Ch1

What is a Socket?

A socket is an abstract representation of a communication endpoint through which an application may send and receive data.

•Programming:

ex1:

illcode.c

```c
#include <stdio.h>
int main(void)
{
int var = 1;
printf("It is not standard C code!\n");
return 0;
}
```

ex2:

ex2.c

```c
#include <stdio.h>
#include <stdlib.h>
typedef struct stuff{
    int val;
    float b;
} Stuff;

int main(void)
{
 Stuff a;
 printf("Plz enter an integer\n");
 scanf("%d", &(a.val));
 printf("enter  a float number\n");
 scanf("%f", &(a.b));
 printf("interger = %d\n", a.val);
 printf("float = %1f\n", a.b);
return 0;
}
```

ex3:

ex3.c

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct {
            int a;
            short s[2];
} MSG;

int main(){
    MSG *mp, m = {4, 1, 0};  //a =4 s= 4 -> fp
    char *fp, *tp;// fp指向m中的s[0], tp指向mp中的s[0]
    mp = (MSG *)malloc(sizeof(MSG));          // tp在mp s[0]-s[1]
```

```
    for (fp = (char *)m.s, tp = (char *)mp->s; tp < (char *)
(mp+1);)
            *tp++ = *fp++;//*(tp++) = *(fp++);   //fp=1 -> tp
printf("\nprint results:\n");
printf("\nm={%d, %d, %d}\n", m.a, m.s[0], m.s[1]);
printf("\nmp={%d, %d, %d}\n", mp->a, mp->s[0], mp->s[1]);

    return 0;
}
```

ex4:

ex4.h

```
#include <stdio.h>
#include <stdlib.h>

typedef struct {
            int a;
            short s[2];
} MSG;
```

ex4.c

```
#include <stdio.h>
#include <stdlib.h>
#include "ex4.h"

int main(){
    MSG *mp, m = {4, 1, 0};  //a =4 s= 4 -> fp
    char *fp, *tp;// fp指向m中的s[0], tp指向mp中的s[0]
    mp = (MSG *)malloc(sizeof(MSG));          // tp在mp s[0]-s[1]
    for (fp = (char *)m.s, tp = (char *)mp->s; tp < (char *)
(mp+1);)
            *tp++ = *fp++;//*(tp++) = *(fp++);   //fp=1 -> tp
```

```
printf("\nprint results:\n");
printf("\nm={%d, %d, %d}\n", m.a, m.s[0], m.s[1]);
printf("\nmp={%d, %d, %d}\n", mp->a, mp->s[0], mp->s[1]);

    return 0;
}
```

Ch1.1

—What is TCP/IP?

Transmission Control Protocol / Internet Protocol is a suite of communication protocols used to interconnect network devices on the internet.

—What is TCP?

TCP is an alternative transport layer protocol supported by TCP/IP.
TCP provides a connection-oriented, reliable, byte stream service (lots of overhead).

—What is UDP?

UDP is a transport protocol. UDP uses IP to deliver datagrams to the right host, uses *ports* to provide communication services to individual processes.

UDP offers minimal datagram delivery service (as little overhead as possible).

—What is Telnet?

Telnet is a protocol used on the Internet or local area network to provide a bidirectional interactive text-oriented communication facility using a virtual terminal connection. User data is interspersed in-band with Telnet control information in an 8-bit byte oriented data connection over the Transmission Control Protocol (TCP).

—What is SSH?

Secure Shell (SSH) is a cryptographic network protocol for operating network services securely over an unsecured network.

Ch1.3

What is a Socket?

A socket is an abstract representation of a communication endpoint through which an application may send and receive data.

Programming :

ex1

```
#include <arpa/inet.h>
#include <netinet/in.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <unistd.h>
#define PORT 8080

int main(void)
{
    int server_fd, valread;
    struct sockaddr_in address;
    int addrlen = sizeof(address);
```

```
    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) < 0) // create
    {
        perror("socket failed");
        exit(EXIT_FAILURE);
    }
   memset(&address, 0, sizeof(address));     /* Zero out structure */

    if (bind(server_fd, (struct sockaddr *)&address, sizeof(address)) <
0)
    {
        perror("bind failed");
        exit(EXIT_FAILURE);
    }
    close(server_fd);
    return 0;
}
```

ex2

