TOPICS IN SOFTWARE SYSTEMS

Chapter 1: Introduction to system programming

Sembatya Erios Naiga

2018/9/20

CONTENTS

1. Introduction
2. Unix architecture
3. Logging in
4. Files and directories
5. Input and output
6. Programs and processes
7. Error handling
8. User Identification
9. Signals
10. Time values
11. System Calls and Library Functions

INTRODUCTION

1. Familiarize with UNIX
2. Fundamentals and Operating system concepts
    2.1 Multi-user concepts
    2.2 Basic and advanced I/O
    2.3 Process
3. Interprocess communication

UNIX ARCHITECTURE

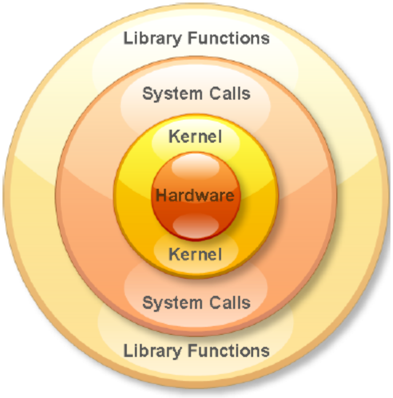


Figure 1: Architecture of UNIX operating system

UNIX ARCHITECTURE

Kernel is a software (SW) that controls the hardware (HW) of the computer and provides an environment under which programs can run (Figure 1).

System calls: are entry points inti which kernel codes where the functions are implemented.

Library calls (functios): are transfers to user code with which performs the desired functions.

An **operating system (OS)** consists of the kernel and all other SWs that make a computer useful and gives the computer its personality. Other SW include system utilities, applications, shells, libraries of common functions etc. for example, for linux, kernel used by GNU OS is GNU(sometimes referred to as GNU linux OS) has an advantage of being more siccunt.

LOGGING IN

After log in, the system looks up for your login in its password file (/etc/passwd) which is composed of 7 colon separated fields (login name, encrypted password, user ID 205), numeric group ID (105), a comment field, home directory (/home/sar) and shell program (/bin/ksh). That is, sar: x: 205: 105: Sembaty Erios: home/sar: /bin/ksh. (chapter 6).

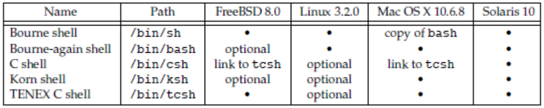
Shell: is a command-line interpreter that reads user input and executes commands. It is a special application that provides an interface for running other applications.

After log in, you type commands to the shell and the system knows which shell to execute for us based on the field in our entry password file.

LOGGING IN

The system knows which shell to execute for us based on the final field in our entry in the password file.

There are 5 common shells used (chapter 2).



Name	Path	FreeBSD 8.0	Linux 3.2.0	Mac OS X 10.6.8	Solaris 10
Bourne shell	`/bin/sh`	•	•	copy of `bash`	•
Bourne-again shell	`/bin/bash`	optional	•	•	•
C shell	`/bin/csh`	link to `tcsh`	optional	link to `tcsh`	•
Korn shell	`/bin/ksh`	optional	optional	•	•
TENEX C shell	`/bin/tcsh`	•	optional	•	•

Figure 2: Common shells used on UNIX systems.

FILES AND DIRECTORIES

NOTE: The research Unix system and some older Unix system v-file systems filenames are restricted to 14 characters. BSD versions – extended limit of 255 characters and almost all commercial Unix file systems support 255 character filenames.

Pathname: A sequence of 1 or more filenames separated by slashes (relative pathname) and starting with a slash (an absolute pathname). Relative pathname files are relative to the current directory.

FILES AND DIRECTORIES

Working directory: every process has a working directory (current working directory) which interprets all relative pathnames. A process can change its directory with the chdir function.

Home directory: after log in the working directory is set to our home directory which is obtained from our entry in the password.

INPUT AND OUTPUT

File descriptors: are small non-negative integers that the kernel uses to identify the files accessed by a process. Whenever the kernel opens an existing file or creates a new file, it returns a file descriptor that we use when we want to read or write a file.

Standard input, standard output and standard error: usually, all shells provide a way to redirect and/or all the descriptors to any file. If nothing is done, like a simple command 1s, then all d 3 are connected to the terminal (chapter 18). For example; s¿file.list executes the 1s command with its standard output redirected to the file named file.list (chapter 5).

