The golden age of hacking

OS

UNIX

GNU/Linux

UNIX access control

What Security Goals Does an Operating System Provide?

- Goal 1: Enabling multiple users securely share a computer
  - Separation and sharing of processes, memory, files, devices, etc.
- How to achieve it?
  - Memory protection
  - Processor modes
  - Authentication
  - File access control
- Goal 2: Ensure secure operation in networked environment
- How to achieve it?
  - Authentication
  - Access Control
  - Secure Communication (using cryptography and/or signatures)
  - Logging and Auditing
  - Intrusion Prevention and Detection (IPS/IDS)
  - Recovery

Memory Protection: access control to memory

- Ensures that one user's process cannot access other's memory
  - http://en.wikipedia.org/wiki/Memory_protection
  - Segmentation of memory
    - CS, DS, SS, etc.
  - Paged virtual memory
  - Protection keys
  - …
- Operating system and user processes need to have different privileges
- User example: priorities, scheduling etc.
  - Static vs. dynamic priority

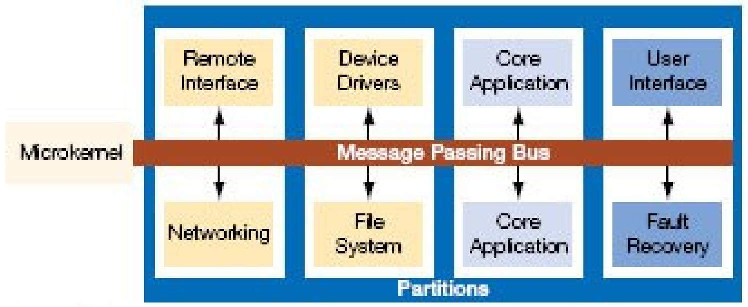
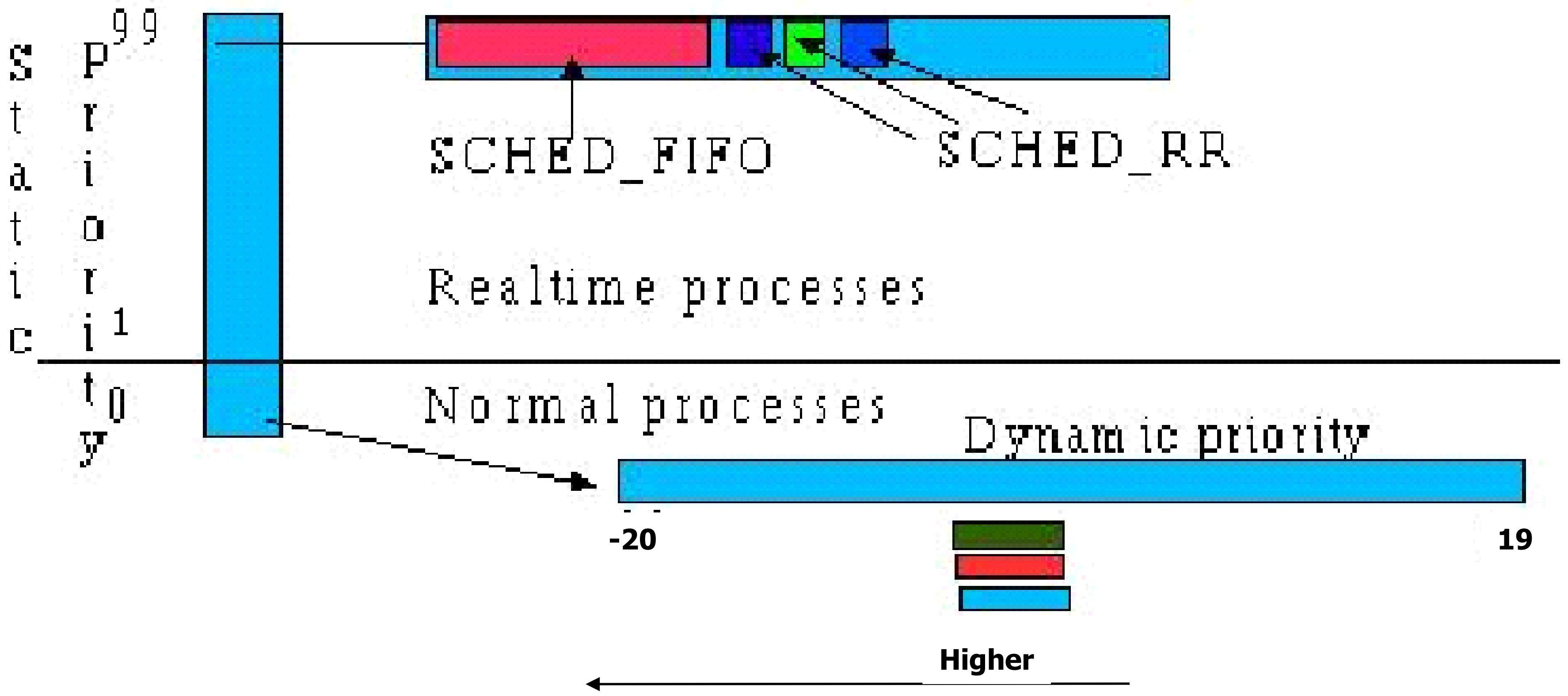


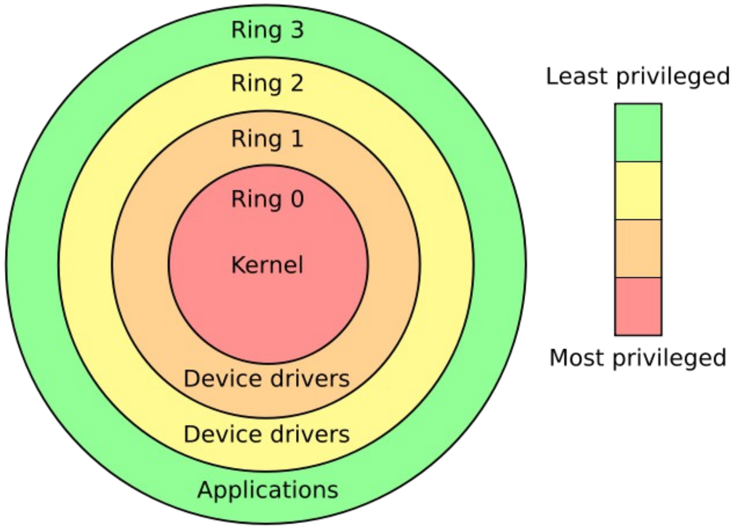
Figure 1 With resource partitioning, a system designer can place software processes into separate compartments and allocate each compartment a guaranteed portion of memory and CPU time. This approach prevents malicious or poorly written processes from monopolizing resources needed by other processes.

