1. Different IPC mechanisms in QNX & Linux.

ans:

Linux IPC mechanisms:

Linux supports three types of interprocess communication mechanisms which first appeared in Unix System V (1983).

These are message queues, semaphores and shared memory.

Mesaage queues : Message queues allow one or more processes to write messages which will be read by one or more reading processes.

sempaphores : Semaphores can be used to implement critical regions, areas of critical code which only one process at a time should be executing.

shared memory : Shared memory allows one or more processes to communicate via memory that appears in all of their virtual address spaces.

The pages of the virtual memory is referenced by page table entries in each of the sharing processes page tables.It does not have

to be at the same address in all of the processes virtual memory.

QNX IPC mechanisms:

Service: Implemented in:

Message-passing Kernel

Signals Kernel

POSIX message queues External process

Shared memory Process manager

Pipes External process

FIFOs External process

2.Discussion on Shared Memory

ans:

Processes and threads can communicate directly with one another by sharing parts of their memory space and then reading and writing the data

stored in the shared memory.

Synchronization of shared memory is the responsibility of the application program.Shared memory is a memory shared between two or more processes.

shared memory-->1.process1, 2.process2

1. Create the shared memory segment or use an already created shared memory segment (shmget())

2. Attach the process to the already created shared memory segment (shmat())

3 .Detach the process from the already attached shared memory segment (shmdt())

4. Control operations on the shared memory segment (shmctl())

3..What is greb loader

ans:

1. GRUB stands for GRand Unified Bootloader. Its function is to take over from BIOS at boot time, load itself, load the Linux kernel into memory,

and then turn over execution to the kernel. Once the kernel takes over, GRUB has done its job and it is no longer needed.

2. GRUB is a multi boot boot-loader, is a programm who run at the system start and allow you to select what Operation system you want to start,

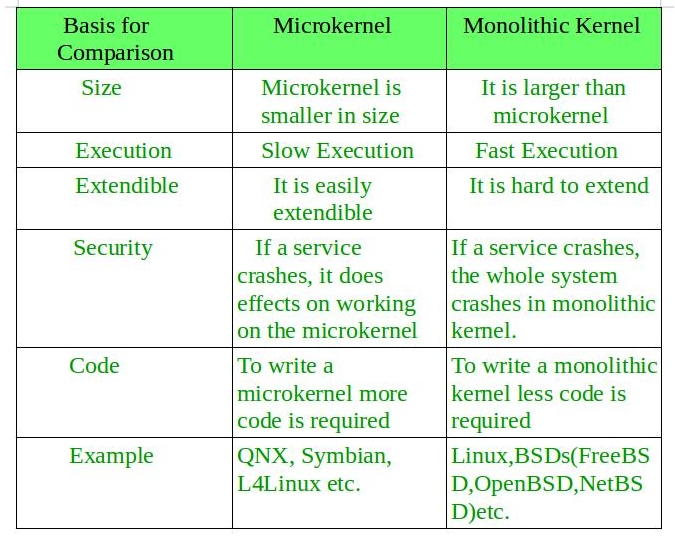
like if you want ubuntu and windows on the same computer, with GRUB you can select those at system start.

3. Boot loader is a program which is called by BIOS and once again initializes. boot related hardware and finally boot loader is the one

who exactly Knows(memory location) where the Kernel image is.

4. difference between monolithical and micro kernel?

ans:



5. Difference between process and thread?

Ans:

| S.NO | Process | Thread |
| --- | --- | --- |
| 1. | Process means any program is in execution. | Thread means a segment of a process. |
| 2. | The process takes more time to terminate. | The thread takes less time to terminate. |
| 3. | It takes more time for creation. | It takes less time for creation. |
| 4. | It also takes more time for context switching. | It takes less time for context switching. |
| 5. | The process is less efficient in terms of communication. | Thread is more efficient in terms of communication. |
| 6. | Multiprogramming holds the concepts of multi-process. | We don’t need multi programs in action for multiple threads because a single process consists of multiple threads. |
| 7. | The process is isolated. | Threads share memory. |
| 8. | The process is called the heavyweight process. | A Thread is lightweight as each thread in a process shares code, data, and resources. |
| 9. | Process switching uses an interface in an operating system. | Thread switching does not require calling an operating system and causes an interrupt to the kernel. |
| 10. | If one process is blocked then it will not affect the execution of other processes | If a user-level thread is blocked, then all other user-level threads are blocked. |
| 11. | The process has its own Process Control Block, Stack, and Address Space. | Thread has Parents’ PCB, its own Thread Control Block, and Stack and common Address space. |
| 12. | Changes to the parent process do not affect child processes. | Since all threads of the same process share address space and other resources so any changes to the main thread may affect the behavior of the other threads of the process. |
| 13. | A system call is involved in it. | No system call is involved, it is created using APIs. |
| 14. | The process does not share data with each other. | Threads share data with each other. |

6. Difference between i2c and spi?

Ans:

|  |  |
| --- | --- |
| **I2C** | **SPI** |
| I2C can be multi-master and multi-slave, which means there can be more than one master and slave attached to the I2C bus. | SPI can be multi-slave but does not a multi-master serial protocol, which means there can be only one master attached to the SPI bus. |
| I2C is a half-duplex communication protocol. | SPI is a full-duplex commination protocol. |
| I2C has the feature of clock stretching, which means if the slave cannot able to send fast data as fast enough then it suppresses the clock to stop the communication. | Clock stretching is not the feature of SPI. |
| I2C is used only two wire for the communication, one wire is used for the data and the second wire is used for the clock. | SPI needs three or four-wire for communication ((depends on requirement), MOSI, MISO, SCL, and Chip-select pin. |
| I2C is slower than SPI. | In comparison to I2C, SPI is faster. |
| I2C draws more power than SPI. | Draws less power as compared to I2C. |
| I2C is less susceptible to noise than SPI. | SPI is more susceptible to noise than I2C. |
| I2C is cheaper to implement than the SPI communication protocol. | Costly as compared to I2C. |
| I2C work on wire and logic and it has a pull-up resistor. | There is no requirement of a pull-up resistor in the case of the SPI. |
| In I2C communication we get the acknowledgment bit after each byte. | Acknowledgment bit is not supported by the SPI communication protocol. |
| I2C ensures that the data sent is received by the slave device. | SPI does not verify that data is received correctly or not. |
| I2C support multi-master communication. | SPI does not support multi-master communication. |
| I2C is a multi-master communication protocol that’s why it has the feature of arbitration. | SPI is not a multi-master communication protocol, so it does not consist of the properties of arbitration. |
| I2C is the address base bus protocol, you have to send the address of the slave for the communication. | In the case of the SPI, you have to select the slave using the slave select pin for the communication. |
| I2C has some extra overhead due to start and stop bits. | SPI does not have a start and stop bits. |
| I2C supports multiple devices on the same bus without any additional select lines (work on the basis of device address). | SPI requires additional signal (slave select lines) lines to manage multiple devices on the same bus. |
| I2C is better for long-distance. | SPI is better for a short distance. |
| I2C is developed by NXP. | SPI is developed by Motorola. |

