UNIT – I

Introduction to UNIX and its Commands

Contents …

- UNIX and ANSI Standards:
- The ANSI C Standard, The ANSI/ISO C++ Standards,
- Difference between ANSI C and C++,
- The POSIX Standards, The POSIX.1 FIPS Standard.
- UNIX and POSIX APIs: The POSIX APIs, The UNIX and POSIX Development Environment,
- API Common Characteristics,
- The File System: The File, What's in a (File)name,
- The Parent-Child relationship,
- The UNIX File System,
- pwd, Absolute pathnames, cd, Relative pathnames, mkdir, rmdir, cp, rm, mv, cat, ls.

The ANSI C Standard

- Developed by Ken Thomson and Dennis Ritchie

- It was developed in 1960's.
- Many features were being added, that led to multiple versions of UNIX.
- As a result, system developers found it difficult to write different application for different versions of UNIX.
- This problem, led to development of two standards in 1980's.
  - ANSI C
  - POSIX
- This standards provided uniform set of libraries and APIs for all conforming operating systems.
- Standards define operating system environment for C based applications for application programmer for system calls and library

- Standards also define signatures

UNIX and ANSI Standards:

- The ISO (International Standards Organization) defines “standards are documented agreements containing technical specifications or other precise criteria to be used consistently as
  - rules,
  - guidelines or definitions of characteristics to ensure that
  - materials, products, processes and services are fit for their purpose”.
- Most official computer standards are set by one of the following organizations:
- ANSI (American National Standards Institute)
- ITU (International Telecommunication Union)
- IEEE (Institute of Electrical and Electronic Engineers)
- ISO (International Standards Organization)
- VESA (Video Electronics Standards)

The ANSI C Standard

- This standard was proposed by
- American ANSI in the year 1989 for C programming Language Standard called X3.159-1989
- To standardize the C programming language constructs and libraries.

Contents …

- UNIX and ANSI Standards:
- The ANSI C Standard, The ANSI/ISO C++ Standards,
- Difference between ANSI C and C++,
- The POSIX Standards, The POSIX.1 FIPS Standard.
- UNIX and POSIX APIs: The POSIX APIs, The UNIX and POSIX Development Environment,
- API Common Characteristics,
- The File System: The File, What’s in a (File)name,
- The Parent-Child relationship,
- The UNIX File System,
- pwd, Absolute pathnames, cd, Relative pathnames, mkdir, rmdir, cp, rm, mv, cat, ls.

Major differences between ANSI C and K & R C

- ANSI C supports Function Prototyping
- ANSI C support of the const & volatile data type qualifier
- ANSI C support wide characters and internationalization, Defines setlocale function
- ANSI C permits function pointers to be used without dereferencing

1. Function prototyping

- ANSI C adopts C++ function prototype technique where function definition and declaration include function names, arguments' data types, and return value data types.
- This enables ANSI C compilers to check for function calls in user programs that pass invalid number of arguments or incompatible arguments' data type.
- These fix a major weakness of K&R C compilers: invalid function calls in user programs often pass compilation but cause programs to crash when they are executed.
- **Eg: unsigned long demo(char * fmt, double data)**
- **{**
- **/*body of demo*/**
- **}**
- External declaration of this function demo is
- **unsigned long demo(char * fmt, double data);**

- **eg: int printf(const char* fmt,...........);** specify variable number of arguments

K & R C

```
#include<stdio.h>
int main( )
{
  int sum;
sum = add(5,6);
Printf(“Sum = %d”,sum);
return 0;
}
int add(a,b);
{
int a, int b;
return(a + b);
}
```

ANSI C

```
#include<stdio.h>
int add(int, int); //function prototyping
int main( )
{
  int sum;
sum = add(5,6);
Printf(“Sum = %d”,sum);
return 0;
}
int add(a,b);
{
int a, int b;
return(a + b);
}
```

2. Support of the const and volatile data type qualifiers

- The **const** keyword declares that some data cannot be changed.

- Eg: **int printf(const char* fmt,..........);**

- Declares a fmt argument that is of a const char * data type, meaning that the function printf cannot modify data in any character array that is passed as an actual argument value to fmt.

Look at the example first!!!

K & R C

```
#include<stdio.h>
int main( )
{
int a,i;
 for (i=0;i<10;i++)
{
 a = 5;
Printf(“ I have %d books”,a);
}
return 0;
}
```

- The loops executes 10 times.
- But each time value of a=5.
- Which is waste of execution.
- Hence, compiler logically removes the statement a=5 to improve efficiency.
- This is done by optimization algorithm.

2. Support of the const and volatile data type qualifiers

- Volatile keyword specifies that the values of some variables may change asynchronously, giving an hint to the compiler’s optimization algorithm not to remove any “redundant” statements that involve “volatile” objects.

