Kathmandu University

Department of Computer Science and Engineering

Dhulikhel, Kavre

Case Study on POP OS

[Course title: COMP 307]

Submitted by:

Bisheshwor Neupane(35)

Submitted to:

Mr. Dhiraj Shrestha

Department of Computer Science and Engineering

Submission Date:

August 28, 2021

Abstract

The Linux operating system occupies a special position in the world of computer science.Linux is famed both for its stability and for its efficiency, often running for months, or occasionally years at a time without having to be rebooted, while also achieving excellent performance. It conveys many of the properties of UNIX that have made that operating system extremely popular among computer science professionals. Linux source code is as freely available as the executable code, thus giving users complete freedom to modify and adapt the operating system to the special needs of their systems.Pop!_OS is a free and open-source Linux distribution, based upon Ubuntu, featuring a custom GNOME desktop (https://en.wikipedia.org/wiki/Pop!_OS, n.d.). The distribution is developed by American Linux computer manufacturer System76. Pop!_OS is primarily built to be bundled with the computers built by System76, but can also be downloaded and installed on most computers. Pop!_OS provides full out-of-the-box support for both AMD and Nvidia GPUs. (System76, Inc, n.d.)

Table of Contents

Chapter 1: Introduction:

Pop!_OS is a free and open-source Linux distribution, based upon Ubuntu, featuring a custom GNOME desktop. The distribution is developed by American Linux computer manufacturer System76. Pop!_OS is primarily built to be bundled with the computers built by System76, but can also be downloaded and installed on most computers (System76, Inc, n.d.).Pop!_OS provides full out-of-the-box support for both AMD and Nvidia GPUs. It is regarded as an easy distribution to set up for gaming, mainly due to its built-in GPU support. Pop!_OS provides default disk encryption, streamlined window and workspace management, keyboard shortcuts for navigation as well as built-in power management profiles. The latest releases also have packages that allow for easy setup for TensorFlow and CUDA.Pop!_OS primarily uses free software, with some proprietary software used for hardware drivers for Wi-Fi, discrete GPU and media codecs. It comes with a wide range of default software, including LibreOffice, Firefox and Geary. Additional software can be downloaded using the package manager. (https://en.wikipedia.org/wiki/Pop!_OS, n.d.)

Release History:

Release history[3]

Version	Release date	General support until	Base
17.10	2017-10-27	n/a	Ubuntu 17.10
18.04 LTS	2018-04-30	2020-04-30	Ubuntu 18.04 LTS
18.10	2018-10-19	2019-07	Ubuntu 18.10
19.04	2019-04-20	2020-01	Ubuntu 19.04
19.10	2019-10-19	2020-07	Ubuntu 19.10
20.04 LTS	2020-04-30	Next LTS release	Ubuntu 20.04 LTS
20.10	2020-10-23	2021-07	Ubuntu 20.10
21.04	2021-06-29	-	Ubuntu 21.04

Features in POP OS:

- Nvidia Graphics Support in POP OS linux
- Disk Encryption
- Workspace and window management (with autotiling)
- Free software as well as proprietary software
- Reset Linux like Windows without losing data

Chapter 2: Design Principles

POP OS is an open source, portable OS designed based on the principle of advanced features like multi-user, multiprogramming, Hierarchical file system, a different kind of shell and best security in it.

Chapter 2.1: Process Management

Linux Process Management implementation is similar to UNIX implementation. Process management involves various tasks like creation, scheduling, termination of processes, and a deadlock. Process is a program that is under execution, which is an important part of modern-day operating systems. The OS must allocate resources that enable processes to share and exchange information. It also protects the resources of each process from other methods and allows synchronization among processes.

Life Cycle of a process:

When a process creates a new process, the creating process (parent process) issues a fork() system call. When a fork() system call is issued, it gets a process descriptor for the newly created process (child process) and sets a new process id. It copies the values of the parent process' process descriptor to the child's. At this time the entire address space of the parent process is not copied; both processes share the same address space.

The exec() system call copies the new program to the address space of the child process. Because both processes share the same address space, writing new program data causes a page fault exception. At this point, the kernel assigns the new physical page to the child process.

When program execution has completed, the child process terminates with an exit() system call. The exit() system call releases most of the data structure of the process and notifies the parent process of the termination sending a signal

The child process will not be completely removed until the parent process knows of the termination of its child process by the wait() system call. As soon as the parent process is notified of the child process termination, it removes all the data structure of the child process and releases the process descriptor.

THREAD

A thread is an execution unit generated in a single process. It runs parallel with other threads in the same process. They can share the same resources such as memory, address space, open files, and so on. They can access the same set of application data. A thread is also called Light Weight Process (LWP). Because they share resources, each thread should not change their shared resources at the same time. The implementation of mutual exclusion, locking, serialization, and so on, are the user application's responsibility

PROCESS PRIORITY AND NICE LEVEL

Process priority is a number that determines the order in which the process is handled by the CPU and is determined by dynamic priority and static priority. A process which has higher process priority has a greater chance of getting permission to run on a processor.

