**Handout: Operating Systems**

**HND in Computing & Systems Development**

**Introduction to Operating Systems**

An operating system is system software that makes efficient use of the resources like the hardware of a computer system while creating an interface between the user and the system for communication.

Operating Systems can be broken down into their usage order as;

1. Single user – Single tasking (DOS)
2. Single user – Multi tasking (Windows 7)
3. Multi user – Multi tasking (GNU/Linux based OSes)
4. Real-time

There are multiple types of interfaces each operating system employs to provide services to the user in the most efficient manner;

1. Graphical User Interface- Has Windows, Icons, Menus and Pointers
2. Command Line Interfaces- Where a command line is provided to type dedicated commands into it to achieve tasks.

GUI operating systems are much more common nowadays due to the sheer difficulty in using a CLI to perform simple tasks. For example, creating a folder in Windows using the command prompt would require “cd”-ing into the correct path, then entering the “mkdir <nameOfTheFolder>” to create a folder.

Then to check if it was made, you have to enter “dir”. This process is a hassle for average users. All of this hassle is replaced by a simple right-click -> new -> folder. Just 3 clicks and the job is done!

**Single User – Single Tasking**

This type of operating system only has to deal with one person at a time, running one user application at a time. What is known as the “single-user mode” is also one such example. What this mode allows is to allow booting of a multiuser system into one individual superuser. This particular mode is used. It is mainly used for maintenance of multi-user environments.

The safe mode in the windows systems that reverts to the last known best configuration is a day-to-day example. Most mobile OSes from back in the time, are also single-user OSes. They only allowed one user application to be run at a particular time.



Figure 1.2 express.co.uk, 2017/02/17, Aaron Brown

Figure 1.1 Geekdashboard.com, 2017/09/21, Amar Ilindra

**Single-user Multi-tasking**

This is the type of operating system most people use on their desktop and laptop computers today. Microsoft's Windows and Apple's MacOS platforms are both examples of operating systems that will let a single user have several programs in operation at the same time.

For example, it's entirely possible for a Windows user to be writing a note in a word processor while downloading a file from the Internet while printing the text of an e-mail message.

The difference compared to the Single-Use, Single Application operating system is that it must now handle many different applications all running at the same time. The memory available is also very different, for example it is quite normal to have Gigabytes of RAM available on a personal computer which is what allows so many applications to run.

