**Table Of Contents**

[**List of Abbreviation** ii](#_Toc108026385)

[**List of Figures and Tables** iii](#_Toc108026386)

[**1. Introduction** 1](#_Toc108026387)

[**2.** **History** 2](#_Toc108026388)

[**3.** **Kernel Modules** 3](#_Toc108026389)

[**4.** **Process Management And Scheduling Approaches** 4](#_Toc108026390)

[4.1 The Process 4](#_Toc108026391)

[4.2 Process Identity 4](#_Toc108026392)

[4.3 Process Control Block 5](#_Toc108026393)

[4.4 Threads 6](#_Toc108026394)

[4.5 Process Scheduling 6](#_Toc108026395)

[**5.** **Inter-process Communication** 7](#_Toc108026396)

[**6.** **Memory Management Approaches** 8](#_Toc108026397)

[**7.** **File System Management Approaches** 10](#_Toc108026398)

[**8.** **Device Management Approaches** 12](#_Toc108026399)

[8.1 Polling and Interrupts 12](#_Toc108026400)

[8.2 Direct Memory Access (DMA) 13](#_Toc108026401)

[**9. Summary** 14](#_Toc108026402)

[**10. References** 15](#_Toc108026403)

# **List of Abbreviation**

FIFO: first in first out 6

GPL: GNU Public License 3

OS: Operating System 1, 2

PDA: Personal Digital Assistant 1

RISC: Reduced Instruction Set Computer 2

RR: Round Robin 6

# **List of Figures and Tables**

[Figure 1: Interaction of Linux kernel modules with their environment 3](#_Toc107956437)

[Figure 2: Layout of a process in memory 4](#_Toc107956438)

[Figure 3: Process Control Block 5](#_Toc107956439)

[Figure 4 : Shared Memory and Message Passing 7](#_Toc107956440)

[Figure 5: a.Process A's Virtual Address Space, b. Physical Memory, c. Process B's Virtual Address Space 9](#_Toc107956441)

[Figure 6: Linux file system directories 10](#_Toc107956442)

[Figure 7: System calls for file management 11](#_Toc107956443)

[Figure 8: System calls for Directory Management 11](#_Toc107956444)

**Linux Case Study**

# **1. Introduction**

Linux is an open source operating system which is a variant of UNIX. [Linux](https://www.redhat.com/en/topics/linux) was designed to be similar to UNIX, but in the course of time it has evolved to run on a wide variety of hardware from phones to [supercomputers](https://www.redhat.com/en/blog/red-hat-enterprise-linux-builds-foundation-worlds-fastest-supercomputers) [1] . Every Linux-based OS involves the [Linux kernel](https://www.redhat.com/en/topics/linux/what-is-the-linux-kernel)—which manages hardware resources—and a set of software packages that make up the rest of the operating system.

Linux is one of those operating systems which is rapidly evolving over the course of time. Because Linux is an open source OS, combinations of software can vary between Linux distributions.

On the server side, Linux is well-known as a stable and reliable platform, providing database and trading services for companies like Amazon, the well-known online bookshop, US Post Office and many others. It is also worth to note that modern Linux not only runs on workstations, mid- and high-end servers, but as well as on gadgets like PDA's, mobiles, a shipload of embedded applications and even on experimental wristwatches [2].