Linux priorities and polices

- **Dynamic range goes from -20 (highest priority) to 19 (lowest)**
  - **Default is 0**
- **Console commands nice and renice (only root can rise priority)**
  - **nice - run a program with modified scheduling priority**
  - **renice - alter the priority of running processes**



CPU Modes (a.k.a. processor modes or privilege)



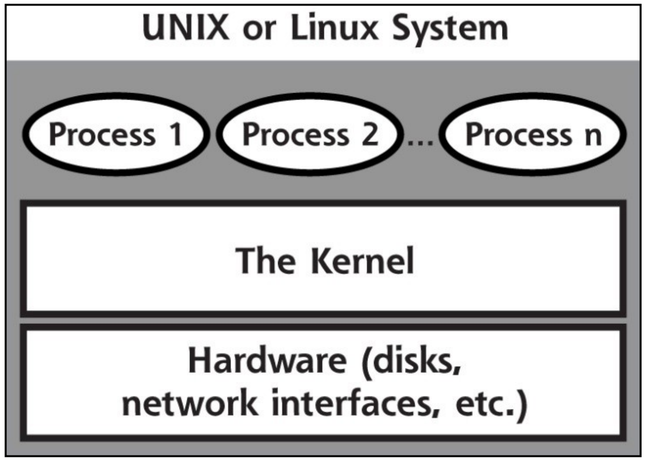
- User mode
  - Access to memory is limited, cannot execute some instructions, e.g., disable interrupts, change arbitrary processor state, access memory management units
- System mode (privileged mode, master mode, supervisor mode, kernel mode)
  - Can execute any instruction and access any memory location, e.g., accessing hardware devices, enabling and disabling interrupts, changing privileged processor state, accessing memory management units, modifying registers for various descriptor tables
- Transition from user mode to system mode must be done through well defined call gates (system calls)
- Reading: http://en.wikipedia.org/wiki/CPU_modes

System Calls

- Guarded gates from user mode (space, land) into kernel mode (space, land)
  - Use a special CPU instruction (often an interruption), transfers control to predefined entry point in more privileged code; allows the more privileged code to specify where it will be entered as well as important processor state at the time of entry
  - The higher privileged code, by examining processor state set by the less privileged code and/or its stack, determines what is being requested and whether to allow it
  - Reading: http://en.wikipedia.org/wiki/System_call
- Linux 2.6 vmsplice() system call - **Local Root Exploit!**
  - Basically a kernel buffer overflow
  - Affects Linux 2.6.17 - 2.6.24.1
  - man vmsplice > splice user memory pages into a pipe
    - http://it.slashdot.org/article.pl?sid=08/02/10/2011257#
    - http://lwn.net/Articles/268783/

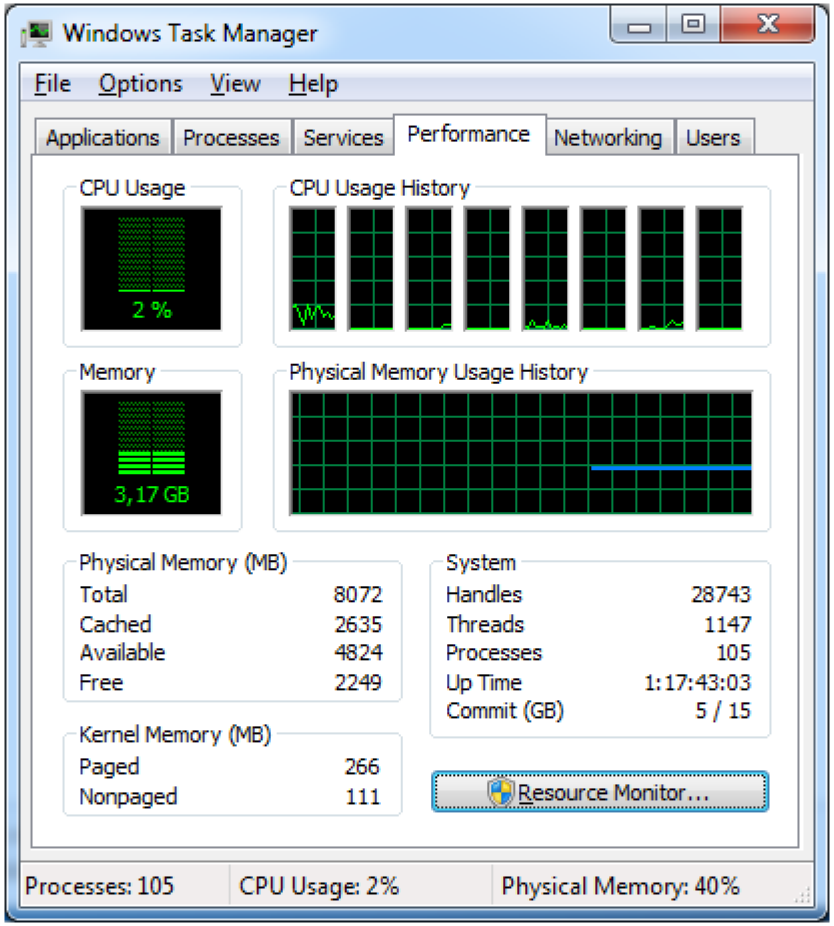
Kernel space vs. User space

- Part of the OS runs in the kernel mode
  - Known as the OS kernel
- Other parts of the OS run in the user mode, including service programs (daemon programs), user applications, etc.
  - They run as processes
  - They form the user space (or the user land)



- Difference between kernel mode and processes running as root (or superuser, administrator)?

