Tutorial-2

1. How can an operating system be viewed as an event-driven application?

   You can create applications and software using three types of programming, **procedural, object-orientated and event-driven.** Operating Systems are **applications which allow you to operate your software and hardware which is inside of your computer.** Operating Systems can be referred to as an **event driven program.** An event driven programming is when the **application runs constantly and depends on events happening in order to execute code and functions.** These events can happen automatically when **timed or manually when the user clicks, types or says something**.

   Example 1: Click on the browser icon to load the browser. This example can be used on most operating systems including Windows, Linux and Android.

   Example 2: Downloading/saving of a file.

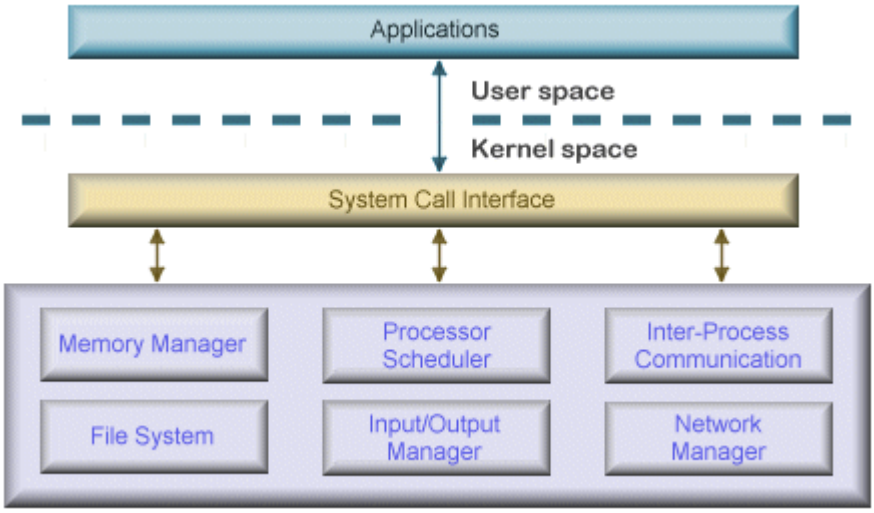
2. Discuss operating system architecture. How monolithic architecture is different from layered architecture?

   Kernel:

   - The core software components of an operating system.
   - The kernel has unrestricted access to all the resources on the system.

   Monolithic OS architecture: Each component of the operating system was contained within the kernel, could communicate directly with any other component, and had unrestricted system access.

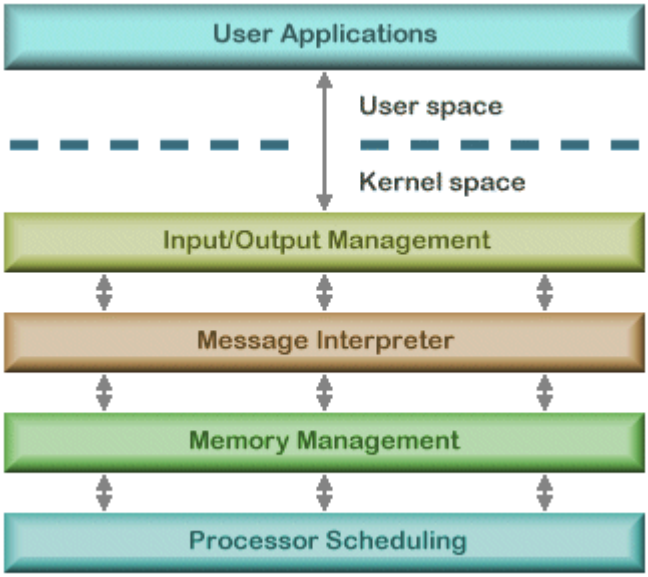
   Advantage and disadvantage: While this made the operating system **very efficient**, it also meant that **errors were more difficult to isolate**, and there was a high risk of damage due to erroneous or malicious code.



As operating systems became larger and more complex, this approach was largely abandoned in favour of a modular approach which grouped components with **similar functionality into layers to help operating system designers to manage the complexity of the system.**

