Introdução aos Processos

Chapter 3, Livro do William Stallings

Sistemas Operativos, 2004-2005

What is a Process?

- A *process* is a program in execution.
- A process needs certain resources, including CPU time, memory, files, and I/O devices, to accomplish its task.
- A process includes:
  - Code section
  - Program counter
  - Stack
  - Data section
- The operating system is responsible for the following activities:
  - Process creation and deletion.
  - Process suspension and resumption.
  - Process synchronization
  - Process communication

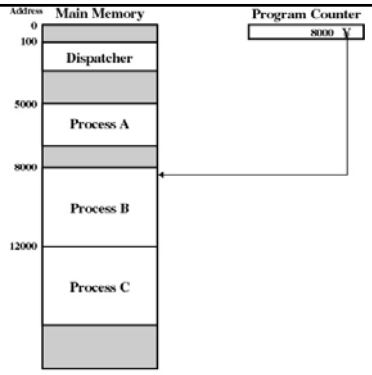


Figure 3.1 Snapshot of Example Execution (Figure 3.3) at Instruction Cycle 13

(a) Trace of Process A	(b) Trace of Process B	(c) Trace of Process C
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

Figure 3.2 Traces of Processes of Figure 3.1

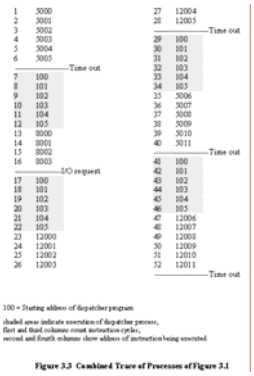
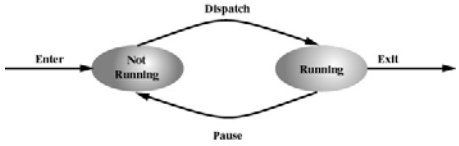


Figure 3.3 Combined Trace of Processes of Figure 3.1

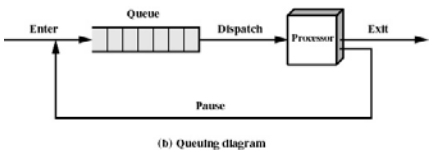
Two-State Process Model

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



(a) State transition diagram

Not-Running Process in a Queue



(b) Queuing diagram

Process Creation

- Submission of a batch job.
- User logs on.
- Provide a service such as printing.
- Process creates another process.

Process Termination

- Batch job issues *Halt* instruction.
- User logs off.
- Quit an application.
- Error and fault conditions.

Reasons for Process Termination

- Normal completion
- Time limit exceeded
- Memory unavailable
- Bounds violation
- Protection error
  - example write to read-only file
- Arithmetic error
- Time overrun
  - process waited longer than a specified maximum for an event

Reasons for Process Termination

- I/O failure
- Invalid instruction
  - happens when try to execute data
- Privileged instruction
- Data misuse
- Operating system intervention
  - such as when deadlock occurs
- Parent terminates so child processes terminate
- Parent request

Processes

- Not-running
  - ready to execute
- Blocked
  - waiting for I/O
- Dispatcher cannot just select the process that has been in the queue the longest because it may be blocked.

