Questions

1) {}, <>, []

<>test[

<[()]>

<{[{}]>

```
def is_matched(expression):

 mapping = dict(zip('({[', ')}]'))

 queue = []
 for letter in expression:
 if letter in mapping:
 queue.append(mapping[letter])
 elif not (queue and letter == queue.pop()):
return False return not queue
```

2) List comprehension :

List comprehensions provide a concise way to create lists.

```
It consists of brackets containing an expression followed by a for clause, then
zero or more for or if clauses. The expressions can be anything, meaning you can
put in all kinds of objects in lists.
```

syntax : [expression for item in list if conditional]

3) Disk full still able to write to it ?

4) Disk 70%full still not able to write to it ?

Bcoz of lot of I/O happening.

5) systend / init

Init- In Linux, init is a abbreviation for Initialization. The init is a daemon process which starts as soon as the computer starts and continue running till, it is shutdown. In-fact init is the first process that starts when a computer boots, making it the parent of all other running processes directly or indirectly and hence typically it is assigned “pid=1“.

A systemd is a System Management Daemon named with UNIX convention to add ‘d‘ at the end of daemon. So, that they can be easily recognized. Initially it was released under GNU General Public License, but now the releases are made under GNU Lesser General Public License. Similar to init, systemd is the parent of all other processes directly or indirectly and is the first process that starts at boot hence typically assigned a “pid=1“.

6) Last word of each line

cat file | awk '{ print $NF }'

7) System slowness

While there are many things that can potentially cause system slowness, CPU, RAM, and disk I/O are behind the vast majority of performance problems.

to check run following commands:

```
cat /proc/cpuinfo

lscpu

service --status-all


chkconfig --list


initctl list

sudo systemctl list-unit-files --state=enabled (for systemd)
```

top

free

```
sudo apt install iotop
```

8) CPU and Memory check command

```
 # top
# iostat
# vmstat 5
```

What is File Descriptor?

n Unix and related computer operating systems, a **file descriptor** (**FD**, less frequently **fildes**) is an abstract indicator (handle) used to access a file or other input/output resource, such as a pipe or network socket. File descriptors form part of the POSIX application programming interface. A file descriptor is a non-negative integer, generally represented in the C programming language as the type int (negative values being reserved to indicate "no value" or an error condition).

Each Unix process (except perhaps a daemon) should expect to have three standard POSIX file descriptors, corresponding to the three standard streams:

Integer value	Name	<unistd.h> symbolic constant[1]	<stdio.h> file stream[2]
0	Standard input	STDIN_FILENO	stdin
1	Standard output	STDOUT_FILENO	stdout
2	Standard error	STDERR_FILENO	stderr

Changing File Descriptor on Fly

We can change file Descriptor on the fly. reason why we have to do that?

Imagine . run an important shell command and adding debug output to see that it works. then realizing that i will take hours to finish, and spitting gigabytes of debug to an xterm through ssh does not help. The dream is of course to just redirect that output to stdout in a screen process. Or just put it in /dev/null.

schenario 2:

An other typical example could be a finding an ill managed system with some daemon without proper logfile handling.

Restarting that process right now is just out of the question, copy-truncating that 16GB logfile will take too much time, and by the way, the disk is almost full.

Is it possible to just move the file descriptor for a running process? Yes it is. Welcome to the dark side of gdb.

With the power of gdb at your hand, you can hook into the inner parts of any running program, and change, well, virtually anything. Sounds insanely dangerous for systems in production, right? This hack is not that ugly. It does just what the doctor ordered: It changes a process' fds for you while it's running.

```
$ fdswap /var/log/mydaemon/output.log /dev/null 1234

#!/bin/bash
#
# fdswap
#
if [ "$2" = "" ]; then
        echo "
    Usage: $0 /path/to/oldfile /path/to/newfile [pids]
    Example: $0 /var/log/daemon.log /var/log/newvolume/daemon.log 1234
    Example: $0 /dev/pts/53 /dev/null 2345"; exit 0
fi

if gdb --version > /dev/null 2>&1; then true
else echo "Unable to find gdb."; exit 1
fi

src="$1"; dst="$2"; shift; shift
pids=$*

for pid in ${pids:=$( /sbin/fuser $src | cut -d ':' -f 2 )};
do
    echo "src=$src, dst=$dst"
    echo "$src has $pid using it"
    (
        echo "attach $pid"
        echo 'call open("'$dst'", 66, 0666)'
        for ufd in $(LANG=C ls -l /proc/$pid/fd | \
        grep "$src"\$ | awk ' { print $9; } ');
             do echo 'call dup2($1,'"$ufd"')'; done
        echo 'call close($1)'
        echo 'detach'; echo 'quit'
        sleep 5
    ) | gdb -q -x -
done
```

Limit open files of a specific process

A process can change its limits via the `setrlimit(2)` system call. When you run `ulimit -n` you should see a number. That's the current limit on number of open file descriptors (which includes files, sockets, pipes, etc) for the process. The `ulimit` command executed the `getrlimit(2)`system call to find out what the current value is.

Here's the key point: a process inherits its current limit from its parent process. So if you ran `ulimit -n 64` you would set that shell's limit of open file descriptors to 64. Any process that shell starts would have the same limit, unless that new process calls `setrlimit()` appropriately.

To change `mongodb's` open file descriptor limit, you'd run `ulimit -n 2048` (or whatever large number your kernel allows) in a shell. You'd then use that shell to start `mongodb`. As a child process, `mongodb` would inherit the (large) limit on open file descriptors.

To modify the system's open file limit, which seems more like the sum of all processes open file descriptor limit, you have to do something like modify `/etc/sysctl.conf` and run `sysctl -p`. Look at the value of the `fs.file-max` parameter in `/etc/sysctl.conf`.

