

## CHAPTER 1

1Q.What is operating system and how it works?

1A.Operating system is a resource allocator and it also allow to control program making efficient use of hardware and managing execution of user programs.

2Q.What is computer system architecture

2A. computer architecture is a set of rules and methods that describe the functionality, organization, and implementation of computer systems. It is divided into 4 components

1.Hardware :provides basic computing resources

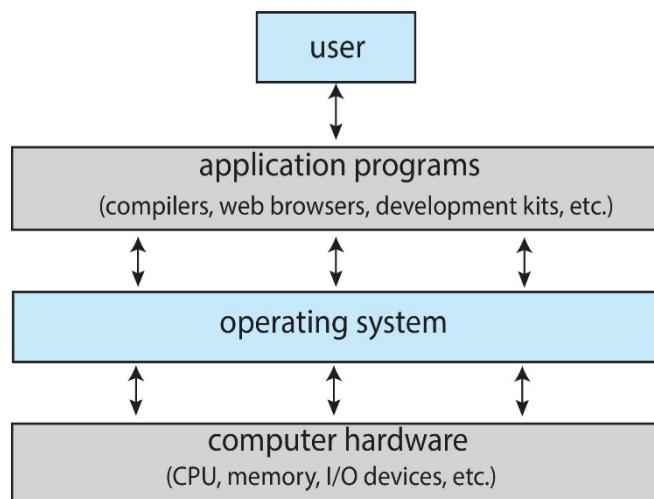
Ex : CPU, Memory, I/O devices

2.Operating systems :Controls and coordinates use of hardware among various applications and users

3.Application programs : Define the ways in which the system resources are used to solve the computing problem of the users

Ex : database systems,Web browser,video games

Users :people,machines and other computers



3Q .What is interrupts?

3A.Interrupt is an event that alters the sequence in which the processor executes instructions.An interrupt may be planned or unplanned

Ex of planned interrupt : software instructions

EX of unplanned interrupt : Program errors while executing

Class	Cause	Async/Sync	Return behavior
Interrupt	Signal from I/O device	Async	Always returns to next instruction
Trap	Intentional exception	Sync	Always returns to next instruction
Fault	Potentially recoverable error	Sync	Might return to current instruction
Aboit	Nonrecoverable error	Sync	Never returns

 Figure 8.4 Classes of exceptions. Asynchronous exceptions occur as a result of events in I/O devices that are external to the processor. Synchronous exceptions occur as a direct result of executing an instruction.
 Operating System Concepts - 10th Edition 1.17 Silberschatz, Galvin and Gagne ©2016
 

4Q : What is storage structure?

4A. Storage structure is divided into 4 types

1. Main memory : It is a large storage area where only CPU can access directly

- Ex : **Random access (pinpoint any point of address) (RAM)**
- Typically **volatile**
- Typically **random-access memory** in the form of **Dynamic Random-access Memory (DRAM)**

2. Secondary storage – extension of **main memory** that provides large **nonvolatile** storage capacity

- **Hard Disk Drives (HDD)** – rigid metal or glass platters covered with magnetic recording material
  - Disk surface is logically divided into **tracks**, which are subdivided into **sectors**
  - The **disk controller** determines the logical interaction between the **device** and the **computer**
- **Non-volatile memory (NVM) (SSD drive)** devices – faster than hard disks, nonvolatile
  - Various technologies
  - Becoming more popular as capacity and performance increases, price drops

5Q. Computer system architecture?

5A.

- 1. Single-Processor Systems (single-pc) : Most systems use a **single general-purpose processor**

Most systems have **special-purpose processors** as well



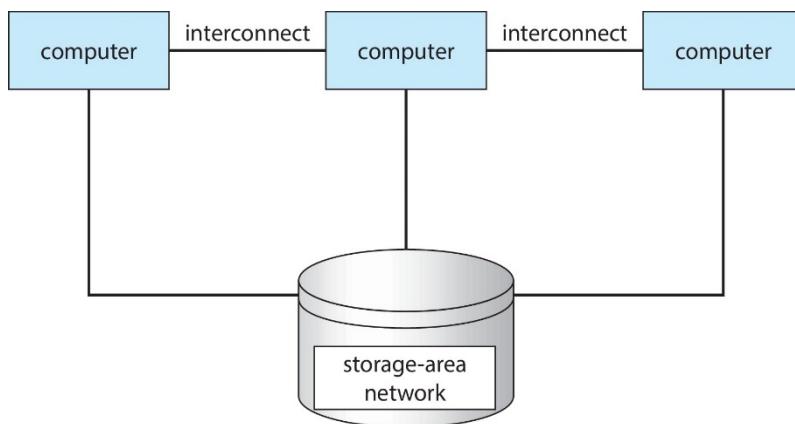
Edit with WPS Office

## 2. Multiprocessor Systems(single-pc) :

- **Multiprocessors** systems growing in use and importance
  - Also known as **parallel systems, tightly-coupled systems**
  - Advantages include:
    1. **Increased throughput**
    2. **Economy of scale(1+1>2)**
    3. **Increased reliability** – graceful degradation or **fault tolerance**
- Two types:
  1. **Asymmetric Multiprocessing** – each processor is assigned a specific task.
  - Symmetric Multiprocessing** – each processor performs all tasks (but time-slice)

## 3. Clustered Systems(multi-pc)

- Like multiprocessor systems, but **multiple systems** working together
  - Usually sharing storage via a **storage-area network (SAN)**
  - Provides a **high-availability** service which survives failures
    - **Asymmetric clustering** has **one machine in hot-standby mode?**
    - **Symmetric clustering** has **multiple nodes** running applications, monitoring each other?
- Some clusters are for **high-performance computing (HPC)**
  - Applications must be written to use **parallelization**
- Some have **distributed lock manager (DLM)** to avoid conflicting operations



What is Macro and micro?

Macro and micro



Edit with WPS Office

In macro, all the processes(apps) are running parallel in the CPU.

In micro, all the processes are synchronized(one after another)(time-slice)

Timer: to prevent infinite loop (or process hogging resources)

Timer is set to interrupt the computer after some time period

From one process to another.

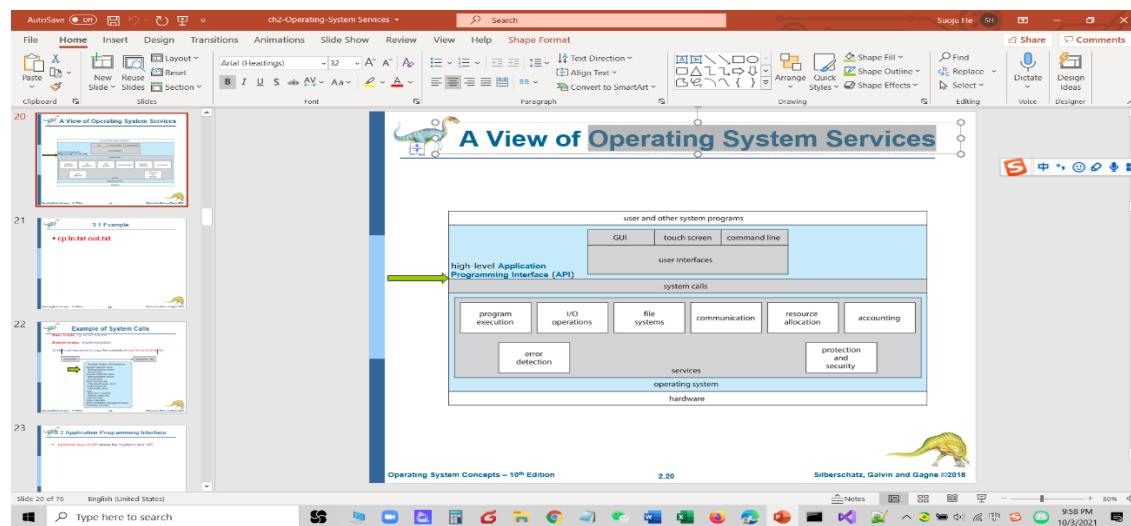
## CHAPTER 2

1Q :What are operating system services and Relation among GUI, system call and service?