CONTEXT SWITCHING

During process execution, information on the running process is stored in registers on the processor and its cache. The set of data that is loaded to the register for the executing process is called the context. To switch processes, the context of the running process is stored and the context of the next running process is restored to the register. The process descriptor and the area called kernel mode stack are used to store the context. This switching process is called context switching. Having too much context switching is undesirable because the processor has to flush its register and cache every time to make

INTERRUPT HANDLING

Interrupt handling is one of the highest priority tasks. Interrupts are usually generated by I/O devices such as a network interface card, keyboard, disk controller, serial adapter, and so on. The interrupt handler notifies the Linux kernel of an event (such as keyboard input, ethernet frame arrival, and so on). It tells the kernel to interrupt process execution and perform interrupt handling as quickly as possible because some device requires quick responsiveness. This is critical for system stability. When an interrupt signal arrives to the kernel, the kernel must switch a current execution process to a new one to handle the

In Linux implementations, there are two types of interrupts. A hard interrupt is generated for devices which require responsiveness (disk I/O interrupt, network adapter interrupt, keyboard interrupt, mouse interrupt). A soft interrupt is used for tasks which can be deferred (TCP/IP operation, SCSI protocol operation, and so on). You can see information related to hard interrupts at /proc/interrupts.

PROCESS STATE

Every process has its own state that shows what is currently happening in the process. Process state changes during process execution. Some of the possible states are as follows:

TASK_RUNNING: In this state, a process is running on a CPU or waiting to run in the queue (run queue).

TASK_STOPPED is **a** process suspended by certain signals (for example SIGINT, SIGSTOP) in this state. The process is waiting to be resumed by a signal such as SIGCONT.

TASK_INTERRUPTIBLE : In this state, the process is suspended and waits for a certain condition to be satisfied. If a process is in TASK_INTERRUPTIBLE state and it receives a signal to stop, the process state is changed and operation will be interrupted. A typical example of a

TASK_INTERRUPTIBLE process is a process waiting for keyboard interrupt.

TASK_UNINTERRUPTIBLE Similar to TASK_INTERRUPTIBLE. While a process in

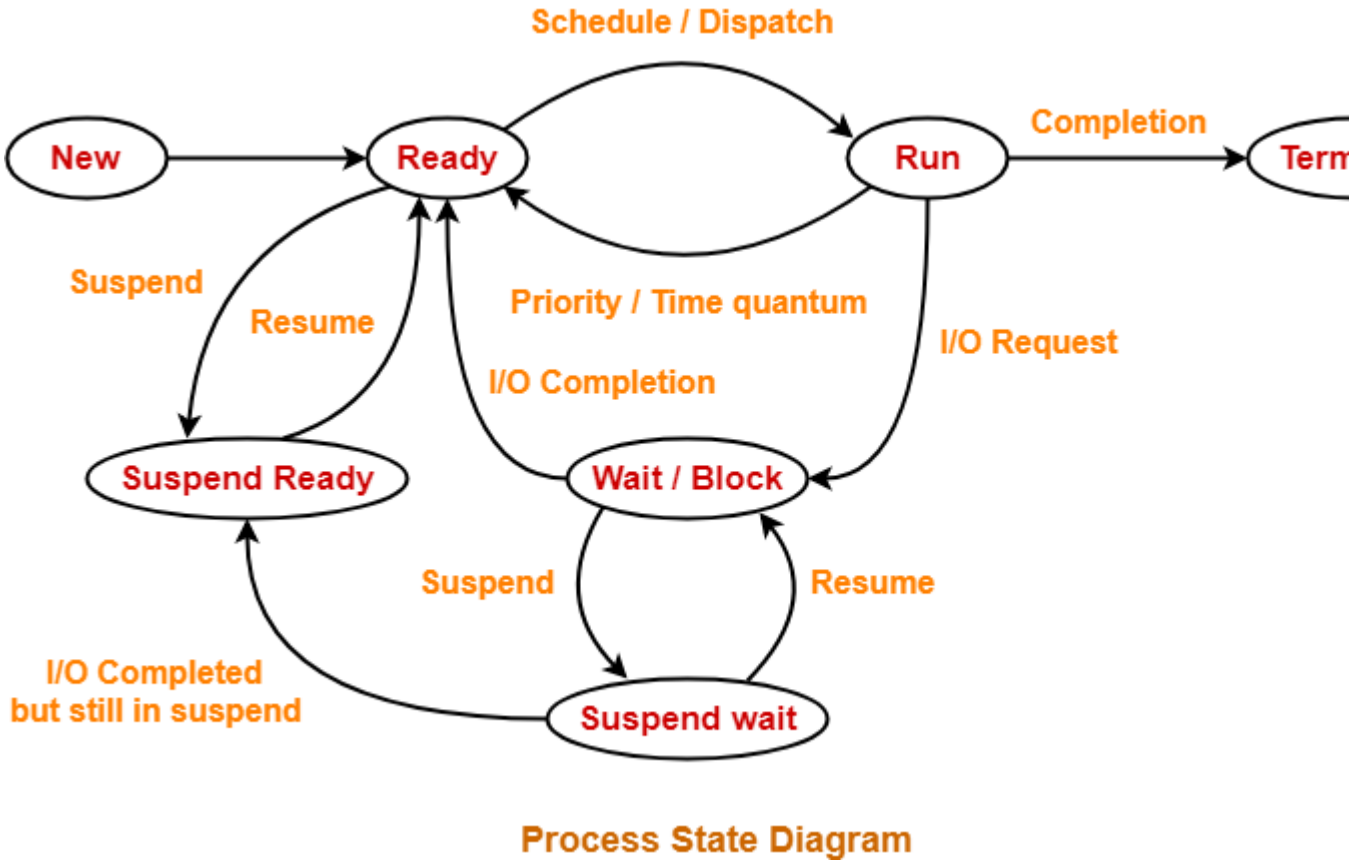
TASK_INTERRUPTIBLE state can be interrupted, sending a signal does nothing to the process in TASK_UNINTERRUPTIBLE state. A typical example of a

TASK_UNINTERRUPTIBLE process is a process waiting for disk I/O operation.

