Operating System

A program that controls **the execution** of application programs and acts as an **standardized interface** between applications and hardware.

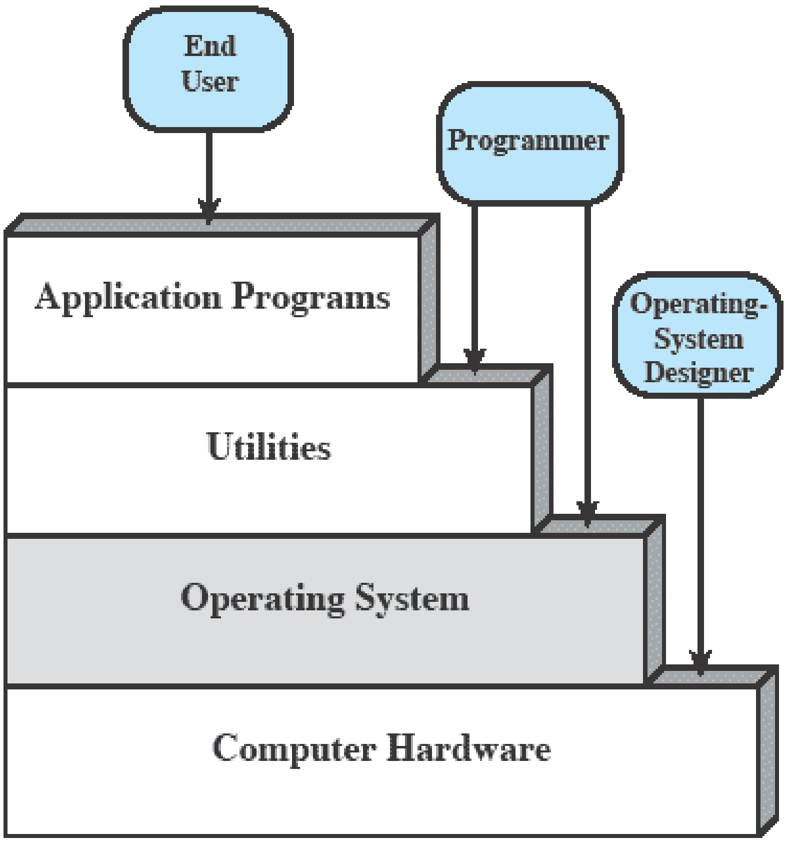
2

Operating System Objectives

- Convenience
  - Need no knowledge of the underlying hardware.
  - Provides an abstraction of standard services
- Efficiency
  - Move burden of optimization from developers to tools
- Ability to evolve ($\rightarrow$ interfaces & components)
  - Can replace internals as long as the interface stays the same

3

Layers and Views



Note: doing all this for one application is an incredibly complex task.

Figure 2.1 Layers and Views of a Computer System

4

Examples of OS Services

- Program development
  - Editors and debuggers & dtrace
- Program execution
  - In multi programming OS
  - Also in micro-controllers (startup routine)
- Access I/O devices
  - Uniform interfaces
  - Reduce required knowledge to read() and write()

5

Examples of OS Services

- Controlled access to files
  - Basic I/O, but also protection mechanisms in multi-user environments
- System access control
  - Generalized form of controlled resource access
  - Protection from unauthorized access

6

Examples of OS Services

- Error detection and response
  - Internal and external hardware errors
  - Software errors
  - Operating system cannot grant request of application (e.g., insufficient resources)

7

Examples of OS Services

- Accounting
  - Collect usage statistics
  - Monitor performance
  - Used to anticipate future enhancements
  - Used for billing purposes

  - Try out ‘psacct’ and ‘sa’

8

Operating System

- Responsible for managing the resources available in the computer
- Functions same way as ordinary computer software
  - It is a program that is executed
- Operating system relinquishes control of the processor

9

The operating system is a program that gets executed which means that the user program must relinquish control of the processor (this is done through interrupts)