Processes, threads and CPU



Physical Memory (MB)

Total	8072
Cached	2635
Available	4824
Free	2249

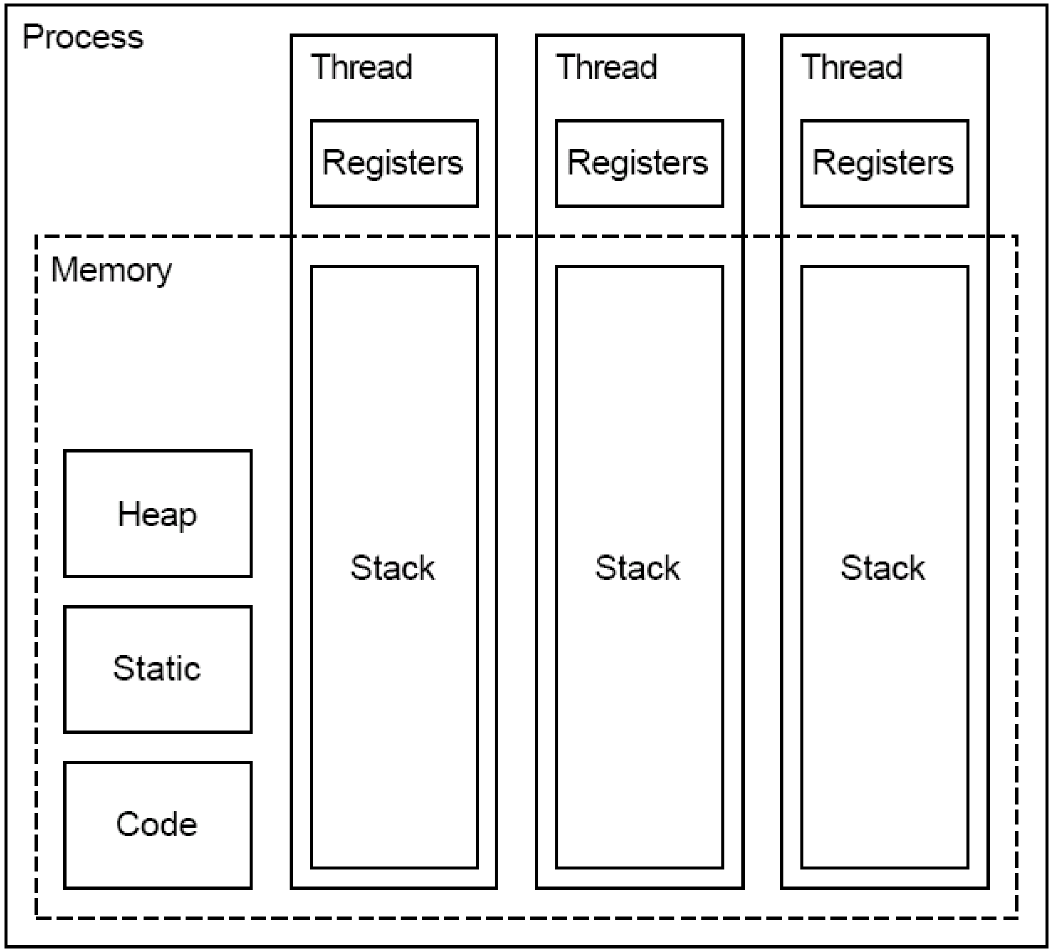
Kernel Memory (MB)

Paged	266
Nonpaged	111

System

Handles	28743
Threads	1147
Processes	105
Up Time	1:17:43:03
Commit (GB)	5 / 15

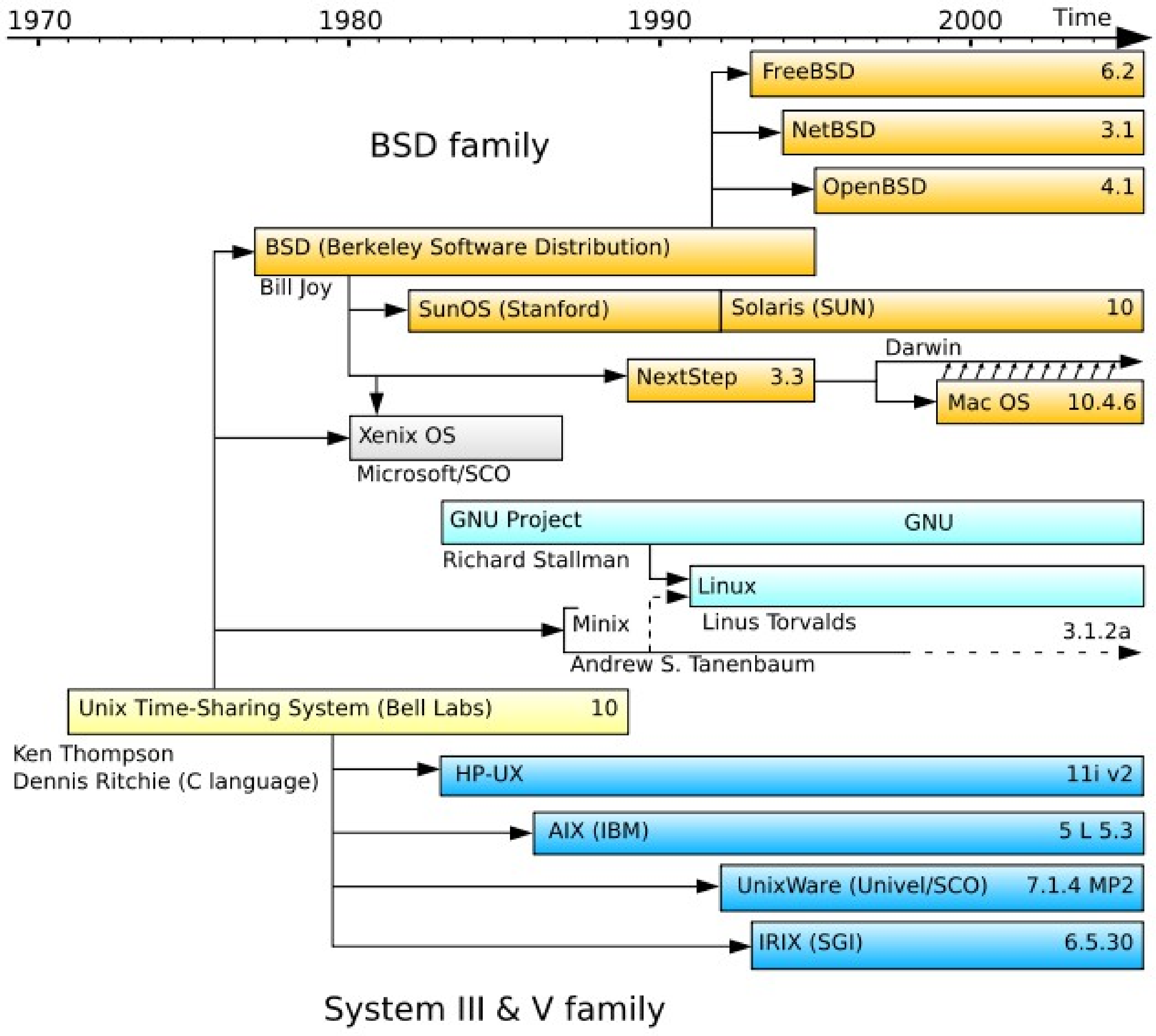
Processes: 105 CPU Usage: 2% Physical Memory: 40%



