<https://linux-kernel-labs.github.io/refs/heads/master/lectures/intro.html>

**Kernel responsibilities:**

1. Process management:

->In charge of creating and destroying process.

-> communication between input and output world

-> communication between different processes like pipe, ipc, signals

-> in addition, scheduler, which controls how processes share CPU

2. Memory management:

-> Creates virtual address space on top of limited resources (malloc, free)

3.File system:

-> FAT, ext3

4. Device control

-> device control operations are performed by device code that is specific to the device being addressed. This code called device driver.

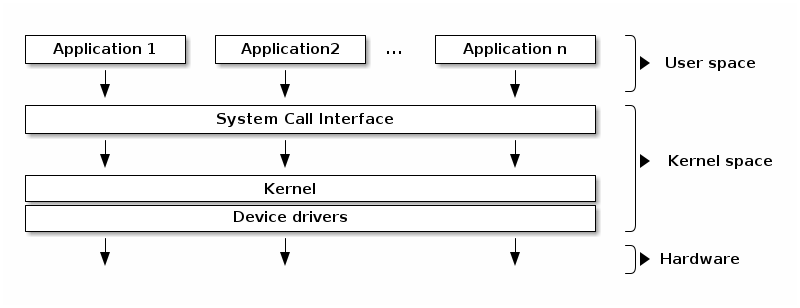
User space -application runs in user space

Kernel - has full control of CPU

local interrupts of cpu can be enabled or disabled in kernel mode, if this operation executes in user mode, kernel will handle it.

, the kernel space is the memory area that is reserved to the kernel while user space is the memory area reserved to a particular user process.

The kernel space is accessed protected so that user applications cannot access it directly, while user space can be directly accessed from code running in kernel mode.

System kernel is responsible for access and share the Hardware in a secure way and fine way with multiple application.

Kernel code -> Core kernel -> common one. It has subsystems like process management, memory management, networking, file system.

Device drivers code -> specific to device.

Kernel API -> system call interfaces

**Monolithic kernel:**

Monolithic kernel has no access protection between various subsystems. Public functions can be called between subsystems.

A diagram of a software system

Description automatically generated

**MicorKernel:**

Large part of the **kernel protected from each other. M**ost of the kernel subsystem runs in user space,A diagram of a computer hardware system

Description automatically generated

In microkernel, IPC, Scheduler, memory management will run from kernel space. Basic memory management system to protect between different services and applications.

One of the advantages is, if any service is crashed no need to start all the server, only restart the service which is crashed.

Offers memory protection between different services. Now simple call between two services in monolithic, goes through IPC, scheduler which increases the cost.

**Address Space:**

The physical address space is referred to the way how RAM and other device memory visible on the memory bus. In intel 32-bit architecture, RAM is mapped to lower part of the physical address space, and graphics is mapped to higher part of the physical address space.

Virtual address space is referred to the way how CPU sees the memory is the virtual memory module is activated or paging enabled or protected mode enabled.

Kernel is responsible to setup the mapping between virtual address space and physical address space.

Virtual address space

Kernel address space Process(user) address space (starts from 0 and continuous memory)

Typical implementation of user and kernel space is one where virtual address is shared between user process and kernel.

A diagram of a computer

Description automatically generated

Interrupt context runs in kernel mode as a result of interrupt trigger, this code shouldn’t have any blocking call or calls to user space.

Process context runs in process code (user space) or in kernel by executing system call.

**Multitasking:**

Multitasking is the ability of operating system to simultaneously executes multiple programs. It does by switching between running processes.

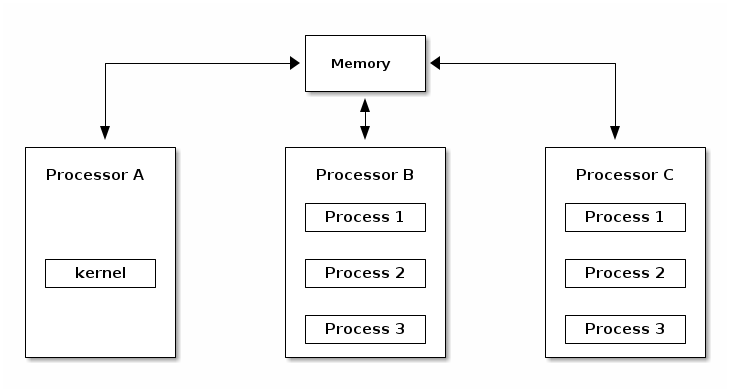
Cooperative multitasking -> program will run, and voluntarily it gives up the CPU to OS, which will then schedule another program.

Preemptive Multitasking -> Kernel will enforce strict timings limits for each process. So that all the process gets fair chance to run.

Preemptive kernel -> If a process can be preempted while running in a kernel mode. But if the kernel is non preemptive, even then it can support preemptive multitasking.

**Asymmetric Multiprocessing (ASMP)**

* It’s a way of supporting multiple processors (cores) by a kernel where processor is dedicated to the kernel, and other processors runs user space programs.
* The disadvantage of this approach is that kernel throughput (system calls, interrupt handling) doesn’t scale with number of processors. Hence typical processes frequently use system calls.



**Symmetric Multiprocessing:**

* Kernel can run on any of the existing processes as a user process. It creates a race condition in the kernel, if two processes run kernel function that access same memory location.
* In order to support SMP the kernel must implement synchronization primitives (spin lock) to guarantee that only one processor is executing a critical section.

A diagram of a computer process

Description automatically generated

CPU scalability:

It refers how well performance scales with the number of cores.

* Use of lock free algorithms when possible.
* Use fine grained locking for high contention areas.
* Attention to algorithm complexity.

**Linux Kernel Architecture:**

