BCS-052 (Network Programming and Administration)

July-2013 Solved Assignment

Ans-1 (a)

A run level is a state of **init** and the whole system that defines what system services are operating. Run levels are identified by numbers. Some system administrators use run levels to define which subsystems are working, e.g., whether X is running, whether the network is operational, and so on. Others have all subsystems always running or start and stop them individually, without changing run levels, since run levels are too coarse for controlling their systems. You need to decide for yourself, but it might be easiest to follow the way your Linux distribution does things.

The following table defines how most Linux Distributions define the different run levels. However, run-levels 2 through 5 can be modified to suit your own tastes.

Table 1: Linux Runlevel Descriptions

Runlevel	**Runlevel Description**
Runlevel 0	The halt runlevel - this is the runlevel at which the system shuts down. For obvious reasons it is unlikely you would want this as your default runlevel.
Runlevel 1	Single runlevel. This causes the system to start up in a single user mode under which only the root user can log in. In this mode the system does not start any networking or X windowing, X or multi-user services. This run level is ideal for system administrators to perform system maintenance or repair activities.
Runlevel 2	Boots the system into a multi-user mode with text based console login capability. This runlevel does not, however, start the network.
Runlevel 3	Similar to runlevel 2 except that networking services are started. This is the most common runlevel forserver based systems that do not require any kind of graphical desktop environment.

Runlevel 4	Undefined runlevel. This runlevel can be configured to provide a custom boot state.
Runlevel 5	Boots the system into a networked, multi-user state with X Window System capability. By default thegraphical desktop environment will start at the end of the boot process. This is the most common run level for desktop or workstation use.
Runlevel 6	Reboots the system. Another runlevel that you are unlikely to want as your default.

Ans-1 (b)

The TCP/IP Model separates networking functions into discrete layers. Each layer performs a specific function and is transparent to the layer above it and the layer below it. Network models are used to conceptualize how networks should work, so that hardware and network protocols can interoperate. The TCP/IP model is one of the two most common network models, the other being the OSI Model.

The TCP/IP Model of networking is a different way of looking at networking. Because the model was developed to describe TCP/IP, it is the closest model of the Internet, which uses TCP/IP.

The TCP/IP network model breaks down into four (4) layers:

- Application Layer
- Transport Layer
- Internet Layer
- Network Access Layer

TCP/IP Model Layers

Application Layer

The Application Layer provides the user with the interface to communication. This could be your web browser, e-mail client (Outlook, Eudora or Thunderbird), or a file transfer client.

The Application Layer is where your web browser, a telnet, ftp, e-mail or other client application runs. Basically, any application that rides on top of TCP and/or UDP that uses a pair of virtual network sockets and a pair of IP addresses.

The Application Layer sends to, and receives data from, the Transport Layer.

Transport Layer

The Transport Layer provides the means for the transport of data segments across the Internet Layer. The Transport Layer is concerned with end-to-end (host-to-host) communication.
Transmission Control Protocol provides reliable, connection-oriented transport of data between two endpoints (sockets) on two computers that use Internet Protocol to communicate.
User Datagram Protocol provides unreliable, connectionless transport of data between two endpoints (sockets) on two computers that use Internet Protocol to communicate.
The Transport Layer sends data to the Internet layer when transmitting and sends data to the Application Layer when receiving.

Internet Layer

The Internet Layer provides connectionless communication across one or more networks, a global logical addressing scheme and packetization of data. The Internet Layer is concerned with network to network communication.
The Internet Layer is responsible for packetization, addressing and routing of data on the network. Internet Protocol provides the packetization, logical addressing and routing functions that forward packets from one computer to another.
The Internet Layer communicates with the Transport Layer when receiving and sends data to the Network Access Layer when transmitting.

Network Access Layer

The Network Access Layer provides access to the physical network.
This is your network interface card. Ethernet, FDDI, Token Ring, ATM, OC, HSSI, or even Wi-Fi are all examples of network interfaces. The purpose of a network interface is to allow your computer to access the wire, wireless or fiber optic network infrastructure and send data to other computers.
The Network Access Layer transmits data on the physical network when sending and transmits data to the Internet Layer when receiving.

All Internet-based applications and their data, whether it is a web browser downloading a web page, Microsoft Outlook sending an e-mail, a file, an instant message, a Skype video or voice call; the data is chopped into data segments and encapsulated in Transport Layer Protocol Data Units or PDU's (TCP or UDP segments). The Transport Layer PDU's are then encapsulated in Internet Layer's Internet Protocol packets. The Internet Protocol packets are then chopped into frames at the Network Access layer and transmitted across the physial media (copper wires, fiber optic cables or the air) to the next station in the network.
The OSI Model uses seven layers, and differs quite a bit from the TCP/IP model. The TCP/IP model does a better job of representing how TCP/IP works in a network, but the OSI Model is

still the networking model most technical people refer to during troubleshooting or network architecture discussions.

ANS-1 (c)

Identify the address classes of the following IP address:

(a) 255.255.130.0 -> Class E
(b) 216.11.5.2 -> Class C
(c) 150.156.1.1 ->Class B

ANS -1 (d)

Webpages have no memories. A user going from page to page will be treated by the website as a completely new visitor. Session cookies enable the website you are visiting to keep track of your movement from page to page so you don't get asked for the same information you've already given to the site. Cookies allow you to proceed through many pages of a site quickly and easily without having to authenticate or reprocess each new area you visit.

Session cookies allow users to be recognized within a website so any page changes or item or data selection you do is remembered from page to page. The most common example of this functionality is the shopping cart feature of any e-commerce site. When you visit one page of a catalog and select some items, the session cookie remembers your selection so your shopping cart will have the items you selected when you are ready to check out. Without session cookies, if you click CHECKOUT, the new page does not recognize your past activities on prior pages and your shopping cart will always be empty.

You can adjust your session cookies through the settings feature of your browser.

Without cookies, websites and their servers have no memory. A cookie, like a key, enables swift passage from one place to the next. Without a cookie every time you open a new web page the server where that page is stored will treat you like a completely new visitor.

