CSC 553 Operating Systems

Lecture 2- Operating System Overview

What is an Operating System?

- A program that controls the execution of application programs
- An interface between applications and hardware
- Main objectives of an OS:
  – Convenience
  – Efficiency
  – Ability to evolve

Hardware and Software Structure for a Computer

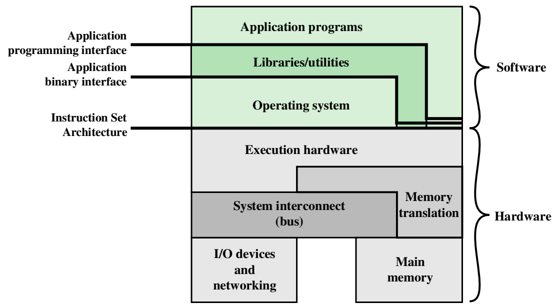


Figure 2.1 Computer Hardware and Software Structure

Operating System Services

- Program development
- Program execution
- Access I/O devices
- Controlled access to files
- System access
- Error detection and response
- Accounting

Key Interfaces

- Instruction set architecture (ISA)
- Application binary interface (ABI)
- Application programming interface (API)

The Operating System as Resource Manager

- The OS is responsible for controlling the use of a computer's resources, such as:
  - I/O
  - Main and secondary memory
  - Processor execution time

Operating System as Resource Manager

- Functions in the same way as ordinary computer software
- Program, or suite of programs, executed by the processor
- Frequently relinquishes control and must depend on the processor to allow it to regain control

Operating System as Resource Manager

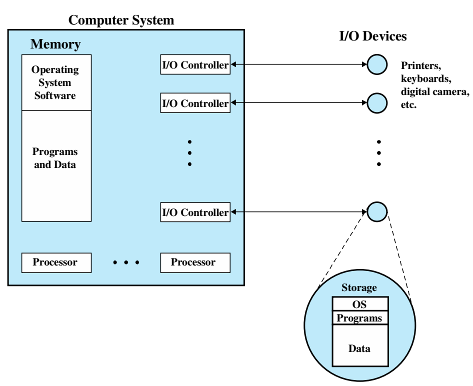
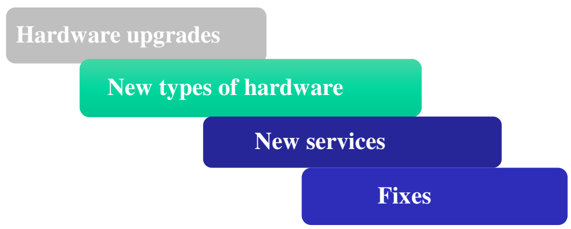


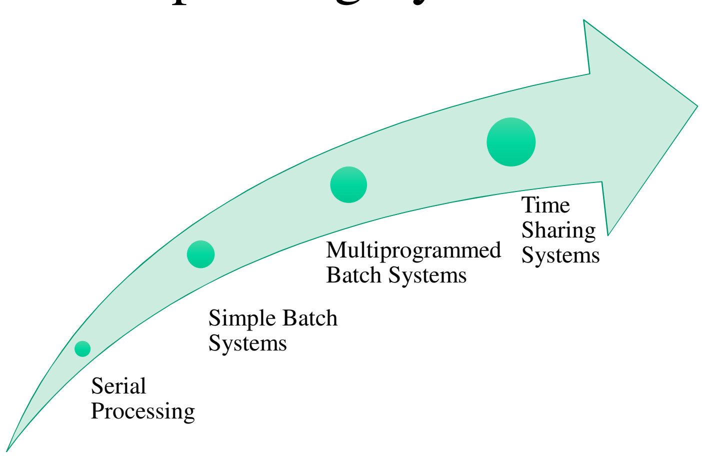
Figure 2.2 The Operating System as Resource Manager

Evolution of Operating Systems

- A major OS will evolve over time for a number of reasons:



Evolution of Operating Systems



Serial Processing – Earliest Computers

- No operating system
  - Programmers interacted directly with the computer hardware
- Computers ran from a console with display lights, toggle switches, some form of input device, and a printer
- Users have access to the computer in "series"

Serial Processing - Problems

- Scheduling:
  - Most installations used a hardcopy sign-up sheet to reserve computer time
    - Time allocations could run short or long, resulting in wasted computer time
- Setup time
  - A considerable amount of time was spent on setting up the program to run

Simple Batch Systems

- Early computers were very expensive
  - Important to maximize processor utilization
- Monitor
  - User no longer has direct access to processor
  - Job is submitted to computer operator who batches them together and places them on an input device
  - Program branches back to the monitor when finished

Monitor Point of View

- Monitor controls the sequence of events
- *Resident Monitor* is software always in memory
- Monitor reads in job and gives control
- Job returns control to monitor

