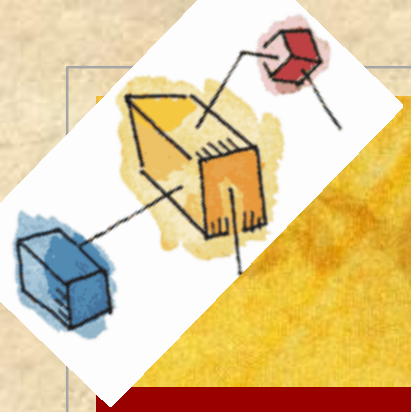
IT 540 Operating Systems
ECE519 Advanced Operating Systems

Prof. Dr. Hasan Hüseyin BALIK

(2nd Week)

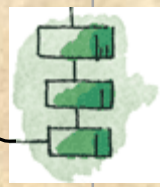
(Advanced) Operating Systems

2. Operating System Overview



2. Outline

- Operating System Objectives and Functions
- The Evolution of Operating Systems
- Major Achievements
- Developments Leading to Modern Operating Systems
- Virtual Machines
- OS Design Considerations for Multiprocessor and Multicore



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

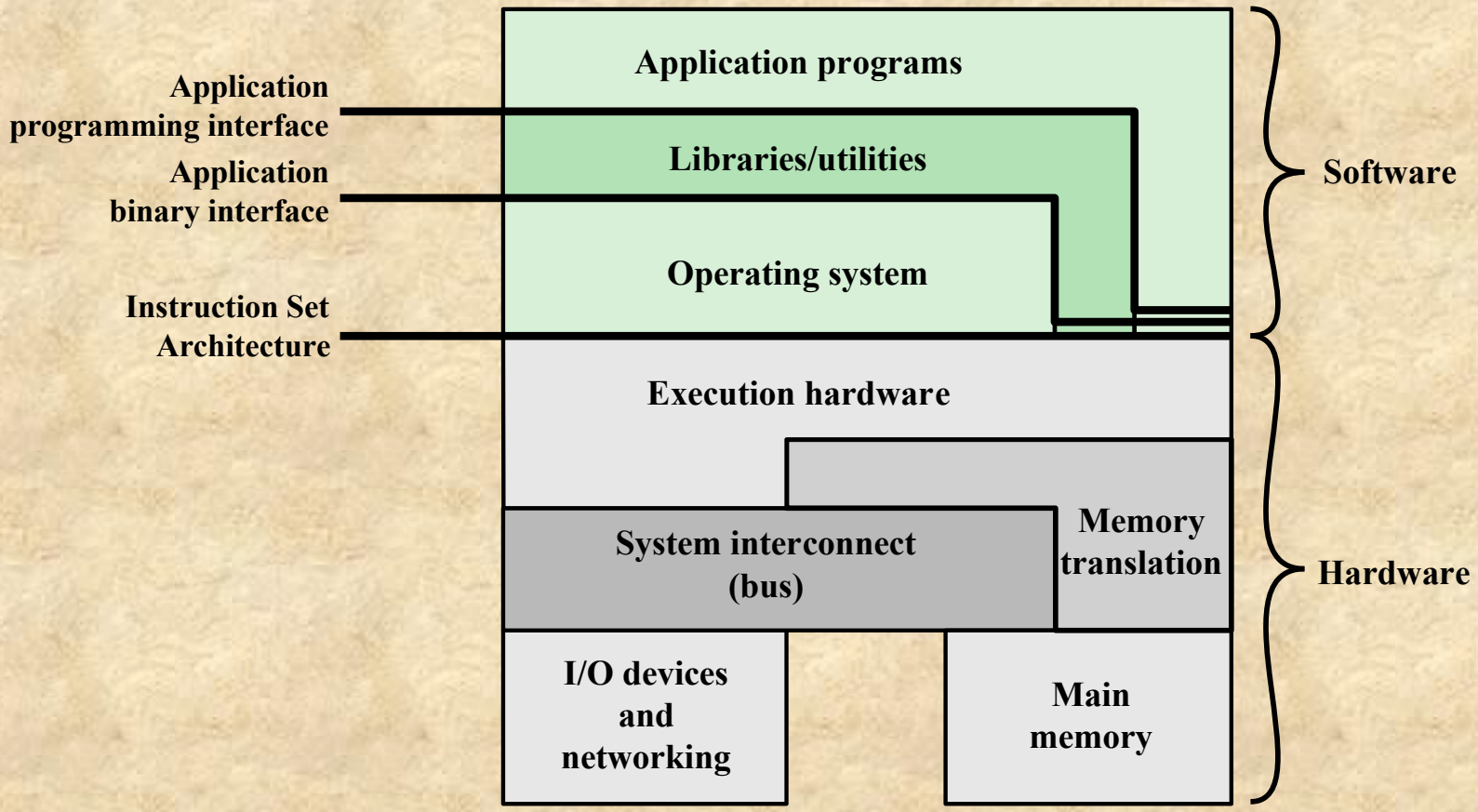


Figure 2.1 Computer Hardware and Software Structure

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 Role of an OS

- A computer is a set of resources for the movement, storage, and processing of data
- The OS is responsible for managing these resources