TASK_ZOMBIE After a process exits with exit() system call, its parent should know of the termination. In TASK_ZOMBIE state, a process is waiting for its parent to be notified to release all the data structure.

ZOMBIE PROCESSES

When a process has already terminated, having received a signal to do so, it normally takes some time to finish all tasks (such as closing open files) before ending itself. In that normally very short time frame, the process is a zombie.



Process State Diagram

Scheduling

The scheduler is the kernel component that decides which runnable thread will be executed by the CPU next. Each thread has an associated scheduling policy and a static scheduling priority, sched_priority. The scheduler makes its decisions based on knowledge of the scheduling policy and static priority of all threads on the system.

For threads scheduled under one of the normal scheduling policies (SCHED_OTHER, SCHED_IDLE, SCHED_BATCH), sched_priority is not used in scheduling decisions (it must be specified as 0).

Processes scheduled under one of the real-time policies (SCHED_FIFO, SCHED_RR) have a sched_priority value in the range 1 (low) to 99 (high). (As the numbers imply, real-time threads always have higher priority than normal threads.) Note well: POSIX.1 requires an implementation to support only a minimum 32 distinct priority levels for the real-time policies, and some systems supply just this minimum. Portable programs

should use sched_get_priority_min(2) and sched_get_priority_max(2) to find the range of priorities supported for a particular policy.

Conceptually, the scheduler maintains a list of runnable threads for each possible sched_priority value. In order to determine which thread runs next, the scheduler looks for the nonempty list with the highest static priority and selects the thread at the head of this list.

A thread's scheduling policy determines where it will be inserted into the list of threads with equal static priority and how it will move inside this list.

All scheduling is preemptive: if a thread with a higher static priority becomes ready to run, the currently running thread will be preempted and returned to the wait list for its static priority level. The scheduling policy determines the ordering only within the list of runnable threads with equal static priority.

SCHED_FIFO: First in-first out scheduling

SCHED_FIFO can be used only with static priorities higher than 0, which means that when a SCHED_FIFO threads becomes runnable, it will always immediately preempt any currently running SCHED_OTHER, SCHED_BATCH, or SCHED_IDLE thread. SCHED_FIFO is a simple scheduling algorithm without time slicing. For threads scheduled under the SCHED_FIFO policy, the following rules apply:

1) A running SCHED_FIFO thread that has been preempted by another thread of higher priority will stay at the head of the list for its priority and will resume execution as soon as all threads of higher priority are blocked again.

2) When a blocked SCHED_FIFO thread becomes runnable, it will be inserted at the end of the list for its priority.

3) If a call to sched_setscheduler(2), sched_setparam(2), sched_setattr(2), pthread_setschedparam(3), or pthread_setschedprio(3) changes the priority of the running or runnable SCHED_FIFO thread identified by pid the effect on the thread's position in the list depends on the direction of the change to threads priority:

· If the thread's priority is raised, it is placed at the end of the list for its new priority.As a consequence, it may preempt a currently running thread with the same priority.

· If the thread's priority is unchanged, its position in the run list is unchanged.

· If the thread's priority is lowered, it is placed at the front of the list for its new priority.

According to POSIX.1-2008, changes to a thread's priority (or policy) using any mechanism other than pthread_setschedprio(3) should result in the thread being placed at the end of the list for its priority.

4) A thread calling sched_yield(2) will be put at the end of the list.

No other events will move a thread scheduled under the SCHED_FIFO policy in the wait list of runnable threads with equal static priority.

A SCHED_FIFO thread runs until either it is blocked by an I/O request, it is preempted by a higher priority thread, or it calls sched_yield(2).

SCHED_RR: Round-robin scheduling

SCHED_RR is a simple enhancement of SCHED_FIFO. Everything described above for SCHED_FIFO also applies to SCHED_RR, except that each thread is allowed to run only for a maximum time quantum. If a SCHED_RR thread has been running for a time period equal to or longer than the time quantum, it will be put at the end of the list for its priority. A SCHED_RR thread that has been preempted by a higher priority thread and subsequently resumes execution as a running thread will complete the unexpired portion of its round-robin time quantum. The length of the time quantum can be retrieved using sched_rr_get_interval(2).

SCHED_DEADLINE: Sporadic task model deadline scheduling

Since version 3.14, Linux provides a deadline scheduling policy (SCHED_DEADLINE). This policy is currently implemented using GEDF (Global Earliest Deadline First) in conjunction with CBS (Constant Bandwidth Server). To set and fetch this policy and associated attributes, one must use the Linux-specific sched_setattr(2) and sched_getattr(2) system calls.

A sporadic task is one that has a sequence of jobs, where each job is activated at most once the relative deadline to the arrival time.

per period. Each job also has a relative deadline, before which it should finish execution, and a computation time, which is the CPU time necessary for executing the job. The moment when a task wakes up because a new job has to be executed is called the arrival time (also referred to as the request time or release time). The start time is the time at which a task starts its execution. The absolute deadline is thus obtained by adding.

Kernel synchronization

Tasks may try to access the same internal data structures. Since POP OS is preemptive, if another task with higher priority is ready to use CPU time, the running task will be preempted even if running in the kernel.