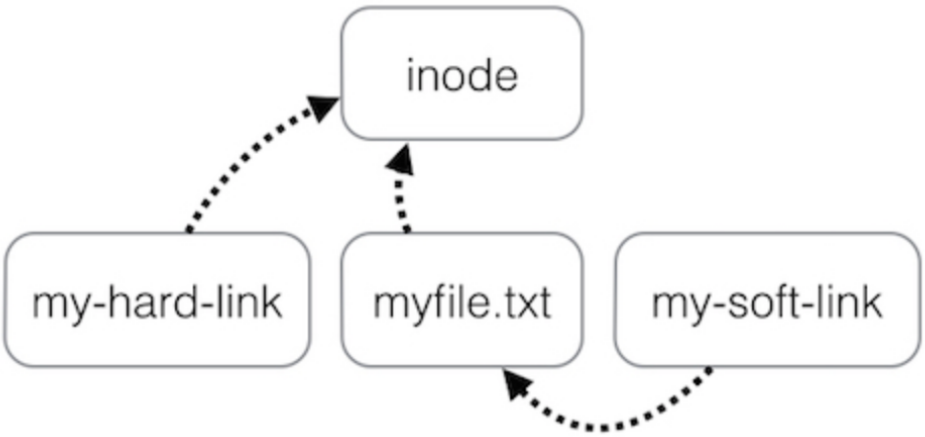
How to view the output of a running process in another bash session?

If all you want to do is spy on the existing process, you can use `strace -p1234 -s9999 -e write`where 1234 is the process ID. (`-s9999` avoids having strings truncated to 32 characters, and `write` the system call that produces output.) If you want to view only data written on a particular file descriptor, you can use something like `strace -p1234 -e trace= -e write=3` to see only data written to file descriptor 3 (`-e trace=` prevents the system calls from being loged). That won't give you output that's already been produced.

If the output is scrolling by too fast, you can pipe it into a pager such as `less`, or send it to a file with `strace -o trace.log ...`.

With many programs, you can divert subsequent output with a ptrace hack, either to your current terminal or to a new screen session. See How can I disown a running process and associate it to a new screen shell? and other linked threads.

Hard Link vs Soft Link



Once again, one of the very basic question asked in the interviews, What is the difference between Hard links and Soft links. You can explain few basic things, but emphasizing on internals about the linux filesystem, is something which can impress the interviewer

What is a Soft Link or Symbolic Link or Symlink ?

Symbolic links or Symlinks are the easiest to understand, because for sure you have used them, at least when you were using Windows. Soft links are very similar to what we say “Shortcut” in windows, is a way to link to a file or directory. Symlinks doesn’t contain any information about the destination file or contents of the file, instead of that, it simply contains the pointer to the location of the destination file. In more technical words, in soft link, a new file is created with a new inode, which have the pointer to the inode location of the original file. This can be better explained with a diagram:

Symbolic links are created with the “ln” command in linux. The syntax of the command is:

$ ln -s
-s = This flag tells to create a symlink (if you don’t use this it will create a hard link, which we will talk about soon).

For Example, if you want to create a soft link of one fo your favorite application, like gedit, on your desktop, use the command like this:

$ ln -s /usr/bin/gedit ~/Desktop/gedit

I hope now the concept of Soft Links should be clear.

What is a Hard Link ?

Hard link is a bit different object when compared to a symlink. In softlink a new file and a new Inode is created, but in hard link, only an entry into directory structure is created for the file, but it points to the inode location of the original file. Which means there is no new inode creation in the hard link. This can be explained like this:

So, in hard link, you are referencing the inode directly on the disk, which means that there should be a way to know how many hard links exist to a file. For the same, in the inode information, you have an option for “links”, which will tell how many links exists to a file. You can find the same information by using this command:

$ stat <file name>

$ stat 01
Size: 923383 Blocks: 1816 IO Block: 4096 regular file
Device: 803h/2051d Inode: 12684895 **Links: 3**
Access: (0644/-rw-r–r–) Uid: (0/ root) Gid: (0/ root)
Access: 2012-09-07 01:46:54.000000000 -0500
Modify: 2012-04-27 06:22:02.000000000 -0500
Change: 2012-04-27 06:22:02.000000000 -0500

In this example, it means that the specific file have 2 hard links, which makes the count to 3.

You can create a hard link with the same command “ln” like this

ln

So, to create a hard link of gedit program on your desktop, you will use the command like this:

ln /usr/bin/gedit ~/Desktop/gedit

Now, the bigger question is, who will decide what is better and when to use soft link or hard link

When to use Soft Link:

1. Link across filesystems: If you want to link files across the filesystems, you can only use symlinks/soft links.
2. Links to directory: If you want to link directories, then you must be using Soft links, as you can't create a hard link to a directory.

When to use Hard Link:

1. Storage Space: Hard links takes very negligible amount of space, as there are no new inodes created while creating hard links. In soft links we create a file which consumes space (usually 4KB, depending upon the filesystem)
2. Performance: Performance will be slightly better while accessing a hard link, as you are directly accessing the disk pointer instead of going through another file.
3. Moving file location: If you move the source file to some other location on the same filesystem, the hard link will still work, but soft link will fail.
4. Redundancy: If you want to make sure safety of your data, you should be using hard link, as in hard link, the data is safe, until all the links to the files are deleted, instead of that in soft link, you will lose the data if the master instance of the file is deleted.

The above points gives you a small idea where to use what, but doesn't tell you that those are the only options. Everything depends on your setup.

How file is deleted having hard links:

So, as it's pretty clear from the above article that hard links are just the reference to the main file location, and even if you delete one link, the data will still be intact. So, to remove a hard link, you need to remove all the links, which are referring to the file. Once the "link count" goes to "0", then the inode is removed by the filesystem, and file is deleted.

FAQs:

Q. What is the one line answer to the question "What is the main difference between hard links & soft links" ?

A. A softlink will have a different Inode number than the source file, which will be having a pointer to the source file but hardlink will be using the same Inode number as the source file.

Q. How can I find all the Soft Links in my system ?

A. Use this command for the same "find /etc -type l -exec ls -li {} \;"

Q. How can I find all the files having Hard Links in my system ?

A. Use this command for the same "find / -links +2 -type f -exec ls -li {} \;"

Q. How to find whether a file is a softlink ?

A. Simply using this command "ls -l" will tell you whether a file is pointing to some other file or not.
Q. How to check whether a file have any softlink pointing to it ?

