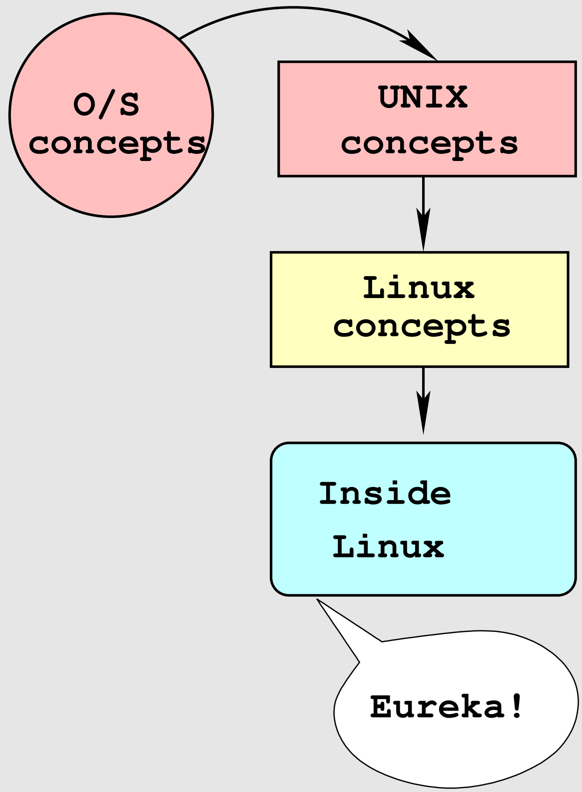
Operating Systems Concepts and Linux

Prof. H.B.Dave, eInfochips

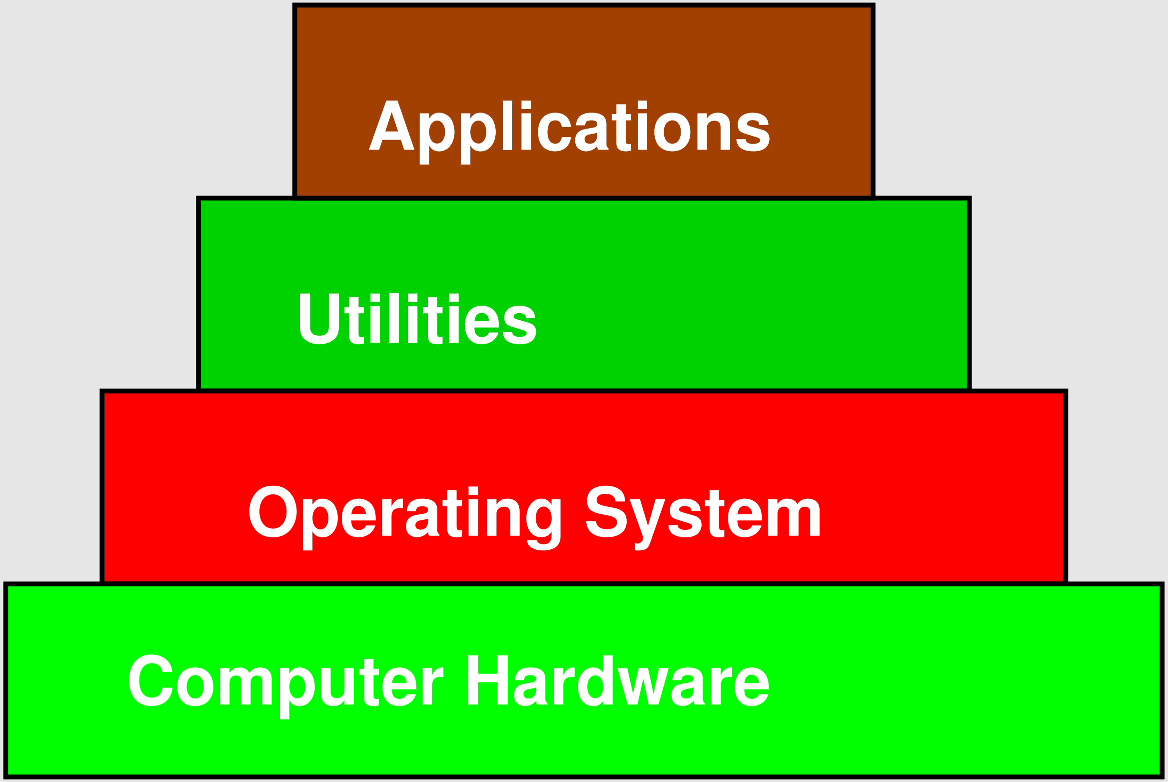
June 2013

O/S Concepts and Linux: Study Plan



Contents

Where does an Operating System stand



Overall characteristics as an O/S

- What is an O/S?
- Linux as an O/S
- Processes and Files
- Permission system

What is an O/S?

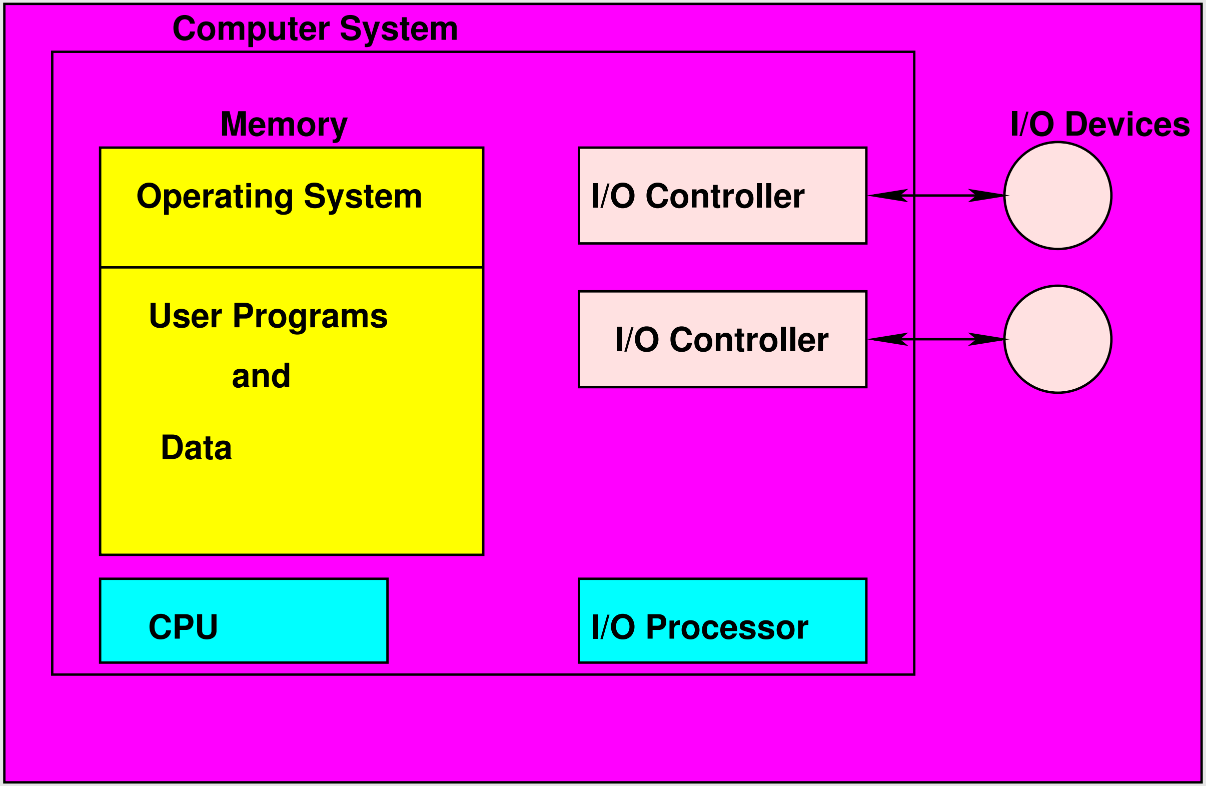
An Operating System is:

- An Interface between a user and Hardware
- A Resource Manager
- An Extended Machine
- Set of Concurrent processes

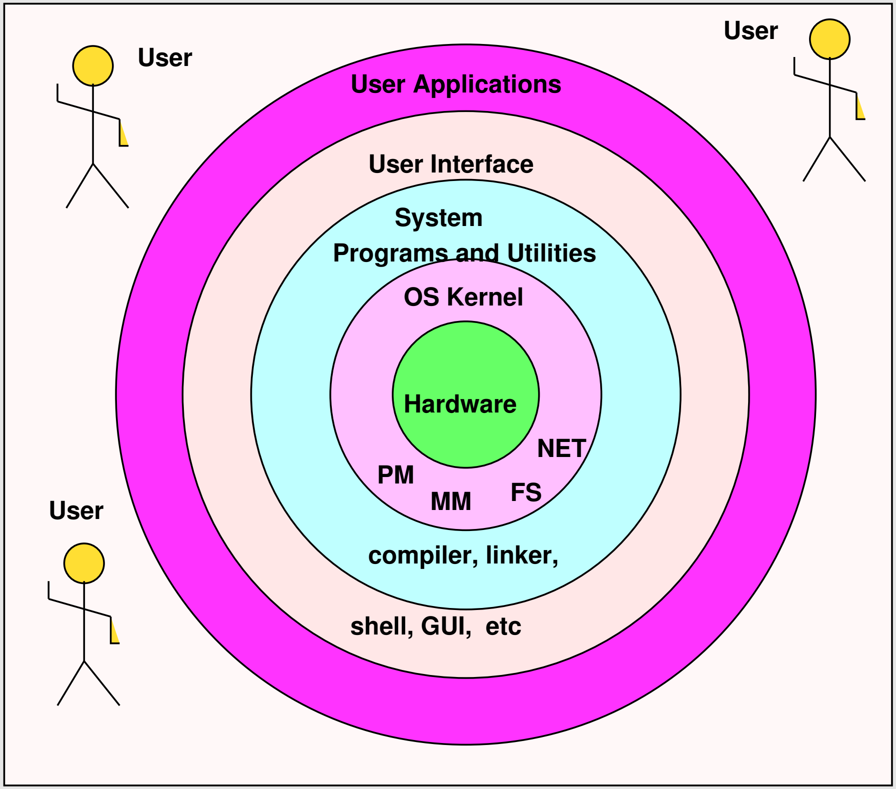
Two Fundamental concepts in O/S:

- Processes: operates on information, various definitions
- File: stores information, in Unix/Linux everything is a File, (with one exception - network devices are not files)

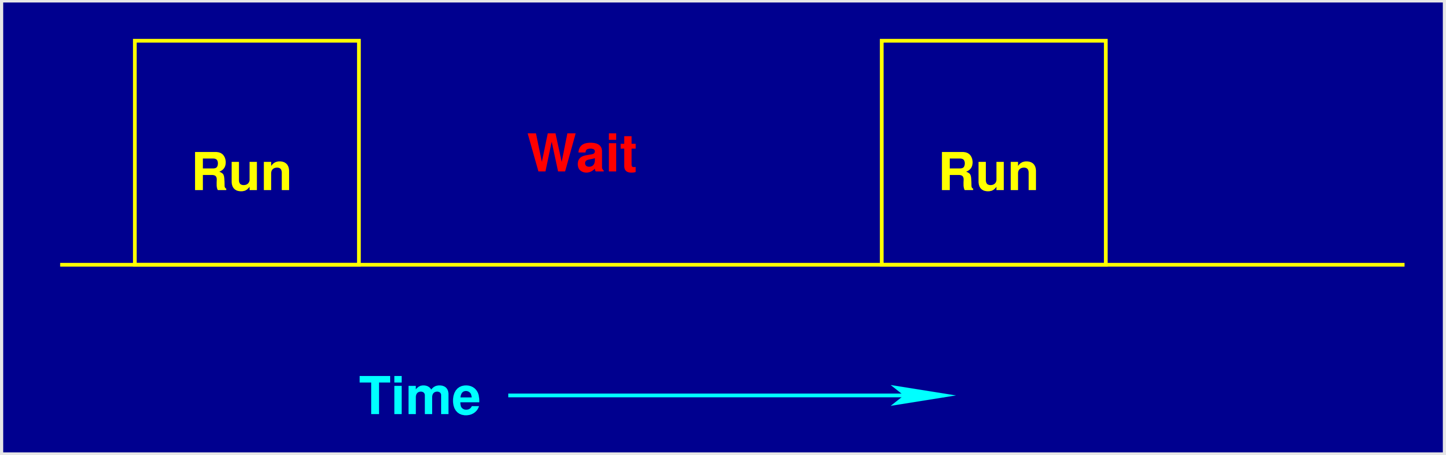
Some Resources handled by an Operating System



