INTRODUCTION TO UNIX

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.

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
- ANSI C defines a set of preprocessor symbols
- ANSI C defines a set of standard library functions and associated headers.

The ANSI / ISO C++ Standard

The C++ language is one of the OOP languages. It was developed by Bjarne Stroustrup at At &T Bell Laboratories. C++ is an extension of C with a major addition of the class construct features of Simula 67. The three most important facilities that C++ adds on to C are classes, function overloading, & operator overloading.

In 1989, Bjarne Stroustrup published "*The Annotated C++ Reference Manual" ,* this manual become the base for the draft ANSI C++ standard. WG21 committee of the ISO joined the ANSI X3J16 committee to develop a unify ANSI/ISO C++ standard. A draft version of ANSI/ISO standard was published in 1994.

Major Differences between ANSI and C++

- Function Declaration or Function Prototype
- Functions that take a variable number of arguments
- Type safe linkage , Linkage Directives

POSIX Standards

POSIX is acronym for Portable Operating System Interface. There are three subgroups in POSIX. They are :

POSIX.1 :

- Committee proposes a standard for base operating system APIs.
- This standard is formally known as the IEEE standard 1003.1-1990.
- This standard specifies the APIs for the file manipulation and processes (for Process Creation and Control).

POSIX.1b:

- Committee proposes a standard for real time operating system APIs
- This standard is formally known as the IEEE standard 1003.4-1993
- This standard specifies the APIs for the interprocess communication (Semaphores,Message Passing Shared Memory).

POSIX.1c:

- Committee proposes a standard for multithreaded programming interface
- This standard specifies the APIs for Thread Creation, Control, and Cleanup, Thread Scheduling,Thread Synchronization and for Signal Handling .

To ensure a user program conforms to the POSIX.1 standard, the user should define the manifested constant _POSIX_SOURCE at the beginning of each program(before the inclusion of any header files) as:

#define _ POSIX_SOURCE or

specify the –D_ POSIX_SOURCE option to a C++ compiler during compilation.

$g++ –D_ POSIX_SOURCE filename.cpp

In general a user program that must be strictly POSIX.1and POSIX.1b compliant may be written as follows:

```
#define _POSIX_SOURCE
#define _POSIX_C_SOURCE 199309L
#include <iostream.h>
#include <unistd.h>
int main( )
{
        ....
}
```

POSIX Feature Test Macros

Feature Test Macro	Effects if defined on a System
_POSIX_JOB_CONTROL	It allow us to start multiple jobs(groups of processes) from a single terminal and control which jobs can access the terminal and which jobs are to run in the background. Hence It supports BSD version Job Control Feature.
_POSIX_SAVED_IDS	Each process running on the system keeps the saved set-UID and set-GID, so that it can change effective user ID and group ID to those values via *setuid* and *setgid* APIs respectively.
_POSIX_CHOWN_RESTRICTED	If the defined value is -1, users may change ownership of files owned by them. Otherwise only users with special previlege may change ownership of any files on a system.
_POSIX_NO_TRUNC	If the defined value is -1, any long path name passed to an API is silently truncated to NAME_MAX bytes, otherwise error is generated.
_POSIX_VDISABLE	If the defined value is -1, there is no disabling character for special characters for all terminal device files, otherwise the value is the disabling character value.

Limits Checking at Compile Time and at Run Time

- The POSIX.1 and POSIX.1b standards specify a number of parameters that describe capacity limitations of the system.
- Limits are defined in <limits.h>.
- These are prefixed with the name _POSIX _

sysconf, pathcomf and fpathconf

To find out the actual implemented configuration limits

- System wide using *sysconf* during run time
- On individual objects during run time using, *pathconf* and *fpathconf.*

```
#include <unistd.h>

long   sysconf (int  parameter);

long  fpathconf(int fildes, int flimit_name));

long  pathconf(const char *path, int  flimit_name);
```

- For pathconf(), the *path* argument points to the pathname of a file or directory.
- For fpathconf (), the *fildes* argument is an open file descriptor.

Source : http://elearningatria.files.wordpress.com/2013/10/cse-vi-unix-system-programming-10cs62-notes.pdf