
Assignment 1 – Study of System Commands and System Calls for Simple Client-Server Programs

Date: 19/08/2022

AIM:

To study the given system commands and calls for simple client-server programs.

SYSTEM COMMANDS:

ifconfig:

This command is used for displaying current network configuration information, setting up an ip address, netmask, or broadcast address to a network interface, creating an alias for the network interface, setting up hardware address, and enabling or disabling network interfaces.

Options:

-a : Display all interfaces.

-s : Display short list.

-v: Display more detail about execution

Syntax: ifconfig [-v] [-a] [-s] [interface]

ifconfig [-v] interface [aftype] options | address…

```
root@splcse:~# ifconfig
enp3s0    Link encap:Ethernet  HWaddr 40:a8:f0:5c:e7:bc
          inet addr:10.6.10.2  Bcast:10.6.255.255  Mask:255.255.0.0
          inet6 addr: fe80::42a8:f0ff:fe5c:e7bc/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:121243 errors:0 dropped:0 overruns:0 frame:0
          TX packets:34223 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:78388673 (78.3 MB)  TX bytes:10901540 (10.9 MB)
          Interrupt:18

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:65536  Metric:1
          RX packets:7910 errors:0 dropped:0 overruns:0 frame:0
          TX packets:7910 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1
          RX bytes:635823 (635.8 KB)  TX bytes:635823 (635.8 KB)
```

```
root@splcse:~# ifconfig -s
Iface   MTU Met   RX-OK RX-ERR RX-DRP RX-OVR    TX-OK TX-ERR TX-DRP TX-OVR Flg
enp3s0     1500 0    135274      0      0 0         39906      0      0      0 B
MRU
lo        65536 0      9960      0      0 0          9960      0      0      0 L
RU
```

nslookup:

It is the name of a program that lets an Internet server administrator or any computer user enter a host name (for example, "whatis.com") and find out the corresponding IP address or domain name system (DNS) record. The user can also enter a command for it to do a reverse DNS lookup and find the host name for an IP address that is specified.

Syntax: nslookup [-option] [name 1 -] [server]
Example: nslookup www.microsoft.com

```
root@splcse:~# nslookup
> google.com
Server:         127.0.1.1
Address:        127.0.1.1#53

Non-authoritative answer:
Name:   google.com
Address: 142.250.195.78
> www.facebook.com
Server:         127.0.1.1
Address:        127.0.1.1#53

Non-authoritative answer:
www.facebook.com        canonical name = star-mini.c10r.facebook.com.
Name:   star-mini.c10r.facebook.com
Address: 157.240.239.35
>
```

traceroute:

It is a command used in network troubleshooting for mapping the path packets travel through the network. It tracks the route packets taken from an IP network on their way to a given host. It utilizes the IP protocol's time to live (TTL) field and attempts to elicit an ICMP TIME_EXCEEDED response from each gateway along the path to the host.

Options:

-I : use ICHP echo for probes.

-T : use TCP for probes.

-d : socket level debugging.

Syntax: traceroute [domain]

Example: traceroute www.microsoft.com

```
root@splcse:~# traceroute www.spotify.com
traceroute to www.spotify.com (35.186.224.25), 30 hops max, 60 byte packets
 1  10.6.0.1 (10.6.0.1)  5.323 ms  5.330 ms  5.496 ms
 2  10.106.0.2 (10.106.0.2)  0.414 ms  0.437 ms  0.422 ms
 3  * * *
 4  * * *
 5  * * *
 6  * * *
 7  * * *
 8  * * *
 9  * * *
10  * * *
11  * * *
12  * * *
13  * * *
14  * * *
15  * * *
16  * * *
17  * * *
18  * * *
19  * * *
20  * * *
21  * * *
22  * * *
23  * * *
24  * * *
25  * * *
26  * * *
27  * * *
28  * * *
29  * * *
30  * * *
```

tcpdump:

It is a command line utility that allows you to capture and analyze network traffic going through your system. It is often used to help troubleshoot network issues, as well as a security tool. *Tcpdump* will, if not run with the **-c** flag, continue capturing packets until it is interrupted by a SIGINT signal.