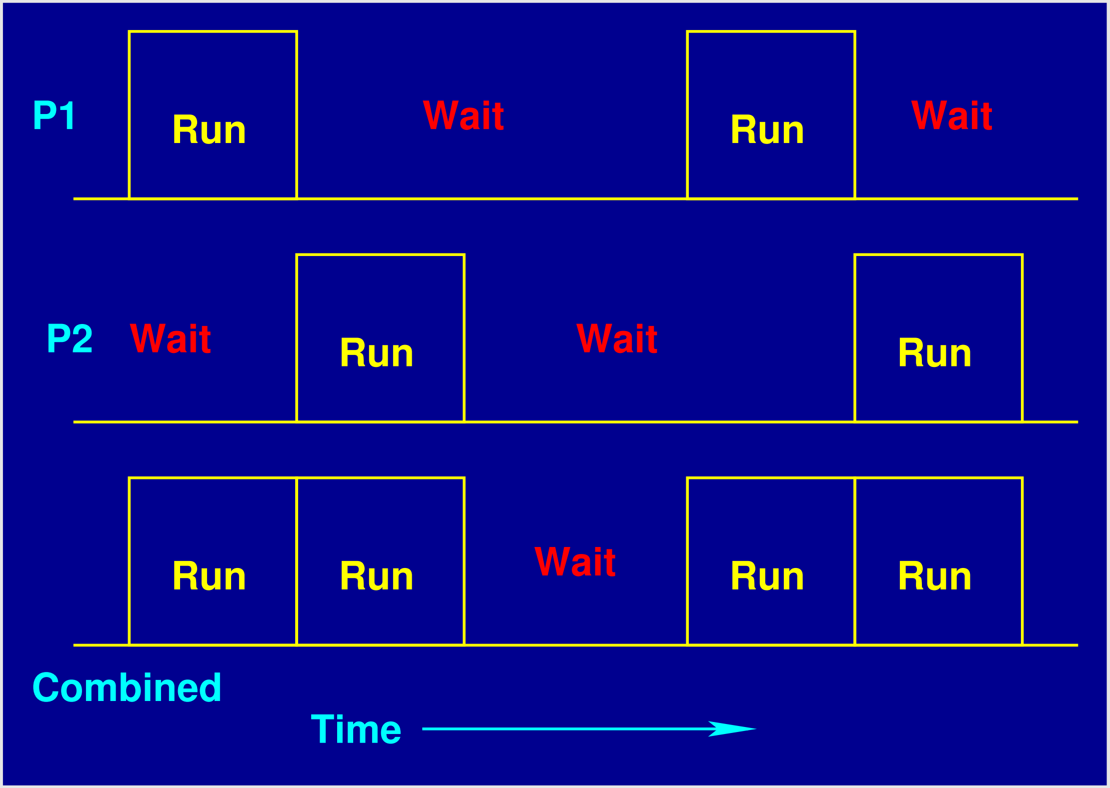
Operating System as an Extended machine



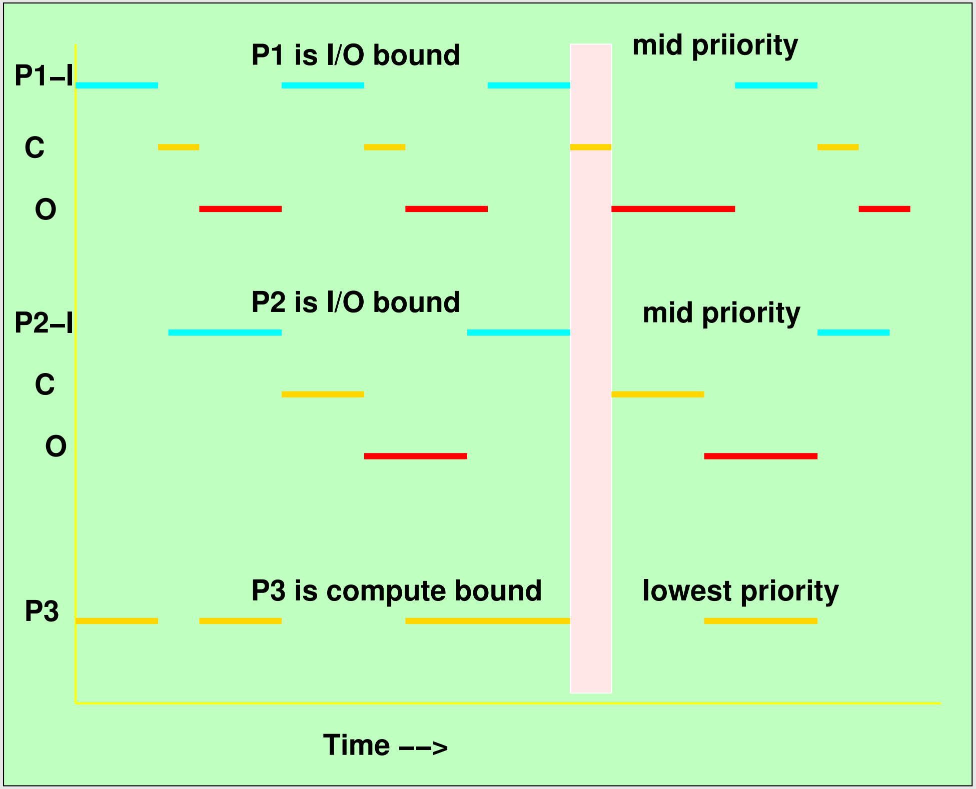
Need for multi-tasking – single task running



Need for multi-tasking – several tasks running



Need for multi-tasking – resource utilization



Linux as an O/S

- Multi-tasking
- Multi-user
- Real-time versions
- O/S kernel exensible via modules
- Permission system
- basic two levels of users - super and normal
- user groups
- all information handeling artifacts are *files*
- processes and threads

- a large set of utilities
- VFS allows handeling of various FS
- CLI shell and GUI
- Built-in TCP/IP stack
- GUI via X11

What Services are expected of a Kernel?

- Controlling execution of processes – *creation, suspension, termination* and *inter-process communication*
- Scheduling Processes – share one or more CPU/s in time-shared manner
- Allocating Main Memory – *private memory, protection from interference, memory swapping in/out*
- Allocating Secondary Storage – *file-system, access control*
- Allowing the processes a controlled access to Peripheral devices – *disks, terminals, pronters, network*

What Assumptions are made about Hardware?

- CPU provides *execution modes* – user mode and kernel mode
- *priviledged operations* possible only in kernel mode
- Interrupt and Exceptions – levels of interrupts
- Memory management and protection
- DMA
- Virtual Memory to provide linear address space, (compiler, loader)

Note that the kernel runs on behalf of a process, it is not a set of separate processes running in parallel with the user processes.

Processes and Files

Two basic entities:

- process
- file

Process: a program in execution:

- allocated memory
- scheduling priority
- usually 3 files opened - STDIN, STDOUT, STDERR
- competes for resources - physical memory, CPU cycles, disk access, file access, etc.

File:

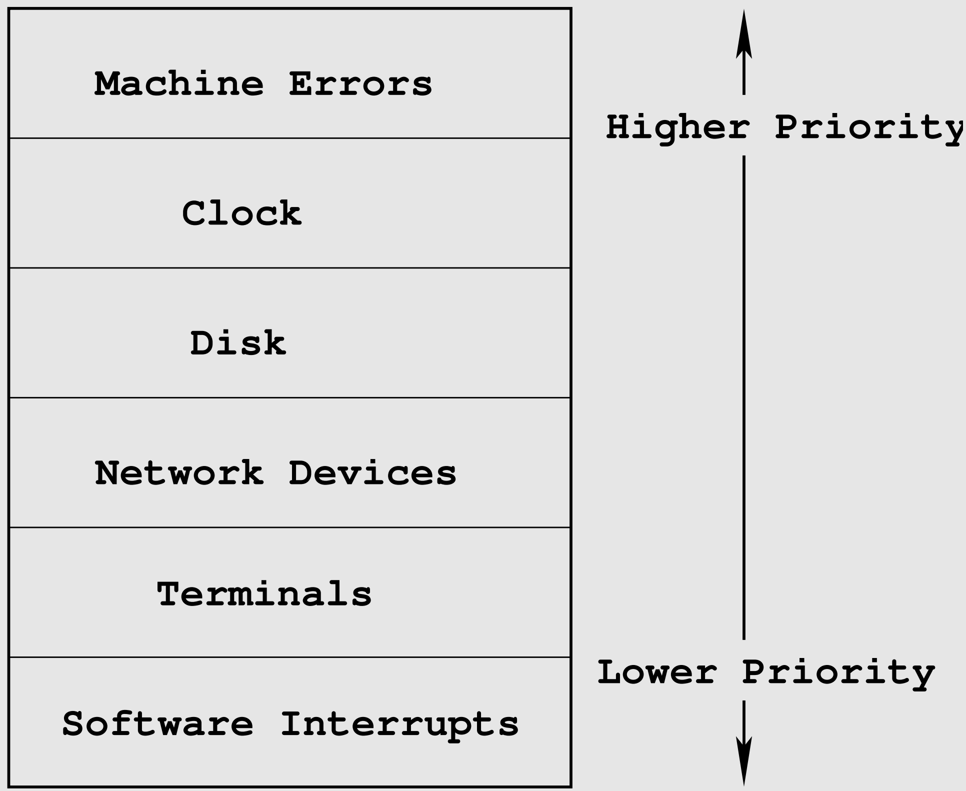
- any source or sink of data (even memory)
- owner, permissions
- time stamps
- internally - *inode* = a token number and data structure
- multiple names (links) – hard and soft links
- basic set of operations – open, close, read, write, seek, ...
- Virtual File System (VFS)

Permission system

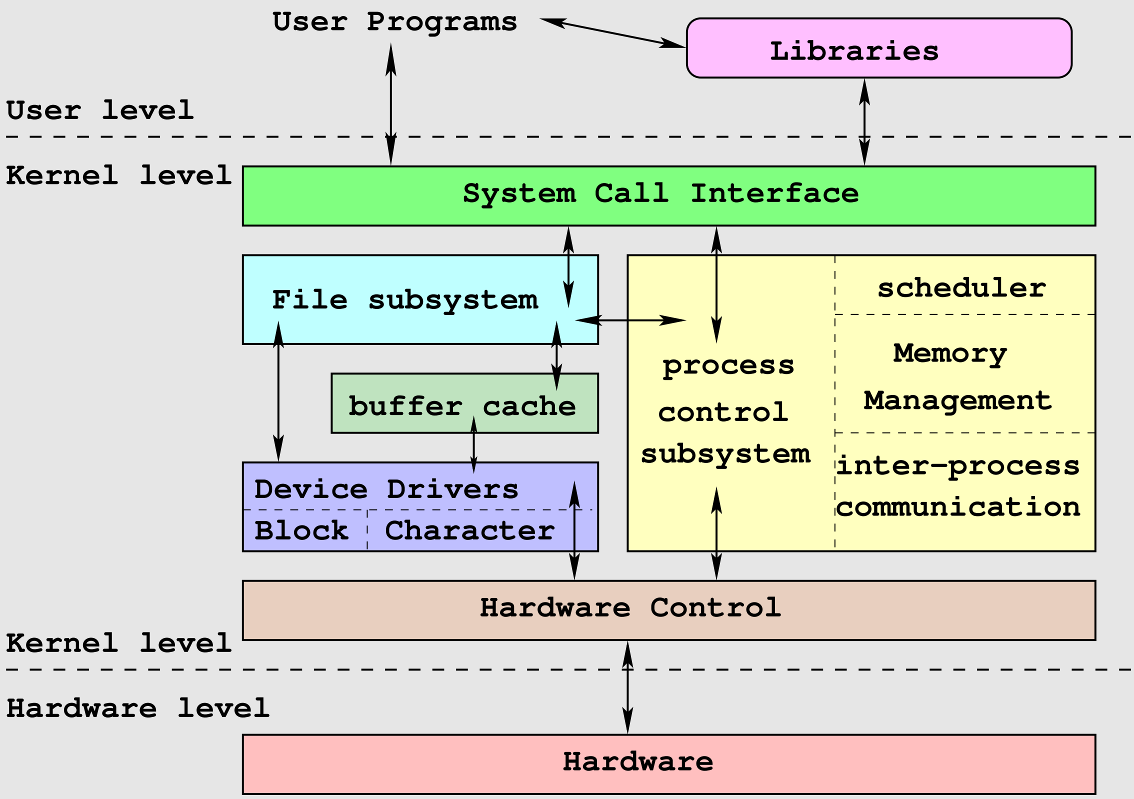
Basic security depends upon permissions.

- three sets of users – owner, group/s, world
- permissions – read, write and execute
- owner: generally the user who created the file
- displayed by `ls -l` as:
  `-rwxr-xr-x 1 hbd users 6588 2011-05-30 09:12 factorial`
-
$$\frac{rwx}{owner}\ \frac{r-x}{group}\ \frac{r-x}{world}$$
- To see permissions of a process VM:
  `less /proc/<pid>/maps`

Typical Interrupt Levels



Block Diagram of a Unix-like Kernel



Linux - user view point

- man-pages, HOWTOs, kernel docs
- most-used basic commands
- pipelining and redirection
- utilities

man-pages, HOWTOs, Kernel Docs

A lot of on-line help.

man-pages:

- On-line Manual pages
- Organized by chapters
- man1 – CLI (via shell) commands
- man2 – C library: system programmers functions
- man3 – C library: application programmers functions
- e.g. `man ls`

HOWTOs: How to do something or implement something, usually experience based information.

Kernel Documentation: `/usr/src/linux-x.x.x/Documentation`

most-used basic commands

- System Administration:
  - top – dynamic system snap-shots
  - mount, umount – introduction of volumes
  - dd – low-level disk to disk copy
  - ldd – list shared libraries needed by an executable
  - important files: /etc/hosts, /etc/fstab, /etc/inittab, /etc/rc.d/*

- System Information:
  - pwd – present working directory
  - ls – list files
  - ps – list processes
  - df – list disk free area
  - du – display amount of disk utilized
  - apropos – search for all commands and functions related to a key word
  - locate – fast ``find'' by a data-base

- File manipulations
  - ln – set-up link (additional name) to a file
  - cp – copy file/s
  - mv – move/rename file/s
  - rm – delete file/s (can not be undeleted!)
  - mkdir – make a new directory
  - touch – change the time-stamps of a file

- basic editing, content search
  - ed – line-wise editor, very fast but primitive
  - grep – Regular Expression search and print
  - file – intelligent guess about file type (irrespective of file-name extention)

pipelining and redirection

Basic concept: build an application by combining well-tested, small, general-purpose utilities.

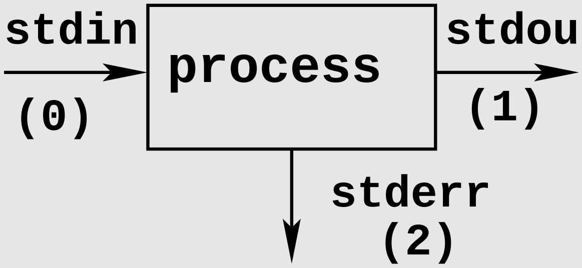


Figure 1 standard files

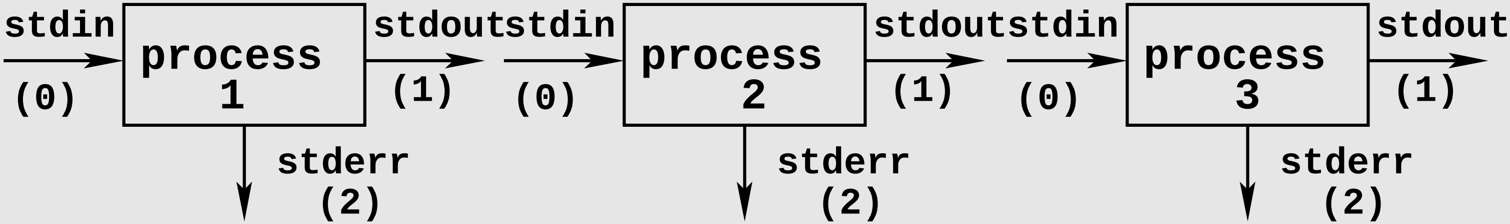


Figure 2 pipeline

e.g. `ls *.c | grep my`

Redirection:

e.g. `ls *.c | grep my > my_c_source.lst`

`myprog 2>myprog.err >myprog.out`

Development utilities

Also known as bin-utilities.