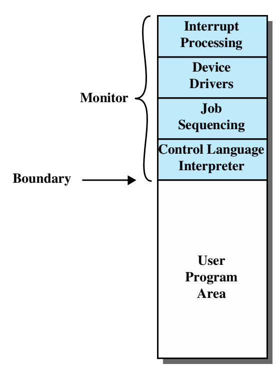


Figure 2.3 Memory Layout for a Resident Monitor

Processor Point of View

- Processor executes instruction from the memory containing the monitor
- Executes the instructions in the user program until it encounters an ending or error condition
- "*Control is passed to a job*" means processor is fetching and executing instructions in a user program
- "*Control is returned to the monitor*" means that the processor is fetching and executing instructions from the monitor program

Job Control Language (JCL)

Special type of programming language used to provide instructions to the monitor

↓

What compiler to use

↓

What data to use

Desirable Hardware Features

Memory protection

- While the user program is executing, it must not alter the memory area containing the monitor

Timer

- Prevents a job from monopolizing the system

Privileged instructions

- Can only be executed by the monitor

Interrupts

- Gives OS more flexibility in controlling user programs

Modes of Operation

User Mode

- User program executes in user mode
- Certain areas of memory are protected from user access
- Certain instructions may not be executed

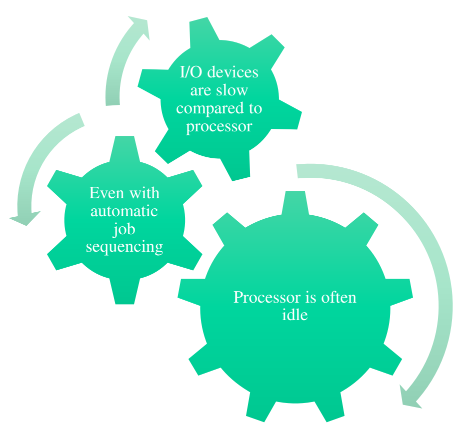
Kernel Mode

- Monitor executes in kernel mode
- Privileged instructions may be executed
- Protected areas of memory may be accessed

Simple Batch System Overhead

- Processor time alternates between execution of user programs and execution of the monitor
- Sacrifices:
  - Some main memory is now given over to the monitor
  - Some processor time is consumed by the monitor
- Despite overhead, the simple batch system improves utilization of the computer

Multiprogrammed Batch Systems



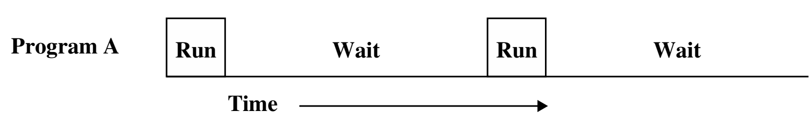
System Utilization

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

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

Figure 2.4 System Utilization Example

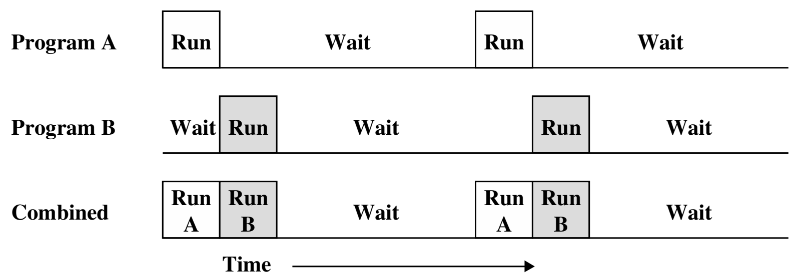
Uniprogramming



(a) Uniprogramming

The processor spends a certain amount of time executing, until it reaches an I/O instruction; it must then wait until that I/O instruction concludes before proceeding

Multiprogramming

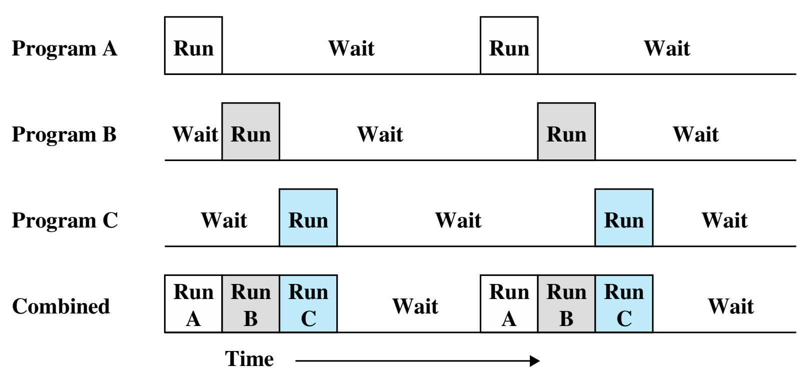


(b) Multiprogramming with two programs

There must be enough memory to hold the OS (resident monitor) and one user program

When one job needs to wait for I/O, the processor can switch to the other job, which is likely not waiting for I/O