Websites typically use session cookies to ensure that you are recognised when you move from page to page within one site and that any information you have entered is remembered. For example, if an e-commerce site did not use session cookies then items placed in a shopping basket would disappear by the time you reach the checkout. You can choose to accept session cookies by changing the settings in your browser.

ANS -2 (a)

```
UDP is good for sending messages from one system to another when the
  order isn't important and you don't need all of the messages to get to
  the other machine.  This is why I've only used UDP once to write the
  example code for the faq.  Usually TCP is a better solution.  It saves
  you having to write code to ensure that messages make it to the
  desired destination, or to ensure the message ordering.  Keep in mind
  that every additional line of code you add to your project in another
  line that could contain a potentially expensive bug.

  If you find that TCP is too slow for your needs you may be able to get
  better performance with UDP so long as you are willing to sacrifice
  message order and/or reliability.

  UDP must be used to multicast messages to more than one other machine
  at the same time.  With TCP an application would have to open separate
  connections to each of the destination machines and send the message
  once to each target machine.  This limits your application to only
  communicate with machines that it already knows about.
```

<u>TCP provides a mechanism to handle urgent data</u>

- Urgent data is received before octets already in the stream
- Sender:
  - Sets urgent bit in segment header
  - Puts urgent data at the beginning of the data field
  - Sets urgent pointer to the end of the urgent data
- Receiver:
  - Notified of the urgent data as soon as it arrives
  - Enters “urgent mode” until all urgent data has been consumed
  - Returns to “normal mode”

ANS -2(b)

(i)

Client program

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <unistd.h>
#include <netdb.h>
#include <string.h>
#include <errno.h> #include <arpa/inet.h> #include <sys/errno.h> #include <netinet/in.h> #include
<memory.h>
const int MAXSTR=1024; const long PORTNUM=54715;
main( int argc, char *argv[])
{
int portNumber;
char hostName[MAXSTR];
if ( (argc != 5) && (argc != 3) && (argc != 1) )
{
printf ("Invalid command line.\n"); exit(1);
}
if ( argc == 1 )
{
strcpy(hostName,"ernie.eecs.uic.edu"); portNumber = PORTNUM;
}
else if (argc == 3 )
{
if (!strcmp(argv[1],"-h"))
{
strcpy(hostName,argv[2]); portNumber = PORTNUM;
}
else if (!strcmp(argv[1],"-p"))
{
portNumber = atoi(argv[2]); strcpy(hostName,"ernie.eecs.uic.edu");
}
else
{
printf("Invalid command line.\n"); exit(1);
}
}
else
{
if ((!strcmp(argv[1],"-h")) && (!strcmp(argv[3],"-p")))
{
strcpy(hostName,argv[2]); portNumber = atoi(argv[4]);
}
else
{
printf("Invalid command line.\n"); exit(1);
}
}
if (portNumber < 10000)
{
printf("Port number has to be atleast 10000.\n"); exit(1);
}
printf("host =%s port = %d" ,hostName, portNumber );
struct hostent *hostptr;
if ((hostptr = gethostbyname(hostName)) == NULL)
{
printf("Error locating host : %s\n",hostName); exit(1);
}
struct sockaddr_in serv_addr;
bzero((char *)&serv_addr,sizeof(serv_addr)); serv_addr.sin_family = AF_INET;
memcpy((char *)&serv_addr.sin_addr,(char *)hostptr->h_addr,hostptr->h_length); serv_addr.sin_port =
htons(portNumber);
int sockfd,newsockfd;
if ((sockfd = socket(AF_INET,SOCK_STREAM,0)) < 0)
{
printf("client : socket error.\n"); exit(1);
}
if ( connect(sockfd,(struct sockaddr *)&serv_addr,sizeof(serv_addr)) <0)
{
printf("client : connect error.\n"); exit(1);
}
```

```
char data[MAXSTR]; char answer[MAXSTR]; char filename[MAXSTR];
printf("String to send to the server : ");
scanf("%s",data); data[strlen(data)] = '\0';
write(sockfd,data,strlen(data)); read(sockfd,(void *)answer,MAXSTR);
printf("%s\n",answer);
int ch;
do{
printf("Enter the filename :"); scanf("%s",filename);
write(sockfd,filename,strlen(filename));
read(sockfd,(void *)answer,MAXSTR); while(strlen(answer)>0){
printf("%s",answer);
read(sockfd,(void *)answer,MAXSTR);
}
printf("\n\nWant to continue :"); ch=getchar();
}
while(ch=='y'|| ch=='Y');
close(sockfd);
}
```

(ii) Server program

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <memory.h>
#include <string.h>
#include <errno.h>
#include <sys/errno.h>
const int MAXSTR=1024; // Maximum String length
const long PORTNUM=54715; // The "Well-known" port number
//const int PASSWORD = 21; // The password that the server expects.
void handleRequest(int); // Function to handle clients' request(s)
//int numberChildren; // The current number of child processes.
//void childExit(int); // signal handler for SIGCHLD
main( int argc, char *argv[])
```

```
{
int portNumber;
//char executableName[MAXSTR]; //struct sigaction action;
// Validate and read from the command line if ( (argc != 3) && (argc != 1) )
{
printf("Invalid command line.\n"); exit(1);
}
if ( argc == 1 )
{
portNumber = PORTNUM;
}
else
{
if (!strcmp(argv[1],"-p"))
{
portNumber = atoi(argv[2]);
}
else
{
printf ("Invalid command line.\n"); exit(1);
}
}
if (portNumber < 10000)
{
printf("Port number has to be atleast 10000.\n"); exit(1);
}
// At this point, the command line has been validated.
int sockfd,newsockfd,clilen,childpid; struct sockaddr_in serv_addr,cli_addr;
// Create a new TCP socket...
if ((sockfd = socket(AF_INET,SOCK_STREAM,0)) < 0)
{
printf("Cant open stream socket.\n"); exit(0);
}
bzero((char *)&serv_addr,sizeof(serv_addr)) ;
serv_addr.sin_family = AF_INET; serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
serv_addr.sin_port = htons(portNumber);
if (bind(sockfd,(struct sockaddr *)&serv_addr,sizeof(serv_addr)) < 0)
{
printf("Cant bind local address.\n"); exit(1);
}
```

```
int status;
printf(" Connected to port : %d \n",portNumber);
char buf[MAXSTR],ack[MAXSTR];
{
clilen = sizeof(cli_addr);
```