1A. **Operating systems** provide an environment for **execution of programs** and **services to programs and users**

- **User interface (nothing to do with services)** - Almost all operating systems have a user interface (**UI**).
  - Varies between **Command-Line (CLI)**, **Graphics User Interface (GUI)**, **touch-screen**, **Batch**
- **Program execution** - The system must be able to **load a program into memory** and to **run that program**, end execution, either normally or abnormally (indicating error)
- **I/O operations** - A running program may require I/O, which may involve **a file** or an **I/O device**

**File-system manipulation** - The file system is of particular interest. Programs need **to read and write files and directories**, create and delete them, search them, list file Information, permission management.



Edit with WPS Office

## 2Q: Relation among Compiler, Linker and Loader?

5. Linkers and Loaders

- Compiler: Source code compiled into object files designed to be loaded into any physical memory location – relocatable object file
- Linker: combines these into single binary executable file
  - Also brings in libraries
- Loader: Program resides on secondary storage as binary executable
- Must be brought into memory by loader to be executed
  - Relocation assigns final addresses to program parts and adjusts code and data in program to match those addresses
- Modern general purpose systems don't link libraries into executables
  - Rather, dynamically linked libraries (in Windows, DLLs) are loaded as needed, shared by all that use the same version of that same library (loaded once)
- Object, executable files have standard formats, so operating system knows how to load and start them

Operating System Concepts – 10<sup>th</sup> Edition 2.43 Silberschatz, Galvin and Gagne ©2018

## Why Applications are Operating System Specific?

A View of Operating System Services

The diagram illustrates the layered architecture of the operating system:

- user and other system programs**: GUI, touch screen, command line, user interfaces
- high-level Application Programming Interface (API)**
- system calls**: program execution, I/O operations, file systems, communication, resource allocation, accounting
- operating system services**: error detection, protection and security
- hardware**

Operating System Concepts – 10<sup>th</sup> Edition 2.20 Silberschatz, Galvin and Gagne ©2018



Edit with WPS Office

## Relation among Compiler, Linker and Loader?

The screenshot shows a Microsoft Word document with the following content:

**5. Linkers and Loaders**

- Compiler: Source code **compiled** into **object files** designed to be **loaded** into any physical memory location – **relocatable object file**
- Linker: combines these into **single binary executable file**
  - Also brings in libraries
- Loader: Program resides on **secondary storage** as binary executable
- Must be **brought into memory** by **loader** to be executed
  - Relocation assigns final addresses to program parts and adjusts code and data in program to match those addresses
- Modern **general purpose systems** don't link libraries into executables
  - Rather, **dynamically linked libraries** (in Windows, **DLLs**) are loaded as needed, shared by all that use the same version of that same library (loaded once)
- Object, executable files have standard formats, so operating system knows how to load and start them

Operating System Concepts – 10<sup>th</sup> Edition 2.43 Silberschatz, Galvin and Gagne ©2018

Slide 42 of 76 English (United States) 1000 PM 10/3/2021

## Why Applications are Operating System Specific?

The screenshot shows a Microsoft Word document with the following content:

**6. Why Applications are Operating System Specific**

- Apps compiled on one system usually not executable on other operating systems
- Each operating system provides its own unique system calls
  - Own file formats, etc.
- Apps can be multi-operating system
  - Written in interpreted language like Python, Ruby, and interpreter available on multiple operating systems
  - App written in language that includes a VM containing the running app (like Java) C#/VB .net → .net framework
  - Use standard language (like C), compile separately on each operating system to run on each
- Application Binary Interface (ABI)** is architecture equivalent of API, defines how different components of binary code can interface for a given operating system on a given architecture, CPU, etc.

Operating System Concepts – 10<sup>th</sup> Edition 2.45 Silberschatz, Galvin and Gagne ©2018

Slide 43 of 76 English (United States) 1001 PM 10/3/2021

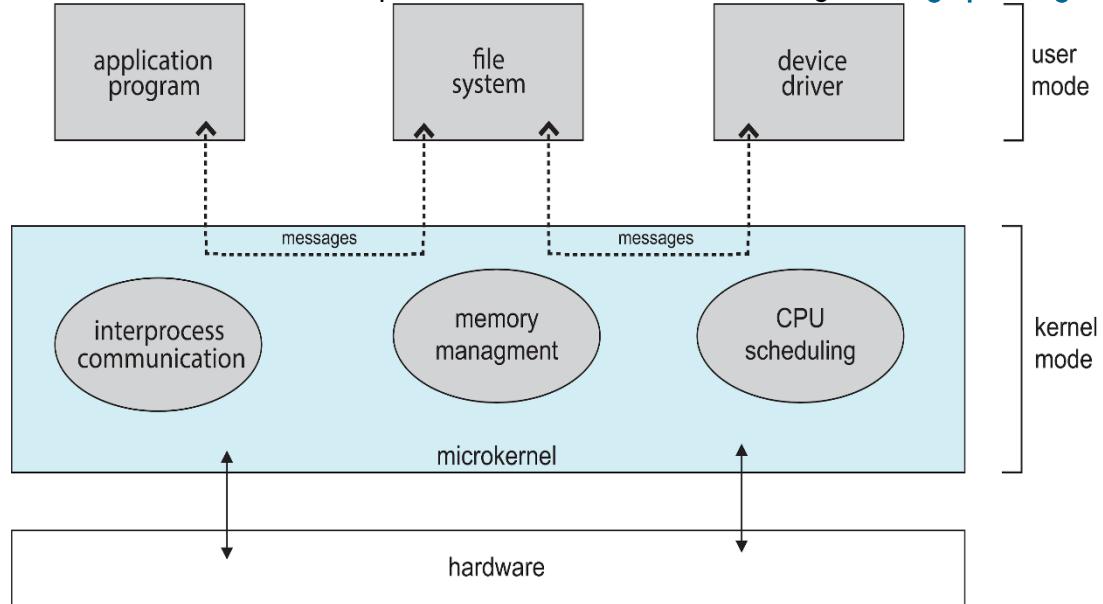


Edit with WPS Office

## What is Microkernels OS Operating System and its Structure?

- We have already seen that the **original UNIX system** had a **monolithic** structure.
- As UNIX expanded, the kernel became large and difficult to manage.
- In the mid-1980s, researchers at Carnegie Mellon University developed an operating system called **Mach** that modularized the **kernel** using the **microkernel** approach.
- This method structures the operating system by **removing all nonessential components from the kernel** and implementing them as **user-level programs** that reside in **separate address spaces**. The result is a smaller kernel.
- **Moves as much** from the kernel into **user space**
- **Mach** is an example of **microkernel**
  - Mac OS X kernel (**Darwin**) partly based on Mach

- Communication takes place between **user modules** using **message passing**



## CHAPTER 3

### What is the Process ?

a program in execution (in the RAM, with CPU/no CPU);  
process execution must progress in sequential fashion.  
No parallel execution of instructions of a single process

Program (code) is passive entity stored on disk

### Difference between process and program?

**Process:** process is **active**, program becomes **process** when an executable file is loaded into memory.

**Program (code)** is **passive** entity stored on disk (**executable file**); process is **active**

### Process States and their transition?

New: The process is being created

Running: Instructions are being executed

Waiting: The process is waiting for some event to occur

Ready: The process is waiting to be assigned to a processor

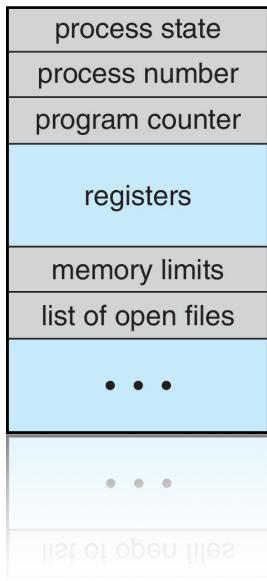
Terminated: The process has finished execution

### Process Control Block (PCB)(ID of process) ?

Information associated with each process(also called **task control block**)

- Process state – running, waiting, etc.

