Alternative OS – Lecture 2 – 19.09.2006

Plan:

1. History of Unix
2. Unix Structure
3. Unix File System
4. Directories, Files and Inodes
5. Unix Programs

History of Unix

1965 AT&T Bell Laboratories joins with MIT and General Electric in development effort for the new operating system, Multics. Multics was supposed to be

- multi-user,
- multi-processor,
- equipped with multi-level (hierarchical) file system
- other forward-looking features.

1969 AT&T was unhappy with the progress, and drops out of the Multics project. However, good ideas do not die easily. Some of the Bell Labs programmers who had worked on Multics project, *Ken Thompson, Dennis Ritchie, Rudd Canaday, and Doug McIlroy* designed and implemented the first version of the Unix File system on a PDP-7 computer along with a few utilities. It was given the name UNIX, by Brian Kernigan as a pun on Multics.

1970, January 1 time zero for UNIX.

1971 The system runs on a PDP-11 (16Kbytes RAM, including 8Kbytes for user programs, 512Kbyte disk).

Its fist real use was as a text processing tool for the patent department at Bell Labs. This use justified funding, so that the people got paid for the further development of UNIX.

Unix caught on among programmers because of the features it had had from the very beginning:

- programmers environment
- simple user interface (that is text mode and command line)
- simple utilities that can be combined to perform powerful functions
- hierarchical file system
- simple interface to devices, namely, devices are handled the same way as files
- Multi-user, multi-process system
- Unix is architecture independent and transparent to the user

1973 Unix is re-written mostly in C, a new language developed by Dennis Ritchie. Now Unix could be easily ported to new machines.

1974 Thompson and Ritchie publish a paper in t he Communications of the ACM, describing the new UNIX OS. Egghead computer scientists see a potentially great tool for teaching programming systems development. The 1956 Consent Decree prevented AT&T from marketing the product. Therefore, AT&T licensed UNIX to Universities for educational purposes and to commercial entities.

1977 There are now about 500 Unix sites across the world.

1980 BSD 4.1 (Berkeley Software Development) with TCP/IP support in the kernrl, Microsoft Xenix

1983 SunOS, BSD 4.2, System V

1984 There are now about 100 000 Unix sites running on many different hardware platforms, of very different capabilities.

1988 POSIX standard for Unix API. AT&T and Sun Microsystems jointly develop System V Release 4 (SVR4) This was the end of competing versions of Unix existed before. This would also later be developed into UnixWare and Solaris 2.

1993 Novell buys UNIX from AT&T, most Unix systems are based on System V with BSD features added on top.

1994 Novell gives the name "UNIX" to X/OPEN

1995 Santa Cruz Operations buys UnixWare from Novell. SCO and Hewlett-Packard announce that they will jointly develop a 64-bit version of Unix.

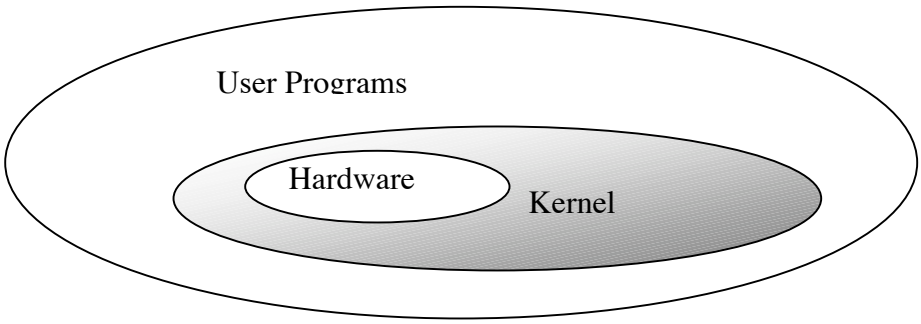
1997 There is over 3 million Unix systems shipped world-wide.

2000 SCO sold its Unix business to Caldera Systems, later renamed into SCO Group. A huge legal campaign against Linux began by this company.

2005 Sun Microsystems released the bulk of its Solaris system into an open source project called OpenSolaris.

Unix Structure

Unix is like a cake, it has layers. The deepest layer is the *Hardware*. The topmost layer is the *user programs*. These two layers never interact directly. There is a thin layer of *Kernel* between the *Programs* and *the Hardware*.



It is due to the Kernel that most well written user programs for Unix arte independent of underlying hardware. This being the biggest appeal for programmers.

User programs interact with the kernel through a set of *system calls*. These system calls request services to be provided by the kernel. Such services include:

- accessing a file (open, close, read, write, link, or execute a file),
- starting or updating accounting records,
- changing ownership of file or directory,
- changing to a new directory,
- creating (suspending, or killing) a process,
- enabling access to hardware devices,
- and setting limits on system resources.

