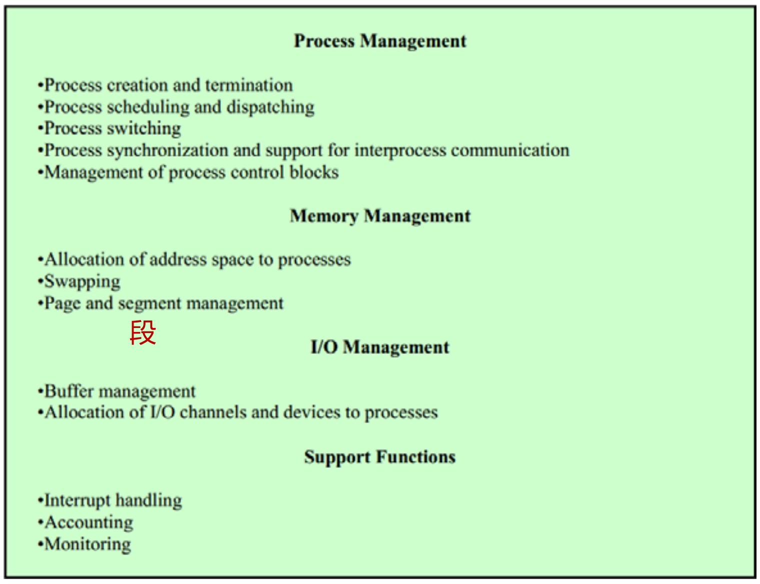
- The most important data structure in an OS
  - contains all of the information about a process that is needed by the OS
  - blocks are read and/or modified by virtually every module in the OS
  - defines the state of the OS
- Difficulty is not access, but protection
  - a bug in a single routine(例程) 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(语义) of the process control block could affect a number of modules in the OS

2.4 Process Control

1 Modes of Execution

- User Mode
  - less-privileged mode
  - user programs typically execute in this mode
- System Mode(Control Mode/Kernel Mode)
  - more-privileged mode
  - also referred to as control mode or kernel mode
  - kernel of the operating system

Typical Functions of an OS Kernel



Process Management

•Process creation and termination
•Process scheduling and dispatching
•Process switching
•Process synchronization and support for interprocess communication
•Management of process control blocks

Memory Management

•Allocation of address space to processes
•Swapping
•Page and segment management

段

I/O Management

•Buffer management
•Allocation of I/O channels and devices to processes

Support Functions

•Interrupt handling
•Accounting
•Monitoring

2.4 Process Control

2 Process Creation

1. assigns a unique process identifier
2. allocates space for the process
3. initializes the process control block
4. sets the appropriate linkages
5. creates or expands other data structures

2.4 Process Control

3 Process Switching

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

When to Switch a Process?

- **Interrupt**: Due to some sort of event that is external to and independent of the currently running process
  - clock interrupt: process has executed for the maximum allowable time slice. **Time slice**: the maximum amount of time that a process can execute before being interrupted
  - I/O interrupt
  - memory fault

When to Switch a Process?.

- **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
- **Supervisor call/system call**
  - such as file open

Mode Switching

- If an interrupt is pending the processor:
  - sets the program counter to the starting address of an interrupt handler program
  - switches from user mode to kernel mode so that the interrupt processing code may include privileged instructions
- **Mode Switching**: A mode switch may occur without changing the state of the process that is currently in the Running state. In that case, the context saving and subsequent restoral involve little overhead.

- 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

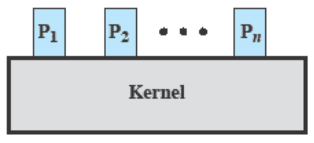
How to Switch a Process?

- Save context of processor including program counter and other registers
- Update the process control block of the process that is currently in the Running state
- Move process control block to appropriate queue – ready; blocked;
- Select another process for execution
- Update the process control block of the process selected
- Update memory-management data structures
- Restore context of the selected process

2.5 Execution of the Operating System

Non-process Kernel

- Non-process Kernel
  - Execute kernel outside of any process
  - Operating system code is executed as a separate entity that operates in privileged mode

