Introduction to Network Programming

Lecture 1

What is a Socket?

- A *socket* is one endpoint of a two-way communication link between two programs running on the network.
- A socket is bound to a **port number** so that the TCP layer can identify the application that data is destined to be sent.

Socket address structure

- Socket functions require a **pointer to a socket address structure** as an argument
- Names of the structure begins with ***sockaddr_*** with a unique suffix for each protocol suite

IPv4 Socket address structure – 1/2

- Commonly known as *“Internet socket address structure”*
- Named as ***sockaddr_in***
- Defined by including ***<netinet/in.h>*** header

IPv4 Socket address structure – 2/2


```
struct in_addr
{
    in_addr_t s_addr; /* 32-bit IP address */
};
```

```
struct sockaddr_in
{
    uint8_t sin_len; //length of structure
    sa_family_t sin_family; // AF_INET
    in_port_t sin_port; //TCP or UDP port
    struct in_addr sin_addr; //IP address
    char sin_zero[8]; //unused
};
```

Refer to the data types in <types.h> file

IPv6 Socket address structure

- Defined in **<netinet/in.h>**

```
struct in6_addr
{
    uint8_t s6_addr[16]; /* 128-bit IP
    address */
};
```

```
struct sockaddr_in6
{
    uint8_t sin6_len; //length of structure
    sa_family_t sin6_family; //AF_INET6
    in_port_t sin6_port; //TCP or UDP port
    struct in6_addr sin6_addr; //IP address
    uint32_t sin6_flowinfo; //priority & flow
    label
};
```

Refer to the data types in <types.h> file

Value-Result arguments

- Socket address is passed to a socket function by reference
- Length of structure is also passed as an argument
- Way in which length is passed depends on which direction the structure is being passed
  - Process to kernel
  - Kernel to process

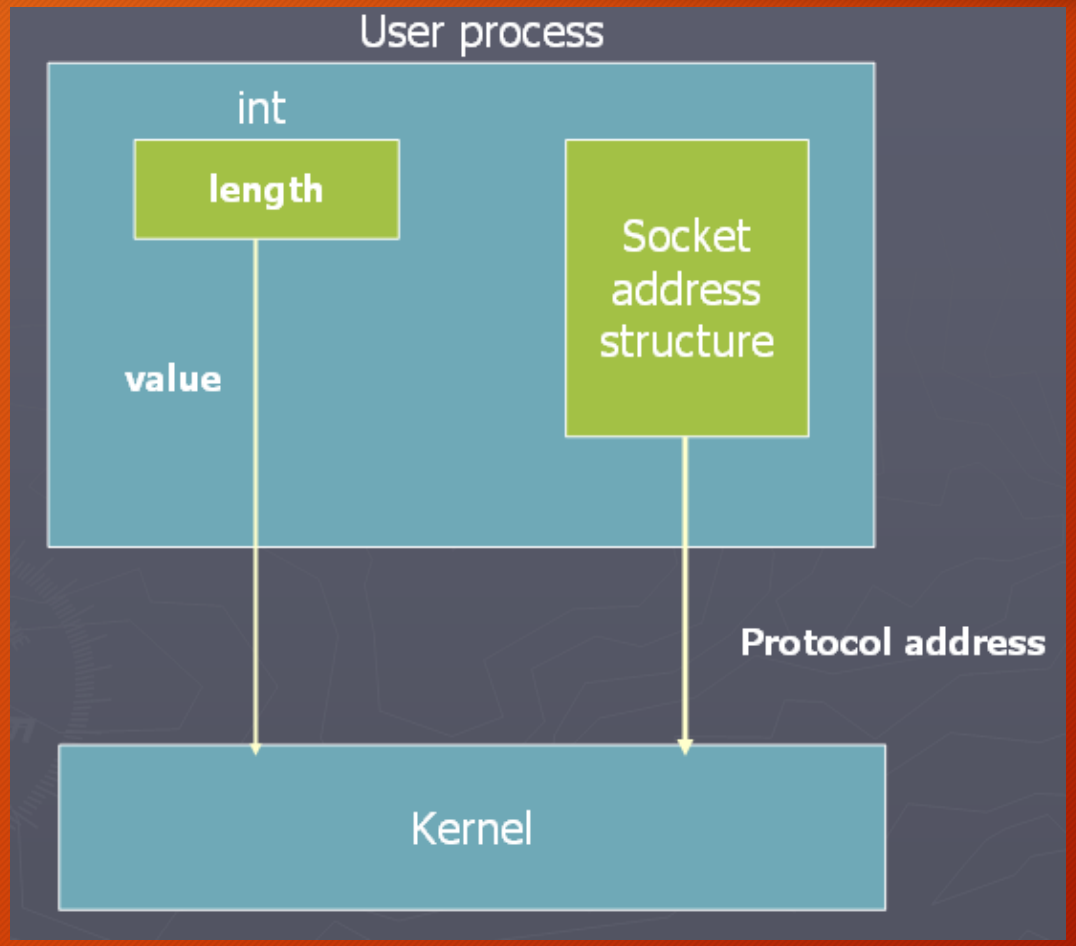
Process to Kernel – 1/2

- **Value-only**: *bind*, *connect*, *sendto*
- Example:

```
struct sockaddr_in serv;     //fill in serv{}
connect (sockfd, (struct sockaddr *) &serv, sizeof(serv));
```