A Five-State Model

- Running
- Ready
- Blocked
- New
- Exit

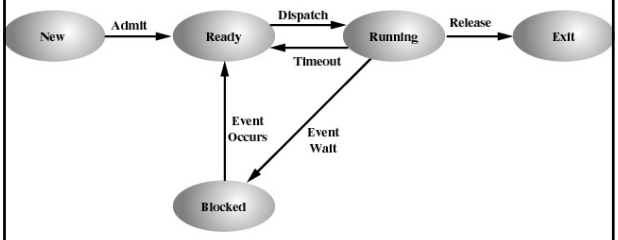


Figure 3.5 Five-State Process Model

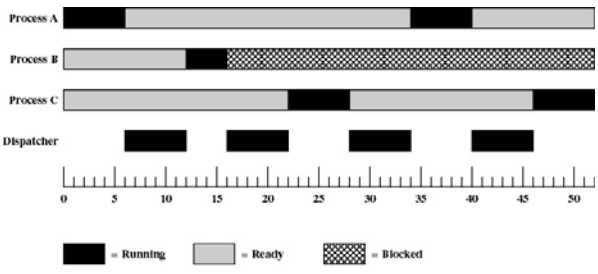
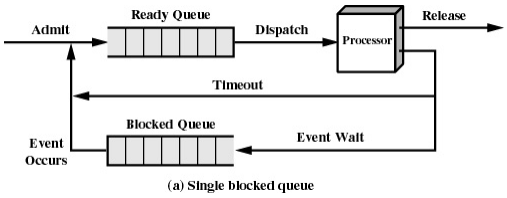
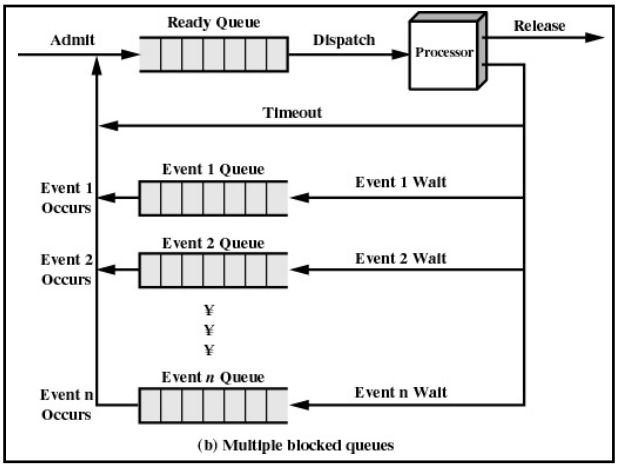


Figure 3.6 Process States for Trace of Figure 3.3

Using Two Queues



(a) Single blocked queue

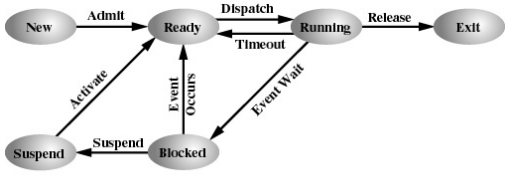


(b) Multiple blocked queues

Suspended Processes

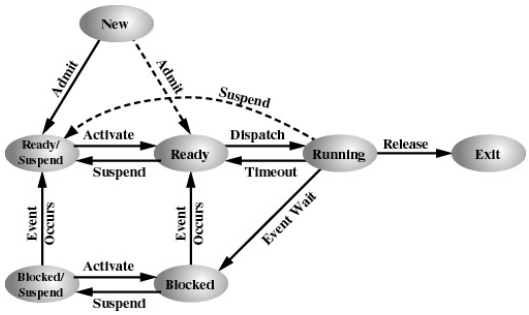
- Processor is faster than I/O so all processes could be waiting for I/O.
- Swap these processes to disk to free up more memory.
- Blocked state becomes suspend state when swapped to disk.
- Two new states:
  - Blocked, suspend
  - Ready, suspend

One Suspend State



(a) With One Suspend State

Two Suspend States



(b) With Two Suspend States

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.

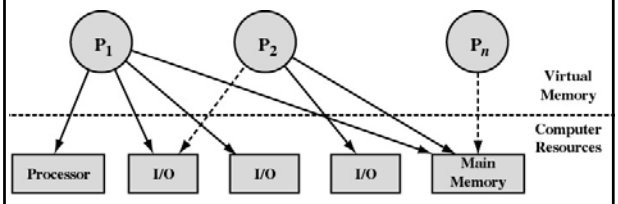


Figure 3.9 Processes and Resources (resource allocation at one snapshot in time)

Que informação deve ser mantida pelo Sistema Operativo para controlar os processos e gerir os recursos usados?

Restart here...

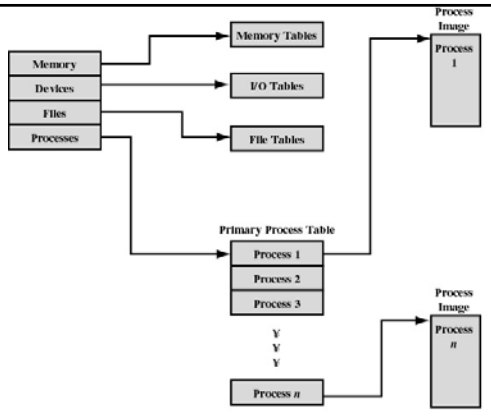


Figure 3.10 General Structure of Operating System Control Tables

Memory Tables

- 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.