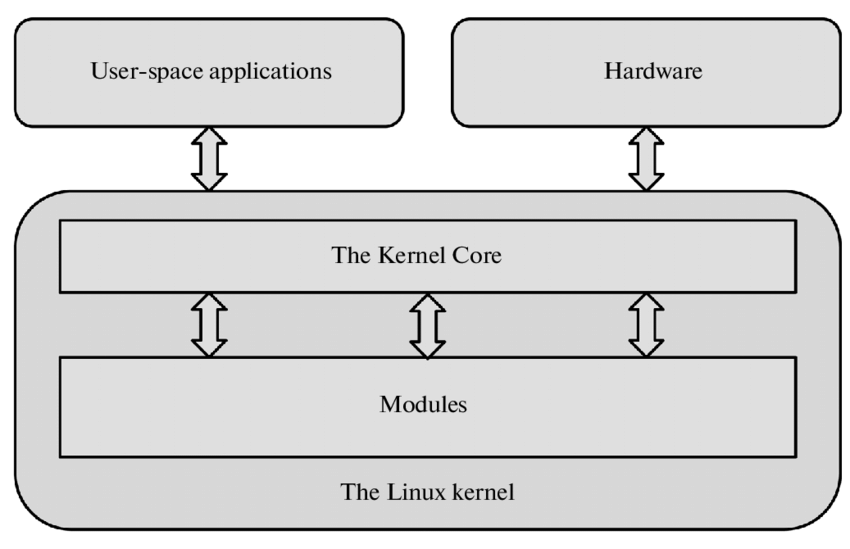
# 

# **History**

* **1991:** Finnish computer scientist Linus Torvalds begins to program the Linux kernel as an independent project.
* **1992:** Linux kernel Version 0.01 is released onto the internet.
* **1993:** The first Debian Linux is distributed; Mark Ewing and Bob Young establish Red Hat Incorporated.
* **1994:** Linux 1.0 is released.
* **1996:** The mascot of Linux, Tux, is created by Larry Ewing; Linux 2.0 adds support to more processors**.**
* **1999:** The GNOME desktop environment is released for Linux.
* **2004:** Ubuntu 4.10 is released.
* **2011:** Google announces the Chromebook and cloud-based Chrome OS.
* **2013:** The Ubuntu Touch OS for mobile phones is revealed.
* **2019:** Major additions include: WireGuard, USB 4, 2038 fix, Spectre fixes, RISC-V support, exFAT, AMDGPU and so much more!

[3]

# **Kernel Modules**

* Sections of kernel code that can be compiled, loaded and unloaded independent of the rest of the kernel.
* A kernel module may typically implement a device driver, a file system, or a networking protocol.
* The module interface allows third parties to write and distribute on their own terms, device drivers or file systems that could not be distributed under the GPL.
* Kernel modules allow a Linux System to be set up with a standard, minimal kernel, without any extra device drivers built in.
* Components to support the Linux Module:
* **Module-management system**
* **Module loader and unloader**
* **Driver registration system**
* **Conflict resolution mechanism** [4]  Figure : Interaction of Linux kernel modules with their environment

# **Process Management And Scheduling Approaches**

A process is a program in execution. A process is the unit of work in most systems. Systems consist of a collection of processes: operating-system processes execute system code, and user processes execute user code. All these processes may execute concurrently [5].

Processes are created in Linux in an especially simple manner. The fork system call creates an exact copy of the original process. The forking process is called the parent process. The new process is called the child process. The parent and child each have their own, private memory images [6].

# **The Process**

The memory layout of a process is typically divided into multiple sections:

* + Text section— the executable code
  + Data section—global variables
  + Heap section—memory that is dynamically allocated during program run time
  + Stack section— temporary data storage when invoking functions

[5]

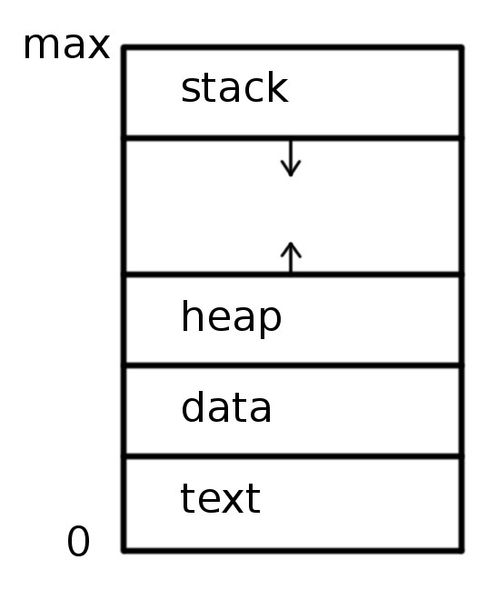


Figure : Layout of a process in memory

# **Process Identity**

Every process in the system has a process identifier. The process identifier is not an index into the task vector, it is simply a number. Each process also has User and group identifiers, these are used to control this processes access to the files and devices in the system [6].

# **Process Control Block**

Each process is represented in the operating system by a process control block. The process control stores many data items that are needed for efficient process management.