ANSI C

```
#include<stdio.h>
int main( )
{
volatile int a,i;
 for (i=0;i<10;i++)
{
 a = 5;
Printf(“ I have %d books”,a);
}
return 0;
}
```

3. Support wide characters and internationalization

- ANSI C supports internationalisation by allowing C-program to use wide characters. Wide characters use more than one byte of storage per character.
- ANSI C defines the **setlocale** function, which allows users to specify the format of date, monetary and real number representations For eg: most countries display the date in dd/mm/yyyy format whereas US displays it in mm/dd/yyyy format.
- Function prototype of setlocale function is: #include<locale.h>

- char setlocale(int category, const char* locale);

3. Support wide characters and internationalization

- The setlocale function prototype and possible values of the category argument are declared in the <locale.h> header.

- The category values specify what format class(es) is to be changed. Some of the possible values of the category argument are:

Category Value		Effect on standard C functions/macros
LC_CTYPE	⇒	Affects behavior of the <ctype.h> macros
LC_TIME	⇒	Affects date and time format.
LC_NUMERIC	⇒	Affects number representation format
LC_MONETARY	⇒	Affects monetary values format
LC_ALL	⇒	combines the affect of all above

Eg:

setlocale(LC_ALL, “C”);

SETLOCALE

```
#include <locale.h>
Char setlocale (int    category, const char* locale);
```

Category	*Locale*	
LC_CTYPE	en_US	//US
LC_MONETARY	fr_FR	//French
LC_NUMERIC	de_DE	//German
LC_TIME	C	
LC_ALL	POSIX	

4. Permit function pointers to be used without dereferencing

- ANSI C specifies that a function pointer may be used like a function name. No referencing is needed when function whose address is contained in the calling a pointer.

- For Example:
- extern void foo(double xyz,const int *ptr);
- void (*funptr)(double,const int *)=foo;

- The function can be called directly or through function pointer as given below:
- foo(12.78,"Hello world");
- funptr(12.78,"Hello world");

- K& R C requires funptr be dereferenced to call foo:
- (* funptr) (13.48,"Hello usp");

- ANSI C also defines a set of CPP symbols which may be used in user programs

STDC : Feature test macro Value is 1 if underlying system is ANSI C compliant , 0 Otherwise

LINE : Physical line number of the module

FILE : filename of module where the symbol is present

DATE : date of compilation of the module

TIME : time of compilation of the module

PROGRAM TO ILLUSTRATE THE USE OF THESE SYMBOLS

```
#include <stdio.h>
      int main()
       {
         #if __STDC__ == 0 && !defined(__cplusplus)
             printf("cc is not ANSI C compliant\n");
          #else
          printf(" %s compiled at %s:%s. This statement is
                 at   line    %d\n",
           __FILE__, __DATE__, __TIME__, __LINE__);
          #endif
                  return 0;
      }
```

Contents …

- UNIX and ANSI Standards:
- The ANSI C Standard, The ANSI/ISO C++ Standards,
- Difference between ANSI C and C++,
- The POSIX Standards, The POSIX.1 FIPS Standard.
- UNIX and POSIX APIs: The POSIX APIs, The UNIX and POSIX Development Environment,
- API Common Characteristics,
- The File System: The File, What’s in a (File)name,
- The Parent-Child relationship,
- The UNIX File System,
- pwd, Absolute pathnames, cd, Relative pathnames, mkdir, rmdir, cp, rm, mv, cat, ls.

Difference between ANSI C & C++

ANSI C	ANSI C++
- Uses default prototype if called before declaration or defn	- Requires that all functions must be declared and defined before the can be referenced.
-int foo() is same as int foo(…)	-int foo() is same as int foo(void)
-no type safe linkage	-encryptes all external function names for type safe linkage (ld reports error)

Contents …

- UNIX and ANSI Standards:
- The ANSI C Standard, The ANSI/ISO C++ Standards,
- Difference between ANSI C and C++,
- The POSIX Standards, The POSIX.1 FIPS Standard.
- UNIX and POSIX APIs: The POSIX APIs, The UNIX and POSIX Development Environment,
- API Common Characteristics,
- The File System: The File, What's in a (File)name,
- The Parent-Child relationship,
- The UNIX File System,
- pwd, Absolute pathnames, cd, Relative pathnames, mkdir, rmdir, cp, rm, mv, cat, ls.

THE POSIX STANDARDS

- Because of many UNIX vendors ,each UNIX version provide its own set of API's
- IEEE society formed a special task force called POSIX.
- POSIX.1a : Known as IEEE 1003.1-1990 standard adapted by ISO as ISO/IEC 9945:1:1990 standard
  - gives standard for base operating system i.e for files and processes
- POSIX.1b : Known as IEEE 1003.4-1993
  * gives standard APIs for real time operating system interface which includes interprocess communication