Operating System as Software

- 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

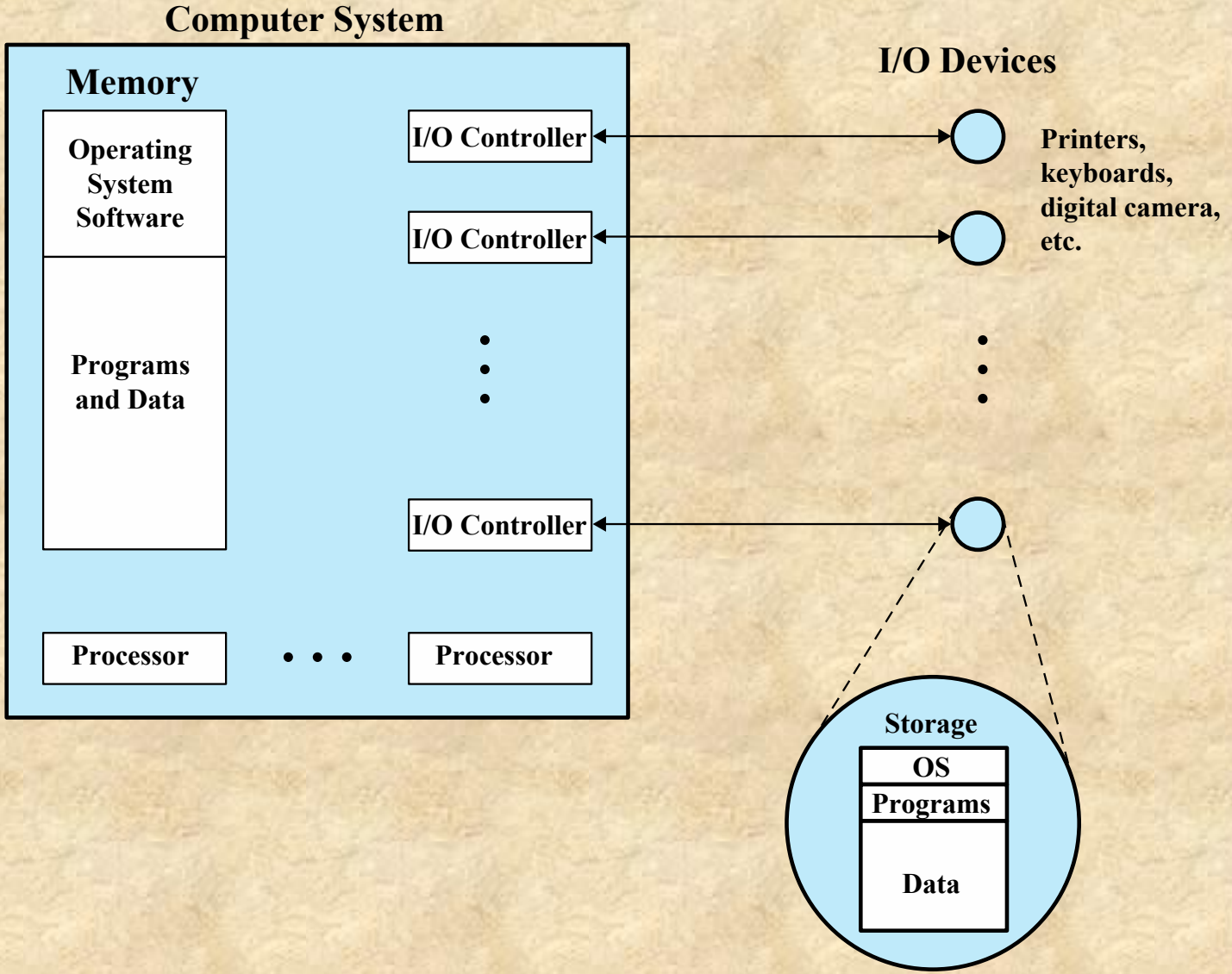


Figure 2.2 The Operating System as Resource Manager

Evolution of Operating Systems

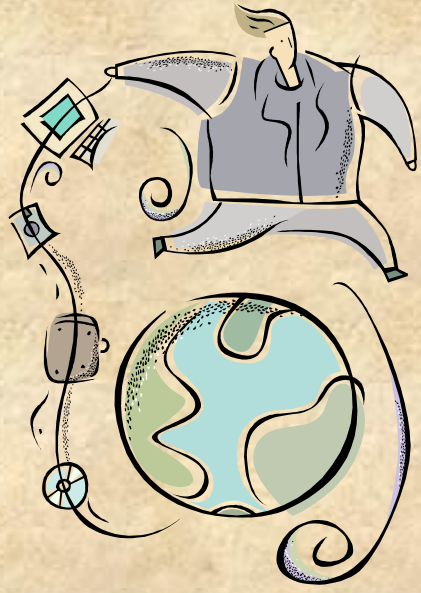
