More on Linux

Prof. P.C.P. Bhatt

Linux : Historical Back-ground

- Linus Torvald’s quest to have a system on PC which “feels like Unix” led to the development of Linux OS.
- Quite recent as it was first attempted on a 80386 processor.
- Since the code was made public, many users have contributed since and still evolving!!

A Little More on History

- *Andrew Tenenbaum* had a small OS called *Minix* for teaching purposes. *Linus Torvalds* had his first experiences with *Minix*. Even today a good number of carry with nostalgia a *"minix"* file system option.
- Kernel version 1.0 did not support peripherals. Subsequent versions 1.1 began with support for floppy drives, CD ROM devices, sound cards and other devices.
- From March 1995, version 1.2 Linux also emulates the MS environment and also was adopted by SUN for promoting Unix like environment.
- Version 2.0 is supported on a variety of platforms and processors – intel, saprc and Motorola. Current version: 2.6.

The Kernel: Background

- The Linux kernel development is the core of the project.
- Linux borrows heavily from BSD Unix, the X-Windows from MIT and the GNU project of Open systems foundations.
- Linux project has also influenced the improvements in the GNU family offering like GCC compiler.
- The primary repository is maintained in the File System Hierarchy standard maintained by the Linux community.

Linux Distributions and Licenses

- The best known distribution is from RedHat, Debian and Slackware.
- There are other distributions like SuSE and Caldera and craftworks.
- There are free downloads available. It is advisable to look up the internet for these.

Linux Installation

- Linux can be installed on a *wide range of machines.*
- *Options* for installation
  - ✓ Booting to install using a *floppy disk.*
  - ✓ Using a *hard drive partition* to hold the installation software
  - ✓ Booting to an install and installing software using *FTP or HTTP protocols.*
  - ✓ Booting to an install and installing software from an *NFS-mounted hard drive.*

The Installation Program

Red Hat Linux is installed by booting to the install directory from CD-ROM

Options available at boot screen

<Enter>	- Start the installation using a Graphical interface
Text	- Start the installation using a text interface
nofb	- Start the install using a video frame buffer
expert	
Linux rescue	
Linux dd	

Selecting the Type of Installations

- Workstation
- Server
- Laptop
- Custom
- Upgrade an existing system

Choosing Partition Scheme

- *Auto Partition* - format the hard drive according to the type of selected installation and configures it.
- *Disk Druid* - launch a graphical editor listing the free spaces available.
- *Fdisk* - offers an ability to create nearly 60 different types of partitions.

Firewall Configuration

- Select a firewall configuration
  - ✓ High
  - ✓ Medium
  - ✓ None

Finishing the Installation

- Enter the user-id and password when prompted and reboot the system.
- Boot Loaders
  - ✓ *LILO* (Linux Loader)
  - ✓ *GRUB* (default loader for RedHat)