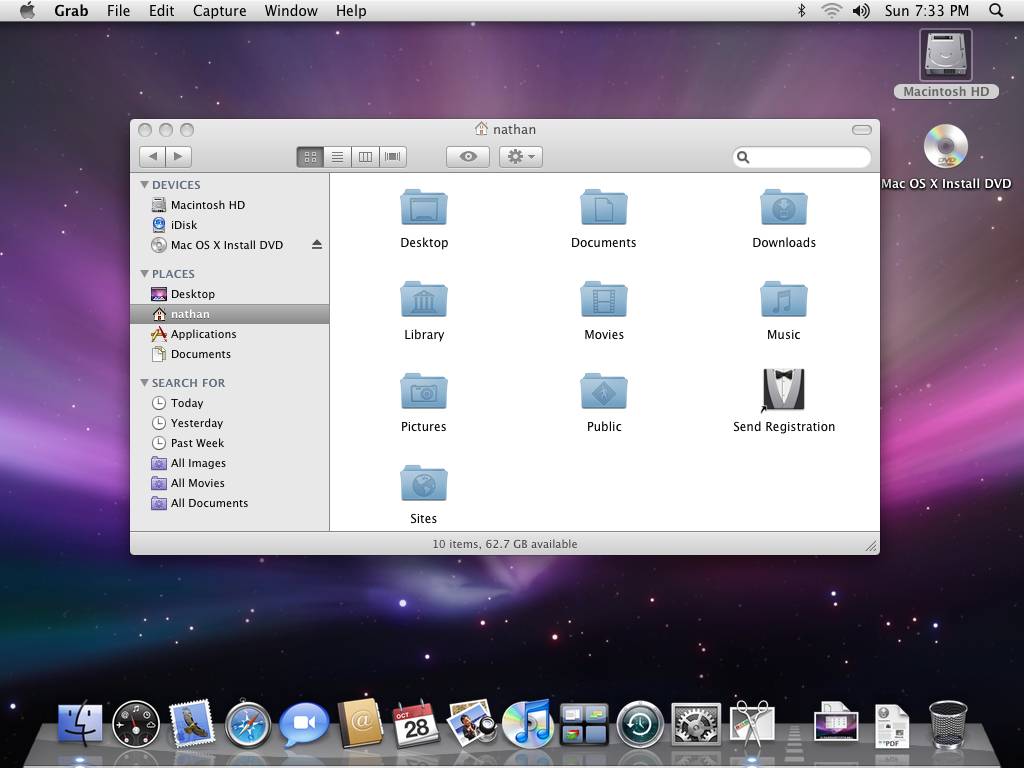
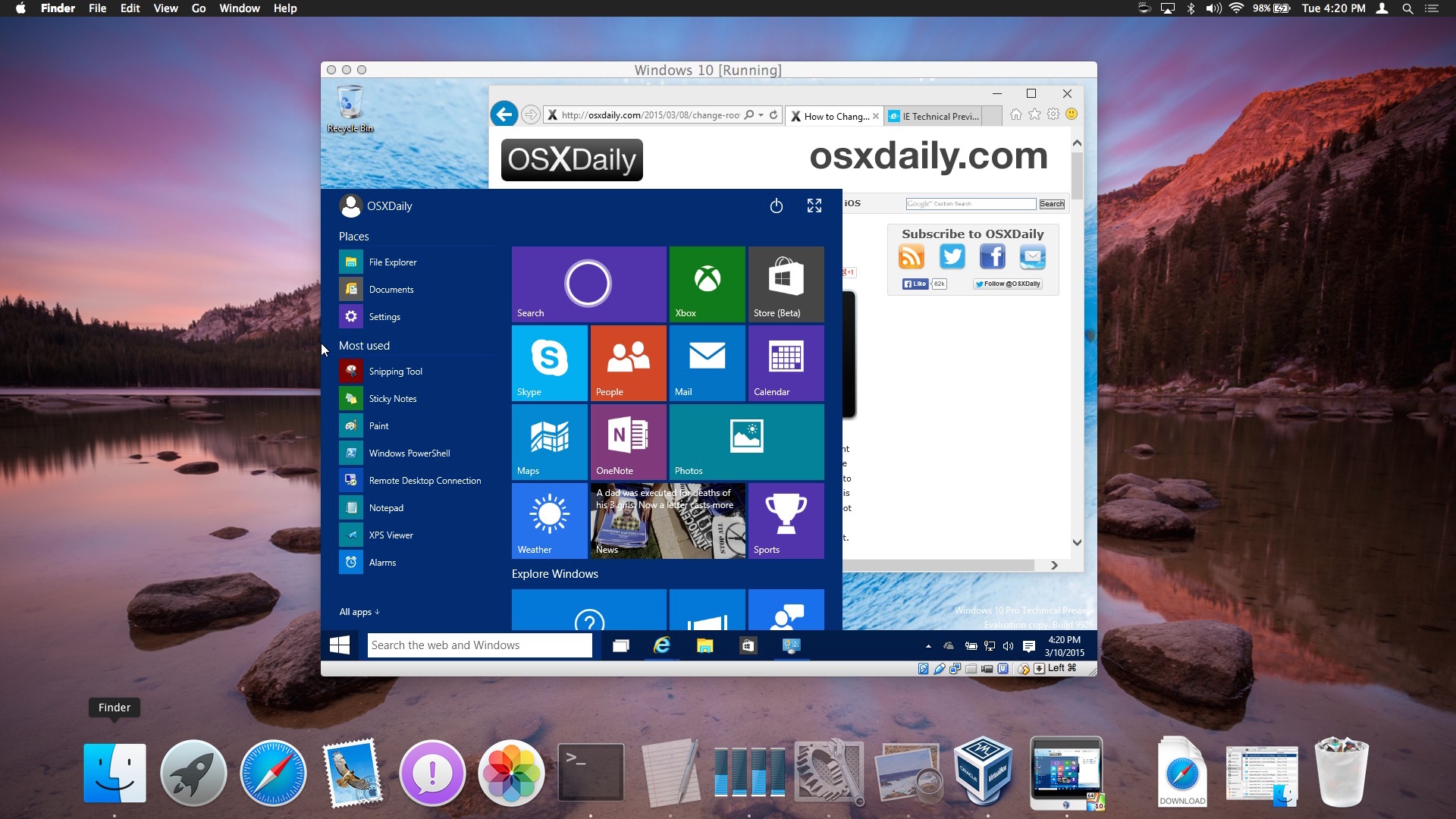