PROGRAMS AND PROCESSES

Program is an executable file residing on disk in a directory. A program is read into memory and is executed by the kernel as a result of one of the seven exec functions (Section 8.10).

Processes and Process ID:

A process (task): An executing instance of a program. The UNIX System guarantees that every process has a unique numeric identifier (is always a non-negative integer) called the process ID.

PROGRAMS AND PROCESSES

Program is an executable file residing on disk in a directory. A program is read into memory and is executed by the kernel as a result of one of the seven exec functions **(Section 8.10).**
Processes and Process ID:

A process (task): An executing instance of a program.

The UNIX System guarantees that every process has a unique numeric identifier (is always a non-negative integer) called the **process ID.**

PROGRAMS AND PROCESSES

Process Control: There are three primary functions for process control: fork, exec (has seven variants), and waitpid. Process control of UNIX system is demonstrated by the bare-bornes implementation of the shell-like program (chapter 8).

PROGRAMS AND PROCESSES

Threads: A thread is a set of machine instructions executing at a time. A process usually has only 1 thread of control. Some problems are easy to solve when more than 1 thread of control operates on different parts of the problem. Multiple threads of control can exploit the parallelism possible on multiprocessor systems.

All threads within a process share the same address space, file descriptors, stacks and process-related attributes. Each thread executes on its own stack but any thread can access stacks of other threads in the process. They need to synchronize access to share data amongst themselves to avoid inconsistent.

PROGRAMS AND PROCESSES

Thread ID: It functions to control threads parallel to other threads used to control a process. Threads are localized to a process (chapter 12). A thread ID from one process has no meaning in another process. We use thread IDs to refer to specific threads as we manipulate the threads within a process.

ERROR HANDLING

Occurrence of an error in UNIX system functions returns a negative value and the integer errno is usually set to a value that notifies you. Eg, open function returns either a non-negative file descriptor in all to indicate all is ok or -1 for error. That is; more functions that return a pointer to an object return a null pointer to indicate an error. An error from open has about 15 possible errno values, say, file does not exist, permission problem (section 2).

On LINUX, the error constants are listed in the errno (3) manual page. In an environment that supports threads, the process address space is shared by amongst multiple threads with each thread having its own local copy of errno to prevent them from interfering with each other.

ERROR HANDLING

There are 2 rules to be aware of with respect to errno.

1. Its value is never cleared by a routine if an error does not occur. Thus, we should examine its value only when the return of errno is never set to 0 by any of the functions and no constants defined in errno.h has a 0 value.

2. The value of errno is never set to 0 by any of the functions and none of the constants defined in ¡errno.h¿ has a value 0.

ERROR RECOVERY

Fatal error has no recovery action. Where you only print an error message on the user 's screen or to a log file, and then exit.
Nonfatal errors can are temporary, such as a resource shortage, and occur when there is less activity on the system.
Resource-related nonfatal errors:- EAGAIN, ENFILE, ENOBUFS, ENOLCK, ENOSPC, EWOULDBLOCK, and sometimes ENOMEM. EBUSY can be treated as nonfatal when it indicates that a shared resource is in use as well as EINTR when it interrupts a slow system call.

The recovery action for a resource-related nonfatal error is to delay and retry later. Eg, network connection is no longer functioning. Some applications use an exponential back off algorithm, which takes a long time in each subsequent iteration and is dependent on the developer.

USER IDENTIFICATION (ID)

The **user ID** from our entry in the password file is a numeric value that identifies us to the system. User whose user ID is 0 either root or the superuser. The entry in the password file normally has a login name of root and the special privileges of this user as superuser privileges (chapter 4).

Group ID: the password file contains multiple entries specifying the same group ID. Groups are normally used to collect users together into projects or departments and allows sharing of resources, such as files, among members of the same group. The group file (/etc/group) maps group names into numeric group IDs.

For every file on disk, the file system stores both the user ID and the group ID (which values requires only four bytes) of a file's owner assuming that each is stored as a two-byte integer.

SUPPLEMENTARY GROUP ID

These supplementary group IDs are obtained at login time by reading the file /etc/group and finding the first 16 entries that list the user as a member (chapter 2).

Most versions of the UNIX System allow a user to belong to other groups (it started with 4.2BSD, at least 16 additional groups).