7. Difference between spinlock and semaphore?

Ans:

|  |  |  |
| --- | --- | --- |
| **S.No.** | **SPINLOCK** | **SEMAPHORE** |
| 1. | Spinlocks can be used only for  mutual exclusion. | Semaphores can be used either for mutual exclusion or as a counting semaphore. |
| 2. | A spinlock is a low-level synchronization mechanism. | A semaphore is a signaling mechanism. |
| 3. | Spinlocks allows only one process at any given time to access the critical section. | Semaphores allow more than one process at any given time to access the critical section. |
| 4. | Spinlock can be wasteful if they are hold for a long time duration. | In semaphore there is no resource wastage of process time and resources. |
| 5. | Only one thread is allowed at a time to acquire the lock and proceed it with a critical section. | One or several thread is allowed to access the critical section. |
| 6. | Spinlock are very efficient because they are blocked only for a short period of time. | Semaphore are held for a longer period of time. To access its control structure it uses spin lock. |
| 7. | In spinlock, a process is waiting for lock will keep the processor busy by continuously polling the lock. | In semaphore, a process is waiting for a semaphore will go into sleep to be woken up at a any time and the try for the lock again. |
| 8. | Spinlocks are valid for only one process. | Semaphores can be used to synchronize between different processes,. |
| 9. | In spinlock, a process waiting for lock will instantly get access to critical region as the process will poll continuously for the lock. | In semaphore, a process waiting for a lock might not get into critical region as soon as the lock is free because the process would have gone to sleep and when it is wakened up it will enter into critical section. |
| 10. | It is busy wait process. | It is sleep wait process. |
| 11. | Spinlock can have only two values – LOCKED and UNLOCKED | In semaphore, mutex will have value 1 or 0, but if used as counting semaphore it can have different values. |
| 12 | In uniprocessor system spinlock are not very useful because they will keep the processor busy every time while polling for the lock , thus disabling any other process from running. | In uniprocessor system semaphore are convenient because semaphore don’t keep the processor busy while waiting the lock. |
| 13. | In spinlock it is recommended to disable the interrupts while holding a spinlock. | Semaphore can be locked with interrupt enabled. |
| 14. | Thread cannot sleep while waiting for the lock when failed to get the lock, but it continues loop of trying to get locked. | Thread goes sleep for waiting lock when fail to get the lock. |

8.difference between unix and linux?

Ans:

|  |  |
| --- | --- |
| **Unix** | **Linux** |
| Unix is an operating system used in Intel, HP, internet servers, and more. | It is an operating system used for computer hardware & software, game development, and more. |
| Different versions of UNIX have different cost structures. | It has both free distributions and paid versions are available. |
| It is mainly used in mainframes, internet servers, and workstations. | It can be used by anyone including home users, developers, etc. |
| The file support system includes jfs, gpfs, hfs, ufs, vxfs. | The file support system includes Ext2, Ext3, Ext4, Jfs, ReiserFS, Xfs, Btrfs, FAT, NTFS. |
| The common desktop environment serves as a GUI. | It offers two GUIs: KDE and Gnome. |
| It is also highly secured and has around 85-120 viruses listed to date. | Offers higher security. It has around 60-100 viruses listed to date. |
| Examples: Solaris, OS X, All Linux. | Examples: Red Hat, Fedora, Kali Linux, Debian, Android, Ubuntu. |

9.difference between semaphores and mutex?

Ans:

| **Parameters** | **Semaphore** | **Mutex** |
| --- | --- | --- |
| Mechanism | It is a type of signaling mechanism. | It is a locking mechanism. |
| Data Type | Semaphore is an integer variable. | Mutex is just an object. |
| Modification | The wait and signal operations can modify a semaphore. | It is modified only by the process that may request or release a resource. |
| Resource management | If no resource is free, then the process requires a resource that should execute wait operation. It should wait until the count of the semaphore is greater than 0. | If it is locked, the process has to wait. The process should be kept in a queue. This needs to be accessed only when the mutex is unlocked. |
| Thread | You can have multiple program threads. | You can have multiple program threads in mutex but not simultaneously. |
| Ownership | Value can be changed by any process releasing or obtaining the resource. | Object lock is released only by the process, which has obtained the lock on it. |
| Types | Types of Semaphore are counting semaphore and binary semaphore. | Mutex has no subtypes. |
| Operation | Semaphore value is modified using wait () and signal () operation. | Mutex object is locked or unlocked. |
| Resources Occupancy | It is occupied if all resources are being used and the process requesting for resource performs wait () operation and blocks itself until semaphore count becomes >1. | In case if the object is already locked, the process requesting resources waits and is queued by the system before lock is released. |

10. Difference between vmalloc() and kmalloc()?

Ans:

|  |  |
| --- | --- |
| **vmalloc()** | **kmalloc()** |
| The memory allocated is virtually contiguous but not physically contiguous | The memory allocated is physically as well as virtually contiguous |
| Slower than kmalloc() | Faster than vmalloc() |
| More flexible than kmalloc() | Less flexible than vmalloc() |
| Cannot be used to allocate memory in interrupt handler | Can be used to allocate memory in interrupt handler using GFP\_ATOMIC flag |
| Performance is lower than kmalloc() as there is an overhead of mapping of non-contiguous physical memory in contiguous virtual memory | Performance is better than vmalloc() |

11. difference between RTOS and GPOS?

Ans:

|  |  |
| --- | --- |
| **Real-Time Operating System** | **General Purpose Operating System** |
| The RTOS always uses priority-based scheduling. | Task scheduling in a GPOS isn't necessarily based on which application or process is the most important. Threads and processes are often dispatched using a "fairness" policy. |
| The time response of the RTOS is deterministic. | The time response of the general-purpose operating system is not deterministic. |
| A low-priority job in an RTOS would be pre-empted by a high-priority one if required, even executing a kernel call. | A high-priority thread in a GPOS cannot preempt a kernel call. |
| The real-time operating system optimizes memory resources. | The GPOS does not optimize the memory resources. |
| The RTOS is mainly used in the embedded system. | GPOS is mainly used in PC, servers, tablets, and mobile phones. |
| The real-time operating system has a task deadline. | The general-purpose operating system has no task deadline. |
| It doesn't have large memory. | It has a large memory. |
| GPOS code is not often modular in nature when it comes to development. | RTOS kernel code is intended to be scalable, allowing developers to selectively select kernel objects. |
| RTOS is designed and developed for a single-user environment. | GPOS is designed for a multi-user environment. |
| Examples: FreeRTOS, Contiki source code, etc. | Examples: Linux, Windows, IOS, etc. |

11. Difference between static library and dynamic library?

Ans:

| properties | Static library | Shared library |
| --- | --- | --- |
| Linking time | It happens as the last step of the compilation process. After the program is placed in the memory | Shared libraries are added during linking process when executable file and libraries are added to the memory. |
| Means | Performed by linkers | Performed by operating System |
| Size | Static libraries are much bigger in size, because external programs are built in the executable file. | Dynamic libraries are much smaller, because there is only one copy of dynamic library that is kept in memory. |
| External file changes | Executable file will have to be recompiled if any changes were applied to external files. | In shared libraries, no need to recompile the executable. |
| Time | Takes longer to execute, because loading into the memory happens every time while executing. | It is faster because shared library code is already in the memory. |
| Compatibility | Never has compatibility issue, since all code is in one executable module. | Programs are dependent on having a compatible library. Dependent program will not work if library gets removed from the system . |

12. Difference between ARM and X86?

Ans:

|  |  |
| --- | --- |
| **ARM** | **X86** |
| Uses Reduced Instruction Set computing Architecture (RISC). | Uses Complex Instruction Set computing Architecture (CISC). |
| Executes single instruction per cycle. | Executes complex instruction at a time, and it takes more than a cycle. |
| Optimization of performance with Software focused approach. | Hardware approach to optimize performance. |
| Requires less registers, more memory. | It uses more registers and less memory |
| Pipelining of instructions is a unique feature. | Less pipelined. |
| Faster Execution of Instructions reduces time. | Time to execute is more. |
| Complex addressing is managed by software. | Inherently designed to handle complex addresses. |
| Compiler plays a key role in managing operations. | The micro program does the trick. |
| Multiple Instructions are generated from a complex one and executed individually. | Its Architecture is capable of managing complex statement execution at a time. |
| Managing code expansion is difficult. | Code expansion is managed easily. |
| Decoding of instruction is handled easily. | Decoding is handled in a complex way. |
| Uses available memory for calculations. | Needs supplement memory for calculations. |
| Deployed in mobile devices where size, power consumption speed matters. | Deployed in Servers, Desktops, Laptops where high performance and stability matters. |

13.difference between RISC and CISC?

Ans:

| CISC | RISC |
| --- | --- |
| A large number of instructions are present in the architecture. | Very few instructions are present. The number of instructions is generally less than 100. |
| Some instructions with long execution times. These include instructions that copy an entire block from one part of memory to another and others that copy multiple registers to and from memory. | No instruction with a long execution time due to a very simple instruction set. Some early RISC machines did not even have an integer multiply instruction, requiring compilers to implement multiplication as a sequence of additions. |
| Variable-length encodings of the instructions.  **Example:** IA32 instruction size can range from 1 to 15 bytes. | Fixed-length encodings of the instructions are used.  **Example:** In IA32, generally all instructions are encoded as 4 bytes. |
| Multiple formats are supported for specifying operands. A memory operand specifier can have many different combinations of displacement, base, and index register. | Simple addressing formats are supported. Only base and displacement addressing is allowed. |
| CISC supports array. | RISC does not support an array. |
| Arithmetic and logical operations can be applied to both memory and register operands. | Arithmetic and logical operations only use register operands. Memory referencing is only allowed by loading and storing instructions, i.e. reading from memory into a register and writing from a register to memory respectively. |
| Implementation programs are hidden from machine-level programs. The ISA provides a clean abstraction between programs and how they get executed. | Implementation programs exposed to machine-level programs. Few RISC machines do not allow specific instruction sequences. |
| Condition codes are used. | No condition codes are used. |
| The stack is being used for procedure arguments and returns addresses. | Registers are being used for procedure arguments and return addresses. Memory references can be avoided by some procedures. |

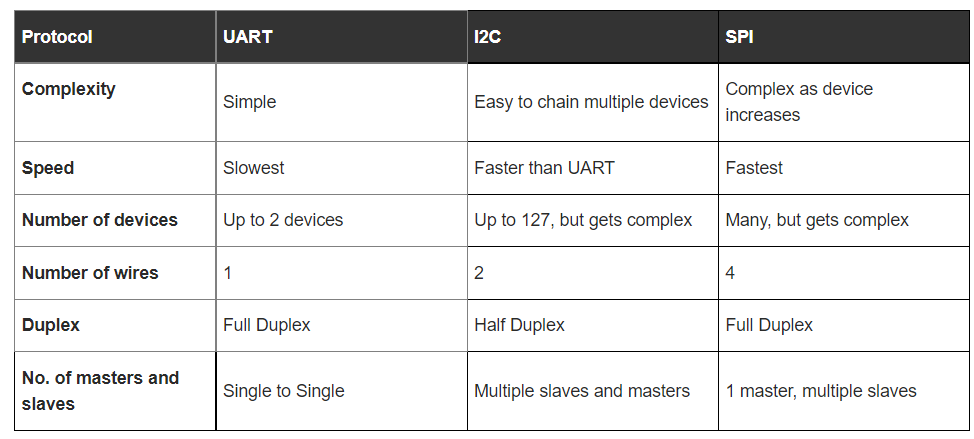
14.difference between user level thread and kernel level thread?

Ans:

| S. No. | Parameters | **User Level Thread** | **Kernel Level Thread** |
| --- | --- | --- | --- |
| **1.** | **Implemented by** | User threads are implemented by users. | Kernel threads are implemented by Operating System (OS). |
| **2.** | **Recognize** | Operating System doesn’t recognize user level threads. | Kernel threads are recognized by Operating System. |
| **3.** | **Implementation** | Implementation of User threads is easy. | Implementation of Kernel thread is complicated. |
| **4.** | **Context switch time** | Context switch time is less. | Context switch time is more. |
| **5.** | **Hardware support** | Context switch requires no hardware support. | Hardware support is needed. |
| **6.** | **Blocking operation** | If one user level thread performs blocking operation then entire process will be blocked. | If one kernel thread perform blocking operation then another thread can continue execution. |
| **7.** | **Multithreading** | Multithread applications cannot take advantage of multiprocessing. | Kernels can be multithreaded. |
| **8.** | **Creation and Management** | User level threads can be created and managed more quickly. | Kernel level threads take more time to create and manage. |
| **9.** | **Operating System** | Any operating system can support user-level threads. | Kernel level threads are operating system-specific. |
| **10.** | **Thread Management** | The thread library contains the code for thread creation, message passing, thread scheduling, data transfer and thread destroying | The application code does not contain thread management code. It is merely an API to the kernel mode. The Windows operating system makes use of this feature. |
| **11.** | **Example** | Example: Java thread, POSIX threads. | Example: Window Solaris. |
| **12.** | **Advantages** | * User Level Threads are simple and quick to create. * Can run on any operating system * They perform better than kernel threads since they don’t need to make system calls to create threads. * In user level threads, switching between threads does not need kernel mode privileges. | * Scheduling of multiple threads that belong to same process on different processors is possible in kernel level threads. * Multithreading can be there for kernel routines. * When a thread at the kernel level is halted, the kernel can schedule another thread for the same process. |
| **13.** | **Disadvantages** | * Multithreaded applications on user-level threads cannot benefit from multiprocessing. * If a single user-level thread performs a blocking operation, the entire process is halted. | * Transferring control within a process from one thread to another necessitates a mode switch to kernel mode. * Kernel level threads takes more time to create and manage than user level threads. |

15.difference between uart, spi, and i2c?

Ans:



**16. Difference between Deadlock and Starvation:**

|  |  |  |
| --- | --- | --- |
| S.NO | Deadlock | Starvation |
| 1. | All processes keep waiting for each other to complete and none get executed | High priority processes keep executing and low priority processes are blocked |
| 2. | Resources are blocked by the processes | Resources are continuously utilized by high priority processes |
| 3. | Necessary conditions Mutual Exclusion, Hold and Wait, No preemption, Circular Wait | Priorities are assigned to the processes |
| 4. | Also known as Circular wait | Also known as lived lock |
| 5. | It can be prevented by avoiding the necessary conditions for deadlock | It can be prevented by Aging |

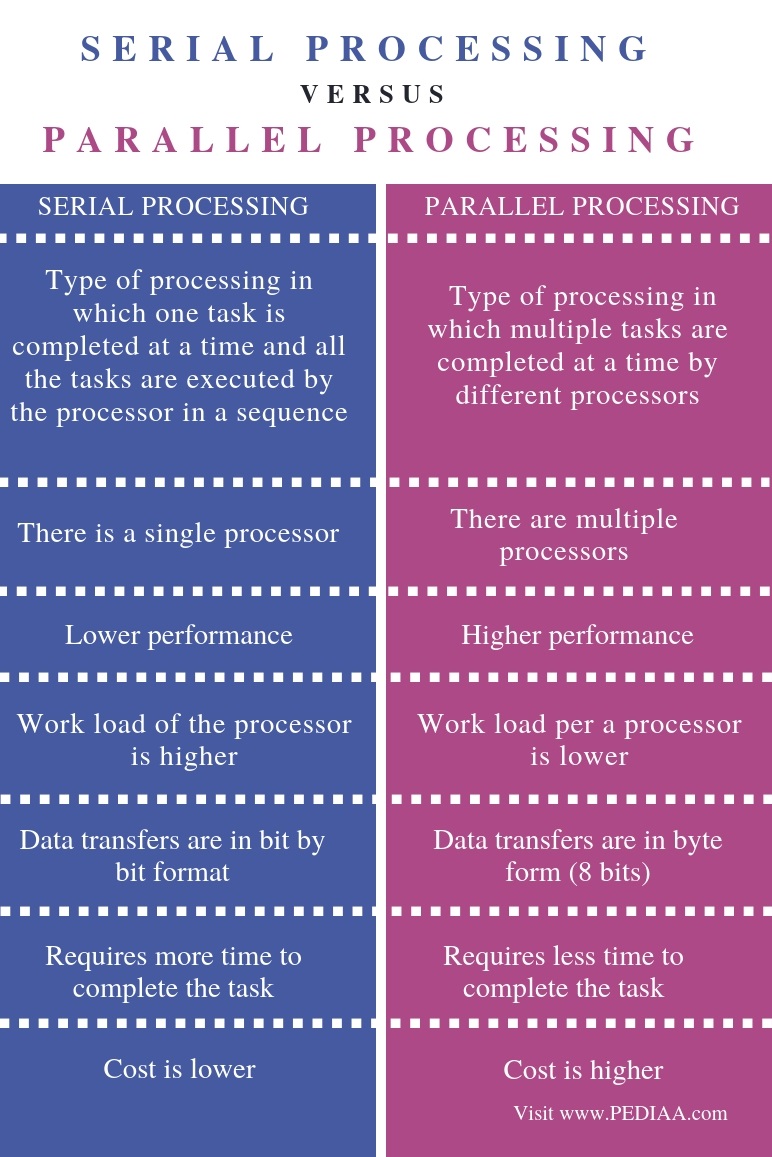
17.Difference between interrupt and polling?

Ans:

| S.NO | Interrupt | Polling |
| --- | --- | --- |
| 1. | In interrupt, the device notices the CPU that it requires its attention. | Whereas, in polling, CPU steadily checks whether the device needs attention. |
| 2. | An interrupt is not a protocol, its a hardware mechanism. | Whereas it isn’t a hardware mechanism, its a protocol. |
| 3. | In interrupt, the device is serviced by interrupt handler. | While in polling, the device is serviced by CPU. |
| 4. | Interrupt can take place at any time. | Whereas CPU steadily ballots the device at regular or proper interval. |
| 5. | In interrupt, interrupt request line is used as indication for indicating that device requires servicing. | While in polling, Command ready bit is used as indication for indicating that device requires servicing. |
| 6. | In interrupts, processor is simply disturbed once any device interrupts it. | On the opposite hand, in polling, processor waste countless processor cycles by repeatedly checking the command-ready little bit of each device. |

18.Difference between serial and parallel processing?

Ans:



**19. Difference between Interrupt and Exception :**

| Interrupt | Exception |
| --- | --- |
| These are Hardware interrupts. | These are Software Interrupts. |
| Occurrences of hardware interrupts usually disable other hardware interrupts. | This is not a true case in terms of Exception. |
| These are asynchronous external requests for service (like keyboard or printer needs service). | These are synchronous internal requests for service based upon abnormal events (think of illegal instructions, illegal address, overflow etc). |
| Being asynchronous, interrupts can occur at any place in the program. | Being synchronous, exceptions occur when there is abnormal event in your program like, divide by zero or illegal memory location. |
| These are normal events and shouldn’t interfere with the normal running of a computer. | These are abnormal events and often result in the termination of a program |

20. Difference between fork() and vfork()?

Ans:

| S.No. | FORK() | VFORK() |
| --- | --- | --- |
| 1. | In fork() system call, child and parent process have separate memory space. | While in vfork() system call, child and parent process share same address space. |
| 2. | The child process and parent process gets executed simultaneously. | Once child process is executed then parent process starts its execution. |
| 3. | The fork() system call uses copy-on-write as an alternative. | While vfork() system call does not use copy-on-write. |
| 4. | Child process does not suspend parent process execution in fork() system call. | Child process suspends parent process execution in vfork() system call. |
| 5. | Page of one process is not affected by page of other process. | Page of one process is affected by page of other process. |
| 6. | fork() system call is more used. | vfork() system call is less used. |
| 7. | There is wastage of address space. | There is no wastage of address space. |
| 8. | If child process alters page in address space, it is invisible to parent process. | If child process alters page in address space, it is visible to parent process. |

21. What is zombie process?

Ans:

Zombie processes in Linux are sometimes **also referred to as defunct or dead processes**. They’re processes that have completed their execution, but their entries are not removed from the process table.

22.Difference between orphan, zombie and daemon process?

Ans:

**Difference between Zombie, Orphan, and daemon Processes:**

| Sl.No | Zombie Process | Orphan Process | Daemon Process |
| --- | --- | --- | --- |
| 1. | A Zombie is a process that has completed its task but still, it shows an entry in a process table. | A child process that remains running even after its parent process is terminated or completed without waiting for the child process execution is called an orphan. | A daemon process is a system-related process always running in the background. |
| 2. | Zombie process states always indicated by Z | The orphan process was created unknowingly due to a system crash. | Daemon process state indicated by ? in the field of *tty*column in the output |
| 3. | The zombie process has controlling terminals | Orphan The zombie process has controlling terminals. | The daemon process does not have controlling terminals. |
| 4. | The zombie process treated as dead they are not used for system processing | An orphan process is a computer process even after their parent terminates init is become a parent and continue the remaining task. | A program that runs for a long time makes them as a daemon process and runs it in the background. |
| 5. | To remove the zombie process execute the kill command. | Terminate the Orphan process use the SIGHUP signal. | Daemon process only when system shutdown. |

22.what is process context?

Ans:

The context of a process includes its address space, stack space, virtual address space, register set image (e.g. Program Counter (PC), Stack Pointer (SP), Instruction Register (IR), Program Status Word (PSW) and other general processor registers), updating profiling or accounting information, making a snapshot image of its associated kernel data structures and updating the current state of the process (waiting, ready, etc).

 This state information is saved in the process's process control block which is then moved to the appropriate scheduling queue.

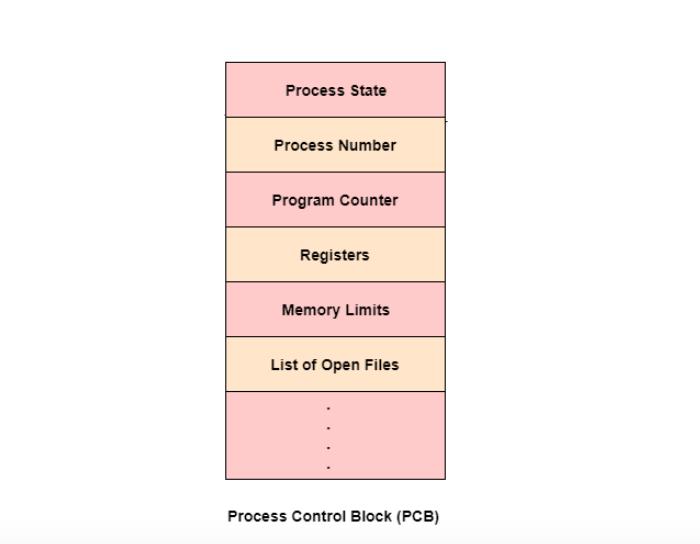
**\*\* Context Switching:**The process of saving the context of one process and loading the context of another process is known as Context Switching. In simple terms, it is like loading and unloading the process from the running state to the ready state.

\*\* **CPU-Bound vs I/O-Bound Processes:** A CPU-bound process requires more CPU time or spends more time in the running state.   
An I/O-bound process requires more I/O time and less CPU time. An I/O-bound process spends more time in the waiting state.

23.What is process control block?

Ans:

Process Control Block is a data structure that contains information of the process related to it. The process control block is also known as a task control block, entry of the process table, etc.



When the process is created by the operating system it creates a data structure to store the information of that process. This is known as Process Control Block (PCB).

Process Control block (PCB) is a data structure that stores information of a process.

Process priority:

Linux kernel implements two separate priority ranges.

When you start any program or process without any defined priority, nice sets a default priority of 10.

First is the nice value. a number from -20 to +19 with default value 0,larger nice values correspond to low priority.

The second range is the Real time priority. The values are configurable but default range from 0 to 99.low real time priority corresponds to low priority.

24.what is ptrace?

Ans: ptrace provides a mechanism by which a parent process may observe and control the execution of another process. It can examine and change its core image and registers and is used primarily **to implement breakpoint debugging and system call tracing**.

System calls:

1.What is system call?

Ans:

* System calls are sometimes called kernel calls.
* syscall is declared in unistd.h.
* A system call is **a procedure that provides the interface between a process and the operating system**. It is the way by which a computer program requests a service from the kernel of the operating system.

2. what is strace?

Ans: A system call is a programmatic way a program requests a service from the kernel, and strace is **a powerful tool that allows you to trace the thin layer between user processes and the Linux kernel**.

strace is **a diagnostic, debugging and instructional userspace utility for Linux**. It is used to monitor and tamper with interactions between processes and the Linux kernel, which include system calls, signal deliveries, and changes of process state.

**What is the File Descriptor?**  
File descriptor is integer that uniquely identifies an open file of the process.

**File Descriptor table**: File descriptor table is the collection of integer array indices that are file descriptors in which elements are pointers to file table entries. One unique file descriptors table is provided in operating system for each process.

**File Table Entry:** File table entries is a structure In-memory surrogate for an open file, which is created when process request to opens file and these entries maintains file position.

## 3. What is System Call in Operating System?

A **system call** is a mechanism that provides the interface between a process and the operating system. It is a programmatic method in which a computer program requests a service from the kernel of the OS.

System call offers the services of the operating system to the user programs via API (Application Programming Interface). System calls are the only entry points for the kernel system.

## 4. Why do you need System Calls in OS?

Following are situations which need system calls in OS:

* Reading and writing from files demand system calls.
* If a file system wants to create or delete files, system calls are required.
* System calls are used for the creation and management of new processes.
* Network connections need system calls for sending and receiving packets.
* Access to hardware devices like scanner, printer, need a system call.

## 5. Types of System calls

Here are the five types of System Calls in OS:

* Process Control 🡪 fork(), exec(), exit().
* File Management 🡪open(), close(),read(),write().
* Device Management🡪ioctl
* Information Maintenance 🡪getpid(), alarm(), sleep().
* Communications 🡪pipe(), shmget(), mmap()

INTER PROCESS COMMUNICATION:

## 1. What is Inter Process Communication?

**Inter process communication (IPC)** is used for exchanging data between multiple threads in one or more processes or programs. The Processes may be running on single or multiple computers connected by a network. The full form of IPC is Inter-process communication.

* These are a few different approaches for Inter- Process Communication:
* **Pipes**
* **Named Pipes**
* **Shared Memory**
* **Message Queue**
* **Semaphore**

**Pipes**

Pipe is widely used for communication between two related processes. This is a half-duplex method, so the first process communicates with the second process. However, in order to achieve a full-duplex, another pipe is needed.

**Message Passing:**

It is a mechanism for a process to communicate and synchronize. Using message passing, the process communicates with each other without resorting to shared variables.

IPC mechanism provides two operations:

* Send (message)- message size fixed or variable
* Received (message)

### Message Queues:

A message queue is a linked list of messages stored within the kernel. It is identified by a message queue identifier. This method offers communication between single or multiple processes with full-duplex capacity.

* msgget():Either returns the message queue identifier for a newly created message queue.
* msgsnd(): Data is placed on to a message queue by calling msgsnd()
* msgrcv(): Data is received from a queue.
* msgctl(): It perfoms various operations on queue,generally destroys queue.

Direct Communication:

In this type of inter-process communication process, should name each other explicitly. In this method, a link is established between one pair of communicating processes, and between each pair, only one link exists.

### Indirect Communication:

Indirect communication establishes like only when processes share a common mailbox each pair of processes sharing several communication links. A link can communicate with many processes. The link may be bi-directional or unidirectional.

### Shared Memory:

Shared memory is a memory shared between two or more processes that are established using shared memory between all the processes. This type of memory requires to protected from each other by synchronizing access across all the processes.

### FIFO:

Communication between two unrelated processes. It is a full-duplex method, which means that the first process can communicate with the second process, and the opposite can also happen.

Important notes in ipc

* **Definition:** Inter-process communication is used for exchanging data between multiple threads in one or more processes or programs.
* Pipe is widely used for communication between two related processes.
* Message passing is a mechanism for a process to communicate and synchronize.
* A message queue is a linked list of messages stored within the kernel
* Direct process is a type of inter-process communication process, should name each other explicitly.
* Indirect communication establishes like only when processes share a common mailbox each pair of processes sharing several communication links.
* Shared memory is a memory shared between two or more processes that are established using shared memory between all the processes.
* Inter Process Communication method helps to speedup modularity.
* A semaphore is a signaling mechanism technique.
* Signaling is a method to communicate between multiple processes by way of signaling.
* Like FIFO follows FIFO method whereas Unlike FIFO use method to pull specific urgent messages before they reach the front.

## 2.What is Like FIFOS and Unlike FIFOS

|  |  |
| --- | --- |
| **Like FIFOS** | **Unlike FIFOS** |
| It follows FIFO method | Method to pull specific urgent messages before they reach the front |
| FIFO exists independently of both sending and receiving processes. | Always ready, so don’t need to open or close. |
| Allows data transfer among unrelated processes. | Not have any synchronization problems between open & close. |

**MEMORY MANAGEMENT:**

1.what is fragmentation?

Ans:  As the process is loaded and unloaded from memory, these areas are fragmented into small pieces of memory that cannot be allocated to incoming processes. It is called **fragmentation**.

The main memory is available, but the space isn’t sufficient in order to load other processes since the allocation of the main memory processes is dynamic.

2.what is segmentation?

Ans:

In Operating Systems, Segmentation is a memory management technique in which the memory is divided into the variable size parts. Each part is known as a segment which can be allocated to a process.

Segmentation is **a memory management technique in which each job is divided into several segments of different sizes, one for each module that contains pieces that perform related functions**. Each segment is actually a different logical address space of the program.

3.What are memory zones?

Ans:

The Linux kernel divides memory into memory zones. On a mainframe, three zones are used: DMA, Normal, and Movable.

* Memory in the DMA zone is below 2 GB, and some I/O operations require that memory buffers are located in this zone.
* Memory in the Normal zone is above 2 GB, and it can be used for all memory allocations that do not require zone DMA.
* Memory in the Movable zone cannot be used for arbitrary kernel allocations, but only for memory buffers that can easily be moved by the kernel, such as user memory allocations and page cache memory. Memory in the Movable zone can more easily be taken offline than memory in other zones.

## 4. What is Process Address Space?

Address space is a space in computer memory. And process Address Space means a space that is allocated in memory for a process. Every process has an address space

Address Space can be of two types

1. Physical Address Space
2. Virtual Address Space

### Physical address space

The physical address is the actual location in the memory that exist physically. System access the data in the main memory, with the help of physical address. Every thing in the computer has a unique physical address. We needs to mapped it to make the address accessible. MMU is responsible for mapping.

### Virtual Address Space

Virtual Address Space is an address space that is created outside the main memory inside the virtual memory, and it is created in the [hard disk](https://t4tutorials.com/how-hard-disk-works/).

When our main memory is less and we want to get more benefit from this less memory, then we create virtual memory.

Logical Address is the address is always generated by CPU while running a program.

## **Process Address Space**

The process address space is the set of logical addresses that a process references in its code. For example, when 32-bit addressing is in use, addresses can range from 0 to 0x7fffffff; that is, 2^31 possible numbers, for a total theoretical size of 2 gigabytes.

The components can be -**variable names, constants, and instruction labels**.

SYNCHRONIZATION:

# **1.Kernel synchronization**

Shared resources in the kernel require protection from concurrent access. This is because other threads of execution might modify data at the same time, resulting in problems like the data being overwritten by one thread, or data being accessed in an inconsistent state

## 2. What is Process Synchronization?

**Process Synchronization** is the task of coordinating the execution of processes in a way that no two processes can have access to the same shared data and resources.

It is specially needed in a multi-process system when multiple processes are running together, and more than one processes try to gain access to the same shared resource or data at the same time.

## 3. What is Critical Section Problem?

A critical section is a segment of code which can be accessed by a signal process at a specific point of time. The section consists of shared data resources that required to be accessed by other processes.

* The entry to the critical section is handled by the wait() function, and it is represented as P().
* The exit from a critical section is controlled by the signal() function, represented as V().

In the critical section, only a single process can be executed. Other processes, waiting to execute their critical section, need to wait until the current process completes its execution.

## Summary:

* Process synchronization is the task of coordinating the execution of processes in a way that no two processes can have access to the same shared data and resources.
* Four elements of critical section are 1) Entry section 2) Critical section 3) Exit section 4) Reminder section
* A critical section is a segment of code which can be accessed by a signal process at a specific point of time.
* Three must rules which must enforce by critical section are : 1) Mutual Exclusion 2) Process solution 3)Bound waiting
* Mutual Exclusion is a special type of binary semaphore which is used for controlling access to the shared resource.
* Process solution is used when no one is in the critical section, and someone wants in.
* In bound waiting solution, after a process makes a request for getting into its critical section, there is a limit for how many other processes can get into their critical section.
* Peterson’s solution is widely used solution to critical section problems.
* Problems of the Critical Section are also resolved by synchronization of hardware
* Synchronization hardware is not a simple method to implement for everyone, so the strict software method known as Mutex Locks was also introduced.
* Semaphore is another algorithm or solution to the critical section problem.

SEMAPHORES:

## 1.What is Semaphore?

**Semaphore** is simply a variable that is non-negative and shared between threads. A semaphore is a signaling mechanism, and a thread that is waiting on a semaphore can be signaled by another thread. It uses two atomic operations, 1) Wait, and 2) Signal for the process synchronization.

A semaphore either allows or disallows access to the resource, which depends on how it is set up.

## Types of Semaphores

The two common kinds of semaphores are

* Counting semaphores
* Binary semaphores.

**Binary Semaphore :**

1. A Binary Semaphore is a semaphore whose integer value range over 0 and 1.
2. It is nothing, but similar to a lock, with two values: 0 and 1. Here 0 means busy, while 1 means free.
3. The idea behind using a binary semaphore is that, it allows only one process at a time to enter the critical section(thus allowing it to access the shared resource).
4. Here, 0 represents that a process or a thread is in the critical section(i.e. it is accessing the shared resource), while the other process or thread should wait for it to complete. On the other hand, 1 means that no process is accessing the shared resource, and the critical section is free.
5. It guarantees mutual exclusion since no two processes can be in the critical section at any point in time.
6. Since it is just a variable, which holds an integer value, it cannot guarantee bounded waiting. It might happen, that a process may never get a chance to enter the critical section, which may lead to its starvation. And we don’t want that.

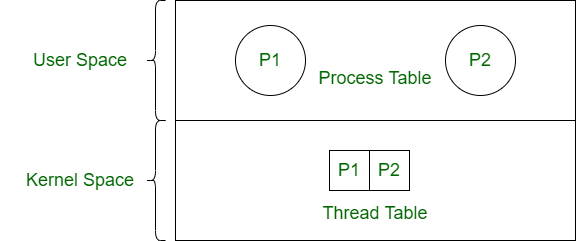
**2. Counting Semaphore :**

1. A counting semaphore is a semaphore that has multiple values of the counter. The value can range over an unrestricted domain.
2. It is a structure, which comprises a variable, known as a semaphore variable that can take more than two values and a list of task or entity, which is nothing but the process or the thread.
3. The value of the semaphore variable is the number of process or thread that is to be allowed inside the critical section.
4. The value of the counting semaphore can range between 0 to N, where N is the number of the number of process which is free to enter and exit the critical section.
5. As mentioned, a counting semaphore can allow multiple processes or threads to access the critical section, hence mutual exclusion is not guaranteed.
6. Since multiple instances of process can access the shared resource at any time, counting semaphore guarantees bounded wait. Using such a semaphore, a process which enters the critical section has to wait for the other process to get inside the critical section, implying that no process will starve.

---------------------------------------------------------------------------------------------------------------------------------

**1. Process:**  
Process is an activity of executing a program. Process is of two types – User process and System process. Process control block controls the operation of the process.

**2. Kernel Thread:**  
Kernel thread is a type of thread in which threads of a process are managed at kernel level. Kernel threads are scheduled by operating system (kernel mode).



**Difference between Process and Kernel Thread:**

| PROCESS | KERNEL THREAD |
| --- | --- |
| Process is a program being executed. | Kernel thread is the thread managed at kernel level. |
| It is high overhead. | It is medium overhead. |
| There is no sharing between processes. | Kernel threads share address space. |
| Process is scheduled by operating system using process table. | Kernel thread is scheduled by operating system using thread table. |
| It is heavy weight activity. | It is light weight as compared to process. |
| It can be suspended. | It can not be suspended. |
| Suspension of a process does not affect other processes. | Suspension of kernel thread leads to all the threads stop running. |
| Its types are – user process and system process. | Its types are – kernel level single thread and kernel level multi thread. |

| S. No. | Parameters | **User Level Thread** | **Kernel Level Thread** |
| --- | --- | --- | --- |
| **1.** | **Implemented by** | User threads are implemented by users. | Kernel threads are implemented by Operating System (OS). |
| **2.** | **Recognize** | Operating System doesn’t recognize user level threads. | Kernel threads are recognized by Operating System. |
| **3.** | **Implementation** | Implementation of User threads is easy. | Implementation of Kernel thread is complicated. |
| **4.** | **Context switch time** | Context switch time is less. | Context switch time is more. |
| **5.** | **Hardware support** | Context switch requires no hardware support. | Hardware support is needed. |
| **6.** | **Blocking operation** | If one user level thread performs blocking operation then entire process will be blocked. | If one kernel thread perform blocking operation then another thread can continue execution. |
| **7.** | **Multithreading** | Multithread applications cannot take advantage of multiprocessing. | Kernels can be multithreaded. |
| **8.** | **Creation and Management** | User level threads can be created and managed more quickly. | Kernel level threads take more time to create and manage. |
| **9.** | **Operating System** | Any operating system can support user-level threads. | Kernel level threads are operating system-specific. |
| **10.** | **Thread Management** | The thread library contains the code for thread creation, message passing, thread scheduling, data transfer and thread destroying | The application code does not contain thread management code. It is merely an API to the kernel mode. The Windows operating system makes use of this feature. |
| **11.** | **Example** | Example: Java thread, POSIX threads. | Example: Window Solaris. |
| **12.** | **Advantages** | * User Level Threads are simple and quick to create. * Can run on any operating system * They perform better than kernel threads since they don’t need to make system calls to create threads. * In user level threads, switching between threads does not need kernel mode privileges. | * Scheduling of multiple threads that belong to same process on different processors is possible in kernel level threads. * Multithreading can be there for kernel routines. * When a thread at the kernel level is halted, the kernel can schedule another thread for the same process. |
| **13.** | **Disadvantages** | * Multithreaded applications on user-level threads cannot benefit from multiprocessing. * If a single user-level thread performs a blocking operation, the entire process is halted. | * Transferring control within a process from one thread to another necessitates a mode switch to kernel mode. * Kernel level threads takes more time to create and manage than user level threads. |

1.what is device tree in linux?

Ans:

 a **devicetree** (also written **device tree**) is a [**data structure**](https://en.wikipedia.org/wiki/Data_structure) describing the hardware components of a particular computer so that the [operating system](https://en.wikipedia.org/wiki/Operating_system)'s [kernel](https://en.wikipedia.org/wiki/Kernel_(operating_system)) can use and manage those components, including the [CPU](https://en.wikipedia.org/wiki/Central_processing_unit) or CPUs, the [memory](https://en.wikipedia.org/wiki/Computer_memory), the [buses](https://en.wikipedia.org/wiki/Bus_(computing)) and the [integrated peripherals](https://en.wikipedia.org/wiki/Integrated_peripheral).

 A **device tree** is a tree data structure that describes the hardware configuration of the system to the **Linux** operating system.

2.what is inode?

Ans: Inodes keep track of all the files on a Linux system. Except for the file name and the actual content of the file, inodes save everything else. It’s like a file-based data structure that holds metadata about all of the files in the system

for every file, along with other metadata, including:

* The size of the file
* Various storage devices and locations where files are stored
* The access permissions associated with the files
* Owner information for the files
* Timestamps of the files (eg., date created, last modified, etc.)

3.what are called jiffies?

Ans:

jiffies holds the number of times the system timer has popped since the system booted. The kernel increments the jiffies variable, HZ times every second. Thus, on a kernel with a HZ value of 100, a jiffy is a 10-millisecond duration, whereas on a kernel with HZ set to 1000, a jiffy is only 1-millisecond long. A **jiffy is a kernel unit of time declared in <linux/jiffies.h**> , To understand jiffies, we need to introduce a new constant, HZ, which is the number of times jiffies is incremented in one second. Each increment is called a tick. In other words, HZ represents the size of a jiffy.

1. What is DMA?

Ans:

Direct memory access (DMA) is a method that allows an input/output (I/O) device to send or receive data directly to or from the main memory, bypassing the CPU to speed up memory operations.

The process is managed by a chip known as a DMA controller (DMAC).

Typical examples are **disk controllers, Ethernet controllers, USB controllers, and video controllers**. Usually the DMA controller built into these devices can only move data between the device itself and main memory