Figure 1.4 osxdaily.com, 2015/03/25, Paul Horowitz

Figure 1.3 toastytech.com, 2014/05/13

**Multi-user, Multi-tasking**

A multi-user operating system allows many different users to take advantage of the computer's resources simultaneously.

The operating system must make sure that the requirements of the various users are balanced, and that each of the programs they are using has sufficient and separate resources so that a problem with one user doesn't affect the entire community of users.

UNIX, VMS and mainframe operating systems, such as MVS, are examples of multi-user operating systems.

The *system administrator* is the only "user" for systems like Windows 2000. The network support and all of the remote user logins the network enables are, in the overall plan of the operating system, a program being run by the administrative user.

Most modern day GNU/Linux operation systems are multi user, multi-tasking systems with more than one user but single superuser.



Figure 1.5 wouodl.wordpress.com, 2015/01/23 Ms. Parasathy

Effective resource handling and performance must be primitive concerns when implementing a multi-user operating system. The kernel with related to this particular subset of operating systems will be discussed further soon.

**Real time**

Real time processing is usually found in systems which use computer control. This processing method is used when it is essential that the input request is dealt with quickly enough so as to be able to control an output properly.

For example, the computer inside the engine’s CU in a car has to manage the engine at every moment based on what the driver wants to do.

Real time processing has to be programmed very carefully to ensure that no input events are missed.

Not only input events should be carefully monitored in such a way they are not missed, but in fact if the event fails either the system must acknowledge this or revive the input event.

If the engine is running low on fuel, there will be a light signal or the meter at the edge saying that the driver must refill. If there were no such ways of letting the driver know, there’ll be more than just a mess.

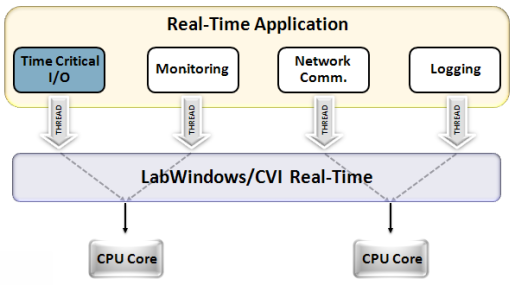


Figure 1.6, itexplanation.com, 2016/07/04

**The key features of an Operating System**

1. **Device Management**

An Operating Systems must be capable of interacting with the subsystems of the computer system and not only request services from hardware, but check for availability, ensure its proper functionality, direct it with rich instructions and ensure an output is received.

If not, to trigger an error to notify the user of the issue that has taken place with the required error information and error code.

The Operating System, in order to ensure Device Management, must be capable of interacting with the Input/Output subsystem, which in turn must be capable of processing the request through the CPU and store necessary date in the Main Memory during the fetch, decode & execute cycle.

A rich OS will not only maintain its devices to ensure functionality but also ensure qualitative performance.