- **ar** Create, modify, and extract from archives
- **nm** List symbols from object files
- **objcopy** Copy and translate object files
- **objdump** Display information from object files
- **ranlib** Generate index to archive contents
- **readelf** Display the contents of ELF format files.
- **size List** file section sizes and total size
- **strings** List printable strings from files

- **strip** Discard symbols
- **c++filt** Demangle encoded C++ symbols (on MS-DOS, this program is named cxxfilt)
- **addr2line** Convert addresses into file names and line numbers

Program development and utilities

- Editors: `vi`, `joe`, `pine`, `emacs`
- Compiler, debugger: `gcc`, `gdb`
- debugging: ldd, strace

Basic System Administration

- File permissions:
  `chmod, chown, chgrp`
- Mount, unmount:
  `mount, umount, remount`
- user admin:
  `adduser`

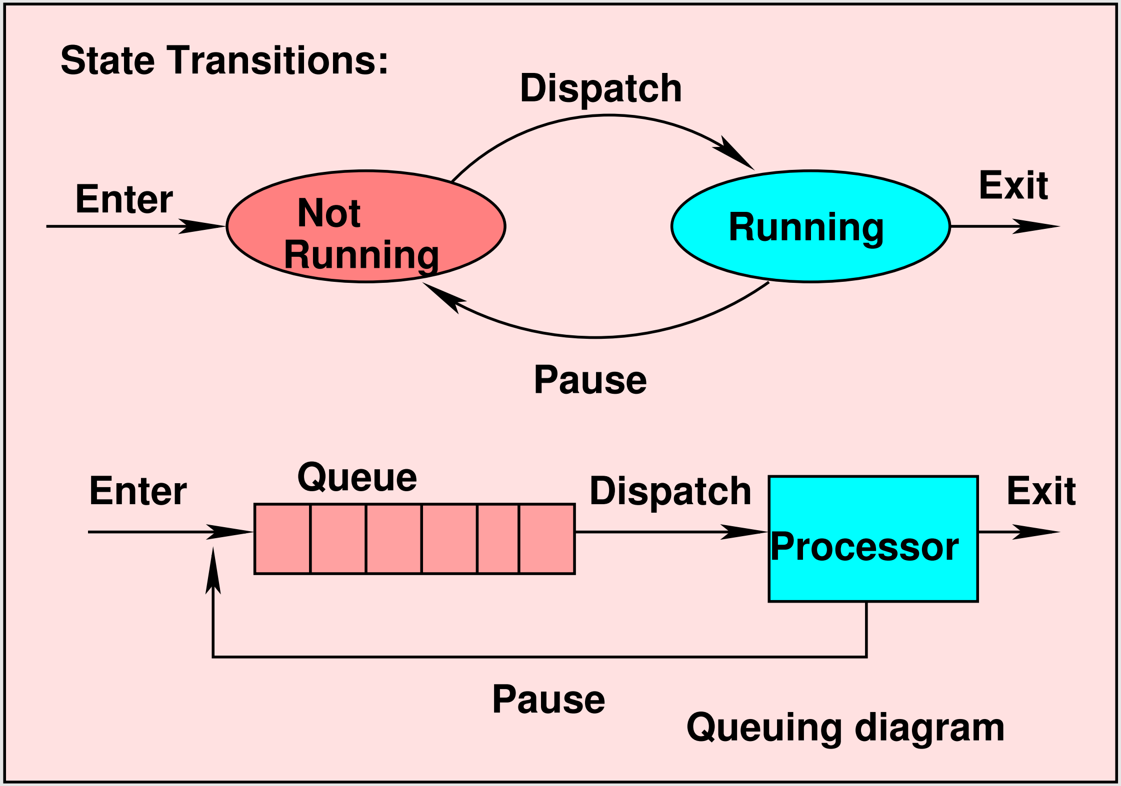
Process

- Is a running program
- Is an active entity
- Has context and state
- Is sequentially executed – a single instruction is executed on behalf of a process at any time
- Also called: Job on batch systems, Task on time-sharing systems

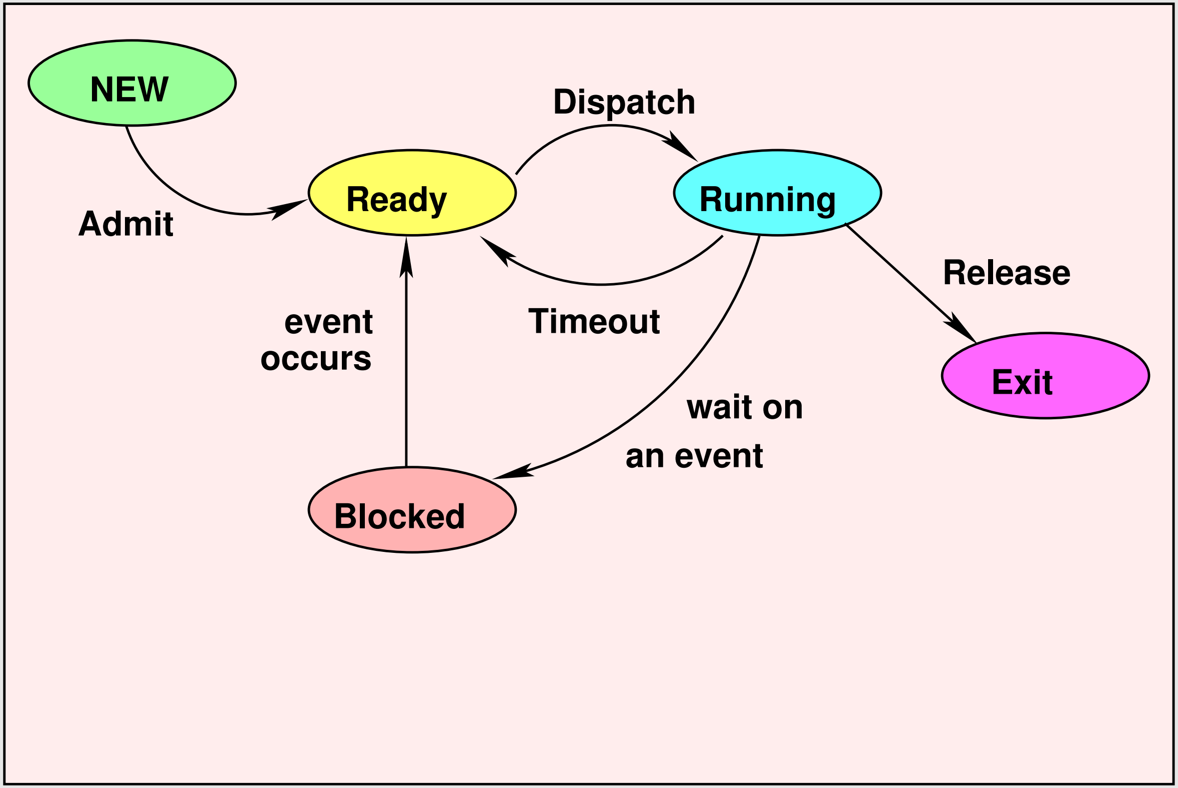
Process States

- Running: process instructions are being executed
- Waiting: process is waiting for some event (e.g. I/O completion)
- Ready: process is ready for execution, but must waiting for a processor to become available

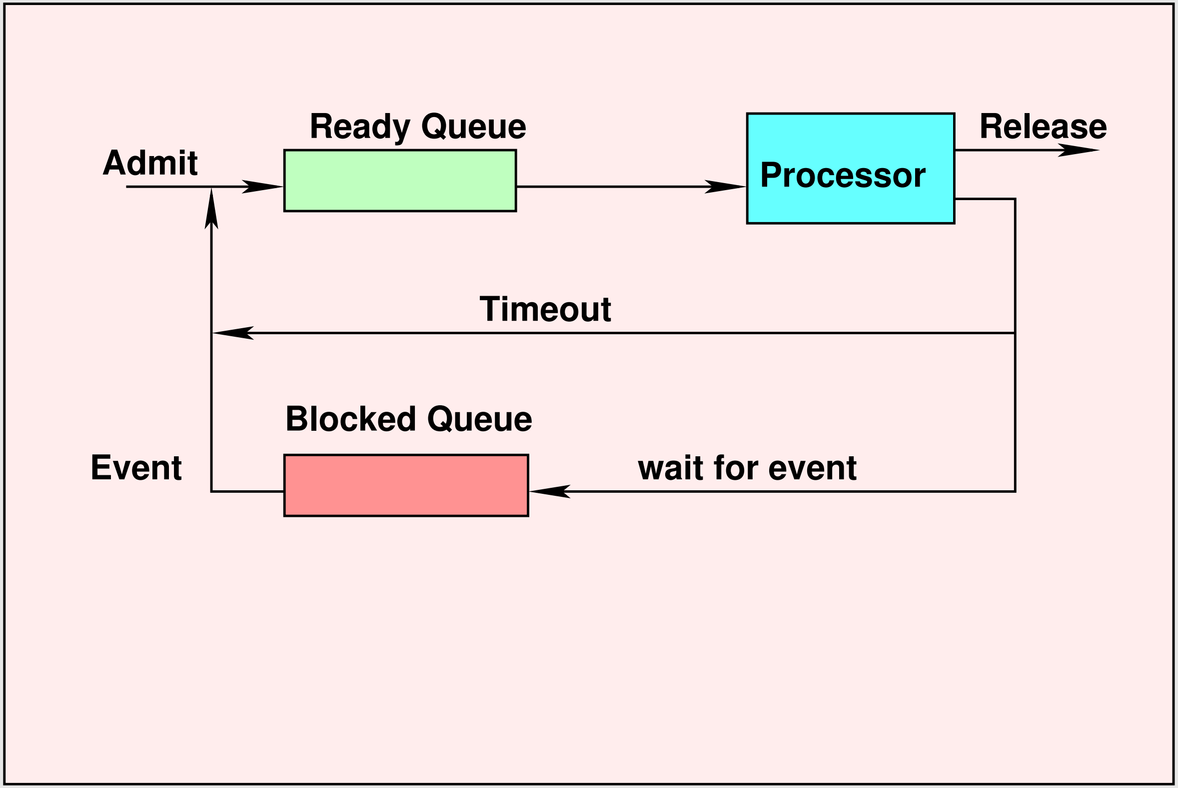
Two State and Queuing models of a Process



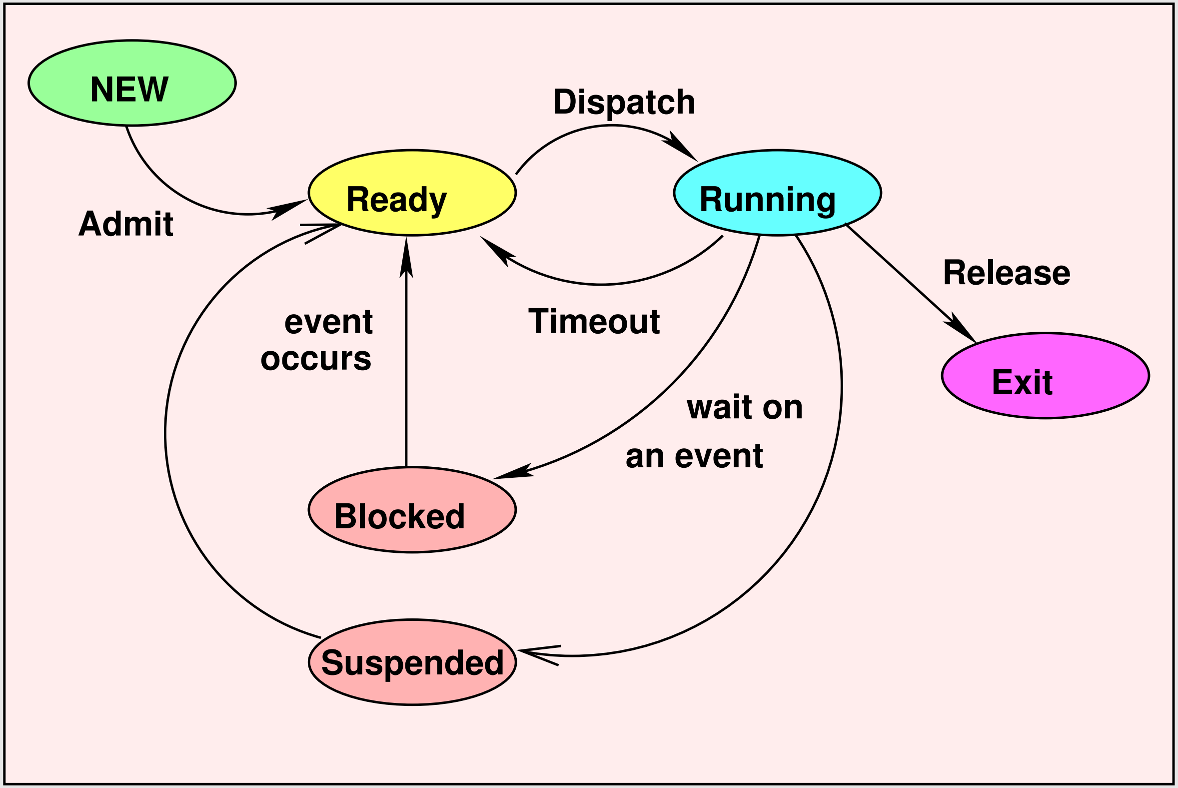
Five States model of a Process



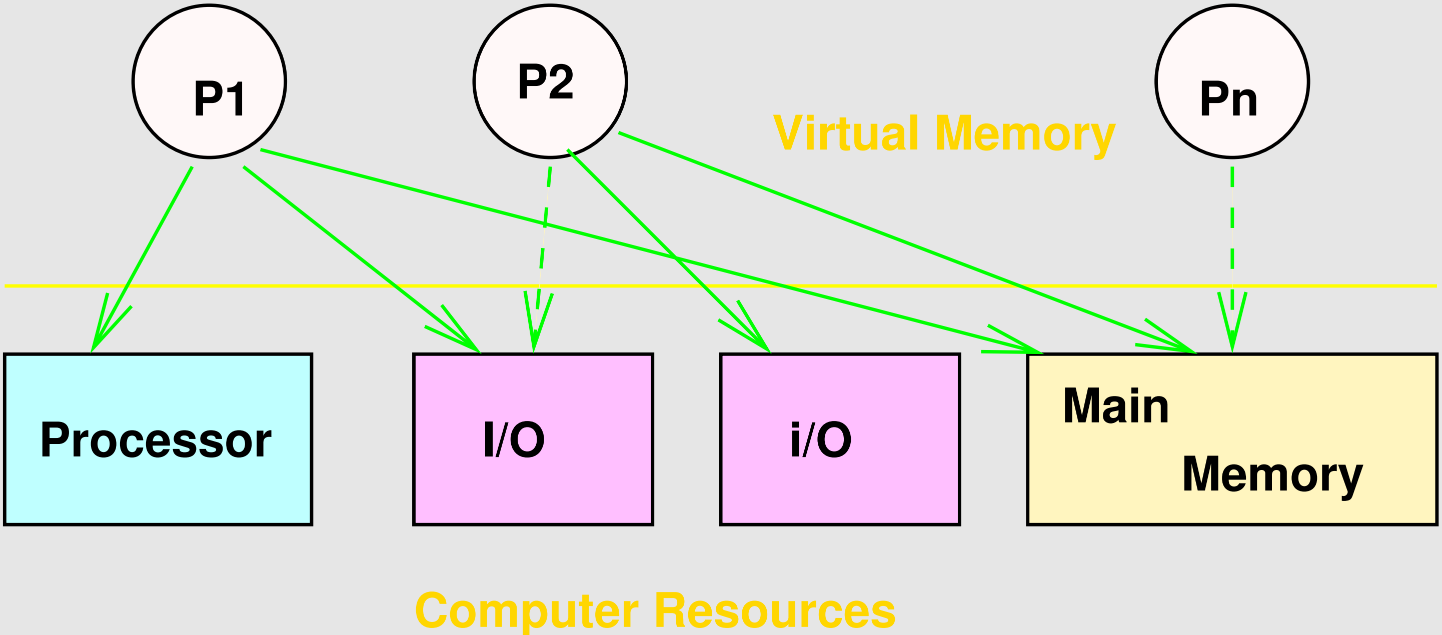
Q model for Five States model of a Process