Multiprogramming



(c) Multiprogramming with three programs

Also known as multitasking

Memory is expanded to hold three, four, or more programs and switch among all of them

Multiprogramming Example

	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

Effects of Multiprogramming on Resource Utilization

	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

Utilization Histograms for Uniprogramming and Multiprogramming

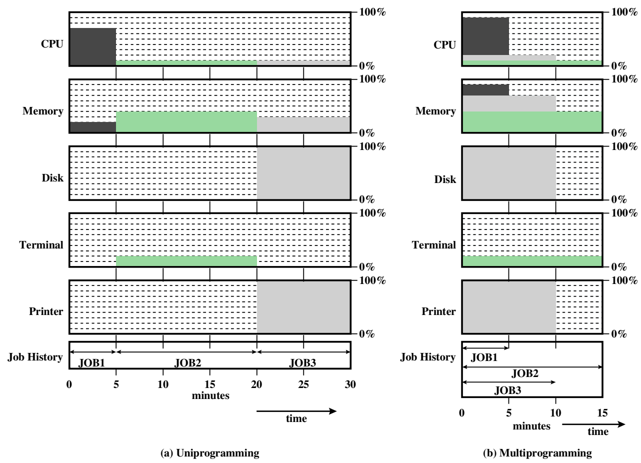


Figure 2.6 Utilization Histograms

Time-Sharing Systems

- Can be used to handle multiple interactive jobs
- Processor time is shared among multiple users
- Multiple users simultaneously access the system through terminals, with the OS interleaving the execution of each user program in a short burst or quantum of computation

Batch Multiprogramming vs Timesharing

	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

Compatible Time-Sharing System (CTSS)

- One of the first time-sharing operating systems
  - Developed at MIT by a group known as Project MAC
  - The system was first developed for the IBM 709 in 1961
  - Ran on a computer with 32,000 36-bit words of main memory, with the resident monitor consuming 5000 of that

Compatible Time-Sharing System (CTSS)

- CTSS utilized a technique known as *time slicing*

– System clock generated interrupts at a rate of approximately one every 0.2 seconds

– At each clock interrupt the OS regained control and could assign the processor to another user

– Thus, at regular time intervals the current user would be pre-empted and another user loaded in

Compatible Time-Sharing System (CTSS)

- CTSS utilized a technique known as *time slicing*

– To preserve the old user program status for later resumption, the old user programs and data were written out to disk before the new user programs and data were read in

– Old user program code and data were restored in main memory when that program was next given a turn

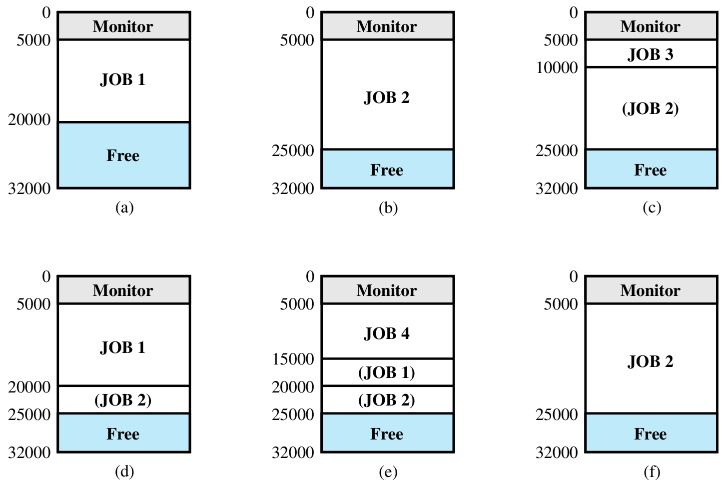


Figure 2.7 CTSS Operation

Major Achievements

- Operating Systems are among the most complex pieces of software ever developed
- Major advances in development include:
  - Processes
  - Memory management
  - Information protection and security
  - Scheduling and resource management
  - System structure

Process

- Fundamental to the structure of operating systems

A *process* can be defined as:

- A program in execution
- An instance of a running program
- The entity 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, and an associated set of system resources

Causes of Errors

- Improper synchronization
  - It is often the case that a routine must be suspended awaiting an event elsewhere in the system
  - Improper design of the signaling mechanism can result in loss or duplication

Causes of Errors

- Failed mutual exclusion
  - More than one user or program attempts to make use of a shared resource at the same time
  - There must be some sort of mutual exclusion mechanism that permits only one routine at a time to perform an update against the file

Causes of Errors

- Nondeterminate program operation
  - When programs share memory, and their execution is interleaved by the processor, they may interfere with each other by overwriting common memory areas in unpredictable ways
  - The order in which programs are scheduled may affect the outcome of any particular program

Causes of Errors

- Deadlocks
  - It is possible for two or more programs to be hung up waiting for each other