Most OSes are capable of this, given the fact that device management falls within the scope of that particular OS. Ex: Windows 8/8.1 is ideal for Device Management, but Disk Operating Systems might not. Ex: Device Manager does this in Windows.

1. **Memory Management**

This particular function is primarily based on maintaining the Main Memory of a computer system. Particularly while a process is going on, the Random Access Memory must be handled in such a way the RAM doesn’t overload, crumple under demand or stop responding.

Memory management is not only the simple case of allocating the right amount of space for a process to run but also ensuring that when the process is no longer running to free it, and to suspend the process of its service and let any other awaiting process to know that the particular memory slot is free now.

Memory addressing in particular is one such function of Memory management. Memory addresses are given to units of information or data so that, the currently allocated memory is not overwritten until its purpose is done with. Ex: In windows, Task manager does this.

1. **Process Management**

An Operating system must interact with the CPU in order to perform the fetch, decode and execute cycle. When an input is made through the interface an OS provides to the user, the OS must let the CPU know that an input has been made, then the CPU processes the request and provides an output. The OS should then be able to take this output while ensuring its validity, accuracy and status then provide it to the user in a comprehensible manner. Ex: Task manager does this in windows

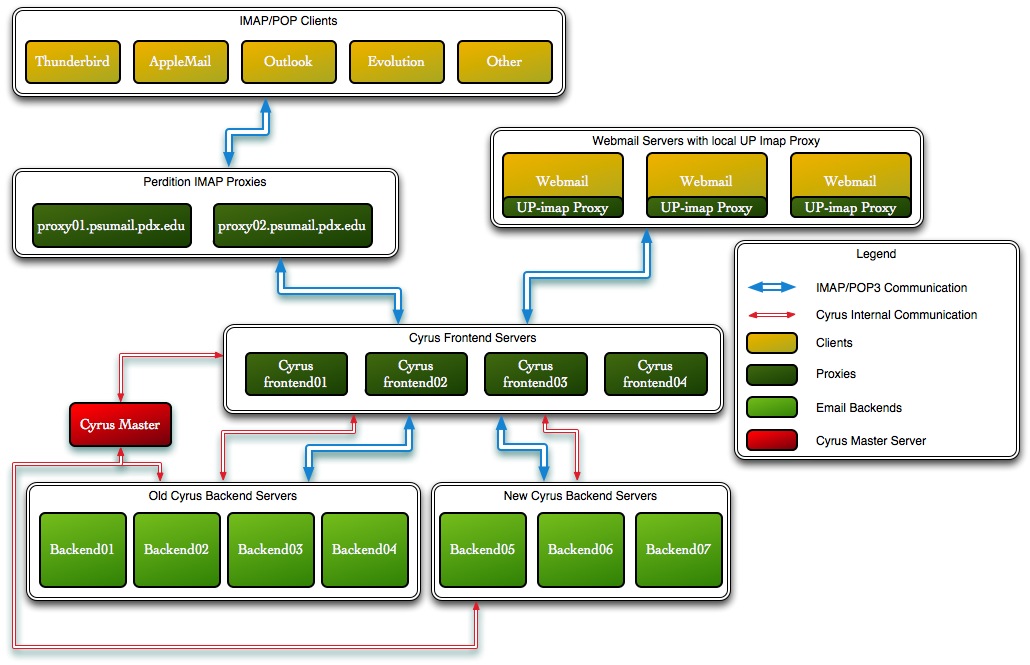
1. **File Management**

Retaining correct records/paths, memory in the hard disk and ensuring a file’s validity is also a function of an Operating system. Application software or system software, any software has files that must be perfectly aligned and ready to be used. They must be store in the hard disk in such a way that whenever the user requests it, the OS must be capable of requesting the file and showing the computer system where the file is and then load it into memory.

1. **Security Management**

Not only should all the resources of a system be handled by the OS, but also the health status of the system must be maintained in such a way that malicious activity has no say in the system. This does not only go so far as to say that the OS must ensure there are no viruses, malware etc. in the system but also keep the information of any individual process private.