As it was mentioned, Unix is a *multi-user*, *multi-tasking* operating system. Many users can be logged into a system simultaneously, each running many programs. Kernel keeps each process and user separate, regulates access to system hardware (cpu, memory, disc and other I/O devices).

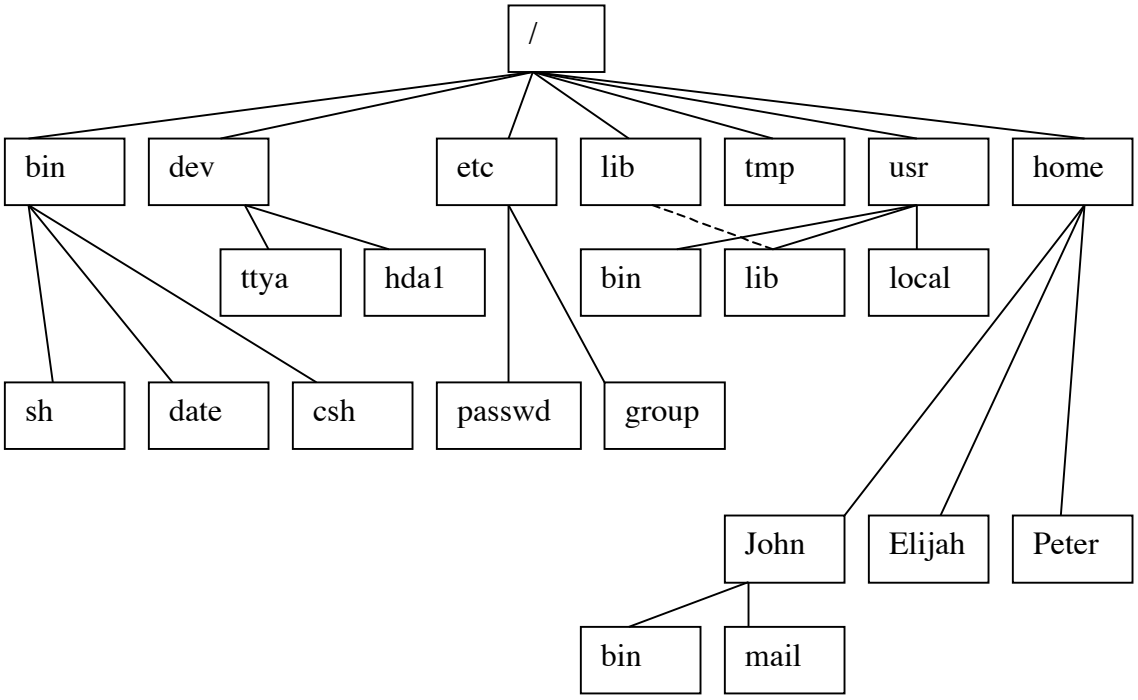
Unix File System

The Unix file system looks like a tree put upside down, with its *root* (denoted /) at the top and all subdirectories underneath it. Each node in this tree is either a *file* or a *directory* of files. Each directory can contain other files and directories. Each file or directory is uniquely specified by its *path name*. The *full* path name starts with the root /, and follows all branches of the file system, each separated by /, until you reach the desired file.

Example.

/home/john/bin

A *relative path name* specifies the path relative to another, usually the current working directory that you are at. Three special directory entries can be used to specify a relative path:



. the current directory
.. the *parent* of the current directory
~ home directory of the current user

Example.

If you are at /home then three following paths specify the same file.
~/bin
./john/bin
../home/john/bin
/home/john/bin

Directories, Files and Inodes

- Every directory and file is *listed* in its parent directory. The parent of the root is itself.
- A directory is a file containing a table listing the file names contained in the directory, and corresponding *inode* numbers. An inode is a special file designed to be read by the kernel. It specifies
    - the permissions of the corresponding file,
    - ownership,
    - date of creation,
    - date of last access,
    - date of last change,

- the physical location of the data blocks on the disc containing the file.

The file itself can be any sequence of zeroes and ones. This sequence can be interpreted as a text file, a *binary* (that is, executable), a directory table, junk, or anything else.

There is no header, trailer, label information or **EOF** character as a part of the file.

Unix Programs

A *program*, or *command,* interacts with the kernel to provide the environment and perform the functions called by the user.

A *program* can be:

- an executable shell file (shell script)
- built-in shell command
- source compiled, object code file

The *shell* is a command line interpreter. The user interacts with the kernel through the shell. One can write an ASCII text scripts to be acted upon by the shell.

System programs are usually binary, having been compiled from C source code. These are found in places like:

- /bin
- /usr/bin
- /usr/local/bin
- /usr/ucb

Some of these programs are: sh, csh, date, who, more. These programs provide the functions that are normally thought of when we think about Unix.