A. Till now, I am not aware of any way to do that. If I will find any, I will surely update my post.

Q. How can I find out the source file of a hard link ?

A. No, you can't find out the source file of a hard link. Once hard link is created, there is no way to tell which was the first file created.

Q. Can I make a Soft link to a Hard link and Vice Versa ?

A. Yes, both soft links and hard links acts as normal files of the file system, so you can do both.

Fork Bomb

Thursday, May 24th, 2012 | By Napster | 0 Comments

Fork Bomb, Pretty much clear by name, anything which uses the "fork" operation and explodes like a bomb is known as Fork Bomb. It's a form of DOS (Denial Of Service) attack, using fork operation, in which a running process can create another running process. Fork bomb is sometimes referred as wabbit . In other words, Fork Bomb is a particular species of wabbit that can be written in one line of code.

How Fork Bomb Works:

A fork bomb process 'explodes' by recursively spawning copies of itself very quickly. Eventually it eats all the process table entries and effectively wedges the system. That means it will eat up complete process table and one won't be able to fork any process anymore until any other process terminates. Even if that happens, it is not likely that a useful program may be started since all the instances of the bomb program will each attempt to take any newly-available slot themselves.

Examples of Fork Bomb:

```
:(){ :|: & };:
```

Or in much explained manner you can say something like that:

```
:(){

 :|:&

};:
```

This was the simplest fork bomb. One more for example

```
$0 & $0 &
```

How to avoid Fork Bomb:

There could be different ideas to protect the fork bomb. One of them is to limit the number of processes a user or group can run, which can easily put a tab on the total number of processes in the system.

```
$ sudo vi /etc/security/limits.conf
*    -    nproc    200
napster hard nproc 300
@student hard nproc 50
```

Or else you can simply put this limit on all the processes by putting this line in /etc/profile

```
if [ `/usr/bin/id -u` != 0 ] .  # ID is 0 for the root user
then
ulimit -u 250 2> /dev/null
else
ulimit -u unlimited
fi
```

How to kill the Fork Bomb:

Fortunately, fork bombs are relatively easy to spot and kill. Well, first thing which anyone can follow is to restart your system. Trying to use a program to kill the rogue processes normally requires creating another process — a difficult or impossible task if the host machine has no empty slots in its process table, or no space in its memory structures.

Alternatively you can do a KILLALL and kill all the instances of the process, preventing them to spawn any other process.

```
$ killall -KILL BombProcessName
```

There are few other ways to do that, please refer to this link for the same – Defusing the Fork Bomb.

FAQs:

Q. How long it will take a Fork Bomb to kill/suspend a system ?

A. It takes nothing more than couple of seconds for a fork bomb to make any system non-responsive.

Q. Can we avoid for bombs from executing ?

A. Yes, we can. Just follow any of the steps written above.

Q. Is Fork Bomb a virus ?

A. No, it's not. Fork Bomb is just a normal procedure happening at big scale.

Orphan Process

Tuesday, March 22nd, 2011 | By Napster | 4 Comments

What is Orphan Process?

It's a pretty much common question asked in most of the interviews related to Linux, and most of the time people got it confused with Zombie Process. But these two are totally different from each other. An **Orphan Process** is nearly the same thing which we see in real world. Orphan means someone whose parents are dead. The same way this is a process, whose parents are dead, that means parents are either terminated, killed or exited but the child process is still alive.

In Linux/Unix like operating systems, as soon as parents of any process are dead, re-parenting occurs, automatically. Re-parenting means processes whose parents are dead, means Orphaned processes, are immediately adopted by special process “init”. Thing to notice here is that even after re-parenting, the process still remains Orphan as the parent which created the process is dead,

Reasons for Orphan Processes:

A process can be orphaned either intentionally or unintentionally. Sometime a parent process exits/terminates or crashes leaving the child process still running, and then they become orphans.

Also, a process can be intentionally orphaned just to keep it running. For example when you need to run a job in the background which don’t need any manual intervention and going to take long time, then you detach it from user session and leave it there. Same way, when you need to run a process in the background for infinite time, you need to do the same thing. Processes running in the background like this are known as daemon process.

At the same time, when a client connects to a remote server and initiated a process, and due to some reason the client crashes unexpectedly, the process on the server becomes Orphan.

Finding a Orphan Process:

It’s very easy to spot a Orphan process. Orphan process is a user process, which is having init (process id – 1) as parent. You can use this command in linux to find the Orphan processes.

```
# ps -elf | head -1; ps -elf | awk '{if ($5 == 1 && $3 != "root") {print $0}}' | head
```

This will show you all the orphan processes running in your system. The output from this command confirms that they are Orphan processes but doesn’t mean that they are all useless, so confirm from some other source also before killing them.

Killing a Orphan Process:

As orphaned processes waste server resources, so it’s not advised to have lots of orphan processes running into the system. To kill a orphan process is same as killing a normal process.

```
# kill -15 <PID>
```

If that don’t work then simply use

```
# kill -9 <PID>
```

FAQs:

Q. Is Orphan process different from an Zombie process ?

A. Yes, Orphan process are totally different from Zombie processes. Zombie processes are the ones which are not alive but still have entry in parent table. For more details, please refer to — Zombie Processes.

Q. Are Orphan processes harmful for system ?

A. Yes. Orphan processes take resources while they are in the system, and can potentially leave a server starved for resources. Having too many Orphan processes will overload the init process and can hang-up a Linux system. We can say that a normal user who has access to your Linux server is capable to easily kill your Linux server in a minute.

Zombie Process

Tuesday, March 22nd, 2011 | By Napster | 18 Comments

What is a Zombie Process ?

It’s a pretty much common question asked in most of the interviews related to Linux, and most of the time people got it confused with Orphan Process. But these two are totally different from each other. A Zombie Process is nearly the same thing which we see in lot of horror movies. Like the dead people which don’t have any life force left, Zombie processes are the dead processes sitting in the process table and doing nothing.

To explain this in much better way, “Zombie process or defunct process is a process that have completed the execution, have released all the resources (CPU, memory) but still had an entry in the process table.”