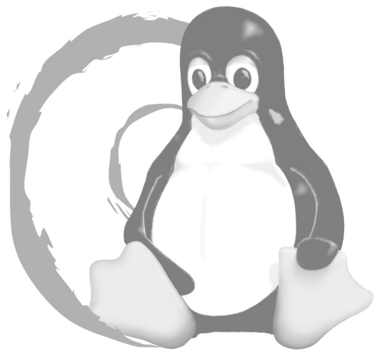
Monolithic kernel

- One big kernel provides all services, e.g., file system, network services, device drivers, etc.
  - All kernel code run in one address space and can directly affect each other
  - E.g., Linux 2.6.x kernel has about 6 millions of code
  - Pros: efficiency
  - Cons: complexity, bugs in one part affects all
- Examples
  - UNIX-variants
- Kernels with loadable kernel modules are still monolithic

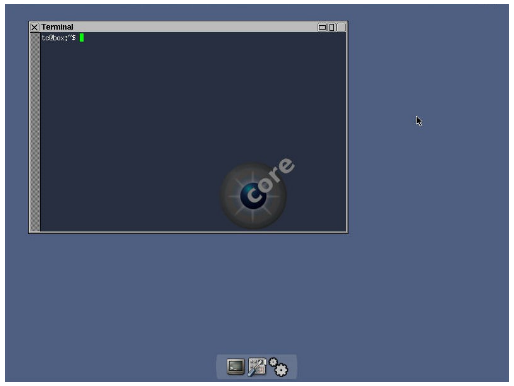
History of UNIX Family of Operating Systems



What is GNU/Linux?



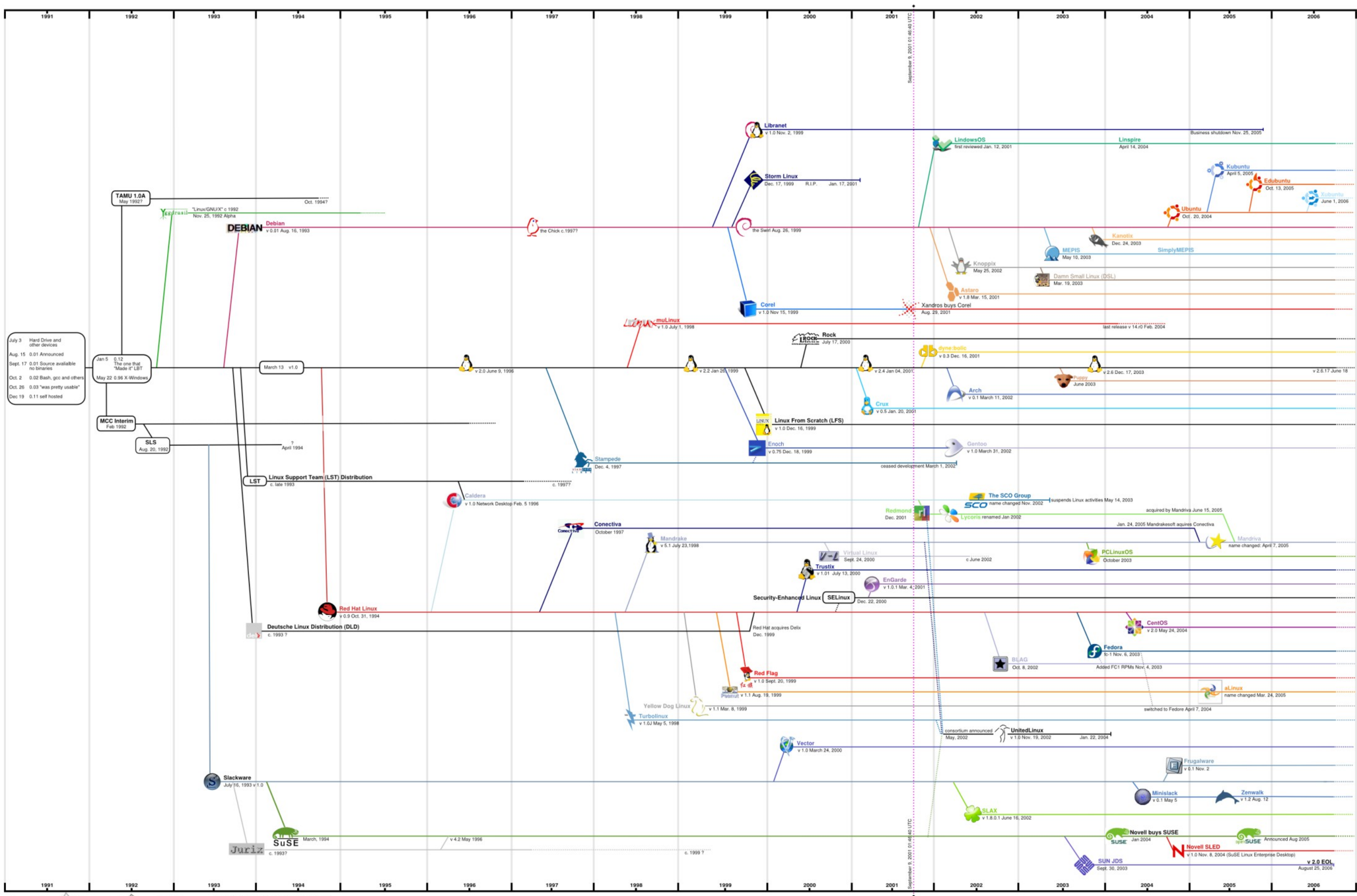
- GNU system tools and applications
  - System programs, user programs, compilers, etc.
- GNU common system libraries
  - Contains functions that applications use system calls on to communicate with the kernel, example: libc, pthread
  - Tip! ldd <executable file> shows which shared libraries are compiled in
- Linux (kernel)
  - 1991 Linus Torvalds, Intel 80386 processor
  - Learn more: http://kernelnewbies.org
- Open Source (GNU) + free UNIX-clone = GNU/Linux
- Advantages
  - Stable, scalable, “swiss armyknife”, modular
  - Performance, economical on resources
  - Open source, large number of free software/drivers
  - Programming development, network handling
  - Embedded possibilities, 10MB - http://tinycorelinux.com/
- Disadvantages
  - Support? Drivers? Security? No unified distributions…
  - High learning threshold?
    - Windows is for sure far more complex under the hood!



```
tc@box:~$
```