- Accept a client's request, and get the client's address info into the local variable cli_addr.

```
newsockfd = accept(sockfd,(struct sockaddr *)&cli_addr,&clilen);
```

- After accepting the connection, all transaction with this client would happen with the new socket
descriptor - newsockfd.

```
if ((newsockfd < 0) && (errno != EINTR)) printf(" server : accept error.\n");
else if (newsockfd > 0)
{
handleRequest(newsockfd); close(newsockfd);
}
}
}
void handleRequest(int newsockfd)
{
char buf[MAXSTR],execName[MAXSTR],ack[MAXSTR]; char data[MAXSTR];
char filename[MAXSTR]; char content[MAXSTR];
// Read from the socket, for the password and the requested command. read(newsockfd,(void
*)buf,MAXSTR); sscanf(buf,"%s\n",execName);
sprintf(ack,"Server echo : %s",execName); printf("From client: %s",execName);
// Write back the response to the socket write(newsockfd,ack,strlen(ack));
read(newsockfd,(void *)filename,MAXSTR);
do{
FILE *fp; fp=fopen(filename,"r");
while(fread(content,1,MAXSTR,fp)) write(newsockfd,content,strlen(content));
fclose(fp);
read(newsockfd,(void *)filename,MAXSTR);
}
while(strlen(filename)>0);
}
```

ANS-2(C)

No matter how hard you try, you can't be everywhere at once. Servers and workstations that you need to support can be based in various locations, sometimes even thousands of miles away. Physical access to these systems on short notice is impossible. Fortunately, there are several widely used remote administration methods for Linux that let you cross that distance in little more than a few mouse clicks. Here are four major Linux remote administration utilities and some pluses and minuses for each.

Using Secure Shell
Secure Shell (SSH) is probably the most popular remote administration tool. SSH offers command line access over an encrypted tunnel. Many Linux distributions come with an SSH server already installed. As with any tool, patches must be installed and restrictions should be put in place to keep unauthorized users from using the service.

Once you have an SSH server installed, configuring it is fairly easy. There are a couple of major settings that you should have in place to minimize risk, including:

- **PermitRootLogin**—This value should be set to No, since root should never log in remotely. If you want to administer the box, create a normal user and SSH in with that account. Once in, you can use the su command to log in as root.
- **X11 Forwarding—**This value will be used for getting a graphical connection. If you just want to use the console, you can set this value to No; otherwise, set it to Yes.

Using Telnet
Telnet, much like SSH, allows for remote console-level access to a system. The major difference between these two is that Telnet is not encrypted, meaning it is open for all to see. Most major Linux distributions come with a Telnet package that can be installed, but is not by default.

You can install a Telnet daemon using your distribution's RPM installer. Because of the inherent insecurity in using an unencrypted connection, most distributions have the Telnet service disabled. Most Telnet daemons run through Xinetd, a management system for services. To enable a Telnet daemon that runs through Xinetd, edit the /etc/xinetd.d/telnet file and change the value for Disable to No. Then, type *service xinetd restart* to have the changes take effect.

Connecting to a server with Telnet is even easier than with SSH. Nearly every operating system has a built-in Telnet client

Using FTP
FTP, a file transfer mechanism, can't be used for major remote administration but is great for transferring files that you may need on the remote workstation. FTP is not encrypted, but there's an encrypted alternative, SFTP.

Most distributions come with an FTP service that you can install by using the RPM installer. Note that most of the major Linux FTP servers have had major vulnerabilities found, so make sure to get the latest version. If you have an SSH server installed, you can also use SFTP through it.

Using X
Using an X Server to manage a system remotely allows you full graphical access to the remote machine. X is the graphical interface used by most Linux distributions. It can function as both a local and remote graphical server.

There are several ways to establish an X session, but the most secure of these is to connect to your server with SSH or Telnet and spawn back your X window. This method prohibits users that are not authenticated from getting access. X sessions are a fairly secure and very efficient method for managing remote servers. However, many corporate firewalls block outbound X traffic.

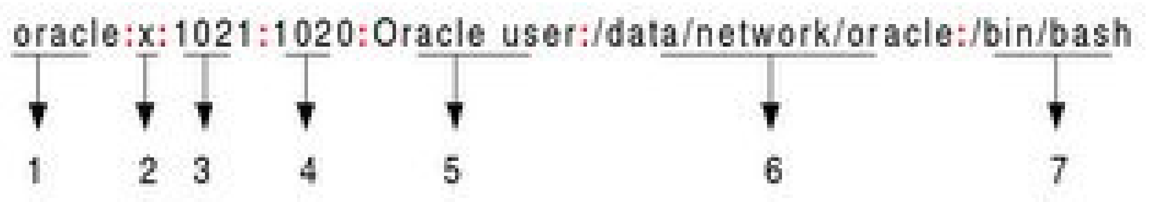
ANS-3 (a)

In Linux, /etc/passwd file stores essential information, which is required during login i.e. user account information. /etc/passwd is a text file, that contains a list of the system's accounts, giving for each account some useful information like user ID, group ID, home directory, shell, etc. It should have general read permission as many utilities, like ls use it to map user IDs to user names, but write access only for the superuser (root).

Understanding fields in /etc/passwd

The /etc/passwd contains one entry per line for each user (or user account) of the system. All fields are separated by a colon (:) symbol. Total seven fields as follows.