OS as Resource Manager

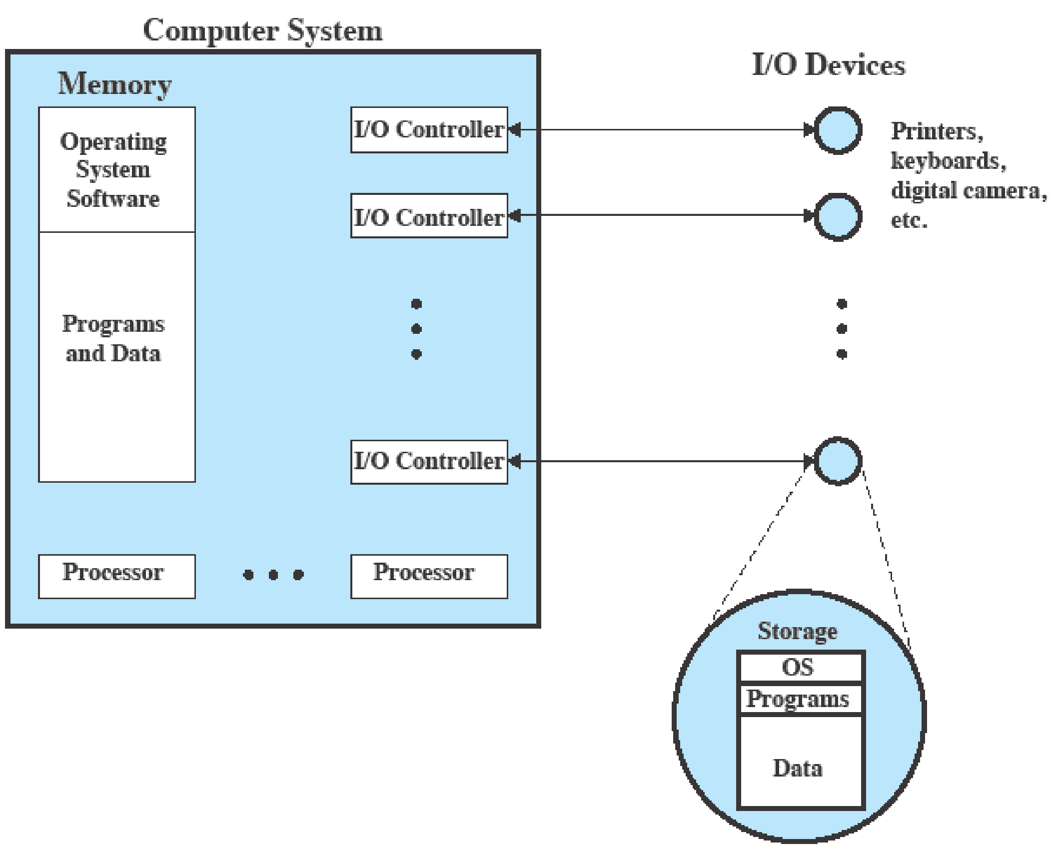


Figure 2.2 The Operating System as Resource Manager

Kernel

- Portion of operating system that is in main memory
- Contains most frequently used functions
- Also called the nucleus
- Embedded OSs are often only a nucleus.

Ease of Evolution in OSs

- Reasons for requiring the evolution:
  - Hardware upgrades plus new types of hardware
  - New services
  - Fixes

12

Evolution of OSs

- Serial processing (40-50s)
  - No operating system
  - Machines run from a console with display lights, toggle switches, input device, and printer

13

Evolution of OSs

- Serial processing shortcomings
  - Schedule time
    - Reserve time by means of hard copy signup sheets
    - Non-optimal work environment (forced interruptions)
  - Long lead times for programs
    - Setup included loading the compiler, source program, saving compiled program, and loading and linking

Developement on these machines was very hard because you needed physical access to one of a few machines resulting in large amounts of downtime between tests. There was a lot of setup required to run the program (load all of the things)

Evolution of OSs

- Simple batch system (goal: improve utilization)
  - Monitor
    - Software that controls the sequence of events
    - Batch jobs together
    - Program returns control to monitor when finished

This lets us bunch operations together to make things run much faster. The problem with this was security, you could override the monitor to make it so that your code had a higher priority.

Job Control Language

- Special type of programming language
- Provides instruction to the monitor
  - What compiler to use
  - What data to use

16

Hardware Features

- Memory protection
  - Do not allow the memory area containing the monitor to be altered
- Timer (duration)
  - Prevents a job from monopolizing the system

This helped stop people from overriding the monitor making things much more secure. Similarly is makes things more efficient by preventing one job from monopolizing the system. The monitor was a periodic interrupt that checked on things and jumped between programs as needed.

Hardware Features

- Privileged instructions
  - Certain machine level instructions can only be executed by the monitor (e.g., IO instructions)

- Interrupts
  - Early computer models did not have this capability

18

Modes of Operation

- Reason: protect users from each other, protect the kernel from users
  [prevent cheating with scheduler]

- Two modes:
  – User mode
    - Certain instructions may not be executed
  – Kernel mode (monitor)
    - Privileged instructions
    - Access to protected memory areas

19

We really want to protect the kernal from the users, to keep them from being cheating assholes. This lead to rings of protection. Basically you have two modes, user mode (where certain instructions are strictly not allowed) and kernal mode (where we have access to protected areas). Some processors have two more modes, but almost all operating systems only use these two.

Batch Systems Not Good Enough

Automatic job sequencing improves throughput, but IO is still slow.

Read one record from file	15 μs
Execute 100 instructions	1 μs
Write one record to file	15 μs
TOTAL	31 μs

$$\text{Percent CPU Utilization} = \frac{1}{31} = 0.032 = 3.2\%$$

96.8% waiting!

Consider EEPROM: takes several ms (!) to write, no interrupts allowed!

We care alot about CPU Utilization which is the percentage of time spent on the CPU in any function. We want this to be very high so that as little time as possible is wasted.

