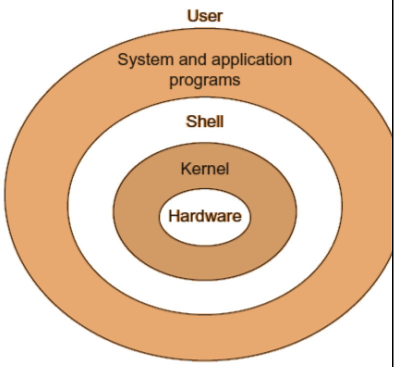
Linux Architecture with diagram:

Kernel

- The kernel is one of the core section of an operating system.
- It is responsible for each of the major actions of the Linux OS.
- The kernel facilitates required abstraction for hiding details of low-level hardware or application programs to the system.

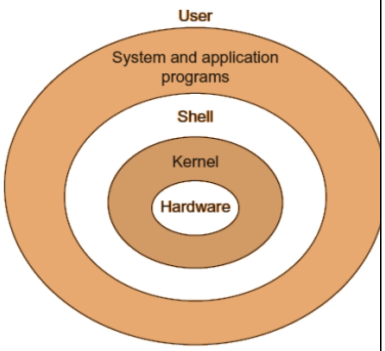


Hardware layer

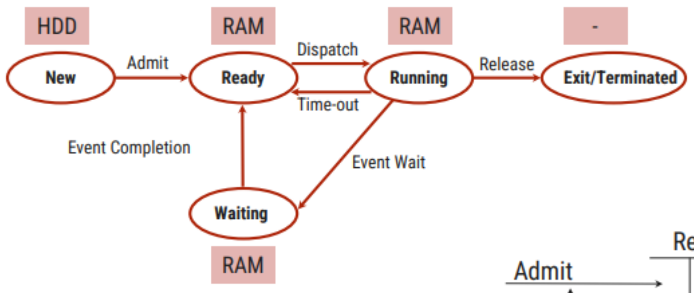
- Linux operating system contains a hardware layer that consists of several peripheral devices like CPU, HDD, and RAM.

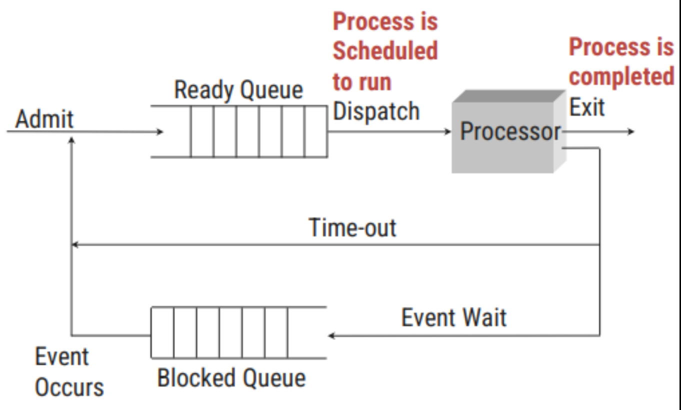
Shell

- It is an interface among the kernel and user.
- It can take commands through the user and runs the functions of the kernel.
- The shell is available in distinct types of OSes. These operating systems are categorized into two different types, which are the **graphical shells** and **command-line shells**.
- The graphical line shells facilitate the graphical user interface, while the command line shells facilitate the command line interface.
- Thus, both of these shells implement operations.
- However, the graphical user interface shells work slower as compared to the command-line interface shells.



5 State process model with diagram:





Functions Of OS:

1. User Interface ⊒This interface can take several forms. 1. DTrace command line interface - which uses text commands and a method for entering them (say, a program to allow entering and editing of commands). 2. Batch interface - in which commands and directives to control those commands are entered into files, and those files are executed. 3. Graphical user interface - the interface with a pointing device to direct I/0, choose from menus, and make selections and a keyboard to enter text. ⊒ Some systems provide two or all three of these variations

2. 2. Program Execution ⊒ The system must be able to load a program into memory and to run that program.The program must be able to end its execution, either normally or abnormally (indicating error).