Generally, passwd file entry looks as follows (click to enlarge image):


oracle:x:1021:1020:Oracle user:/data/network/oracle:/bin/bash
1
2
3
4
5
6
7

1. **Username**: It is used when user logs in. It should be between 1 and 32 characters in length.
2. **Password**: An x character indicates that encrypted password is stored in /etc/shadow file.
3. **User ID (UID)**: Each user must be assigned a user ID (UID). UID 0 (zero) is reserved for root and UIDs 1-99 are reserved for other predefined accounts. Further UID 100-999 are reserved by system for administrative and system accounts/groups.
4. **Group ID (GID)**: The primary group ID (stored in /etc/group file)
5. **User ID Info**: The comment field. It allow you to add extra information about the users such as user's full name, phone number etc. This field use by finger command.
6. **Home directory**: The absolute path to the directory the user will be in when they log in. If this directory does not exists then users directory becomes /
7. **Command/shell**: The absolute path of a command or shell (/bin/bash). Typically, this is a shell. Please note that it does not have to be a shell.

ANS –3(b)

The SMTP Server

Whenever you send a piece of e-mail, your e-mail client interacts with the SMTP server to handle the sending. The SMTP server on your host may have conversations with other SMTP servers to deliver the e-mail.

Let's assume that I want to send a piece of e-mail. My e-mail ID is brain, and I have my account on howstuffworks.com. I want to send e-mail to jsmith@mindspring.com. I am using a stand-alone e-mail client like Outlook Express.

When I set up my account at howstuffworks, I told Outlook Express the name of the mail server -- mail.howstuffworks.com. When I compose a message and press the Send button, here's what happens:

1. Outlook Express connects to the SMTP server at mail.howstuffworks.com using port 25.
2. Outlook Express has a conversation with the SMTP server, telling the SMTP server the address of the sender and the address of the recipient, as well as the body of the message.
3. The SMTP server takes the "to" address (jsmith@mindspring.com) and breaks it into two parts: the recipient name (jsmith) and the domain name (mindspring.com). If the "to" address had been another user at howstuffworks.com, the SMTP server would simply hand the message to the POP3 server for howstuffworks.com (using a little program called the **delivery agent**). Since the recipient is at another domain, SMTP needs to communicate with that domain.
4. The SMTP server has a conversation with a **Domain Name Server**, or **DNS** (see How Web Servers Work for details). It says, "Can you give me the IP address of the SMTP server for mindspring.com?" The DNS replies with the one or more IP addresses for the SMTP server(s) that Mindspring operates.
5. The SMTP server at howstuffworks.com connects with the SMTP server at Mindspring using port 25. It has the same simple text conversation that my e-mail client had with the SMTP server for HowStuffWorks, and gives the message to the Mindspring server. The Mindspring server recognizes that the domain name for jsmith is at Mindspring, so it hands the message to Mindspring's POP3 server, which puts the message in jsmith's mailbox.

If, for some reason, the SMTP server at HowStuffWorks cannot connect with the SMTP server at Mindspring, then the message goes into a queue. The SMTP server on most machines uses a program called **sendmail** to do the actual sending, so this queue is called the **sendmail queue**. Sendmail will periodically try to resend the messages in its queue. For example, it might retry every 15 minutes. After four hours, it will usually send you a piece of mail that tells you there is some sort of problem. After five days, most sendmail configurations give up and return the mail to you undelivered.

The SMTP server understands very simple text commands like HELO, MAIL, RCPT and DATA. The most common commands are:

- **HELO** - introduce yourself
- **EHLO** - introduce yourself and request extended mode
- **MAIL FROM:** - specify the sender
- **RCPT TO:** - specify the recipient
- **DATA** - specify the body of the message (To, From and Subject should be the first three lines.)
- **RSET** - reset
- **QUIT** - quit the session
- **HELP** - get help on commands
- **VRFY** - verify an address
- **EXPN** - expand an address

- **VERB** – verbose

PROTOCOLS

POP

POP is the older design, and hails from an era when intermittent connection via modem (dial-up) was the norm. POP allows users to retrieve email when connected, and then act on the retrieved messages without needing to stay "on-line." This is an important benefit when connection charges are expensive.

The basic POP procedure is to retrieve all inbound messages for storage on the client, delete them on server, and then disconnect. (The email server functions like a mailbox at the Post Office -- a temporary holding area until mail gets to its final destination, your computer.)

Outbound mail is generated on the client, and held for transmission to the email server until the next time the user's connection is active. After it's uploaded, the server forwards the outgoing mail to other email servers, until it reaches its final destination.

Most POP clients also provide an option to leave copies of email on the server. In this case, messages are only removed from the server when greater than a certain "age" or when they have been explicitly deleted on the client. It's the copies on the client that are considered the "real" ones, however, with those left on the server merely temporary backups.