- A major OS will evolve over time for a number of reasons:

hardware upgrades

new types of hardware

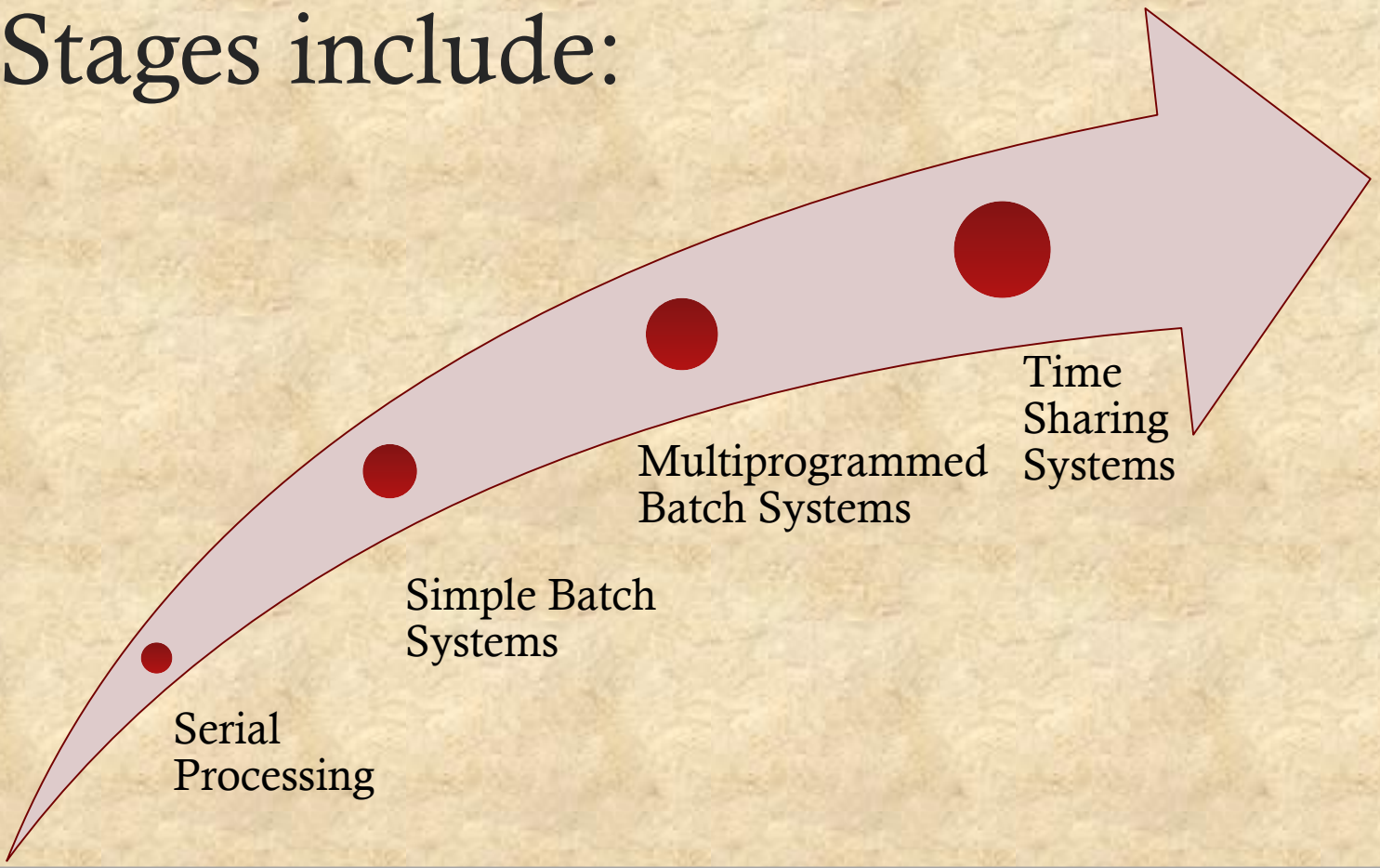
new services

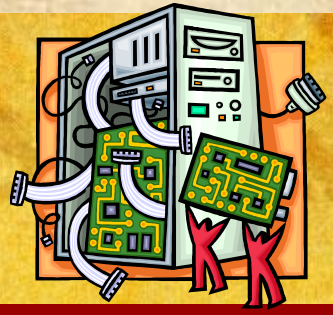
Fixes



Evolution of Operating Systems

- Stages include:





