CHAPTER I: INTRODUCTION TO THE UNIX OPERATING SYSTEM

B. Boufama

University of Windsor

Contents

Introduction?

What are the objectives of this course?

- Introduce you to Operating Systems
- Familiarize the students with Unix and systems programming
- Improve your C programming skills.

Operating Systems

Recall : A computer hardware cannot function without software.
In particular, a computer system must possess an operating system.
Example of services provided by an operating system :

- provide a framework for executing programs,
- share system resources (CPU, memory, disk) among programs and users,
- allow communication with devices(monitor, keyboard, network, etc.) and other programs,
- open a file, read from a file,
- get the time of the day, etc.

A tour of UNIX : services and features

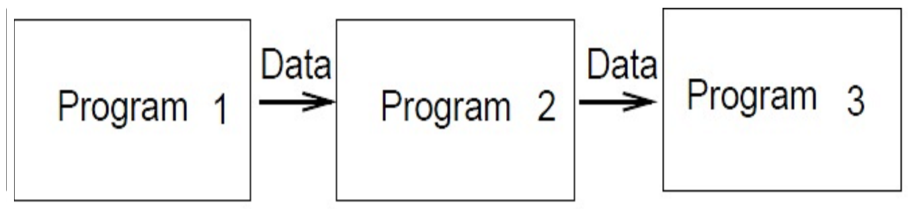
There are many operating systems, however, only UNIX is available for all platforms : PCs, minis and mainframes computers.
UNIX is among few operating systems that allows more than one user to share a computer system at a time $\longrightarrow$ UNIX is best choice for big businesses.

Unix has a simple philosophy :

- a program(utility) should do one thing and do it well
- a complex problem should be solved by combining multiple existing utilities.

$\longrightarrow$ UNIX achieves this gaol using pipes.

Pipes : a mechanism that allows the user to specify that the output of one program is to be used as the input of another program.
$\longrightarrow$ several programs can be connected in this fashion to make a pipeline of data flowing from the first process through the last one.



Example :

ps -e | grep netscape | more
where the three commands mean :
ps -e: report status on every process now running
grep : search a file for a pattern
more : page through a text file

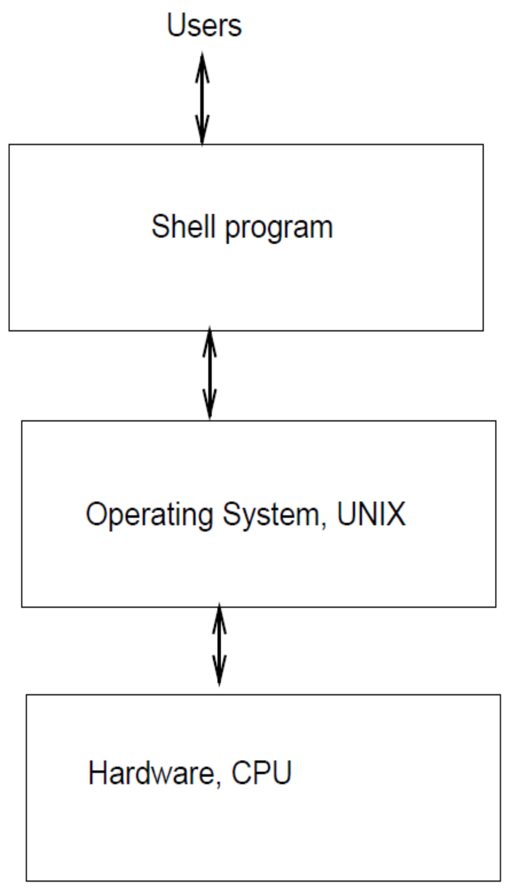
Login name and passwd:
Unix check our login name in **/etc/passwd** and match our password with the one in the encripted file **/etc/shadow**.

The file /etc/shadow can only be read by *super users.*

Shells : A *shell* is a command line interpreter that reads and executes user commands.
A shell reads user inputs either from a terminal or, from a file called *script file.* Example of shells:

- the Bourne shell, /bin/sh
- the C shell, /bin/csh
- the Korn shell, /bin/ksh

Important : Unix is case sensitive.


Users
Shell program
Operating System, UNIX
Hardware, CPU

Files and Directory:

The Unix file system is a hierarchical arrangement of files and directories.
The *root* directory is represented by the slash character (/).
A directory is a file that contains entries for files and directories.