IMAP

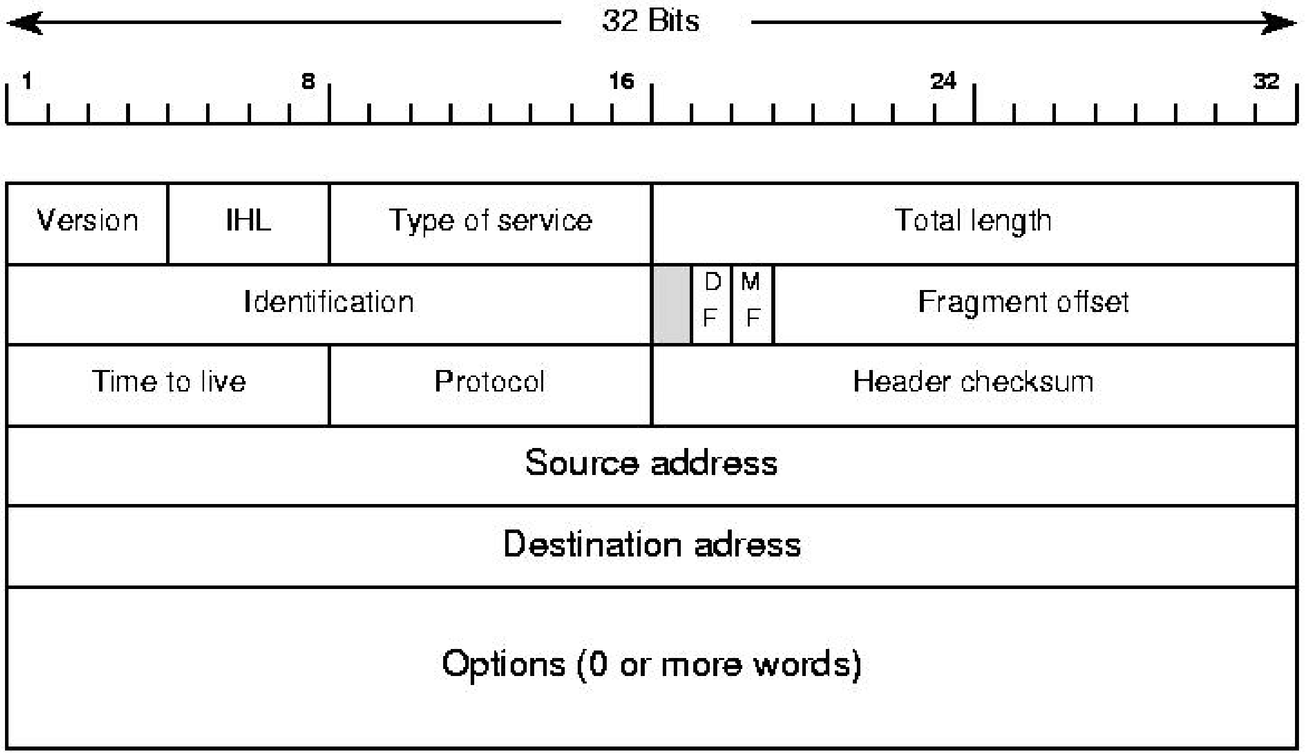
IMAP is the newer protocol and oriented toward a "connected" mode of operation. The standard IMAP procedure is to leave messages on the server instead of retrieving copies, so email is only accessible when "on-line."

IMAP is more suited to a world of always-on connections, particularly the fast connections offered by broadband mechanisms. Having to be connected to read your email is a trivial obstacle when the connection is always available. (It's a little like leaving your messages at the Post Office, and going there every time you want to read them. That might be difficult in the physical world, but it's easy in the virtual one.)

Because messages remain on the server, until explicitly deleted by the user, they can be accessed by multiple client computers -- an important advantage when you use more than one computer to check your email.

IMAP does not preclude keeping copies on the client, but, in an inversion of the way POP works, it's the server's copies that are considered the "real" ones. That offers an important security benefit -- you won't lose your email if, for some reason, your client computer's storage media fails.

ANS -3 (C)



VERSION - Each datagram begins with a 4-bit protocol version number

H.LEN– 4-bit header specifies the number of 32-bit quantities in the header

SERVICE TYPE – 8-bit field that carries a class of service for the datagram

TOTAL LENGTH– 16-bit integer that specifies the total number of bytes in the datagram (including both the header and the data)

IDENTIFICATION– 16-bit number (usually sequential) assigned to the datagram

• used to gather all fragments for reassembly to the datagram

FLAGS– 3-bit field with individual bits specifying whether the datagram is a fragment

• If so, then whether the fragment corresponds to the rightmost piece of the

original datagram

FRAGMENT OFFSET– 13-bit field that specifies where in the original datagram the data in

this fragment belongs

– the value of the field is multiplied by 8 to obtain an offset

TIME TO LIVE – 8-bit integer initialized by the original sender

– it is decremented by each router that processes the datagram

– if the value reaches zero (0)

the datagram is discarded and an error message is sent back to the source

TYPE – 8-bit field that specifies the type of the payload

HEADER CHECKSUM– 16-bit ones-complement checksum of header fields

– it is computed according to Algorithm 8.1

SOURCE IP ADDRESS

– 32-bit Internet address of the original sender

– (the addresses of intermediate routers do not appear in the header)

DESTINATION IP ADDRESS

– The 32-bit Internet address of the ultimate destination

– The addresses of intermediate routers do not appear in the header

OPTIONS

– Optional header fields used to control routing and datagram

processing

– Most datagrams do not contain any options

• which means the IP OPTIONS field is omitted from the header

PADDING

– If options do not end on a 32-bit boundary

• zero bits of padding are added to make the header a multiple of 32 bits

CHECKSUM FIELD IS UNRELIABLE PROTOCOL:

As a consequence of this design, the Internet Protocol only provides best effort delivery and its service is characterized as unreliable. In network architectural language it is a connection-less protocol, in contrast to so-called connection-oriented modes of transmission. The lack of reliability permits various error conditions, such data corruption, packet loss and duplication, as well as out-of-order packet delivery. Since routing is dynamic for every packet and the network maintains no state of the path of prior packets, it is possible that some packets are routed on a longer path to their destination, resulting in improper sequencing at the receiver.

The only assistance that the Internet Protocol provides in Version 4 (IPv4) is to ensure that the IP packet header is error-free through computation of a checksum at the routing nodes. This has the side-effect of discarding packets with bad headers on the spot. In this case no notification is required to be sent to either end node, although a facility exists in the Internet Control Message Protocol (ICMP) to do so.

IPv6, on the other hand, has abandoned the use of IP header checksums for the benefit of rapid forwarding through routing elements in the network.

ANS-3 (d)

Various devices are used to connect network of a computer The most common devices are:

1) Routers

2) Gateways

3) Repeaters

4) Bridges

5) Hub

6) Modem

ROUTER

Routers are devices which connect two are more networks that use similar protocol. A router consists of hard ware and software.

Hard ware can be a computer is specific device.

Software consists of special management program that controls flow of data between networks.

Routers operate at a network layer of O.S.I model.

Routers use logical and physical address to connect two or more logically separate network. They make this connection by organizing the large network into logical network segment (some times small sub network or sub nets). Each of these sub nets is given a logical address. Data is grouped into packets or block of data.

Each packet in addition to having a physical device address, has a logical address. The network address allows routers to calculate more accurately and efficiently the path of the computer.