Uniprogramming

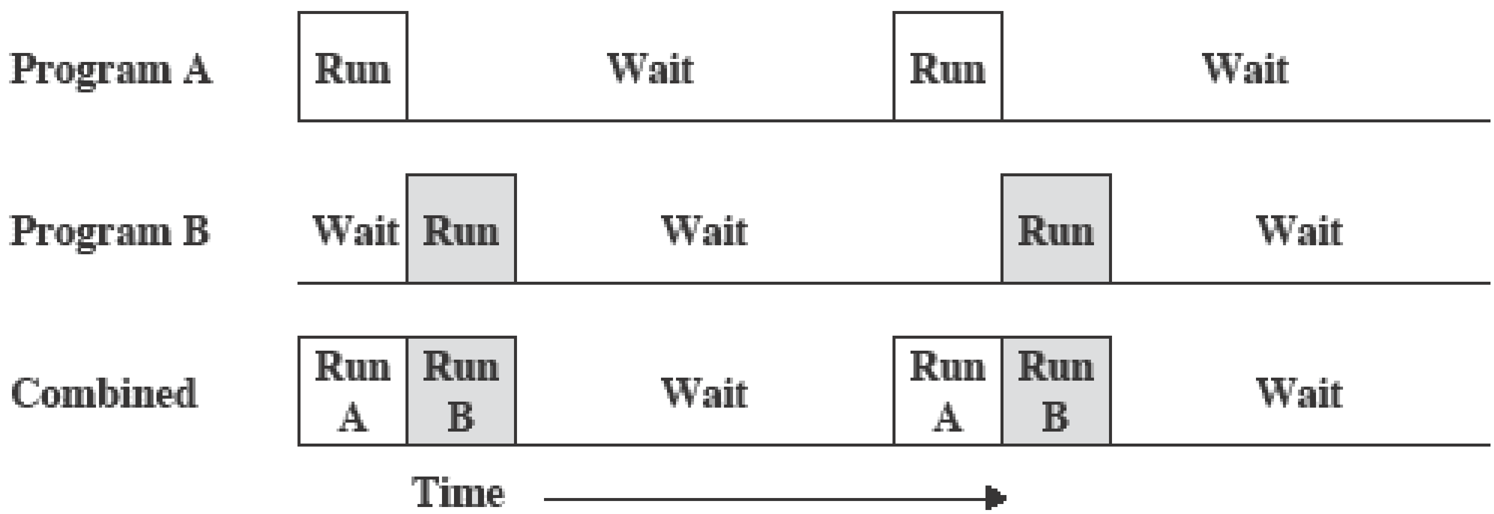
- Processor must wait for I/O instruction to complete before preceding