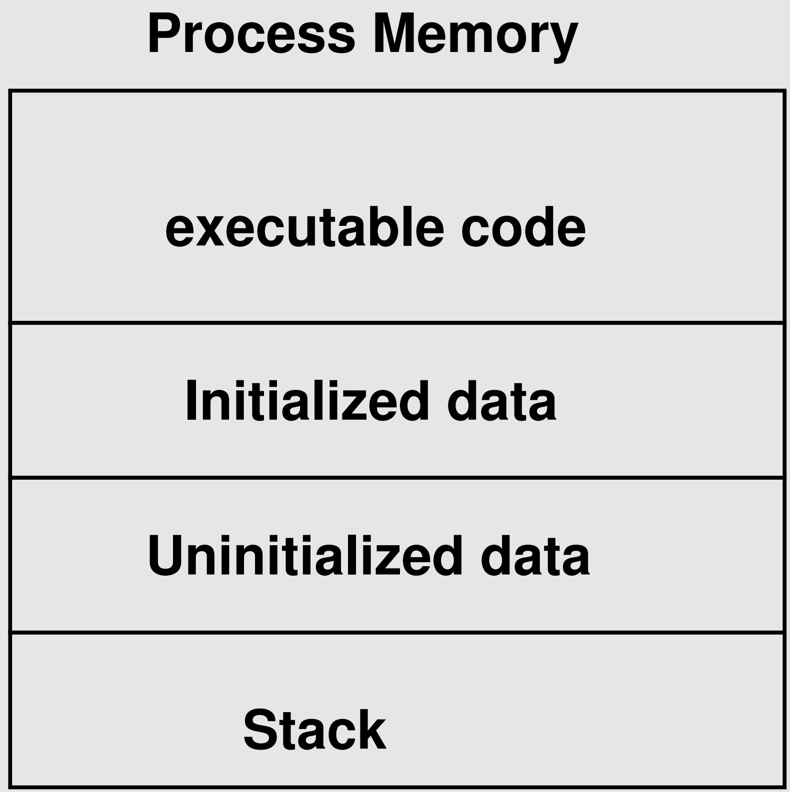
Six States model of a Process



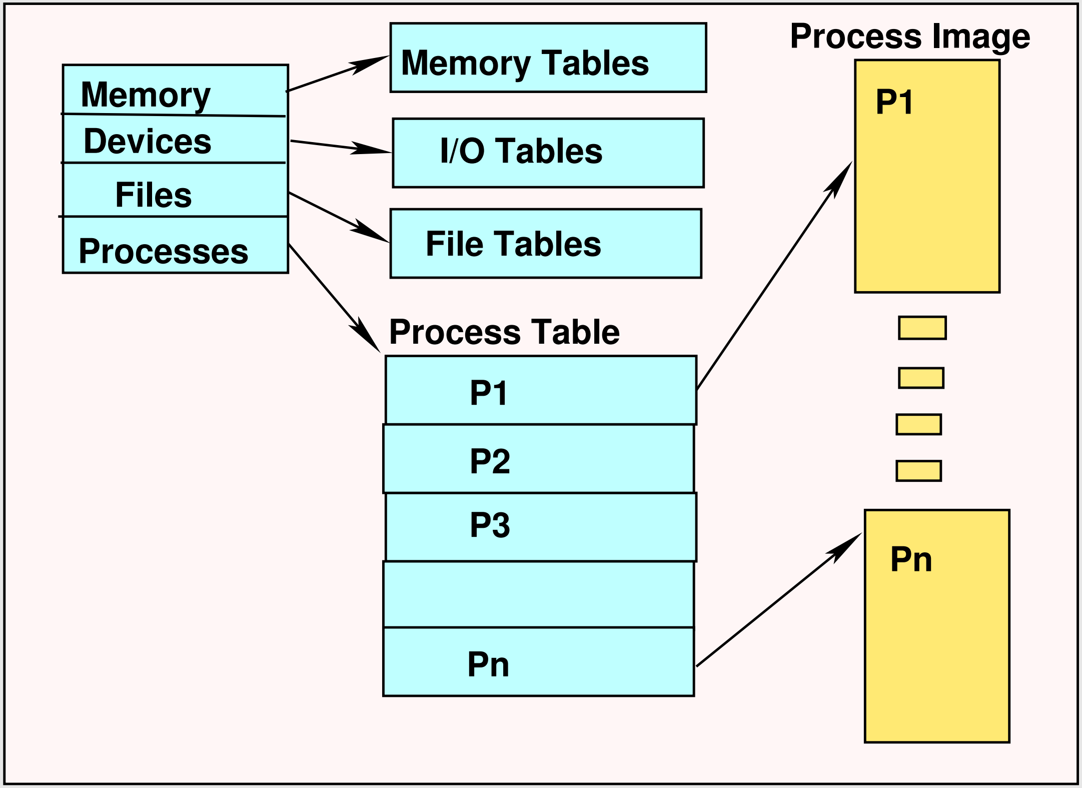
Process Resource Sharing



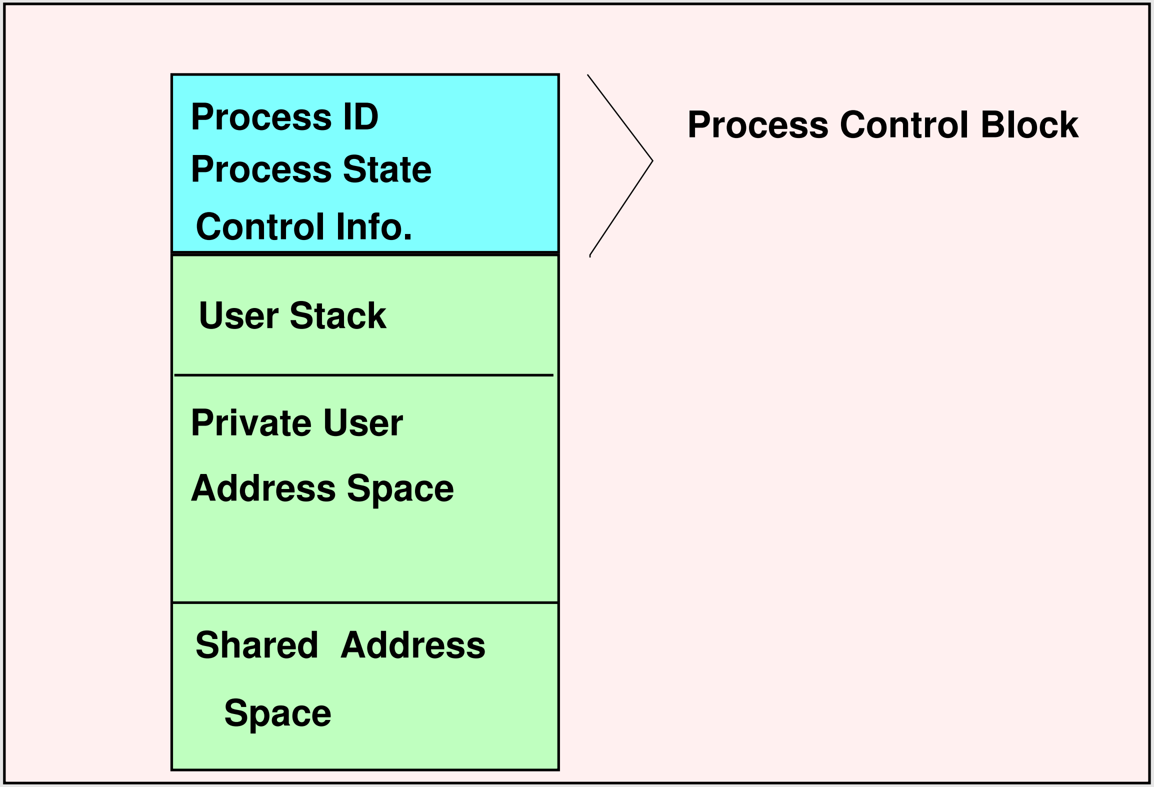
Logical Process Memory Layout



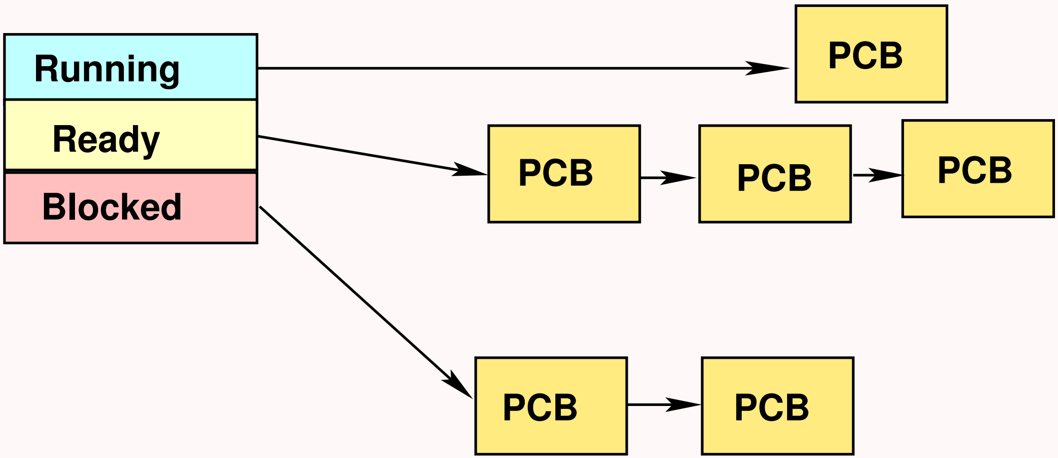
Process Management



PCB and process memory

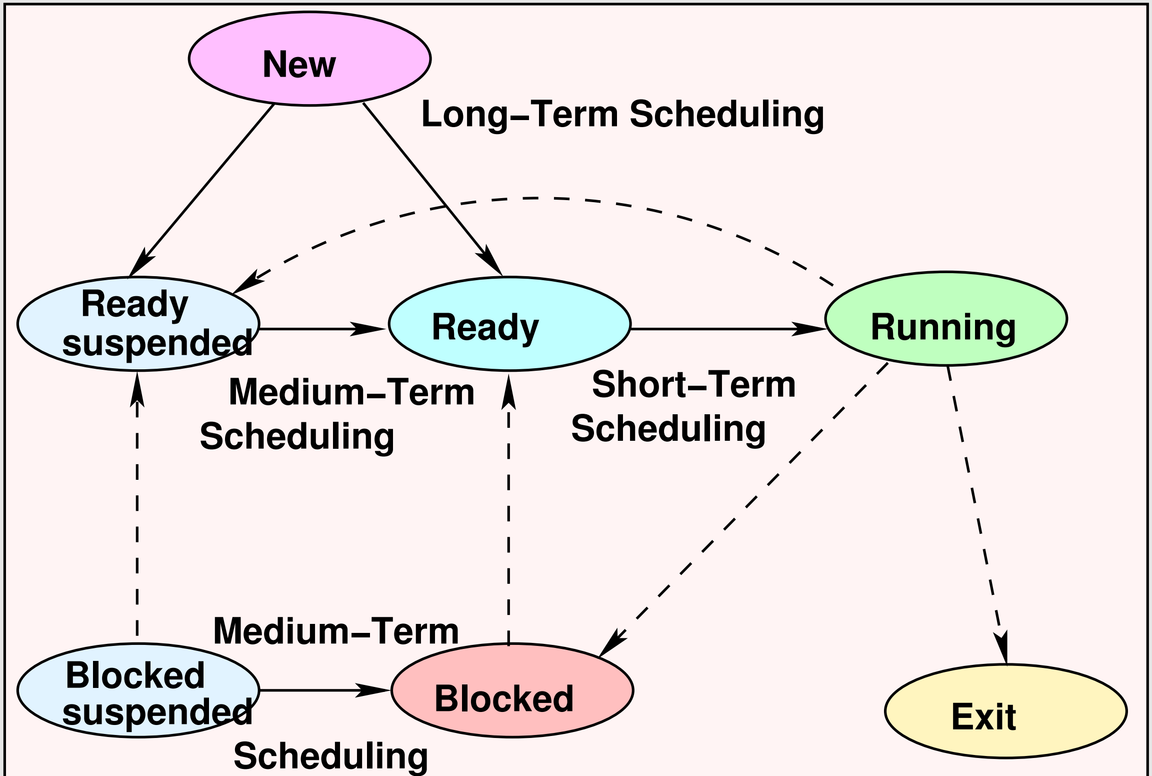


Process list structure

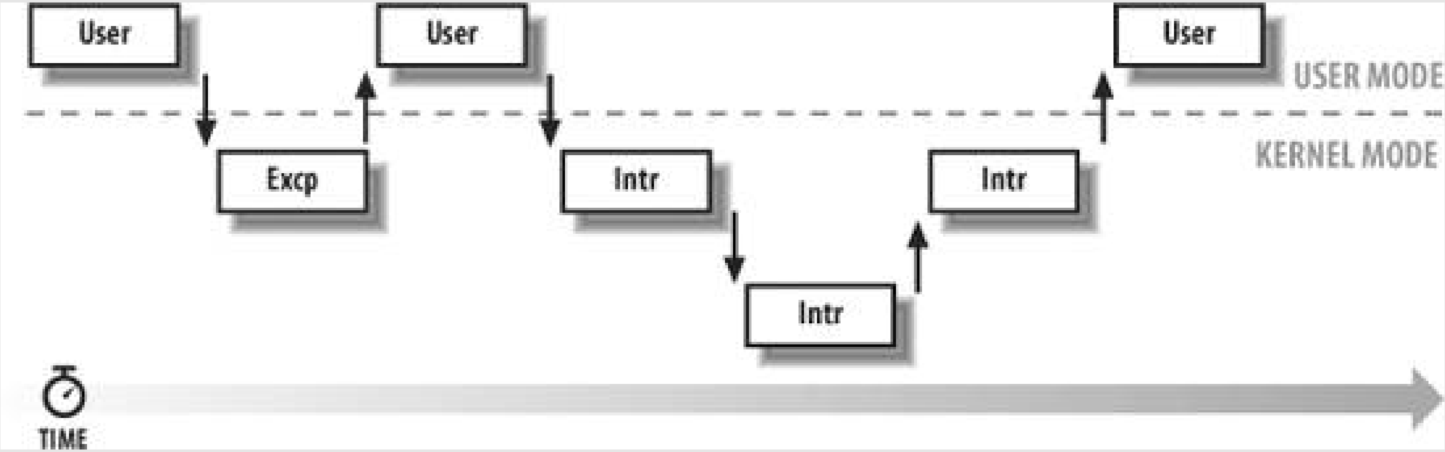


Process List Structure.

