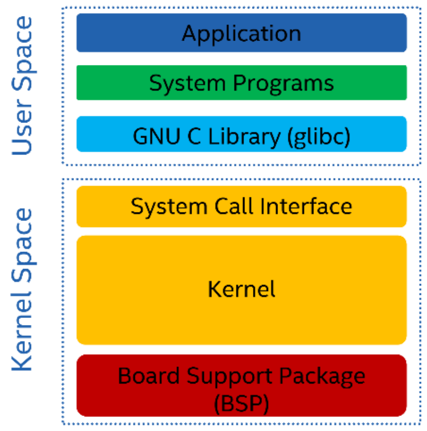
Embedded Linux kernel

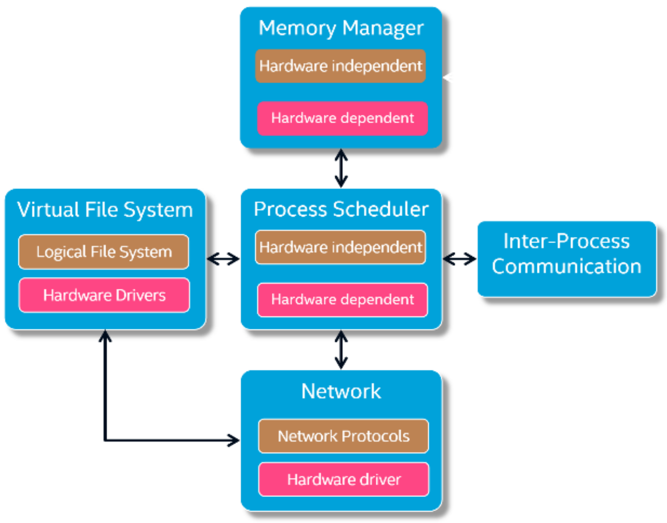
Date	@November 25, 2024

Linux architecture

At the upper level of the architecture you have application and system programs, in addition to GNU C library (glibc). Then we have system calls, the kernel, in which we have the device drivers that work on real hardware, and then board support package which contains functions to access the low level drive.



The kernel can be divided in five subsystems and most of them are composed of hardware-independent code or hardware-dependent code.



Process scheduler

The Linux scheduler is for multiple CPU cores, it is not real time and is not priority based. It sends signals to user processes, manages the hardware timer and provides support for loadable Kernel modules. Since Linux is not real time, all the things done in FreeROTS for deferred interrupts here is not really important unless you use soft-real time implementations. In Linux you have two types of interrupt requests: fast interrupt and the normal interrupt, which differ for the priority.

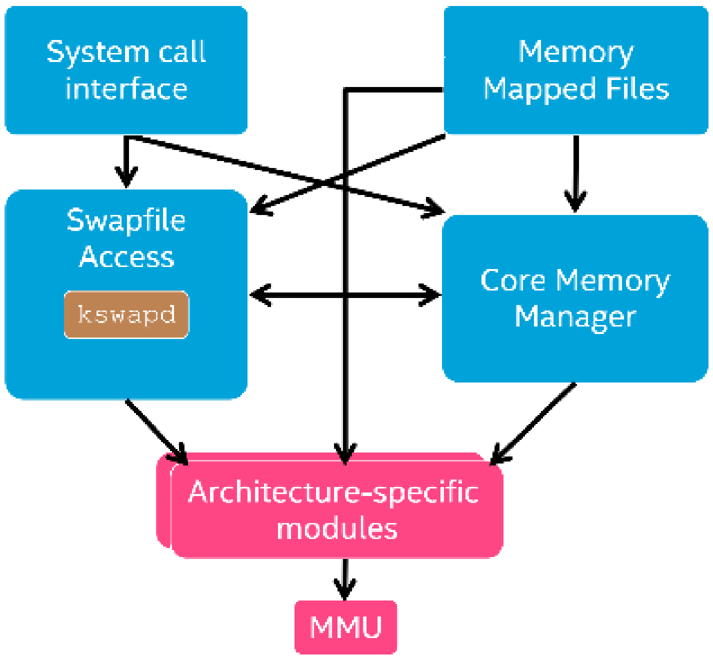
Memory management

Linux has a very complex memory management that allows programs to use more memory than the physical memory that you have. So you can implement virtual memory but you need special hardware. Since we have memory protection, if you want to share memory you need to ask the OS. So if your hardware doesn't have an MMU you won't be able to run Linux.

Since we have complex memory management in Linux you have lots of functions that in order to work needs a complex memory manager and you have functions to allocate memory in user space, and others to allocate memory in kernel space. So there are two memory managers, one for the user space and one for the kernel space.