Options:

-A : used to print each packet.

-b : used to print AS number in BCTP packets.

-c : Count - used to count packets.

Syntax: tcpdump [-options] [-value]

```
root@splcse:~# tcpdump -D
1.enp3s0 [Up, Running]
2.any (Pseudo-device that captures on all interfaces) [Up, Running]
3.lo [Up, Running, Loopback]
4.nflog (Linux netfilter log (NFLOG) interface)
5.nfqueue (Linux netfilter queue (NFQUEUE) interface)
6.usbmon1 (USB bus number 1)
7.usbmon2 (USB bus number 2)
8.usbmon3 (USB bus number 3)
9.usbmon4 (USB bus number 4)
10.usbmon5 (USB bus number 5)
11.usbmon6 (USB bus number 6)
12.usbmon7 (USB bus number 7)
13.usbmon8 (USB bus number 8)
14.usbmon9 (USB bus number 9)
```

```
root@splcse:~# tcpdump -c 5
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on enp3s0, link-type EN10MB (Ethernet), capture size 262144 bytes
11:08:03.734738 IP 10.6.10.2.818 > 10.6.10.98.810: UDP, length 96
11:08:03.735012 IP 10.6.10.98.810 > 10.6.10.2.818: UDP, length 32
11:08:03.736060 IP 10.6.10.2.37745 > 10.6.10.98.sunrpc: Flags [S], seq 399268594
6, win 29200, options [mss 1460,sackOK,TS val 853916 ecr 0,nop,wscale 7], length
 0
11:08:03.736225 IP 10.6.10.98.sunrpc > 10.6.10.2.37745: Flags [S.], seq 86244022
9, ack 3992685947, win 28960, options [mss 1460,sackOK,TS val 2187473 ecr 853916
,nop,wscale 7], length 0
11:08:03.736256 IP 10.6.10.2.37745 > 10.6.10.98.sunrpc: Flags [.], ack 1, win 22
9, options [nop,nop,TS val 853916 ecr 2187473], length 0
5 packets captured
38 packets received by filter
31 packets dropped by kernel
root@splcse:~#
```

ping:

This command is used to troubleshoot and diagnose network connectivity issues. It is used to check whether the host is reachable. The ping command sends ICMP Echo Request (ECHO_REQUEST) packets to the host once per second. Each packet that is echoed back via an ICMP Echo Response packet is written to the standard output, including round-trip time.

Options:

-c: count stop after sending.

-d: set the SO.DEBUG option to be used.

-l : preload

Syntax: ping [-options]

```
root@splcse:~# ping -c 5 www.apple.com
PING e6858.dscx.akamaiedge.net (23.1.37.20) 56(84) bytes of data.
64 bytes from a23-1-37-20.deploy.static.akamaitechnologies.com (23.1.37.20): icm
p_seq=1 ttl=60 time=3.51 ms
64 bytes from a23-1-37-20.deploy.static.akamaitechnologies.com (23.1.37.20): icm
p_seq=2 ttl=60 time=3.39 ms
64 bytes from a23-1-37-20.deploy.static.akamaitechnologies.com (23.1.37.20): icm
p_seq=3 ttl=60 time=3.38 ms
64 bytes from a23-1-37-20.deploy.static.akamaitechnologies.com (23.1.37.20): icm
p_seq=4 ttl=60 time=3.37 ms
64 bytes from a23-1-37-20.deploy.static.akamaitechnologies.com (23.1.37.20): icm
p_seq=5 ttl=60 time=3.43 ms

--- e6858.dscx.akamaiedge.net ping statistics ---
5 packets transmitted, 5 received, 0% packet loss, time 4006ms
rtt min/avg/max/mdev = 3.374/3.422/3.512/0.061 ms
root@splcse:~#
```

netstat:

It prints information about the Linux networking subsystem. The type of information printed is controlled by the first argument. The type of information printed is controlled by the first argument. The network statistics can show details about individual network connections, overall and protocol-specific networking statistics.