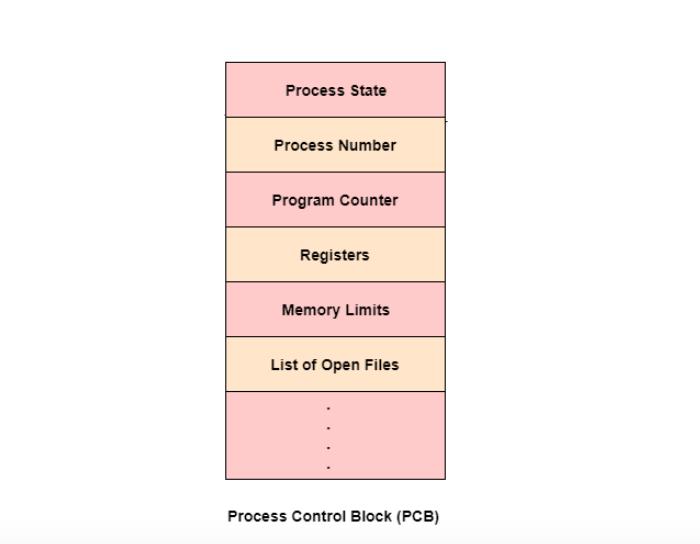


Figure : Process Control Block

The following are the data items −

**Process State:** This specifies the process state i.e. new, ready, running, waiting or terminated.

**Process Number:** This shows the number of the particular process.

**Program Counter:** This contains the address of the next instruction that needs to be executed in the process.

**Registers:** This specifies the registers that are used by the process. They may include accumulators, index registers, stack pointers, general purpose registers etc.

**List of Open Files:** These are the different files that are associated with the process

**CPU Scheduling Information:** The process priority, pointers to scheduling queues etc. is the CPU scheduling information that is contained in the PCB. This may also include any other scheduling parameters.

**Memory Management Information:** The memory management information includes the page tables or the segment tables depending on the memory system used. It also contains the value of the base registers, limit registers etc.

**I/O Status Information:** This information includes the list of I/O devices used by the process, the list of files etc.

**Accounting information:** The time limits, account numbers, amount of CPU used, process numbers etc. are all a part of the PCB accounting information [7].

# **Threads**

**Thread** is the set of instructions executed within a process that can range from a single thread to multiple. The process is the one that allocates the memory and resources that are later used by the thread. As it operates in parallel, the application’s performance will also be improved as well. Having the same address space of threads and processes means that communication between threads will cost less. A failure of one thread will most definitely affect other threads.

# **4.5 Process Scheduling**

A process scheduler deals with process scheduling in Linux. Scheduling in Linux deals with the removal of the current process from the CPU and selecting another process for execution.

Linux has two types of processes:

* Real time process
* Normal process

Each process type has a different scheduling algorithm.

**Real-Time Scheduling**

There are two scheduling policies when it comes to real-time scheduling, SCHED\_RR and SCHED\_FIFO.

* **SCHED\_FIFO:** Here FIFO means first in first out. This policy schedules the process according to arrival time of the process.
* **SCHED\_RR:** Here RR means Round Robin. This policy schedules the process by giving them a fixed amount of time for execution. This fixed time is known as time quantum.

**Normal (SCHED\_NORMAL or SCHED\_OTHER)**

SCHED\_NORMAL / SCHED\_OTHER is the default or standard scheduling policy used in the LINUX operating system. A time-sharing mechanism is used in the normal policy. A time-sharing mechanism means assigning some specific amount of time to a process for its execution. Normal policy deals with all the threads of processes that do not need any real-time mechanism.

[8]

**CFS — Completely Fair Scheduler**

The Completely Fair Scheduler (CFS) is a process scheduler which was merged into the Linux kernel and is the default scheduler. It handles CPU resource allocation for executing processes, and aims to maximize overall CPU utilization while also maximizing interactive performance [6].

# **Inter-process Communication**

 Inter-process communication (IPC) is a mechanism that allows processes to communicate with each other and synchronize their actions.

There are two fundamental models of inter-process communication:

* **Shared memory:** In the shared-memory model, a region of memory that is shared by the cooperating processes is established. Processes can then exchange information by reading and writing data to the shared region. Shared memory can be faster than message passing. In shared-memory systems, system calls are required only to establish shared memory regions. Once shared memory is established, all accesses are treated as routine memory accesses, and no assistance from the kernel is required
* **Message passing:** In the message-passing model, communication takes place by means of messages exchanged between the cooperating processes. Message passing is useful for exchanging smaller amounts of data, because no conflicts need be avoided. Message passing is also easier to implement in a distributed system than shared memory.