Advantages of Router

They use high level of intelligence to rout data

Routers can also act as a bridge to handle non rout able protocols such as NetBEUI (Network Bios Extended User Interface)

Disadvantages of Router:

- High level of intelligence take more processing time which can effect performance
- Routers are very complicated which installation and maintenance difficult.

GATEWAYS

Gateways are devices which connect two are more networks that use different protocols. They are similar in function to routes but they are more powerful and intelligent devices. A gateway can actually convert data so that network with an application on a computers on the other side of the gateway e.g a get way can receive email messages in one format in convert them into another format. Gateway can operate at all seven layer of OSI model. Since Gateway perform data conversion so they are slower in speed and very expensive devices.

REPEATERS

Repeaters are used within network to extend the length of communication.

Data process through transmission media in the farm of waves or signals. The transmission media weaken signals that move through it. The weakening of signal is called attenuation. If the data is to be transmitted beyond the maximum length of a communication media, signals have amplified. The devices that are used to amplify the signals are called repeaters. Repeaters work at the physical layer of OSI model.

Repeaters are normally two ports boxes that connect two segments. As a signal comes in one port , it is Regenerated and send out to the other port.

The signal is read as 1s and 0s. As 1s and 0s are transmitted, the noise can be cleaned out.

Advantages of Reapeater

- Repeaters easily extend the length of network.
- They require no processing over head, so very little if any performance degradation occurs.
- It can connect signals from the same network type that use different types of cables.

Disadvantages of Repeaters

- Repeaters can not be used to connect segments of different network types.
- They cannot be used to segment traffic on a network to reduce congestion .
- Many types of network have a limit on the number of network s that can be used at once .

BRIDGES

Bridges are used to connect similar network segments.

A bridge does not pass or signals it receives. When a bridge receive a signal , it determines its destination by looking at its destination and it sends the signals towards it. For example in a above figure a bridge has been used to join two network segments A AND B.

When the bridge receives the signals it read address of both sender and receiver. If the sender is a computer in segment A and the receiver is also segment A, it would not pass the signals to the segments B. It will however pass signals if the sender is in one segment and the receiver in other segment. Bridge works at the data link layer of O.S.I model.

Advantages of Bridges

- Bridge extends network segments by connecting them together to make one logical network.
- They can affect the segment traffic between networks by filtering data if it does not need to pass.
- Like repeaters they can connect similar network types with different cabling.

Disadvantages of Bridges

- Bridge possess information about the data they receive with can slow performance.

HUB

Hubs are basically multi ports repeaters for U.T.P cables. Some hubs have ports for other type of cable such as coaxial cable. Hubs range in size from four ports up to and for specific to the network types. These are some hubs which are

I. Passive Hub

II. Active Hub

III. Switch/ Intelligent Hub

Passive Hub

It provides no signal regeneration. They are simply cables connected together so that the signal is broken out to other nodes with out regeneration. These are not used often today because of loss of cable length that is allowed.

Active Hub

It acts as repeaters and regenerates the data signals to all ports. They have no real intelligence to tell weather the signal needs to go to all ports that is blindly repeated.

Switch Hub

Switches are multi ports bridges. They filter traffic between the ports on the switch by using the address of computers transmitting to them.

Switches can be used when data performance is needed or when collision need to be reduce.

Advantages of Hub

- Hubs need almost no configuration.
- Active hub can extend maximum network media distance.

No processing is done at the hub to slow down performance

Disadvantages of Hub

- Passive hubs can greatly limit maximum media distance.
- Hubs have no intelligence to filter traffic so all data is send out on all ports whether it is need or not.

Since hubs can act as repeaters the network using them must follow the same rules as repeaters

MODEM

The device that converts digital signals into analog signals and analog signals to digital signals is called Modem. The word modem stands for modulation and demodulation. The process of converting digital signals to analog signals is called modulation. The process of converting analog signals to digital signals is called demodulation. Modems are used with computers to transfer data from one computer to another computer through telephone lines.

Modems have two connections these are.

- **Analog connection**
- **Digital connection**

Analog connection.

The connection between the modem and the telephone line is called analog connection.

Types of Modem

THERE ARE TWO TYPES OF MODEMS

- **Internal modem**
- **External modem**

Digital connection.

The connection of modem to computer is called digital connection

INTERNAL MODEM

It fits into expansion slots inside the computer. It is directly linked to the telephone lines through the telephone jack. It is normally less inexpensive than external modem. Its transmission speed is also less external modem.

EXTERNAL MODEM

It is the external unit of computer and is connected to the computer through serial port. It is also linked to the telephone line through a telephone jack. External modems are expensive and have more operation features and high transmission speed.

Advantages of Modem

i. Inexpensive hardware and telephone lines.

ii. Easy to setup and maintain.

Disadvantages of Modem

i. Very slow performance.

ANS-4 (a)

We need to use the `ps` command. It provide information about the currently running processes, including their process identification numbers (PIDs). Both Linux and UNIX support the `ps command` to display information about all running process. The ps command gives a snapshot of the current processes.

ps command

Type the following **ps command** to display all running process:

```
# ps aux | less
```

Where,

- -A: select all processes
- a: select all processes on a terminal, including those of other users
- x: select processes without controlling ttys

Task: see every process on the system

```
# ps -A
```

```
# ps -e
```

Task: See every process except those running as root

```
# ps -U root -u root -N
```

Task: See process run by user vivek

```
# ps -u vivek
```

Task: top command

The top program provides a dynamic real-time view of a running system. Type the top at command prompt:

```
# top
```

Task: display a tree of processes

pstree shows running processes as a tree. The tree is rooted at either pid or init if pid is omitted. If a user name is specified, all process trees rooted at processes owned by that user are shown.

```
$ pstree
```

Task: Print a process tree using ps

```
# ps -ejH
```

```
# ps axjf
```

Task: Get info about threads

Type the following command:

```
# ps -eLf
```

```
# ps axms
```

Task: Get security info

Type the following command:

```
# ps -eo euser,ruser,suser,fuser,f,comm,label
# ps axZ
```

```
# ps -eM
```

ANS-4 (b)

TCP uses an end-to-end flow control protocol to avoid having the sender send data too fast for the TCP receiver to receive and process it reliably. Having a mechanism for flow control is essential in an environment where machines of diverse network speeds communicate. For example, if a PC sends data to a smartphone that is slowly processing received data, the smartphone must regulate the data flow so as not to be overwhelmed.[2]

TCP uses a sliding window flow control protocol. In each TCP segment, the receiver specifies in the *receive window* field the amount of additionally received data (in bytes) that it is willing to buffer for the connection. The sending host can send only up to that amount of data before it must wait for an acknowledgment and window update from the receiving host.