Options:

-r: display kernel routing tables.

-g : display multicast routing tables group membership.

-c: display list of connections.

-s : display summary.

Syntax: netstat [address] [-options]

```
root@splcse:~# netstat -r
Kernel IP routing table
Destination     Gateway         Genmask         Flags   MSS Window  irtt Iface
default         10.6.0.1        0.0.0.0         UG        0 0          0 enp3s0
10.6.0.0        *               255.255.0.0     U         0 0          0 enp3s0
link-local      *               255.255.0.0     U         0 0          0 enp3s0
root@splcse:~#
```

```
root@splcse:~# netstat -g
IPv6/IPv4 Group Memberships
Interface       RefCnt Group
--------------- ------ ---------------------
lo              1      all-systems.mcast.net
enp3s0          1      224.0.0.251
enp3s0          1      all-systems.mcast.net
lo              1      ip6-allnodes
lo              1      ff01::1
enp3s0          1      ff02::fb
enp3s0          1      ff02::1:ff5c:e7bc
enp3s0          1      ip6-allnodes
enp3s0          1      ff01::1
root@splcse:~#
```

`

SYSTEM CALLS:

1. socket()

socket() creates an endpoint for communication and returns a file descriptor that refers to that endpoint. The file descriptor returned by a successful call will be the lowest-numbered file descriptor not currently open for the process.

#include<sys/socket.h>
int socket(int domain, int type, int protocol);

domain - specifies a communication domain; this selects the protocol family which will be used for communication.
type - in addition to specifying a socket type, it may include the bitwise OR of particular values, to modify the behavior of the socket()
protocol - specifies a particular protocol to be used with the socket.

2. bind()

When a socket is created with socket(2), it exists in a namespace (address family) but has no address assigned to it. bind() assigns the address specified by *addr*, and assigns a name to the socket. Traditionally, this operation is called "assigning a name to a socket".

#include <sys/socket.h>
int bind(int *sockfd***, const struct sockaddr ****addr***, socklen_t** *addrlen***)**

On success, zero is returned. On error, -1 is returned, and *errno* is set to indicate the error.

sockfd - file descriptor referring to a socket.
addr - specifies the particular address to be binded to socket.
addrlen - specifies the size of the address structure pointed to by *addr*.

3. listen()

listen() marks the socket referred to by *sockfd* as a passive socket, that is, as a socket that will be used to accept incoming connection requests

#include <sys/socket.h>
int listen(int *sockfd***, int** *backlog***);**

On success, zero is returned. On error, -1 is returned, and *errno* is set to indicate the error.

sockfd - a file descriptor that refers to a socket of type **SOCK_STREAM** or **SOCK_SEQPACKET**.
backlog - defines the maximum length to which the queue of pending connections for *sockfd* may grow.

4. **connect()**

connect() connects the socket to an address by initiating a connection. For stream sockets, the connect() call attempts to establish a connection between two sockets. For datagram sockets, the connect() call specifies the peer for a socket.

#include <sys/socket.h>
int connect(int *sockfd***, const struct sockaddr ****addr***, socklen_t** *addrlen***);**

On success, zero is returned. On error, -1 is returned, and *errno* is set to indicate the error.

sockfd - file descriptor referring to a socket.
addr - specifies the particular address to be binded to socket.
addrlen - specifies the size of the address structure pointed to by *addr*.

5. accept()

The accept() system call is used with connection-based socket types (**SOCK_STREAM**, **SOCK_SEQPACKET**). It extracts the first connection request on the queue of pending connections for the listening socket, *sockfd*, creates a new connected socket, and returns a new file descriptor referring to that socket. The newly created socket is not in the listening state. The original socket *sockfd* is unaffected by this call.

#include <sys/socket.h>
int accept(int *sockfd***, struct sockaddr *restrict** *addr***, socklen_t *restrict** *addrlen***);**

On success, these system calls return a file descriptor for the accepted socket (a nonnegative integer). On error, -1 is returned, *errno* is set to indicate the error, and *addrlen* is left unchanged.

sockfd - file descriptor referring to a socket.
addr - specifies the particular address to be binded to socket.
addrlen - specifies the size of the address structure pointed to by *addr*.