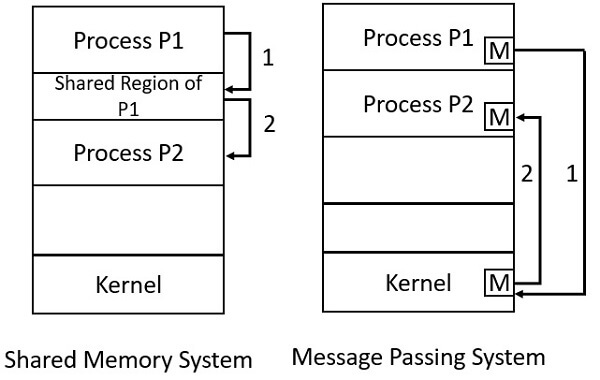
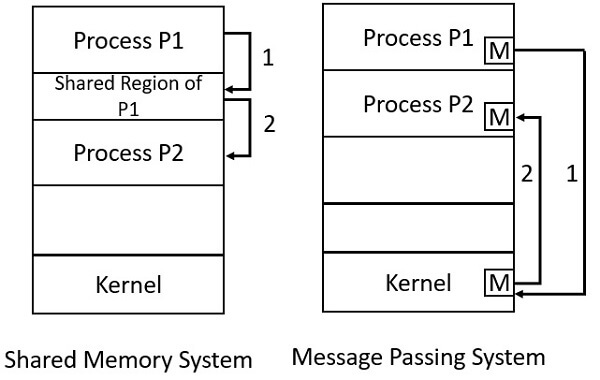


Figure : Shared Memory and Message Passing

# **Memory Management Approaches**

* Memory management is one of the most complex activity done by Linux kernel. It has various concepts/issues associated with it.
* The memory management subsystem is one of the most important parts of the operating system. Since the early days of computing, there has been a need for more memory than exists physically in a system. Strategies have been developed to overcome this limitation and the most successful of these is virtual memory. Virtual memory makes the system appear to have more memory than it actually has by sharing it between competing processes as they need it.
* The memory management subsystem provides:
* **Large Address Spaces:** The operating system makes the system appear as if it has a larger amount of memory than it actually has. The virtual memory can be many times larger than the physical memory in the system.
* **Protection:** Each process in the system has its own virtual address space. These virtual address spaces are completely separate from each other and so a process running one application cannot affect another. Also, the hardware virtual memory mechanisms allow areas of memory to be protected against writing. This protects code and data from being overwritten by rogue applications.
* **Memory Mapping:** Memory mapping is used to map image and data files into a processes address space. In memory mapping, the contents of a file are linked directly into the virtual address space of a process.
* **Fair Physical Memory Allocation:** The memory management subsystem allows each running process in the system a fair share of the physical memory of the system,
* **Shared Virtual Memory:** Although virtual memory allows processes to have separate (virtual) address spaces, there are times when you need processes to share memory. For example there could be several processes in the system running the bash command shell. Rather than have several copies of bash, one in each processes virtual address space, it is better to have only one copy in physical memory and all of the processes running bash share it. Dynamic libraries are another common example of executing code shared between several processes. Shared memory can also be used as an Inter Process Communication (IPC) mechanism, with two or more processes exchanging information via memory common to all of them. Linux supports the Unix TM System V shared memory IPC.