- POSIX.1c : specifies multi threaded programming interface

- Other POSIX compliant systems

  -VMS of DEC

  -OS/2 of IBM

  -Windows -NT of Microsoft

  -Sun solaris 2.x

  -HP-UX 9.05

- **To ensure program confirms to POSIX.1 standard user should define**

  _POSIX_SOURCE as

1. **#define _POSIX_SOURCE OR**

2. **CC -D _POSIX_SOURCE *. C**

_POSIX_C_SOURCE : ITS VALUE INDICATING POSIX VERSION

- **_POSIX_C_SOURCE value----Meaning**

  198808L----   First version of POSIX.1 compliance

  199009L---- Second version of POSIX.1 compliance

  199309L---- POSIX.1 and POSIX.1b compliance

Program that shows the POSIX version

```
#define _POSIX_SOURCE
#define _POSIX_C_SOURCE 199309L
#include <iostream.h>
#include <unistd.h>
int main()
{
    #ifdef _POSIX_VERSION
        cout << "System conforms to POSIX: " <<    _POSIX_VERSION << endl;
    #else
       cout << "_POSIX_VERSION is undefined\n";
    #endif
    return 0;
}
```

POSIX ENVIRONMENT

- **Difference between POSIX and UNIX**

  In UNIX C and C++ header files are included in /usr/include

  In POSIX they are just headers not header files and /usr/include need not be a physical file present

- **UNIX – Superuser has special privilege and the superuser ID is always 0**

  POSIX – Does not mandate the support for the concept of superuser nor the ID is 0 requires special privilege

THE POSIX FEATURE TEST MACROS

- **Some UNIX features are optional to be implemented on POSIX-confirming systems**

- **_POSIX_JOB_CONTROL**
  - **— The system supports BSD type job control**

- **_POSIX_SAVED_ID**
  - **— keeps saved set-UID and set-GID**

- **_POSIX_CHOWN_RESTRICTED**

  — If -1 user may change ownership of files owned by them else only users with special privilege can do so

- _POSIX_NO_TRUNC
  - If -1 then any long path name is automatically truncated to NAME_MAX else an error is generated
- _POSIX_VDISABLE
  - — If -1 then there is no dissabling character for special characters for all terminal devices otherwise the value is the disabling character value

Program to print POSIX-defined configuration options supported on any given system

```
#define _POSIX_SOURCE
#define _POSIX_C_SOURCE     199309L
#include <iostream.h>
#include <unistd.h>

int main()
{
#ifdef _POSIX_JOB_CONTROL
   cout << "System supports job control\n";
#else
   cout << "System does not support job control\n";
#endif
```

```
#ifdef _POSIX_SAVED_IDS

   cout << "System supports saved set-UID and saved
                    set-GID\n";
#else

      cout << "System does not support saved set-UID
                    and saved set-GID\n";
#endif
```

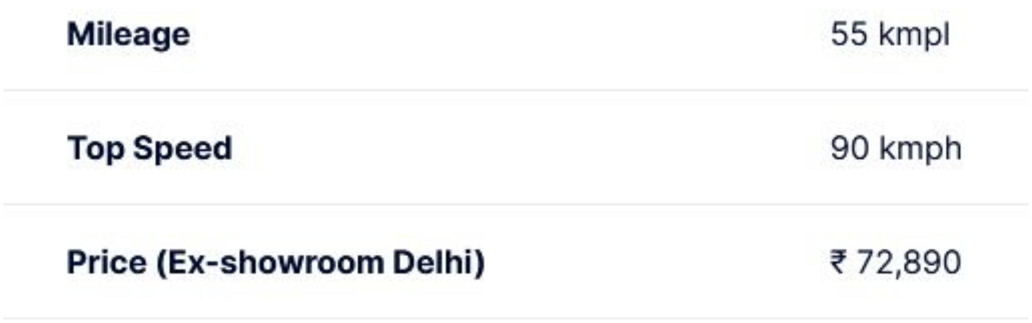
```
#ifdef _POSIX_CHOWN_RESTRICTED

    cout << "chown restricted option is: " <<
          _POSIX_CHOWN_RESTRICTED <<endl;
#else

     cout << "System does not support system-wide
                chown_restricted option\n";
#endif
```

```
#ifdef _POSIX_NO_TRUNC

    cout << "Pathname trucnation option is: " <<
                _POSIX_NO_TRUNC << endl;
#else

   cout << "System does not support system-wide
                pathname trucnation option\n";

#endif

}
```

```
#ifdef _POSIX_VDISABLE

      cout << "Diable character for terminal files is: "
               << _POSIX_VDISABLE << endl;
#else

    cout << "System does not support
                    _POSIX_VDISABLE\n";
#endif

    return 0;
```

COMPILE-TIME VALUES & RUN-TIME VALUES

- Consider you want to buy a bike!!!!!



Mileage	55 kmpl
Top Speed	90 kmph
Price (Ex-showroom Delhi)	₹ 72,890

- **Does your bike gives the same result??????**
- **NO!!!!**
- **Compile time means: standard defined values**
- **Run time means: Actual performance values or exact value**

POSIX.1 and POSIX.1b Limits

- Eg: How many apps you can open in your mobile?

- Answer: It depends on the Mobile phone you are using it.!!!

- Different systems have different limits(values)!!!!
- The POSIX.1 and POSIX.1b standards defines a set of system configuration limit values!!!!

Different Limit Values are…..