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"

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 just 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

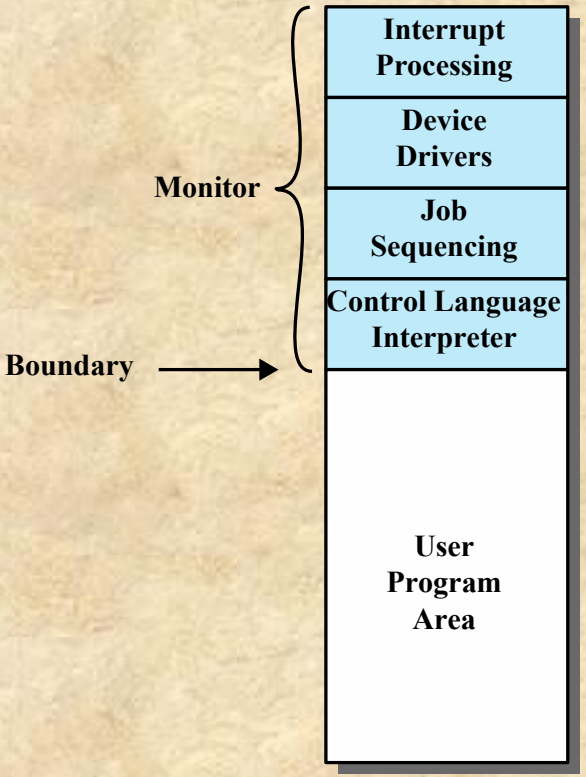


Figure 2.3 Memory Layout for a Resident Monitor

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 for monitor

- 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

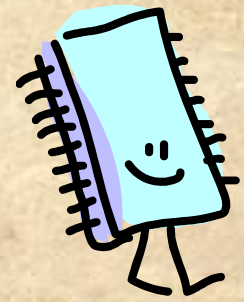
- user program executes in user mode
- certain areas of memory are protected from user access
- certain instructions may not be executed

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

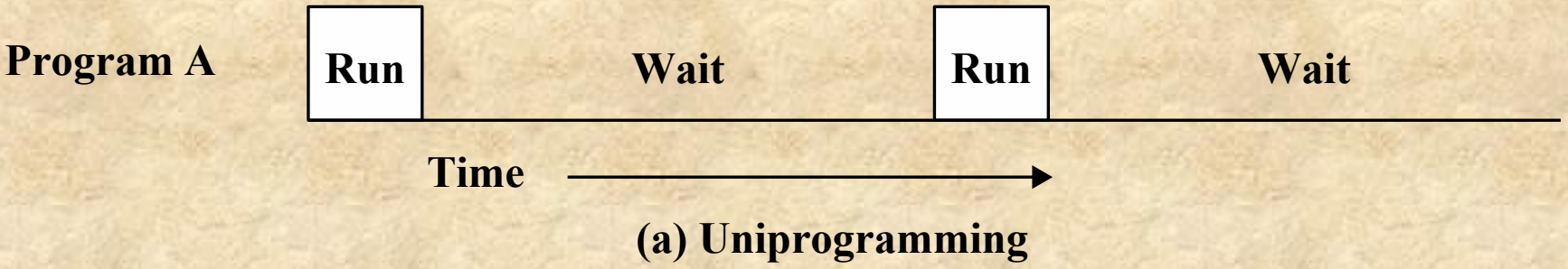
Read one record from file	15 μs
Execute 100 instructions	1 μs
Write one record to file	15 μs
TOTAL	31 μs

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

Figure 2.4 System Utilization Example

- Processor is often idle
  - even with automatic job sequencing
  - I/O devices are slow compared to processor

