A Case Study

Linux OS

Agenda

1. Introduction
2. Design principles
3. Kernel modules
4. Process management

Introduction

Linux kernel:

Original piece of software built by the linux community that manages all the system resources and interacts directly with the computer hardware.

Linux system:

Includes more components and is a standard environment for applications and user programming. However, It does not include any standard means of managing the available functionality as a whole.

Linux Distributions:

Includes all standard components, has administrative tools to simplify installation and removal of other packages on the system, has tools for management of file systems, user accounts, administers networks, web browsers etc.

Design principles

1. Introduction
2. Components of a linux system

Design Principles - Introduction

- Non-microkernel based operating system
- Multiuser
- Preemptively multitasking
- Includes unix standard networking models
- Capable of running on multiple architectures, with maximum functionality even with limited resources.
- Main design goals:
  - Speed
  - Efficiency
  - Standardization (has to comply with POSIX standards, standard user and programmer interfaces)

Design principles - Components

1. Kernel : Maintains all important abstractions, manages memory and processes. It is monolithic and improves performance.
2. System libraries: includes a standard set of functions through with applications can interact with the kernel without full privileges of the kernel mode. Eg libc
3. System utilities: programs that perform individual specialized management tasks. Some are invoked once during configurations, daemons run permanently.

Design principles - Components

Kernel

Linux has a monolithic kernel

All kernel code including code for device drivers, filesystems and networking and the datastructures are kept in a single address space.

1. There are no context switches when processes calls an operating system function or when a hardware interrupt is delivered.
2. Kernel can pass data and make requests between subsystems using cheap C function instead of using IPC.

Design principles - Components

System libraries:

System calls transfer control from user mode to kernel mode.

More complex versions of basic system calls provide more advanced control of functionalities. Eg C languages’s buffered file handling functions

Provide routines like sorting algorithms, math functions etc.

Design principles - Components

Utilities:

Linux provides both system and user utilities.

System utilities include all the programs used to initialize and administer the system for purposes like setting up network interfaces, adding and removing users from system.

User utilities, do not need elevated privileges and includes utilities for simple file management like copying files, creating directories etc. The shell is also an important user utility.

Kernel modules

1. Introduction
2. Module management system
3. Module loader and unloader
4. Driver registration system
5. Conflict resolution mechanism

Kernel modules

Kernel can load and unload modules that run in kernel mode dynamically at run time.

These modules can implement device drivers, a file system or a networking protocol.

A module can be compiled on its own and loaded into a running kernel for use and distribution.

A module like a device driver can be loaded explicitly by the system at startup or automatically on demand and unloaded when not in use. Eg a mouse driver

Linux module support - Module management system

Allows modules to be loaded into the memory and to communicate with the rest of the kernel.

Ensures that all the references to kernel symbols or entry points are updated to the current locations in the kernel’s address space.

Linux maintains an internal symbol table comprised of symbols that have been explicitly exported from modules and these symbols constitute the module-kernel interactive interface.

External symbols referenced by modules but not declared by it are marked as unresolved in the final module binary produced by the compiler.

A system utility, tries to resolve unresolved references by referring to the symbol table, If some cannot be resolved, the module is rejected.

Linux module support - Module management system

Module loader requests the kernel to reserve a continuous area of virtual kernel memory and the kernel returns the address of the memory allocated.

The loader utility relocates the module’s machine code to the correct loading address.

A second system call passes the module and exported symbol table to the kernel and the module is copied into the memory space and the kernel symbol table is updated.

Linux module support - Module management system

The final module-management component is the module requester.

The kernel defines a communication interface to which a module management program can connect.

When a process requests a module that is not loaded, the manager will load the service and the original request will be complete.

The manager process regularly queries the kernel to check if a module is in use, and unloads the module when not needed.

Linux module support - Driver Registration

The kernel maintains a dynamic table of drivers, provides routines to add and remove drivers, calls the module’s start up routine when loaded and clean up routine before it is unloaded and registers a module’s functionality.

