

Kernel Architecture : UNIX Kernel

- It is responsible for scheduling running of user and other processes.
- It is responsible for allocating memory.
- It is responsible for managing the swapping between memory and disk.
- It is responsible for moving data to and from the peripherals.
- It receives service requests from the processes and honours them.



User Mode and Kernel - 1

At any one time we have *one process engaging the CPU*. This may be a *user process or a system routine* (like *ls, chmod*) that is *providing a service*.

The following three situations result in switching to kernel mode from user mode of operation:

1. The scheduler allocates a user process a slice of time (about 0.1 second) and then system clock interrupts. This entails storage of the currently running process status and selecting another runnable process to execute. This switching is done in kernel mode. A point that ought to be noted is: on being switched the current process's priority is re-evaluated (usually lowered).



User Mode and Kernel - 2

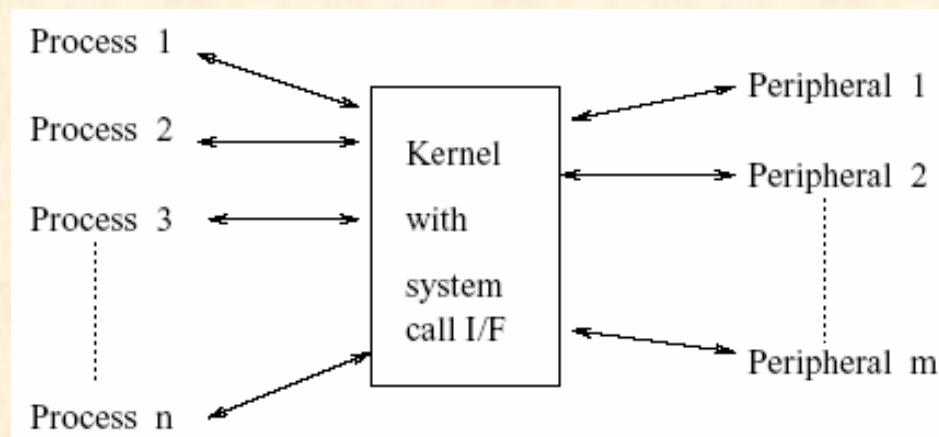
The Unix priorities are ordered in decreasing order as follows:

- *HW errors*
- *Clock interrupt*
- *Disk I/O*
- *Keyboard*
- *SW traps and interrupts*

2. Services are provided by kernel by switching to the kernel mode. So if a user program needs a service (such as print service, or access to another file for some data) the operation switches to the kernel mode. If the user is seeking a peripheral transfer like reading a data from keyboard, the scheduler puts the currently running process to “sleep” mode.

User Mode and Kernel - 3

3. Suppose a user process had sought a data and the peripheral is now ready to provide the data, then the process interrupts. The hardware interrupts are handled by switching to the kernel mode. In other words, the kernel acts as the via-media between all the processes and the hardware as depicted in the below figure



The kernel interface.



System Calls in UNIX

The following are typical system calls in Unix:

Intent of a process The C function call

- Open a file open
- Close a file close
- Perform I/O read/write
- Send a signal kill (actually there are several signals)
- Create a pipe pipe
- Create a socket socket
- Duplicate a process fork
- Overlay a process exec
- Terminate a process exit



Unix Kernel Steps

Typically, Unix kernels execute the following secure seven steps on a system call:

1. Arguments (if present) for the system call are determined.
2. Arguments (if present) for the system call are pushed in a stack.
3. The state of calling process is saved in a user structure.
4. The process switches to kernel mode.
5. The syscall vector is used as an interface to the kernel routine.
6. The kernel initiates the services routine and a return value is obtained from the kernel service routine.
7. The return value is converted to a *c* version (usually an integer or a long integer). The value is returned to process which initiated the call. The system also logs the userid of the process that initiated that call.



An Example of a System Call

Let us trace the sequence when a system call to open a file occurs.

- User process executes a syscall open a file.
- User process links to a c runtime library for open and sets up the needed parameters in registers.
- A SW trap is executed now and the operation switches to the kernel mode.
 - The kernel looks up the syscall vector to call "open"
 - The kernel tables are modified to open the file.
 - Return to the point of call with exit status.
- Return to the user process with value and status.
- The user process may resume now with modified status on file or abort on error with exit status.



Process States in Unix

Unix has the following process state transitions:

idle ----> runnable -----> running.

running ----> sleep (usually when a process seeks an event like I/O, it sleeps awaiting event completion).

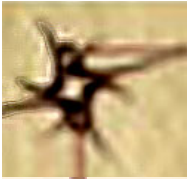
running ----> suspended (suspended on a signal).

running ----> Zombie (process has terminate but has yet to return to its exit code to parent. In unix every process reports its exit status to the parent.)

sleeping ---> runnable

suspended---> runnable

Note that it is the sleep operation which gives the user process an illusion of synchronous operation.



