 **How would you check**

* **Memory used by a process(RAM)**
* **total number of open files by a process**
* **running duration of a process**

ps -eo pid,ppid,cmd,%mem,%cpu --sort=-%mem | head

# pmap -x 37649

#netstat –tulpn

# top -p 37649

**Total number of open files by a process:**

lsof –p pid

Running duration of a process:

# ps -p 1173 -o etime

ELAPSED

02:43

**What is file descriptor:**

A kernel creates a file descriptor whenever it encounters an open call. In many ways, the gateway into the kernel’s abstractions of underlying hardware can be considered as file descriptors. In the Unix operating system, the standard input is represented by file descriptor 0, the standard output is represented by file descriptor 1 and standard error file is represented by file descriptor 2. In other words, corresponding to the three standard streams, each UNIX process would have three standard file descriptors. Both streams and file descriptors can represent a device connection, however for controlling specific devices, file descriptors need to be used. In most operating systems such as UNIX, file descriptors are represented as objects of type “int.” The file descriptor is used by the kernel as an index in the file description table in order to determine which process originally opened a specific file and then allow performing the requested operations on the opened device or file.

**How to kill a process**

* **forcefully**
* **gracefully**

kill –HUP pid

kill -9 pid

kill -s TERM pid

List of Signals

There is an easy way to list down all the signals supported by your system. Just issue the kill -l command and it would display all the supported signals −

$ kill -l

1) SIGHUP 2) SIGINT 3) SIGQUIT 4) SIGILL

5) SIGTRAP 6) SIGABRT 7) SIGBUS 8) SIGFPE

9) SIGKILL 10) SIGUSR1 11) SIGSEGV 12) SIGUSR2

13) SIGPIPE 14) SIGALRM 15) SIGTERM 16) SIGSTKFLT

17) SIGCHLD 18) SIGCONT 19) SIGSTOP 20) SIGTSTP

21) SIGTTIN 22) SIGTTOU 23) SIGURG 24) SIGXCPU

25) SIGXFSZ 26) SIGVTALRM 27) SIGPROF 28) SIGWINCH

29) SIGIO 30) SIGPWR 31) SIGSYS 34) SIGRTMIN

35) SIGRTMIN+1 36) SIGRTMIN+2 37) SIGRTMIN+3 38) SIGRTMIN+4

39) SIGRTMIN+5 40) SIGRTMIN+6 41) SIGRTMIN+7 42) SIGRTMIN+8

43) SIGRTMIN+9 44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12

47) SIGRTMIN+13 48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14

51) SIGRTMAX-13 52) SIGRTMAX-12 53) SIGRTMAX-11 54) SIGRTMAX-10

55) SIGRTMAX-9 56) SIGRTMAX-8 57) SIGRTMAX-7 58) SIGRTMAX-6

59) SIGRTMAX-5 60) SIGRTMAX-4 61) SIGRTMAX-3 62) SIGRTMAX-2

63) SIGRTMAX-1 64) SIGRTMAX

**What is parent process ID**

In Linux, an executable stored on disk is called a **program**, and a program loaded into memory and running is called a **process**. When a process is started, it is given a unique number called process ID (**PID**) that identifies that process to the system.

**Print pid of current shell**

ps -p $$

PID TTY TIME CMD

1329 pts/0 00:00:00 bash

**How to clear a log file of running process**

echo > /var/log/httpd/access\_log

**What will happen if you delete a log file of running process**

Exactly how you do this depends on the programs. Some programs will close and reopen their logfiles when you send them a signal (e.g. HUP) and this can be used to prevent log records being written to a 'deleted' logfile, without disrupting service.

But you should not do this while something is still has the file open and still appending to it, because it would write to the now deleted file, and you would lose those log messages

**How do you check all the running process in the system**

1. top command : Display and update sorted information about processes.
2. atop command : Advanced System & Process Monitor.
3. htop command : Interactive process viewer.
4. pgrep command : Look up or signal processes based on name and other attributes.
5. pstree command : Display a tree of processes.
6. ps -aux : List process

**How do you check those process that are waiting for the resources**

**What init process is responsible for**

## Init

The kernel, once it is loaded, finds init in sbin and executes it.

When init starts, it becomes the parent or grandparent of all of the processes that start up automatically on your Linux system. The first thing init does, is reading its initialization file, /etc/inittab. This instructs init to read an initial configuration script for the environment, which sets the path, starts swapping, checks the file systems, and so on. Basically, this step takes care of everything that your system needs to have done at system initialization: setting the clock, initializing serial ports and so forth.

Then init continues to read the /etc/inittab file, which describes how the system should be set up in each run level and sets the default *run level*

**How do you you elevate the priority of a process:**

Changing Linux Process Priority

On the Linux system, all active processes have a priority and certain nice value. Processes with higher priority will normally get more CPU time than lower priority processes.

However, a system user with root privileges can influence this with the nice and renice commands.

From the output of the top command, the NI shows the process nice value:

Use the nice command to set a nice value for a process. Keep in mind that normal users can attribute a nice value from zero to 20 to processes they own.

Only the root user can use negative nice values.

To renice the priority of a process, use the renice command as follows:

$ renice +8 2687

$ renice +8 2103

**How do you you elevate the priority of a process:**

Changing Linux Process Priority

On the Linux system, all active processes have a priority and certain nice value. Processes with higher priority will normally get more CPU time than lower priority processes.

However, a system user with root privileges can influence this with the nice and renice commands.

From the output of the top command, the NI shows the process nice value:

Use the nice command to set a nice value for a process. Keep in mind that normal users can attribute a nice value from zero to 20 to processes they own.

Only the root user can use negative nice values.

To renice the priority of a process, use the renice command as follows:

$ renice +8 2687

$ renice +8 2103

**How many tables are there in iptables. What filter and nat table responsible for?**

iptables firewall is used to manage packet filtering and NAT rules.

On a high-level iptables might contain multiple tables. Tables might contain multiple chains. Chains can be built-in or user-defined. Chains might contain multiple rules. Rules are defined for the packets.

So, the structure is: iptables -> Tables -> Chains -> Rules. This is defined in the following diagram.

tables are bunch of chains, and chains are bunch of firewall rules.

## I. IPTABLES TABLES and CHAINS

IPTables has the following 4 built-in tables.

### 1. Filter Table

Filter is default table for iptables. So, if you don’t define you own table, you’ll be using filter table. Iptables’s filter table has the following built-in chains.

* INPUT chain – Incoming to firewall. For packets coming to the local server.
* OUTPUT chain – Outgoing from firewall. For packets generated locally and going out of the local server.
* FORWARD chain – Packet for another NIC on the local server. For packets routed through the local server.

### 2. NAT table

Iptable’s NAT table has the following built-in chains.

* PREROUTING chain – Alters packets before routing. i.e Packet translation happens immediately after the packet comes to the system (and before routing). This helps to translate the destination ip address of the packets to something that matches the routing on the local server. This is used for DNAT (destination NAT).
* POSTROUTING chain – Alters packets after routing. i.e Packet translation happens when the packets are leaving the system. This helps to translate the source ip address of the packets to something that might match the routing on the desintation server. This is used for SNAT (source NAT).
* OUTPUT chain – NAT for locally generated packets on the firewall.

### 3. Mangle table

Iptables’s Mangle table is for specialized packet alteration. This alters QOS bits in the TCP header. Mangle table has the following built-in chains.

* PREROUTING chain
* OUTPUT chain
* FORWARD chain
* INPUT chain
* POSTROUTING chain

### 4. Raw table

Iptable’s Raw table is for configuration excemptions. Raw table has the following built-in chains.

* PREROUTING chain
* OUTPUT chain

What is the difference between -I and -A while applying a rule in iptables

-A — Appends the iptables rule to the end of the specified chain. This is the command used to add a rule when rule order in the chain does not matter.

-I — Inserts a rule in a chain at a point specified by a user-defined integer value. If no number is specified, iptables places the command at the top of the chain.

Be aware when using the -A or -I option that the order of the rules within a chain are important for determining which rules apply to which packets.

iptables -A appends rules at the end of the ruleset whereas iptables -I inserts the rule at a specific position in the ruleset as you've pointed out.

use -A when you want to append a rule at the end or when it doesn't matter where your rule will be. If you want your rule at a specific position, use -I like this for example: iptables -I INPUT 6 -p tcp -j DROP (this will add a DROP statement for all tcp packets addressed to the host itself on position 6 in the INPUT ruleset.)