(a) Uniprogramming

21

Multiprogramming

- When one job needs to wait for I/O, the processor can switch to the other job

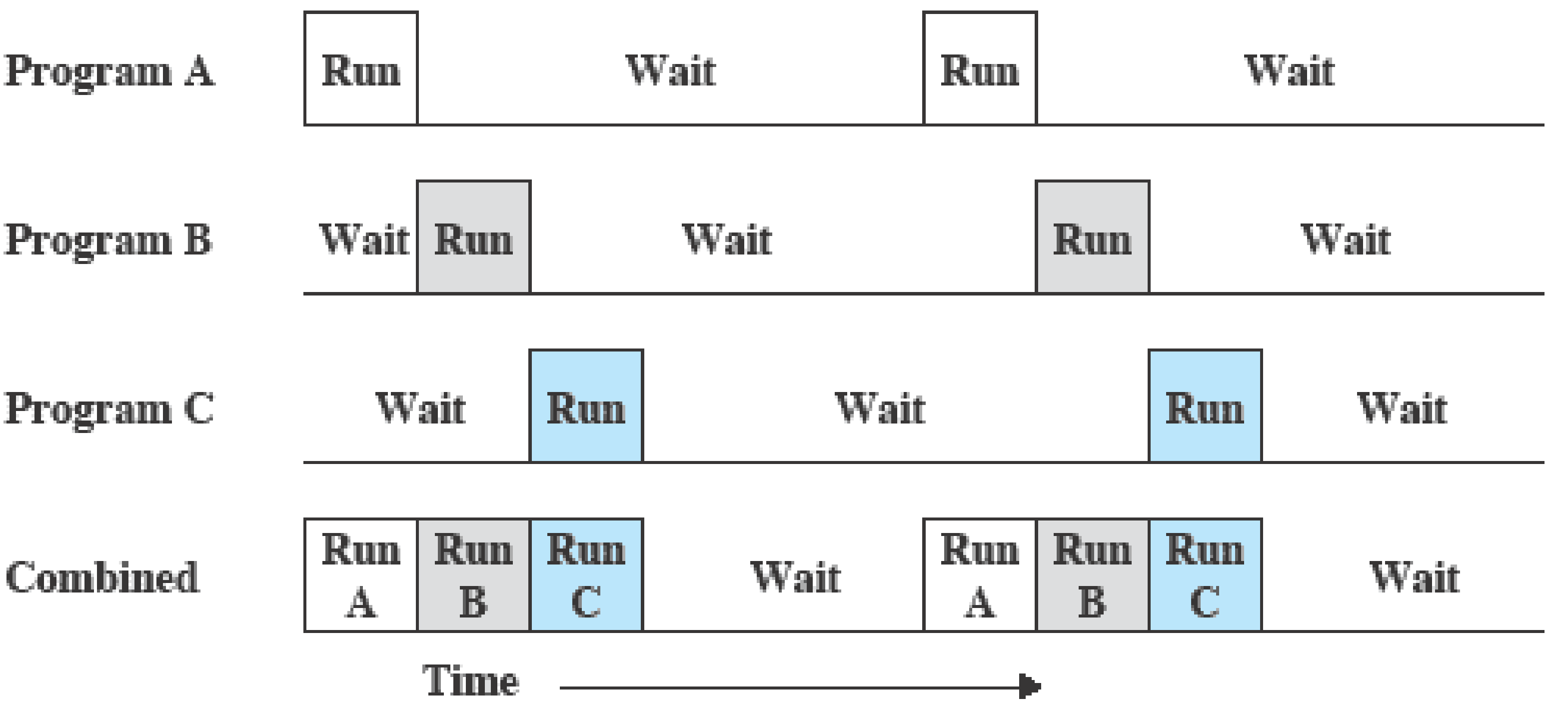


(b) Multiprogramming with two programs

22

The most basic way to efficinize the use of the CPU is to allow multiple programs to use the CPU at the same time during the wait periods. This is the core feature of all OS. We really like this, without it most things would be impossible.

Multiprogramming



(c) Multiprogramming with three programs

Example

Table 2.1 Sample Program Execution Attributes

	JOB1	JOB2	JOB3
Type of job	Heavy compute	Heavy I/O	Heavy I/O
Duration	5 min	15 min	10 min
Memory required	50 M	100 M	75 M
Need disk?	No	No	Yes
Need terminal?	No	Yes	No
Need printer?	No	No	Yes

Utilization Histograms

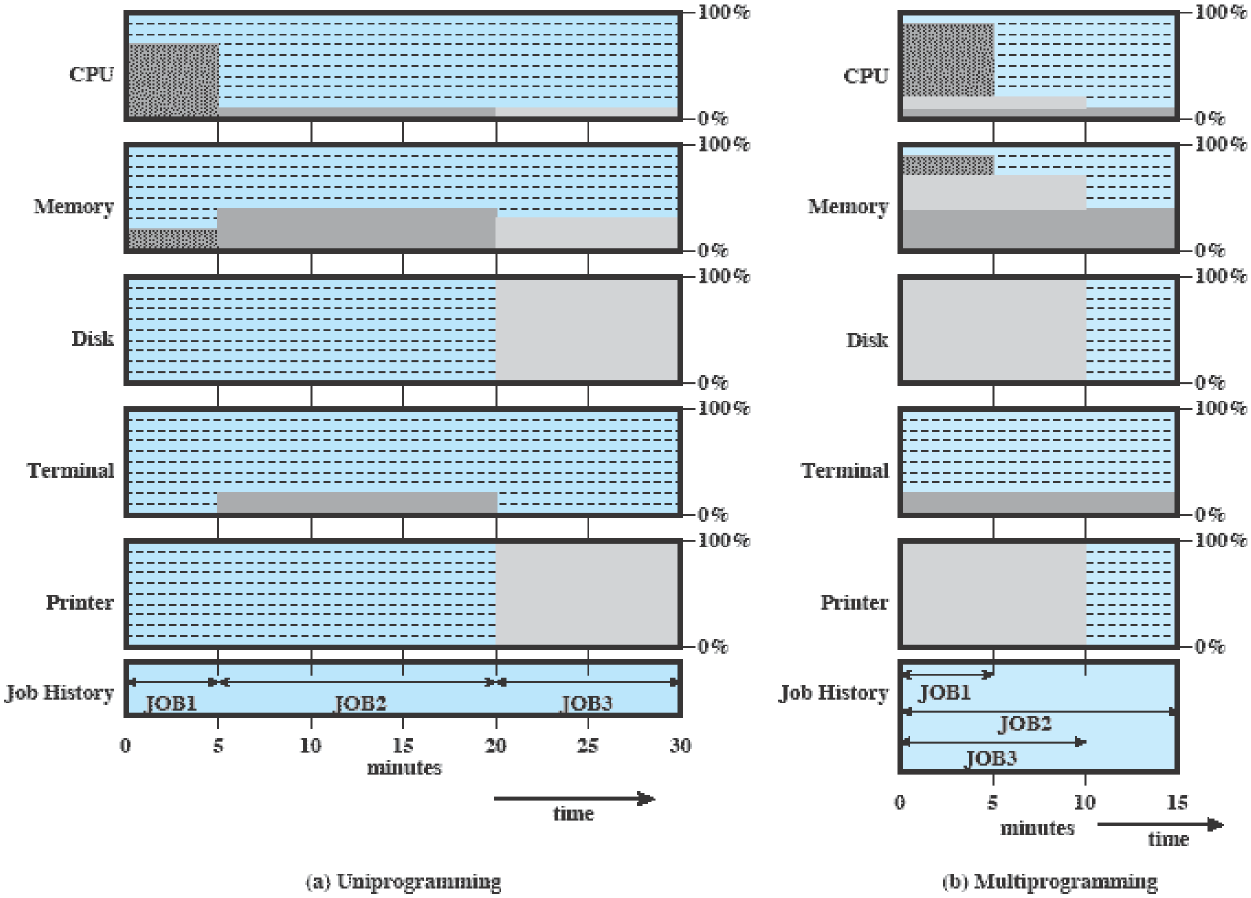


Figure 2.6 Utilization Histograms

25

We can see in this example that uniprocessing would suck balls to do this (30 min). If we use multiprocessing we can cut this down to the shortest period of time.