How does Kernel process differ from User Process

1. The first major key difference between the kernel process and other processes lies in the fact that kernel also maintains the needed data-structures on behalf of Unix. Kernel maintains most of this data-structure in the main memory itself. The OS based paging or segmentation cannot swap these data structures in or out of the main memory.
2. Another way the kernel differs from the user processes is that it can access the scheduler. Kernel also maintains a disk cache, basically a buffer, which is synchronised ever so often (usually every 30 seconds) to maintain disk file consistency. During this period all the other processes except kernel are suspended.
3. Finally, kernel can also issue signals to kill any process (like a process parent can send a signal to child to abort). Also, no other process can abort kernel.



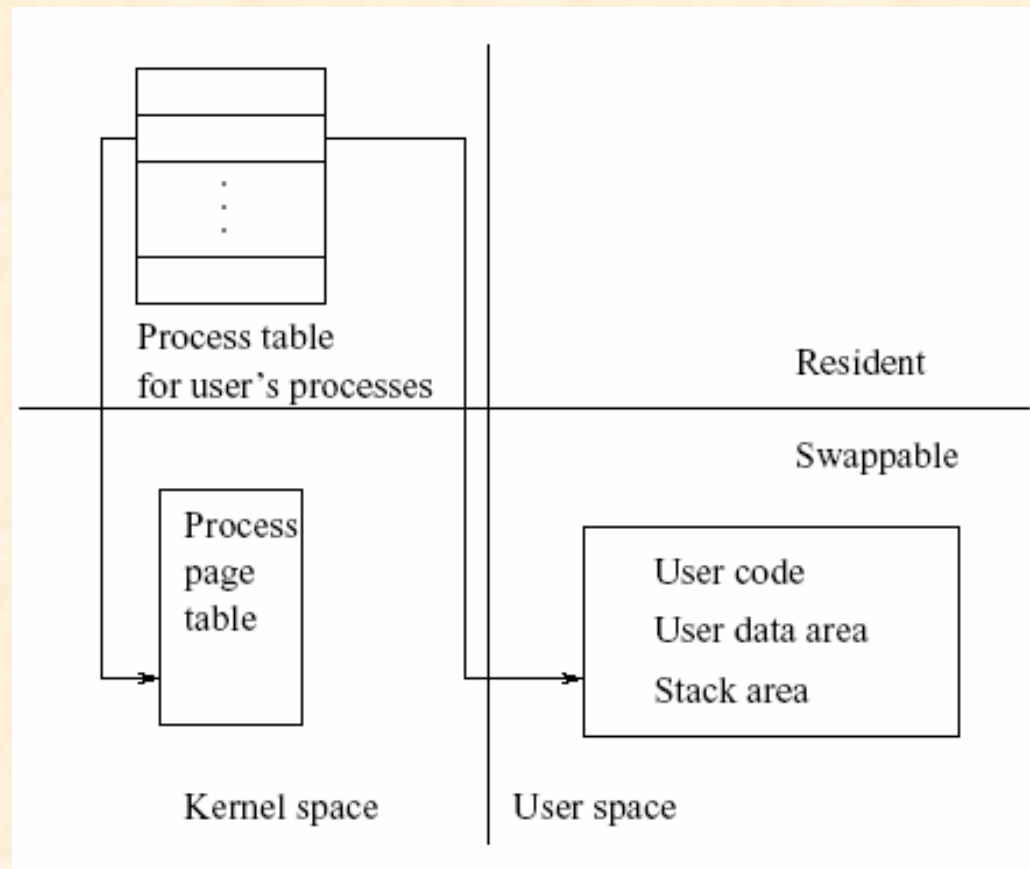
Page Table Format

A fundamental data structure in main memory is page table which maps pages in virtual address space to the main memory. Typically, a page table entry may have the following information.


1. The page mapping as a page frame number, i.e. which disk area it mirrors.
2. The date page was created.
3. Page protection bit for read/write protections.
4. Bits to indicate if the page was modified following the last read.
5. The current page address validity (vis-a-vis the disk).

Mapping of Data and Code Area for a User - 1

The typical User – Kernel Mode is shown in the following figure



User and Kernel Space.



Mapping of Data and Code Area for a User - 2

The memory is often divided into four quadrants as shown in previous figure. The vertical line shows division between the user and the kernel space. The horizontal line shows the swappable and memory resident division.

User processes use the following areas :

- Code area: Contains the executable code.
- Data area: Contains the static data used by the process.
- Stack area: Usually contains temporary storages needed by the process.
- User area : Stores the housekeeping data.
- Page tables : Used for memory management and accessed by kernel.



Region Tables in UNIX

A region table stores the following information.