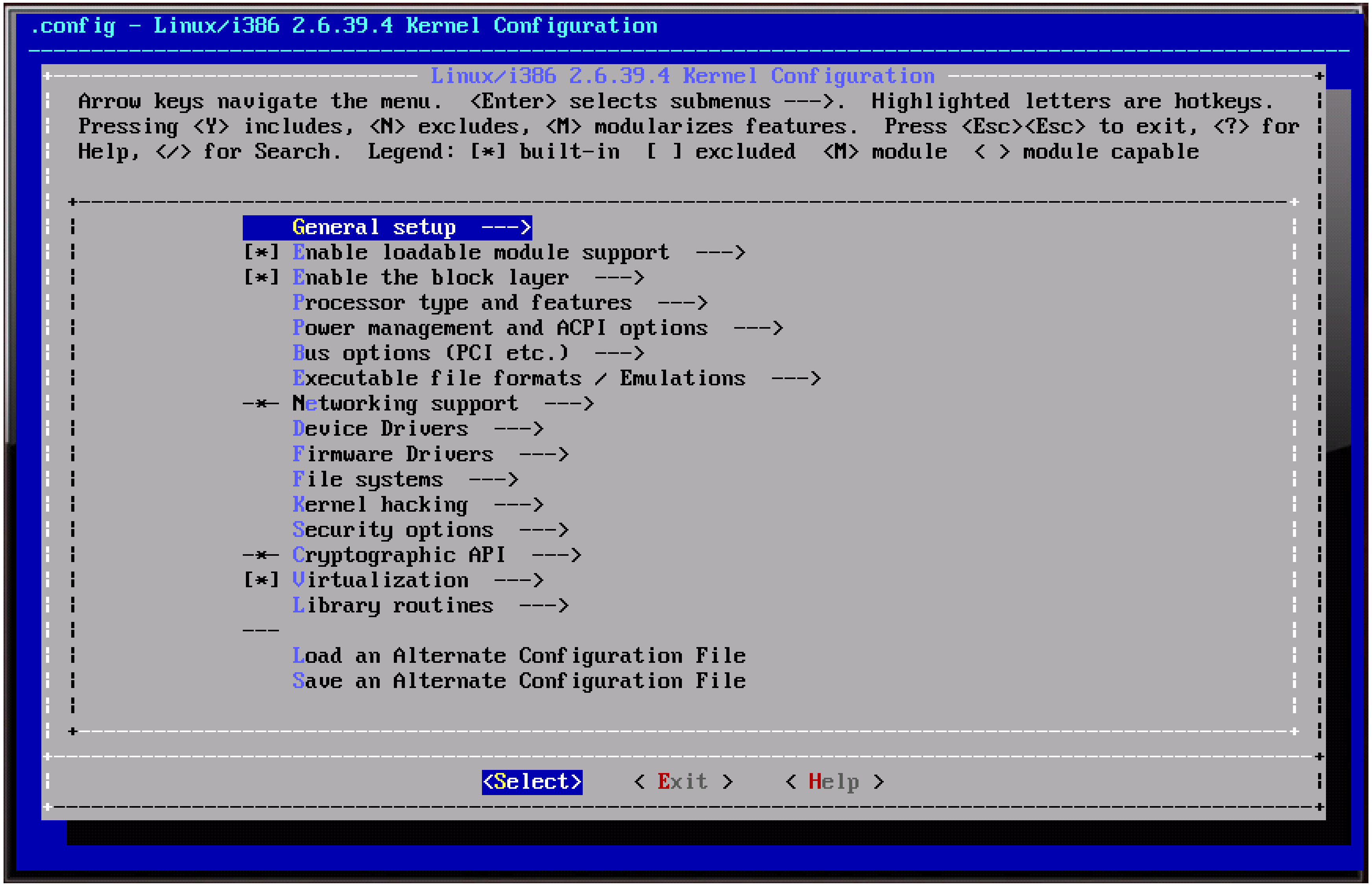
http://www.kde-files.org/content/show.php?content=46315

Yet Another Linux Distribution Timeline



Version 1.01 by jlk@osml.eu (Open Source Migrations)
Inspired by A. Sandoval's "Linea del tiempo Distribuciones Linux"
http://microteknologias.cl and "Linux distro timeline" by
nonplusx@gmail.com http://www.kde-files.org/content/show.php?content=44218