Constant Name	Description	Value
_POSIX_ARG_MAX	Length of the argument	4096
_POSIX_CHILD_MAX	Number of child process per user ID	6
_POSIX_LINK_MAX	Number of links to a file	8
_POSIX_NAME_MAX	Number of bytes in a filename	14
_POSIX_OPEN_MAX	Number of open files per process	16
_POSIX_PATH_MAX	Number of bytes in a pathname	255

// Program to check Max. Argument Length at compile time

```
#define _POSIX_SOURCE
#define _POSIX_C_SOURCE     199309L
#include <iostream.h>
#include <unistd.h>

int main()
{
#ifdef _POSIX_ARG_MAX
   cout << “Max Length of argument”<<_POSIX_ARG_MAX";
#else
   cout << “ERROR";
#endif
}
```

// Program to check Max. Child Process at compile time

```
#define _POSIX_SOURCE
#define _POSIX_C_SOURCE     199309L
#include <iostream.h>
#include <unistd.h>

int main()
{
#ifdef _POSIX_CHILD_MAX
   cout << “Max. No. of Child Process”<<_POSIX_CHILD_MAX;
#else
   cout << “ERROR";
#endif
}
```

// Program to check Max. Open Files at compile time

```
#define _POSIX_SOURCE
#define _POSIX_C_SOURCE     199309L
#include <iostream.h>
#include <unistd.h>

int main()
{
#ifdef _POSIX_OPEN_MAX
    cout << “Max. No. of Open Files”<<_POSIX_OPEN_MAX";
#else
    cout << “ERROR";
#endif
}
```

Limits checking at run time

- To find out the actual implemented configuration limits at run time use one among the below functions

- long sysconf(const int limit_name);
  - Used to query the system wide configuration limits

- long pathconf(const char* pathname, int flimit_name);
  - To query file-related configuration limits, takes file pathname

- long fpathconf(const int fdesc, int flimitname);
  - To query file-related configuration limits, takes file descriptor

Limit Values for Sysconf( )

Name	Data Returned
_SC_ARG_MAX	Max. size in bytes
_SC_CHILD_MAX	Max number of child process that may be owned by a process
_SC_OPEN_MAX	Max number of opened files
_SC_CLK_TCK	Number of clock ticks per second
_SC_JOB_CONTROL	The _POSIX_JOB_CONTROL values
_SC_VERSION	The _POSIX_VERSION value

// Program to check Max. Arguments at Run Time

```
#define _POSIX_SOURCE
#define _POSIX_C_SOURCE     199309L
#include <iostream.h>
#include <unistd.h>

main( )
{
int res;
if(res=sysconf(_SC_ARG_MAX)) = = -1)
                perror(“sysconf”);
else
            cout<<“Max argument values”<<res;
}
```

// Program to check Max. Child Process at Run Time

```
#define _POSIX_SOURCE
#define _POSIX_C_SOURCE     199309L
#include <iostream.h>
#include <unistd.h>

main( )
{
int res;
if(res=sysconf(_SC_CHILD_MAX)) = = -1)
            perror(“sysconf”);
else
         cout<<“Max Child Process”<<res;
}
```

// Program to check Max. Open Files at Run Time

```
#define _POSIX_SOURCE
#define _POSIX_C_SOURCE     199309L
#include <iostream.h>
#include <unistd.h>

main( )
{
int res;
if(res=sysconf(_SC_OPEN_MAX)) = = -1)
                perror(“sysconf”);
else
            cout<<“Max Open Files”<<res;
}
```

Limit Values for Pathconf( ) / Fpathconf

Name	Data Returned
_PC_PATH_MAX	Max. length in bytes of a path name
_PC_LINK_MAX	Max number of links a file may have
_PC_NAME_MAX	Max length, in bytes, of a filename
_PC_NO_TRUNC	Returns the _POSIX_NO_TRUNC value
_PC_PIPE_BUF	Max size of a block that may read or written to a pipe file
_PC_MAX_INPUT	Max capacity, in bytes, of a terminal queue.

// Program to check Max. Path length at Run Time

```
#define _POSIX_SOURCE
#define _POSIX_C_SOURCE     199309L
#include <iostream.h>
#include <unistd.h>

main( )
{
int res;
if(res=sysconf(_PC_PIPE_MAX)) = = -1)
                perror(“sysconf”);
else
           cout<<“Max Open Files”<<res;

res=fpathconf(0,_PC_CHOWN_RESTRICTED);
cout <<“chown restricted for stdin”<<res;

}
```

THE POSIX.1 FIPS STANDARD

- **FIPS stands for Federal Information Processing Standard developed by National Institute of Standards and Technology.**

- **It is a guidelines for federal agencies acquiring computer systems**

- **The features to be implemented on FIPS systems are**

- **Job control :**
  - **_POSIX_JOB_CONTROL must be defined**
- **Saved set-UID and set-GID :**
  - **_POSIX_SAVED_IDS must be defined**
- **Long path name is supported**
  - **_POSIX_NO_TRUNC != -1**
- **_only authorised user can change ownership**
  - **_POSIX_CHOWN_RESTRICTED != -1**