Reasons of Zombie Process:

Most of the time, the reason for existence of Zombie process is bad coding. Normally, when a child (subprocess) finishes it's task and exits, then it's parent is suppose to use the "wait" system call and get the status of the process. So, until the parent process don't check for the child's exit status, the process is a Zombie process, but it usually is very small duration. But if due to any reason (bad programming or a bug), the parent process didn't check the status of the child and don't call "wait", the child process stays in the state of Zombie waiting for parent to check it's status.

Finding Zombie processes:

To check whether you have any Zombie processes in your system, simply run linux command line utility "top". It shows the number of zombie processes at the upper-right side of its output.

At the same time, you can use one more command to check that

```
# ps aux
```

All the processes which are having "z" in their Stat column are Zombie processes.

Killing a Zombie Process:

Well, before taking any decision of killing the Zombie process, you should wait, as it is possible that the parent process is intentionally leaving the process in a zombie state to ensure that future children that it may create will not receive the same pid. Or perhaps the parent is occupied, and will reap the child process momentarily.

If that didn't happen then you can send a SIGCHLD signal to the parent process of zombie which will instruct parents to reap their zombie children.

```
# kill -s SIGCHLD <PPID>
```

Even if this don't work, then the last option you will have is to kill the parent process. You can easily find out the parent's process ID with this command:

```
# ps aux -eo ppid | grep <Zombie Process ID>
# kill -9 <PPID>
```

So when a Zombie process loses it's parent process, it becomes orphan and adopted by "init". Init periodically executes the wait system call to reap any zombies with init as parent.

FAQs:

Q. **Why I can't kill a Zombie process with "kill" command ?**
A. Zombie process is already dead, so killing them with "kill -9" won't help at all.

Q. **Is it bad to have Zombie processes on your system ?**
A. Well, as Zombie processes are not taking any resources of your system, leaving a small entry in process table, it's not at all harmful to have Zombie processes in your system, but it may hurt you sometime under heavy load. So, it's always better not to have them.

Q. **I am seeing a lots of Zombie processes around in my system, what could be the reason ?**
A. If this is the case then you are using some code/software in your system which is having a lot of bugs or not properly written. Look for something like that and fix it.

Q. **Is Zombie process different from an Orphan process ?**
A. Yes, Zombie is something which is already dead, but Orphan processes are those whose parents are dead.

Forking vs Threading

Monday, January 11th, 2010 | By Napster | 63 Comments

So, finally after long time, i am able to figure out the difference between forking and threading :)

When i have been surfing around, i see a lots of threads/questions regarding forking and threading, lots of queries which one should be used in the applications. So i wrote this post which could clarify the difference between these two based on which you could decide what you want to use in your application/scripts.

What is Fork/Forking:

Fork is nothing but a new process that looks exactly like the old or the parent process but still it is a different process with different process ID and having it's own memory. Parent process creates a separate address space for child. Both parent and child process possess the same code segment, but execute independently from each other.

The simplest example of forking is when you run a command on shell in unix/linux. Each time a user issues a command, the shell forks a child process and the task is done.

When a fork system call is issued, a copy of all the pages corresponding to the parent process is created, loaded into a separate memory location by the OS for the child process, but in certain cases, this is not needed. Like in 'exec' system calls, there is not need to copy the parent process pages, as execv replaces the address space of the parent process itself.

Few things to note about forking are:

- The child process will be having it's own unique process ID.
- The child process shall have it's own copy of parent's file descriptor.
- File locks set by parent process shall not be inherited by child process.
- Any semaphores that are open in the parent process shall also be open in the child process.
- Child process shall have it's own copy of message queue descriptors of the parents.
- Child will have it's own address space and memory.

Fork is universally accepted than thread because of the following reasons:

- Development is much easier on fork based implementations.
- Fork based code a more maintainable.
- Forking is much safer and more secure because each forked process runs in its own virtual address space. If one process crashes or has a buffer overrun, it does not affect any other process at all.
- Threads code is much harder to debug than fork.
- Fork are more portable than threads.
- Forking is faster than threading on single cpu as there are no locking over-heads or context switching.

Some of the applications in which forking is used are: *telnetd(freebsd), vsftpd, proftpd, Apache13, Apache2, thttpd, PostgreSQL.*

Pitfalls in Fork:

- In fork, every new process should have it's own memory/address space, hence a longer startup and stopping time.
- If you fork, you have two independent processes which need to talk to each other in some way. This inter-process communication is really costly.
- When the parent exits before the forked child, you will get a ghost process. That is all much easier with a thread. You can end, suspend and resume threads from the parent easily. And if your parent exits suddenly the thread will be ended automatically.
- In-sufficient storage space could lead the fork system to fail.

What are Threads/Threading:

Threads are *Light Weight Processes (LWPs).* Traditionally, a thread is just a CPU (and some other minimal state) state with the process containing the remains (data, stack, I/O, signals). Threads require less overhead than "forking" or spawning a new process because the system does not initialize a new system virtual memory space and environment for the process. While most effective on a multiprocessor system where the process flow can be scheduled to run on another processor thus gaining speed through parallel or distributed processing, gains are also found on uniprocessor systems which exploit latency in I/O and other system functions which may halt process execution.

Threads in the same process share:

- Process instructions
- Most data
- open files (descriptors)
- signals and signal handlers
- current working directory
- User and group id

Each thread has a unique:

- Thread ID
- set of registers, stack pointer
- stack for local variables, return addresses
- signal mask
- priority
- Return value: errno

Few things to note about threading are:

- Thread are most effective on multi-processor or multi-core systems.
- For thread – only one process/thread table and one scheduler is needed.
- All threads within a process share the same address space.
- A thread does not maintain a list of created threads, nor does it know the thread that created it.
- Threads reduce overhead by sharing fundamental parts.
- Threads are more effective in memory management because they uses the same memory block of the parent instead of creating new.