The main components of a memory manager architecture are:

- system call interface, which provides memory manager services to the user space
- memory mapped files, which implements memory file mapping algorithms
- core memory manager, which is responsible for implementing memory allocation algorithms
- swapfile access, which controls the paging file access
- architecture-specific modules, which handle hardware-specific operations related to memory management



Virtual file system

The virtual file system is responsible for handling:

- multiple hardware devices, since it provides access to hardware devices
- multiple logical file systems, since it supports many different logical organizations of information on storage media
- multiple executable formats, since it supports different executable file formats
- homogeneity, since it presents a common interface to all of the logical file systems and all hardware devices
- performance, since it provides high-speed access too files
- safety, since it enforces policies to not lose or corrupt data
- security, since it enforces policies to grant access to files only to allowed users and it restricts user total file size with quotas

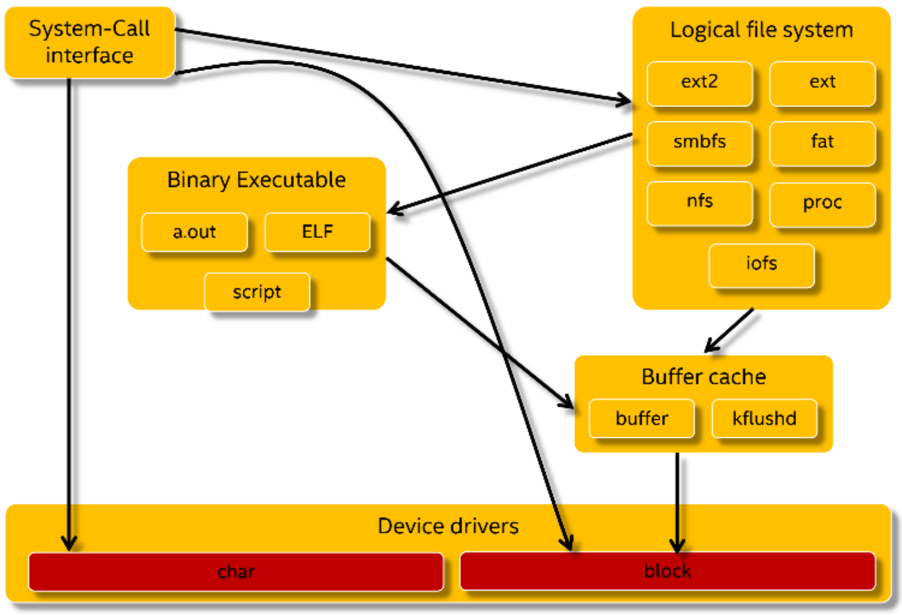
The basic idea is to take whatever is on your computer (I/O devices, real files, etc) and to create a unique interface to handle everything that is a system call that works on a file. So if you're working with your keyboard, in Linux the keyboard is a file; as well as your printer, your USB port, your screen, and so on.

I-NODE

Linux works with file systems based on I-nodes and the data structure which define an I-node in Linux is

```
struct inode {
  struct hlist_node        i_hash;
  struct list_head         i_list;
  struct list_head         i_sb_list
  struct list_head         i_dentry;
  unsigned long            i_ino;
  atomic_t                 i_count;
  umode_t                  i_mode;
  unsigned int             i_nlink;
  uid_t                    i_uid;
  gid_t                    i_gid;
  dev_t                    i_rdev;
  loff_t                   i_size;
  struct timespec          i_atime;
  struct timespec          i_mtime;
  struct timespec          i_ctime;
```

Every file will be associated to an I-node with information and based on that information the OS understands what is connected to that file and how to handle it.



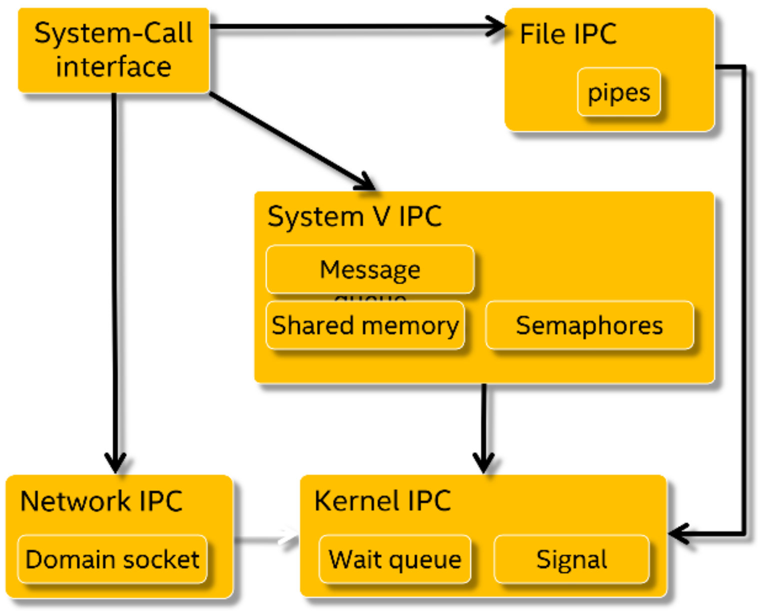
Inter-process communcation

In Linux we have the same inter-process communication elements, such as sempahore, mutex, etc, as in FreeRTOS, but they are much more powerful. Inter-process communication provides mechanisms to processes for allowing:

- resource sharing

- synchronization
- data exchange

Also most of the inter-process communication mechanisms are accessible through system call interfaces.



Network

Linux provides support for network connectivity. It implements network protocols through hardware-independent code and implements network card drivers through hardware-specific code.