- Program counter – location of instruction to next execute
- CPU registers – contents of all process-centric registers?
- CPU scheduling information- priorities, scheduling queue pointers
- Memory-management information – memory allocated to the process
- Accounting information – CPU used, clock time elapsed since start, time limits
- I/O status information – I/O devices allocated to process, list of open files



## Process Threads ?

- So far, process has a single thread of execution
- Consider having multiple program counters per process(TCB)?
  - Multiple locations can execute at once
    - Multiple threads of control -> **threads**

- Must then have storage for thread details, multiple program counters in PCB

## Process Scheduling ?

Ready queue – set of all processes residing in main memory, ready and waiting to execute

Wait queues – set of processes waiting for an event (i.e., I/O)

Processes migrate among the various queues

## Context Switch ?

When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process via a context switch

Context of a process represented in the PCB

## Interprocess Communication ?

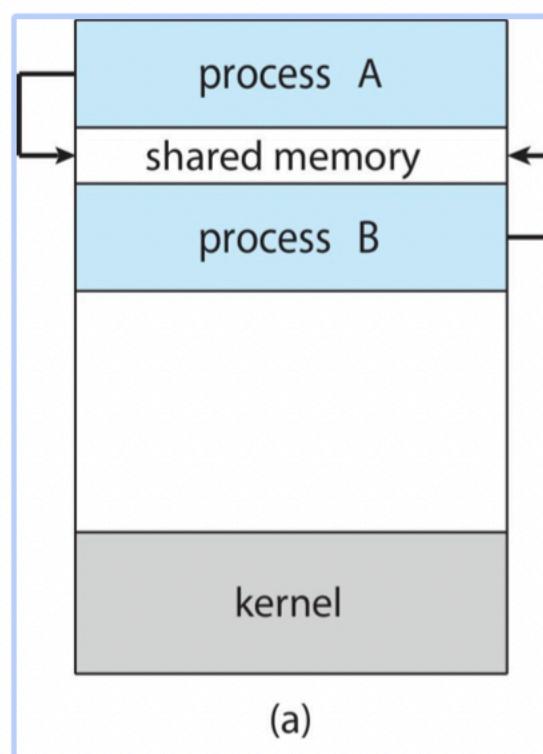
- Processes within a system(**local pc/remote**) may be **independent** or **cooperating**
- Cooperating process(ex: **google/google map in ios**) can affect or be affected by other processes, including sharing data
- Reasons for **cooperating processes**:
  - Information sharing
  - Computation speedup

- Modularity
- Convenience
- Cooperating processes need **interprocess communication (IPC)**
- Two models of IPC
  - **Shared memory**
  - **Message passing**

The process would be **app, object, or function**

**Shared Memory ?**

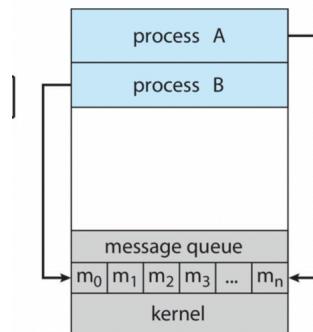
(a) Shared memory.      (b)



- Shared memory can be **faster than message passing**(*exchanging larger amounts of data*), since message-passing systems are typically implemented using system calls and thus **require the more time-consuming task of kernel intervention**.

## Message Passing ?

) Message passing.



- Message passing is useful for exchanging **smaller amounts of data**, because **no conflicts need be avoided?**.

## Producer-Consumer Problem ?

- Paradigm for cooperating processes:
  - *producer* process produces information that is consumed by a *consumer* process
- Two variations:
  - **unbounded-buffer** places no practical limit on the size of the buffer:
    - Producer never waits

- Consumer waits if there is no buffer to consume
- **bounded-buffer** assumes that there is a fixed buffer size
  - Producer must wait if all buffers are full
  - Consumer waits if there is no buffer to consume

**Why we say that Interprocess Communication is a Producer-Consumer Problem?**

**Process Communications in Client-Server Systems, IPC in diff PCs?**



## **8.(process)Communications in Client-Server Systems IPC in diff PCs?**

---

- 8.1 Sockets (**process/app in one pc to another process/app in another pc**)
- 8.2 Remote Procedure Calls (**same as the above**)
  
- **Web server: apache**(The default port of Apache is **80**)/**tomcat**(default port 8080)



## 8.2 Remote Procedure Calls

---

- Remote procedure call (RPC) abstracts procedure calls between processes on networked systems
  - Again uses ports for service differentiation
- Stubs – client-side proxy for the actual procedure on the server
- The client-side stub locates the server and marshalls the parameters
- The server-side stub receives this message, unpacks the marshalled parameters, and performs the procedure on the server
- On Windows, stub code compile from specification written in Microsoft Interface Definition Language (MIDL)

Remote  
Procedure Calls ?



## Remote Procedure Calls (Cont.) ?

- Data representation handled via **External Data Representation (XDL)** format to account for different architectures
  - **Big-endian** and **little-endian**
- Remote communication has more failure scenarios than local
  - Messages can be delivered **exactly once** rather than **at most once**
- OS typically provides a rendezvous (or **matchmaker**) service to connect client and server

## **Advance Operating Systems**

### **CHAPTER-4**

#### **1. Contrast or differ threads and processes?**

Answer:

- Most modern applications are multithreaded.
  - Thread: Threads run within application (process)
  - Process: A process is a program under execution i.e., an active program.
- 
- Multiple tasks within the application (process) can be implemented by separate threads (sub-contractor, multi-program counter)
  - Update display
  - Fetch data
  - Spell checking
  - Answer a network request
- 
- Process creation is heavy-weight while thread creation is light-weight
  - Can simplify code, increase efficiency
  - Kernels are generally multithreaded
  - Most software applications that run on modern computers and mobile devices are multithreaded.
  - An application typically is implemented as a separate process with several threads of control.

#### **2. Examples of multithreaded applications?**

Answer:

An application that creates photo thumbnails from a collection of images may use a separate thread to generate a thumbnail from each separate image.

A web browser might have :

- one thread displays images or text
- another thread retrieves data from the network.

A word processor may have :

- a thread for displaying graphics,
- another thread for responding to keystrokes from the user,
- and a third thread for performing spelling and grammar checking in the background.

### **3. Why we say there is no process scheduling, but thread scheduling?**

Answer:

A thread is a **lightweight process that can be managed independently by a scheduler**. Processes require more time for context switching as they are heavier. Threads require less time for context switching as they are lighter than processes. Processes are totally independent and don't share memory.

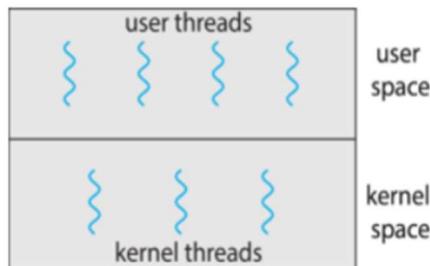
## 4. User Threads and Kernel Threads

Answer:

- User threads - management done by user-level threads library
  - Three primary thread libraries:
    - POSIX Pthreads
    - Windows threads
    - Java threads
  - Kernel threads - Supported by the Kernel
- 
- Examples – virtually all general -purpose operating systems, including:
  - Windows, Linux, Mac OS X, iOS, Android
  - Support for threads may be provided either at the user level, for user threads, or by the kernel, for kernel threads.
  - User threads are supported above the kernel and are managed without kernel support, whereas kernel threads are supported and managed directly by the operating system.
- 
- Virtually all contemporary operating systems—including Windows, Linux, and macOS—support kernel threads



### User and Kernel Threads



## 5. Multithreading Models and pro and con of each model

### Many-to-One

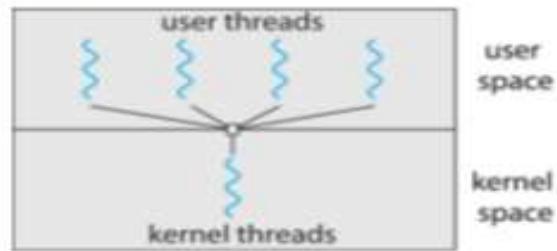
Answer:



### 3.1 Many-to-One

- Many user-level threads mapped to single kernel thread
- One thread blocking causes all to block (if one user's thread use kernel thread and block, and another user's thread want to call the kernel thread)
- Multiple threads may not run in parallel on multicore system because only one may be in kernel at a time?
- Few systems currently use this model
- Examples:
  - Solaris Green Threads
  - GNU Portable Threads

Multiple user threads  
Share one kernel thread  
(like one cpu), scheduling is required



- The many-to-one model (Figure 4.7) maps many user-level threads to one kernel thread.
- Thread management is done by the thread library in user space, so it is efficient (we discuss thread libraries in Section 4.4).
- However, the entire process (own multi-threads) will block if a thread makes a blocking system call (ex: read()), only after this, another user thread can call kernel thread).
- Also, because only one thread can access the kernel at a time, multiple threads are unable to run
- However, very few systems continue to use the model because of its inability to take advantage of multiple processing cores (in the kernel), which have now become standard on most computer systems in parallel on multicore systems.

## One-to-One

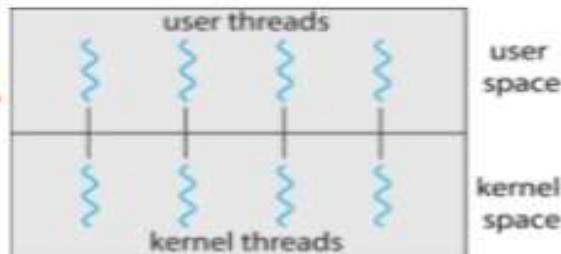
Answer:



### 3.2 One-to-One

- Each user-level thread maps to kernel thread
- Creating a user-level thread creates a kernel thread
- More concurrency than many-to-one
- Number of threads per process sometimes restricted due to overhead (thread context switch?)
- Examples
  - Windows
  - Linux

*Threads belongs to single process*



- It provides more concurrency than the many-to-one model by allowing another thread to run when a thread makes a **blocking system call**.
- It also allows multiple threads(**kernel**) to run in parallel on **multiprocessors**.
- The only drawback to this model:
  - is that creating a user thread requires creating the corresponding kernel thread, and a large number of kernel threads(**assign resource**) may burden the performance of a system.
- Linux, along with the family of Windows operating systems, implement the one-to-one model.

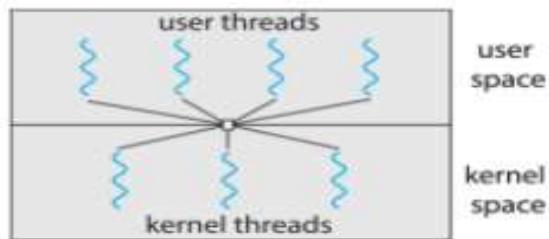
## Many-to-Many

Answer:



### 3.3 Many-to-Many Model

- Allows many **user level threads** to be mapped to many **kernel threads**
- Allows the **operating system** to create a **sufficient number of kernel threads**
- Windows with the *ThreadFiber* package
- Otherwise not very common



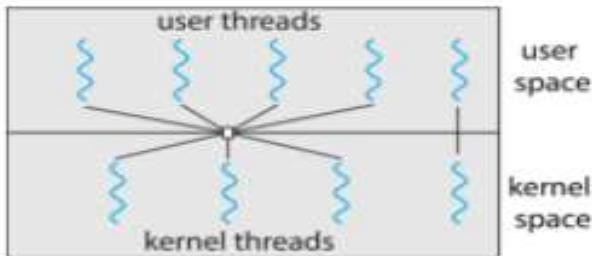
## Two-level Model

Answer:



### Two-level Model

- Similar to M:M, except that it allows a user thread to be **bound** to kernel thread



One variation on the many-to-many model still **multiplexes** many userlevel threads to a smaller or equal number of kernel threads but also allows a **user-level thread to be bound to a kernel thread**. This variation is sometimes referred to as the two-level model (Figure 4.10).

## 6. Why we have Implicit Threading? And its types?

Answer:



## 5. Implicit Threading

- With the continued growth of multicore processing, **applications containing hundreds**—or even thousands—of threads are looming on the horizon.
  - Designing such applications is **not a trivial undertaking**:
    - programmers must address not only the challenges outlined in Section 4.2 but additional difficulties as well.
    - These difficulties, which relate to program correctness, are covered in Chapter 6 and Chapter 8.
  - One way to address these difficulties and better support the design of **concurrent** and **parallel** applications is to transfer the creation and **management of threading** from application developers to compilers and run-time libraries (**system to take care of the detail, not the user**).
  - This strategy, termed **implicit threading**, is an increasingly popular trend
- 
- Growing in **popularity** as numbers of threads increase, program correctness more difficult with **explicit threads**
  - Creation and **management** of threads done by **compilers** and **run-time libraries** rather than **programmers**
  - **Five methods explored**
    - Thread Pools
    - Fork-Join
    - OpenMP
    - Grand Central Dispatch
    - Intel Threading Building Blocks



## 6. What are Thread Pools (resource pool) and its benefit?

Answer:



### 5.1 Thread Pools (resource pool)

- The general idea behind a thread pool is to **create a number of threads at start-up** and place them into a pool, where they sit and wait for work.
- When a server receives a request, rather than creating a thread, it instead submits the request to the thread pool and resumes waiting for additional requests.
- If there is an **available thread** in the pool, it is awakened, and the request is serviced immediately.
- If the pool contains **no available thread**, the **task** is queued until one becomes free.
- Once a thread completes its service, it returns to the pool and awaits more work.
- Create a number of threads in a pool where they await work
- Advantages:
  - Usually slightly faster to service a request with an existing thread than create a new thread(create real time take time)
  - Allows the number of threads in the application(s) to be bound to the size of the pool
  - Separating task to be performed from mechanics of creating task allows different strategies for running task??
    - ↳ i.e., Tasks could be scheduled to run periodically??
- Windows API supports **thread pools**:

```
DWORD WINAPI PoolFunction(VOID Param) {
    /*
     * this function runs as a separate thread.
     */
}
```

### 1a)Preemptive

used when a process switches from running state to ready state or from the waiting state(wait for I/O) to ready state

### 1b)Non preemptive Scheduling

Is used when a process terminates, or a process switches from running to the waiting state(wait for I/O)

## 2)Comparison of Preemptive and Nonpreemptive Scheduling

### **Comparison Chart:**

Parameter	PREEMPTIVE SCHEDULING	NON-PREEMPTIVE SCHEDULING
Basic	In this resources(CPU Cycle) are allocated to a process for a limited time.	Once resources(CPU Cycle) are allocated to a process, the process holds it till it completes its burst time or switches to waiting state.
Interrupt	Process can be interrupted in between.	Process can not be interrupted until it terminates itself or its time is up.
Starvation	If a process having high priority frequently arrives in the ready queue, a low priority process may starve.	If a process with a long burst time is running CPU, then later coming process with less CPU burst time may starve.
Overhead	It has overheads of scheduling the processes.	It does not have overheads.
Parameter	PREEMPTIVE SCHEDULING	NON-PREEMPTIVE SCHEDULING
Flexibility	flexible	rigid
Cost	cost associated	no cost associated
CPU Utilization	In preemptive scheduling, CPU utilization is high.	It is low in non preemptive scheduling.
Examples	Examples of preemptive scheduling are Round Robin and Shortest Remaining Time First.	Examples of non-preemptive scheduling are First Come First Serve and Shortest Job First.

### 3)Race Conditions and when does it happen?

They occur when two computer program processes, or threads, attempt to access the same resource at the same time and cause problems in the system. Race conditions are considered a common issue for multithreaded applications.