3. 3. I/O Operations ⊒ A running program may require I/O, which may involve a file or an I/O device. ⊒ For efficiency and protection, users usually cannot control I/O devices directly. ⊒ Therefore, the operating system must provide a means to do I/O

4. 4. File Management ⊒The operating system manages resource allocation and de-allocation. ⊒It specifies which process receives the file and for how long. ⊒It also keeps track of information, location, uses, status, and so on. ⊒These groupings of resources are referred to as file systems. ⊒The files on a system are stored in different directories. ⊒The OS: ♣ Keeps records of the status and locations of files. ♣ Allocates and deallocates resources. ♣ Decides who gets the resources.

5. 5. Resource Allocation ⊒When there are multiple users or multiple jobs running at the same time, resources must be allocated to each of them. ⊒The operating system manages many different types of resources, such as CPU cycles, main memory, file storage, I/O devices

6. 6. Communication ⊇In multitasking environment, the processes need to communicate with each other and to exchange their information. ⊇Operating system performs the communication among various types of processes in the form of shared memory or message passing, in which packets of information in predefined formats are moved between processes by the operating system

7. 7. Error detection and response ⊇An error may occur in CPU, in I/O devices or in the memory hardware. ⊇Following are the major activities of an operating system with respect to error handling − ♣ The OS constantly checks for possible errors. ♣ The OS takes an appropriate action to ensure correct and consistent computing.

8. 8. Accounting ⊇We want to keep track of which users use how much and what kinds of computer resources. ⊇This record keeping may be used for accounting (so that users can be billed) or simply for accumulating usage statistics.

9. 9. Protection & Security ⊇Protection involves ensuring that all access to system resources is controlled. ⊇Security of the system from outsiders is also important. ⊇Such security starts with requiring each user to authenticate himself or herself to the system, usually by means of a password, to gain access to system resources. ⊇If a system is to be protected and secure, precautions must be instituted throughout it.

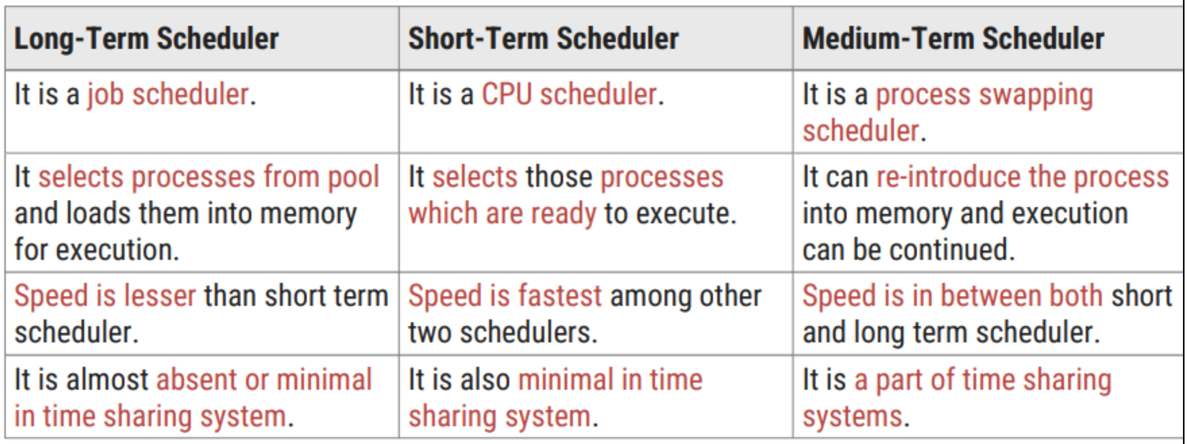
10. 10. Virtual Machine Manager ⊇ The fundamental idea behind a virtual machine is to abstract the hardware of a single computer (the CPU, memory, disk drives, network interface cards, and so forth) into several different execution environments, thereby creating the illusion that each separate execution environment is running its own private computer.

System Call and its Types:

A system call is a way for programs to interact with the operating system. } A system call is a mechanism that provides the interface between a process and the operating system. } A computer program makes a system call when it makes a request to the operating system's kernel. } It is a programmatic method in which a computer program requests a service from the kernel of the OS. } System call provides the services of the operating system to the user programs via Application Program Interface(API). } System calls are the only entry points for the kernel system

1. Process Control: This system calls perform the task of process creation, process termination, etc. ⊇ Functions: ♣ End and abort ♣ Load and execute ♣ Create process and terminate process ♣ Wait and signed event ♣ Allocate and free memory }
2. File Management: File management system calls handle file manipulation jobs like creating a file, reading, and writing, etc. ⊇ Functions: ♣ Create a file ♣ Delete file ♣ Open and close file ♣ Read, write and reposition ♣ Get and set file attributes
3. } Device Management: Device management does the job of device manipulation like reading from device buffers, writing into device buffers, etc. ⊇ Functions ♣ Request and release device ♣ Logically attach/ detach devices ♣ Get and Set device attributes }
4. Information Maintenance: It handles information and its transfer between the OS and user program. ⊇ Functions: ♣ Get or set time and date ♣ Get process and device attributes
5. }Communication: These types of system calls are specially used for interprocess communications (IPC). ⊇Functions: ♣ Create, delete communications connections ♣ Send, receive message ♣ Help OS to transfer status information ♣ Attach or detach remote devices

Types of Schedulers with example:



Long-Term Scheduler	Short-Term Scheduler	Medium-Term Scheduler
It is a job scheduler.	It is a CPU scheduler.	It is a process swapping scheduler.
It selects processes from pool and loads them into memory for execution.	It selects those processes which are ready to execute.	It can re-introduce the process into memory and execution can be continued.
Speed is lesser than short term scheduler.	Speed is fastest among other two schedulers.	Speed is in between both short and long term scheduler.
It is almost absent or minimal in time sharing system.	It is also minimal in time sharing system.	It is a part of time sharing systems.

Condition for deadlock and prevention:

- Deadlock can be defined as the permanent blocking of a set of processes that either compete for system resources or communicate with each other.
- A set of processes is deadlocked when each process in the set is blocked awaiting an event (typically the freeing up of some requested resource) that can only be triggered by another blocked process in the set.

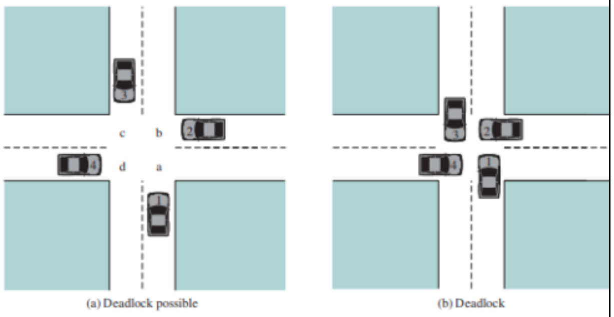


Figure 6.1 **Illustration of Deadlock**

Conditions for Deadlock

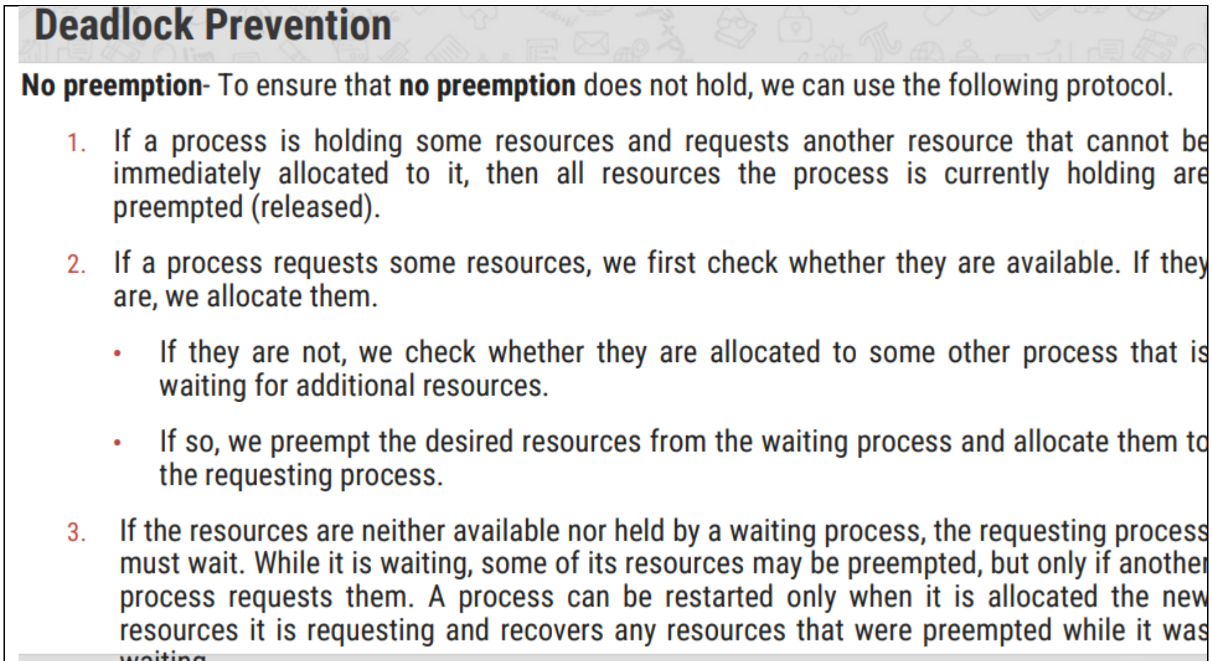
Four conditions must be present for a deadlock to be possible:

- **Mutual exclusion**- Only one process may use a resource at a time. No process may access a resource unit that has been allocated to another process.
- **Hold and wait**- A process may hold allocated resources while awaiting assignment of other resources.
- **No preemption**- No resource can be forcibly removed from a process holding it.
- **Circular wait**- A closed chain of processes exists, such that each process holds at least one resource needed by the next process in the chain

Deadlock Prevention

If at least one of these conditions cannot hold, we can prevent the occurrence of a deadlock.

- **Mutual exclusion**- To prevent deadlock, mutual exclusion of shareable resources must be prevented.
- **Hold and wait**- To ensure that the hold-and-wait condition never occurs in the system, we must guarantee that, whenever a process requests a resource, it does not hold any other resources.
  - Each process to request and be allocated all its resources before it begins execution.
  - Allows a process to request resources only when it has none. A process may request some resources and use them. Before it can request any additional resources, it must release all the resources that it is currently allocated.

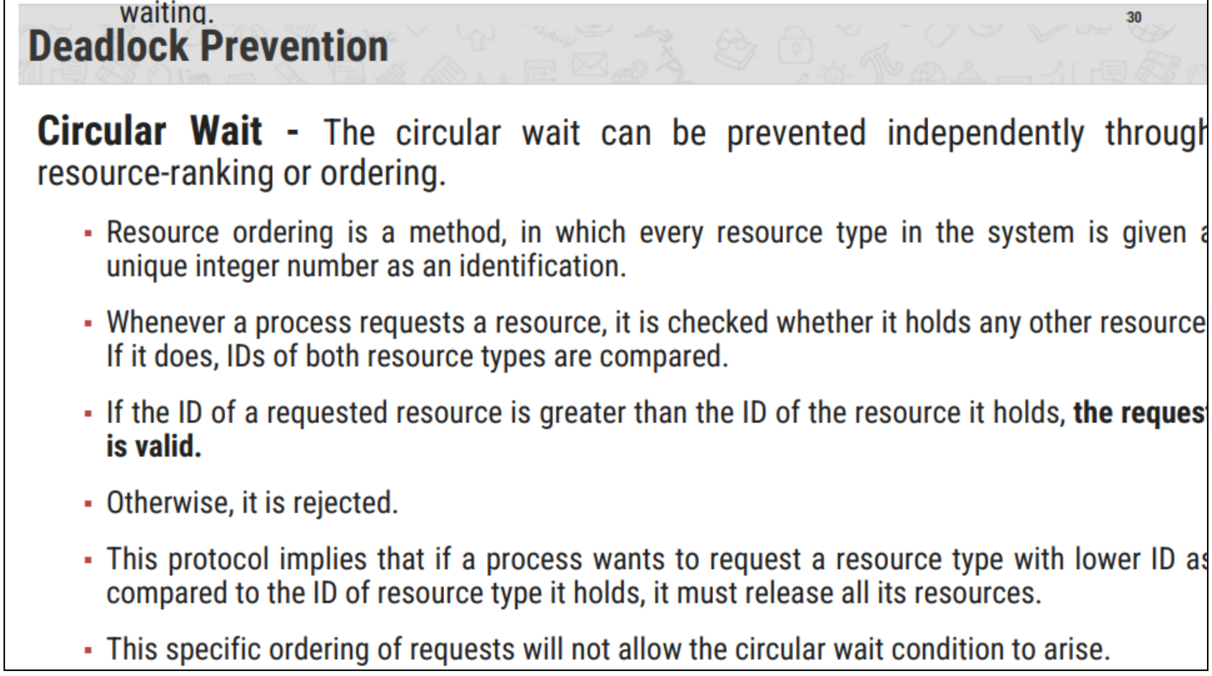


Deadlock Prevention

No preemption- To ensure that **no preemption** does not hold, we can use the following protocol.

1. If a process is holding some resources and requests another resource that cannot be immediately allocated to it, then all resources the process is currently holding are preempted (released).
2. If a process requests some resources, we first check whether they are available. If they are, we allocate them.
   - If they are not, we check whether they are allocated to some other process that is waiting for additional resources.
   - If so, we preempt the desired resources from the waiting process and allocate them to the requesting process.
3. If the resources are neither available nor held by a waiting process, the requesting process must wait. While it is waiting, some of its resources may be preempted, but only if another process requests them. A process can be restarted only when it is allocated the new resources it is requesting and recovers any resources that were preempted while it was waiting.

30



Deadlock Prevention

Circular Wait - The circular wait can be prevented independently through resource-ranking or ordering.

- Resource ordering is a method, in which every resource type in the system is given a unique integer number as an identification.
- Whenever a process requests a resource, it is checked whether it holds any other resource If it does, IDs of both resource types are compared.
- If the ID of a requested resource is greater than the ID of the resource it holds, **the request is valid.**
- Otherwise, it is rejected.
- This protocol implies that if a process wants to request a resource type with lower ID as compared to the ID of resource type it holds, it must release all its resources.
- This specific ordering of requests will not allow the circular wait condition to arise.

Types of Threads:

Scheduling algorithm/Sums/Bankers Algorithm:

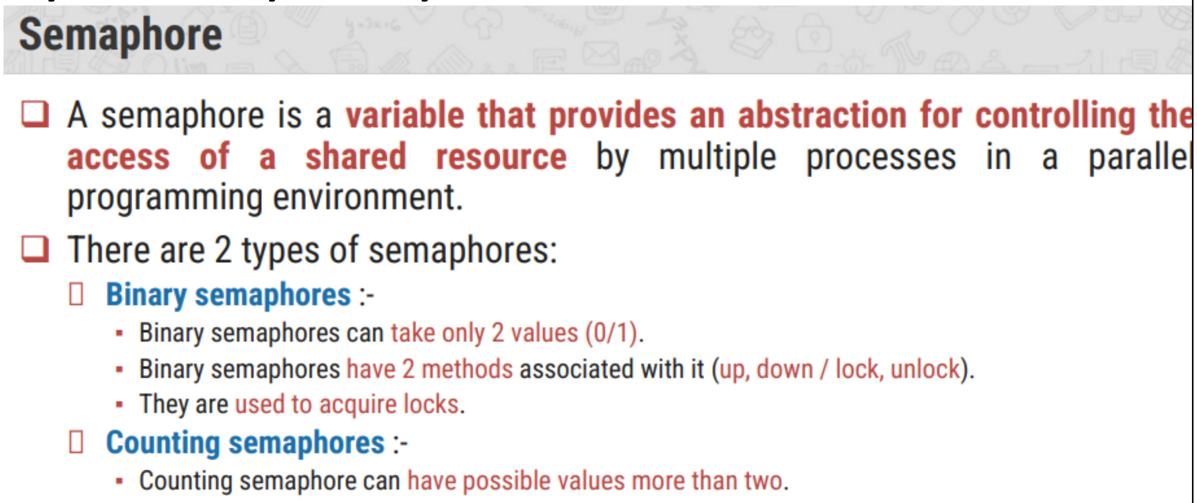
Race Around Condition:

The situation where several processes access and manipulate shared data concurrently. The final value of the shared data depends upon which process finishes last.
A race condition is an undesirable situation that occurs when a device or system attempts to perform two or more operations at the same time. But, because of the nature of the device or system, the operations must be done in the proper sequence to be done correctly. To prevent race conditions, concurrent processes must be synchronized

Characteristics of Deadlock:

Mutal Exclusion, Hold and wait, No Pre-emption, Circular Hold

Explain Semaphore operation with codes:



Semaphore

- A semaphore is a **variable that provides an abstraction for controlling the access of a shared resource** by multiple processes in a paralle programming environment.
- There are 2 types of semaphores:
  - **Binary semaphores** :-
    - Binary semaphores can take only 2 values (0/1).
    - Binary semaphores have 2 methods associated with it (up, down / lock, unlock).
    - They are used to acquire locks.
  - **Counting semaphores** :-
    - Counting semaphore can have possible values more than two.