The operating system must provide privacy on user’s demand and be capable of attuning to an environment when the system is infected. Either through 3rd party software or implicit software, an OS must be capable of not only combatting but completely erasing such dangers from a system.



A rough representation of a virtual private server operating system, VPS OSes are very common in the job industry as the modern era is moving towards cloud storage faster than ever. Proxies, front ends & backends. These all exist in the OS we all love, Windows except in a much less noticeable manner.

Figure 2.0, computing.pdx.edu, 2017/09/22

**Where does performance come into an Operating System?**

If the goal of an OS is to creating an interface and maintaining resources, in what aspect does it ensure proper performance? It is by ensuring that all the key features mentioned above are completed to the highest notch possible.

Completion isn’t enough; it must also be capable of utilizing error management systems to notify the user when there are errors. Errors must be provided in proper detail so that troubleshooting will be easy.

Troubleshooting methods must also be employed by implicit error management systems, which will ensure the security of the system by keeping the user from consulting the 3rd party environments about an error.

When troubleshooting is impossible, an OS must be capable of letting the user know any information that can be given to the user in order to refer 3rd party environments for consultancy.

Errors building up on a system are eventually the result of a virus. So to ensure the highest performance of a system, an operating system must perform all its key features but also ensure the security to an extra extent.

**What exactly is a kernel?**

A user that is familiar with only the Windows series of operating system may not be used to hearing the term “kernel”.

The kernel is a superior subset in any system that directly interacts with hardware and provides services and receives requests from the application level.

Every Operating system architecture based on kernels have 2 modes. One being kernel, the other being the user mode. In the kernel mode, the kernel by itself and all raw tangible resources lie, while in the user mode all application level software reside.

An Operating System includes a kernel, in order to achieve tasks using proper resource management through the help of the kernel.

On nearly ANY system, the kernel is booted at the very beginning as without it’s presence there will be no connection between applications and hardware. Windows NT is the kernel used in Windows!

**Various Operating Systems**

Initially there was a kernel that was only capable of interacting with hardware but provides application-level functions or gives access to the kernel, called the UNIX kernel. This masterpiece was developed by Dennis Macalister Ritchie & his comrade Ken Thompson just after developing the C++ programming language. UNIX wasn’t advanced, it was small. Systems back then could run UNIX with no difficulties.

Unix is not used in the modern day for tasks but is regularly used by computer scientists to produce new distributions and to produce operating systems to cater towards their personal needs.

Unix can be run in an ESM manager & agent system with 2GB Physical Memory, 25GB HDD and 2.8GHz processer.

Then later on, the linux kernel was written by Linus Torvalds using the UNIX kernel as a base. This new kernel together with GNU bindings produced the current modern day, operating systems that are fully capable of allowing the application level to interact with the kernel. Ex: Ubuntu, Fedora, Debian etc.

The modern-day favorites: Ubuntu- 16.04 LTS system can run on very low specs; 2GHz dual core processor speed, 2 GiB RAM, 25GB HDD space and a VGA that’s capable of 1024x768 .

Fedora- Runs on an even low set of specs; 1GHz processor speed, 1GB RAM and 10GB HDD space.

CentOS- Depending on the number of entries, the requirements change but the maximum requirement is 8GB HDD space and 1GB RAM.

KaliLinux- A penetration testing suite with low requirements and high speed; 20GB HDD space and 2GB RAM

As you can see, the GNU/Linux derivatives are optimum for budget systems running low on specs but can provide a good performance. GNU/Linux systems are purpose-oriented.

They were not produced by previously mentioned developers for day to day tasks. Linux systems were produced to assist developers, to ease their workload and to bring the development era to the next generation.

So this being an issue, the Macintosh brand took the UNIX operating system and produced a day to day use operating system known as the MacOS. MacOS is a specific operating system for a very peculiar architecture that the Macintosh Company came up with.

The MacOS employs micro-kernel architecture with a hint of layered concept.