I/O Tables

- 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.

File Tables

- Existence of files
- Location on secondary memory
- Current Status
- Attributes
- Sometimes this information is maintained by a file-management system

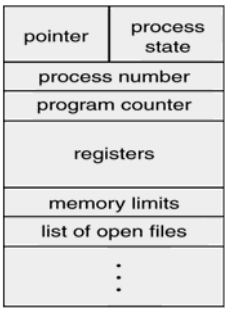
Process Table

- Process location.
- Attributes necessary for its management:
  - Process ID
  - Process state
  - Location in memory

Process Location

- Process includes set of programs to be executed
  - Data locations for local and global variables
  - Any defined constants
  - Stack
- **Process control block**
  - Collection of attributes
- Process image
  - Collection of program, data, stack, and attributes

Process Control Block (PCB)



Process Control Block

- Process identifiers (pid, ppid, uid)
- Process state
- Program counter
- CPU registers
- CPU scheduling information
- Memory-management information
- Resource ownership
- Accounting information
- I/O status information
- Stack pointers

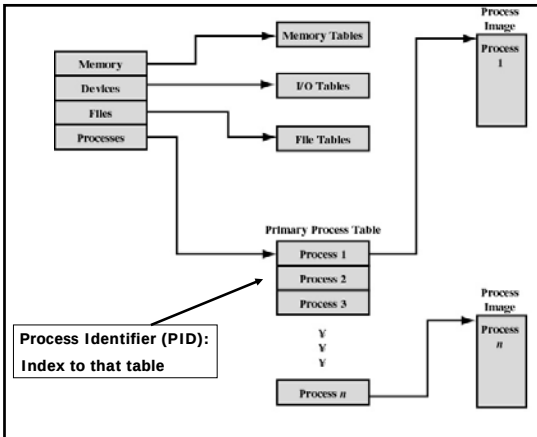


Figure 3.10 General Structure of Operating System Control Tables

Process Control Block

- Is the most important data structure in an operating system.
- Set of “Process Control Blocks” defines the state of the operating system.

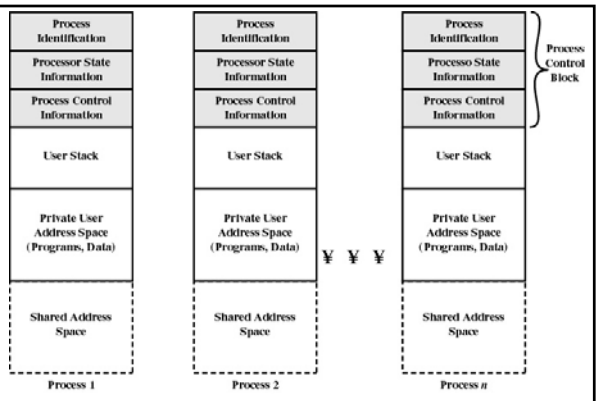


Figure 3.12 User Processes in Virtual Memory

Modes of Execution

User-mode vs Kernel-mode

Modes of Execution

- **User mode**
  - Less-privileged mode
  - User programs execute in this mode
- **System mode, control mode, or kernel mode**
  - More-privileged mode
  - Kernel of the operating system

Modes of Execution

- Kernel Mode: it is necessary to protect the operating system data structures and tables from interference by user programs.
- There is a bit in the Program Status Word that indicates the mode of execution.
- When a user makes a call to an operating system service the mode is set to kernel mode.

Typical Functions of an Operating System Kernel

Process Creation

- Assign a unique process identifier.
- Allocate space for the process.
- Initialize process control block.
- Set up appropriate linkages
  - Ex: add new process to linked list used for scheduling queue
- Create of expand other data structures
  - Ex: maintain an accounting file

When to Switch a Process

- Clock interrupt
  - process has executed for the maximum allowable time slice
- I/O interrupt
- Memory fault
  - memory address is in virtual memory so it must be brought into main memory

When to Switch a Process

- Trap
  - error occurred
  - may cause process to be moved to Exit state
- Supervisor call
  - such as file open; this call results in a transfer to a routine that is part of the operating system code.

