11 - Lecture - Introduction to UNIX

Intro to OS

What is OS?

- software that sits between hardware and other software
- core part is called the "kernel"
- ex) Windows, Linux, Mac OS

What does OS do?

- a dictator and a servant at the same time
- controls hardware resources and logical resources
- provides a (virtual) environment in which programs run
    - linear address space
    - exclusive use of CPU
    - hardware devices that responds to nice, easy commands

How does OS do that?

- privileged operations (aided by CPU)
- periodic timer interrupts
- predefined entry points into the kernel: system calls

Software organization

```
    applications: emacs, gcc, firefox, bash, mdb-lookup-cs3157
    -----------------------------------------------------------------
    library functions: printf(), strcpy(), malloc(), fopen(), fread()
    -----------------------------------------------------------------
    system calls: open(), read(), fork(), signal()
    -----------------------------------------------------------------
    OS kernel
    -----------------------------------------------------------------
    hardware: processor, memory, disk, video card, keyboard, printer
```

History of OS

1945-1970:

- vacuum tubes
- mainframes with punch cards
- IBM 360
- MULTICS

1970: Ken Thompson & Dennis Ritchie invent UNIX and C

Since then, many UNIX variants come and go including:

- AT&T System V Release 4 (SVR4)
- 4.4BSD (Berkeley Software Distribution)
- and others:
    - Microsoft Xenix
    - IBM AIX

    - HP-UX
    - IRIX

Currently, four main competitors remain:

- Linux: created by Linus Torvalds in 1991
- Solaris: SVR4-based commercial offering by Sun
- FreeBSD: based on 4.4BSD
- Mac OS X: combo of Mach kernel and FreeBSD

OS for personal computers

- 1977: CP/M by Kildall - dominant OS for 8-bit PCs
- 1977: Apple II by Steve Jobs and Steve Wozniak
- Early 80s: MS-DOS for IBM PC by Microsoft
- 1984: Apple Macintosh
- 1985-1996: NeXT by Steve Jobs
    - precursor to Mac OS X
- Late 80s & early 90s: MS Windows up to 3.11
    - shell on top of MS-DOS
- Mid 90s to the present: Windows NT, 2000, XP, 7, 8
    - true 32/64-bit OS comparable to UNIX
- 2001-present: Mac OS X

UNIX Overview

User name, User ID, Group, Permission

- every user is equal, except "root" (uid 0)
- example:

```
jae@tbilisi:˜/cs3157-pub/bin$ ls -al
total 84
drwxr-xr-x 2 jae phd  4096 2011-10-26 00:06 .
drwxr-xr-x 7 jae phd  4096 2011-10-25 23:35 ..
-rwxr-xr-x 1 jae phd 16740 2011-10-25 23:47 mdb-add
-rwsr-xr-x 1 jae phd 16755 2011-10-25 23:58 mdb-add-cs3157
-rw-r--r-- 1 jae phd  5480 2011-10-29 16:58 mdb-cs3157
-rwxr-xr-x 1 jae phd 20905 2011-10-25 23:47 mdb-lookup
-rwxr-xr-x 1 jae phd    83 2011-10-26 00:06 mdb-lookup-cs3157
jae@tbilisi:˜/cs3157-pub/bin$
```

File system

- single root directory: "/"
- relative path v. absolute path
- everything is a file: even a directory, even a hardware device!

UNIX I/O using file descriptors

- file descriptors are small integers representing open files
- when a program starts, kernal opens 3 files without being asked
    - stdin, stdout, stderr on descriptors 0, 1, 2
    - keyboard, screen, screen unless redirected
    - subsequent open files get 3,4,5,6,...

- unbuffered
- file descriptors are used for sockets too

- example:

```
int fd = open("myfile", O_RDONLY, 0);
if (fd == -1) {
        // open error
}

int n;
char buf[BUF_SIZE];
while ((n = read(fd, buf, BUF_SIZE)) > 0)
        if (write(1, buf, n) != n) {
                // write error
        }
if (n < 0) {
        // read error
```

Processes

- program v. process
- process ID: getpid()

- a process is created by fork & exec by an existing process
- example:

```
// NOTE: this is pseudo-code

......

if ((pid = fork()) < 0) { // "called once, returns twice"
        die("fork err");

} else if (pid == 0) {
        // comes here in child process
        exec("ls");
        die("exec err");

} else {
        // comes here in parent process
        ......
```

- kernel starts "init" process, which in turn starts various login managers: getty, xdm, sshd, etc.
- ps command: try "ps auxfww"

Signals (optional topic)

- OS’s way of telling a process something happened. For example:
    - user pressed crtl-c
    - you did something wrong: divide by zero, illegal memory access, etc.
    - one of your child process has quit
    - etc, etc.

- it can come anytime; a process can either:

- let the default action take place
- explicitly ignore the signal (not always possible)
- catch the signal and do your own thing (not always possible)

- you can generate signal: "kill" command or kill() function

- example:

```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <unistd.h>

static void sig_int(int signo)
{
    printf("stop pressing ctrl-c!\n");
}

int main()
{
    if (signal(SIGINT, &sig_int) == SIG_ERR) {
        perror("signal() failed");
        exit(1);
    }

    int i = 0;
    for (;;) {
        printf("%d\n", i++);
        sleep(1);
    }
}
```