Effects of Multi-Programming

	Uniprogramming	Multiprogramming
Processor use	20%	40%
Memory use	33%	67%
Disk use	33%	67%
Printer use	33%	67%
Elapsed time	30 min	15 min
Throughput	6 jobs/hr	12 jobs/hr
Mean response time	18 min	10 min

By running multiple things at once we make everything way more efficient.

Time Sharing Systems

- Multiple users simultaneously access the system through terminals
  - Share time on the CPU
- Using multiprogramming to handle multiple interactive jobs
- Processor's time is shared among multiple users

27

Batch programming doesnt help when a program needs user interaction.

Batch Multiprogramming vs Time Sharing

Table 2.3 Batch Multiprogramming versus Time Sharing

	Batch Multiprogramming	Time Sharing
Principal objective	Maximize processor use	Minimize response time
Source of directives to operating system	Job control language commands provided with the job	Commands entered at the terminal

Major Achievements

- Processes
- Memory management
- Information protection and security
- Scheduling and resource management
- System structure

- (Virtualization)

Achievement I:

Process (possible definitions)

- 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

30

Process (possible definitions)

- A unit of activity characterized by
  – A single sequential thread of execution
  – A current state
  – An associated set of system resources

31

Process

- Consists of three components
  – An executable program
  – Associated data needed by the program
  – Execution context of the program
    - All information the operating system needs to manage the process

32

Process

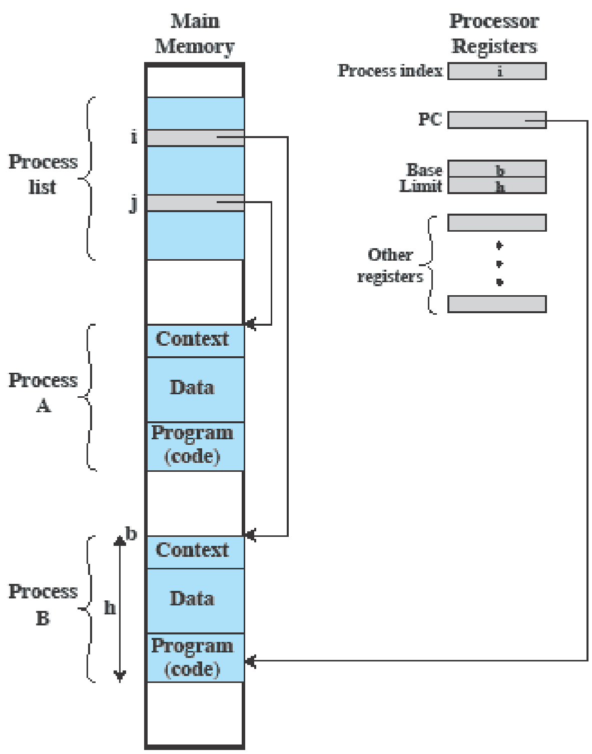


Figure 2.8 Typical Process Implementation

The most basic means of memory protection is to identify the set the chunk of memory allowed to that program restrict access outside of that.

Difficulties with Designing System Software

- Mix: multiple programs, multiple threads, interrupts, IO, shared resources …
- Common problems:
  - Improper synchronization (signaling availability of data → lost data or empty read)
  - Failed mutual exclusion (state corruption on shared memory)
  - Nondeterminate program operation (interference among programs in the OS due to memory allocation, IO access, …)
  - Deadlocks (resource access)

Achievement II: Responsibility of the OS wrt. Memory Management

- Process isolation
- Automatic allocation and management
  (you allocate 1kb but you don’t care how it’s actually done)
- Support of modular programming
  (swap program code)
- Protection and access control
  (shared memory)
- Long-term storage

35

Virtual Memory

- Allows programmers to address memory from a logical point of view without regard to how much memory is available
- Virtual address: page number plus offset in the page
- Real address: physical address

36

Paging

- Allows process to be comprised of a number of fixed-size blocks, called pages
- Virtual address is a page number and an offset within the page
- Each page may be located any where in main memory