TCP sequence numbers and receive windows behave very much like a clock. The receive window shifts each time the receiver receives and acknowledges a new segment of data. Once it runs out of sequence numbers, the sequence number loops back to 0.

When a receiver advertises a window size of 0, the sender stops sending data and starts the *persist timer*. The persist timer is used to protect TCP from a deadlock situation that could arise if a subsequent window size update from the receiver is lost, and the sender cannot send more data until receiving a new window size update from the receiver. When the persist timer expires, the TCP sender attempts recovery by sending a small packet so that the receiver responds by sending another acknowledgement containing the new window size.

If a receiver is processing incoming data in small increments, it may repeatedly advertise a small receive window. This is referred to as the silly window syndrome, since it is inefficient to send only a few bytes of data in a TCP segment, given the relatively large overhead of the TCP header.

Enhancing TCP to reliably handle loss, minimize errors, manage congestion and go fast in very high-speed environments are ongoing areas of research and standards development. As a result, there are a number of TCP congestion avoidance algorithm variations.

SLIDING WINDOW PROTOCOL

In sliding window protocol, a window is maintained for each connection. The window defines the size of the buffer e.g. the total number of bytes that can be sent by a terminal at a given time shown in figure. This also shows the total number of blocks signifies the total size of window. The sliding window also keeps track of bytes which have been sent but are unacknowledged; bytes still stored in buffer but have not been sent.

Initially a window size is negotiated between the end terminals especially from the receiving side, which establishing a connection. Since TCP provides a byte stream connection, sequence numbers are assigned to each byte in the stream. TCP divides this contiguous byte stream into TCP segments to transmit them. Therefore, the window principle is used at the byte level, that is, the segments sent and acknowledgement received will carry byte sequence numbers and the window size is expressed as a number of bytes.

It has 2 techniques
(i)

Go-Back-N ARQ is a specific instance of the automatic repeat request (ARQ) protocol, in which the sending process continues to send a number of frames specified by a *window size* even without receiving an acknowledgement (ACK) packet from the receiver. It is a special case of the general sliding window protocol with the transmit window size of N and receive window size of 1.

The receiver process keeps track of the sequence number of the next frame it expects to receive, and sends that number with every ACK it sends. The receiver will discard any frame that does not have the exact sequence number it expects (either a duplicate frame it already acknowledged, or an out-of-order frame it expects to receive later) and will resend an ACK for the last correct in-order frame. [1] Once the sender has sent all of the frames in its *window*, it will detect that all of the frames since the first lost frame are *outstanding*, and will go back to sequence number of the last ACK it received from the receiver process and fill its window starting with that frame and continue the process over again.

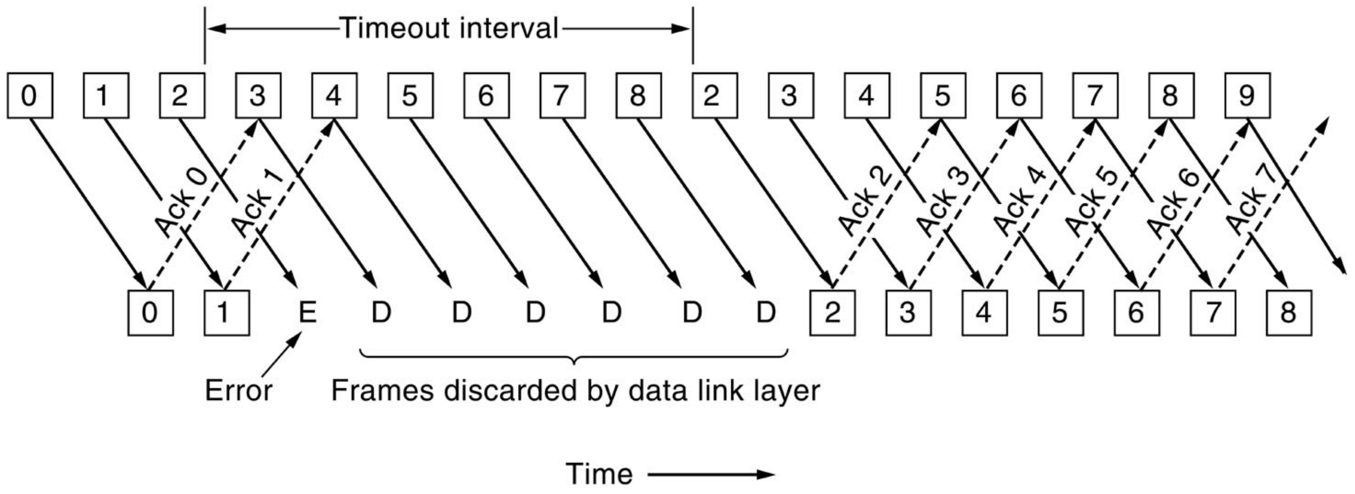
Go-Back-N ARQ is a more efficient use of a connection than Stop-and-wait ARQ, since unlike waiting for an acknowledgement for each packet, the connection is still being utilized as packets are being sent. In other words, during the time that would otherwise be spent waiting, more packets are being sent. However, this method also results in sending frames multiple times – if any frame was lost or damaged, or the ACK acknowledging them was lost or damaged, then that frame and all following frames in the window (even if they were received without error) will be re-sent. To avoid this, Selective Repeat ARQ can be used

(ii)
Selective Repeat is one of the automatic repeat-request (ARQ) techniques. With selective repeat, the sender sends a number of frames specified by a window size even without the need to wait for individual ACK from the receiver as in Go-back N ARQ. However, the receiver sends ACK for each frame individually, which is not like cumulative ACK as used with go-back-n. The receiver accepts out-of-order frames and buffers them. The sender individually retransmits frames that have timed out.