```
#include <stdio.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>

int main(int argc, char *argv[])
{
    int sock;
    struct sockaddr_in echoServAddr,peerAddr;
    unsigned short echoServPort;
    char *servIP;
    if ((argc < 2) || (argc > 3))
    {
       fprintf(stderr, "Usage: %s <Server IP> [<Echo Port>]\n",
               argv[0]);
       exit(1);
    }
```

```
    servIP = argv[1];

    if (argc == 3)
        echoServPort = atoi(argv[2]);
    else
        echoServPort = 7;
    if ((sock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0){
        perror("socket failed");
        exit(EXIT_FAILURE);
    }
    memset(&echoServAddr, 0, sizeof(echoServAddr));
    echoServAddr.sin_family      = AF_INET;
    echoServAddr.sin_addr.s_addr = inet_addr(servIP);
    echoServAddr.sin_port        = htons(echoServPort);

    if (bind(sock, (struct sockaddr *)&echoServAddr,
sizeof(echoServAddr)) < 0)
    {
        perror("bind failed");
        exit(EXIT_FAILURE);
    }
    int echoServLen = sizeof(echoServAddr);
    int peerLen = sizeof(peerAddr);
    getsockname(sock, (struct sockaddr *)&echoServAddr, (socklen_t
*)&echoServLen);
    getpeername(sock, (struct sockaddr *)&peerAddr, (socklen_t
*)&peerLen);
    printf("connected server address = %s:%d\n",
inet_ntoa(echoServAddr.sin_addr),
           ntohs(echoServAddr.sin_port));
    printf("connected peer address = %s:%d\n",
           inet_ntoa(peerAddr.sin_addr),
           ntohs(peerAddr.sin_port));
    printf("\n");    /* Print a final linefeed */

    close(sock);
    exit(0);
}
```

ex3:

```c
#include <netdb.h>
#include <stdio.h>      /* for printf() and fprintf() */
#include <sys/socket.h> /* for socket(), connect(), send(), and
recv() */
#include <arpa/inet.h>  /* for sockaddr_in and inet_addr() */
#include <stdlib.h>     /* for atoi() and exit() */
#include <string.h>     /* for memset() */
#include <unistd.h>     /* for close() */

#define Hostname "10.4.40.89"
int main(){
    struct hostent *he;
    struct in_addr **addr_list;
    he = gethostbyname(Hostname);
    printf("All addresses: ");
    addr_list = (struct in_addr **)he->h_addr_list;
    for(int i = 0; addr_list[i] != NULL; i++) {
        printf("%s \n", inet_ntoa(*addr_list[i]));
    }
    return 0;
}
```

ex4:

```c
# include <unistd.h>
# include <sys/types.h>
#include <stdlib.h>
#include <unistd.h>
int main ()
{
pid_t pid;
pid=fork();
if (pid < 0)
    printf("error in fork!\n");
else if (pid == 0)
```

```
    printf("i am the child process, my process id is
%d\n",getpid());
else
    printf("i am the parent process, my process id is
%d\n",getpid());
return 1;
}
```

output:
i am the parent process, my process id is 69678
i am the child process, my process id is 69679

reason:
fork create a new child process, after finish running parent process, it will return to child process.

Ch2

Programming

ex1

```
#include <stdio.h>      /* for printf() and fprintf() */
```

```c
#include <sys/socket.h> /* for socket(), connect(), send(), and
recv() */
#include <arpa/inet.h>  /* for sockaddr_in and inet_addr() */
#include <stdlib.h>     /* for atoi() and exit() */
#include <string.h>     /* for memset() */
#include <unistd.h>     /* for close() */

#define RCVBUFSIZE 32   /* Size of receive buffer */

void DieWithError(char *errorMessage);  /* Error handling function
*/

int main(int argc, char *argv[])
{
    int sock;                        /* Socket descriptor */
    struct sockaddr_in echoServAddr,peerAddr; /* Echo server
address */
    unsigned short echoServPort;     /* Echo server port */
    char *servIP;                    /* Server IP address (dotted
quad) */
    char *echoString;                /* String to send to echo
server */
    char echoBuffer[RCVBUFSIZE];     /* Buffer for echo string */
    unsigned int echoStringLen;      /* Length of string to echo
*/
    int bytesRcvd, totalBytesRcvd;   /* Bytes read in single
recv()
                                        and total bytes read */

    if ((argc < 3) || (argc > 4))    /* Test for correct number of
arguments */
    {
       fprintf(stderr, "Usage: %s <Server IP> <Echo Word> [<Echo
Port>]\n",
               argv[0]);
       exit(1);
    }
```