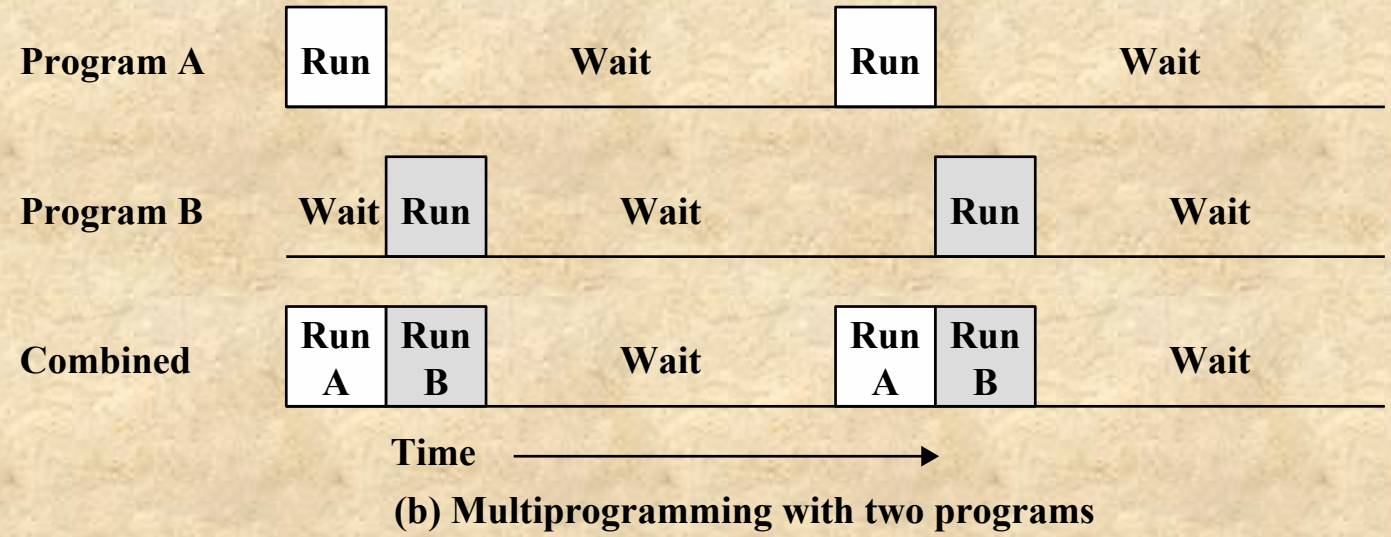
Uniprogramming



(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

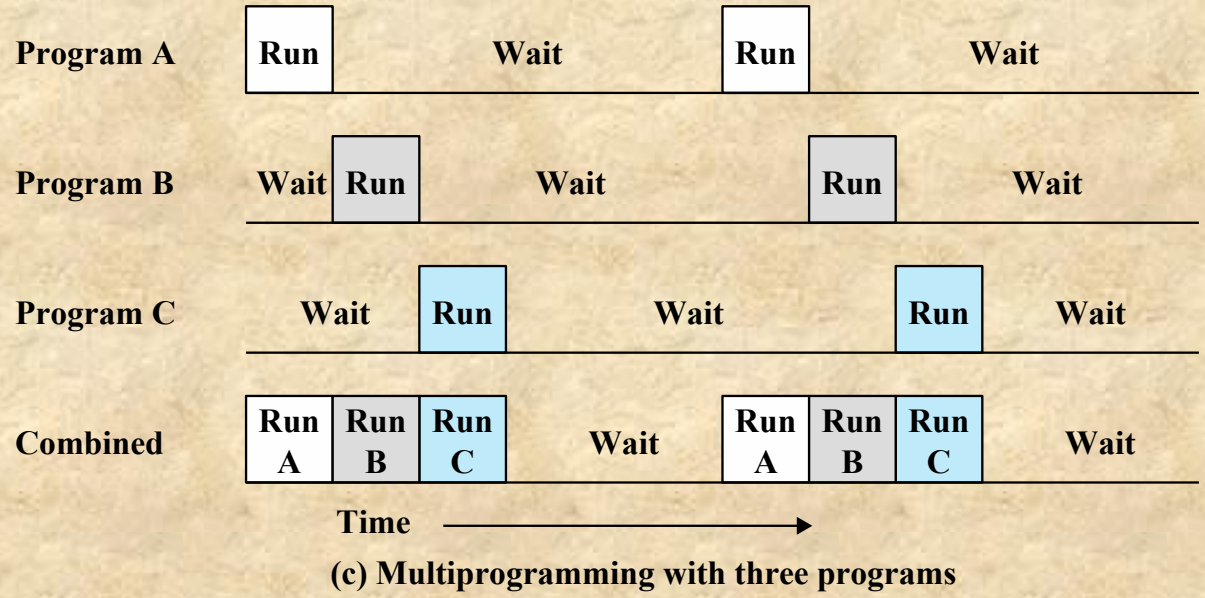
Multiprogramming



(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

- Multiprogramming
  - 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

Table 2.1 Sample Program Execution Attributes

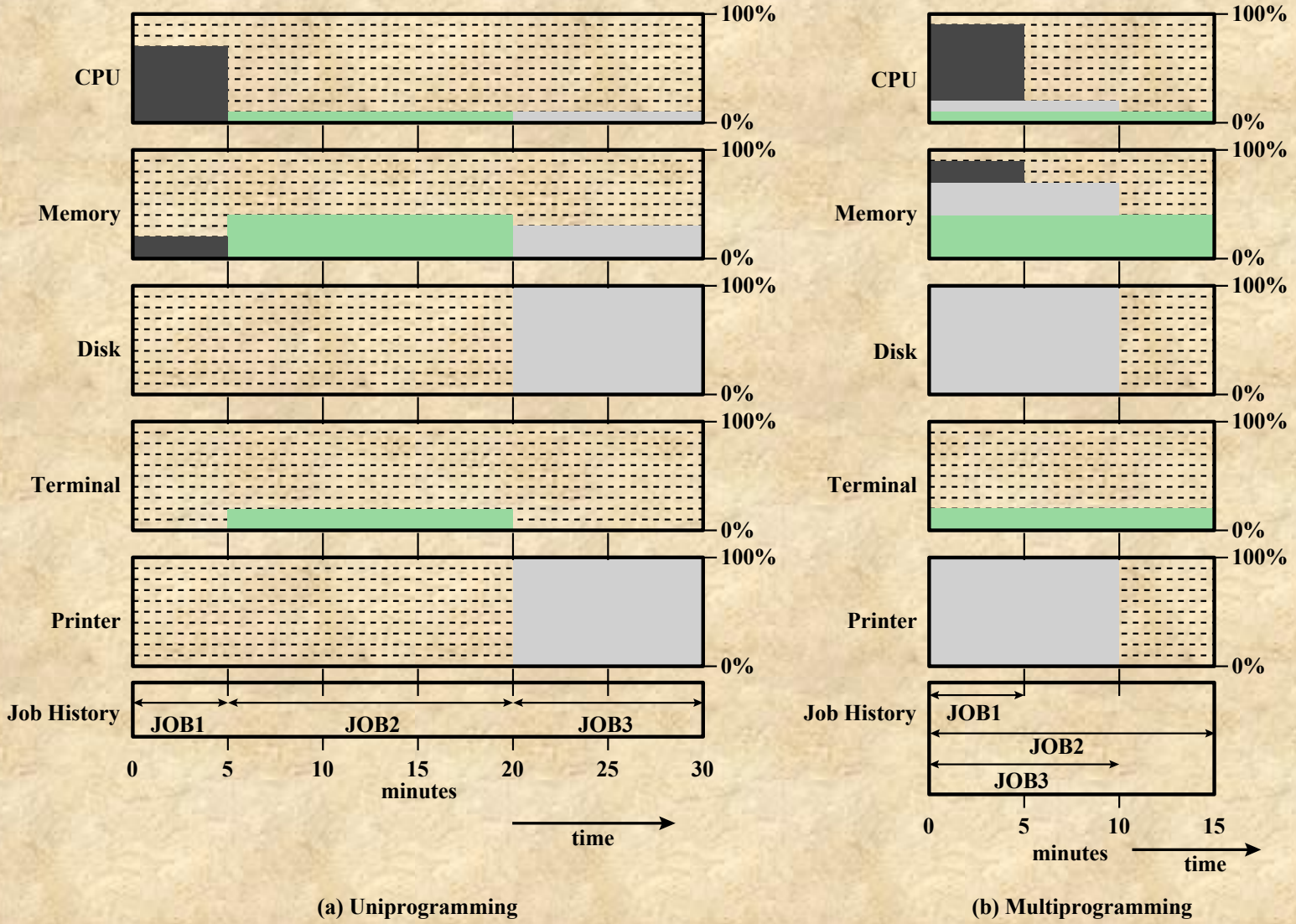


Figure 2.6 Utilization Histograms

Effects 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

Table 2.2 Effects of Multiprogramming on Resource Utilization

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. 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

Compatible Time-Sharing Systems

CTSS

- One of the first time-sharing operating systems
- Developed at MIT by a group known as Project MAC
- Ran on a computer with 32,000 36-bit words of main memory, with the resident monitor consuming 5000 of that
- To simplify both the monitor and memory management a program was always loaded to start at the location of the 5000th word