- _POSIX_VDISABLE should be defined
- NGROUP_MAX
  - – value should be at least 8
- Read and write APIs should return the number of bytes transferred after the APIs have been interrupted by signals
- The group id of newly created file must inherit group ID of its containing directory

THE X/OPEN STANDARDS

- By a group of European companies to propose a common operating system interface for computer systems

- X/Open portability guide, ISSUE 3 (XPG3) --- 1989
- X/Open portability guide, ISSUE 4 (XPG4) --- 1999

- The portability guide specifies a set of common facilities and C application program interface function to be provided on all UNIX-based “open systems”

- In 1993 HP,IBM Novel,Open Software Foundation and Sun iniated a project called COSE (Common Open Software Environment)

- The X/Open CAE specifications have broader scope than POSIX and ANSI

UNIX APIs (System Calls)

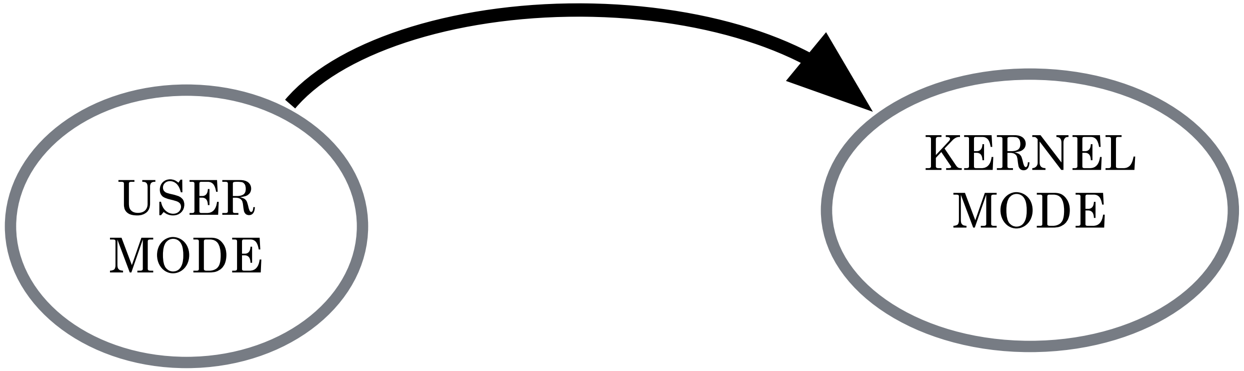
What do they do?

When called by users they perform **system specific functions**

- such as manipulate the files and processes

What happens when they are invoked ?

Execution context of the process

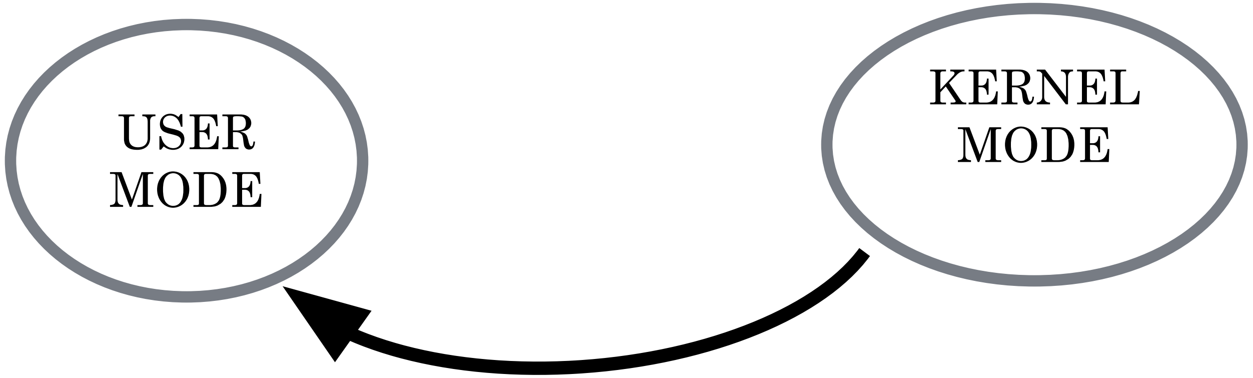


User mode is the normal execution environment ,allows the process to access process specific data only

Kernel mode is the safe execution environment ,kernel’s data is accessed in a restricted manner

What happens when the API execution is complete ?

Execution context of the process



Switches from Kernel mode back to the user mode

- Advantages of using the APIs
  - Kernel is accessed in a controlled manner
  - Kernel is not damaged by any erroneous applications

- Disadvantages of using the APIs
  - **Time consuming** because of the **context switching**

- Functions of the APIs
  - Determine system configuration and user info
  - Files manipulation
  - Processes creation and control
  - Interprocess communication
  - Network communication

UNIX APIs and C/C++ Library fns

UNIX APIs

- Directly manipulate files and processes

C/C++ Library functions

- cannot manipulate files and processes directly

UNIX APIs and C/C++ Library fns

- Many C/C++ Library functions call these APIs to manipulate the system

- So a user can call these APIs directly instead of using C/C++ Library functions