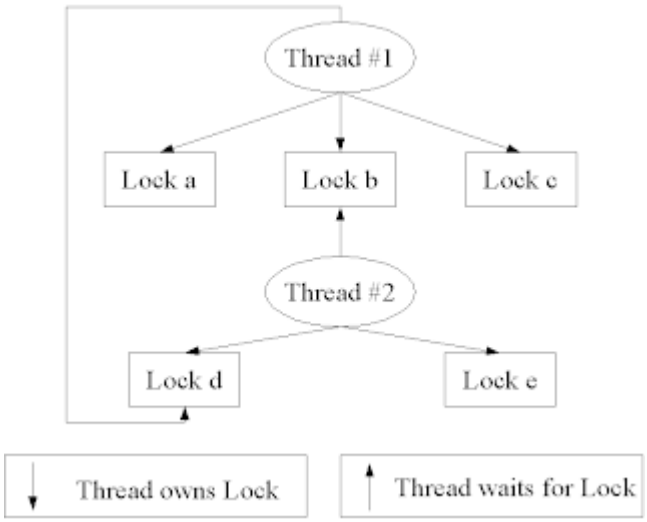
In order to lock the kernel, POP OS provides spinlocks and semaphores.For Symmetric Multiprocessing machines, spinlocks are the fundamental mechanism. This cannot happen in single processor machines since there is only one CPU time and locking for short periods, which is what spinslocks do, would have a high impact on the system performance. Instead of spinlocks, in single processor machines, the preemptive ability is enabled and disabled instead.

Administration utilities used from the command-line

	Name	Description
1	ps	With the appropriate arguments it can show all the running processes, the process running from a user
2	pstree	Shows the running processes in a tree scheme
3	kill	Kill a process by PID
4	killall	Kill a process by name
5	nice	Sets the priority of new process
6	renice	Changes the priority of a currently running process
7	skill	Reports process status

Chapter 2.2 Definition of Deadlock

This can be simply defined as a situation where there are two or more (a set of) processes or threads are mutually blocking each other(Dawson et al., n.d., #). Consider an example consisting of two threads. Thread 1 is waiting for thread 2 to release its lock to perform an action. This event will be understood by looking at the following diagram.



Neither thread will not be able to release any locks. So the execution of the action will be jammed in this circumstance (Sams 2008, p.319). This state can be considered a deadlock kind of a situation.

Mutually blocking for an event can be:

- Blocking for access to a certain section
- Blocking for a resource
- If two programs are sharing the same resource at once, it results in both programs not functioning well.

Conditions for Deadlock

- Mutual exclusion: Resources can not be shared in this phase
- Acquired piece meal: Not taking the resources at once that are needed to complete a certain task

· No preemption: Resources can not be taken away from process while it is functioning
· Resources are not given up (Circular wait): a certain process does not give up its resources until it has satisfied all its outstanding requests for resources. To get more clearer idea of this, see above diagram

How to overcome from Deadlock situation

· Ignore the problem altogether
· Detection and recovery
· Avoidance by careful resource allocation
· Prevention by structurally negating one of the four necessary conditions

Deadlock avoidance

A deadlock can be avoided if more information on executing a process is described. System will see whether it will lead to a deadlock situation when the request of a user is fulfilled. In another words system is looking for *safe* state to fulfill the request of the user. System is calculating this by considering the number and type of all resources in existence, available, and requested.

Deadlock prevention

Utilizing the resources in such a way that will not lead to a deadlock situation is described in this section. The following steps will lead to help from causing deadlocks.

· Eliminating the mutual exclusion clause means all processes can access any resource & proves unfeasible for resources that cannot be spooled, and even with spooled resources deadlock could still occur.
· Requesting resources before starting a task. So there will be no waiting time for resources in between several processes. That will assist to get rid of the deadlock situation.
· Removing no preemption condition may be hard to ignore because each process should have at least few resources to perform a given task.

Circular wait conditions can be easily removed. So that a process may be allowed to possess only one resource at a time, or a ranking may be imposed such that no waiting cycles are possible. Hierarchical order can be used to determine the allocation of resources into process.

Deadlock detection

Deadlock detection and clean up is used by employing an algorithm that tracks the circular waiting and kills one or more of the processes. Deadlock can be removed easily with that feat. It should be noted that this problem is undecidable in general, because the halting problem can be rephrased as a deadlock scenario (Dawson et al., n.d., #).

Chapter 2.3 Memory Management

POP OS as a GNU/Linux distribution based on UBUNTU is designed for different architectures. Due to this characteristic, POP OS separates physical memory into three different zones:

zone_DMA

It is the zone that includes the first 16MB of physical memory. It consists of the DMA-able pages (Direct Access Memory). This happens because certain hardware, Industry Standard Architecture (ISA) devices, can only access the lower 16MB of physical memory.

zone_NORMAL

It is the zone that includes the normally addressable pages up to the first 896 MB of physical memory.

zone_HIGHMEM

The remaining physical memory (>896 MB) is allocated from zone_HIGHMEM. It is referred to as high memory and includes dynamically mapped pages.

There are two mechanisms for allocating Kernel memory:

- buddy service technique
- slab allocation

kmalloc is the normal method of allocating memory in the kernel.

Using a “buddy service” type technique of allocation, the kmalloc() service is used, when the size of the request is not known in advance and it can be a few bytes, to allocate entire pages on demand and then split them into smaller pieces. The allocation by kmalloc() service is permanent until they are freed explicitly.

The slab technique is used for allocating memory for Kernel data structures and is made up for one or more physically contiguous pages”. The slab allocation algorithm uses caches to store kernel objects. A cache consists of one or more slabs. Each kernel structure has its own cache. “A slab may be in one of three possible states:

-
- Full. All objects in the slab are marked as used.
- Empty. All objects in the slab are marked as free.
- Partial. The slab consists of both used and free objects.

Priority is first given to partial slabs. If one does not exist, then a free object is assigned from a free one.