```
    servIP = argv[1];             /* First arg: server IP address
(dotted quad) */
    echoString = argv[2];         /* Second arg: string to echo */

    if (argc == 4)
        echoServPort = atoi(argv[3]); /* Use given port, if any */
    else
        echoServPort = 7;  /* 7 is the well-known port for the
echo service */

    /* Create a reliable, stream socket using TCP */
    if ((sock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0)
        DieWithError("socket() failed");

    /* Construct the server address structure */
    memset(&echoServAddr, 0, sizeof(echoServAddr));     /* Zero
out structure */
    echoServAddr.sin_family      = AF_INET;             /*
Internet address family */
    echoServAddr.sin_addr.s_addr = inet_addr(servIP);   /* Server
IP address */
    echoServAddr.sin_port        = htons(echoServPort); /* Server
port */

    /* Establish the connection to the echo server */
    if (connect(sock, (struct sockaddr *) &echoServAddr,
sizeof(echoServAddr)) < 0)
        DieWithError("connect() failed");

    echoStringLen = strlen(echoString);          /* Determine
input length */

    /* Send the string to the server */
    if (send(sock, echoString, echoStringLen, 0) != echoStringLen)
        DieWithError("send() sent a different number of bytes than
expected");

    /* Receive the same string back from the server */
    totalBytesRcvd = 0;
```

```
    printf("Received: ");                /* Setup to print the
echoed string */
    while (totalBytesRcvd < echoStringLen)
    {
        /* Receive up to the buffer size (minus 1 to leave space
for
           a null terminator) bytes from the sender */
        if ((bytesRcvd = recv(sock, echoBuffer, RCVBUFSIZE - 1,
0)) <= 0)
            DieWithError("recv() failed or connection closed
prematurely");
        totalBytesRcvd += bytesRcvd;   /* Keep tally of total
bytes */
        echoBuffer[bytesRcvd] = '\0';  /* Terminate the string! */
        printf("%s", echoBuffer);      /* Print the echo buffer */
    }
    int echoServLen = sizeof(echoServAddr);
    int peerLen = sizeof(peerAddr);

    getsockname(sock, (struct sockaddr *)&echoServAddr, (socklen_t
*)&echoServLen);
    getpeername(sock, (struct sockaddr *)&peerAddr, (socklen_t
*)&peerLen);

    printf("connected server address = %s:%d\n",
inet_ntoa(echoServAddr.sin_addr),
           ntohs(echoServAddr.sin_port));
    printf("connected peer address = %s:%d\n",
           inet_ntop(AF_INET, &peerAddr.sin_addr, echoBuffer,
sizeof(echoBuffer)),
           ntohs(peerAddr.sin_port));
    printf("\n");    /* Print a final linefeed */

    close(sock);
    exit(0);
}
```

ex2

output:

connect() failed: Connection refused

ex3

```c
#include <stdio.h>      /* for printf() and fprintf() */
#include <sys/socket.h> /* for socket(), bind(), and connect() */
#include <arpa/inet.h>  /* for sockaddr_in and inet_ntoa() */
#include <stdlib.h>     /* for atoi() and exit() */
#include <string.h>     /* for memset() */
#include <unistd.h>     /* for close() */

#define MAXPENDING 5    /* Maximum outstanding connection requests
*/

void DieWithError(char *errorMessage);  /* Error handling function
*/
void HandleTCPClient(int clntSocket);   /* TCP client handling
function */

int main(int argc, char *argv[])
{
    int servSock;                   /* Socket descriptor for
server */
    int clntSock;                   /* Socket descriptor for
client */
    struct sockaddr_in echoServAddr; /* Local address */
    struct sockaddr_in echoClntAddr; /* Client address */
    unsigned short echoServPort;     /* Server port */
    unsigned int clntLen,servLen;            /* Length of client
address data structure */

    if (argc != 2)     /* Test for correct number of arguments */
    {
        fprintf(stderr, "Usage:  %s <Server Port>\n", argv[0]);
        exit(1);
    }
```