6. close()
close() closes a file descriptor, so that it no longer refers to any file and may be reused. The resources associated with the file descriptor, if it is the last instance of its type, are also freed.

#include <unistd.h>
int close(int *fd***);**

close() returns zero on success. On error, -1 is returned, and *errno* is set to indicate the error.
fd - file descriptor acting as a reference to the required object.

7. bzero()

The bzero() function erases the data in the *n* bytes of the memory starting at the location pointed to by *s*, by writing zeros (bytes containing '\0') to that area.

#include <strings.h>
void bzero(void *s, size_t n);

s - pointer to location in memory to write zeros

8. htons, htonl, ntohs, ntohl()

The functions are used to convert values between host and network byte order. The htonl() function converts the unsigned integer *hostlong* from host byte order to network byte order. The htons() function converts the unsigned short integer *hostshort* from host byte order to network byte order. The ntohl() function converts the unsigned integer *netlong* from network byte order to host byte order. The ntohs() function converts the unsigned short integer *netshort* from network byte order to host byte order.

include <arpa/inet.h>
uint32_t htonl(uint32_t *hostlong***);**
uint16_t htons(uint16_t *hostshort***);**
uint32_t ntohl(uint32_t *netlong***);**
uint16_t ntohs(uint16_t *netshort***);**

9. read()

#include <unistd.h>
ssize_t read(int *fd***, void *** *buf***, size_t** *count***);**

read() attempts to read up to *count* bytes from file descriptor *fd* into the buffer starting at *buf*.
On success, the number of bytes read is returned (zero indicates end of file), and the file position is advanced by this number.
On error, -1 is returned, and *errno* is set to indicate the error. In this case, it is left unspecified whether the file position (if any) changes.

10. write()

#include <unistd.h>
ssize_t write(int *fd***, const void *** *buf***, size_t** *count***);**

write() writes up to *count* bytes from the buffer starting at *buf* to the file referred to by the file descriptor *fd*. The number of bytes written may be less than *count* if, for example, there is insufficient space on the underlying physical medium.

On success, the number of bytes written is returned. On error, -1 is returned, and *errno* is set to indicate the error.

11. send()

The system call send() is used to transmit a message to another socket. The send() call may be used only when the socket is in a *connected* state (so that the intended recipient is known). The only difference between send() and write(2) is the presence of *flags*. With a zero *flags* argument, send() is equivalent to write(2).

#include <sys/socket.h>
ssize_t send(int *sockfd***, const void ****buf***, size_t** *len***, int** *flags***);**

On success, these calls return the number of bytes sent. On error, -1 is returned, and *errno* is set to indicate the error.
sockfd - file descriptor referring to a socket.
buff - specifies the message to be sent to socket.
flag - specifies the mode of the system call.

12. receive()

The **recv**() call is used to receive messages from a socket. It may be used to receive data on both connectionless and connection-oriented sockets.

#include <sys/socket.h>
ssize_t recv(int *sockfd***, void ****buf***, size_t** *len***, int** *flags***);**

These calls return the number of bytes received, or -1 if an error occurred.In the event of an error, *errno* is set to indicate the error.
sockfd - file descriptor referring to a socket.
buff - specifies the message received by socket.
len - specifies the length of the message received.
flag - specifies the mode of the system call.

13. sendto()

#include <sys/socket.h>

ssize_t sendto(int *sockfd***, const void *** *buf***, size_t** *len***, int** *flags***, const struct sockaddr *** *dest_addr***, socklen_t** *addrlen***);**

sendto() transmits messages to a connection-mode socket.
sockfd - file descriptor of sending socket
dest_t - address of the target
addrlen - specifies the size of the address structure pointed to by *addr*.
flag - specifies the mode of the system call.