### 4)Why Preemptive scheduling can result in race conditions?

- Preemptive scheduling can result in race conditions when data are shared among several processes.
- Consider the case of two processes that share data.
- While one process is updating the data, it is preempted so that the second process can run.

The second process then tries to read the data, which are in an inconsistent state.

### 5)Scheduling Criteria

- **CPU utilization.** We want to keep the CPU as busy as possible. Conceptually, CPU utilization can range from 0 to 100 percent. In a real system, it should range from 40 percent (for a lightly loaded system) to 90 percent (for a heavily loaded system). (CPU utilization can be obtained by using the `top` command on Linux, macOS, and UNIX systems.)
- **Throughput.** If the CPU is busy executing processes, then work is being done. One measure of work is the number of processes that are completed per time unit, called **throughput**. For long processes, this rate may be one process over several seconds; for short transactions, it may be tens of processes per second.
- **Turnaround time.** From the point of view of a particular process, the important criterion is how long it takes to execute that process. The interval from the time of submission of a process to the time of completion is the turnaround time. Turnaround time is the sum of the periods spent waiting in the ready queue, executing on the CPU, and doing I/O.
- **Waiting time.** The CPU-scheduling algorithm does not affect the amount of time during which a process executes or does I/O. It affects only the amount of time that a process spends waiting in the ready queue. Waiting time is the sum of the periods spent waiting in the ready queue.
- **Response time.** In an interactive system, turnaround time may not be the best criterion. Often, a process can produce some output fairly early and can continue computing new results while previous results are being output to the user. Thus, another measure is the time from the submission of a request until the first response is produced. This measure, called response time, is the time it takes to start responding, not the time it takes to output the response.

## 6)Scheduling Algorithm Optimization Criteria

- Max CPU utilization
- Max throughput
- Min turnaround time
- Min waiting time
- Min response time

## 7)Scheduling Algorithms and their app

FCFS

SJF

RR

Priority Scheduling

Multi level queue scheduling

Multi level feedback queue

## 8)First- Come, First-Served (FCFS) Scheduling

**First Come First Serve (FCFS)** is an operating system scheduling algorithm that automatically executes queued requests and processes in order of their arrival

## 9)Shortest-Job-First (SJF) Scheduling

**Shortest Job First (SJF)** is an algorithm in which the process having the smallest execution time is chosen for the next execution. This scheduling method can be preemptive or non-preemptive.

## 10)Shortest-remaining-time-first preemptive

The shortest remaining time First (SRTF) algorithm is **preemptive version of SJF**. In this algorithm, the scheduler always chooses the processes that have the shortest expected remaining processing time. When a new process joins the ready queue it may in fact have a shorter remaining time than the currently running process.

## 11)Round Robin (RR)

In Round-robin scheduling, each ready task runs turn by turn in a cyclic queue for a limited time slice. This algorithm also offers starvation free execution of processes

## 12)Priority Scheduling(no-preemptive)

In this type of scheduling method, the CPU has been allocated to a specific process. The process that keeps the CPU busy, will release the CPU either by switching context or terminating. It is the only method that can be used for various hardware platforms. That's because it doesn't need special hardware (for example, a timer) like preemptive scheduling.

## 13)Priority Scheduling(preemptive)

In priority Preemptive Scheduling, the tasks are mostly assigned with their priorities. Sometimes it is important to run a task with a higher priority before another lower priority task, even if the lower priority task is still running. The lower priority task holds for some time and resumes when the higher priority task finishes its execution.

## 14)Priority Scheduling w/ Round-Robin

The slide contains a title 'Priority Scheduling w/ Round-Robin' with a small cartoon dinosaur icon. Below the title is a table showing processes P<sub>1</sub> through P<sub>5</sub> with their burst times and priorities:

Process	Burst Time	Priority
P <sub>1</sub>	4	3
P <sub>2</sub>	5	2
P <sub>3</sub>	8	2
P <sub>4</sub>	7	1
P <sub>5</sub>	3	3

Below the table are two bulleted points:

- Run the process with the highest priority. Processes with the same priority run round-robin
- Gantt Chart with time quantum = 2 (p<sub>2</sub> might arrive early than p<sub>3</sub>)

A Gantt chart is shown at the bottom, representing time from 0 to 27 units. The chart shows processes P<sub>4</sub>, P<sub>2</sub>, P<sub>3</sub>, P<sub>2</sub>, P<sub>3</sub>, P<sub>2</sub>, P<sub>3</sub>, P<sub>1</sub>, P<sub>5</sub>, P<sub>1</sub>, P<sub>5</sub>. The chart indicates that P<sub>2</sub> arrives at time 7 and P<sub>3</sub> arrives at time 11.

The slide is part of a larger presentation with other slides visible on the left:

- Slide 43: Priority Scheduling w/ Round-Robin
- Slide 44: 3.5 Multilevel Queue Scheduling
- Slide 45: Multilevel Queue
- Slide 46: 3.6 Multilevel Feedback Queue

At the bottom of the slide, there is footer information: 'Operating System Concepts – 10<sup>th</sup> Edition', '5.43', 'Silberschatz, Galvin and Gagne ©2018', and a Windows taskbar at the very bottom.

## 15) Multilevel Queue Scheduling

ch5-CPU Scheduling\_1\_32 - PowerPoint

File Home Insert Draw Design Transitions Animations Slide Show Review View Recording Help Font PDF Tell me what you want to do

43 Priority Scheduling w/ Round-Robin

44 3.5 Multilevel Queue Scheduling

45 Multilevel Queue

46 3.6 Multilevel Feedback Queue

### 3.5 Multilevel Queue Scheduling

- With priority scheduling, have separate queues for each priority.
- Schedule the process in the highest-priority queue!

priority = 0  $T_0, T_1, T_2, T_3, T_4$

priority = 1  $T_5, T_6, T_7$

priority = 2  $T_8, T_9, T_{10}, T_{11}$

⋮

⋮

⋮

priority = n  $T_x, T_y, T_z$

Operating System Concepts – 10<sup>th</sup> Edition 5.44 Silberschatz, Galvin and Gagne ©2018

Type here to search 9:30 PM 65°F Rain coming 10/11/2021

ch5-CPU Scheduling\_1\_32 - PowerPoint

File Home Insert Draw Design Transitions Animations Slide Show Review View Recording Help Font PDF Tell me what you want to do

43 Priority Scheduling w/ Round-Robin

44 3.5 Multilevel Queue Scheduling

45 Multilevel Queue

46 3.6 Multilevel Feedback Queue

### Multilevel Queue

- Prioritization based upon process type

highest priority

real-time processes

system processes

interactive processes

batch processes

lowest priority

Operating System Concepts – 10<sup>th</sup> Edition 5.45 Silberschatz, Galvin and Gagne ©2018

Type here to search 9:31 PM 65°F Rain coming 10/11/2021

## 16) Multilevel Feedback Queue

- A process can move between the various queues.

- Multilevel-feedback-queue scheduler defined by the following parameters:
  - Number of queues
  - Scheduling algorithms for each queue
  - Method used to determine when to upgrade a process
  - Method used to determine when to demote a process
  - Method used to determine which queue a process will enter when that process needs service
- Aging can be implemented using multilevel feedback queue

## 17)Thread Scheduling: Contention Scope

**4.1 Contention Scope**

- Distinction between **user-level** and **kernel-level** threads
- When threads supported, **threads scheduled**, not processes
- **process-contention scope (PCS)**: Many-to-one and many-to-many models, **thread library** schedules user-level threads to run on **LWP(like a virtual processor)**
  - Known as **process-contention scope (PCS)** since **scheduling competition** is within the process
  - Typically done via priority set by programmer
- **system-contention scope (SCS)**: Kernel thread scheduled onto available CPU is **system-contention scope (SCS)** – **competition among all threads in system**

## 18)process-contention scope (PCS)

process-contention scope (PCS) : Many-to-one and many-to-many models, thread library schedules user-level threads to run on LWP(like a virtual processor)

Known as process-contention scope (PCS) since scheduling competition is within the process.