Short, Medium and Long term Scheduling



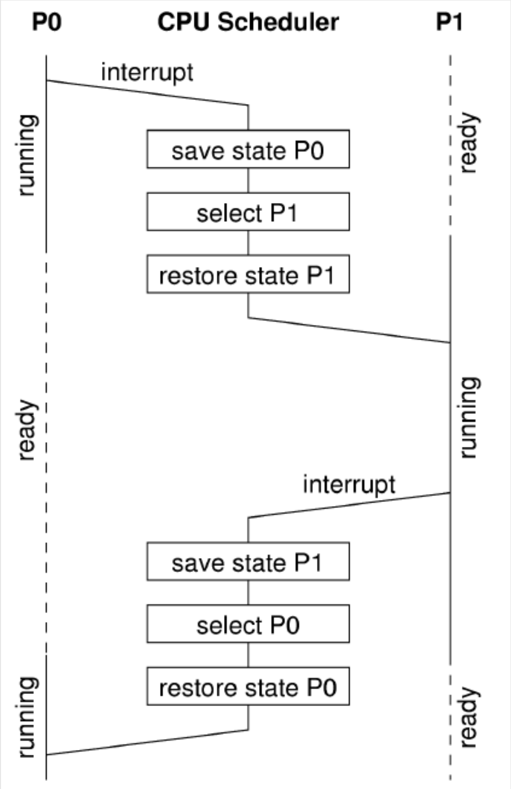
Interleaving of kernel control paths



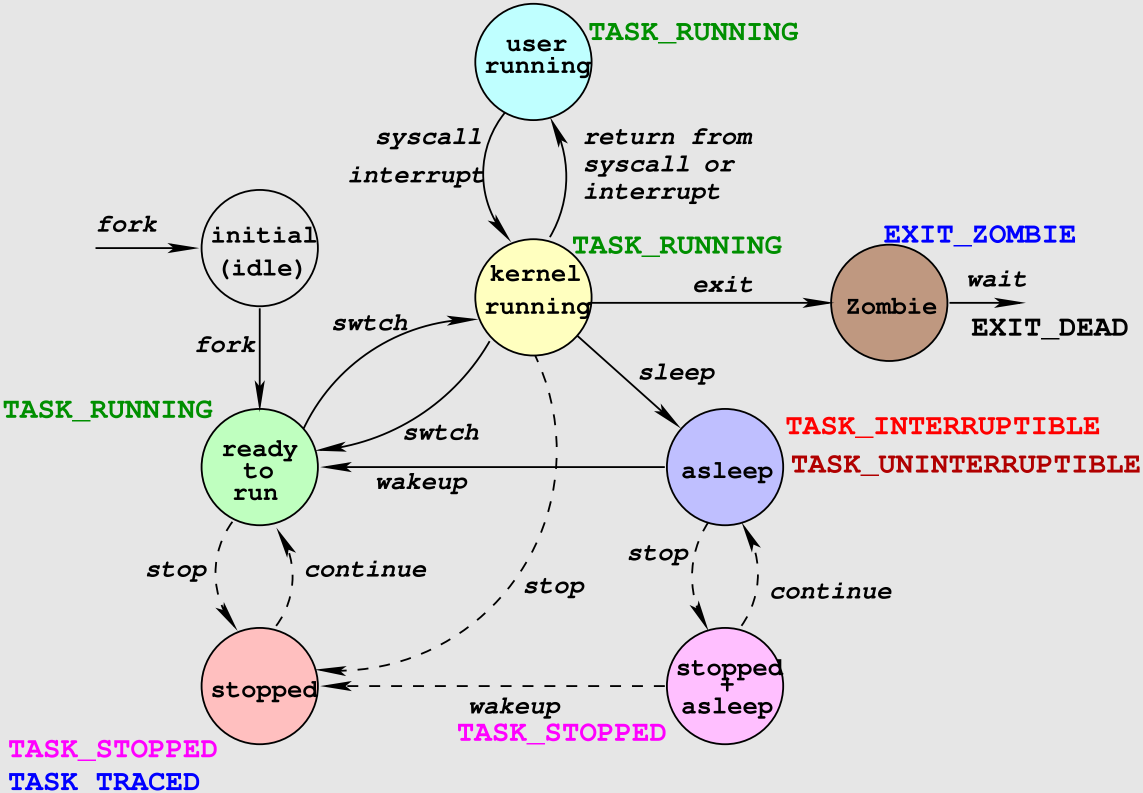
Process context

- Information that allows the OS to resume the execution of a process
- Process Control Block (PCB) – Process Descriptor in Linux
  - State
  - CPU registers
  - Scheduling info
  - Memory info
  - I/O info
  - Accounting info
- Process stack

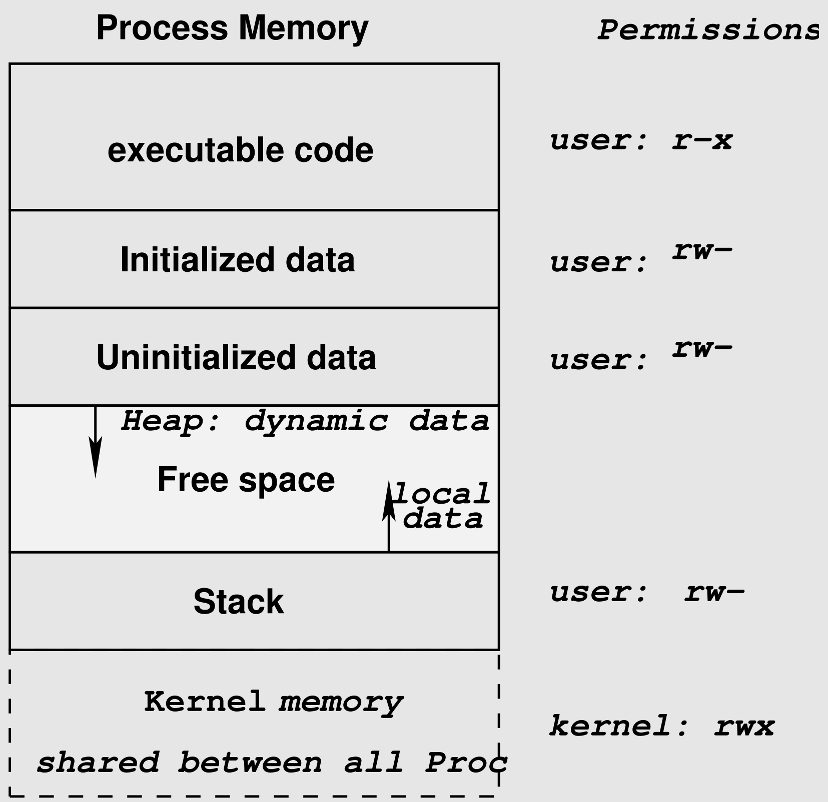
Context Switch



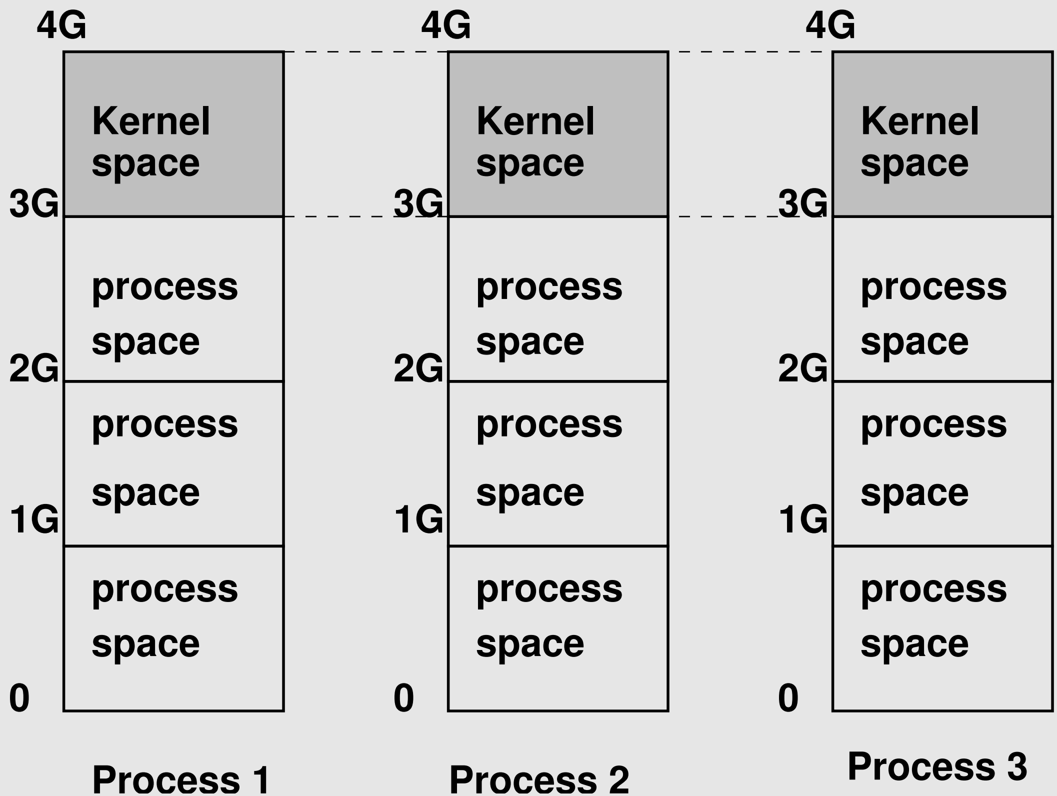
Linux Process States and Transitions



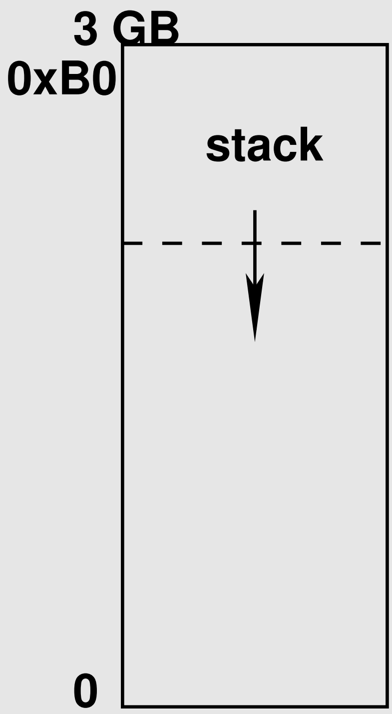
Unix/Linux: Process Memory Layout



Linux – Process and Kernel Virtual Memories



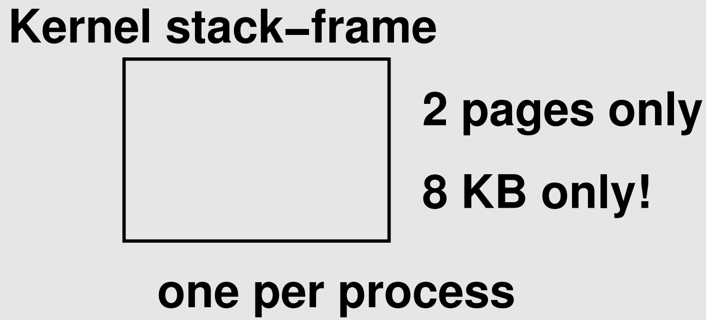
Linux – stacks for a user process and for Kernel



user pages may be

swapped

(only if dirty)