Components of 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

Components of a Process

- The execution context is essential:
  - It is the internal data by which the OS is able to supervise and control the process
  - Includes the contents of the various process 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

Process Management

- The entire state of the process at any instant is contained in its context
- New features can be designed and incorporated into the OS by expanding the context to include any new information needed to support the feature

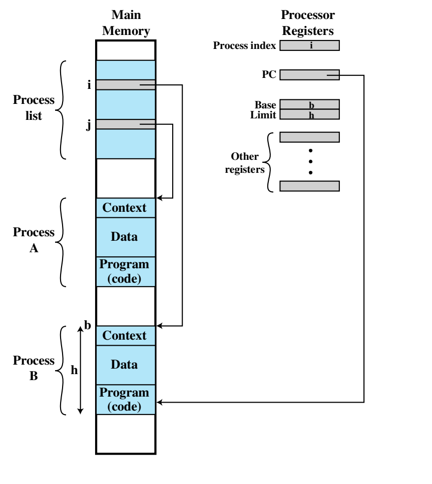


Figure 2.8 Typical Process Implementation

Memory Management

- The OS has five principal storage management responsibilities:
  - Process isolation
  - Automatic allocation and management
  - Support of modular programming
  - Protection and access control
  - Long-term storage

Virtual Memory

- A facility that allows programs to address memory from a logical point of view, without regard to the amount of main memory physically available
- Conceived to meet the requirement of having multiple user jobs reside in main memory concurrently

Paging

- Allows processes to be comprised of a number of fixed-size blocks, called pages
- Program references a word by means of a *virtual address,* consisting of a page number and an offset within the page
- Each page of a process may be located anywhere in main memory
- The paging system provides for a dynamic mapping between the virtual address used in the program and a *real address* (or physical address) in main memory

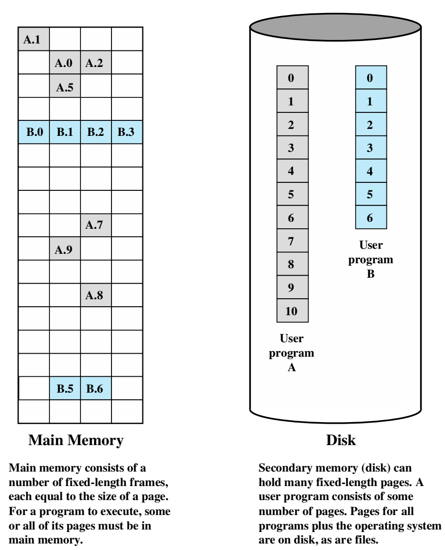


Figure 2.9 Virtual Memory Concepts

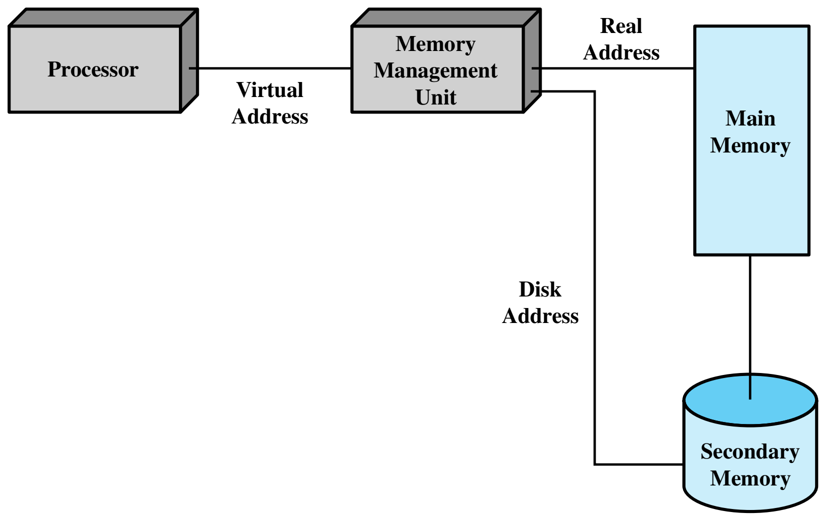
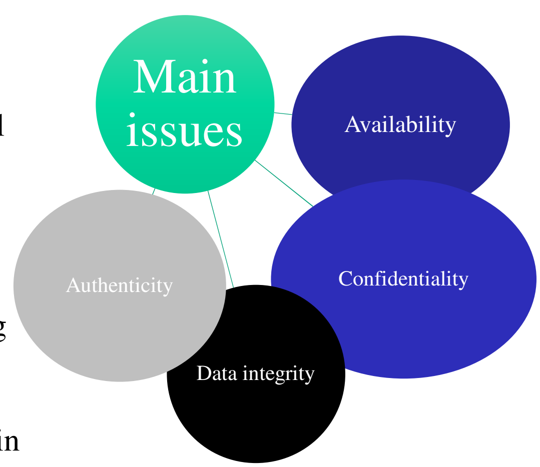


Figure 2.10 Virtual Memory Addressing

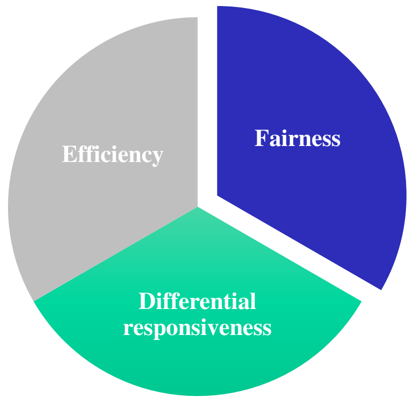
Information Protection and Security

- The nature of the threat that concerns an organization will vary greatly depending on the circumstances
- The problem involves controlling access to computer systems and the information stored in them



Scheduling and Resource Management

- Key responsibility of an OS is managing resources
- Resource allocation policies must consider:



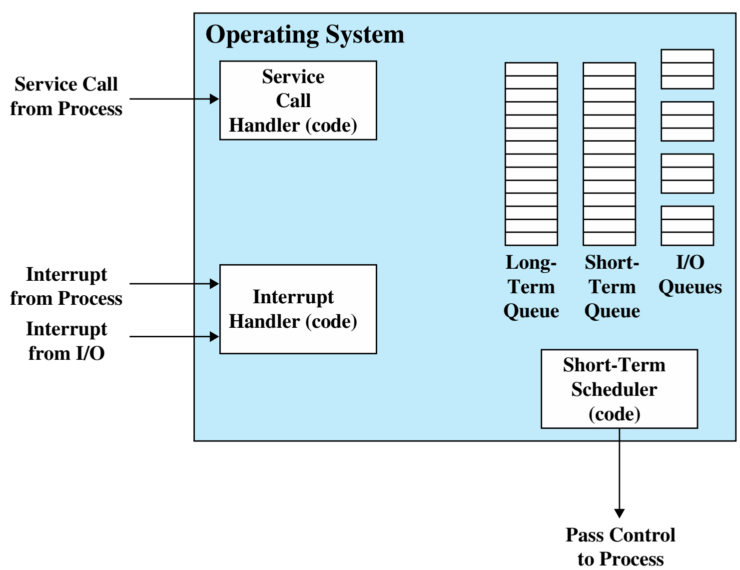


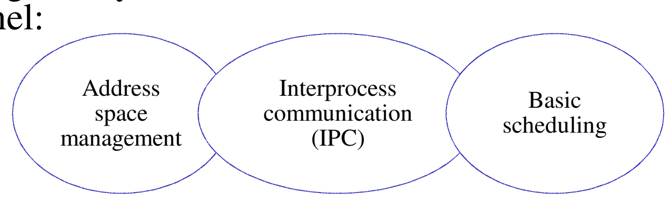
Figure 2.11 Key Elements of an Operating System for Multiprogramming

Different Architectural Approaches

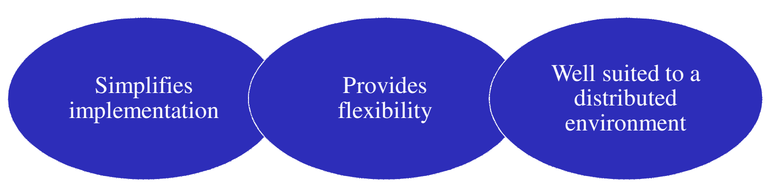
- Demands on operating systems require new ways of organizing the OS
- Different approaches and design elements have been tried:
  - Microkernel architecture
  - Multithreading
  - Symmetric multiprocessing
  - Distributed operating systems
  - Object-oriented design

Microkernel Architecture

- Assigns only a few essential functions to the kernel:

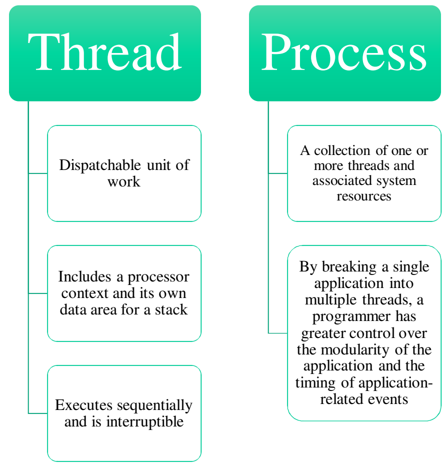


- The approach:



Multithreading

- Technique in which a process, executing an application, is divided into threads that can run concurrently