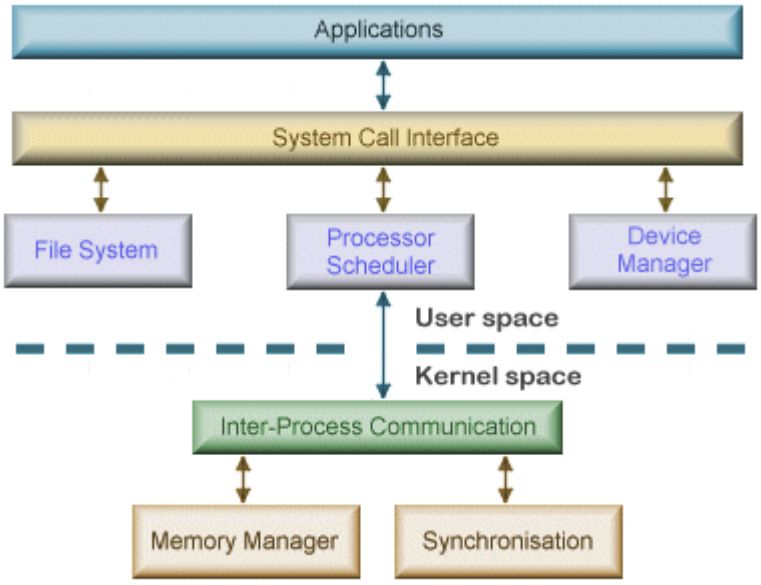
Layered OS architecture:

- Each layer communicates only with the layers immediately above and below it, and lower-level layers **provide services to higher-level ones using an interface that hides their implementation.**
- Modification in any layer will not affect the functioning of adjacent layers.
- Although this modular approach imposes **structure and consistency on the operating system**, **simplifying debugging and modification, a service request from a user process may pass through many layers of system software before it is serviced, and performance compares unfavourably to that of a monolithic kernel.**
- Also, because **all layers still have unrestricted access to the system, the kernel is still susceptible to errant or malicious code. Many of today's operating systems, including Microsoft Windows and Linux, implement some level of layering.**



Microkernel OS architecture:

- It includes **only a very small number of services within the kernel to keep it small and scalable.**
- The services typically include **low-level memory management, inter-process communication and basic process synchronisation to enable processes to cooperate**.
- In microkernel designs, most operating system components, such as **process management and device management, execute outside the kernel with a lower level of system access.**
- Microkernels are **highly modular, making them extensible, portable and scalable**. Operating system components outside the kernel can fail without causing the operating system to fall over. Once again, the **downside is an increased level of inter-module communication which can degrade system performance**.



3. Among all the available OSes, which OS is best for personal use? Which is the fastest OS?

   Server OS and an everyday OS:

   An everyday **OS runs programs including Microsoft word and excel, Adobe Photoshop, and one's favourite computer games.** There are also applications that make browsing the web and checking email easy. It also uses connections such as **LAN and Bluetooth. Every day operating systems are also cheaper than a server OS.**

   A server OS platform typically has **unlimited user connections**, **a greater amount of memory, and can act as a web server, database server, and email server**. It is also optimized for a network instead of for a single user, which means a server OS can handle multiple desktops. A dedicated server that is built to a company's specification and needs is what recommended. This ensures support, management, and the lack of downtime.

   When it comes to an OS for everyday personal use—

   Windows OS or a Mac OS is a great choice, especially if you're doing simple tasks and don't need much customization. At home, you don't need powerful OS especially for simple tasks like writing or browsing the web. For gaming, Windows OS is better than Mac. Games are usually optimized for Windows and their hardware.

   Home server—Windows and a Linux are probably the options you will need to decide between. In simple terms, a home server is a setup of two or more computers that make-up a local area network within a home. One personal computer can serve as a home server if it has enough storage and memory. Home servers are often used to share multimedia content between all devices. If you want to be able to access all your movies, photos, etc. from all your devices, a home server is a good idea.

   Which Is The Fastest OS?

While discussing the fastest OS, there is no argument that **Linux based OS is the lightest and fastest OS in the market right now. It doesn't need a powerful processor unlike Windows to operate at an optimal level.**

Linux based OS like Ubuntu Server, CentOS server, Fedora is great options specially for running business enterprises where substantial computing power is mandatory.

4. Define three styles of switching from user mode to supervisor mode.

   - Interrupts**, when a device sends an interrupt to the CPU**.
   - When a program executes a system call, which is usually implemented by a **trap instruction. (procedural call) and software interrupt.**
   - When a program performs an operation that causes a **hardware exception**, such as **divide by zero, illegal memory access or execution of an illegal opcode**.

5. Explain the difference between *user mode* and *supervisor mode* and explain why modern CPUs include the capability to run in either of these modes.

There are two modes of execution, known as user mode and kernel or supervisor mode. User mode is **restricted in that certain instructions cannot be executed, certain registers cannot be accessed, and I/O devices cannot be accessed.** Kernel mode has none of these restrictions. A system call will set the CPU to kernel mode, as will traps and interrupts. Application programs cannot do this.

Mode bit: **Supervisor** or **User** mode

- *Supervisor mode*
  - Can execute all machine instructions
  - Can reference all memory locations
- *User mode*
  - Can only execute a subset of instructions
  - Can only reference a subset of memory locations

6. What distinguishes *system calls* (such as write) from *library functions* (such as printf)?

   - **Code for system calls are included in the operating system, while code for library functions are included with the running program.**
   - Thus, a system call's code runs in supervisor (unprotected) mode, while a library function's code runs in user mode.
   - A program enters a system call by initiating a software interrupt, while a program enters a library function by calling a subroutine (A set of **Instructions** which are used repeatedly in a program can be referred to as **Subroutine**.) included with the program.