Pitfalls in threads:

- Race conditions: The big loss with threads is that there is no natural protection from having multiple threads working on the same data at the same time without knowing that others are messing with it. This is called *race condition.* While the code may appear on the screen in the order you wish the code to execute, threads are scheduled by the operating system and are executed at random. It cannot be assumed that threads are executed in the order they are created. They may also execute at different speeds. When threads are executing (racing to complete) they may give unexpected results (race condition). Mutexes and joins must be utilized to achieve a predictable execution order and outcome.
- Thread safe code: The threaded routines must call functions which are “thread safe”. This means that there are no static or global variables which other threads may clobber or read assuming single threaded operation. If static or global variables are used then mutexes must be applied or the functions must be re-written to avoid the use of these variables. In C, local variables are dynamically allocated on the stack. Therefore, any function that does not use static data or other shared resources is thread-safe. Thread-unsafe functions may be used by only one thread at a time in a program and the uniqueness of the thread must be ensured. Many non-reentrant functions return a pointer to static data. This can be avoided by returning dynamically allocated data or using caller-provided storage. An example of a non-thread safe function is strtok which is also not re-entrant. The “thread safe” version is the re-entrant version strtok_r.

Advantages in threads:

- Threads share the same memory space hence sharing data between them is really faster means inter-process communication (IPC) is real fast.
- If properly designed and implemented threads give you more speed because there aint any process level context switching in a multi threaded application.
- Threads are really fast to start and terminate.

Some of the applications in which threading is used are: *MySQL, Firebird, Apache2, MySQL 323*

FAQs:

1. Which should i use in my application ?

Ans: That depends on a lot of factors. Forking is more heavy-weight than threading, and have a higher startup and shutdown cost. Interprocess communication (IPC) is also harder and slower than interthread communication. Actually threads really win the race when it comes to inter communication. Conversely, whereas if a thread crashes, it takes down all of the other threads in the process, and if a thread has a buffer overrun, it opens up a security hole in all of the threads.

which would share the same address space with the parent process and they only needed a reduced context switch, which would make the context switch more efficient.

2. Which one is better, threading or forking ?

Ans: That is something which totally depends on what you are looking for. Still to answer, In a contemporary Linux (2.6.x) there is not much difference in performance between a context switch of a process/forking compared to a thread (only the MMU stuff is additional for the thread). There is the issue with the shared address space, which means that a faulty pointer in a thread can corrupt memory of the parent process or another thread within the same address space.

3. What kinds of things should be threaded or multitasked?

Ans: If you are a programmer and would like to take advantage of multithreading, the natural question is what parts of the program should/ should not be threaded. Here are a few rules of thumb (if you say “yes” to these, have fun!):

- Are there groups of lengthy operations that don’t necessarily depend on other processing (like painting a window, printing a document, responding to a mouse-click, calculating a spreadsheet column, signal handling, etc.)?
- Will there be few locks on data (the amount of shared data is identifiable and “small”)?
- Are you prepared to worry about locking (mutually excluding data regions from other threads), deadlocks (a condition where two COEs have locked data that other is trying to get) and race conditions (a nasty, intractable problem where data is not locked properly and gets corrupted through threaded reads & writes)?
- Could the task be broken into various “responsibilities”? E.g. Could one thread handle the signals, another handle GUI stuff, etc.?

Conclusions:

1. Whether you have to use threading or forking, totally depends on the requirement of your application.
2. Threads more powerful than events, but power is not something which is always needed.
3. Threads are much harder to program than forking, so only for experts.
4. Use threads mostly for performance-critical applications.

A virtual LAN (Local Area Network)

A virtual LAN (Local Area Network) is a logical subnetwork that can group together a collection of devices from different physical LANs. Larger business computer networks often set up VLANs to re-partition their network for improved traffic management.

Several different kinds of physical networks support virtual LANs including both Ethernet and Wi-Fi.

Benefits of a VLAN

When set up correctly, virtual LANs can improve the overall performance of busy networks. VLANs are intended to group together client devices that communicate with each other most frequently. The traffic between devices split across two or more physical networks ordinarily needs to be handled by a network's core routers, but with a VLAN that traffic can be handled more efficiently by network switchesinstead.

VLANs also bring additional security benefits on larger networks by allowing greater control over which devices have local access to each other. Wi-Fi guest networks are often implemented using wireless access points that support VLANs.

Static and Dynamic VLANs

Network administrators often refer to static VLANs as “port-based VLANs.” A static VLAN requires an administrator to assign individual ports on the network switch to a virtual network. No matter what device plus into that port, it becomes a member of that same pre-assigned virtual network.

Dynamic VLAN configuration allows an administrator to define network membership according to characteristics of the devices themselves rather than their switch port location. For example, a dynamic VLAN can be defined with a list of physical addresses (MAC addresses) or network account names.

VLAN Tagging and Standard VLANs

VLAN tags for Ethernet networks follow the IEEE 802.1Q industry standard. An 802.1Q tag consists of 32 bits (4 bytes) of data inserted into the Ethernet frame header. The first 16 bits of this field contain the hardcoded number 0x8100 that triggers Ethernet devices to recognize the frame as belonging to a 802.1Q VLAN. The last 12 bits of this field contain the VLAN number, a number between 1 and 4094.

Best practices of VLAN administration define several standard types of virtual networks:

Native LAN: Ethernet VLAN devices treat all untagged frames as belonging to the native LAN by default. The native LAN is VLAN 1, although administrators can change this default number.
Management VLAN: Used to support remote connections from network administrators. Some networks use VLAN 1 as the management VLAN while others set up a special number just for this purpose (to avoid conflicting with other network traffic)

Setting up a VLAN

At a high level, network administrators set up new VLANs as follows:

1. Choose a valid VLAN number
2. Choose a private IP address range for devices on that VLAN to use
3. Configure the switch device with either static or dynamic settings. Static configurations require the administrator to assign a VLAN number to each switch port while dynamic configurations require assigning a list of MAC addresses or usernames to a VLAN number.
4. Configure routing between VLANs as needed. Configuring two or more VLANs to communicate with each other requires the use of either a VLAN-aware router or a Layer 3 switch.

What is the difference between TCP and UDP?

Oskar Åkesson

The network scanner supports TCP and UDP. Here is some information about TCP and UDP and the differences between the different protocols.

General

Both TCP and UDP are protocols used for sending bits of data — known as packets — over the Internet. They both build on top of the Internet protocol. In other words, whether you are sending a packet via TCP or UDP, that packet is sent to an IP address. These packets are treated similarly, as they are forwarded from your computer to intermediary routers and on to the destination.