Except for these two main memory managers, there are two more subsystems that do their own management of physical-memory: the page cache and the virtual memory system.

The page cache is the kernel's main cache for block devices and memory-mapped files and is the main mechanism through which I/O to these devices is performed. The page cache can also cache networked data. Block devices provide the main interface for disks in a system.

POP OS makes very high use of virtual memory. mmap() service is responsible for mapping files or devices into memory. The page-table entries that map to these kernel pages are marked as protected. Two regions are used, one where the core of the kernel, along with all pages allocated by the normal page allocator reside, and another which is reserved for no specific purpose. The vmalloc() service is used to allocate an arbitrary number of physical pages of memory that may not be contiguous into a single region of virtually contiguous kernel memory.

The mmap() service has flag arguments that transform the mapping to be shared with other processes, or create a private copy-on-write mapping. The virtual mapping procedure is directly connected to the way the process was created. If it is created through the exec() service then a new mapping address is provided. If it is created through fork() then the parent's pages are directly copied to the child's, resulting in the parent and child sharing physical pages in memory.

Using Command:

$top command can be customized about the available shown options.

First we get an overview of the current situation, we identify the target process, get the PID and then we move to the proc file at /proc for further investigation.

Additional commands and files containing memory information are shown below:

free -m	Limited memory information
/proc/meminfo	Full memory information

/proc/interrupts	System interrupts served per CPU
uptime	System uptime
vmstat -s	Virtual memory information
sudo sh -c "sync; echo 3 > /proc/sys/vm/drop_caches"	Drops caches
pagesize	Page size currently used
uname -r	Kernel version (PAE included)
sudo swapoff -a	Turning swap off
sudo swapon -a	Turning swap on

Chapter 2.4 Virtual Memory Management

The Virtual Memory management system is one of the most important parts of an operating system. Since the beginning of computer operations there has been a need for more memory than the exciting physical memory in a computer. To fulfill this need many strategies have been developed in the present. Virtual memory is the most successful method of them all. Virtual memory makes the computer appear to have more memory than its actual physical memory.

Virtual Memory in POP OS Desktop Edition

POP OS Linux based operating system's virtual memory is known as the swap. Swap describes the transactions of moving memory pages between physical RAM and the hard Disk, and the region of a disk the pages are stored on. POP OS Commonly uses a whole partition of a hard disk for swapping. That partition is called swap space. Swap space holds the virtual memory of the system. Swapping is a method that allows a computer to execute programs and process data files larger than its own physical memory.

Swap Space

Swap space is located on hard drives, which have a slower access time than physical memory. Swap space can be in the same partition of the hard disk, as a separate partition dedicated to it on the same disk, a dedicated partition on a second hard disk or even a combination of swap partition and swap file. Ubuntu has two forms of swap spaces, the swap partition and the swap file. The swap partition is an independent section of the hard disk used solely for swapping; no other files can reside there. The swap file is a special file in the file system that resides amongst your system and data files. It is recommended to allocate at least 2x capacity of the actual physical memory. Swap space is also used for system hibernation.

Swap Partition

The advantage of using swap partitions, it can be used by two or more Linux based installations on the same PC. Example - dual boot or triple boot with a Windows based operating system. Also a Swap partition comes in handy when the disk is full where the swap

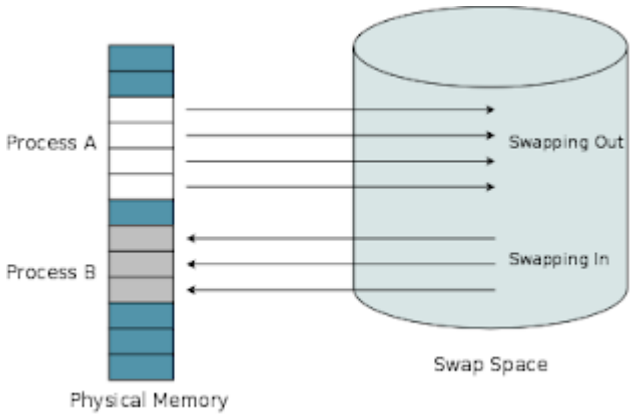
file could be created incompletely. Swap partitions are faster than swap files, and they are commonly used in Linux systems.

Swap file

As well as the swap partition, Linux also supports a swap file that you can create, prepare, and mount in a fashion similar to that of a swap partition. The advantage of swap files is that you don't need to find an empty partition or repartition a disk to add additional swap space . On the other hand swap files are good if you need to change the size of the swap memory regularly as it is easier to resize the swap file through the whole partition.

Swap-In and Swap-Out

In POP OS, the swap space is used when all the physical memory on your computer is reserved by running processes yet an additional demand for memory resources exists. In such scenarios where the physical memory is at full capacity, the kernel allocates inactive pages in the physical memory to the swap space. (Dawson et al., n.d., #) This creates additional space in the physical memory to meet the additional demand. When the pages that are in the swap space are needed by the physical memory again, they are swapped back into the physical memory. These operations are called swapping out and swapping in, respectively.



Available From: http://www.bytetrap.com/blog/2008/06/02/swap_space.png

Demand Paging

POP OS uses this technique to load virtual pages into physical memory as they are accessed is known as demand paging. Ubuntu uses demand paging to load executable images into a process's virtual memory. Whenever a command is executed, the file containing it is opened and its contents are mapped into the process's virtual memory. This is done by modifying the data structures describing this process' memory map and is known as memory mapping. (Pearson Prentice Hall, n.d., #) However, only the first part of the image is actually brought into physical memory. The rest of the image is left on disk. As the image executes, it generates page faults and Linux uses the process's memory map in order to determine which parts of the image to bring into memory for execution.