SIGNALS

Are a technique notifies a process that some condition has occurred. For example, if a process divides by zero, the signal whose name is SIGFPE (floating-point exception) is sent to the process (Chapter 10).

The process has **3 choices for dealing with the signal;-**

1. Ignore the signal. Not recommended for signals that denote a hardware exception, such as dividing by zero or referencing memory outside the address space of the process, as the results are undefined.

2. Let the default action occur. For a divide-by-zero condition, the default is to terminate the process.

3. Provide a function that is called when the signal occurs (called "catching" the signal).

SIGNALS

Interrupt key—(DELETE key or Control-C —) and the **quit key**—(Control-backslash —): used to interrupt the currently running process. Thus to generate a signal is by calling the kill function (from a process to send a signal to another process).

Limitations to call a function : you must be the owner of the other process (or the superuser) to be able to send it a signal.

TIME VALUES

UNIX systems have **maintained two different time values:**
Calendar time: This value counts the number of seconds since the Epoch: 00:00:00 January 1, 1970, Coordinated Universal Time (UTC). (Older manuals refer to UTC as Greenwich Mean Time.) These time values are used to record the time when a file was last modified.

Process time (CPU time): measures the central processor resources used by a process and is measured in clock ticks (have historically been 50, 60, or 100 ticks per second)**(section 2.5.4)**.

TIME VALUES

Measurement of the execution time of a process (Section 3.9): UNIX System maintains three values for a process:

Clock time (wall clock time) - amount of time the process takes to run, and its value depends on the number of other processes being run on the system and the measurements are made with no other activities on the system.

User CPU time - CPU time attributed to user instructions, that is; attributed to the kernel when it executes on behalf of the process.

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

TIME VALUES

Measurement of the clock time, user time, and system time of any process:

simply execute the time(1) command, with the argument to the time command being the command we want to measure.

Note: The output format from the time command depends on the shell being used, because some shells don't run /usr/bin/time, but instead have a separate built-in function to measure the time it takes commands to run (Section 8.17).

SYSTEM CALLS AND LIBRARY FUNCTIONS

An application can either make a system call or call a library routine but library routines invoke a system call **Figure 3.**

The exact number of system calls varies depending on the operating system version (Section 2 of the UNIX Programmer 's Manual defines the general-purpose library functions available to programmers). Linux 3.2.0 has 380 system calls and FreeBSD 8.0 has over 450. The technique used on UNIX systems is for each system call to have a function of the same name in the standard C library. The user process calls this function, using the standard C calling sequence. Which invokes the appropriate kernel service.

SYSTEM CALLS AND LIBRARY FUNCTIONS

An application can either make a system call or call a library routine but library routines invoke a system call (Figure 3). We will use the term **function** to refer to both system calls and library functions, except when the distinction is necessary.

The exact number of system calls varies depending on the operating system version **(check Section 2 of the UNIX Programmer's Manual defines the general-purpose library functions available to programmers).** Eg, Linux 3.2.0 has 380 system calls and FreeBSD 8.0 has over 450. The technique used on UNIX systems is for each system call to have a function of the same name in the standard C library. The user process calls this function, using the standard C calling sequence which invokes the appropriate kernel service.

SYSTEM CALLS AND LIBRARY FUNCTIONS

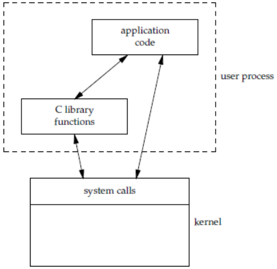
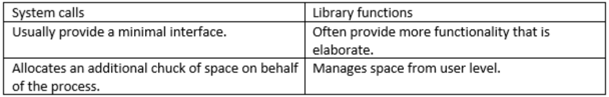


Figure 3: Difference between C library functions and system calls.



System calls	Library functions
Usually provide a minimal interface.	Often provide more functionality that is elaborate.
Allocates an additional chuck of space on behalf of the process.	Manages space from user level.

Figure 4: some of the differences between C library functions and system calls.

SYSTEM CALLS AND LIBRARY FUNCTIONS

From **an implementer's point of view,** the distinction between a system call and a library function is fundamental. Unlike **a user**, for the implementer, both system calls and library functions appear as normal C functions. As both exist to provide services for application programs. We can replace the library functions whereas the system calls usually cannot be replaced.

Section 8.13 - show an implementation of the system function that invokes the basic process control system calls with **an example in Section 10.18** to handle signals correctly.