TCP and UDP are not the only protocols that work on top of IP. However, they are the most widely used. The widely used term “TCP/IP” refers to TCP over IP. UDP over IP could just as well be referred to as “UDP/IP”, although this is not a common term.

TCP

TCP stands for Transmission Control Protocol. It is the most commonly used protocol on the Internet.

When you load a web page, your computer sends TCP packets to the web server’s address, asking it to send the web page to you. The web server responds by sending a stream of TCP packets, which your web browser stitches together to form the web page and display it to you. When you click a link, sign in, post a comment, or do anything else, your web browser sends TCP packets to the server and the server sends TCP packets back. TCP is not just one way communication — the remote system sends packets back to acknowledge it is received your packets.

TCP guarantees the recipient will receive the packets in order by numbering them. The recipient sends messages back to the sender saying it received the messages. If the sender does not get a correct response, it will resend the packets to ensure the recipient received them. Packets are also checked for errors. TCP is all about this reliability — packets sent with TCP are tracked so no data is lost or corrupted in transit. This is why file downloads do not become corrupted even if there are network hiccups. Of course, if the recipient is completely offline, your computer will give up and you will see an error message saying it can not communicate with the remote host.

UDP

UDP stands for User Datagram Protocol — a datagram is the same thing as a packet of information. The UDP protocol works similarly to TCP, but it throws all the error-checking stuff out. All the back-and-forth communication and deliverability guarantees slow things down.

When using UDP, packets are just sent to the recipient. The sender will not wait to make sure the recipient received the packet — it will just continue sending the next packets. If you are the recipient and you miss some UDP packets, too bad — you can not ask for those packets again. There is no guarantee you are getting all the packets and there is no way to ask for a packet again if you miss it, but losing all this overhead means the computers can communicate more quickly.

UDP is used when speed is desirable and error correction is not necessary. For example, UDP is frequently used for live broadcasts and online games.

DNS Records Explained

Summary

DNS (Domain Name System) entries take a human friendly name, such as store.example.com, and translates it to an IP address. DNS can be quickly updated with some propagation time. There are a number of DNS Entries you are able to create. The following DNS Entries can be created or modified from within the DNS Zone Editor. For help see How to Modify Your DNS Records.

- A Record
- CNAME
- MX Entry
- TXT Record
- SRV Record
- AAAA Record
- DNS Glossary

A Record

An A record (Address Record) points a domain or subdomain to an IP address. For example, you can use it for store.website.com or blog.website.com and point it to where you have your store. This is a common practice for people who use Amazon, eBay, Tumblr, etc.

As an example, an A Record is used to point a logical domain name, such as "google.com", to the IP address of Google's hosting server, "74.125.224.147".

These records point traffic from example.com (indicated by @) and ftp.example.com to the IP address 66.147.224.236. They also point localhost.example.com to the server that the domain is hosted on. This allows the end user to type in a human-readable domain, while the computer can continue to work with numbers.

CNAME

A CNAME (Canonical Name) points one domain or subdomain to another domain name, allowing you to update one A Record each time you make a change, regardless of how many Host Records need to resolve to that IP address.

These records point www.example.com to example.com, imap.example.com to mail.example.com, and docs.example.com to ghs.google.com. The first record allows the domain to resolve to the same server with or without the www subdomain. The second record allows you to use an alternative subdomain for email hosting an delivery. The third record allows you to use the docs.example.com subdomain with Google Apps, where you can use Google's document management system. This type of record requires additional configuration with Google.

MX Entry

An MX Entry (Mail Exchanger) directs email to a particular mail server. Like a CNAME, MX Entries must point to a domain and never point directly to an IP address.

TXT Records

A TXT (Text) record was originally intended for human-readable text. These records are dynamic and can be used for several purposes.

The TXT Value is what the record 'points to', but these records aren't used to direct any traffic. Instead they're used to provide needed information to outside sources.
The First record is used for a SPF, Sender Policy Framework, records, those records are used by many email systems to help identify if email is coming from a trusted source, helping filter out spam or messages pretending to be from your domain (called spoofing). More information on SPF records can be found at http://www.openspf.net/
The second record is used for DomainKeys, which is also used to verify that email came from a trusted source. More information on DomainKeys can be found at http://www.dkim.org/

SRV record

An SRV (Service) record points one domain to another domain name using a specific destination port. SRV records allow specific services, such as VOIP or IM, to be be directed to a separate location.

Example:

Enabling your domain to use Google's xmpp server is a good example to showcase. Google's help article states that the SRV record should be in this format:

```
_xmpp-server._tcp.gmail.com. IN SRV 5 0 5269 xmpp-server.l.google.com.
```

Under "Add DNS Record", you will need to enter the settings this way:

- Service: _xmpp-server
- Protocol: _tcp
- Host: chat (If you want to use the chat subdomain. Replace this with the subdomain that you want to us, or @ for the root domain.)
- TTL: 14400
- Type: SRV
- Priority: 5
- Weight: 0
- Port: 5269
- Points To: xmpp-server.l.google.com

AAAA Record

The AAAA record is similar to the A record, but it allows you to point the domain to an Ipv6 address. More information on IPv6 can be found at http://ipv6.com/.

DNS Glossary:

Zone File:This is where all the DNS records are stored for a domain.Host Record:This is the domain or subdomain you wish to use. The @ symbol is used to indicate the root domain itself. In our example the Host Record 'ftp' would be for the subdomain ftp.google.com and '@' would be google.com itself. Points to:This is the destination server that the domain or subdomain is sending the traffic to.TTL:The 'time to live' value indicates the amount of time the record is cached by a DNS Server, such as your Internet service provider. The default (and lowest accepted) value is 14400 seconds (4 hours). You do not normally need to modify this value.Action:This allows you to modify or remove existing records.Weight:This is similar to priority, as it controls the order in which multiple records are used. Records are grouped with other records that have the same Priority value. As with MX Entries, lower numbers are used before higher numbers.Port:This is used by the server or computer to process traffic to specific services, ensuring that all traffic comes through the door that it's expected on.Target:This is the destination that the record is sending the traffic to.