Chapter 2.5 File System

Linux uses UNIX's standard file-system model, where everything appears in the form of files for the user. However a file does not have to be stored as an object on a disk, it can be a stream of data. The kernel handles all these types of files by hiding the implementation details of any type behind a layer of software, the virtual file system (VFS).The VFS is designed with object-oriented principles. It has two components. One specifies what file-system objects are allowed to look like and a layer of software to manipulate the objects. The VFS works with objects each one contains a pointer to a function table. (Dawson et al., n.d., #)

The main VFS objects are:

- inode object
- file object

Both are mechanisms to access files. Inode objects contain pointers to the disk blocks and the file object contains pointers to the actual file data.

the superblock object

Its main responsibility is to provide access to inodes. It is a connected set of files that form a self-contained file-system. The VFS keeps in pairs the file-system/inode numbers to identify each node and it uses the superblock object to get the inode with that pair.

the dentry object

It represents the directory entries found in a path name.

(Operating System Concepts, p.469 / p.830)

File systems in Linux are either loaded during boot time, or manually.

File systems and POP OS

POP OS supports the Linux Logical Volume Manager, which allows us to increase the size of a partition online, while the system is running. In addition, POP OS allows us to use different devices as mount points for several “heavy/crucial” directories. These are:

-
- /boot, the directory containing the basic instructions to load the OS
- / (root) directory
- /home, the user(s) home directory, equivalent to “My documents” folder in MS Windows

A full list of the supported file systems in POP OS can be found using the “cat” command in the virtual file system /proc as shown below.

cat /proc/filesystems

Table 1. Supported file systems in POP OS according to POP OS natty man pages.

minix	is the file system used in the Minix operating system, the first to run under Linux. It has a number of shortcomings: a 64MB partition size limit, short filenames, a single timestamp, etc. It remains useful for floppies and RAM disks.
ext	an elaborate extension of the minix file system. It has been completely superseded by the second version of the extended file system (ext2) and has been removed from the kernel (in 2.1.21).
ext2	the high performance disk file system used by Linux for fixed disks as well as removable media. The second extended file system was designed as an extension of the extended file system (ext). ext2 offers the best performance (in terms of speed and CPU usage) of the file systems supported under Linux.
ext3	a journaling version of the ext2 file system. It is easy to switch back and forth between ext2 and ext3.
ext4	a set of upgrades to ext3 including substantial performance and reliability enhancements, plus large increases in volume, file, and directory size limits.
Reiserfs	a journaling file system, designed by Hans Reiser, that was integrated into Linux in kernel 2.4.1.
XFS	a journaling file system, developed by SGI, that was integrated into Linux in kernel 2.4.20.
JFS	a journaling file system, developed by IBM, that was integrated into Linux in kernel 2.4.24.
msdos	the file system used by DOS, Windows, and some OS/2 computers. ms dos file names can be no longer than 8 characters, followed by an optional period and 3 character extension.
umsdos	an extended DOS file system used by Linux. It adds capability for long filenames, UID/GID, POSIX permissions, and special files (devices, named pipes, etc.) under the DOS file system, without sacrificing compatibility with DOS.
vfat	an extended DOS file system used by Microsoft Windows95 and Windows NT. VFAT adds the capability to use long filenames under the MSDOS file system.

ntfs	replaces Microsoft Windows FAT file systems (VFAT, FAT32). It has reliability, performance, and space-utilization enhancements plus features like ACLs, journaling, encryption, and so on.
proc	a pseudo file system which is used as an interface to kernel data structures rather than reading and interpreting /dev/kmem. In particular, its files do not take disk space.
iso9660	a CD-ROM file system type conforming to the ISO 9660 standard.
High Sierra	Linux supports High Sierra, the precursor to the ISO 9660 standard for CD-ROM file systems. It is automatically recognized within the iso9660 file-system support under Linux.
Rock Ridge	Linux also supports the System Use Sharing Protocol records specified by the Rock Ridge Interchange Protocol. They are used to further describe the files in the iso9660 file system to a Unix host, and provide information such as long filenames, UID/GID, POSIX permissions, and devices. It is automatically recognized within the iso9660 file-system support under Linux.
hpfs	the High Performance Filesystem, used in OS/2. This file system is read-only under Linux due to the lack of available documentation.
sysv	an implementation of the SystemV/Coherent file system for Linux. It implements all of Xenix FS, SystemV/386 FS, and Coherent FS.
nfs	the network file system used to access disks located on remote computers.
smb	a network file system that supports the SMB protocol, used by Windows for Workgroups, Windows NT, and Lan Manager.
ncpfs	a network file system that supports the NCP protocol, used by Novell NetWare.

However, support for additional file systems can be added by adding additional modules to the kernel.

As it can be seen above, POP OS supports several file systems. The result when using the cat /proc/filesystems – command, will include several virtual file systems connected to no devices that are used from the POP OS kernel for administrative actions, such as memory caching. e.g.

“**ramfs**, a very simple file system that exports Linux's disk caching mechanisms (the page cache and dentry cache) as a dynamically resizable RAM-based file system. **rootfs**, a special instance of ramfs (or tmpfs, if that's enabled), which is always present in 2.6 systems. You can't unmount rootfs for approximately the same reason you can't kill the init process; rather

than having special code to check for and handle an empty list, it's smaller and simpler for the kernel to just make sure certain lists can't become empty.” Adding to those above, the swap file system must be noted as well, even though it is a raw file system, used for memory caching. Swap is a very important parameter in Linux.