Time Slicing

- System clock generates interrupts at a rate of approximately one every 0.2 seconds
- At each interrupt OS regained control and could assign processor to another user
- At regular time intervals the current user would be preempted and another user loaded in
- Old user programs and data were written out to disk
- Old user program code and data were restored in main memory when that program was next given a turn

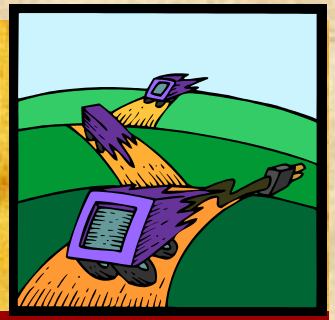
Major Achievements

- 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

Development of the Process

- Three major lines of computer system development created problems in timing and synchronization that contributed to the development:

multiprogramming batch operation

- processor is switched among the various programs residing in main memory

time sharing

- be responsive to the individual user but be able to support many users simultaneously

real-time transaction systems

- a number of users are entering queries or updates against a database

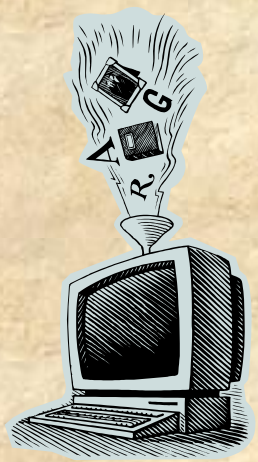
Causes of Errors

- **Improper synchronization**
  - a program must wait until the data are available in a buffer
  - improper design of the signaling mechanism can result in loss or duplication
- **Failed mutual exclusion**
  - more than one user or program attempts to make use of a shared resource at the same time
  - only one routine at a time allowed to perform an update against the file

- **Nondeterminate program operation**
  - program execution is interleaved by the processor when memory is shared
  - the order in which programs are scheduled may affect their outcome
- **Deadlocks**
  - it is possible for two or more programs to be hung up waiting for each other
  - may depend on the chance timing of resource allocation and release

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



- 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

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
  - consists of a page number and an offset within the page
  - each page may be located anywhere in main memory
- Provides for a dynamic mapping between the virtual address used in the program and a real (or physical) address in main memory