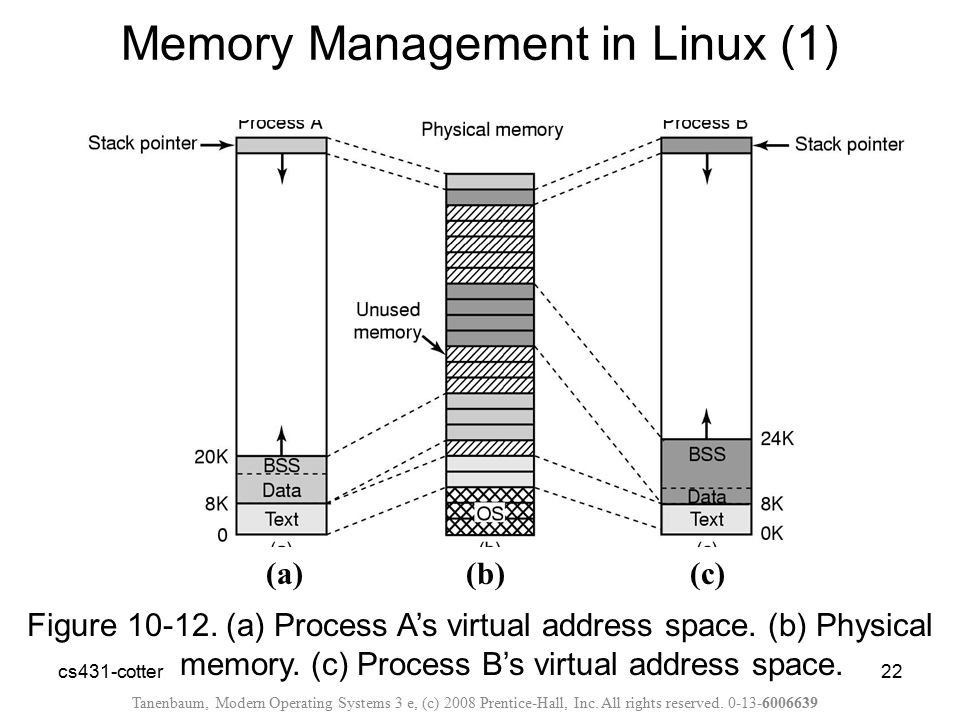


Figure :a.Process A's Virtual Address Space, b. Physical Memory, c. Process B's Virtual Address Space

* The virtual and physical memory is divided into fixed length chunks known as pages.
* Each entry in the theoretical page table contains the following information:
* Valid flag. This indicates if this page table entry is valid,
* The physical page frame number that this entry is describing,
* Access control information. This describes how the page may be used.

[6]

# **File System Management Approaches**

The file is the most basic and fundamental abstraction in Linux. Linux follows the everything-is-a-file philosophy. Consequently, much interaction transpires via filesystem system calls such as reading of and writing to files, even when the object in question is not what you would consider your everyday file. [6]

Linux supports five types of files:

* simple/ordinary file (text file, c++ file, etc)
* directory
* symbolic (soft) link
* special file (device)
* named pipe (FIFO)

In order to be accessed, a file must first be opened. Files can be opened for reading, writing, or both. An open file is referenced via a unique descriptor, a mapping from the metadata associated with the open file back to the specific file itself. Inside the Linux kernel, this descriptor is handled by an integer (of the C type int) called the file descriptor, abbreviated fd. File descriptors are shared with user space, and are used directly by user programs to access files. A large part of Linux system programming consists of opening, manipulating, closing, and otherwise using file descriptors. [6]