Typically done via priority set by programmer.

User lever thread in one process share/compete for one LWP which associates with one kernel thread.

#### 19)system-contention scope (SCS)

system-contention scope (SCS): Kernel thread scheduled onto available CPU is system-contention scope (SCS) – competition among all threads in system.

system-contention scope (SCS): multiple kernel threads share/compete one cpu core.

#### 20)Differ PCS & SCS?

~~Q. DISTINGUISH BETWEEN PCS AND SCS SCHEDULING.~~

**Answer:**

PCS scheduling is done local to the process. It is how the thread library schedules threads onto available LWPs. SCS scheduling is the situation where the operating system schedules kernel threads. On systems using either many-to-one or many-to-many, the two scheduling models are fundamentally different. On systems using one-to-one, PCS and SCS are the same

## 21)Multiple-Processor Scheduling(multi-core)

Chrome File Edit View History Bookmarks Profiles Tab Window Help

drive.google.com/drive/folders/1MtJhzkvmBO13ymwNMpRhmqQ85QjzVqlp

### 5.1 Multiple-Processor Scheduling

Approaches to Multiple-Processor Scheduling(kernel level)

- Symmetric multiprocessing (SMP) is where each processor is **self scheduling**. (**each processor does its individual schedule**)
- All threads may be in a **common ready queue** (a)
- Each processor may have its own **private queue of threads** (b)

Operating System Concepts – 10<sup>th</sup> Edition 5.57 Silberschatz, Galvin and Gagne ©2018

Page 57 / 122

Screen Shot 2021-09...4.24 PM

Screen Shot 2021-10...12.32 PM

Screen Shot 2021-10...4.52 PM

Screen Shot 2021-10...53.51 PM

Threads & Concurrency

Synchronization

1 2 3

Word File Edit View Insert Format Tools Table Window Help

drive.google.com/drive/folders/1MtJhzkvmBO13ymwNMpRhmqQ85QjzVqlp

### 5.2 Multicore Processors

- (individual thread queue for each processor)** The standard approach for supporting multiprocessors is **symmetric multiprocessing (SMP)**, where each processor is self-scheduling.
- (one thread queue for all processors)** Scheduling proceeds by having the **scheduler** for each processor examine the **ready queue** and select a thread to run.
- Note that this provides **two possible strategies** for organizing the threads eligible to be scheduled: ??

Operating System Concepts – 10<sup>th</sup> Edition 5.58 Silberschatz, Galvin and Gagne ©2018

Page 58 / 122

Screen Shot 2021-09...4.24 PM

Screen Shot 2021-10...12.32 PM

Screen Shot 2021-10...4.52 PM

Threads & Concurrency

Synchronization

1 2 3

**5.2 Multicore Processors**

- Recent trend to place:
  - multiple processor cores on same physical chip, Faster and consumes less power
- Multiple threads(hardware) per core
- Multiple threads(hardware) per core also growing
  - Takes advantage of memory stall to make(CPU) progress (compute) on another thread while memory retrieve happens
- Figure

The diagram illustrates a sequence of cycles for a single thread. It shows a repeating pattern of 'compute cycle' (C) and 'memory stall cycle' (M). The timeline is indicated by a horizontal arrow labeled 'time'. The sequence of cycles is: C, M, C, M, C, M, C, M.

## 22) Why we should have Multithreaded Multicore System?

**Multithreaded Multicore System**

- Each core has > 1 hardware threads. (cpu usage improved)
- If one thread has a memory stall, switch to another thread!
- Figure

The diagram illustrates a single core being shared by two threads,  $\text{thread}_1$  and  $\text{thread}_0$ . Both threads exhibit a similar pattern of compute (C) and memory stall (M) cycles. A large green arrow points upwards between the two threads, with the text "Only one core for compute" written below it, emphasizing the shared nature of the core.

Chrome File Edit View History Bookmarks Profiles Tab Window Help

Tue Oct 12 4:18 PM

drive.google.com/drive/folders/1MtJhzkvmBO13ymwNMpRhmqO85QjzVqlp

Operating System Concepts – 10<sup>th</sup> Edition 5.63 Silberschatz, Galvin and Gagne ©2018

**Multithreaded Multicore System**

- Chip-multithreading (CMT) assigns each core multiple hardware threads (**can run in parallel, normally one core can run one task/thread at one time-slice**). (Intel refers to this as hyperthreading)

processor

core\_0  
hardware thread  
hardware thread

core\_1  
hardware thread  
hardware thread

core\_2  
hardware thread  
hardware thread

core\_3  
hardware thread  
hardware thread

operating system view

CPU\_0 CPU\_1 CPU\_2 CPU\_3  
CPU\_4 CPU\_5 CPU\_6 CPU\_7

Silberschatz, Galvin and Gagne ©2018

Page 64 / 122

Threads & Pre-emption

Synchronization Issues

Concurrency

Download All

Sign In

Screen Shot 2021-09...4.24 PM

Screen Shot 2021-10...12.32 PM

Screen Shot 2021-10...4.52 PM

Screen Shot 2021-10...53.51 PM

1 2 3

Mac OS X Dock icons

Chrome File Edit View History Bookmarks Profiles Tab Window Help

Tue Oct 12 4:18 PM

drive.google.com/drive/folders/1MtJhzkvmBO13ymwNMpRhmqO85QjzVqlp

Operating System Concepts – 10<sup>th</sup> Edition 5.64 Silberschatz, Galvin and Gagne ©2018

**Multithreaded Multicore System**

- Two levels of scheduling:
  - The operating system deciding which software thread to run on a logical CPU
  - How each core decides which hardware thread to run on the physical core.

software threads

hardware threads (logical processors)

processing core

level 1

level 2

Silberschatz, Galvin and Gagne ©2018

Threads & Pre-emption

Synchronization Issues

Concurrency

Download All

Sign In

Screen Shot 2021-09...4.24 PM

Screen Shot 2021-10...12.32 PM

Screen Shot 2021-10...4.52 PM

Screen Shot 2021-10...53.51 PM

1 2 3

Mac OS X Dock icons

23) Why we need Load Balancing in Multiple-Processor Scheduling?

If SMP (symmetric multiprocessing, each processor has its own scheduler), need to keep all CPUs loaded for efficiency.

Load balancing attempts to keep workload evenly distributed.

Push migration – periodic task checks load on each processor, and if found pushes task from overloaded CPU to other CPUs.

Pull migration – idle processors pulls waiting task from busy processor.

24) What is Processor Affinity in Multiple Processor Scheduling?

When a thread(task) has been running on one processor, the cache contents of that processor stores the memory accesses by that thread.

We refer to this as a thread having affinity for a processor (i.e., “processor affinity”).

25) Why Load balancing may affect processor affinity?

Load balancing may affect processor affinity as a thread may be moved from one processor to another to balance loads, yet that thread loses the contents of what it had in the cache of the processor it was moved off .

Soft affinity – the operating system attempts to keep a thread running on the same processor, but no guarantees.

Hard affinity – allows a process to specify a set of processors it may run on.

## 26)Real-Time CPU Scheduling

Can present obvious challenges.

Soft real-time systems – Critical real-time tasks have the highest priority, but no guarantee as to when tasks will be scheduled.

Hard real-time systems – task must be serviced by its deadline.

## 27)Soft real-time systems

Soft real-time systems – Critical real-time tasks have the highest priority, but no guarantee as to when tasks will be scheduled.

## 28)Hard real-time systems

Hard real-time systems – task must be serviced by its deadline.