   Programmers typically prefer library functions because they are typically **less platform-dependent (a program written under Linux can more easily be ported to Windows) and provide more convenient functionality** (the OS system calls typically provide bare minimum convenience — like **printf** providing so many formatting options in contrast to no formatting at al with **write**, its closest system-call relative under Linux.

7. What is the primary difference between a kernel-level context switch between processes (address spaces) and a user-level context switch?

    - **User-level threads are threads that the OS is not aware of**. **They exist entirely within a process and are scheduled to run within that process's timeslices.**
    - The OS is aware of kernel-level threads. Kernel threads are scheduled by the OS's scheduling algorithm and require a "lightweight" context switch to switch between (that is, registers, PC, and SP must be changed, but the memory context remains the same among kernel threads in the same process).
    - User-level threads are much faster to switch between, as there is no context switch; further, a problem-domain-dependent algorithm can be used to schedule among them.
    - CPU-bound tasks with interdependent computations, or a task that will switch among threads often, might best be handled by user-level threads. Kernel-level threads are scheduled by the OS, and each thread can be granted its own time slices by the scheduling algorithm.
    - The kernel scheduler can thus make intelligent decisions among threads and avoid scheduling processes which consist of entirely idle threads (or I/O bound threads). A task that has multiple threads that are I/O bound, or that has many threads (and thus will benefit from the additional timeslices that kernel threads will receive) might best be handled by kernel threads.

8. What is the purpose of the command interpreter? Why is it usually separate from the kernel?

    It reads commands from the user or from a file of commands and executes them, usually by turning them into one or more system calls. **It is usually not part of the kernel since the command interpreter is subject to changes.**

    A command interpreter is the part of a computer operating system that understands and executes commands that are entered interactively by a human being or from a program. **In some operating systems, the command interpreter is called the shell.**

    A command interpreter usually separates the kernel as kernel is the central part of an operating system. It manages the operations of the computer and the hardware - most notably memory and CPU time. There are two types of kernels: A micro kernel, which only contains basic functionality; A monolithic kernel, which contains many device drivers.

    yes it is possible for the user to develop a new command interpreter using the system-call interface provided by the operating system.

9. What is the relationship between a guest operating system and a host operating system in a system like VMware? What factors need to be considered in choosing the host operating system?

    A host operating system is the operating system that is in direct communication with the hardware. It has direct hardware access to kernel mode and all the

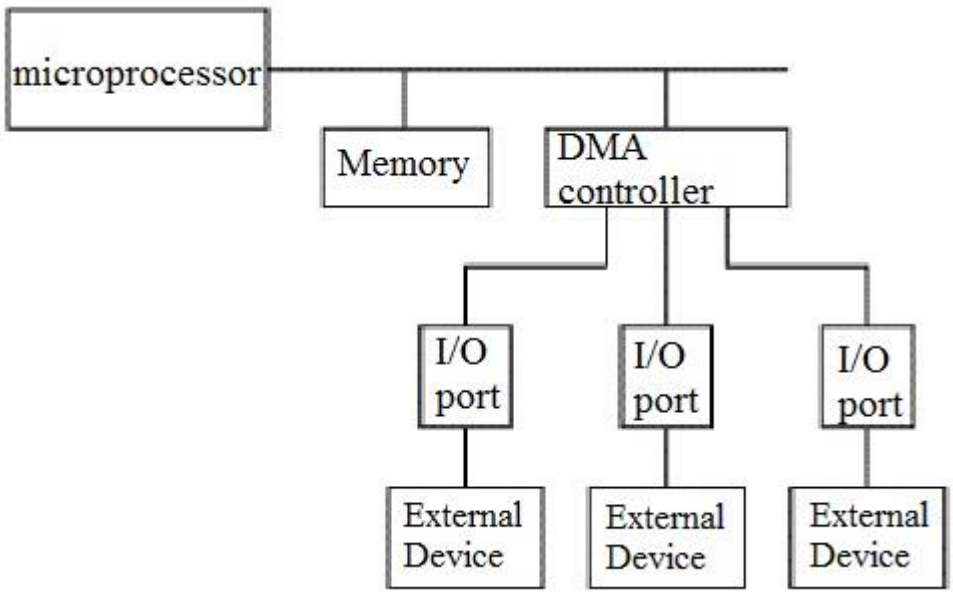
devices on the physical machine. The guest operating system runs on top of a virtualization layer and all the physical devices are virtualized. A host operating system should be as modular and thin as possible to allow the virtualization of the hardware to be as close to the physical hardware as possible, and so that dependencies that exist in the host operating don't restrict operation in the guest operating system.