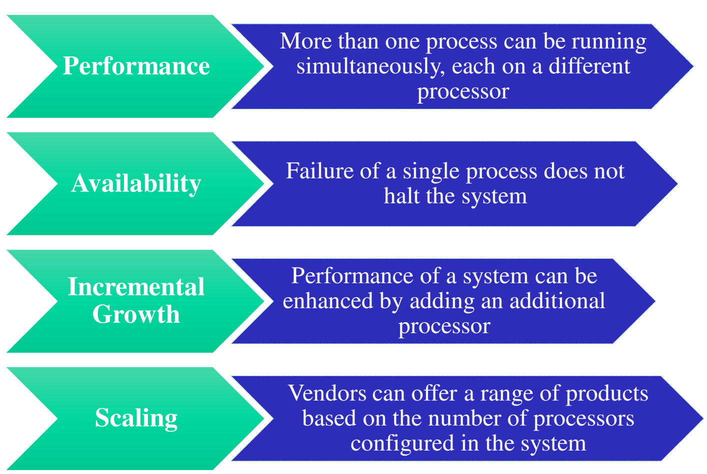
Symmetric Multiprocessing (SMP)

- Term that refers to a computer hardware architecture and also to the OS behavior that exploits that architecture
- The OS of an SMP schedules processes or threads across all of the processors
- The OS must provide tools and functions to exploit the parallelism in an SMP system

Symmetric Multiprocessing (SMP)

- Multithreading and SMP are often discussed together, but the two are independent facilities
- An attractive feature of an SMP is that the existence of multiple processors is transparent to the user

SMP Advantages



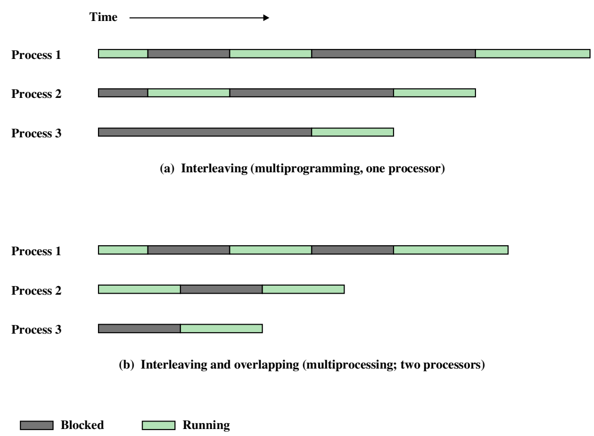


Figure 2.12 Multiprogramming and Multiprocessing

OS Design

- Distributed Operating System
  - Provides the illusion of a single main memory space and a single secondary memory space plus other unified access facilities, such as a distributed file system
  - State of the art for distributed operating systems lags that of uniprocessor and SMP operating systems

OS Design

- Object-Oriented Design
  - Lends discipline to the process of adding modular extensions to a small kernel
  - Enables programmers to customize an operating system without disrupting system integrity
  - Also eases the development of distributed tools and full-blown distributed operating systems

Fault Tolerance

- Refers to the ability of a system or component to continue normal operation despite the presence of hardware or software faults
- Typically involves some degree of redundancy
- Intended to increase the reliability of a system
  - Typically comes with a cost in financial terms or performance
- The extent adoption of fault tolerance measures must be determined by how critical the resource is

Fundamental Concepts

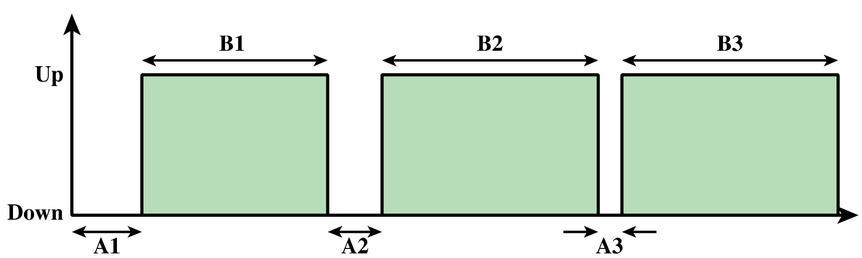
- The basic measures are:
  - Reliability
    - *R(t)*
    - Defined as the probability of its correct operation up to time *t* given that the system was operating correctly at time *t=0*

Fundamental Concepts

- The basic measures are:
  - Mean time to failure (MTTF)
    - Mean time to repair (MTTR) is the average time it takes to repair or replace a faulty element

Fundamental Concepts

- The basic measures are:
  - Availability
    - Defined as the fraction of time the system is available to service users' requests



$$MTTF = \frac{B1 + B2 + B3}{3} \qquad MTTR = \frac{A1 + A2 + A3}{3}$$

Figure 2.13 System Operational States

Availability Classes

Class	Availability	Annual Downtime
Continuous	1.0	0
Fault Tolerant	0.99999	5 minutes
Fault Resilient	0.9999	53 minutes
High Availability	0.999	8.3 hours
Normal Availability	0.99 - 0.995	44-87 hours