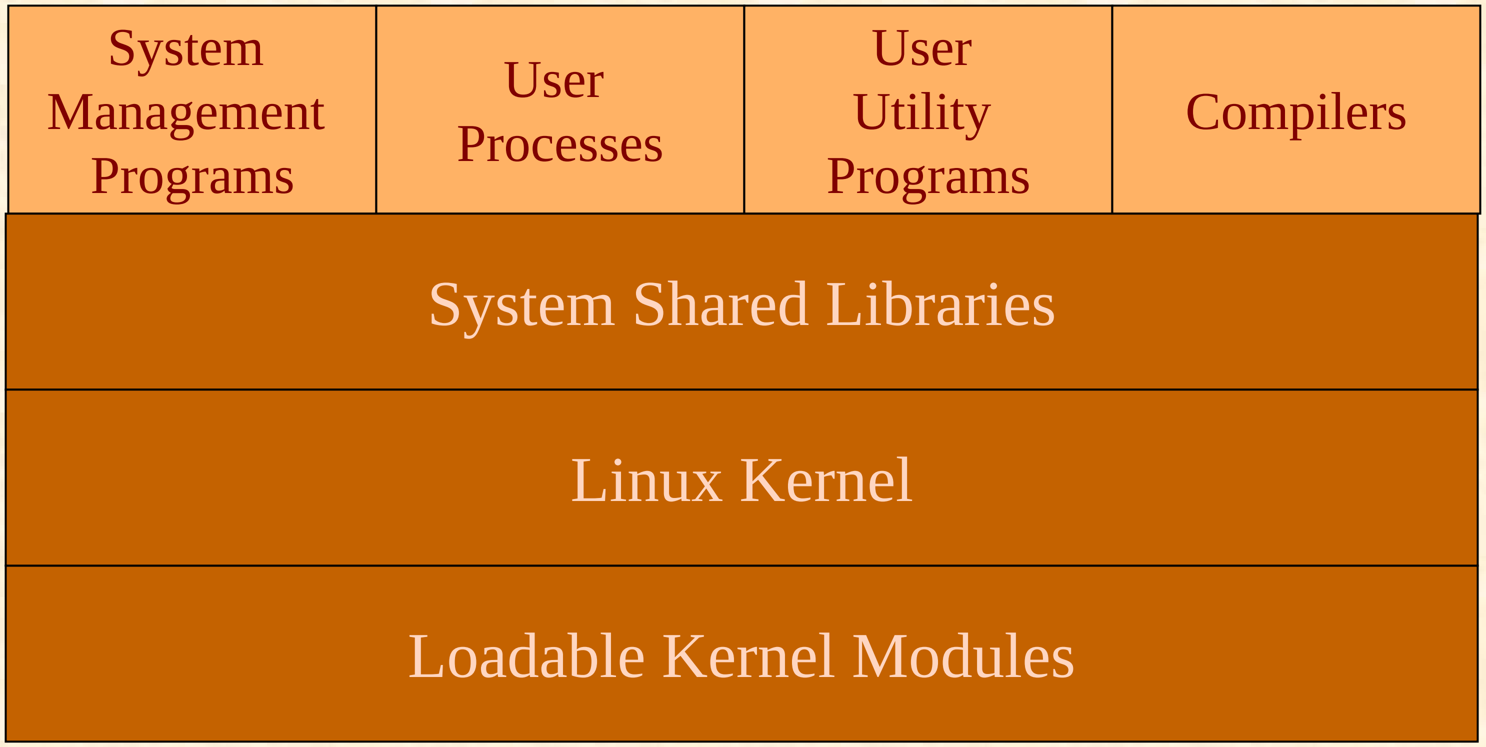
Linux Design Considerations

- Linux is Unix like system: A multi-user, multi-tasking system with its file system as well as networking environment adhering to the Unix semantics.
- From the very beginning Linux has been Posix compliant.
- One of the advantages today: Linux cluster operations in multi-processor environment.

Components of a Linux System

- Like Unix it has three main constituents : kernel, system libraries and utilities.
- Kernel is the core that manages processes and virtual memory.
- System libraries define functions that applications use to seek kernel services without exercising the kernel code privileges.
- The utilities are specialized functions like “sort” or programs, called daemons like login daemons or network connection management daemons.

The Linux Components



The Kernel : Operational Features

- The kernel code executes in privileged mode called kernel mode.
- Any code that does not need to run in privileged mode is put in the system library.
- The interesting thing about Linux kernel is that it has a modular architecture – even with binary codes: Linux kernel can load (and unload) dynamically (at run time) modules just it can load or unload the system library modules.

Linux Kernel : Features

- Kernel code provides for arbitrations and for protected access to HW resources.
- Kernel supports services for the applications through the system libraries. System calls within applications (may be written in C) may also use system library. For instance, the buffered file handling is operated and managed by Linux kernel through system libraries.
- Programs like utilities that are needed to initialize the system and configure network devices are classed as user mode programs and do not run with kernel privileges (unlike in Unix).
- Programs like those that handle login requests are run as system utilities and also do not require kernel privileges (unlike in Unix).