Virtual Memory

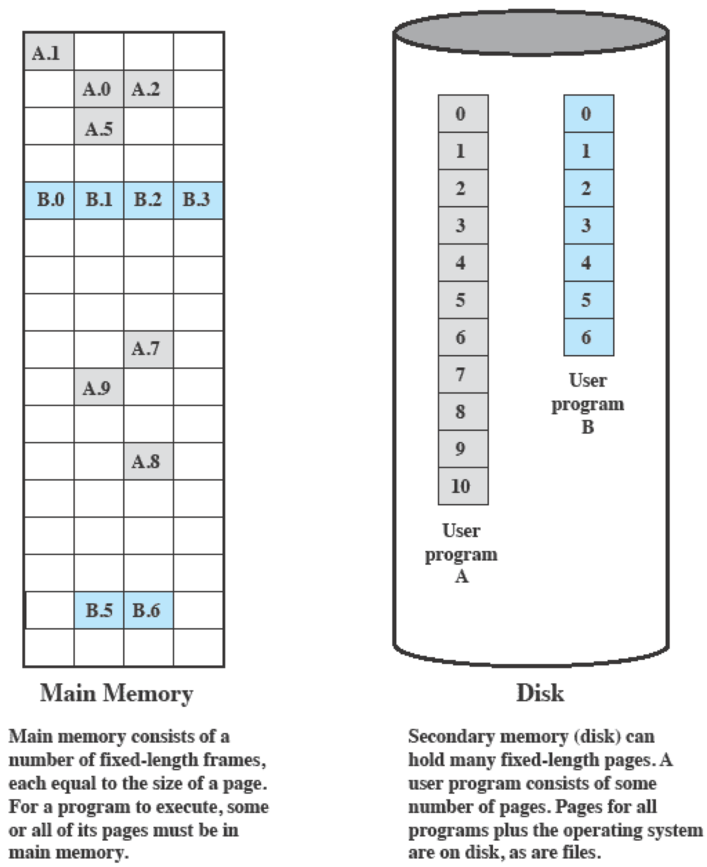


Figure 2.9 Virtual Memory Concepts

38

Virtual Memory Addressing

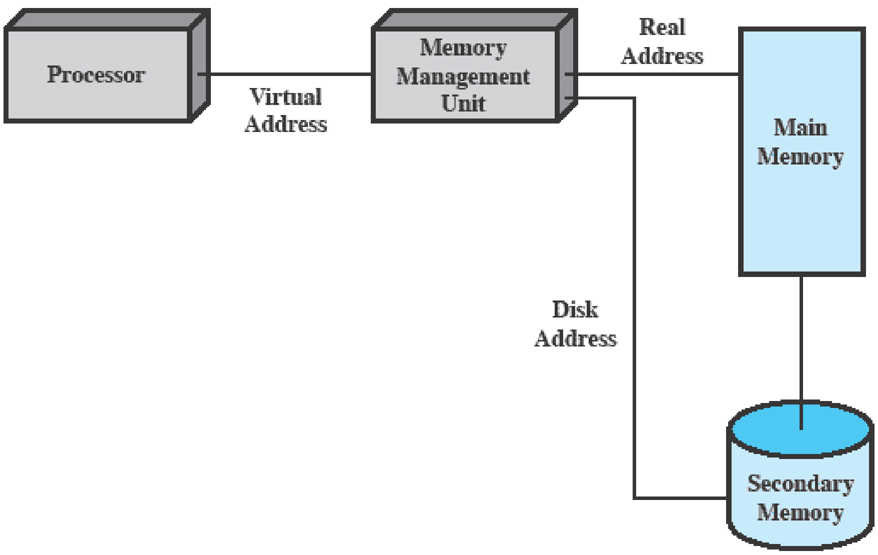


Figure 2.10 Virtual Memory Addressing

39

Achievement III: Information Protection and Security

- Availability
  - Concerned with protecting the system against interruption
    (e.g., RAID, dual-power supply solutions)

- Confidentiality
  - Assuring that users cannot read data for which access is unauthorized
    (‘chmod’)

40

Information Protection and Security

- Data integrity
  - Protection of data from unauthorized modification
    (assuming correct use and stays integer)

- Authenticity
  - Concerned with the proper verification of the identity of users and the validity of messages or data
    (Phishing attacks, cross-scripting)

Achievement IV: Scheduling and Resource Management

- Fairness
  - Give equal and fair access to resources (I really need the CPU, NOW!)

- Differential responsiveness
  - Discriminate among different classes of jobs
  - Often called QoS
  - Necessity for real-time systems

Scheduling and Resource Management

- Efficiency
  - Maximize throughput, minimize response time, and accommodate as many uses as possible