Faults

- Are defined by the IEEE Standards Dictionary as an erroneous hardware or software state resulting from:
  - Component failure
  - Operator error
  - Physical interference from the environment
  - Design error
  - Program error
  - Data structure error

Faults

- The standard also states that a fault manifests itself as:
  - A defect in a hardware device or component
  - An incorrect step, process, or data definition in a computer program

Fault Categories

- Permanent
  - A fault that, after it occurs, is always present
  - The fault persists until the faulty component is replaced or repaired

Fault Categories

- Temporary
  - A fault that is not present all the time for all operating conditions
  - Can be classified as
    - Transient – a fault that occurs only once
    - Intermittent – a fault that occurs at multiple, unpredictable times

Methods of Redundancy

Spatial (physical) redundancy

Involves the use of multiple components that either perform the same function simultaneously or are configured so that one component is available as a backup in case of the failure of another component

Temporal redundancy

Involves repeating a function or operation when an error is detected

Is effective with temporary faults but not useful for permanent faults

Information redundancy

Provides fault tolerance by replicating or coding data in such a way that bit errors can be both detected and corrected

Operating System Mechanisms

- A number of techniques can be incorporated into OS software to support fault tolerance:
  - Process isolation
  - Concurrency controls
  - Virtual machines
  - Checkpoints and rollbacks

Symmetric Multiprocessor OS Considerations

- A multiprocessor OS must provide all the functionality of a multiprogramming system plus additional features to accommodate multiple processors
- Key design issues:
  - Simultaneous concurrent processes or threads
  - Scheduling
  - Synchronization
  - Memory Management
  - Reliability and Fault Tolerance

Symmetric Multiprocessor OS Considerations

- Simultaneous concurrent processes or threads:
  - Kernel routines need to be reentrant to allow several processors to execute the same kernel code simultaneously

Symmetric Multiprocessor OS Considerations

- Scheduling:
  - Any processor may perform scheduling, which complicates the task of enforcing a scheduling policy

Symmetric Multiprocessor OS Considerations

- Synchronization:
  - With multiple active processes having potential access to shared address spaces or shared I/O resources, care must be taken to provide effective synchronization

Symmetric Multiprocessor OS Considerations

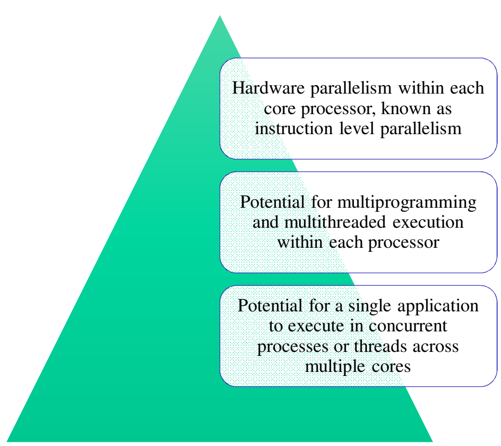
- Memory management:
  - The reuse of physical pages is the biggest problem of concern

Symmetric Multiprocessor OS Considerations

- Reliability and Fault Tolerance:
  - The OS should provide graceful degradation in the face of processor failure

Multicore OS Considerations

- The design challenge for a many-core multicore system is to efficiently harness the multicore processing power and intelligently manage the substantial on-chip resources efficiently
- Potential for parallelism exists at three levels:



Grand Central Dispatch (GCD)

- Is a multicore support capability
  - Once a developer has identified something that can be split off into a separate task, GCD makes it as easy and noninvasive as possible to actually do so
- In essence, GCD is a thread pool mechanism, in which the OS maps tasks onto threads representing an available degree of concurrency

Grand Central Dispatch (GCD)

- Provides the extension to programming languages to allow anonymous functions, called blocks, as a way of specifying tasks
- Makes it easy to break off the entire unit of work while maintaining the existing order and dependencies between subtasks

Virtual Machine Approach

- Allows one or more cores to be dedicated to a particular process and then leave the processor alone to devote its efforts to that process
- Multicore OS could then act as a hypervisor that makes a high-level decision to allocate cores to applications but does little in the way of resource allocation beyond that

Traditional UNIX Systems

- Developed at Bell Labs and became operational on a PDP-7 in 1970
- The first notable milestone was porting the UNIX system from the PDP-7 to the PDP-11
  - First showed that UNIX would be an OS for all computers

Traditional UNIX Systems

- Next milestone was rewriting UNIX in the programming language C
  - Demonstrated the advantages of using a high-level language for system code
- Was described in a technical journal for the first time in 1974
- First widely available version outside Bell Labs was Version 6 in 1976

Traditional UNIX Systems

- Version 7, released in 1978, is the ancestor of most modern UNIX systems
- Most important of the non-AT&T systems was UNIX BSD (Berkeley Software Distribution), running first on PDP and then on VAX computers