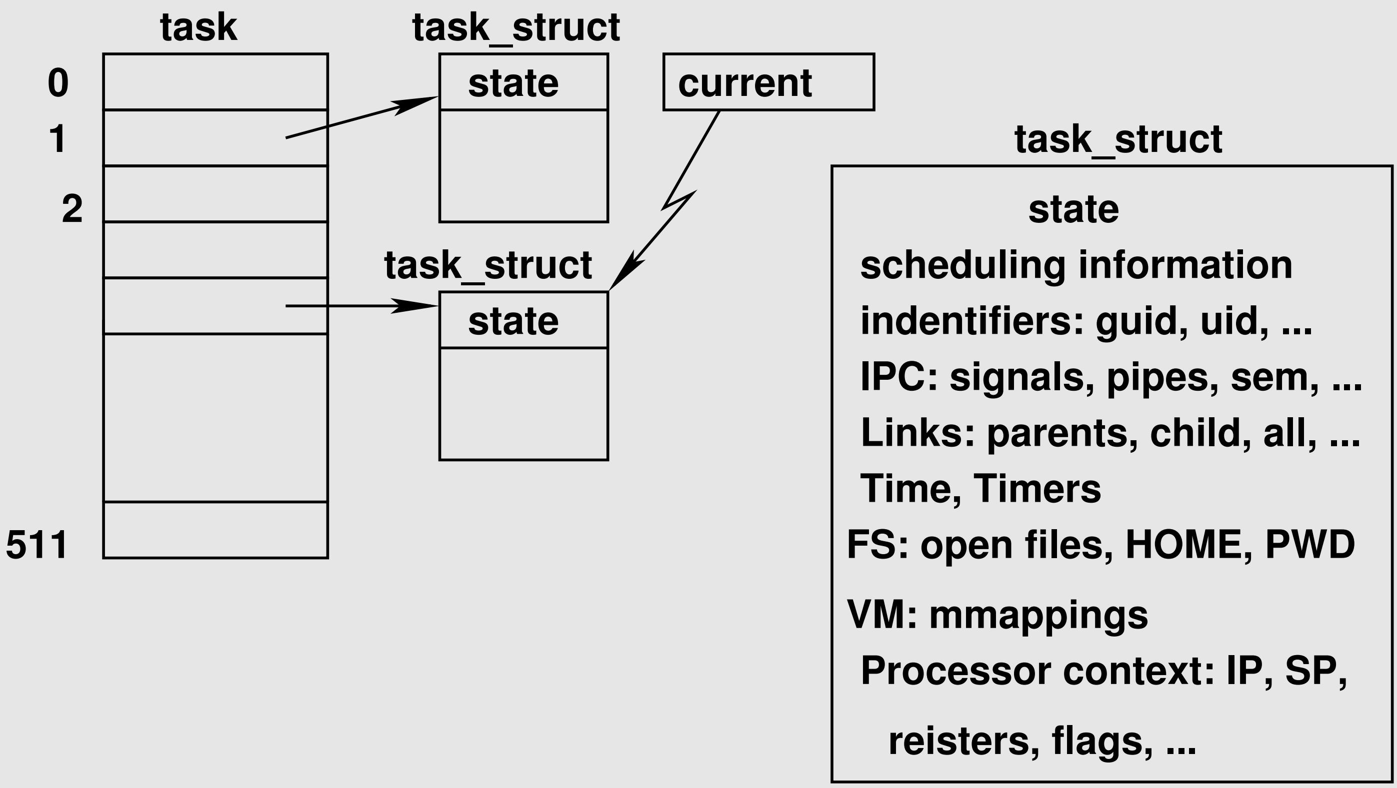


Unix: Process data structures

Minimum information that need be stored:

- The program counter (PC) and stack pointer (SP) registers
- The general purpose registers
- The floating point registers
- The processor control registers (Processor Status Word) containing information about the CPU state
- The memory management registers used to keep track of the RAM accessed by the process

Linux – Summary of task data-structures



Inside Linux

- File Hierarchy Standard (FHS)
  - /proc directory
- Virtual Memory (VM)
  - mmap function
- Virtual File System (VFS)
  - ext2, etx3, ext4 FS

What is a File System?

The words "File System" has, confusingly, three meanings in Unix/Linux world:

- the component of the O/S which handles files for the user; e.g., see source at /usr/src/linux/fs
- the organization of files on a media volume, usually in form of a tree;e.g., the /home directory is on a separate partition;
- the type of file organization, e.g., msdos, ntfs, ext2, etc. e.g., this installation is using ext2 and ext3 file-systms

You will have to understand the meaning by the context in which these words are used.

Filesystem Hierarchy Standard

- a single hierarchical tree structure that represents the file system as one whole single entity.
- each new file system added into this when mounted.
- mount point – the position in the FHS tree where a temporarily mounted FS resides.

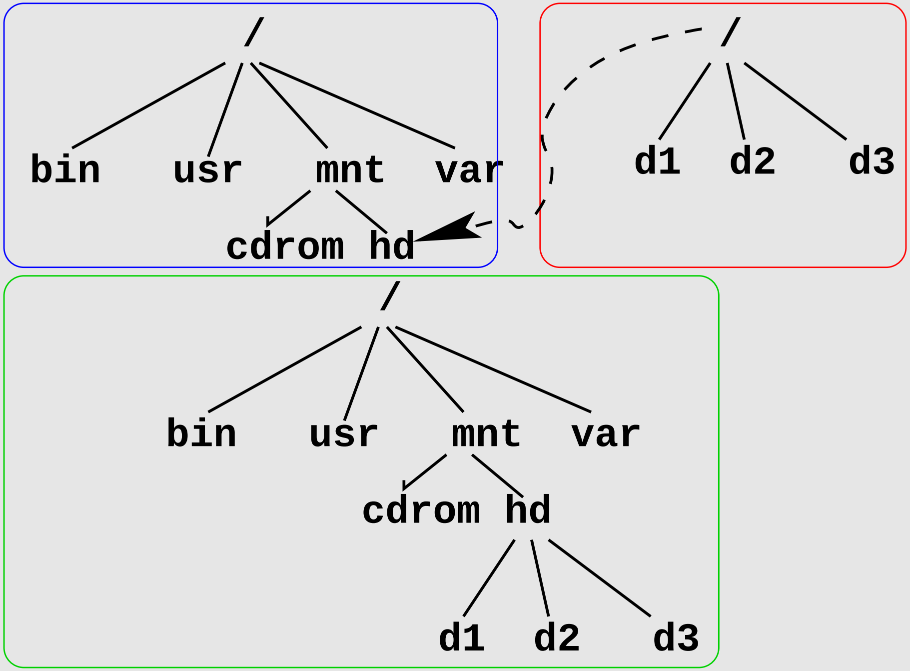
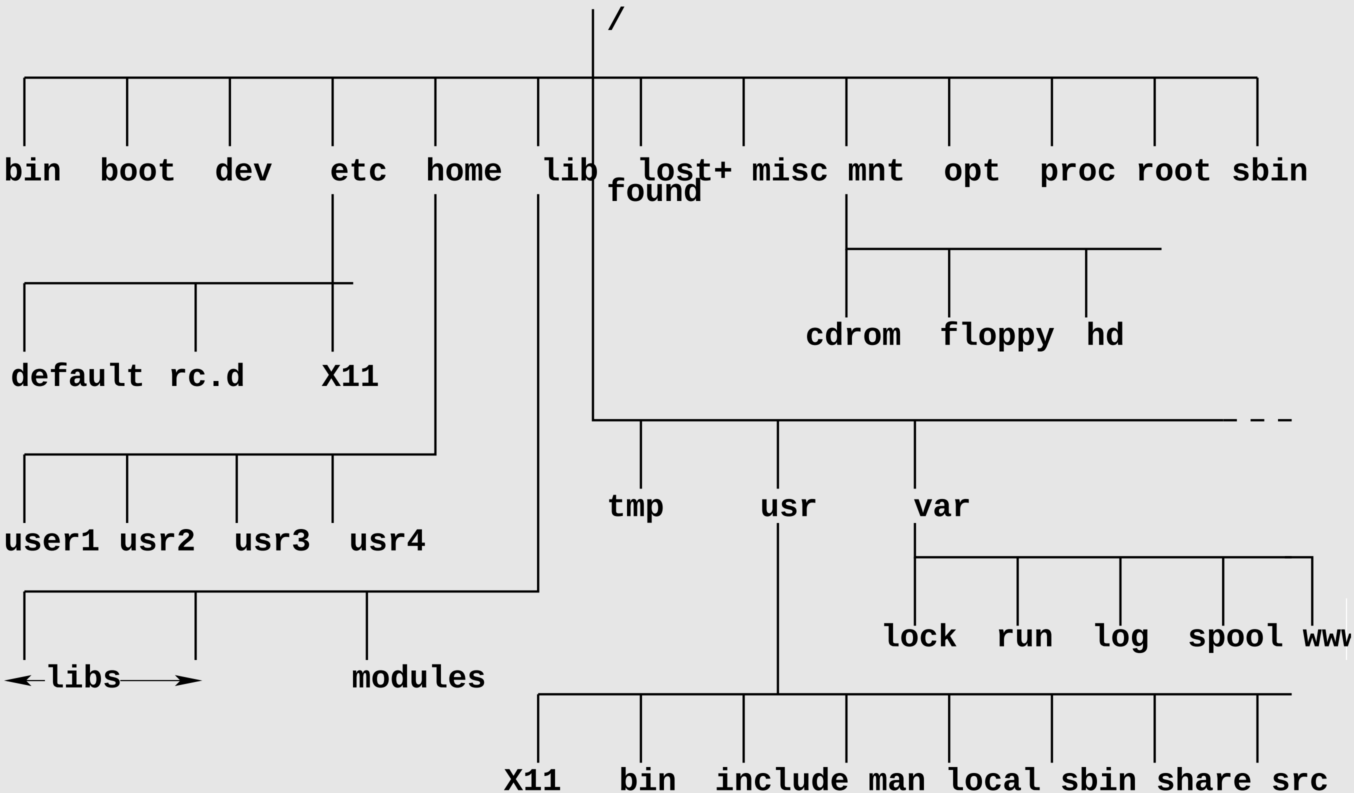


Figure 3 Mounting a FS

File Hierarchy Standard (FHS)



/proc directory

- A look into a working, live kernel
- Some sub-directories contain system-wide information
- One sub-directory for each active process, which contains information related to that process
- Many system parameters can be tuned dynamically by writing into appropriate ``file''.
- There are several utilities which access /proc directory to display information in more easily readable form – e.g. `lsmod, lspci`

Virtual Memory (VM)