## 29)What are two types of latencies that affect real-time systems performance?

The screenshot shows a Mac desktop with a Google Drive presentation open in a Chrome browser window. The presentation title is "Real-Time CPU Scheduling". It contains the following text and diagram:

- Event latency – the amount of time that elapses from when an event occurs to when it is serviced.
- Two types of latencies affect performance
  1. Interrupt latency(RR) – time from arrival of interrupt to start of routine that services interrupt
  2. Dispatch latency – time for scheduler to take current process off CPU and switch to another

A timeline diagram illustrates the sequence of events:  
event E first occurs at time  $t_0$   
event latency is the time between  $t_0$  and  $t_1$   
real-time system responds to E at time  $t_1$

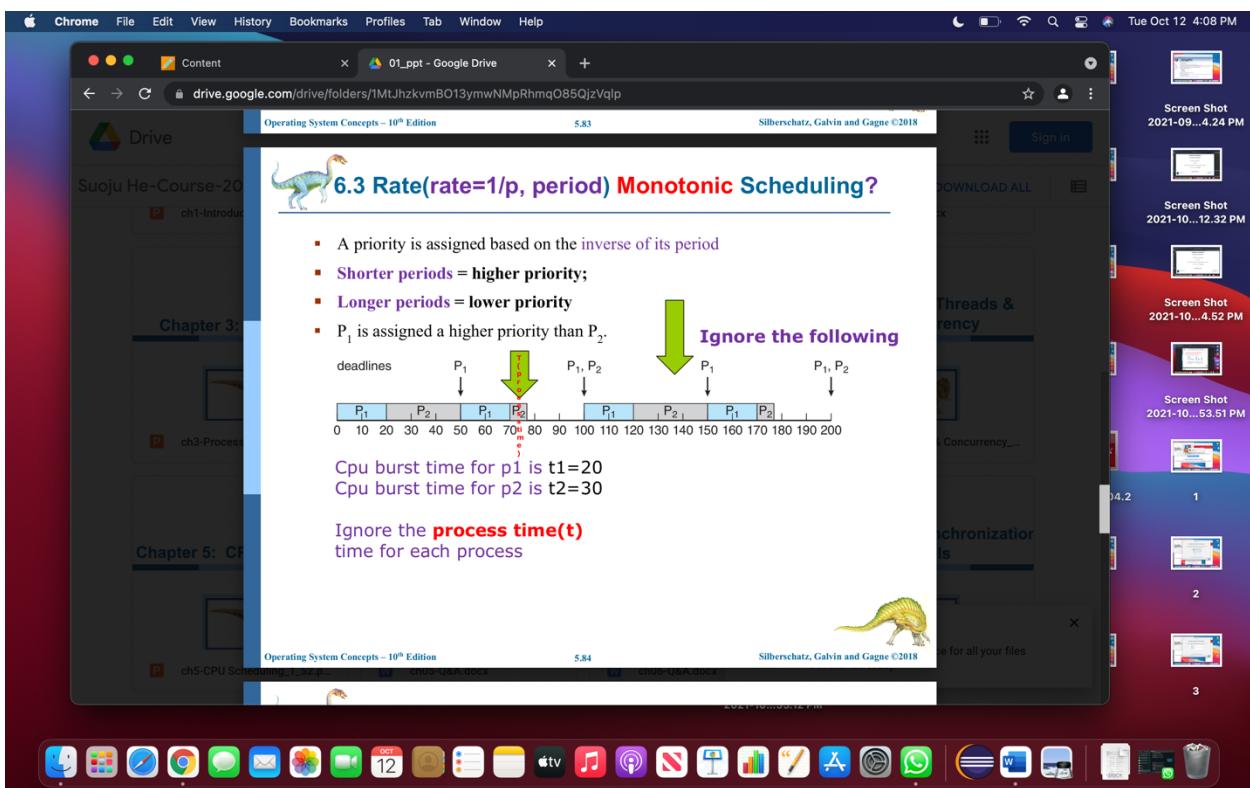
The desktop background shows a list of screen shots and files. The Dock at the bottom contains various application icons.

30) Why Real-Time CPU Scheduling is Priority-based Scheduling?

The most important feature of a real-time operating system is to respond immediately to a real-time process as soon as that process requires the CPU.

As a result, the scheduler for a real-time operating system must support a priority-based algorithm with preemption.

### 31)Rate Monotonic Scheduling



### 32)Earliest Deadline First Scheduling (EDF)

The screenshot shows a Mac desktop with a dark-themed window for a Google Drive presentation. The title slide is titled "6.4 Earliest Deadline First Scheduling (EDF)". It contains two bullet points under the heading "Figure":

- Priorities are assigned according to deadlines:
  - The earlier the deadline, the higher the priority
  - The later the deadline, the lower the priority

Below the text is a Gantt chart illustrating EDF scheduling. The x-axis represents time from 0 to 160. The y-axis lists processes P<sub>1</sub> and P<sub>2</sub>. The chart shows overlapping execution intervals for each process, with vertical arrows indicating their deadlines. P<sub>1</sub> has deadlines at 50, 70, 90, 110, and 130. P<sub>2</sub> has deadlines at 40, 60, 80, 100, 120, and 140. The processes are scheduled based on their earliest deadline.

### 33)Algorithm Evaluation

The screenshot shows a Mac desktop with a dark-themed window for a Google Drive presentation. The title slide is titled "8. Algorithm Evaluation". It contains several bullet points:

- How to select CPU-scheduling algorithm for an OS?
- Determine criteria, then evaluate algorithms
- Deterministic modeling
  - Type of **analytic evaluation**
  - Takes a particular predetermined workload and defines the performance of each algorithm for that workload
- Consider 5 processes arriving at time 0:

Process	Burst Time
P <sub>1</sub>	10
P <sub>2</sub>	29
P <sub>3</sub>	3
P <sub>4</sub>	7
P <sub>5</sub>	12

### 34)Deterministic modeling/Evaluation

Word File Edit View Insert Format Tools Table Window Help

drive.google.com/drive/folders/1MtJhzkvmBO13ymwNMpRhmqQ85QjzVqjp

Operating System Concepts – 10<sup>th</sup> Edition 5.11 Silberschatz, Galvin and Gagne ©2018

Suoju He-Course-2020 ch1-Introduct...

Chapter 3:...

ch3-Process...

Chapter 5: C...

ch5-CPU S...

Operating System Concepts – 10<sup>th</sup> Edition 5.11 Silberschatz, Galvin and Gagne ©2018

8.1 Deterministic Evaluation

- For each algorithm, calculate **minimum average waiting time**
- **Simple and fast**, but requires exact numbers for input, applies only to those inputs
  - FCS is 28ms(**minimum average waiting time**):

$P_1$	$P_2$	$P_3$	$P_4$	$P_5$	
0	10	39	42	49	61
  - Non-preemptive SJF is 13ms:

$P_3$	$P_4$	$P_1$	$P_5$	$P_2$	
0	3	10	20	32	61
  - RR is 23ms(quantum in 10ms):