14. recvfrom()

#include <sys/socket.h>
ssize_t recvfrom(int *socket***, void *restrict** *buffer***, size_t** *length***,**
int *flags***, struct sockaddr *restrict** *address***,**
socklen_t *restrict *address_len***);**

Upon completion, it returns the length of message in bytes. If no messages are there to be received, zero is returned. Otherwise, -1 is returned and errno is set to indicate the error.
sockfd - file descriptor of sending socket
addrlen - specifies the size of the address structure pointed to by *address*.
flag - specifies the mode of the system call.

15. select()

The *select()* function indicates which of the specified file descriptors is ready for reading, ready for writing, or has an error condition pending. If the specified condition is false for all of the specified file descriptors, *select()* blocks, up to the specified timeout interval, until the specified condition is true for at least one of the specified file descriptors.

#include <sys/time.h>
int select(int *nfds***, fd_set *** *readfds***, fd_set *** *writefds***,**
fd_set * *errorfds***, struct timeval *** *timeout***);**
readfds - The file descriptors in this set are watched to see if they are ready for reading.
writefds - The file descriptors in this set are watched to see if they are ready for writing.
exceptfds - The file descriptors in this set are watched for "exceptional conditions".
nfds - This argument should be set to the highest-numbered file descriptor
timeout - The *timeout* argument is a *timeval* structure that specifies the interval

---

16. setsockopt()

The *setsockopt*() function shall set the option specified by the *option_name* argument, at the protocol level specified by the *level* argument, to the value pointed to by the *option_value* argument for the socket associated with the file descriptor specified by the *socket* argument. The *level* argument specifies the protocol level at which the option resides.

#include <sys/socket.h>
int setsockopt(int *socket***, int** *level***, int** *option_name***,**
const void ****option_value*, socklen_t** *option_len***);**

Upon successful completion, *setsockopt*() shall return 0. Otherwise, -1 shall be returned and *errno* set to indicate the error.

17. fcntl()

fcntl() performs one of the available operations on the open file descriptor *fd*. The operation is determined by *cmd*. **fcntl**() can take an optional third argument. Whether or not this argument is required is determined by *cmd*.

#include <fcntl.h>
int fcntl(int *fd***, int** *cmd***, ... /*** *arg* ***/);**

For a successful call, the return value depends on the operation. On error, -1 is returned, and *errno* is set to indicate the error.

18. getpeername()

getpeername() returns the address of the peer connected to the socket *sockfd*, in the buffer pointed to by *addr*. The *addrlen* argument should be initialized to indicate the amount of space pointed to by *addr*. On return it contains the actual size of the name returned (in bytes).

#include <sys/socket.h>
int getpeername(int *sockfd***, struct sockaddr *restrict** *addr***,**
socklen_t *restrict *addrlen***);**

On success, zero is returned. On error, -1 is returned, and *errno* is set to indicate the error.

19. gethostname()

#include <unistd.h>
int gethostname(char ****name*, size_t** *len***);**

gethostname() returns the null-terminated hostname in the character array *name*, which has a length of *len* bytes. If the null-terminated hostname is too large to fit, then the name is truncated, and no error is returned.
On success, zero is returned. Otherwise, -1 is returned and *errno* is set to indicate the error.

20. gethostbyname()

The gethostbyname() function returns a structure of type *hostent* for the given host name. Here *name* is either a hostname or an IPv4 address.

#include <netdb.h>
struct hostent *gethostbyname(const char *_name_);

On successful completion of the operation a hostent structure is returned. If an error is encountered, null object is returned and h_errno holds the error number indicating the type of error that has occurred.

21. gethostbyaddr()

The gethostbyaddr() function returns a structure of type *hostent* for the given host address *addr* of length *len* and address type *type*.

#include <netdb.h>
struct hostent *gethostbyaddr(const void *_addr_, socklen_t _len_, int _type_);

On successful completion of the operation a hostent structure is returned. If an error is encountered, null object is returned and h_errno holds the error number indicating the type of error that has occurred.

LEARNING OUTCOMES:

- Various system calls and commands were studied.
- The common options used with each command were observed.
- The syntax and header files required for each command were understood.
- The purpose and applications of each command were studied.