Operations on Semaphore

Wait(): a process performs a wait operation to **tell the semaphore that it wants exclusive access to the shared resource**.

- If the value of semaphore is one, then the semaphore value is changed to zero and allows the process to continue its execution immediately.

Signal(): a process performs a signal operation to **inform the semaphore that it is finished using the shared resource**.

- If there are processes suspended on the semaphore, the semaphore value is changed to one.

The Producer Consumer Problem

It is **multi-process synchronization** problem.

It is also known as **bounded buffer problem**.

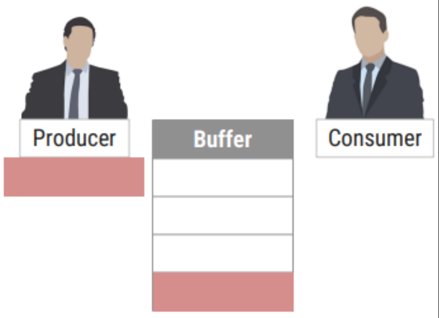
This problem describes two processes producer and consumer, who share common, fixed size buffer.

Producer process

- Produce some information and put it into buffer

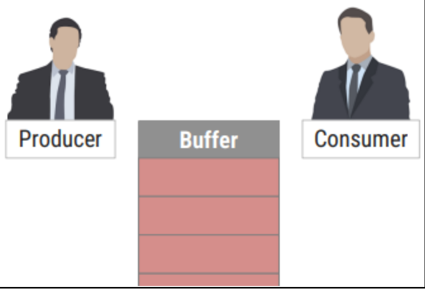
Consumer process

- Consume this information (remove it from the buffer)



What Producer Consumer problem is?

The Problem: ensure that the producer can't add data into full buffer and consumer can't remove data from an empty buffer



Producer Consumer problem Solution

- **The <u>solution for the producer is to go to sleep</u> <u>if the buffer is</u> <u>full.</u>** The next time the consumer removes an item from the buffer, it wakes up the producer who starts to fill the buffer again.

- **In the same way, <u>the consumer goes to sleep if it finds the</u> <u>buffer</u> <u>to be</u> <u>empty</u>**. The next time the producer puts data into the buffer, it wakes up the sleeping consumer. The solution can be reached by means of inter-process communication, typically using semaphores. An inadequate solution could result in a deadlock where both processes are waiting to be awakened.

Producer Consumer problem's Solution with Semaphore

The producer process after producing the item waits on the **empty semaphore** because it needs to check whether there is an empty slot in the buffer.

If there is a slot, the semaphore allows it to go further.

Once it passes this condition, it waits on the **Buffer_access semaphore** to check whether any other process is accessing it.

If any other process is already accessing it, then the semaphore will not allow it.

After getting the permission, the process starts accessing and storing the item in the buffer.

After storing**, it signals the Buffer_access so that any other process in wait** can access the buffer.

Moreover, it also signals the full semaphore to indicate that the buffer now contains one item.

In this way, the producer stores the item in the buffer and also updates the status of the buffer. Similarly, the algorithm of the consumer can be understood.

```
procedure producer() {
  while (true) {
      item = produceItem()

       if (itemCount == BUFFER_SIZE) {
           sleep()
       }

      putItemIntoBuffer(item)
       itemCount = itemCount + 1

      if (itemCount == 1) {
           wakeup(consumer)
       }
   }
}
procedure consumer() {
   while (true) {

       if (itemCount == 0) {
           sleep()
       }

       item = removeItemFromBuffer()
       itemCount = itemCount - 1

       if (itemCount == BUFFER_SIZE - 1) {
           wakeup(producer)
       }

       consumeItem(item)
   }
}
```