Linux kernel config

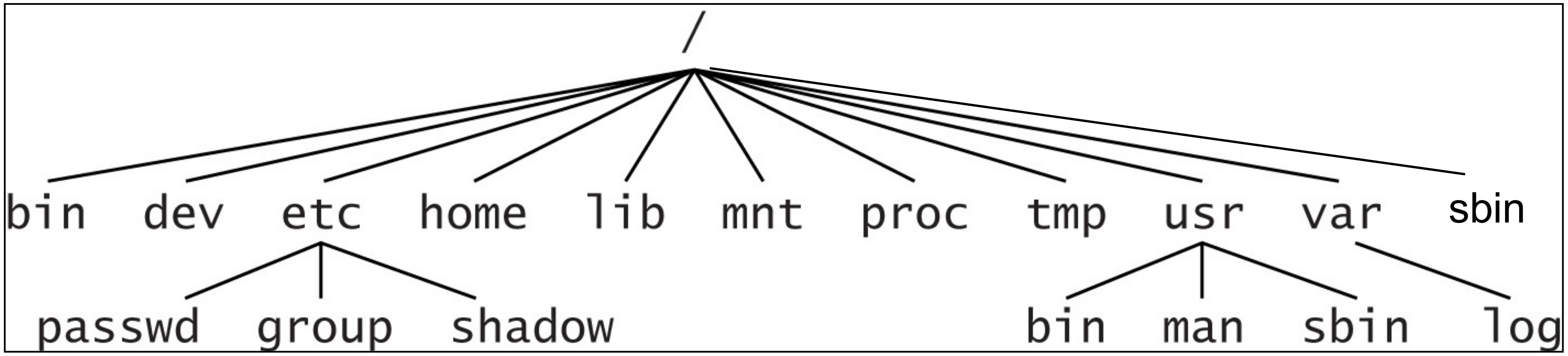


```
.config - Linux/i386 2.6.39.4 Kernel Configuration

                    Linux/i386 2.6.39.4 Kernel Configuration
  Arrow keys navigate the menu.  <Enter> selects submenus --->.  Highlighted letters are hotkeys.
  Pressing <Y> includes, <N> excludes, <M> modularizes features.  Press <Esc><Esc> to exit, <?> for
  Help, </> for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < > module capable

                  General setup  --->
              [*] Enable loadable module support  --->
              [*] Enable the block layer  --->
                  Processor type and features  --->
                  Power management and ACPI options  --->
                  Bus options (PCI etc.)  --->
                  Executable file formats / Emulations  --->
              -*- Networking support  --->
                  Device Drivers  --->
                  Firmware Drivers  --->
                  File systems  --->
                  Kernel hacking  --->
                  Security options  --->
              -*- Cryptographic API  --->
              [*] Virtualization  --->
                  Library routines  --->
              ---
                  Load an Alternate Configuration File
                  Save an Alternate Configuration File

                        <Select>    < Exit >    < Help >
```

Unix GNU/Linux folder tree

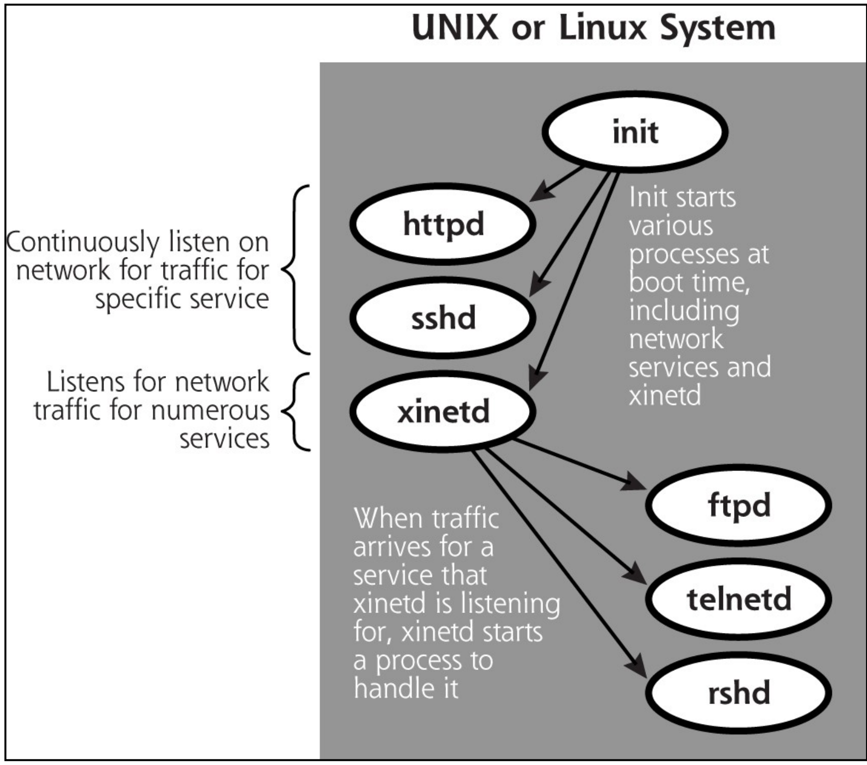
- Almost everything is referenced as a file in Linux/Unix

- /bin - Common binary files
- /boot - Linux kernel, boot loader etc.
- /dev - Device files, interface to all hardware in the system
- /etc - All configuration files for hardware, programs och start up scripts lies here
- /home - Home folder for users on the system lies here
- /lib - Place where library files (*.so, *.dll in Windows) and kernel modules is located
- /proc - Peek into the kernel, information about running programs and installed hardware
- /sbin - Superuser (root) users binaries that only he/she should be able to run
- /usr - Folder where all installed programs are put, mirror of the folder root ”/”
- /var - Folder for variable files as /var/log (log files) and /var/spool (print que, mail etc.).



Daemons and enviroment

- /etc/init.d scripts and symbolic links in /etc/rc**?**.d
  - Start up (S#) and stop scripts (K#), run-levels (the **?**: 0-6, S)
  - S = single user mode
- Inetd, Xinetd
  - The Internet super server that run other daemons
  - Port numbers defined in /etc/services
  - Inetd config in /etc/(x)inetd.conf
- Cron daemon
  - crontab -e / -l
- env
  - set/unset variables etc.
  - Do not put '.' in path!