The Kernel Module

- The “loadable” kernel modules execute in the privileged kernel mode – and therefore have the capabilities to communicate with all of HW.
- Linux kernel source code is free. People may develop their own kernel modules. However, this requires recompiling, linking and loading. Such a code can be distributed under GPL.
- More often the modality is: Start with the standard minimal basic kernel module. Then enrich the environment by the addition of customized drivers. This is the route presently most people in the embedded system area are adopting worldwide.

Module Support in Linux Kernel

- The module support in Linux has three main components :
  - ✓ Module management.
  - ✓ Driver registration.
  - ✓ Conflict resolution mechanism.

Module Management - 1

- For new modules this is done at two levels - the management of kernel referenced symbols and the management of the code in kernel memory.
- The Linux kernel maintains a symbol table and symbols defined here can be exported (that is these definitions can be used elsewhere) explicitly. The new module must seek these symbols. In fact this is like having an external definition in C and then getting the definition at the kernel compile time.

Module Management - 2

- The memory management is done in two stages:
  - ✓ First the loader seeks memory allocation from the kernel.
  - ✓ Next the kernel returns the address of the area for loading the new module.
- The linking for symbols is handled by the compiler.
- The module management system also defines all the required communications interfaces for this newly inserted module.
- With this done, processes can request the services (may be of a device driver) from this module.

Driver Registration - 1

- The kernel maintains a dynamic table which gets modified once a new module is added – some times one may wish to delete also.
- In writing these modules care is taken to ensure that initializations and cleaning up operations are defined for the driver.
- A module may register one or more drivers of one or more types of drivers.
- Usually the registration of drivers is maintained in a registration table of the module.

What’s in a Registration Table

- Driver context identification: Character or bulk device or a network drivers.
- File system context: Essentially the routines employed to store files in Linux virtual file system or network file system like NFS.
- Network protocols and packet filtering rules.
- File formats for executable and other files.

Conflict Resolution

- The PC hardware configuration is supported by a large number of chip set configurations and with a large range of drivers for scsi devices, video display devices and adapters, network cards.
- Consequence: Module device drivers vary over a very wide range.
- This necessitates a conflict resolution mechanism to resolve accesses in a variety of conflicting concurrent accesses.

The Conflict Resolution Mechanism

- Helps in preventing modules from having an access conflict with respect to the HW - for example an access to a printer.
- Modules usually identify the HW resources it needs at the time of loading and the kernel makes these available by using a reservation table.
- The kernel usually maintains information on the address to be used for accessing HW - be it DMA channel or an interrupt line. The drivers avail kernel services to access HW resources.

Process Management

- User process as also the kernel processes seek the CPU and other services.
- Usually a fork system call results in creating a new process. System call *execve* results in execution of a newly forked process.
- Processes, have an id (PID) and also have a user id (UID) like in Unix. Linux additionally has a *personality* associated with a process. Personality of a process is used by emulation libraries to be able to cater to a range of implementations.

Process Environment

- Usually a forked process inherits parent’s environment.
- Two vectors define a process: these are argument vector and environment vector.
- The environment vector essentially has a (name, value) value list wherein different environment variable values are specified.
- The argument vector has the command line arguments used by the process.

More on Process Environment

- Usually the environment is inherited – however, upon execution of execve the process body may be redefined with a new set of environment variables. This helps in the customization of a process's operational environment.
- Usually a process also has some indication on its scheduling context.
- Typically a process context includes information on scheduling, accounting, file tables, capability on signal handling and virtual memory context.

Processes and Threads

- In Linux, internally, both processes and threads have the same kind of representation.
- Linux processes and threads are POSIX compliant and are supported by a threads library package which provides for two kinds of threads: user and kernel. User-controlled scheduling can be used for user threads. The kernel threads are scheduled by the kernel.
- While in a single processor environment there can be only one kernel thread scheduled.
- In a multiprocessor environment one can use the kernel supported library and clone system call to have multiple kernel threads created and scheduled.

Scheduling

- In Linux scheduling is required for the user processes and the kernel tasks.
- Kernel tasks may be internal tasks on behalf of the drivers or initiated by user processes requiring specific
- OS services. Examples are: a page fault (induced by a user process) or because some device driver raises an interrupt.
- In Linux, normally, the kernel mode of operation can not be pre-empted. Kernel code runs to completion - unless it results in a page fault, or an interrupt of some kind or kernel code it self calls the scheduler.