When a new directory is created, two filenames are automatically created :

- **.** (called *dot*) that refers to the current directory
- **..** (called *dot-dot*) that refers to the parent directory.

A *pathname* is made of filenames separated by slashes. If a *pathname* starts with a / then it is an *absolute pathname*, otherwise, it is a *relative* one.
Examples:
/export/home/users/zebra/set.txt(absolute)
images/city/park.jpg (relative)

A bare bones implementation of the *ls* command.

```
#include <sys/types.h>
#include <dirent.h>
#include <stdio.h>

int main(int argc, char *argv[]){
  DIR *dp;
  struct dirent *dirp;

  if(argc==1)
    dp = opendir("./");
  else
    dp = opendir(argv[1]);

  while ( (dirp=readdir(dp)) != NULL)
    printf("%s\n", dirp->d_name);

  closedir(dp);
  exit(0);
}
```

Standard Input, Standard Output, and Standard Error

Whenever a new program is run, three file descriptors are opened :

- the standard input *stdin*, by default the keyboard,
- the standard output *stdout*, by default the monitor,
- the standard error *stderr*, by default the monitor.

Shells provide means to redirect standard input and standard output.
For example :

- ls > ls.outputm, the outputs are stored in the newly created file ls.output instead of the monitor,.
- mailx admin256@uwindsor.ca < assignment1.c, the inputs for mailx come from the file assignment1.c instead of the keyboard.

Example :

```
#include <stdio.h>

int main(int argc, char *argv[]){
  FILE *fd;
  char c;

  if(argc==1)
    fd=stdin;
  else
   if((fd = fopen(argv[1], "r"))==NULL){
     fprintf(stderr,"Cannot open %s\n", argv[1]);
     exit(0);
   }
  while( (c=getc(fd)) != EOF)
    putc(c, stdout);

  exit(0);
}
```

Question : why use *stderr* instead of *stdout*?

Programs and Processes

- A *program* is an executable file residing on a disk.
- A *process* is an executing (running) program, usually with a limited life-time.
  Note that sometimes a process is called *task*.
- A *process ID (PID)* is a unique nonnegative integer assigned by Unix, used to identify a process.

Example of the *ps* command output:

```
  PID  TTY       TIME CMD
10258  pts/6     0:01 gs
 7478  pts/6     0:06 emacs-20
 5598  pts/6     0:01 csh
10184  pts/6     0:01 Chrome
```

ps reports process status.

A process can obtain its PID by the getpid() system call.
A process can also obtain its parent ID PPID by the getppid() system call.

```
#include <stdio.h>

int main(int argc, char *argv[]){

  printf("Hello, my PID is %d\n", getpid());
  printf("Hello, my PPID is %d\n", getppid());
  exit(0);
}

Shell-Prompt> a.out
Hello, my PID is 11723
Hello, my PPID is 5598
```

Process Control

There are three primary functions (system calls) for process control:

- *fork*: allows an existing process to create a new process which is a copy of the caller
- *exec*: allows an existing process to be replaced with a new one.
- *wait*: allows a process to wait for one of its child processes to finish and also to get the termination status value.

Note: The mechanism of spawning new processes is possible with the use of *fork* and *exec*.

User Identification

User ID : Each user is assigned a user ID, a unique nonnegative integer called uid.

```
oconnel:x:1003:10:David O'Connell, M.Sc. student:/users/oconnel:/bin/tcsh
```

The user ID is used by the kernel to check if the user has the appropriate permissions to perform certain tasks like accessing files, etc.
The user *root* (*superuser*) has uid=0. This user has special privileges, like accessing any file on the system.

A process can obtain its uid using the system call getuid().

```
#include <stdio.h>
int main(void){
printf(``hello, my uid is %d\n'', getuid());
}
```

Group ID : a positive integer allowing to group different users into different categories with different privileges.
A group ID (*gid*) is also used by Unix for permissions verifications.

Note that both the user ID and the group ID are assigned by the system administrator at the time of the creation of the user account.
There is a group file, /etc/group, that maps group names into numeric IDs.