$P_1$	$P_2$	$P_3$	$P_4$	$P_5$	$P_2$	$P_3$	$P_2$	
0	10	20	23	30	40	50	52	61

DOWNLOAD ALL

Threads & Precedence

Concurrency

Synchronization

File for all your files

Tue Oct 12 4:10 PM

Screen Shot 2021-09...4.24 PM

Screen Shot 2021-10...12.32 PM

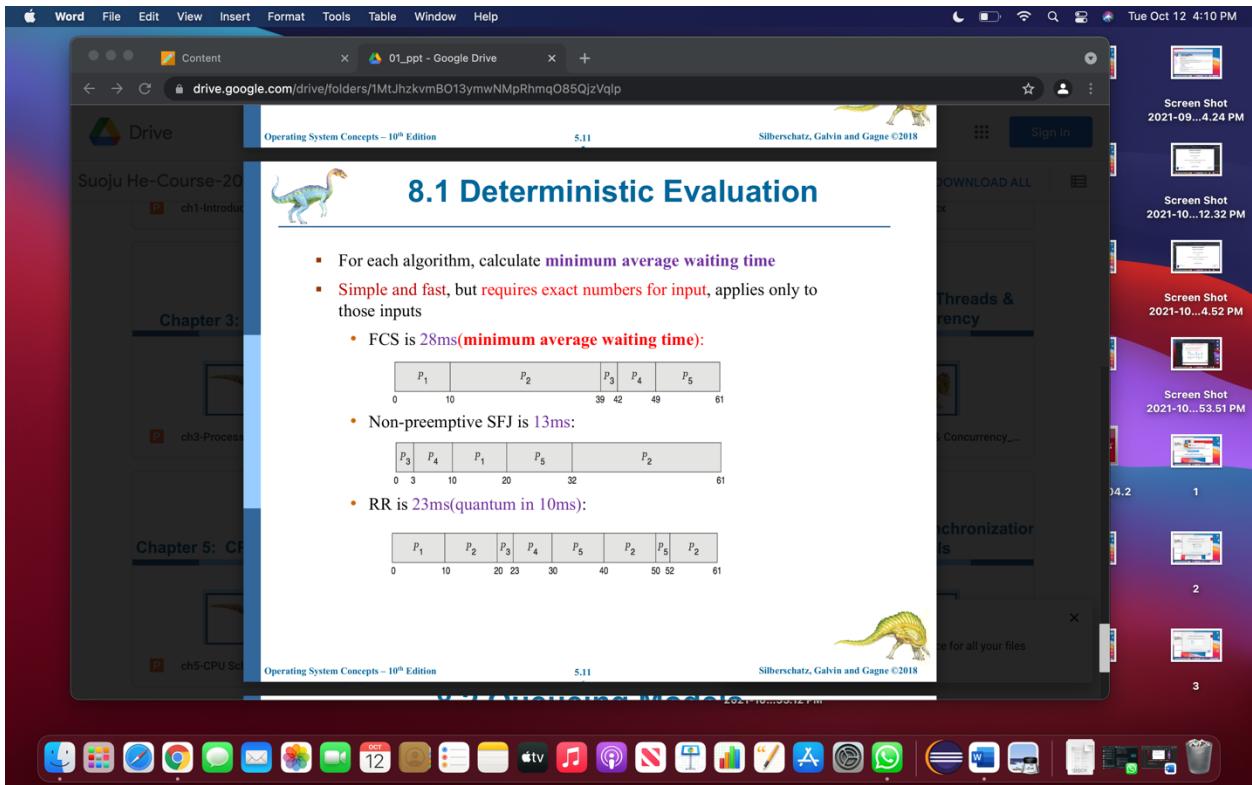
Screen Shot 2021-10...4.52 PM

Screen Shot 2021-10...53.51 PM

1

2

3



1Q.What is race condition? Give an example of it?

→ Several processes access and manipulate the **same data concurrently** and the **outcome of the execution** depends on the particular order in which the access takes place (read/write for same data) , is called a race condition.

- To guard against the race condition above, we need to ensure that **only one process** at a time can be manipulating the **variable count**.
- To make such a guarantee, we require that the **processes be synchronized(scheduled)** in some way.
- Ex: Writer process is not finishing writing, when reader starts to read.

2Q.Describe the critical-section problem and illustrate a race condition?

- Consider system of  **$n$**  processes  $\{p_0, p_1, \dots p_{n-1}\}$
- Each process has **critical section** segment of code
  - Process may be **changing common variables, updating table, writing file**, etc.
  - When one process in critical section, no other may be in its critical section
- **Critical section problem** is to **design protocol** to solve this
- Each process must **ask permission to enter critical section** in **entry section**, may follow critical section with **exit section**, then **remainder section**
  - **entry section**(process must ask permission to **enter CS**)
  - **critical section**
  - **exit section**

- remainder section

## CriticalSection

- General structure of process  $P_i$

```

while (true) {

    entry section

    critical section

    exit section

    remainder section

}

```

3Q. Requirements for solution to critical-section problem?

1. **Mutual Exclusion** - If process  $P_i$  is executing in its critical section, then no other processes can be executing in their critical sections
2. **Progress** - If no process is executing in its critical section and there exist some processes that wish to enter their critical section, then the selection of the process that will enter the critical section next cannot be postponed indefinitely (each process need to progress)
3. **Bounded Waiting** - A bound must exist on the number of times that other processes are allowed to enter their critical sections

after a process has made a request to enter its critical section and before that request is granted

- Assume that each process executes at a nonzero speed
- No assumption concerning **relative speed** of the  $n$  processes
- **Can not wait indefinitely?**

4Q.Can Interrupt-based Solution meet all the requirements for solution to critical-section problem? Why?

5.What is Software Solution No.1 (instructor names it to differ from Peterson's Solution)? Can this solution meet all the requirements for solution to critical-section problem? Why?



## 2.2 Software Solution 1

- Two process solution
  - The two processes share one variable:
    - `int turn;`
  - The variable `turn` indicates whose turn it is to enter the critical section
  - initially, the value of `turn` is set to  $i$
- 
- Assume that the **load** and **store** machine-language instructions are **atomic**; that is, **cannot be interrupted**?
  - The operands for all arithmetic and logic operations are contained in registers. To operate on data in main memory, the data is first copied into registers.
  - A **load** operation copies data from main memory into a register.
  - A **store** operation copies data from a register into main memory .



## Correctness of the Software Solution

- Mutual exclusion is preserved  
 $P_1$  enters critical section only if:  
$$\text{turn} = i$$
and `turn` cannot be both 0 and 1 at the same time
- What about the Progress requirement? (Not)
- What about the Bounded-waiting requirement? (Not)



Why Peterson's Solution is not guaranteed to work on modern architectures(multi-core)?



## Modern Architecture Example(02)

- Two threads share the data:

```
boolean flag = false;
int x = 0;
```
- Thread 1 performs

```
while (!flag)
;
print x //not print when condition not met
```
- Thread 2 performs //change the order

```
flag = true //Thread 1 print when condition met
x = 100;
```
- What is the expected output?  
0



## Peterson's Solution and Modern Architecture

- Although useful for demonstrating an algorithm, Peterson's Solution is not guaranteed to work on modern architectures(multi-core).
  - To improve performance, processors and/or compilers may reorder operations that have no dependencies
- Understanding why it will not work is useful for better understanding race conditions.
- For single-threaded this is ok as the result will always be the same.
- For multithreaded (multi-core) the reordering may produce inconsistent or unexpected results!



Why software solution cannot eventually solve CS problem on modern architectures?

8.Illustrate hardware solutions to the critical-section problem using memory barriers, Test-and-Set, compare-and-swap operations, and atomic variables

Demonstrate how mutex locks, semaphores can be used to solve the critical section problem

The definition of `acquire()` is as follows:

```
acquire() {  
    while (!available)  
        ; /* busy wait */ ←  
    available = false;  
}
```

The definition of `release()` is as follows:

```
release() {  
    available = true;  
}
```



## Solution to CS Problem Using **Mutex Locks**

```
while (true) {  
    acquire lock  
  
    critical section  
  
    release lock  
  
    remainder section  
}
```

Figure 6.10 Solution to the **critical-section** problem using mutex locks.



## Differ Mutex Locks and Semaphore

MUTEX	SEMAPHORE
A program object that allows multiple processes to take turns to share the same resource	A variable that is used to control access to a common resource by multiple processes in a concurrent system such as a multitasking operating system
Locking mechanism	Signaling mechanism
An object	An integer variable
No categorization	Categorized as binary and counting semaphores
If the mutex is locked, the process that requests the lock waits until the system releases the lock	If the semaphore value is 0, the process performs wait() operation until the semaphore becomes greater than 0
Process uses acquire() and release() to access and release mutex	Process uses wait() and signal() to modify semaphore

Visit [www.PEDIAA.com](http://www.PEDIAA.com)

What is the relation between mutex locks & semaphores and hardware solutions, such as Test-and-Set & compare-and-swap?

A mutex object allows multiple process threads to access a single shared resource but only one at a time. On the other hand, semaphore **allows multiple process threads to access the finite instance of the resource until available**. In mutex, the lock can be acquired and released by the same process at a time.

Test and set vs Compare and Swap

test-and-set **modifies the contents of a memory location** and returns its old value as a single atomic operation. compare-and-swap atomically compares the contents of a memory location to a given value and, only if they are the same, modifies the contents of that memory location to a given new value.