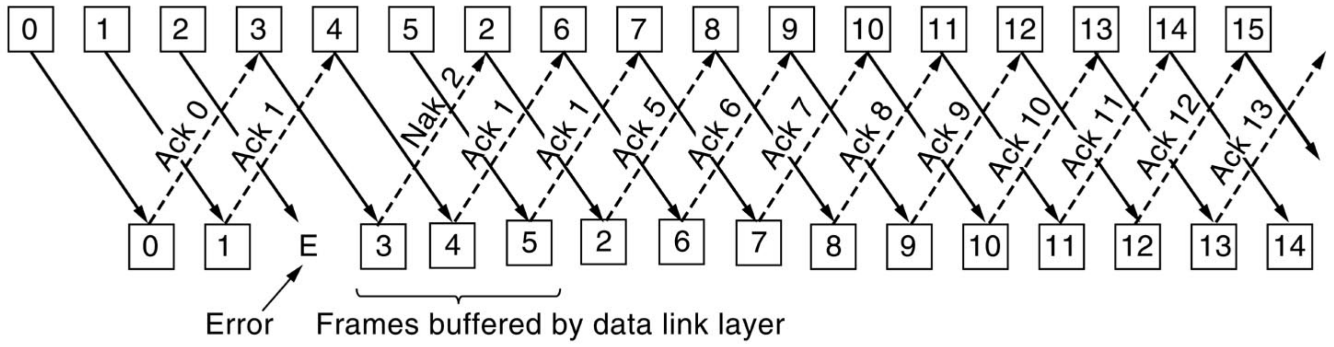
It may be used as a protocol for the delivery and acknowledgement of message units, or it may be used as a protocol for the delivery of subdivided message sub-units.

When used as the protocol for the delivery of **messages**, the sending process continues to send a number of frames specified by a *window size* even after a frame loss. Unlike Go-Back-N ARQ, the receiving process will continue to accept and acknowledge frames sent after an initial error; this is the general case of the sliding window protocol with both transmit and receive window sizes greater than 1.

The receiver process keeps track of the sequence number of the earliest frame it has not received, and sends that number with every acknowledgement (ACK) it sends. If a frame from the sender does not reach the receiver, the sender continues to send subsequent frames until it has emptied its *window*. The receiver continues to fill its receiving window with the subsequent frames, replying each time with an ACK containing the sequence number of the earliest missing frame. Once the sender has sent all the frames in its *window*, it re-sends the frame number given by the ACKs, and then continues where it left off.



(a)



(b)

ANS-4 (c)

Samba-server provides a SMB server which can be used to provide network services to SMB (sometimes called "Lan Manager") clients. Samba uses NetBIOS over TCP/IP (NetBT) protocols and does NOT need NetBEUI (Microsoft Raw NetBIOS frame) protocol. Samba-2.2 features working NT Domain Control capability andincludes the SWAT (Samba Web Administration Tool) that allows samba's smb.conf file to be remotely managed using your favourite web browser. For the time being this is being enabled on TCP port 901 via xinetd. SWAT is now included init's own subpackage, samba-swat. Users are advised to use Samba-2.2 as a Windows NT4 Domain Controller only on networks that do NOT have a WindowsNT Domain Controller. This release does NOT as yet have Backup Domain control ability. Please refer to the WHATSNEW.txt document for fixup information. This binary release includes encrypted password support.

Samba is a suite of programs that gives your Linux box the ability to speak SMB (Server Message Block). SMB is the protocol used to implement file sharing and printer services between computers running OS/2, Windows NT, Windows 95 and Windows for Workgroups. The protocol is analogous to a combination of NFS (Network File System), **lpd** (the standard UNIX printer server) and a distributed authentication framework such as NIS or Kerberos. If you are familiar with Netatalk, Samba does for Windows what Netatalk does for the Macintosh. While running the Samba server programs, your Linux box appears in the “Network Neighborhood” as if it were just another Windows machine. Users of Windows machines can “log into” your Linux server and, depending on the rights they are granted, copy files to and from parts of the UNIX file system, submit print jobs and even send you WinPopup messages. If you use your Linux box in an environment that consists almost completely of Windows NT and Windows 95 machines, Samba is an invaluable tool.

ANS-4(d)

Nmap (*Network Mapper*) is a security scanner originally written by Gordon Lyon used to discover Host and services on a computer network, thus creating a "map" of the network. To accomplish its goal, Nmap sends specially craftedpackets to the target host and then analyzes the responses.

The software provides features for probing computer networks such as host discovery, service and operating system detection, and other in-depth system information. These features are extensible by scripts that provide more advanced service detection vulnerability detection, and other information. Nmap is also capable of adapting to network conditions including latency and network congestion during a scan. Nmap is under development and refinement by its user community.

Nmap features include:

- Host discovery - Identifying hosts on a network. For example, listing the hosts that respond to pings or have a particular port open.
- Port scanning - Enumerating the open ports on target hosts.
- Version detection - Interrogating network services on remote devices to determine application name and version number.
- OS detection - Determining the operating system and hardware characteristics of network devices.
- Scriptable interaction with the target - using Nmap Scripting Engine (NSE) and Lua programming language.

Nmap can provide further information on targets, including reverse DNS names, device types, and MAC addresses.

Basic commands working in Nmap[edit source | editbeta]

- For target specifications:

```
nmap <targets' URL's or IP's>
```

- For OS detection:

```
nmap -O <target-host's URL or IP>
```

- For Version detection:

```
nmap -sV <target-host's URL or IP>
```

- For configuring response timings

```
nmap -T0 -sV -O <target-host's URL or IP>
```