The POSIX APIs

- POSIX.1 and POSIX.1b APIs are derived from UNIX APIs

- POSIX has a new set of APIs from interprocess communication using messages , shared memory and semaphores that would communicated across a LAN which the UNIX V cannot do

THE POSIX APIS (CONTD)

- User's programs should define _POSIX_SOURCE and _POSIX_C_SOURCE in their program to enable the POSIX APIs declarations in header files

The UNIX and POSIX development Environment

- POSIX .1 and UNIX APIs – declared in <unistd.h> header
- <sys> directory has some set of API specific headers (On a UNIX system it is /usr/include/sys directory)
- <stdio.h> - declares perror function that prints error messages for any failed API

API Common Characteristics

- UNIX and POSIX APIs execution fails
  - APIs return -1
  - Global variable ***errno*** is set with a error code
  - ***errno*** is declared in <errno.h> header

API Common Characteristics(contd)

How Do we check errors /messages ?

By using two functions

- ***perror*** function
- ***strerror*** function

API Common Characteristics (contd)

- To check the error/diagnostic message a user can call *perror* function or it may call *strerror* function
  - *perror function*
    - *Prints the diagnostic message to the standard output*
  - *strerror function*
    - *User process can print this message to an output file / error log file*

API Common Characteristics (contd)

- Examples of error status code
  - EACCESS
    - A process does not have access permission
  - EPERM
    - An API aborted because the process does not have superuser privileges
  - EFAULT
    - Invalid address in the API argument
  - ECHILD
    - Process does not have a child to wait on

The File System:

- The File,
- What’s in a (File)name,
- The Parent-Child relationship,
- The UNIX File System,
- pwd,
- Absolute pathnames,
- cd,
- Relative pathnames,
- mkdir, rmdir, cp, rm, mv, cat, ls.

Introduction

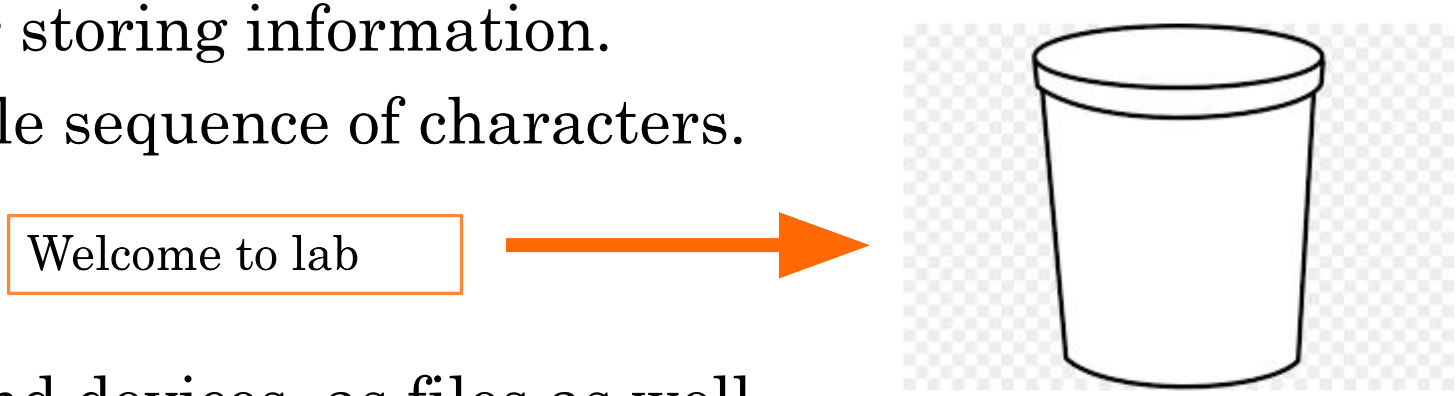
- UNIX looks at everything as a file and any UNIX system has thousands of files.
- If you write a program:
  - You add one more file to the system.(e.g. demo.c)
- When you compile a program:
  - You add one more file to the system.(e.g. demo.obj)
- When you execute a program:
  - You again add one more file to the file system.(e.g. demo.exe)
- Files grow rapidly, if not organized properly, difficult to locate them.
- File system in UNIX is one of its simple and conceptually clean feature.

UNIX File System

- It lets user to access other files not belonging to them.
- It also offers adequate security mechanism so that outsiders are not able to tamper with a file.

The File

- The file is a container for storing information.
- It can be treated as simple sequence of characters.
- Example:



Welcome to lab

- Unix treats directories and devices as files as well.
  - Directory is a simple folder where we store filenames and other directories.
- All physical devices likes,
  - Hard disk, memory, CD ROM, printer etc are treated as files.
- The kernel is also a file.

What's In a (File)Name?

- Filename can consist of up to 255 characters.
- Filename is the name given to a file for identification.
- The following characters are used in a filename:
  - Alphabetic characters and numerals.
  - The period(.), hypen(-), and underscore(_)
  - A file name can have as many dot(.) embedded in its name
- Unix is sensitive to case. (chap1, Chap1, CHAP1- are 3 different filenames)
- Application imposes restriction on file names
  - C complier expects file to have .c extention
  - SQL compiler expects file to have .sql extension

The Parent-Child Relationship

- All files in Unix are related to one another.
- All files are organized in hierarchical tree structure.
- In parent-child relationship, parent is always a directory.
- The root(/) is a directory.
- The root has sub-directories under it.
- The sub-directories can have sub-directories or files in it.