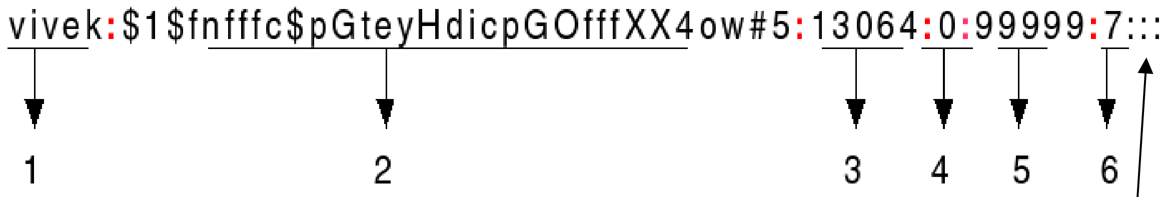

UNIX or Linux System
init
Init starts various processes at boot time, including network services and xinetd
Continuously listen on network for traffic for specific service
httpd
sshd
Listens for network traffic for numerous services
xinetd
When traffic arrives for a service that xinetd is listening for, xinetd starts a process to handle it
ftpd
telnetd
rshd

Most used commands

- **ls** list all files, lsof list all open files (will list almost everything!)
- **cd ..** go up one level in the directory structure. Using <directory> instead goes to that directory
- **cp -r** (recursive) directory name copies a whole directory
- **mkdir** make directory
- **rmdir** remove directory
- **rm -r** removes all underlaying directories and files
- **pico**, **nano** or **vi** <filename>, edits the file filename
- **mv** move or rename a file or directory
- **df -h** shows free disc space in megabyte
- **du -h** shows disc usage
- **less <filename>**, shows what a file contain
- **kill -HUP pidnr** terminates a process, **killall** <process name>
- **ps -aux** list processes
- **man <command name>** shows the manual for the command
- **locate -b <filename>** find a file by name (which match a pattern)

Accounts and groups

- /etc/passwd
  - Login name, uid, gid, GECOS info (full name, extra info etc.), home directory, login shell
  - /etc/shadow
    - Guarded file

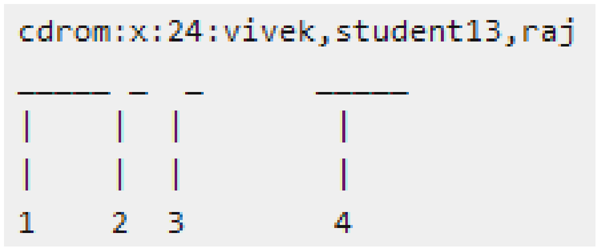


```
vivek:$1$fnfffc$pGteyHdicpGOfffXX4ow#5:13064:0:99999:7:::
```

1. user name, 2. hashed password, 3. last changed, 4. minimum days valid, 5. maximum …, 6. warn days passwd expire, (7. inactive, 8. expire)

"$1$"=MD5, "$5$"=sha-256, "$6$"=sha-512, (man shadow, crypt (3))

- /etc/group
  - 1. Group name, 2. Password, 3. gid, 4. Group members

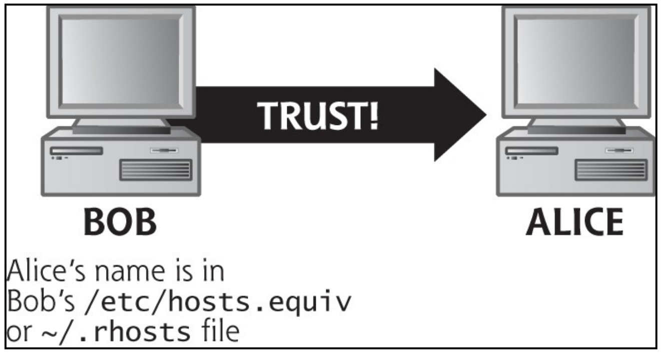


```
cdrom:x:24:vivek,student13,raj
```

- Important configuration files
  - /boot/grub/grub.conf (/etc/lilo.conf), /etc/fstab, /etc/modules

Trusts, remote access and logs

- Trusts
  - Based on IP-address
  - Use r-commands without passwd
- Remote access
  - Do not use rlogin, rcp, rsh, telnet, NFS
  - Use ssh (ssh2), scp instead
- Logs and auditing
  - /var/log/auth.log
  - /var/log/messages
  - /var/log/syslog
  - /var/log/*appliction*/***
- Commands that operate on some of the log files
  - w – who is logged in and what they are doing
  - last – list of last logged in users
  - lastlog - report last login for all users
  - dmesg - print or control the kernel ring buffer



/etc/syslog.conf - configuration file for syslogd
e.g. messages and syslog

Basic Concepts of UNIX Access Control: Users, Groups, Files, Processes

- Each user has a unique UID
- Root (super user) account always got UID 0
- Users can belong to multiple groups
- Processes are subjects
  - Associated with uid/gid pairs, e.g., (euid, egid), (ruid, rgid), (suid, sgid) - Effective, Real and Saved
- Objects are files: each file has the following information
  - owner (user account)
  - group (owner group)
  - 12 permission bits
    - read/write/execute for owner, group and everyone else
    - setuid, setgid, sticky bit

Unix GNU/Linux files and dirs

- Upper & lower case matters in directories and filenames

  ```
  # ls -al  show all user rights in a directory, example:
  -rwxr--rw- 1 hjo users 4746 9 jan 10 13:46 test
  ```

  r = read access, w = write access, x = execute access

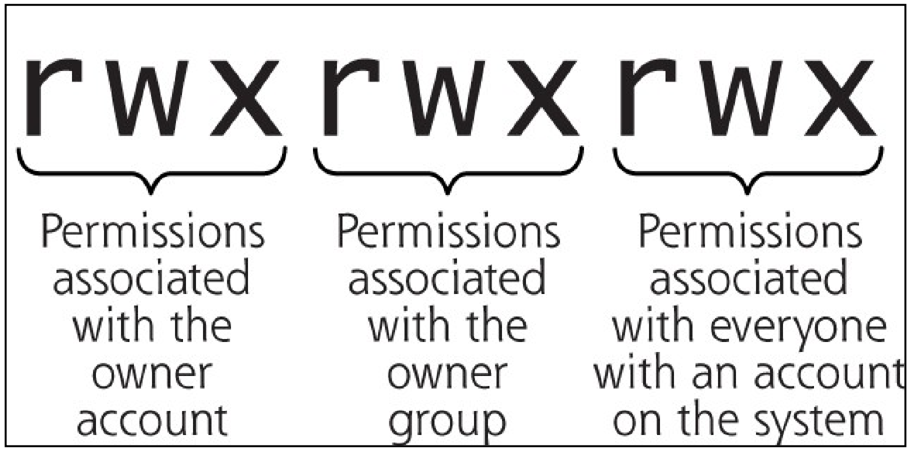
  – Change owner rights (order is UserGroupOther)

  ```
  # chmod g+w test
      -rwxrw-rw- 1 hjo users 4766 9 jan 10 13:46 test
  # chmod 766 test
  ```