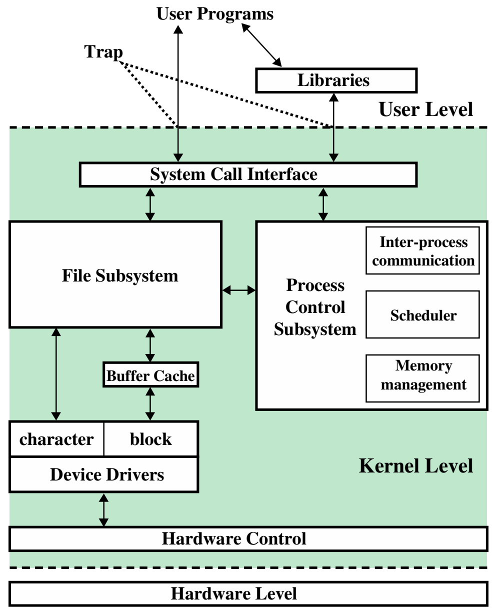
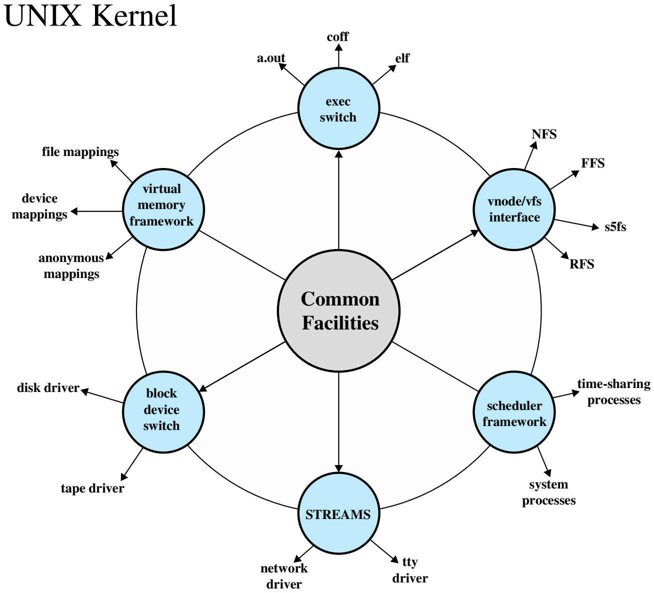


Figure 2.15 Traditional UNIX Kernel

Modern UNIX Kernel



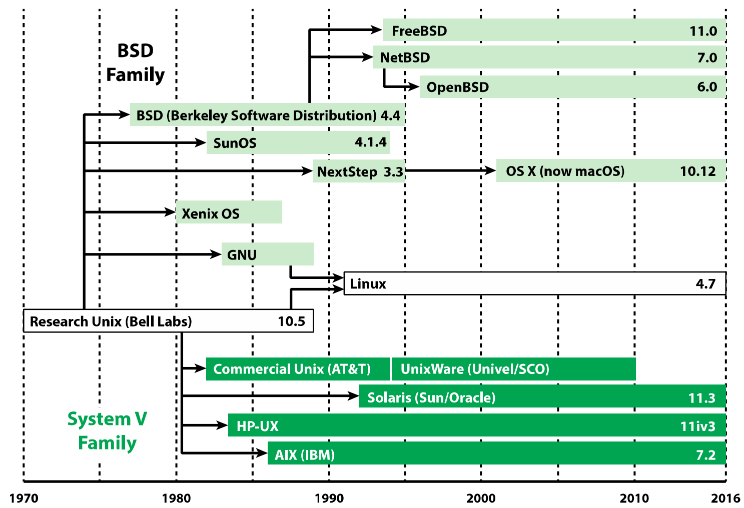


Figure 2.17 Unix Family Tree

System V Release 4 (SVR4)

- Developed jointly by AT&T and Sun Microsystems
- Combines features from SVR3, 4.3BSD, Microsoft Xenix System V, and SunOS

System V Release 4 (SVR4)

- New features in the release include:
  - Real-time processing support
  - Process scheduling classes
  - Dynamically allocated data structures
  - Virtual memory management
  - Virtual file system
  - Preemptive kernel

BSD

- Berkeley Software Distribution
- 4.xBSD is widely used in academic installations and has served as the basis of a number of commercial UNIX products
- 4.4BSD was the final version of BSD to be released by Berkeley

BSD

- There are several widely used, open-source versions of BSD:
  - FreeBSD
    - Popular for Internet-based servers and firewalls
    - Used in a number of embedded systems

BSD

- There are several widely used, open-source versions of BSD:
  - NetBSD
    - Available for many platforms
    - Often used in embedded systems
  - OpenBSD
    - An open-source OS that places special emphasis on security

Solaris 11

- Oracle's SVR4-based UNIX release
- Provides all of the features of SVR4 plus a number of more advanced features such as:
  - A fully preemptable, multithreaded kernel
  - Full support for SMP
  - An object-oriented interface to file systems