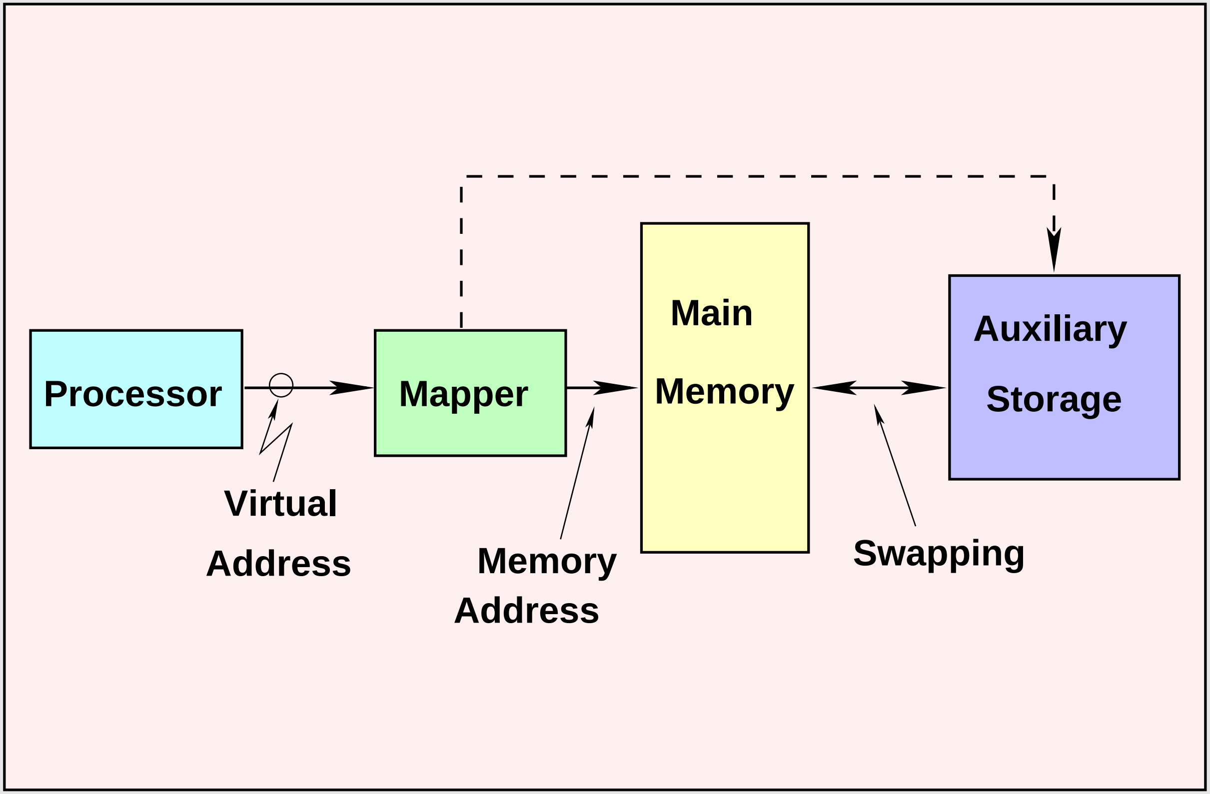


Figure 4 VM: An illusion of a very large Main Memory

Virtual Memory - Swapping

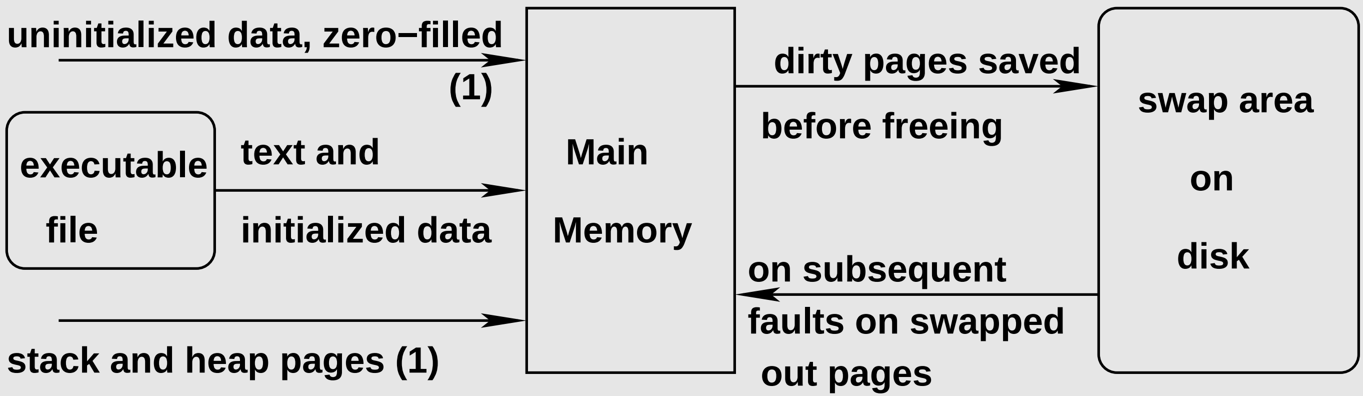
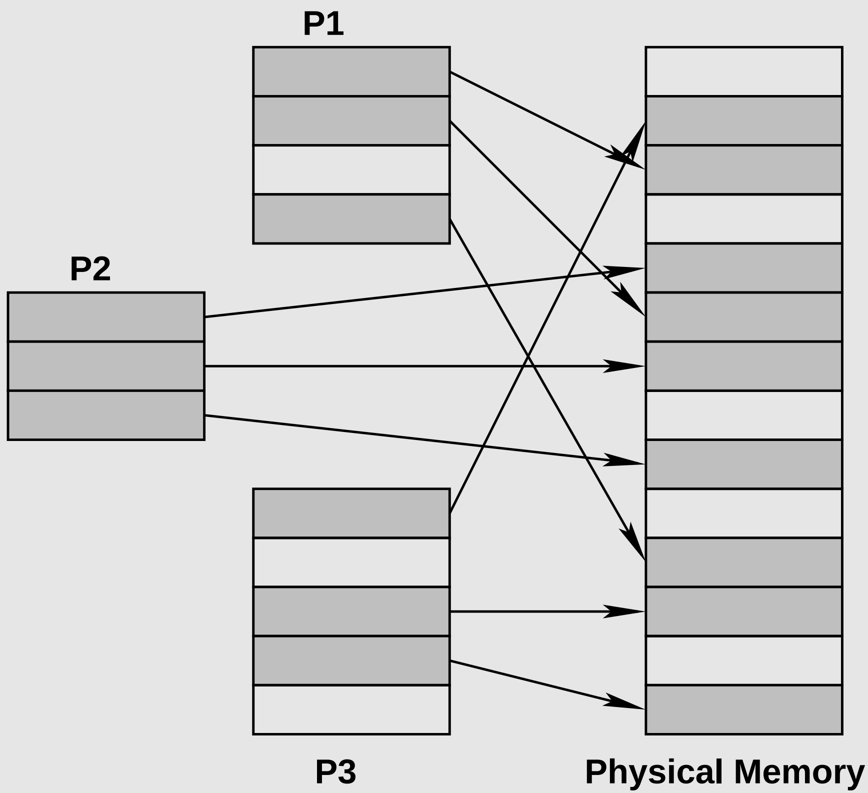
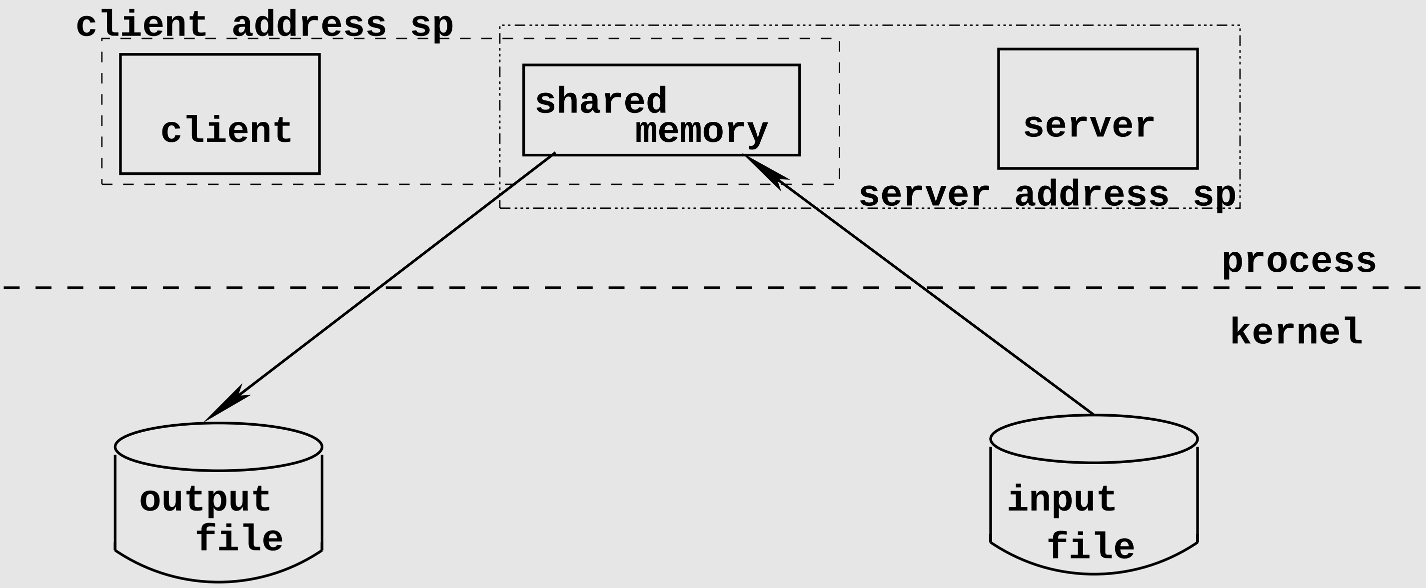


Figure 5 Swapping

Page allocation to processes under VM



Copying file data from server to client using shared memory



server to client using shared memory

- server gets access to a shared memory object using (say) a semaphore
- server reads from the input file into the shared memory object
- when the read is complete, the server notifies the client, using a semaphore
- client writes the data from the shared memory object to the output file

The `mmap` function maps either a file or a Posix shared memory object into the address space of a process. We use this function for three purposes:

- with a regular file to provide memory-mapped I/O
- with special files to provide *anonymous* memory mappings
- with shm_open to provide Posix shared memory between unrelated processes

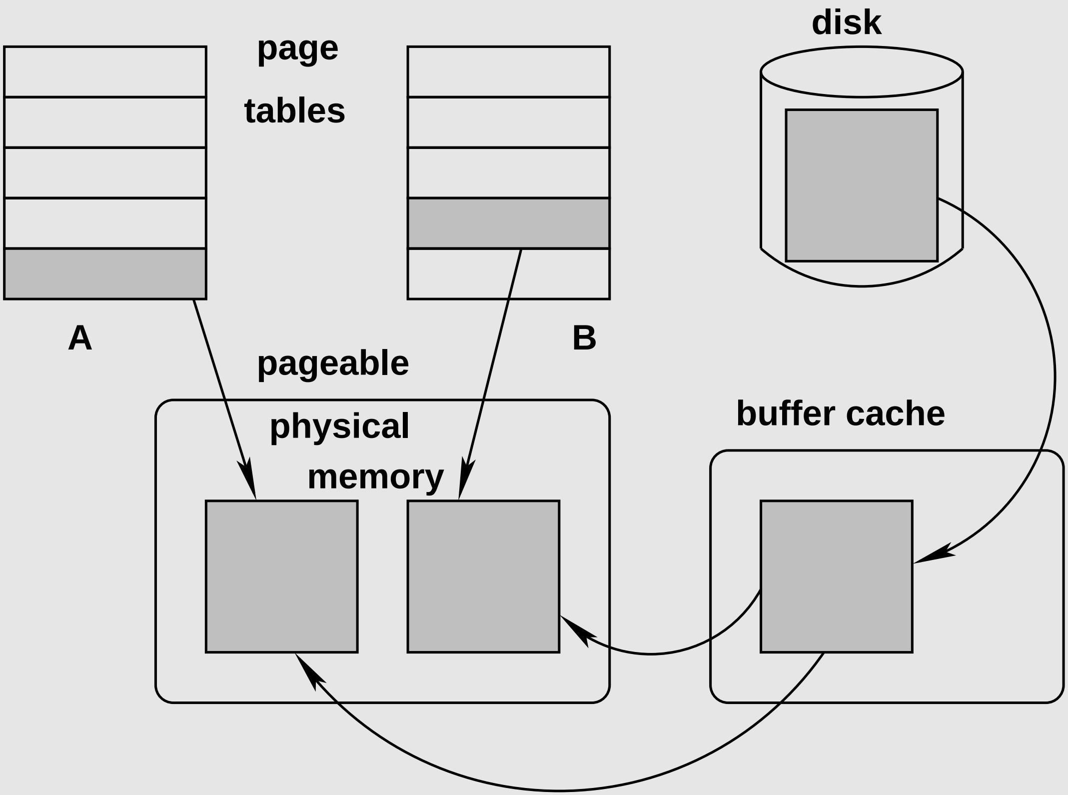
Memory-mapped files

- VM concept: if disk-space can replace RAM, why not vice-versa?
  - "File mapping" a central concept
  - Allows users to map a part of the address-space to a file and then use simple memory instructions to do file I/O
  - Also used as a basic organization scheme in the kernel, which may view the entire address-space as simply a collection of mappings to different objects
-

Traditional way in Unix

- open a file with "open" sys-call;
- use the file with read, write or lseek calls;
- sequential or random access;
- inefficient: time- and memory-wise:
  - one read to bring the page from disk to cache;
  - one mem-to-mem copy data from cache to user;
- for disk access a buffer cache is used;

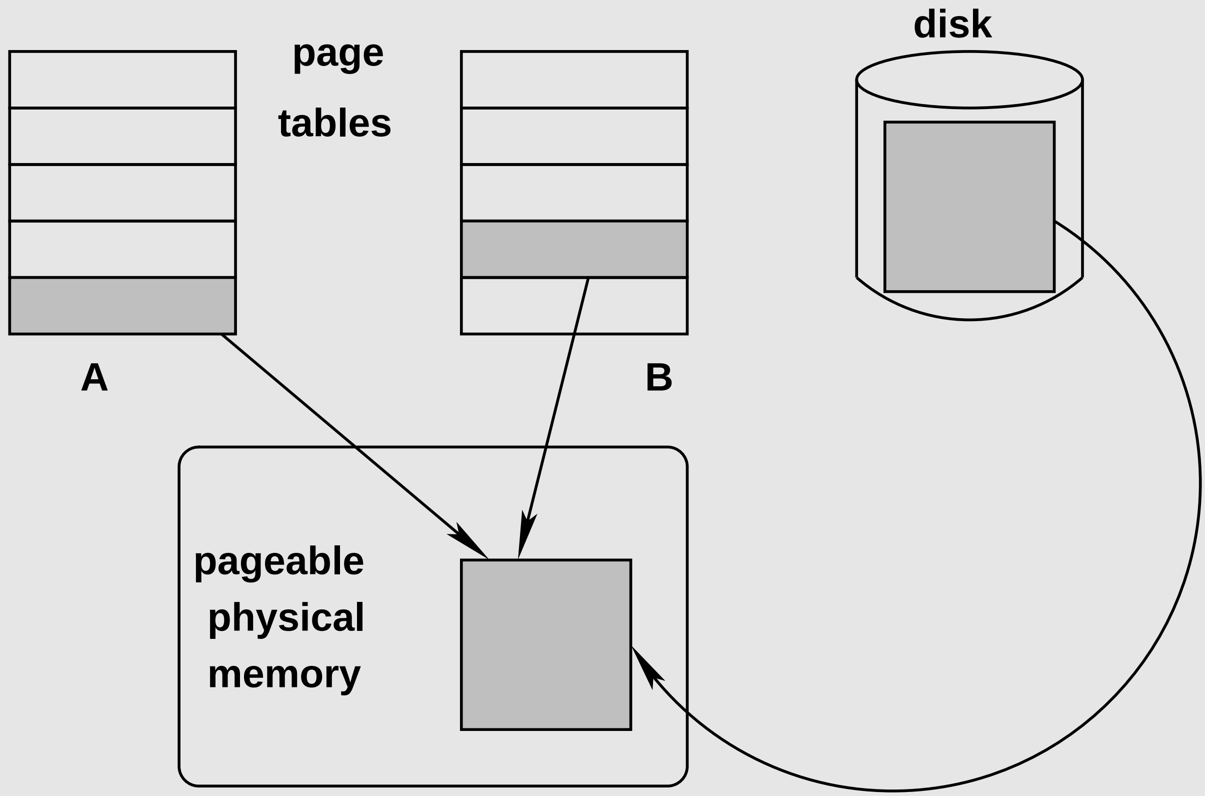
Two processes read the same page via buffer cache



Linux `mmap` way

- cache not used;
- processes map data pages in their address-space;
- only one copy of the page in memory on first page fault;
- process A trying access the page - a page fault;
- similarly, for process B, but no further copy from the disk;

Two processes map the same page into their address space



What happens on writing?

Two types of mapping:

- shared: mapping done to the shared object itself, written back to disk on page flushing;
- private: any modification - a private copy of the page is made, the changed version not written back on flushing;

Are traditional read/write useless?

No

- a read or write system call is "atomic" - the inode is locked during the I/O;
- memory-mapped files are accessed by ordinary program instructions, at most one word will be read or written atomically;
- synchronization is responsibility of the programer.
- changes to a mapped file are visible to all the processes which have mapped it, while in traditional files a read has to be issued to see the changes.

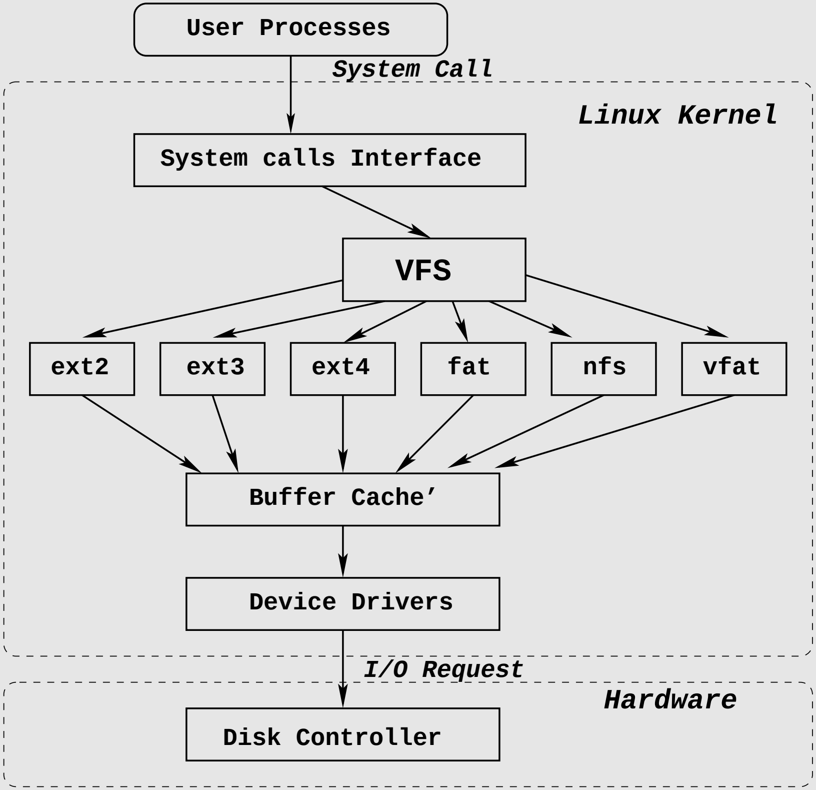
Virtual File System (VFS)

One of the most important features of Linux is its support for many different file systems. Some of them are:

- ext2, ext3, ext4
- xiafs, minix,
- umsdos, msdos, vfat, ntfs (MS)
- hfsplus (Mac OS)
- smb, ncp, iso9660,
- proc, nfs, sysv, hpfs, affs and ufs,
- and more can be added.