Registration tables include :

Device drivers: include character devices, block devices, network interface devices.

File systems : for implementing virtual file system calling routines, implementing a format for storing files on a disk or network file system .

Network protocols: For implementing an entire protocol like TCP or packet filtering rules for network firewall.

Binary format: For recognizing, loading and executing a new type of executable file.

Linux module support - Conflict Resolution

Linux provides a central conflict resolution mechanism to help arbitrate access to certain hardware resources.

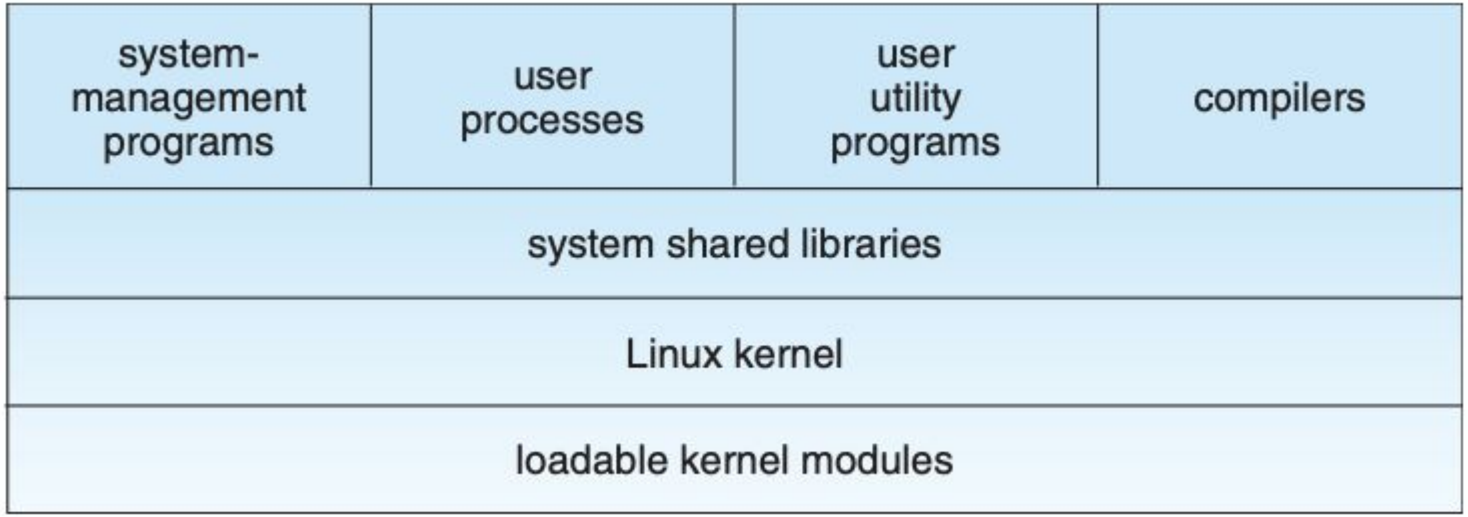
- To prevent modules from clashing over access to hardware resources
- To prevent autoprobes (device driver probes that auto detect device configuration from interfering with existing drivers)
- Resolve conflicts with multiple drivers trying to access the same hardware

1. Kernel maintains list of allocated HW resources

2. Driver reserves resources with kernel database first

3. Reservation request rejected if resource not available

Design principles - Components



Process management

1. Fork() and exec() model
2. Processes and threads

Process management - fork() and exec()

UNIX process management separates the creation of processes and the running of a new program into two distinct operations.