Key Elements of an Operating System

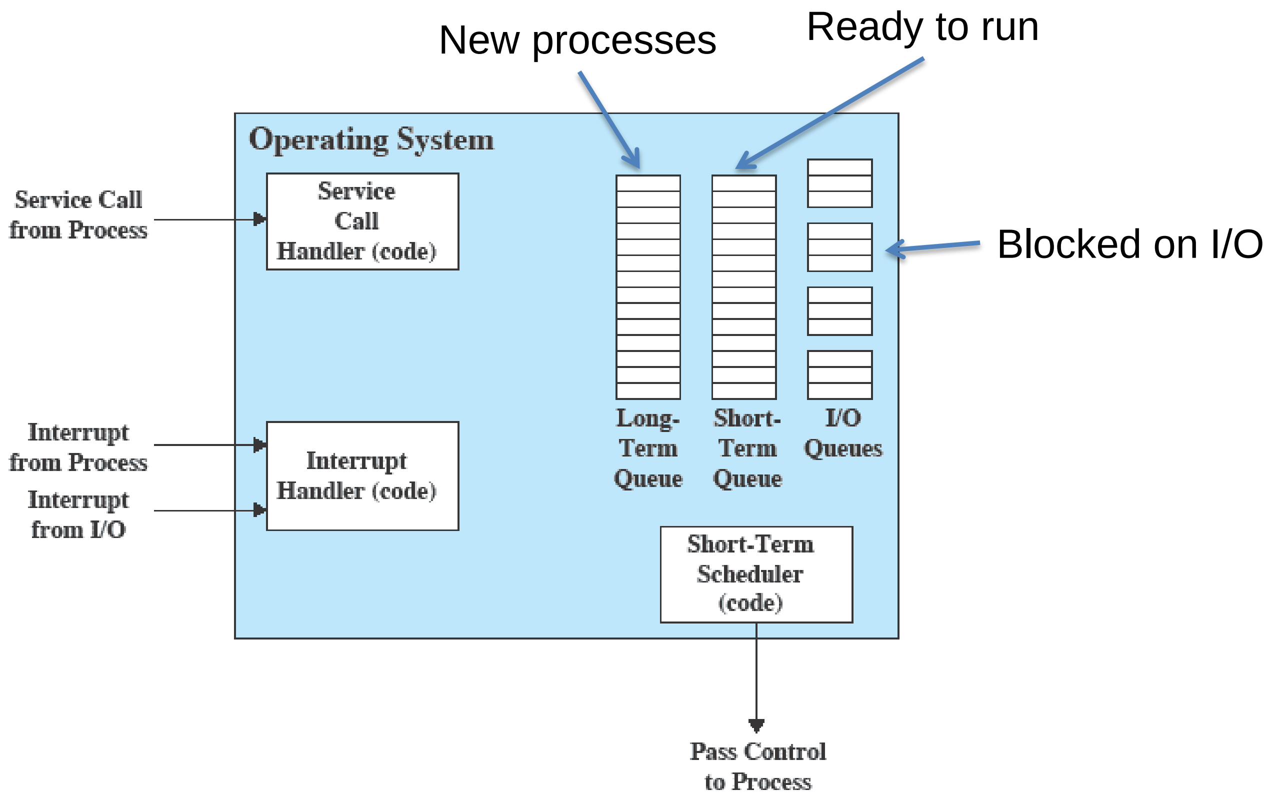


Figure 2.11 Key Elements of an Operating System for Multiprogramming

44

Problems with OSs

- Usually late on delivery
- Latent bugs (not quite banana products, but already close)
- Unexpectedly slow performance
- Security vulnerabilities

- => good structure might help
  (=> hierarchic structure)

45

Achievement V: System Structure

- View the system as a series of levels
- Each level performs a related subset of functions
- Each level relies on the next lower level to perform more primitive functions
- This decomposes a problem into a number of more manageable subproblems

46

Level	Name	Objects	Example Operations
13	Shell	User programming environment	Statements in shell language
12	User processes	User processes	Quit, kill, suspend, resume
11	Directories	Directories	Create, destroy, attach, detach, search, list
10	Devices	External devices, such as printers, displays, and keyboards	Open, close, read, write
9	File system	Files	Create, destroy, open, close, read, write
8	Communications	Pipes	Create, destroy, open, close, read, write
7	Virtual memory	Segments, pages	Read, write, fetch
6	Local secondary store	Blocks of data, device channels	Read, write, allocate, free
5	Primitive processes	Primitive processes, semaphores, ready list	Suspend, resume, wait, signal
4	Interrupts	Interrupt-handling programs	Invoke, mask, unmask, retry
3	Procedures	Procedures, call stack, display	Mark stack, call, return
2	Instruction set	Evaluation stack, microprogram interpreter, scalar and array data	Load, store, add, subtract, branch
1	Electronic circuits	Registers, gates, buses, etc.	Clear, transfer, activate, complement

Levels

- Level 1
  - Electronic circuits
  - Objects are registers, memory cells, and logic gates
  - Operations are clearing a register or reading a memory location
- Level 2
  - Processor’s instruction set
  - Operations such as add, subtract, load, and store

Levels

- Level 3
  – Adds the concept of a procedure or subroutine, plus call/return operations
- Level 4
  – Interrupts

49

Concepts with Multiprogramming

- Level 5
  - Process as a program in execution
  - Suspend and resume processes
- Level 6
  - Secondary storage devices
  - Transfer of blocks of data

Concepts with Multiprogramming

- Level 7
  - Creates logical address space for processes
  - Organizes virtual address space into blocks