DNS (Domain Name Service) is one of the most basic services on the Internet. If you want to effectively set up TCP/IP on your network, you'll probably need to install a DNS server at some point. But what is DNS and how does it work? In this Daily Drill Down, I'll take a brief look at DNS and show you some of its ins and outs.

DNS 101
The early Internet landscape was pretty barren with only a few hundred computers making up the ARPANET, the military/educational precursor to the Internet. Then, as today, each device on the network was a node, and each node needed a unique address to enable data packets to find their destinations. Anyone that has ever used IP addresses knows that it's tough enough to remember a few addresses on your local network, much less keep track of the addresses for remote systems you use often. That's where host names came into the picture.

Host names provide a more "friendly" way to name hosts, making it easier to remember host addresses. For example, when you want to get the news, you can point your browser to www.newsmax.com instead of 64.29.200.227. Add a couple of hundred other addresses to your frequent site list, and you can see that host names are a lot easier on the brain than IP addresses.

But converting host names to IP addresses doesn't happen by magic. It needs some form of translation to make it happen, and the mechanism that enables that translation is the Domain Name System, or DNS.

In the early days when there were only a few hundred nodes, a single text file could easily map host names to their corresponding IP addresses. This text file, called Hosts.txt, was managed by the Standford Research Institute (SRI) and contained all of the name-to-address mappings for all nodes on the ARPANET. Operating systems (primarily UNIX at that point) use the Hosts.txt file to map host names to IP addresses. System administrators copied the Hosts.txt file from SRI to their local systems periodically to update their address maps.

As the number of nodes on the network continued to increase, using a relatively static text file to provide mapping quickly became impractical. New hosts were added so rapidly that neither SRI nor system administrators had any hope of keeping up. So, the DNS system was developed in the mid-1980s to provide a dynamic means of updating and resolving host names to their IP addresses.

About those host and domain names
Each device on the Internet is called a host. Whether the host is a computer, printer, router, and so forth, as long as it has a unique IP address, it's a host. Just as the IP address identifies the host uniquely, so does the host name. For example, assume your computer's host name is rainbow. Your computer resides in the domain techrepublic.com. So, your computer's Fully Qualified Domain Name (FQDN) is rainbow.techrepublic.com. The FQDN identifies the host uniquely within the DNS name space.

Domains aren't limited to a single level. Assume techrepublic.com has two different divisions, one on the east coast and one on the west coast. The east coast domain is east.techrepublic.com and the other is west.techrepublic.com. Sales is located on the west coast, so its domain is sales.west.techrepublic.com. Joe Blow, who works in the sales department, has a computer named jblow. Its FQDN is jblow.sales.west.techrepublic.com. Traffic reaches his computer through something called delegation, which I'll cover a little later.

The DNS name space contains seven common domain suffixes, which are listed in **Table A**.

Table A

Suffix	Purpose	Example

com	Commercial organizations (businesses)	microsoft.com
edu	Educational organizations such as colleges and universities	berkeley.edu
gov	Governmental organizations such as the IRS, SSA, NASA, and so on	nasa.gov
mil	Military organizations	army.mil
net	Networking organizations such as ISPs	mci.net
org	Noncommercial organizations such as the IEEE standards body	ieee.org
int	International organizations such as NATO	nato.int

There are other domain suffixes as well, including national domains such as the us domain, which is used for governmental, regional, and educational entities in the United States. Other countries have their own domains, such as jp for Japan, uk for the United Kingdom, and so forth.

Until recently, an organization known as InterNIC was responsible for managing the majority of the top-level domains in the DNS name space. InterNIC switched from being a nonprofit organization to the now for-profit Network Solutions. When it made that switch, it lost its monopoly on the name space and now there are several entities that register and maintain the DNS name space.

How DNS works
DNS essentially functions as a distributed database using a client/server relationship between clients that need name resolution (mapping host names to IP addresses) and the servers that maintain the DNS data. This distributed database structure enables the DNS name space to be both dynamic and decentralized, giving local domains control over their own portion of the DNS database while still enabling any client to access any part of the database.

At the uppermost level of the DNS name space are the root servers. The root servers manage the top level domains: .com, .net, .org, .mil, .edu, .gov, and .int. With all the domains in existence today, not to mention all the hosts in those domains, you can see why the root servers actually maintain very little information about each domain. In fact, the only data the root servers typically maintain about a domain are the name servers that are authoritative for the domain, or which have authority for the domain's records.

The authoritative name servers actually maintain the records for a domain, or in some cases delegate some of or the entire domain to other name servers. The root servers know the name servers for techrepublic.com, for example, and within those name servers the west.techrepublic.com domain is delegated to another set of name servers that manage that portion of the overall techrepublic.com domain. In most cases, domains and their records are either managed directly by the organization owning the domain or by the ISP that provides the Internet connection for the organization.

How DNS lookup works
A DNS client uses a resolver to request resolution of a host name to an IP address. The resolver is really just a special-purpose application that's sole function is to act as an intermediary between name servers and various applications that need name resolution, such as Web browsers, e-mail applications, and so on. Here's an example: Assume you fire up your browser and direct it to connect to www.techrepublic.com. Your local resolver creates a DNS query and sends it to the name server(s) listed in the local computer's TCP/IP settings.

In this case, I'll assume that you're connected to the Internet through a dial-up connection to an ISP, and the ISP's name servers are specified in your computer's TCP/IP settings. The resolver sends the DNS request to the first of those name servers. The server checks its cache of previously resolved names, and in this case I'll assume that the ISP's name server has no cached results for techrepublic.com. So, that name server sends a DNS query to the root server for the .com domain. The root server responds with the addresses of the name servers that are authoritative for the techrepublic.com domain. Your ISP's name server then builds another request for www.techrepublic.com and submits it to techrepublic.com's name server, which responds with the IP address of www.techrepublic.com. That information is passed back to your resolver, which passes it to your application. Suddenly, techrepublic.com's Web site appears in your browser, and all in a matter of a second or two.

