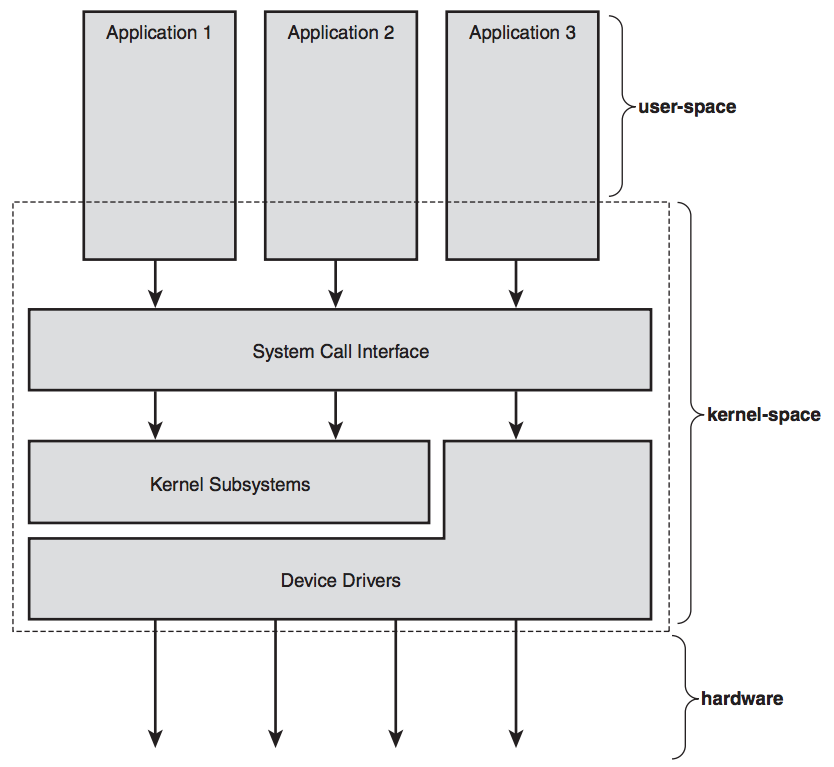
**What is a Kernel?**

A kernel is the core of an OS, it is a computer program which is responsible for the interaction of Application Softwares with the System Hardware. The basic work of the Kernel is to do interrupt handling, process management, process scheduling(to handle multiple processes), memory management and system services such as networking and interprocess communication.

The kernel resides in a high system state other than the normal application software this system state provides a protected memory space and full access to the system hardware, this system state and memory space is collectively known as the *Kernel-Space*.

Now you might be thinking how the user applications communicate with the Kernel well the answer is **System Calls**basically a system call is a programmatic way of requesting services from the kernel such as the creation of new processes, accessing hardware etc.

Relationship between applications, the kernel, and hardware

Kernel also handles *interrupts*basically an interrupt is generated when hardware wants to communicate with the System literally it interrupts the processor which in turn interrupts the kernel for example when you type something on to your keyboard the keyboard controller sends the interrupts to let the system know that there is new data which needs to execute before any process.

**Functions of the Kernel**

Primary functions of the kernel are listed below:

**Memory Management:**

It is one of the most important functions of the kernel. The kernel allocates new space whenever a new process gets created and removes old processes from the main memory to create room for new processes.

**Scheduling Processes:**

*The dispatcher* is a small program inside kernel which is responsible for switching control between processes. The *scheduler*is a type of dispatcher which decides which process needs to be loaded next based on some logic.

**Interrupt Management:**

As you see above I/O devices generate interrupts, the kernel allocates and manages these interrupts to execute the I/O operations.

**Types of Kernels**

Kernels can be classified into two main categories:

* Monolithic kernels
* Microkernels

**Monolithic Kernels**

In traditional monolithic architecture, all the services such as process management, memory management, interrupt handling etc. were packed into a single module which leads to two biggest problems

* Huge Kernel Space
* Complex debugging procedure because you have to load the whole kernel to modify or debug it.

In modern monolithic structure, the kernel is divided into different modules which can be loaded and unloaded separately. This extension in the kernel made modification and debugging of the kernel very much easy as it becomes easy to load and unload the module separately.

[*Linux*](http://fossnow.com/what-is-linux-introduction-to-linux-os.html)*kernel follows the monolithic modular approach.*

**Microkernel**

Microkernel targets the problem of the ever growing size of the kernel code which is a major issue in the monolithic architecture. In microkernels services like device driver management, protocol stack, file system etc are made to run in user-space instead of the Kernel-Space. This reduces the size of the kernel and also increases the security and stability of OS.

In microkernel application programs which are part of the user-space are made to run as servers which can communicate with other programs through inter-process-communication.

**Dynamic-Link Library (DLL)**

1. Its full meaning is Dynamic Link Library
2. It can not execute without the help of EXE file. That means dll files are not independent
3. It has not main function.
4. A dll is a library that an exe (or another dll) may call
5. In one application many dll files may exists
6. A dll can be shared with others applications. That means dll is reusable
7. dll is an In-Process Component
8. A dll can never run in its own memory address space

**Executable File (EXE)**

1. It’s full meaning Executable File.
2. It can execute with the help of OS. That means exe files are independent
3. It has main function
4. An exe is a program
5. In one application only one exe files exists
6. An exe cannot be shared with others applications. That means exe is not reusable
7. EXE is an Out-Process Component
8. An exe always runs in its own memory address space