More on Scheduling

- Linux is a time sharing system. So a timer interrupt happens and rescheduling may be initiated at that time.
- Linux uses a credit based scheduling algorithm. The process with the highest credits gets scheduled. The credits are revised after every run.
- If all run-able processes exhaust all the credits a priority based fresh credit allocation takes place.
- The crediting system usually gives higher credits to interactive or IO bound processes – as these require immediate responses from a user.
- Linux also implements Unix like nice process characterization.

Memory Management

- The two major components in memory management are:
  - ✓ The page management.
  - ✓ The virtual memory management.
- The main program in memory management is the page allocator which is responsible for both allocation as well as freeing the memory.
- The basic memory allocator uses a buddy heap which allocates a contiguous area of size 2^n > the required memory with minimum n obtained by successive generation of “buddies” of equal size.

Generation of Buddies

- Suppose we need memory of size 1556 words. Starting with a memory size 16K we would proceed as follows:
  1. Create 2 buddies of 8k size.
  2. From one of the 8k buddies create two 4k buddies.
  3. From one of the 4k buddies create two 2k buddies.
  4. Use one of the most recently generated buddies to accommodate the 1556 size memory requirement.

Virtual Memory Management

- The basic idea of a virtual memory system is to expose address space to a process.
- Logically, page aligned contiguous address space defines a region in a memory. These regions of memory are organized to form a binary tree structure for fast access.
- Besides the above logical view the Linux kernel maintains the physical view i.e. maps the hardware page table entries that determine the location of the logical page in the exact location on a disk.

More on Virtual Memory Management

- The process address space may have private or shared pages.
- Changes made to a page require that locality is preserved for a process by maintaining a copy-on-write when the pages are private to the process where as these have to be visible when they are shared.

Life Time of a Virtual Address Space

- A process when created following a fork system call finds its allocation with a new entry in the page table – with inherited entries from the parent.
- For a shared page amongst the processes (like parent and child) a reference count is maintained.
- Linux has a far more efficient page swapping algorithm than Unix – it uses a second chance algorithm to define aging of the pages.
- Frequently used pages get a higher age value and a reduction in usage brings the age closer to zero – finally leading to its exit.

Kernel Virtual Memory

- Kernel also maintains for each process a certain amount of “kernel virtual memory” - the page table entries for these are marked “protected”.
- The kernel virtual memory is split into two regions.
- First there is a static region which has the core of the kernel and page table references for all the normally allocated pages that can not be modified.
- The second region is dynamic - page table entries created here may point any where and can be modified.

Execution and Loading of User Programs

- For a process the execution mode is entered following an exec system call.
- This may result in completely rewriting the previous execution context – this, however, requires that the calling process is entitled an access to the called code.
- Once the check is through the loading of the code is initiated.
- Older versions of Linux used to load binary files in the a.out format. The current version also load binary files in ELF format. The ELF format is flexible from the view point that it permits adding additional information for debugging etc.

Linking in Linux

- A process can be executed when all the needed library routines have also been linked to form an executable module.
- Linux supports dynamic linking. The dynamic linking is achieved in two stages.
- First the linking process down loads a very small statically linked function - whose task is to read the list of library functions which are to be dynamically linked.
- Next the dynamic linking follows - resolving all symbolic references to get a loadable executable.

File Systems in Linux

- Linux retains most fundamentals of the Unix file systems.
- While most Linux systems retain minix file systems as well, the more commonly used file systems used are VFS and ext2FS which stand for virtual file system and extended file systems.
- We shall also examine some details of proc file system and motivation for its presence in Linux file systems.

VFS in Linux

- The Kernel maintains one kind of file system on one disk device as a mounted file system. The file system object helps in organizing a hierarchical self-contained directory structure of individual files.
- For management of files VFS employs an underlying definition for three kinds of objects:

  * Inode object   * File object   * File system object.
- Associated with each type of object is a function table which contains the operations that can be performed.
- The function table maintains the addresses of the operational routines.