MacOS has varying requirements depending on what system it is to be installed in. MacOS was introduced as an ideal for the developer that had day-to-day tasks.

But still, the larger audience was catered to. Then came Microsoft with its Distributed Operating System concept. This operating system was named Windows. From Windows 3.1 to today, all the operating systems have been cross-compatible with a lot of systems around the world. Serving as the go to for the average user.

Each version of the operating system has varying requirements but the latest version Windows 10, have the following requirements;

1 GHz processor speed, 1GB RAM, 16GB HDD space for the 32 bit architecture and 20 GB HDD space for the 64 bit architecture. DirectX 9 should be capable of running in machines that wish to run Windows 10. The display should at least be 800x 600 resolutions.

**Categories of an Operating System in terms of architecture.**

1. **Layered Architecture**

The layered architecture was introduced as the improvement for its predecessor; Monolithic OS.

Each layer can interact with the one just above it and the one just below it.

Lowermost layer which directly deals with the bare hardware is mainly meant to perform the functionality of the Input/output subsystem interaction and the uppermost layer which is the application level acts as an interface between user and operating system.

This is a highly advantageous architecture because all the functionalities are on different layers and hence each layer can be tested and debugged separately.

**Pros**- Easier debugging and troubleshooting because all layers are separate, even if one layer stops responding the rest of the operating system can function and adding new layers or removing older layers is very easy.

**Cons**-Non-adjacent layers can’t communicate and dividing functionalities to separate layers is a difficult task and sometimes require a lot of layers.

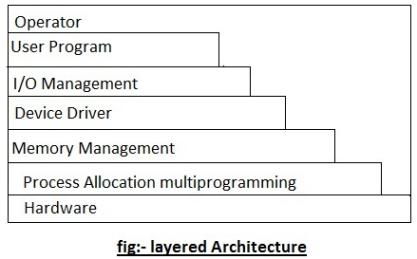


Figure 3.0, sciencehq.com, 2013/06/17

1. **Monolithic Architecture**

It is the oldest architecture of the operating system. It is solely comprised of one layer, where date & code run in the same layer.

The user mode and kernel mode aren’t strictly separated, so the security is very weak. The kernel can access all the resources present in the system.

In the monolithic systems, each component of the operating system is contained within the kernel.

All the basic services of OS like process management, file management, memory management, exception handling, process communication etc. are all present inside the kernel only.

**Pros**-Very easy to implement and faster due to direct access to all services.

**Cons**-Making changes to the system are very difficult & security is very weak.

1. **Micro-kernel Architecture**

The basic concept in this architecture is to keep the kernel as small as possible. As the kernel should be the part that should only deal with the most important services, it can be as small as possible.

In the microkernel architecture, only the most important services are put inside the kernel and rest of the OS services are present in the outer level of the kernel.

The user can easily interact with these services outside the kernel.

The micro-kernel is only responsible for: Communication between processes, Memory management & CPU scheduling.

**Pros**: Expansion of the system is very easy; the security is notably higher than the previous implementation. There are no reasonable disadvantages that serve a significant impact on the system with this architecture.

1. **Distributed Architecture**

A distributed operating system is software that spreads through multiple functioning nodes. These nodes handle tasks through multiple CPUS. Each individual node contains a part of the whole operating system with it. Each node can contain one of the 2 distinct parts.

Either it can be the kernel mode or the user mode supplies the application level interaction. These nodes function as subsets, and subsets as sets to achieve the task of an operating system

The kernel mode of the distributed architecture allows collaboration and maintaining multiple resources and processing functionality thereby producing an efficient and a stable system. This particular process is known as the single system image.

Although we think that the Distributed architecture is one single unit, it is actually sets and sets of nodes.

**Pros**- Makes it easy for any user to work, decent performance and has wide-range of support.

**Cons**- Low security and a user may request services directly from the kernel itself.

Figure 3.1, itexplanation.com, 2016/07/20