- Pointers to i-nodes of files in the region.
- The type of region (the four kinds of files in Unix).
- Region size.
- Pointers to page tables that store the region.
- Bit indicating if the region is locked.
- The process numbers currently accessing the region.



Scheduler - 1

Most Unix schedulers follow the rules given below for scheduling:

1. Usually a scheduler reevaluates the process priorities at 1 second interval. The system maintains queues for each priority level.
2. Every tenth of a second the scheduler selects the topmost process in the runnable queue with the highest priority.
3. If a process is runnable at the end of its allocated time, it joins the tail of the queue maintained for its priority level.
4. If a process is put to sleep awaiting an event, then the scheduler allocates the processor to another process.
5. If a process awaiting an event returns from a system call within its allocated time interval but there is a runnable process with a higher priority then the process is interrupted and higher priority, process is allocated the CPU.

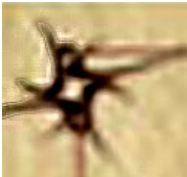


Scheduler - 2

6. Periodic clock interrupts occur at the rate of 100 interruptions per second. The clock is updated for a tick and process priority of a running process is decremented after a count of 4 ticks. The priority is calculated as follows:

$$\text{priority} = (\text{CPU quantum used recently}) / (\text{a constant}) + (\text{base priority}) + (\text{the nice setting}).$$

Usually the priority diminishes as the CPU quantum rises during the window of time allocated. As a consequence compute intensive processes are penalised and processes with I/O intensive activity enjoy higher priorities.



Tour Of Linux Kernel - 1

We shall briefly explore Linux as an example. Later we will describe in greater detail. For now let us look at the major subsystems in Linux environment.

➤ *User Applications* - the set of applications in use on a particular Linux system will be different depending on what the computer system is used for, but typical examples include a *word-processing application* and a *web-browser*.

➤ *O/S Services* - these are services that are typically considered part of the operating system (a *windowing system, command shell, etc.*); also, the *programming interface to the kernel (compiler tool and library)* is included in this subsystem.



Tour Of Linux Kernel - 2

- *Linux Kernel* - this is the main area of interest in this paper; the kernel *abstracts and mediates* access to the hardware resources, including the CPU.
- *Hardware Controllers* - this subsystem is comprised of all the possible *physical devices* in a Linux installation; for example, the CPU, memory hardware, hard disks, and network hardware are all members of this subsystem.



Tour Of Linux Kernel - 3

The Linux kernel is composed of five main subsystems:

The *Process Scheduler (SCHED)* is responsible for controlling process access to the CPU. The scheduler enforces a policy that ensures that *processes will have fair access to the CPU*, while ensuring that necessary hardware actions are performed by the kernel on time.

sourcefiles:

(/usr/src/linux/sched.c)

(header files in /usr/src/include/linux/sched.h)



Tour Of Linux Kernel - 4

The *Memory Manager (MM)* permits multiple process to securely share the machine's main memory system.

In addition, the memory manager *supports virtual memory* that allows Linux to support processes that *use more memory than is available in the system.*

Unused memory is *swapped out to persistent storage* using the file system then *swapped back in* when it is needed.

sources:

(/usr/src/linux/mm)

(header files in /usr/src/include/linux/ and /usr/src/include/asm)



Tour Of Linux Kernel - 5

The *Virtual File System (VFS)* abstracts the details of the variety of hardware devices by presenting a *common file interface to all devices*. In addition, the VFS supports *several file system formats* that are compatible with other operating systems.

(/usr/src/linux/fs)

(header files in /usr/src/include/linux/ and /usr/src/include/asm)



Tour Of Linux Kernel - 6

The *Network Interface (NET)* provides access to several *networking standards* and a variety of *network hardware*.

(/usr/src/linux/net)

*(header files in /usr/src/include/linux/ ,
/usr/src/include/net and /usr/src/include/asm)*



Tour Of Linux Kernel - 7

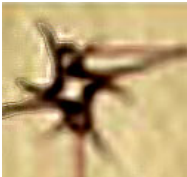
The *Inter-Process Communication (IPC)* subsystem supports several mechanisms for *process-to-process communication* on a single Linux system.

(*/usr/src/linux/ipc*)

(*header files in /usr/src/include/linux/ , /usr/src/include/asm and /usr/src/linux/ipc*)

Many of these subsystems specially *VFS* and *NET* require the help of *device drivers* to talk to the underlying hardware. The code for this is present in

(*/usr/src/linux/drivers*)



Tour Of Linux Kernel - 8

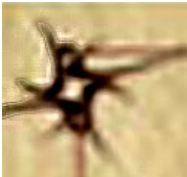
In addition, *Linux isolates the architecture dependent code.*
Linux source code includes two architecture dependent directories:

/usr/src/linux/arch and /usr/src/linux/include

arch:i386 s390

include:asm asm-i386 asm-s390

* Note: We will study the IPC in Module 7 in detail.
Also we shall study more about Linux in Module 19.