What is DNAT and SNAT, Explain both with an example:

Use-Case for Source NAT: A local client behind Firewall or NAT device wanted to browse Internet

Local Client IP: 10.10.10.10/24  
Website URL to visit: [http://www.quora.com](https://www.quora.com/)  
IP Address of Quora: 54.84.216.68  
Now when you type the URL in browser, your browser will establish a connection like this  
Source Address: 10.10.10.10  
Destination Address: 54.84.216.68

Now, when you send a TCP syn, destination has to send ack. Now, ACK will have a   
source address: 54.84.216.68  
Destination: 10.10.10.10

Now, as 10.10.10.10 is not globally unique, it may so happen that Quora may be using 10.10.10.10 in some local systems, so packets instead of going to you will reach their local system and you will not be able to establish connection.

So, what you do now is translate your source IP (10.10.10.10) to something which is globally routable (for example : 14.10.10.10).

Now, you will have source address as 14.10.10.10, your NAT device will need to maintain NAT Table.

Use case Destination NAT (DNAT)

Now imagine a scenario, you are hosting a website and your website local address is 172.19.18.10 (private IP), now people cannot connect to your website from internet because your IP is private and not-globally reachable.

So what you do now is to create a destination NAT entry where public IP is mapped to private IP.

You will have one public for example: 14.10.10.20 and you NAT it to 172.19.18.10 (Your local IP) so any request meant for 14.10.10.20 is actually translated to 172.19.18.10 by your NAT device.

Make a shell script that would configure a firewall as below:

#!/bin/bash

# Flush all current rules from iptables

iptables -F

# Set default policies for INPUT, FORWARD and OUTPUT chains

iptables -P INPUT DROP

iptables -P FORWARD ACCEPT

iptables -P OUTPUT ACCEPT

# Allow ssh from vagrant machine1 only

iptables -A INPUT -p tcp -s vagrant ip --dport 22 -j ACCEPT

# Allow port 80,443 from everywhere

iptables -A INPUT -p tcp --dport 80 -m conntrack --ctstate NEW,ESTABLISHED -j ACCEPT

iptables -A OUTPUT -p tcp --sport 80 -m conntrack --ctstate ESTABLISHED -j ACCEPT

iptables -A INPUT -p tcp --dport 443 -m conntrack --ctstate NEW,ESTABLISHED -j ACCEPT

iptables -A OUTPUT -p tcp --sport 443 -m conntrack --ctstate ESTABLISHED -j ACCEPT

Allow ping from outside

iptables -A INPUT -p icmp --icmp-type echo-request -j ACCEPT

iptables -A OUTPUT -p icmp --icmp-type echo-reply -j ACCEPT

#Allow loopback Access

iptables -A INPUT -i lo -j ACCEPT

iptables -A OUTPUT -o lo -j ACCEPT

#Allow DNS

# DNS

# UDP

iptables -A INPUT -i eth0 -p udp --sport 53 -m state --state ESTABLISHED -j ACCEPT

iptables -A OUTPUT -o eth0 -p udp --dport 53 -m state --state NEW,ESTABLISHED -j ACCEPT

# TCP

iptables -A INPUT -i eth0 -p tcp --sport 53 -m state --state ESTABLISHED -j ACCEPT

iptables -A OUTPUT -o eth0 -p tcp --dport 53 -m state --state NEW,ESTABLISHED -j ACCEPT

#Allow rsync from outside

iptables -A INPUT -i eth0 -p tcp -s 192.168.101.0/24 --dport 873 -m state --state NEW,ESTABLISHED -j ACCEPT

iptables -A OUTPUT -o eth0 -p tcp --sport 873 -m state --state ESTABLISHED -j ACCEPT

#Allow postfix or sendmail

iptables -A OUTPUT -p tcp --sport 25 -j ACCEPT

iptables -A OUTPUT -p tcp --sport 587 -j ACCEPT

# Save settings

/sbin/service iptables save

# List rules

iptables -L –v

Write a rule that would prevent you from DDOS attack [HINT: 25 connections/minute]:

iptables -A INPUT -p tcp --dport 80 -m limit --limit 25/minute --limit-burst 100 -j ACCEPT.