The ones available to use in a physical drive during a clean installation, using the default partitioning tool are:

- ext2
- ext3
- ext4
- reiserfs
- xfs
- jfs
- btrfs

A very brief comparison of the File systems most commonly found in POP OS.

File System	Max File Size	Max Partition Size	Journaling	Notes
Fat16	2 GB	2 GB	No	Legacy
Fat32	4 GB	8 TB	No	Legacy
NTFS	2 TB	256 TB	Yes	For Windows Compatibility) NTFS-3g is installed by default in Ubuntu, allowing Read/Write support)
ext2	2 TB	32 TB	No	Legacy
ext3	2 TB	31 TB	Yes	Standard linux filesystem for many years. Best choice for super-standard installation.
ext4	16 TB	1EB	Yes	Modern iteration of ext3. Best choice for new installations where super-standard isn't necessary.
reiserfs	8 TB	16 TB	Yes	No longer well-maintained.
JFS	4 PB	32 PB	Yes (metadata)	Created by IBM - Not well maintained.

XFS	8 EB	8 EB	Yes (metadata)	Created by SGI. Best choice for a mix of stability and advanced journaling.
GB = Gigabyte (1024 MB) :: TB = Terabyte (1024 GB) :: PB = Petabyte (1024 TB) :: EB = Exabyte (1024 PB)				

However, as mentioned before, not all are available during installation to be used in a physical drive, since not all support the read/write and file permissions required for Ubuntu to work properly. NTFS cannot be used to install Linux at all, and FAT16/32 are not recommended to install Linux, since FAT does not have any of the permissions of a true Unix FS. They are used for compatibility reasons with Microsoft Windows.

From the file systems described above, the most commonly used for POP OS installation, are:

- ext2/ext3
- ext4

  and
- XFS

Ext2 as shown in table 1, is not a journaling FS. It is still commonly used however, to mount in boot time the /boot directory. Since it is not a journaling system, there are less read/write operations and since the /boot directory is only used at boot time to load the kernel, ext2 can result in faster OS boot.

Ext3 is the successor of ext2. However, it is a journaling system. It has been the default file system for a lot of years and since the LTS versions starting from 6.04 are only now starting to expire, many still use this file system for their / (root) and /home directories. POP OS supports the conversion from ext3 to ext4, the new default journaling default file system. The possible limit for a sub directory is 32000. It uses an indirect block mapping scheme to keep track of each block used for the blocks corresponding to the data of a file.

(http://kernelnewbies.org/Ext4)

Ext4 is the evolution of ext3 and it is a journaling file system, but it is an attribute that can be disabled. Ext4 breaks the subdirectory limit and allows an unlimited number of sub directories. Moreover, it uses “extents” to improve performance for large files. “For example, a 100 MB file can be allocated into a single extent of that size, instead of needing to create the indirect mapping for 25600 blocks (4 KB per block). Huge files are split in several extents. Extents improve the performance and also help to reduce the fragmentation, since an extent encourages continuous layouts on the disk. When Ext3 needs to write new data to the disk, there's a block allocator that decides which free blocks will be used to write the data. But the Ext3 block allocator only allocates one block (4KB) at a time. That means that if the system

needs to write the 100 MB data mentioned in the previous point, it will need to call the block allocator 25600 times (and it was just 100 MB!). Not only is this inefficient, it doesn't allow the block allocator to optimize the allocation policy because it doesn't know how much total data is being allocated, it only knows about a single block. Ext4 uses a "multiblock allocator" (mballoc) which allocates many blocks in a single call, instead of a single block per call, avoiding a lot of overhead. This improves the performance, and it's particularly useful with delayed allocation and extents." (http://kernelnewbies.org/Ext4)

In addition, delayed allocation procedure is used in ext4 improving performance and reducing fragmentation and it is supported for online defragmentation.

XFS is a journaled 64-bit file system. It is organized in balanced B+Trees and uses Extends to store data. XFS is fully supported by all POP OS-Versions. It is not recommended to be used to mount the /boot directory, due to GRUB support. It is used to mount the root directory and the /home one.

Advantages

- Part of official Kernel
- Online optimizing (defragmenting) filesystem using "xfs_fsr" (part of xfsdump-package)
- Ability to grow an existing filesystem using "xfs_growfs" (part of xfsprogs-package)
- Own dump- and restore-utilities to keep fs-specific info
- Fast seeking through directories
- Efficient file storage, low fragmentation
- Relatively low memory usage
- Journal (internal or external)
- Balance between speed and safety
- Delayed allocation and improved sparse file handling

Disadvantages

- Slower than non-journaled fs (like ext2)
- Due to balance of speed and safety, there is the possibility to lose data on abrupt power loss (outdated journal metadata), but not file system consistency
- Currently there is no ability to shrink an xfs filesystem. However, there are some people working on this issue
- GRUB-Support is in an early stage

The journal is currently not 32-bit / 64-bit portable. Before mounting a xfs prevously used on 32-bit linux in 64-bit linux (or different arch) the journal has to be emptied using xfs_repair. (System76, Inc, n.d.)

Tools for the ext2,ext3,ext4 and XFS file systems in POP OS.

The available tools for file system information and modifications are separated in this paper in two parts. The one where only the graphical tools are explained, and the second one where the commands in terminal are briefly explained.

GUI tools

A fast overview of the available file systems can be found in the system monitor, in the file systems tab.

Further actions and information for the available storage devices can be found in the Disk Utility in the Control Center.