– The fork() system call creates a new process

– A new program is run after a call to exec()

Under UNIX, a process encompasses all the information that the operating system must maintain to track the context of a single execution of a single program

Process management - fork() and exec()

Process properties include:

Process identity

Process environment

Process context

Process management - Process identity

Process ID (PID) - The unique identifier for the process; used to specify processes to the operating system when an application makes a system call to signal, modify, or wait for another process

• Credentials - Each process must have an associated user ID and one or more group IDs that determine the process's rights to access system resources and files

• Personality - Under Linux each process has an associated personality identifier that can slightly modify the semantics of certain system calls

– Used primarily by emulation libraries to request that system calls be compatible with certain specific flavors of UNIX

• Namespace – Specific view of file system hierarchy

– Most processes share common namespace and operate on a shared file-system hierarchy but each can have unique file-system hierarchy with its own root directory and set of mounted file systems

Process management - Process Environment

The process’s environment is inherited from its parent, stored in the process’s own user mode address space, and is composed of two null-terminated vectors:

– The argument vector lists the command-line arguments used to invoke the running program; conventionally starts with the name of the program itself

– The environment vector is a list of “NAME=VALUE” pairs that associates named environment variables with arbitrary textual values.

A new environment is set up for new programs eg a process must supply the environment when calling exec()

Passing environment variables among processes and inheriting variables by a process’s children are flexible means of passing information to components of the user-mode system software.

The environment-variable mechanism provides a customization of the operating system that can be set on a per-process basis, rather than being configured for the system as a whole

Process management - Process context

The constantly changing state of a running program at any point in time.

Context includes :

1. Scheduling context : Includes saved copies of process's registers, scheduling priority, outstanding signals to be delivered to the process, information about the kernel stack (ie) Information required by the scheduler to suspend and restart processes.
2. Accounting : Information about the resources being consumed by each process and total resources consumed by the process in it's entire life time.
3. File table : an array of pointers to the kernel file structure representing open files. The file descriptor is the index of this table.
4. File-system context : applies to requests to opennew files. Includes the process's root directory, current working directory and namespace.

Process management - Process context

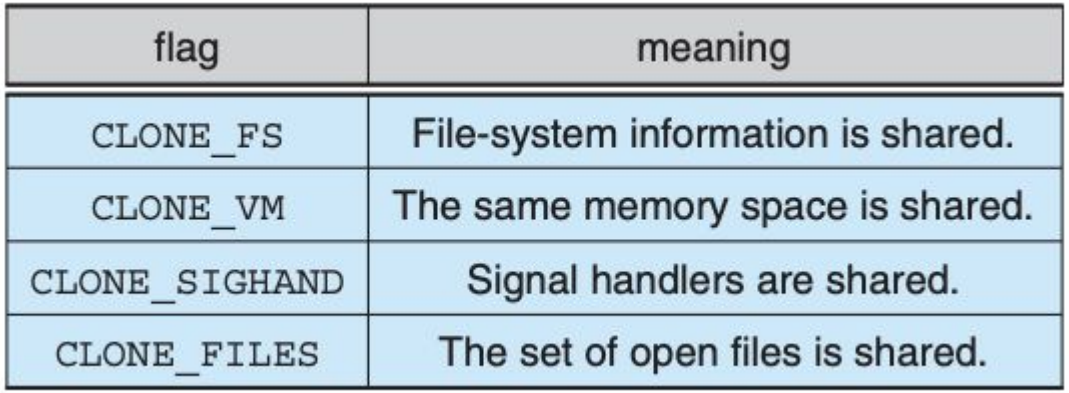
5. Signal handler table: defines the action to take in response to a specific signal.

6. Virtual memory context: describes full contents of a process’s private address space.

Process management - Processes and threads(tasks)

A thread is simply a new process that happens to share parts of it’s context with its parent .

Thread is created using the clone() system call that takes flags to dictate the resources shared.



flag	meaning
CLONE_FS	File-system information is shared.
CLONE_VM	The same memory space is shared.
CLONE_SIGHAND	Signal handlers are shared.
CLONE_FILES	The set of open files is shared.

Process management - Processes and threads(tasks)

Lack of distinction is possible because process contexts are held in separate sub contexts in separate data structures.

The process datastructure contains pointers to these structures and hence making the subcontexts easily sharable according to the arguments passed to clone()