```
    echoServPort = atoi(argv[1]);  /* First arg:  local port */

    /* Create socket for incoming connections */
    if ((servSock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)) <
0)
        DieWithError("socket() failed");

    /* Construct local address structure */
    memset(&echoServAddr, 0, sizeof(echoServAddr));   /* Zero out
structure */
    echoServAddr.sin_family = AF_INET;                /* Internet
address family */
    echoServAddr.sin_addr.s_addr = htonl(INADDR_ANY); /* Any
incoming interface */
    echoServAddr.sin_port = htons(echoServPort);      /* Local
port */


    getsockname(servSock, (struct sockaddr *)&echoServAddr,
(socklen_t *)&servLen);
    getpeername(clntSock, (struct sockaddr *)&echoClntAddr,
(socklen_t *)&clntLen);
    printf("connected server address = %s:%d\n",
inet_ntoa(echoServAddr.sin_addr),
    ntohs(echoServAddr.sin_port));
    printf("connected peer address = %s:%d\n",
    inet_ntoa(echoClntAddr.sin_addr),
    ntohs(echoClntAddr.sin_port));

    /* Bind to the local address */
    if (bind(servSock, (struct sockaddr *) &echoServAddr,
sizeof(echoServAddr)) < 0)
        DieWithError("bind() failed");

    /* Mark the socket so it will listen for incoming connections
*/
    if (listen(servSock, MAXPENDING) < 0)
        DieWithError("listen() failed");

    for (;;) /* Run forever */
```

```
    {
        /* Set the size of the in-out parameter */
        clntLen = sizeof(echoClntAddr);
        servLen = sizeof(echoServAddr);
        /* Wait for a client to connect */
        if ((clntSock = accept(servSock, (struct sockaddr *)
&echoClntAddr,
                               &clntLen)) < 0)
            DieWithError("accept() failed");

        /* clntSock is connected to a client! */


        getsockname(servSock, (struct sockaddr *)&echoServAddr,
(socklen_t *)&servLen);
        getpeername(clntSock, (struct sockaddr *)&echoClntAddr,
(socklen_t *)&clntLen);
        printf("Handling client %s\n",
inet_ntoa(echoClntAddr.sin_addr));
        printf("connected server address = %s:%d\n",
inet_ntoa(echoServAddr.sin_addr),
           ntohs(echoServAddr.sin_port));
        printf("connected peer address = %s:%d\n",
           inet_ntoa(echoClntAddr.sin_addr),
           ntohs(echoClntAddr.sin_port));
        HandleTCPClient(clntSock);
    }
    /* NOT REACHED */
}
```

Ex4
output:
bind failed: Invalid argument

Ch4

Programming

ex1:

```c
#include <stdio.h>
#include <sys/time.h>
#include <sys/types.h>
#include <unistd.h>
#include <string.h>
#include <stdlib.h>

#define STDIN 0  // file descriptor for standard input

int main(void) {
    fd_set rfds;
    struct timeval tv;
    int retval;
    char buf[255];
    int len;
    /* Watch stdin (fd 0) to see when it has input. */
    FD_ZERO(&rfds);
//we must turn on the interested bits (by calling FD_SET) each
time we call select.
    FD_SET(0, &rfds);

    /* Wait up to five seconds. */
    tv.tv_sec = 5;
    tv.tv_usec = 0;

while(1){
        retval = select(STDIN+1, &rfds, NULL, NULL, &tv);
    /* Don't rely on the value of tv now! */

    if (retval == -1)
        perror("select()");
    else if (retval){
        fgets(buf, sizeof(buf), stdin);
        len = strlen(buf) - 1;
```

```
        if (buf[len] == '\n'){
            buf[len] = '\0';
        }
        if(strcmp(buf, "quit") == 0){
            printf("exit\n");
            exit(EXIT_FAILURE);
        }else{
            printf("'%s' was read from stdin.\n", buf);
        }
    }else{
        printf("No data within five seconds.\n");
        exit(EXIT_FAILURE);
    }
}

    return 0;
}
```

Ex2:

For those TCP clients that don't pick their own local address/port
with bind(), the local Internet address and port are determined by
underlying implementation during the call to connect().

Ex3:

bind failed: Can't assign requested address

Ex4:
using send() and recv() function