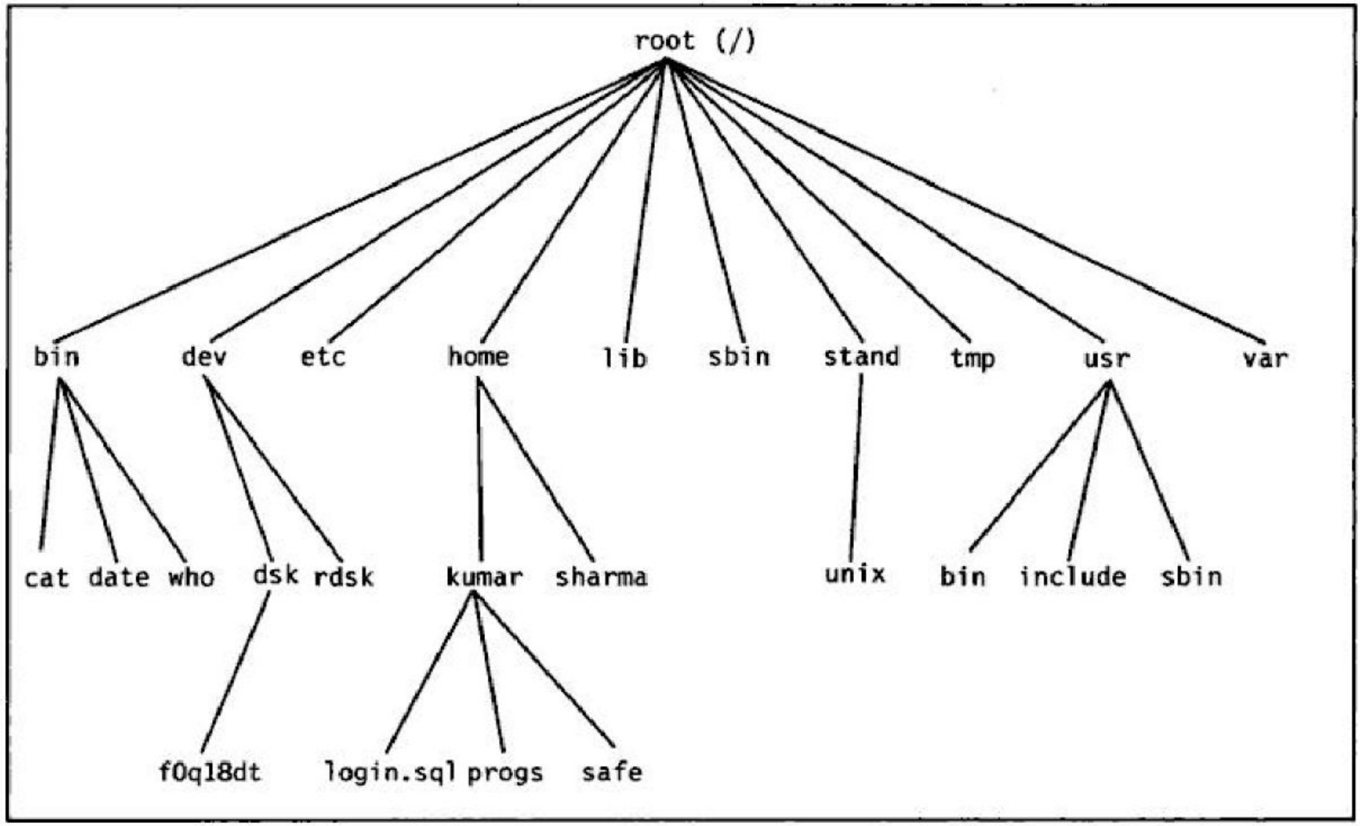
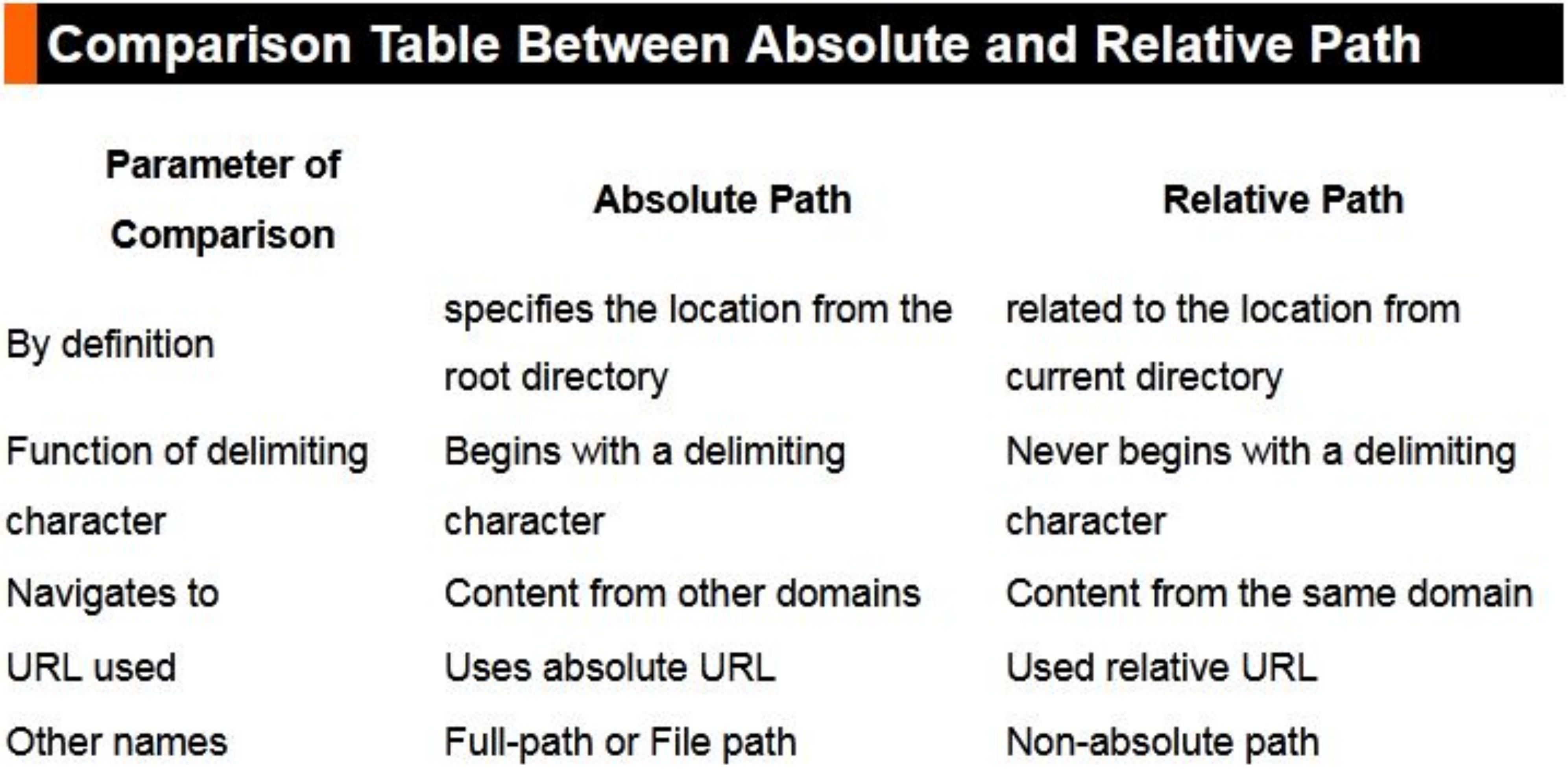