Tour Of Linux Kernel - 9

For example:

The *schedule()* function invokes the *switch_to()* Assembly language function to perform process switching.

The code for *switch_to()* is stored in the *include/asm/system.h* file so depending on the target system ,the asm symbolic link is set to *asm-i386* or *asm-s390* etc.

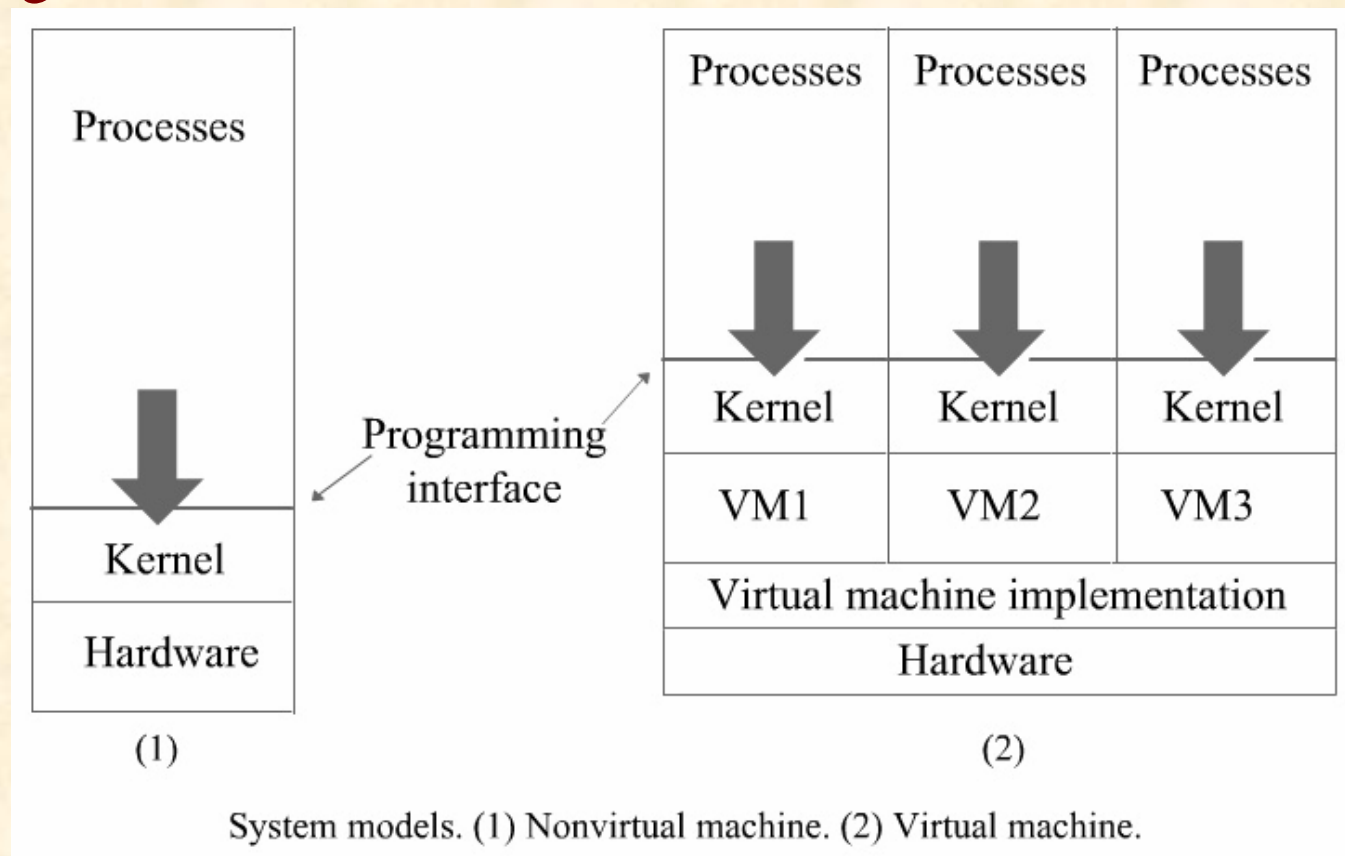


Virtual Machines Concept - 1

- Time shared systems gives an illusion to a user that the machine is to himself.
- This is achieved by CPU scheduling.
- With clever scheduling the user is offered a virtual machine.
- The user may, for instance output on a virtual output device by spooling.

Virtual Machines Concept - 2

- We can now extend the notion to offer each user an illusion of each one having *one virtual machine* - a replica of the original machine as far as services are concerned





Virtual Machines Concept - 3

- In fact this concept has been extended further by creating a software layer which can make one machine offer another machine's operating environment.
- For instance, on sun machine which uses a sparc processor one can offer an intel machine behavior and then create Microsoft's operational environment.