Deal with External Objects

- Level 8
  - Communication of information and messages between processes
- Level 9
  - Supports long-term storage of named files
- Level 10
  - Provides access to external devices using standardized interfaces

52

Deal with External Objects

- Level 11
  - Responsible for maintaining the association between the external and internal identifiers
- Level 12
  - Provides full-featured facility for the support of processes
- Level 13
  - Provides an interface to the OS for the user

DEVELOPMENTS LEADING TO MODERN OPERATING SYSTEMS

54

Modern Operating Systems

- Microkernel architecture
  - Assigns only a few essential functions to the kernel
    - Address spaces
    - Interprocess communication (IPC)
    - Basic scheduling
  - All other elements are in user space
  - QNX

Modern Operating Systems

- Multithreading
  - Process is divided into threads that can run concurrently
    - Thread
      - Dispatchable unit of work
      - executes sequentially and is interruptable
    - Process is a collection of one or more threads

Modern Operating Systems

- Symmetric multiprocessing (SMP)
  - There are multiple processors
  - These processors share same main memory and I/O facilities
  - All processors can perform the same functions

57

Multiprogramming and Multiprocessing

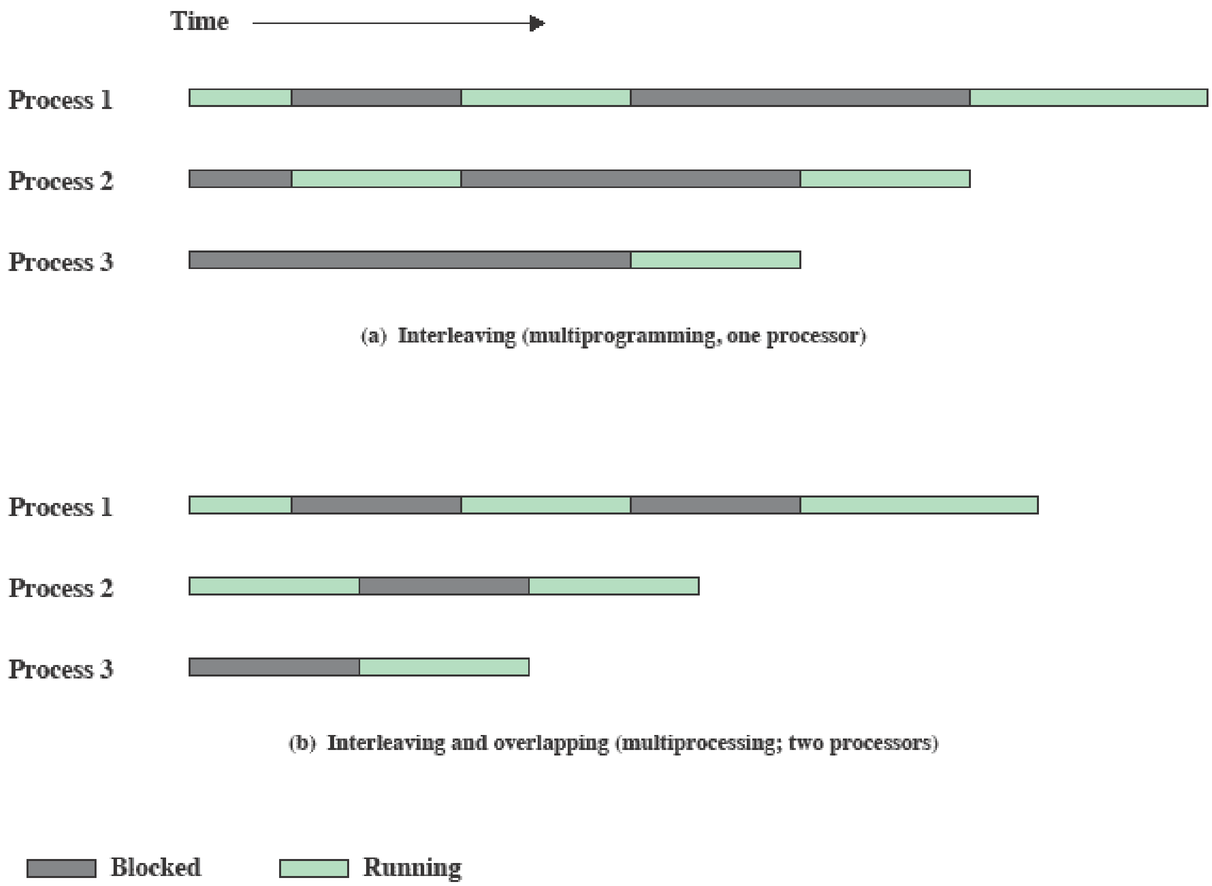


Figure 2.12 Multiprogramming and Multiprocessing

Modern Operating Systems

- Distributed operating systems
  - Provides the illusion of a single main memory space and single secondary memory space

- Network on chip
- Asymmetric multiprocessing
  (moderate success of cell processor)

59

Modern Operating Systems

- Object-oriented design
  - Used for adding modular extensions to a small kernel
  - Enables programmers to customize an operating system without disrupting system integrity

60