Disk Utility provides us with an overview of the storage devices currently connected in the system. It also provides us with an easy graphical interface to:

- get hardware information
- mount devices
- unmount devices
- check the filesystem in the device
- edit partitions
- delete partitions
- format volumes

Chapter 2.6: I/O SYSTEM

The Linux device-oriented file system accesses disk storage through two caches:

- Data is cached in the page cache, which is unified with the virtual memory system
- Metadata is cached in the buffer cache, a separate cache indexed by the physical disk block

Linux splits all devices into three classes:

block devices allow random access to completely independent, fixed size blocks of data

character devices include most other devices; they don't need to support the functionality of regular files

network devices are interfaced via the kernel's networking subsystem

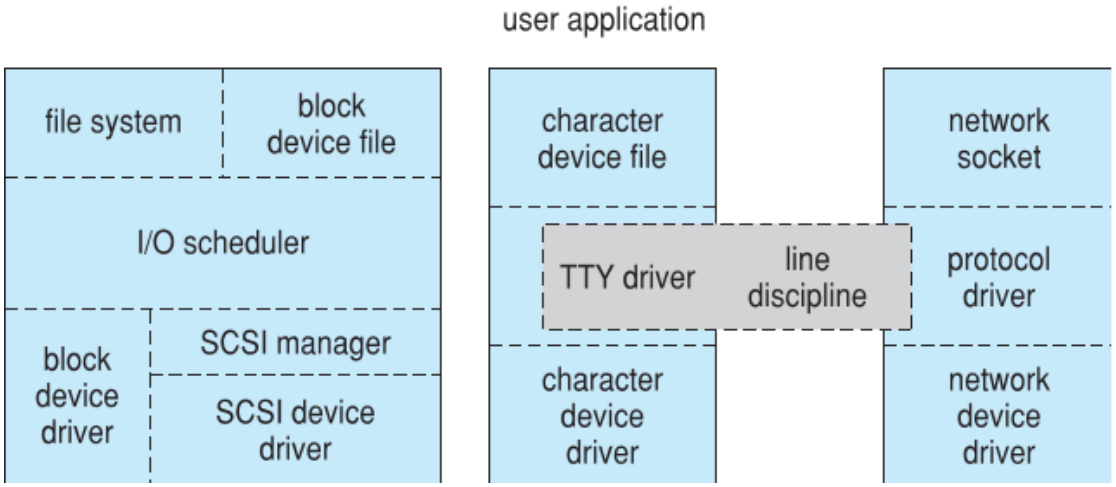


fig: device-driver block structure

Character Devices:

- A device driver which does not offer random access to fixed blocks of data
- A character device driver must register a set of functions which implement the driver's various file I/O operations
- The kernel performs almost no preprocessing of a file read or write request to a character device, but simply passes on the request to the device
- The main exception to this rule is the special subset of character device drivers which implement terminal devices, for which the kernel maintains a standard interface

Line discipline is an interpreter for the information from the terminal device

- The most common line discipline is tty discipline, which glues the terminal's data stream onto standard input and output streams of user's running processes, allowing processes to communicate directly with the user's terminal
- Several processes may be running simultaneously, tty line discipline responsible for attaching and detaching terminal's input and output from various processes connected to it as processes are suspended or awakened by user
- Other line disciplines also are implemented have nothing to do with I/O to user process – i.e. PPP and SLIP networking protocols

Network Structure

- Networking is a key area of functionality for Linux
    - It supports the standard Internet protocols for UNIX to UNIX communications
    - It also implements protocols native to non-UNIX operating systems, in particular, protocols used on PC networks, such as Appletalk and IPX
- Internally, networking in the Linux kernel is implemented by three layers of software:
    - The socket interface
    - Protocol drivers
    - Network device drivers
- Most important set of protocols in the Linux networking system is the internet protocol suite
    - It implements routing between different hosts anywhere on the network
    - On top of the routing protocol are built the UDP, TCP and ICMP protocols
- Packets also pass to firewall management for filtering based on firewall chains of rules

Security:

- The pluggable authentication modules (PAM) system is available under Linux
- PAM is based on a shared library that can be used by any system component that needs to authenticate users
- Access control under UNIX systems, including Linux, is performed through the use of unique numeric identifiers (uid and gid)
- Access control is performed by assigning objects a protections mask, which specifies which access modes—read, write, or execute—are to be granted to processes with owner, group, or world access
- Linux augments the standard UNIX setuid mechanism in two ways:

- It implements the POSIX specification's saved user-id mechanism, which allows a process to repeatedly drop and reacquire its effective uid
- It has added a process characteristic that grants just a subset of the rights of the effective uid
- Linux provides another mechanism that allows a client to selectively pass access to a single file to some server process without granting it any other privileges

Chapter 3: Conclusion:

This research attempted to answer the following research question: how POP OS functions based on process management, deadlock management, memory management, virtual memory management and file system management ? For this purpose, several factors from the OS were examined. While the preliminary findings demonstrate that POP OS users do not require technical support and perceive the system as well integrated and not inconsistent, the study is limited. Future studies should attempt to examine the factors through the prism of different instruments and with a greater number of subjects.

Bibliography

Dawson, M., DeWalt, B., & Cleveland, S. (n.d.). *The Case for UBUNTU Linux Operating System Performance and Usability for Use in Higher Education in a Virtualized Environment*. Association for Information Systems AIS Electronic Library (AISeL).

https://en.wikipedia.org/wiki/Pop!_OS. (n.d.). *Pop!_OS*. wikipedia. https://en.wikipedia.org/wiki/Pop!_OS

Pearson Prentice Hall. (n.d.). CASE STUDY 1: Linux.

System76, Inc. (n.d.). *System76*. Pop!_OS. https://pop.system76.com/