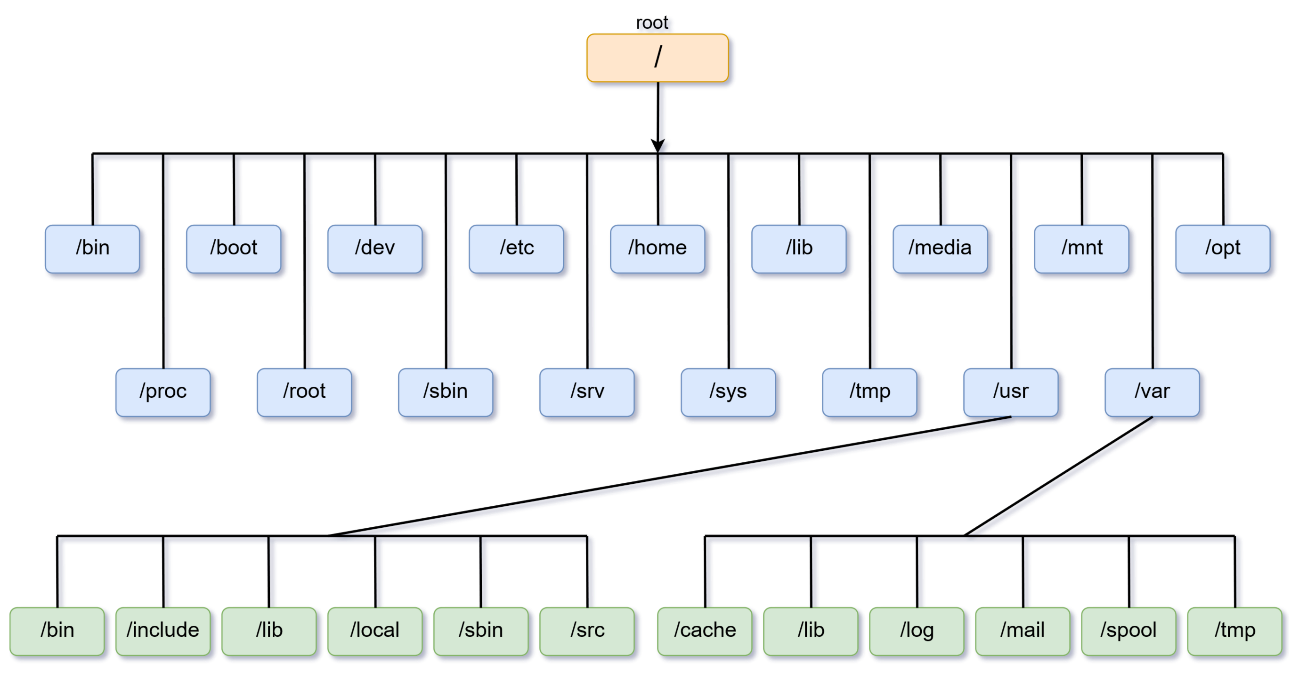


Figure : Linux file system directories

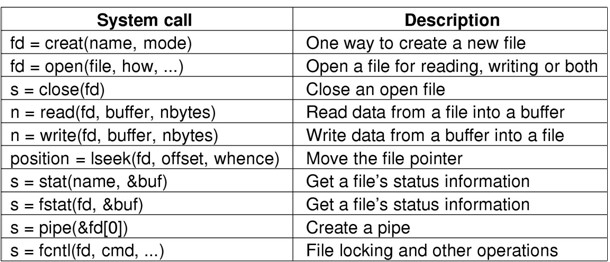


Figure : System calls for file management

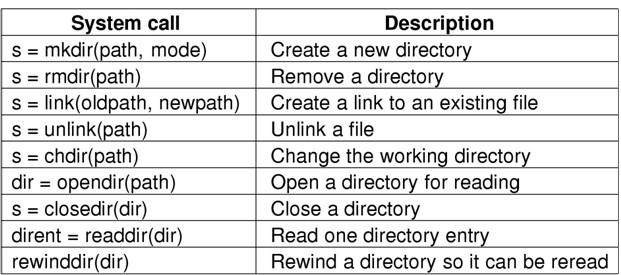


Figure : System calls for Directory Management

# **Device Management Approaches**

Device management is the process of managing the implementation, operation and maintenance of a physical and/or virtual device.

Linux device management starts with user access. IT or DevOps must manage access to the device in order to manage the device itself. Once IT or DevOps controls access, then systems can be put in place without heavy scripting such as from configuration management solutions and with better monitoring and reporting. Device Manager is an application for examining the details of your hardware [9].

**Device Drivers:** The Linux kernel device drivers are essentially a shared library of privileged, memory resident, low level hardware handling routines. It is Linux device drivers that handle the peculiarities of devices they are managing.

**Interfaces:**

* Details of all devices in the system are kept in a /dev directory so that they act and look like normal files residing on the file system.
* Each device file is referred by a major device number and minor device number.

**Types of hardware devices:**

Linux supports three types of hardware device:

* **Character:** Character devices are read and written directly without buffering, for example the system's serial ports /dev/cua0 and /dev/cua1.
* **Block:** Block devices can only be written to and read from in multiples of the block size, typically 512 or 1024 bytes. Block devices are accessed via the buffer cache and may be randomly accessed, that is to say, any block can be read or written no matter where it is on the device. Block devices can be accessed via their device special file but more commonly they are accessed via the file system.
* **Network:** Network devices are accessed via the BSD socket interface and the networking subsytems.

# **8.1 Polling and Interrupts**

Each time the device is given a command, the device driver is given a choice aas to how it finds out the command has completed.

The device drivers can either poll the device or they use interrupts.