10. What is Direct Memory Access? How is it important in the operating system?

For the execution of a **computer program**, it requires the synchronous working of more than one component of a computer. For example, **Processors** – providing necessary control information, addresses…etc, **buses** – to transfer information and data to and from memory to I/O devices…etc. The interesting factor of the system would be the way it handles the transfer of information among processor, memory and I/O devices. Usually, **processors control all the process of transferring data, right from initiating the transfer to the storage of data at the destination. This adds load on the processor and most of the time it stays in the ideal state, thus decreasing the efficiency of the system. To speed up the transfer of data between I/O devices and memory, DMA controller acts as station master. DMA controller transfers data with minimal intervention of the processor.**

What is a DMA Controller?

The term DMA stands for direct memory access. The hardware device used for direct memory access is called the DMA controller. DMA controller is a control unit, part of I/O device's interface circuit, which can transfer blocks of data between I/O devices and main memory with minimal intervention from the processor.



The DMA transfers the data in three modes which include the following.

a) **Burst Mode**: In this mode DMA handover the buses to CPU only after completion of whole data transfer. Meanwhile, if the CPU requires the bus it must stay ideal and wait for data transfer.

b) **Cycle Stealing Mode**: In this mode, DMA gives control of buses to CPU after transfer of every byte. It continuously issues a request for bus control, makes the transfer of one byte and returns the bus. By this CPU doesn't have to wait for a long time if it needs a bus for higher priority task.

c) **Transparent Mode:** Here, DMA transfers data only when CPU is executing the instruction which does not require the use of buses.

11. Explain the steps that an operating system goes through when the CPU receives an interrupt.
    - First, the hardware switches the machine **to supervisor mode and causes control to transfer to a routine at a pre-specified address.**
    - The operating system has already loaded that address with a pointer to a function that handles the interrupt.
    - The function starts by saving all registers, possibly setting the allowed interrupts to a level that guarantees correct execution of the operating system, and then executes the code to serve the interrupt.
    - When the operating system is done, it restores the registers again and calls the "reti" to return to the user program that was running when the interrupt occurred.
    - If there is no runnable program, the operating system jumps to the idle loop and waits for an interrupt.
    - Finally, if the interrupt occurred while the operating system was running, then the OS simply restores the registers and jumps to the instruction following the one at which the interrupt occurred.
12. Differentiate Interrupts, Exceptions, and System Calls.

    Exception is a very general thing. It's just another name for a runtime error.
    - your procedure takes two numbers as input and divide the first number by the second. Somebody calls your procedure with 0 as the second number.
    - you have a pointer/memory reference in your code, but the memory it leads to isn't allocated to you/you don't have permissions for the kind of access you are aiming at.
    - your procedure accepts inputs for two numbers from the keyboard but one of the inputs isn't a number.

    Interrupts are, very loosely speaking, exceptions for the CPU (hardware). The CPU, under normal conditions, fetches and executes code sequentially, until of course jumps/branches are encountered. In any case, the flow of execution is well ordered. When this order is violated, we say that the CPU has been interrupted. These can consist

of anything including, but not limited to:

- reading from/writing to virtual addresses for which no TLB entry is present (that is, they are not been mapped to the executing process' address space). Fact: such a condition is termed a *page fault.*
- A DMA device/sensor device is *ready* and wants to grab the attention of the CPU.
- The CPU is executing integer instructions and suddenly encounters a flaoting-point operation. Such an exception is called a *floating-point exception (FPE).*
- An user process fires a specialized instruction designed to interrupt the execution.

Whenever this happens, the CPU backs up all the hardware registers that store run-time information that are needed to completely specify the state of the executing process immediately before the interrupt and starts executing instructions from a predefined memory address. The code that resides at this address is called the *interrupt service routine [ISR]*. It is part of the kernel of your operating system and is analogous to the idea of the exception handler. The ISR, based on the state of the CPU registers is able to figure out exactly what caused the interrupt, run appropriate code to handle it and, if needed, request the CPU to revert the execution to the exact same point where it was interrupted. This is usually done through a special *return from interrupt* [rfi] instruction that most (or all) hardware vendors provide.

A ***system call*** is really a kind of interrupt. In fact it is a deliberate interrupt triggered by a user-space process just to tell the kernel to do something on behalf of it (such as access things like terminals or disk files which it is not allowed to normally).

The system call is modelled on ordinary procedures and one may be able to pass parameters and get return values. The protocols to be followed by the user space program to get a request through to the kernel is called the *system call interface* for the particular operating system. System libraries like *libc* exports wrappers upon standard system calls that make them look the same as ordinary procedures.