Signals

Signals are used to notify a process of the occurrence of some condition.
For example, the following generate signals:

- A division by zero: the signal *SIGFPE* is sent to the responsible process that has three choices. Ignore the signal, terminate the process or, call a function to handle the situation.
- The *Control-C* key: when pressed, it generates a signal that causes the process receiving it to interrupt.
- Calling the function *kill*: a process can send a signal to another process causing its death.
  This is an example where Unix checks our permissions before allowing the signal to be sent.

Unix Time values

The execution time of a process can be measured with three values :

- Clock time : amount of time the process takes to run. This is more like the *wall clock time* for the process.
- User CPU time : the time of the CPU used on the process instruction.
- System CPU time : CPU time attributed to the kernel, when it executes instructions on behalf of the process, for instance, a disk reading.

The sum of the user CPU time and system CPU time is often called the *CPU time*.

The *time* command can be used to measure the clock time, the user time and, system time.
Examples:

```
/usr/bin/time -p gcc myLs.c -o myLs

real 0.60
user 0.14
sys 0.07
```

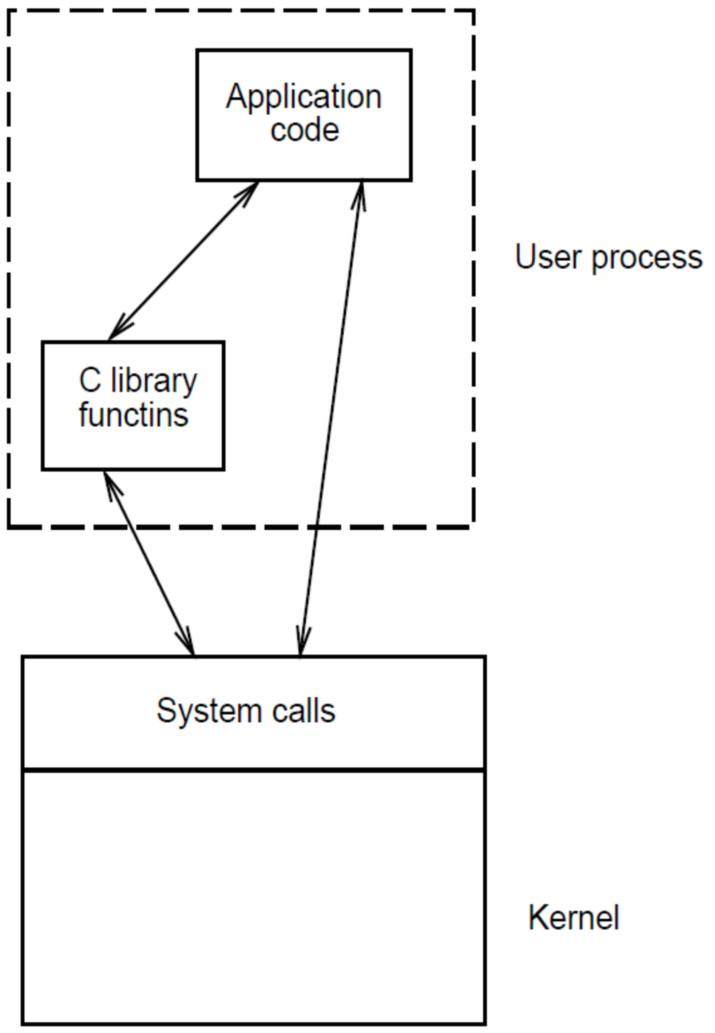
In that case, the CPU time is 0.14 + 0.07 = 0.21 second.

System calls and library functions

Operating systems provide entry points for programs to request services from the kernel.
Entry points are called system calls in Unix.
In particular :

- Each system call in Unix has an interface function, in the C standard library, with the same name that the user process invokes.
- The interface function then invokes the appropriate kernel service, using whatever technique is required on the system.
- An interface function for a system call cannot be replaced, however, a library function, such as *strcpy*, can be rewritten by the user.
- For our purpose, a system call will be viewed as a regular C function.

Note that a user process can invoke either a system call or a library function.
A library function might invoke a system call.



Summary

- Operating system: Software to manage computer resources, in particular,
  - it runs a program for a user
  - it allows communication with devices and processes
- A program is a file contaning instructions
- A process is a program being executed
- Unix is a multiuser operating system
- Most of Unix is written in the C language
- The Unix philosophy is simple: a program should do one thing and do it well.
- Entry points in Unix are called system calls. They allow the user to get services from the kernel.