* **Polling Approach**
* Uses polling to detect completion of i/o operations.
* User program performs an i/o request using system call.
* Device driver starts the device.
* Periodically polls device to inspect its status for detecting completion of i/o operation.
* **Interrupt Approach**
* Involves
* Device driver
* Interrupt Handler
* Device bottom half
* Steps
* Process issues i/o request
* Device driver checks device status, if available starts i/o operation.
* Process blocks but state is TASK\_INTERRUPTIBLE. [10]

# **8.2 Direct Memory Access (DMA)**

All PCs include an auxiliary processor called the Direct Memory Access Controller (DMAC), which can be instructed to transfer data between the RAM and an I/O device. Once activated by the CPU, the DMAC is able to continue the data transfer on its own; when the data transfer is completed, the DMAC issues an interrupt request.

Several I/O drivers use the Direct Memory Access Controller (DMAC) to speed up operations. The DMAC interacts with the device's I/O controller to perform a data transfer and the kernel includes an easy-to-use set of routines to program the DMAC. The I/O controller signals to the CPU, via an IRQ, when the data transfer has finished.When a device driver sets up a DMA operation for some I/O device, it must specify the memory buffer involved by using bus addresses.

As with IRQ lines, the DMAC is a resource that must be assigned dynamically to the drivers that need it. The way the driver starts and ends DMA operations depends on the type of bus. [11]

# **9. Summary**

* Linux is a modern, free operating system based on UNIX standards. It has been designed to run efficiently and reliably on common PC hardware; it also runs on a variety of other platforms, such as mobile phones. It provides a programming interface and user interface compatible with standard UNIX systems and can run a large number of UNIX applications, including an increasing number of commercially supported applications.
* Linux has not evolved in a vacuum. A complete Linux system includes many components that were developed independently of Linux. The core Linux operating-system kernel is entirely original, but it allows much existing free UNIX software to run, resulting in an entire UNIX-compatible operating system free from proprietary code.
* The Linux kernel is implemented as a traditional monolithic kernel for performance reasons, but it is modular enough in design to allow most drivers to be dynamically loaded and unloaded at run time.
* Linux is a multiuser system, providing protection between processes and running multiple processes according to a time-sharing scheduler. Newly created processes can share selective parts of their execution environment with their parent processes, allowing multithreaded programming.
* Interprocess communication is supported by both System V mechanisms —message queues, semaphores, and shared memory—and BSD’s socket interface. Multiple networking protocols can be accessed simultaneously through the socket interface.
* The memory-management system uses page sharing and copy-on-write to minimize the duplication of data shared by different processes.
* To the user, the file system appears as a hierarchical directory tree that obeys UNIX semantics. Internally, Linux uses an abstraction layer to manage multiple file systems. Device-oriented, networked, and virtual file systems are supported. Device-oriented file systems access disk storage through a page cache that is unified with the virtual memory system. [1]

# **10. References**

|  |  |
| --- | --- |
| [1] | P. B. G. a. G. G. Abraham Silberschatz, Operating System Concepts, John Wiley & Sons Publication, 2018. |
| [2] | M. Garrels, Introduction to Linux, Garrels.be, 2008. |
| [3] | 2020. [Online]. Available: https://blogs.pennmanor.net/1to1/2015/10/30/history-of-linux-timeline/,\. |
| [4] | G. a. G. Silberschatz, 1999. [Online]. Available: https://slideplayer.com/slide/10448169/. |
| [5] | P. B. G. a. G. G. Abraham Silberschatz, Operating System Concepts, John Wiley & Sons Publication, 2018. |
| [6] | S. Rizal, "Case study on linux," https://sritsense.weebly.com/, 2002. |
| [7] | A. Onsman, 2020. [Online]. |
| [8] | E. T. Sushant Gaurav, 2022,2021. [Online]. Available: https://www.scaler.com/topics/operating-system/process-scheduling/. |
| [9] | R. Junior, "techtrix.co," August 2021. [Online]. Available: https://techtrix.co/what-is-device-management-in-linux/. |
| [10] | D. A. R. Ahmad, "Advanced Operating Systems," 2018. |
| [11] | "halolinux.us," 4 july 2022. [Online]. Available: https://www.halolinux.us/kernel-reference/direct-memory-access-dma.html. |
| [12] | Manav, "geeksforgeeks," 2020. [Online]. Available: https://www.geeksforgeeks.org/file-management-in-linux/. |