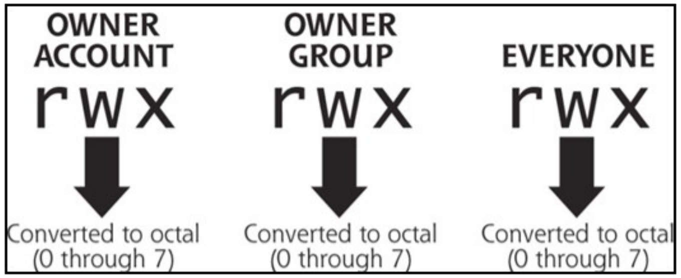
  (faster way to do it)

  – Change owner:group

  ```
  # chown hjo:hjo test
  ```


rwx rwx rwx
Permissions associated with the owner account
Permissions associated with the owner group
Permissions associated with everyone with an account on the system

Permissions Bits on Files & Directories


OWNER ACCOUNT
rwx
Converted to octal (0 through 7)
OWNER GROUP
rwx
Converted to octal (0 through 7)
EVERYONE
rwx
Converted to octal (0 through 7)

- Read controls reading the content of a file
  - i.e., the read system call
- Write controls changing the content of a file
  - i.e., the write system call
- Execute controls loading the file in memory and execute
  - i.e., the execve system call
- Many operations can be performed only by the owner of the file

- Read bit allows one to show file names in a directory
- The execution bit controls traversing a directory
  - Does a lookup, allows one to find the inode # from a file name
  - chdir(path) system call to a directory requires execution rights
- Write + execution control creating/deleting files in the directory
- Accessing a file identified by a path name requires execution to all directories along the path
- In UNIX, access rights on directories are not inherited

The setuid, setgid and sticky bits

http://en.wikipedia.org/wiki/Setuid

- Change an executable file to be setuid for a specific user
  ```
  # chmod 4755 filename
  ```
- X for owner will be changed to S as: -rw**s**rw-rw-
  ```
  # find / -uid 0 -perm -4000 -print     (find all setuid programs)
  ```
- Extreme care must be taken writing/having setuid root programs!

	Setuid value=4	**Setuid value=2**	**Setuid value=1**
Non-executable files	no effect	affect locking	not used anymore
executable files	change euid when executing the file	change egid when executing the file	not used anymore? Or let .text stay in memory (faster load)
directories	no effect	new files inherit group of the directory	only the owner of a file can rename or delete drwxrwxrwT /tmp

Process User ID Model in Modern UNIX Systems

- Each process has three user IDs
  - Real user ID (ruid) → owner of the process
  - Effective user ID (euid) → used in most access control decisions
  - Saved user ID (suid) → used for save and restore of the uid
- And three group IDs
  - Real group ID
  - Effective group ID
  - Saved group ID

http://en.wikipedia.org/wiki/User_identifier

- When a process is created by *fork*
  - It inherits all three users IDs from its parent process
- When a process executes a file by *exec*
  - It keeps its three user IDs unless the set-user-ID bit of the file is set, in which case the effective uid and saved uid are assigned the user ID of the owner of the file

Demo http://en.wikipedia.org/wiki/Setuid

```
[bob@foo]$ cat /etc/passwd
alice:x:1007:1007::/home/alice:/bin/bash
bob:x:1008:1008::/home/bob:/bin/bash
[bob@foo]$ cat printid.c

#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
int main(void) {
    printf(
        "Real      UID = %d\n"
        "Effective UID = %d\n"
        "Real      GID = %d\n"
        "Effective GID = %d\n",
        getuid (),
        geteuid(),
        getgid (),
        getegid()
    );
    return 0;
}

[bob@foo]$ gcc -Wall printid.c -o printid
[bob@foo]$ chmod ug+s printid
[bob@foo]$ su alice
Password:
[alice@foo]$ ls -l
-rwsr-sr-x 1 bob bob 6944 2007-11-06 10:22 printid
[alice@foo]$ ./printid
Real      UID = 1007
Effective UID = 1008
Real      GID = 1007
Effective GID = 1008
[alice@foo]
```

The Need for setuid/setgid Bits

- Some operations are not modeled as files and require user id = 0
  - Halting the system
  - Bind/listen on “privileged ports” (TCP/UDP ports below 1024)
  - Change password
  - Non-root users need some if these privileges
- File level access control is not fine-grained enough
- System integrity requires more than controlling who can write, but also how it is written

Security Problems of Programs with setuid/setgid

- Problem programs are typically setuid root
- Violates the least privilege principle
  - Every program and every user should operate using the least privilege necessary to complete the job
- Why is this bad?
- How would an attacker exploit this problem?
- How to solve this problem?

Password recovery with physical access in GNU/Linux (single user mode)

1. Reboot (try [Ctrl]+[Alt]+[Delete] or other harder technique)
2. During the LILO prompt (press ctrl) type: (kernel/image name) init=/bin/sh rw

   boot: linux init=/bin/bash rw
3. This should start the Linux kernel, with the root file system mounted in read/write mode. The cool thing is that none of your normal init processes (like the gettys that ask for your name and call the login program) will be started.
4. (Maybe) mount your /usr file system with a command like: # mount /usr
5. Change your root password with a command like: # passwd
6. Flush the cache buffers: # sync; sync; sync
7. (Maybe) unmount /usr: # umount /usr
8. Remount the root fs in read-only mode: # mount -o remount,ro /
9. Let init clean up and reboot the system: # exec /sbin/init 6

- Works with GRUB as well (press e in boot menus until you can edit)
  - boot: /boot/linuxkernel root=/dev/sda1 rw single init=/bin/bash
  - http://linuxgazette.net/107/tomar.html