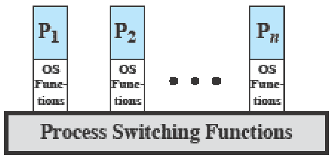


(a) Separate kernel

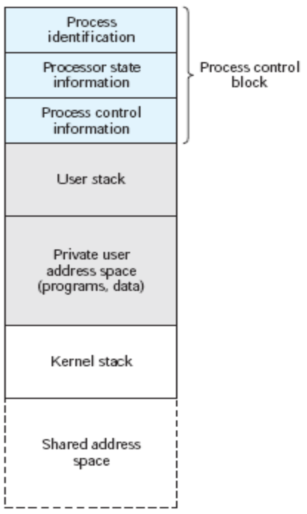
Figure 3.15 Relationship Between Operating System and User Processes

Execution Within User Processes

- Execution Within User Processes
  - OS software within context of a user process, as a collection of routines the user calls to perform functions
  - Process executes in privileged mode when executing OS code
  - When a process switch is to occur, the control is passed to a process switching routine



(b) OS functions execute within user processes

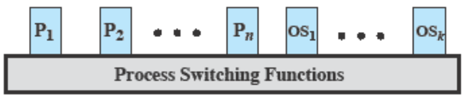


Process identification
Processor state information
Process control information
Process control block
User stack
Private user address space (programs, data)
Kernel stack
Shared address space

Figure 3.16 Process Image: Operating System Executes within User Space

Process-Based Operating System

- Process-Based Operating System
  - Implement operating system as a collection of system processes
  - Useful in multi-processor or multi-computer environment



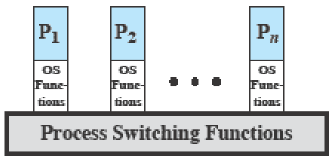
(c) OS functions execute as separate processes

Figure 3.15 Relationship Between Operating System and User Processes

2.6 UNIX SVR4 Process Management

Relationship Between OS and User Processes

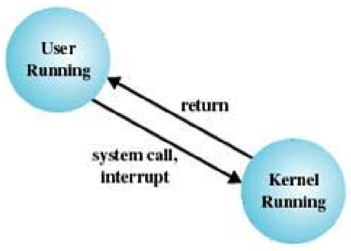
- The operating system executes within the environment of a user process



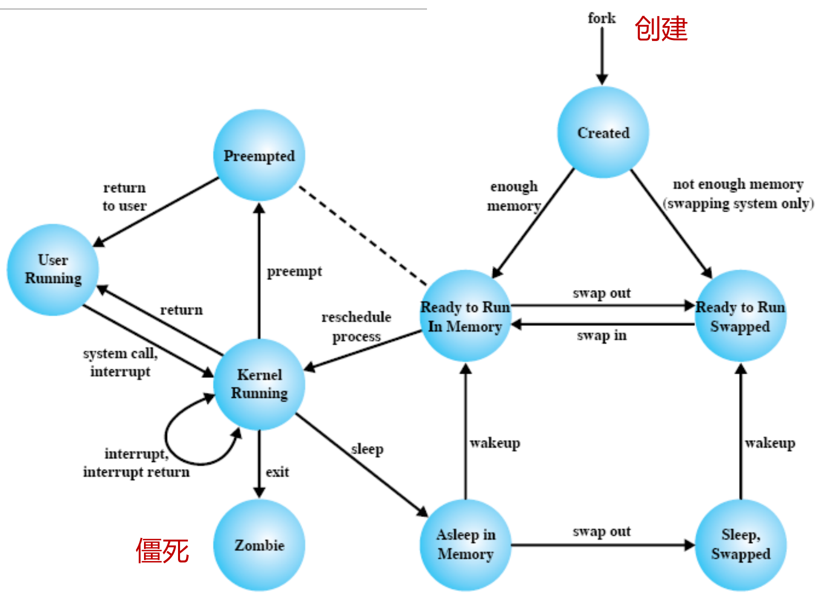
(b) OS functions execute within user processes

Two Modes of UNIX Process Execution

- User mode (用户态)
  - Less-privileged mode
  - User programs typically execute in this mode
- kernel mode(核心态)
  - More-privileged mode
  - Execution of the operating system



UNIX Process State Transition Diagram



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.

Terminology

- concurrency
- process
- trace
- process creation
  - parent process; child process
- process termination
- process states
  - new; running; ready; blocked; suspended; exit
  - ready queue; blocked queues (event queues)
- process transitions
  - admit; dispatch; time out; event wait; event occurs; release; activate; suspend

Terminology.

- process control block (PCB)
- process image
- process identifier (pid)
- processor state
- process switch; mode switch
- interrupt; trap