Process Switching

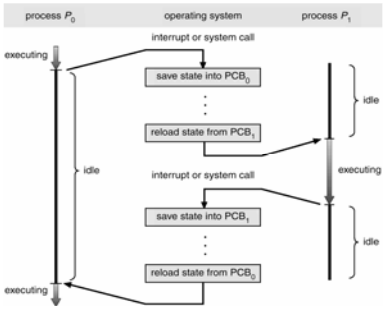
- The processor checks if there is any interrupt signal. If an interrupt is pending the processor does the following:
  - Save the context of the current program.
  - Set the PC to the address of the interrupt-handler routine.
  - Switch from user-mode to kernel-mode (the interrupt routine can execute privileged instructions).

@ Context Switch

- When a process is running some of the information context 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.

Process Switch != Mode Switch

Context Switch



Context-switch time is overhead.

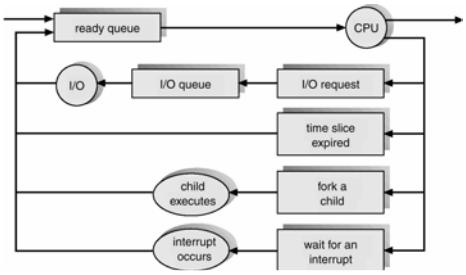
Change of Process State

1. Save context of processor including program counter and other registers.
2. Update the process control block of the process that is currently running.
3. Move process control block to appropriate queue - ready, blocked.
4. Select another process for execution.

Change of Process State

5. Update the process control block of the process selected.
6. Update memory-management data structures.
7. Restore context of the selected process.

Representation of Process Scheduling



I/O and CPU Bound

- Processes can be described as either:
  - ***I/O-bound process*** – spends more time doing I/O than computations, many short CPU bursts.
  - ***CPU-bound process*** – spends more time doing computations; few very long CPU bursts.

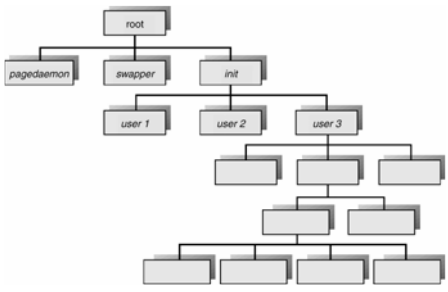
Process Creation

- Parent process create children processes, which, in turn create other processes, forming a tree of processes.
- Resource sharing:
  - Children share subset of parent's resources.
- Execution
  - Parent and children execute concurrently.
  - Parent waits until children terminate.

Process Creation (Cont.)

- Address space:
  - Child duplicate of parent (unix).
  - Child has a program loaded into it (windows).
- UNIX examples
  - **fork** system call creates new process.
  - **exec** system call used after a **fork** to replace the process' memory space with a new program.
- Microsoft: CreateProcess()

Processes Tree on a UNIX System



Process Termination

- Process executes last statement and asks the operating system to decide it (**exit**).
- Parent may terminate execution of children processes (**abort**).
  - Parent is exiting.
    - Operating system does not allow child to continue if its parent terminates.
    - If parent exits: temporary parent (init process)

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.

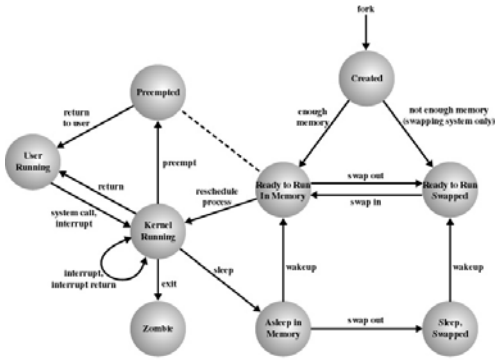


Figure 3.16 UNIX Process State Transition Diagram