This feat is achieved via use of Virtual File System (VFS).

Virtual File System



Virtual File System

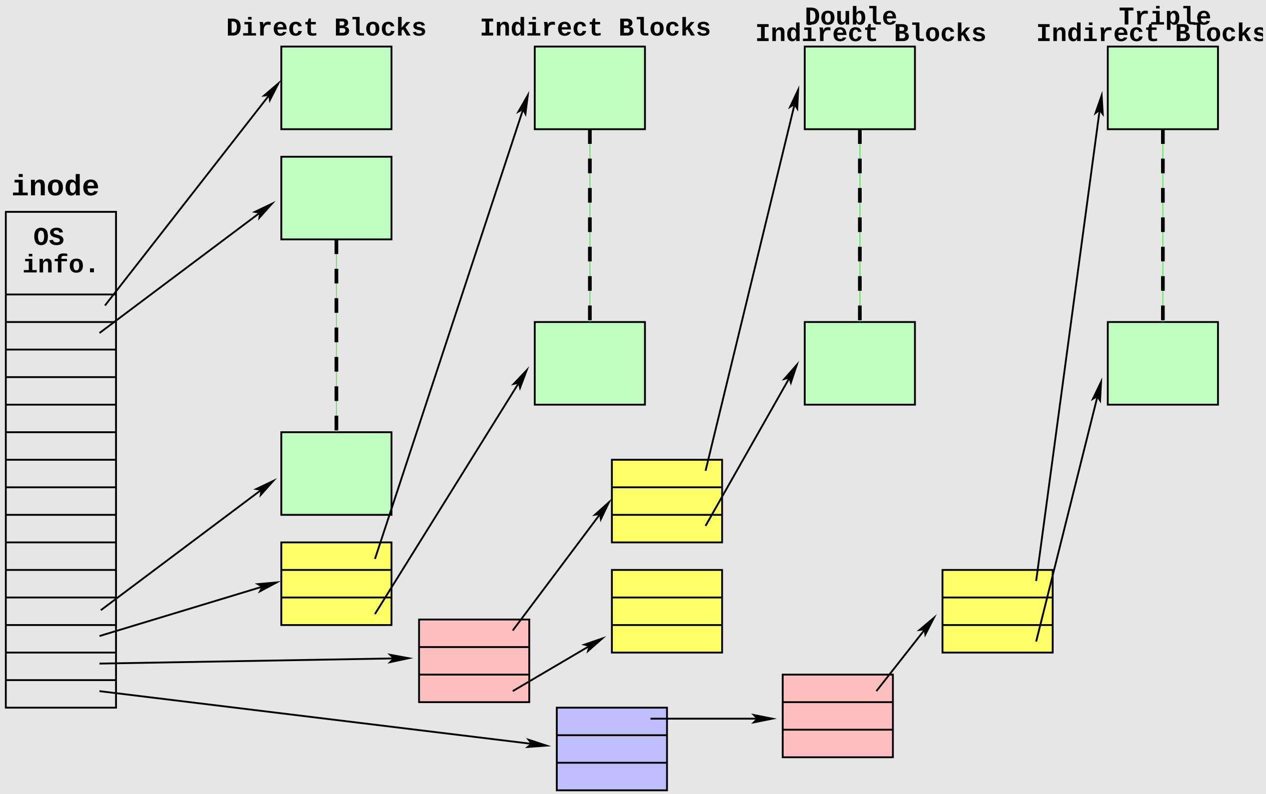
- software layer in the kernel that provides the filesystem interface to user-space programs
- provides an abstraction within the kernel which allows different filesystem implementations to coexist
- system calls `open`, `close`, `stat`, `read`, `write`, `chmod` etc.
- Directory Entry Cache (dcache or dentry cache): very fast pathname argument to dentry mapping
- individual dentry has a pointer to an inode
- inode: filesystem objects such as regular files, directories, FIFOs etc.
- `struct file_operations`

inode

An `inode` data structure, which represents the actual **inode** has the following major components:

- **Mode**
- **Owner info**
- **Size**
- **Time-Stamps**
- **Direct Block pointers**
- **Indirect Block pointers**
- **Double Indirect Block pointers**
- **Triple Indirect Block pointers**

Structure of an inode



Linux File System

- ext2, ext3 and ext4 FS
- FS utilities: fsck, ext2 utilities: dumpe2fs, tune2fs, etc.

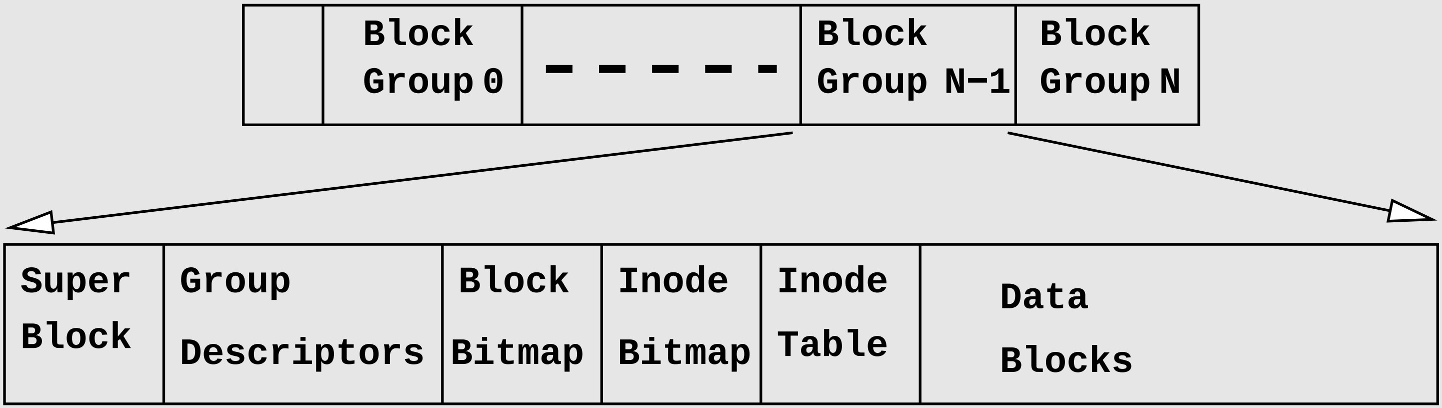
ext2 FS

- similar to traditional Unix filesystems
- hooks for Access Control Lists (ACLs), undeletion, compression
- journalling can be added (ext3)
- Blocks: 1024, 2048 or 4096 bytes (fixed)
- Block Groups: reduce fragmentation, minimise seeks for consecutive data
- block bitmap and inode bitmap: show which blocks and inodes in use
- maximum size of a block group is 8 times the size of a block
- Special files: use inode space differently
- Reserved Space: reserv a certainfraction of blocks for super-user
- Inode: 12 Direct blocks

ext2 FS – Limitations

- the ratio of inodes to blocks is fixed
- block size = 1KB: File size limit = 16GB, Filesystem size limit = 2047GB
- block size = 2KB: File size limit = 256GB, Filesystem size limit = 8192GB
- block size = 4KB: File size limit = 2048GB, Filesystem size limit = 16384GB
- limit of 32000 subdirectories in a single directory

Structure of ext2 FS

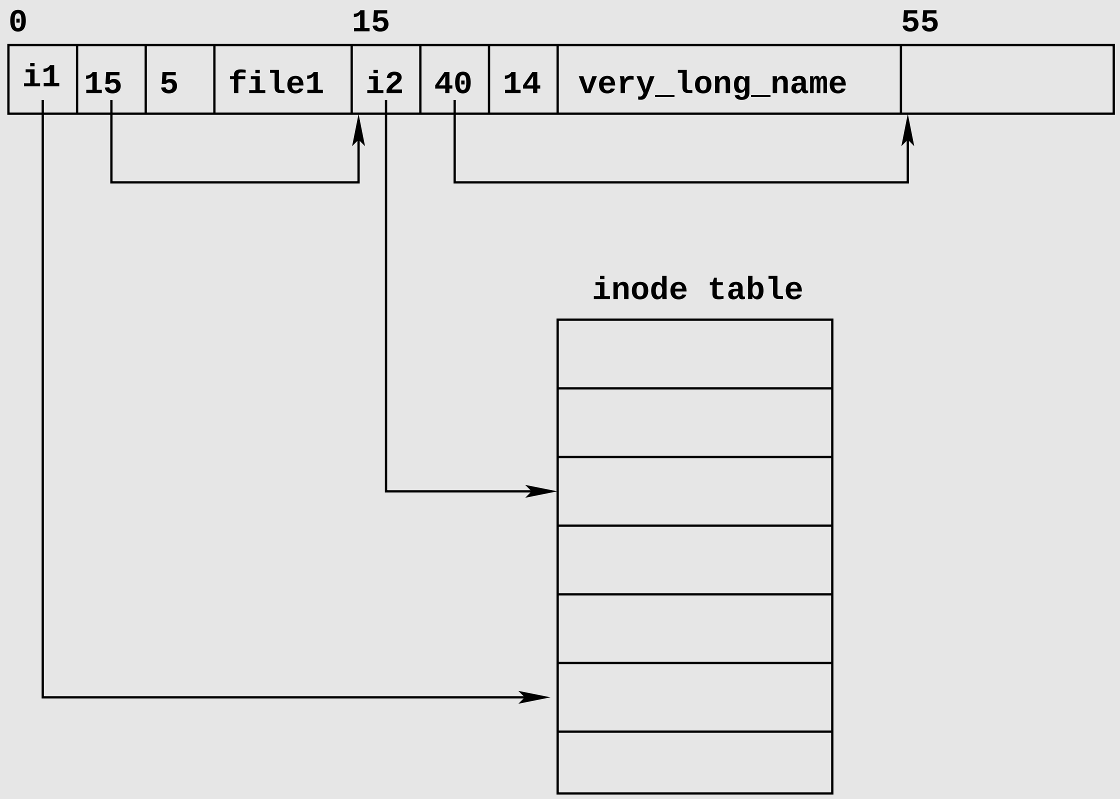


ext2 FS – Typical Lay-out

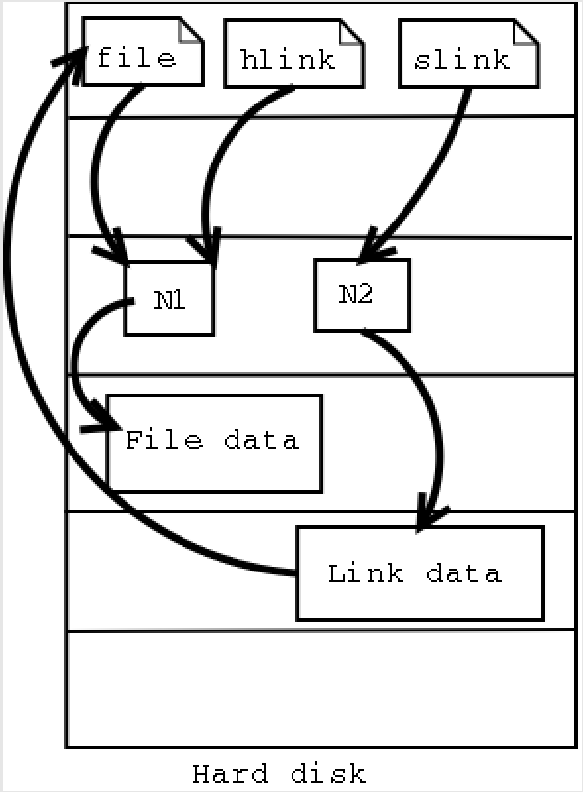
offset	No of blocks	description
0	1	boot record
1	1	superblock – blk grp 0
2	1	group descriptors
3	1	block bitmap
4	1	inode bitmap
5	214	inode table
219	7974	data blocks
8193	1	superblock backup – blk grp 1
8194	1	group descriptors backup
8195	1	block bitmap
8196	1	inode bitmap
8197	214	inode table
8408	7974	data blocks

Table 1 20MB partition meta-data layout

Directory structure in ext2 FS



Hard and Soft Links in ext2 FS



boot-block

Can be the MBR or in the partition.
MBR (Master Boot Record):

- The first sector on the device
- 512 bytes – 446 bytes: primary boot loader, which load secondary boot loader such as LILO or GRUB,
- 64 bytes:partition table, 2 bytes:special code.
- The partition table has enough room for four partitions.
- One of the four can be used as an extended

super-block

- contains all the information about the configuration of FS
- primary copy at offset of 1024 bytes from the start of the device
- backup copies: stored in block groups throughout FS
- super block fields:
  - total number of inodes and blocks in FS
  - number of free inodes and blocks
  - number of inodes and blocks in each block group
  - when the FS mounted and if cleanly unmounted
  - when modified

- FS version
- which OS created FS

- Extra fields in later versions of ext2 FS: volume name, a unique identification number, the inode size, and space for optional filesystem features to store configuration info.

ext2 FS utilities

- debugfs - ext2/ext3 file system debugger
- dumpe2fs - dump ext2/ext3 filesystem information
- e2fsck - check a Linux ext2/ext3 file system
- e2image - Save critical ext2/ext3 filesystem metadata to a file
- e2label - Change the label on an ext2/ext3 filesystem
- filesystems [fs] - Linux filesystem types: minix, ext, ext2, ext3, xia, msdos, umsdos, vfat, proc, nfs, iso9660, hpfs, sysv, smb, ncpfs
- mke2fs - create an ext2/ext3 filesystem
- resize2fs - ext2/ext3 file system resizer
- tune2fs - adjust tunable filesystem parameters on ext2/ext3 filesystems

fsck

- At boot time, most systems run a consistency check (e2fsck) on their filesystems.
- The superblock of the ext2 filesystem contains several fields which indicate whether fsck should actually run (since checking the filesystem at boot can take a long time if it is large).
- `fsck` will run if:
  - the filesystem was not cleanly unmounted,
  - the maximum mount count has been exceeded, or
  - the maximum time between checks has been exceeded.

References

- `/usr/src/linux-x.x.x/Documentation/filesystems`
- Linux Cross Referencer LXR lxr-0.9.5 latest
- Kernel Hacker's Guide khg
- Linux Kernell Internals lki
- The Linux Kernel tlk
- Linux Kernel Programming Guide lkmpg
- Bach "Design of Linux Operating System"