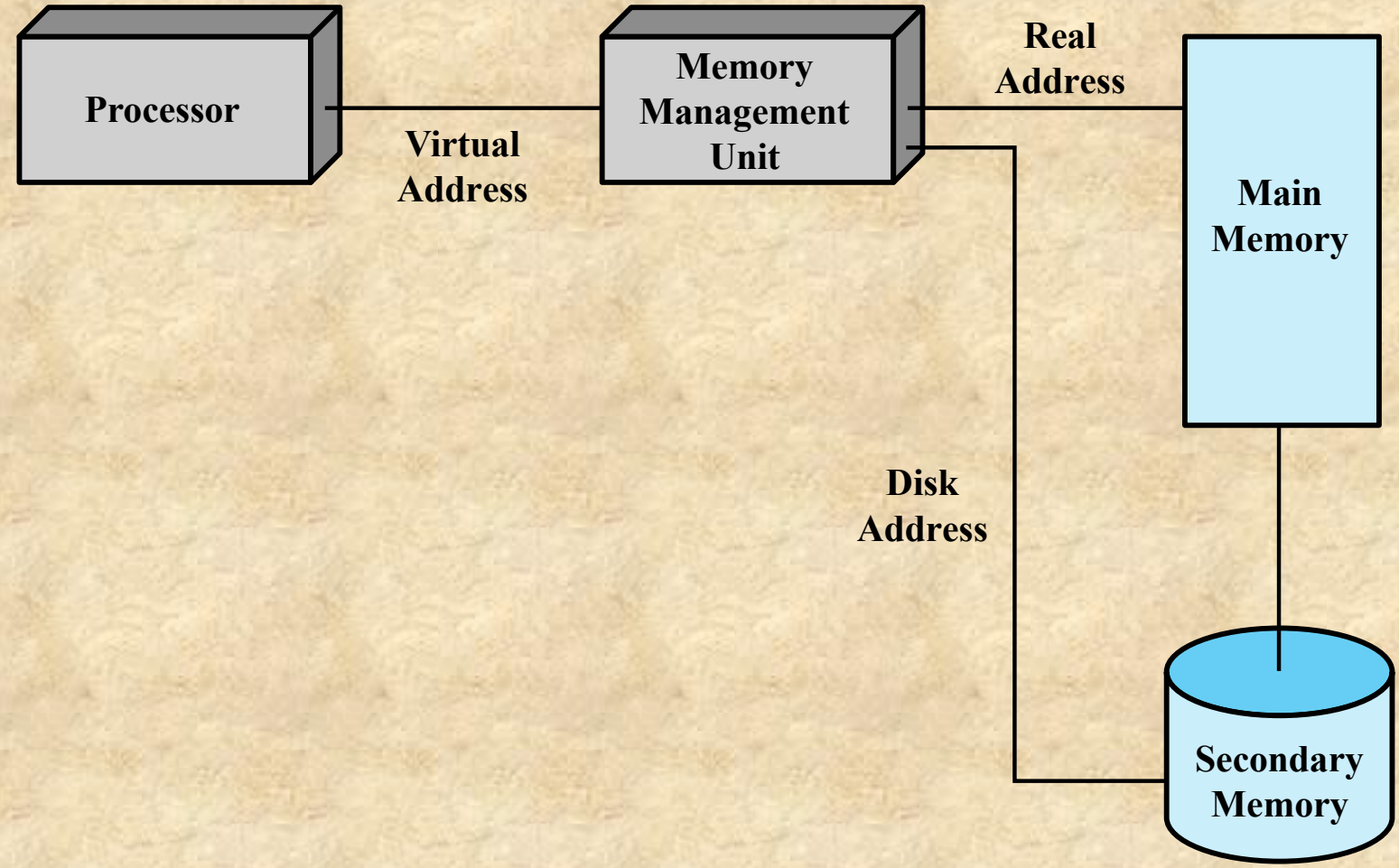


Figure 2.10 Virtual Memory Addressing

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:

Different Architectural Approaches

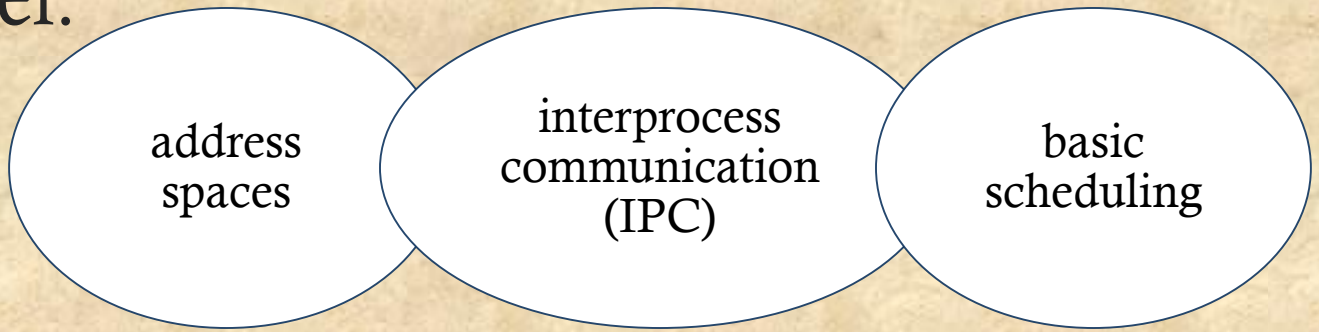
- Demands on operating systems require new ways of organizing the OS

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

Thread

- dispatchable unit of work
- includes a processor context and its own data area to enable subroutine branching
- executes sequentially and is interruptible

Process

- a collection of one or more threads and associated system resources
- programmer has greater control over the modularity of the application and the timing of application related events

Symmetric Multiprocessing (SMP)

- Term that refers to a computer hardware architecture and also to the OS behavior that exploits that architecture
- Several processes can run in parallel
- Multiple processors are transparent to the user
  - these processors share same main memory and I/O facilities
  - all processors can perform the same functions
- The OS takes care of scheduling of threads or processes on individual processors and of synchronization among processors

SMP Advantages

Performance	more than one process can be running simultaneously, each on a different processor
Availability	failure of a single process does not halt the system
Incremental Growth	performance of a system can be enhanced by adding an additional processor
Scaling	vendors can offer a range of products based on the number of processors configured in the system

OS Design

Distributed Operating System

- Provides the illusion of
  - a single main memory space
  - single secondary memory space
  - unified access facilities
- State of the art for distributed operating systems lags that of uniprocessor and SMP 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
- 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=o*
  - Mean time to failure (MTTF)
    - mean time to repair (MTTR) is the average time it takes to repair or replace a faulty element
  - Availability
    - defined as the fraction of time the system is available to service users' requests

Fault Categories

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

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

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
  - 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

kernel routines need to be reentrant to allow several processors to execute the same kernel code simultaneously

Scheduling

any processor may perform scheduling, which complicates the task of enforcing a scheduling policy

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

Memory management

the reuse of physical pages is the biggest problem of concern

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: