Symbolic links:

A symbolic link, also termed a soft link, is a special kind of file that points to another file, much like a shortcut in Windows or a Macintosh alias. Unlike a hard link, a symbolic link does not contain the data in the target file. It simply points to another entry somewhere in the file system. This difference gives symbolic links certain qualities that hard links do not have, such as the ability to link to directories, or to files on remote computers networked through NFS. Also, when you delete a target file, symbolic links to that file become unusable, whereas hard links preserve the contents of the file.

$ ln -s /path/to/file /path/to/symlink

$ ls -lt

If a file is a symbolic link you will see something like this:

myshortcut -> myfile

Hard links:

A hard link is essentially a label or name assigned to a file. Conventionally, we think of a file as consisting of a set of information that has a single name. However, it is possible to create a number of different names that all refer to the same contents. Commands executed upon any of these different names will then operate upon the same file contents.

A hard link is actually the same file that it links to but with a different name. Hard links provide an efficient way to organize files.

$ ln path/to/file /path/to/hard/link

How To Find All Hard Links To A File

All files in your Linux system contain an inode number which uniquely identifies the file. A file and its hard link will contain the same inode.

To see the inode number for a file type the following command:

$ ls -i

The output for a single file will be as follows:

1234567 filename

To find the hard links for a file you just need to do a file search for all the files with the same inode (i.e. 1234567).

You can do that with the following command:

$ find ~/ -xdev -inum 1234567

Absolute path: the absolute path contains the root directory and all other subdirectories in which a file or folder is contained. when using an absolute path you can be in any directory and the computer would know where to open the "calc.exe" file.

Relative path:

A relative path is a way to specify the location of a directory relative to another directory. a relative path starts from some given working directory, avoiding the need to provide the full absolute path. A filename can be considered as a relative path based at the current working directory.

Locale: The locale is an explicit model and definition of a native-language environment. The notion of a locale is explicitly defined and included in the library definitions of the ANSI C Language standard. A locale consists of a number of categories for which country-dependent formatting or other specifications exist. A program's locale defines its code sets, date and time formatting conventions, monetary conventions, decimal formatting conventions, and collation (sort) order. A locale can be composed of a base language, country (territory) of use, and an optional codeset. Codeset is usually assumed. For example, German is de, an abbreviation for Deutsch, while Swiss German is de\_CH, CH being an abbreviation for Confederation Helvetica. This convention allows for specific differences by country, such as currency unit notation.

Behavior Affected by Locales

Different cultures often use different conventions to format numbers, to write the date and time, to delimit words and phrases, or to quote written and spoken material. A locale determines how the following operations, files, formats, and expressions are handled for different regions:

Encoding and processing of text data

Language identification and encoding of resource files

Rendering and layout of text strings

Interchange of text between clients

Input method selection to meet the codeset and text processing requirements of the chosen script

Font and icon files that are culturally specific

Actions and file types

User Interface Definition (UID) files

Date and time formats

Numeric formats

Monetary formats

Collation order

Regular expression handling specific to the locale

Format for informative and diagnostic messages and interactive responses

Environment variables: Linux environment variables act as placeholders for information stored within the system that passes data to programs launched in shells or subshells.Admins have the ability to modify environment variables to fit personal or larger group needs of users within their environments. As you’ll notice below, admins can alter the hostname, command-line prompt, coloring in shells for text, and various other environment variables to better suit user preference.

Locale Categories

The locale categories are as follows:

LC\_CTYPE

Controls the behavior of character handling functions.

LC\_TIME

Specifies date and time formats, including month names, days of the week, and common full and abbreviated representations.

LC\_MONETARY

Specifies monetary formats, including the currency symbol for the locale, thousands separator, sign position, the number of fractional digits, and so forth.

LC\_NUMERIC

Specifies the decimal delimiter (or radix character), the thousands separator, and the grouping.

LC\_COLLATE

Specifies a collation order and regular expression definition for the locale.

LC\_MESSAGES

Specifies the language in which the localized messages are written, and affirmative and negative responses of the locale (yes and no strings and expressions).

LO\_LTYPE

Specifies the layout engine that provides information about language rendering. Language rendering (or text rendering) depends on the shape and direction attributes of a script.

Regular expression: A regular expression (or RE) specifies a set of strings that matches it; the functions in this module let you check if a particular string matches a given regular expression (or if a given regular expression matches a particular string, which comes down to the same thing).

Compilation process:

Preprocessing

The first stage of compilation is called preprocessing. In this stage, lines starting with a # character are interpreted by the preprocessor as preprocessor commands. These commands form a simple macro language with its own syntax and semantics. This language is used to reduce repetition in source code by providing functionality to inline files, define macros and to conditionally omit code.

Compiling

The second stage of compilation is confusingly enough called compilation. In this stage, the preprocessed code is translated to assembly instructions specific to the target processor architecture. These form an intermediate human readable language.

Assembly

The second stage of compilation is confusingly enough called compilation. In this stage, the preprocessed code is translated to assembly instructions specific to the target processor architecture. These form an intermediate human readable language.

Linking

The object code generated in the assembly stage is composed of machine instructions that the processor understands but some pieces of the program are out of order or missing. To produce an executable program, the existing pieces have to be rearranged and the missing ones filled in. This process is called linking.

The linker will arrange the pieces of object code so that functions in some pieces can successfully call functions in other pieces. It will also add pieces containing the instructions for library functions used by the program. In the case of the “Hello, World!” program, the linker will add the object code for the puts function.

The result of this stage is the final executable program.

System calls: In computing, a system call is the programmatic way in which a computer program requests a service from the kernel of the operating system it is executed on. A system call is a way for programs to interact with the operating system. A computer program makes a system call when it makes a request to the operating system’s kernel. System call provides the services of the operating system to the user programs via Application Program Interface(API). It provides an interface between a process and operating system to allow user-level processes to request services of the operating system. System calls are the only entry points into the kernel system. All programs needing resources must use system calls.

Services Provided by System Calls :

Process creation and management

Main memory management

File Access, Directory and File system management

Device handling(I/O)

Protection

Networking, etc.

Types of System Calls : There are 5 different categories of system calls –

Process control: end, abort, create, terminate, allocate and free memory.

File management: create, open, close, delete, read file etc.

Device management

Information maintenance

Communication

Parallelism: A multi-threaded program consists of sub-programs each of which is handled separately by different threads. Multi-threading allows for parallelism in program execution. All the active threads run concurrently, sharing the CPU resources effectively and thereby, making the program execution faster. Multi-threading is generally used when:

There are sub-programs whose output needs to be combined by the main program.

The main program contains sections of code that are relatively independent of each other.

Multithreading:

Multithreading in C

What is a Thread?

A thread is a single sequence stream within in a process. Because threads have some of the properties of processes, they are sometimes called lightweight processes.

What are the differences between process and thread?

Threads are not independent of one other like processes as a result threads shares with other threads their code section, data section and OS resources like open files and signals. But, like process, a thread has its own program counter (PC), a register set, and a stack space.

Why Multithreading?

Threads are popular way to improve application through parallelism. For example, in a browser, multiple tabs can be different threads. MS word uses multiple threads, one thread to format the text, other thread to process inputs, etc.

Threads operate faster than processes due to following reasons:

1) Thread creation is much faster.

2) Context switching between threads is much faster.

3) Threads can be terminated easily

4) Communication between threads is faster.

Dynamic linking: When one program is dependent on some other program. In such a case, rather than loading all the dependent programs, CPU links the dependent programs to the main executing program when its required. This mechanism is known as Dynamic Linking.

Dynamic linking refers to the linking that is done during load or run-time and not when the exe is created.

In case of dynamic linking the linker while creating the exe does minimal work.For the dynamic linker to work it actually has to load the libraries too.Hence it's also called linking loader.

Dynamic Linking:-

1) In dynamic linking the names of the external libraries (shared libraries) are placed in the final executable file while the actual linking takes place at run time when both executable file and libraries are placed in the memory. Dynamic linking lets several programs use a single copy of an executable module.

2) Dynamic linking is performed at run time by the operating system.

3) In dynamic linking only one copy of shared library is kept in memory. This significantly reduces the size of executable programs, thereby saving memory and disk space.

4) In dynamic linking this is not the case and individual shared modules can be updated and recompiled. This is one of the greatest advantages dynamic linking offers.

5) In dynamic linking load time might be reduced if the shared library code is already present in memory.

Symmetric encryption: Symmetrical encryption is an old and best-known technique. It uses a secret key that can either be a number, a word or a string of random letters. It is a blended with the plain text of a message to change the content in a particular way. The sender and the recipient should know the secret key that is used to encrypt and decrypt all the messages. The main disadvantage of the symmetric key encryption is that all parties involved have to exchange the key used to encrypt the data before they can decrypt it. Examples of popular symmetric-key algorithms include Twofish, Serpent, AES (Rijndael), Blowfish, CAST5, Kuznyechik, RC4, 3DES, Skipjack, Safer+/++ (Bluetooth), and IDEA. The most obvious application for symmetric encryption is transferring secret information between 2 people (or 2 computers) over an insecure channel - for instance by email, or on removable media.

Asymmetric encryption: Asymmetric encryption uses two keys to encrypt a plain text. Secret keys are exchanged over the Internet or a large network. It ensures that malicious persons do not misuse the keys. It is important to note that anyone with a secret key can decrypt the message and this is why asymmetrical encryption uses two related keys to boosting security. A public key is made freely available to anyone who might want to send you a message. The second private key is kept a secret so that you can only know.

A message that is encrypted using a public key can only be decrypted using a private key, while also, a message encrypted using a private key can be decrypted using a public key. Security of the public key is not required because it is publicly available and can be passed over the internet. Asymmetric key has a far better power in ensuring the security of information transmitted during communication.

Asymmetric encryption is mostly used in day-to-day communication channels, especially over the Internet. Popular asymmetric key encryption algorithm includes EIGamal, RSA, DSA, Elliptic curve techniques, PKCS.

SSH: SSH uses public-key cryptography to authenticate the remote computer and allow it to authenticate the user, if necessary.There are several ways to use SSH; one is to use automatically generated public-private key pairs to simply encrypt a network connection, and then use password authentication to log on. Another is to use a manually generated public-private key pair to perform the authentication, allowing users or programs to log in without having to specify a password. In this scenario, anyone can produce a matching pair of different keys (public and private). The public key is placed on all computers that must allow access to the owner of the matching private key (the owner keeps the private key secret). While authentication is based on the private key, the key itself is never transferred through the network during authentication. SSH only verifies whether the same person offering the public key also owns the matching private key. In all versions of SSH it is important to verify unknown public keys, i.e. associate the public keys with identities, before accepting them as valid. Accepting an attacker's public key without validation will authorize an unauthorized attacker as a valid user.

Make:

The purpose of the make utility is to determine automatically which pieces of a large program need to be recompiled, and issue the commands to recompile them. The manual describes the GNU implementation of make, which was written by Richard Stallman and Roland McGrath, and is currently maintained by Paul Smith. Our examples show C programs, since they are most common, but you can use make with any programming language whose compiler can be run with a shell command. In fact, make is not limited to programs. You can use it to describe any task where some files must be updated automatically from others whenever the others change.

Python default function argument:

# Return a function that returns its argument incremented by 'n'

def make\_incrementer(n):

def increment(x, n=n):

return x+n

return increment

add1 = make\_incrementer(1)

print add1(3) # This prints '4'