Resolving a host name to an IP address is called forward lookup. There are actually other ways the forward lookup can happen, depending on the way the name servers involved are configured. For now, let's assume it happens as described above.

Zones, domains, and subnets
You might think there is a relationship between an IP subnet and a domain, but there is actually no correlation at all. A given domain can encompass any number of subnets, none of which have any relationship to the domain itself. In some cases, records in the domain can even point to hosts outside of the domain's subnets. For example, assume your company maintains its own name servers for its domain but outsources hosting for its Web site. The company name servers contain a record that points the name www to the hosting company's subnet, outside of those used by your company.

A name server in most cases maintains the records for a portion of the DNS name space called a zone. In many cases, the terms zone and domain seem synonymous, but they're actually not the same thing. A zone comprises all the data for a given domain except those parts of the domain that are delegated to other name servers. So, a zone is the portion of a domain hosted on a specific name server. When the entire domain resides on a single name server, then domain and zone are synonymous.

As mentioned above, a name server can be authoritative for a zone. This means the server has full information about the zone. A single name server can manage many different zones, a case in point being a hosting company that might host several hundred or even thousands of domains. The hosting company's name servers would typically manage at least one zone for each hosted domain. In addition, a name server can be authoritative for some zones and non-authoritative for others.

Zones also fall into two categories: primary master or secondary master. A primary master maintains the original records for the zone. The zone's administrator can create and modify records in the primary zone. A secondary master serves as a read-only copy of the primary (essentially a backup copy of the zone). The name server on which the secondary resides updates its copy of the records through a periodic zone transfer, the frequency of which is controlled by the zone's configuration. A given name server can maintain primary zones for some domains and secondary zones for others, so the distinction between primary and secondary is zone-related, not server-related.

How about those DNS records?
The main purpose for DNS is to map host names to IP addresses, and the data that makes that possible are stored as records in a zone file on the DNS server hosting the zone. Within each zone file (really just a text file) are resource records that define host names and other domain elements. There are

several different types of resource records, each of which performs a specific function. **Table B** lists resource record types supported by the Windows 2000 DNS service.

Table B

Record	Purpose
SOA	Specifies authoritative server for the zone
NS	Specifies address of domain's name server(s)
A	Maps host name to an address
PTR	Maps address to a host name for reverse lookup
CNAME	Creates alias (synonymous) name for specified host
MX	Mail exchange server for domain
SRV	Defines servers for specific purpose such as HTTP, FTP, and so on
AAAA	Maps host name to Ipv6 address
AFSDB	Location of AFS cell database server or DCE cell's authenticated server
HINFO	Identifies host's hardware and OS type
ISDN	Maps host name to ISDN address (phone number)
MB	Associates host with specified mailbox; experimental
MG	Associates host name with mail group; experimental
MINFO	Specifies mailbox name responsible for mail group; experimental
MR	Specifies mailbox name that is proper rename of other mailbox; experimental
RP	Identifies responsible person for domain or host
RT	Specifies intermediate host that routes packets to destination host
TXT	Associates textual information with item in the zone
WKS	Describes services provided by specific protocol on specific port
X.25	Maps host name to X.121 address (X.25 networks); used in conjunction with RT records
WINS	Allows lookup of host portion of domain name through WINS server
WINS-R	Reverses lookup through WINS server
ATMA	Maps domain name to ATM address

As you can see in Table B, there are a lot of resource record types to deal with. Fortunately, most installations only require a few of the more common types, including SOA, A, NS, PTR, CNAME, and MX. The SOA record indicates that the server is authoritative for the zone, and Windows 2000 automatically creates an SOA record when you create a zone. The NS records identify the name servers for the zone.

The A record, also called a host record, maps a host name to an IP address. A given host might have different host names all mapped to the same IP address. For example, a multipurpose server might map the www, FTP, and mail host names to the same IP address, since the server performs all of those functions (Web server, FTP server, and mail server). In addition, a zone might have multiple entries for the same host name, each mapped to a different IP address. The server returns all matching addresses to the resolver. When the client is on the same network as the name server, the name server sorts the results so that the address closest to the client is at the top of the list for performance reasons. If the client is on a different network, the server cycles through the addresses in round-robin fashion. In one query, the first matching record is returned at the top of the list. In the second query, the second match is returned at the top of the list and so on. This gives the server the ability to essentially load-balance queries. For example, if you're hosting the same Web site on redundant servers, round-robin query results enable the DNS server to help balance the load across all of the servers.

CNAME stands for Canonical NAME. CNAME records map an alias name to a Fully Qualified Domain Name (FQDN). They are also called alias records. Host (A) records and CNAME records are usually applied in conjunction with one another. You might map a server's host name using an A record, then map (for example) www, FTP, and mail using CNAME records.

Mail Exchanger (MX) records are used to route e-mail. An MX record includes the FQDN of the mail server for the zone, along with a preference number from 0 through 65535. The preference number establishes priority for mail, and when there are multiple MX records with different preference numbers, the one with the lowest number is attempted first. Failing that, the sending server uses the record with the next highest number. If all MX records in the zone have the same preference number, the remote mail server decides which record to use to attempt mail delivery.

The Pointer (PTR) record is another common record type. PTR records perform the reverse of what host records do by mapping IP addresses to host names. PTR records enable reverse lookup. When you create or modify an A record, the Windows 2000 DNS service can automatically create or modify the associated PTR record for you.

Regardless of its type, each resource record has certain properties associated with it, and this information is stored along with the record in the zone file. Among these properties is a time-to-live (TTL) property that specifies the number of seconds the resolver should cache the record before it is considered stale and is discarded. When the specified TTL period is reached, the resolver discards the record and a subsequent attempt to resolve the name will pull it from the authoritative name server for the record rather than from the local cache.

The Internet is a dynamic place with hosts coming and going or changing their addresses frequently. The TTL enables records to be cached locally to improve response time and reduce the load on the network, since resolution doesn't go past the initially queried name server where the record is cached. When the record grows stale, the caching server (or resolver) throws it out, ensuring that the next query will pull an up-to-date version of the record.

Difference between cname and uname

Protocol used to handshake between different edge locations

How to increase postgres performance

How to traverse in nested dictionary