More on VFS

- The file objects and inode objects maintain all the access mechanism for each file’s access.
- To access an inode object the process must obtain a pointer to it from the corresponding file object.
- The file object maintains from where a certain file is currently being read or written to ensure sequential IO. File objects usually belong to a single process.
- The inode object maintains such information as the owner, time of file creation and modification.

The ext2fs File System

- This is the most commonly used file system in Linux. In fact, it extends the original minix FS which had several restrictions – such as file name length being limited to 14 characters and the file system size limited to 64K etc.
- The ext2FS permits three levels of indirections to store really large files (as in BSD fast file system).
- Small files and fragments are stored in 1KB (kilo bytes) blocks. It is possible to support 2KB or 4KB blocks sizes. 1KB is the default size.

More on ext2fs File System

- Physical block allocation policy attempts to place logically close blocks are also placed physically close so that IO is expedited.
- This is achieved by having two forms of groups:

  1. block group 2. cylinder group.

- Usually the file allocation is attempted with the block group with the inode of the file in the same block group. Also within a block group physical proximity is attempted.
- As for the cylinder group the distribution depends the way head movement can be optimized.

The Proc File System in Linux

- The proc system is also called the process file system of Linux.
- The proc file system contains a lot of useful information. For instance, /cpu/cpuinfo will typically contain the information like: processor:0, vendorid: Authentic AMD, cpufamily: 5, model name: AMD-K6 etc…
- The proc fs contains the properties of all the processes that are running at any time.

The Type of Information in proc File System

- /proc/PID cmdline: Contains command line args.
- /proc/PID/cwd: Link to working directory.
- /proc/PIDenviron: Values of environment vars.
- /proc/PID/mem: Memory maps to library files and executables.
- /proc/PID/root: Link to root directory for this process.
- /proc/PID/status: Process status in human readable form.

Mapping of inode in proc File System

- Typically inode maintains the information about the files.
- In the case of proc file system it is a special file system for current processes.
- In this context the inode structure needs to keep information about the PID of the process. In fact the 32 bit inode information of identification is split in two parts:
    - ✓ PID a 16 bit information.
    - ✓ 16 bits to identify the type of information being requested about that process.

Input and Output

- Basically we need to understand the way the device drivers are organized in Linux.

Inter - Process Communication

- There is considerable similarity with Unix.
- Signals – parent and child processes may communicate using signals.
- Processes may synchronize using wait.
- Processes may communicate using pipe mechanism.
- Processes may use shared memory mechanism for communication.

Network Structure

- The networking features in Linux are implemented in three layers:

  * Socket I/F * Protocol drivers * Network drivers.

- User applications first I/F is the socket. The socket definition is similar to BSD 4.3 Unix which provides a general purpose interconnection framework.
- The protocol layer supports what is often referred to as protocol stack. The data may come from either an application or from a network driver.
- The protocol layer manages routing, error reporting, reliable retransmission of data.

More on Network Support - 1

- The most important support is the IP suite which guides in routing of packets between hosts. On top of the routing are built higher layers like UDP or TCP.
- The routing is actually done by IP driver.
- The IP driver also helps in disassembly/assembly of the packets.
- The routing is done in two ways:
  - ✓ One uses recent cached routing decision.
  - ✓ The other uses a table, a persistent forwarding base.

More on Network Support - 2

- Generally packets are stored in a buffer and have a tag to identify the protocol that need to be used.
- After the selection of the appropriate protocol the IP driver then hands it over to the network device driver to manage the packet movement.
- The firewall management maintains several chains – with each chain having its own set of rules of filtering the packets.

Security Support

- Linux offers a possibility to have pluggable authentication module (PAM). This offers some flexibility. Any application can now seek service to obtain authentication before the access is permitted – authentication services are made available on demand.
- The password is also encrypted by adding a random (“salt”) value and using a strong one way function.
- The access is offered using the owner, group and world access as in Unix and uses access masks for rwe access.