Fig. 4.1 The UNIX File System Tree

Absolute & Relative Pathnames

- A path is a unique location to a file or a folder in a file system of an OS.
- A path to a file is a combination of / and alpha-numeric characters.
- The difference between an absolute and a relative path is that an absolute path specifies the location from the root directory whereas a relative path is related to the current directory.
- Another visible difference between the two pathways is that an absolute pathway starts with a delimiting character such as "/" whereas a relative pathway never begins with such characters.



Comparison Table Between Absolute and Relative Path

Parameter of Comparison	Absolute Path	Relative Path
By definition	specifies the location from the root directory	related to the location from current directory
Function of delimiting character	Begins with a delimiting character	Never begins with a delimiting character
Navigates to	Content from other domains	Content from the same domain
URL used	Uses absolute URL	Used relative URL
Other names	Full-path or File path	Non-absolute path

COMMANDS…..

- pwd: Present working directory
- cp: **cp** stands for copy. This **command** is used to copy files or group of files or **directory**.
- cd: change of directory
- mkdir: to create a new directory
- rmdir: to remove a directory
- rm: to remove a file
- cat: to display the content of a file
- mv: to move file from one location to another
- ls: listing of files.

Sample Questions

1. What are the major differences between ANSI C and K & R C? explain
2. What is an API ? How are they different from C library functions ? Calling an API is more time consuming than calling a user function . Justify or contradict
3. Write a POSIX compliant C/C++ program to check following limits

   a) Maximum path length
   b) Maximum characters in a file name
   c) Maximum number of open files per process (many other limits can be asked)
4. What is POSIX standard? Explain different subsets of POSIX standard .
5. Write the structure of the program to filter out non-POSIX compliant codes for a user program

1. Write a C++ program that prints the POSIX defined configuration options supported on any given system using feature test macros
2. List out all POSIX.1 and POSIX 1b defined system configuration limits in manifested constants with complete time limit , minimum value and meaning
3. What are API characteristics? List the various error control flags.
4. Explain the various functions along with its syntax, used to get the limit values at runtime.
5. Differentiate between absolute and relative pathname.
6. Explain the Parent-Child relationship with a neat diagram.