- One argument to these three functions is the pointer to socket address structure and another is the integer size of structure
- Since kernel is passed, both pointer and size of what pointer points to, knows exactly how much data to copy from process into kernel

Process to Kernel – 2/2



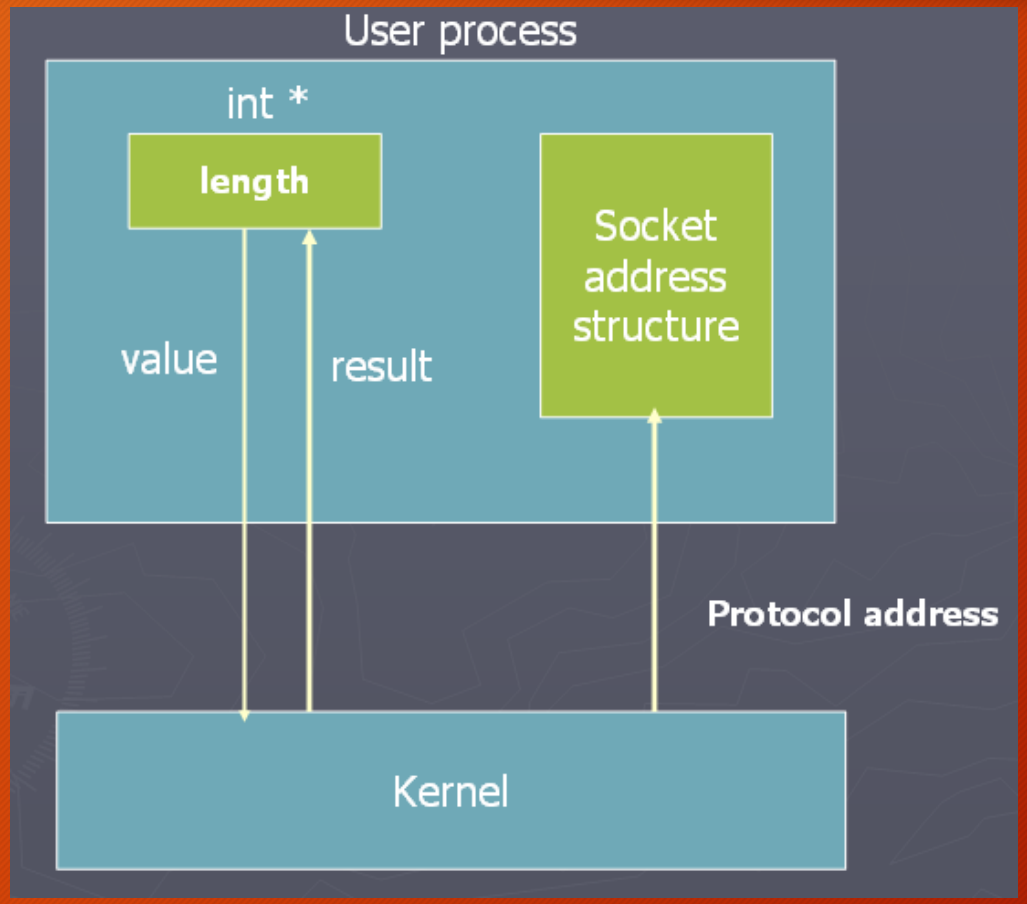
Kernel to Process – 1/2

- ***Value-Result***: *accept, recvfrom, getsockname, getpeername*

- Example:

```
struct  sockaddr_in  cli ;
socklen_t len;
len = sizeof (cli);
getpeername (unixfd, (struct sockaddr*) &cli, & len);
//len may have changed
```

Kernel to Process – 2/2



Byte ordering functions – 1/2

- 16-bit integer made up of 2 bytes
- Two ways to store bytes in memory
  - Little-endian :- low-order byte at starting address
  - Big-endian :- high-order byte at starting address
- Refer to the byte ordering used by a system as *host byte order*

Byte ordering functions – 2/2

- Network protocol must specify network byte order
- Uses big-endian byte ordering
- Functions to convert between two byte orders
  - uint16_t htons (uint16_t host16bitvalue);
  - uint32_t htonl (uint32_t host32bitvalue);
    - Both return value in network byte order
  - uint16_t ntohs (uint16_t net16bitvalue);
  - Uint32_t ntohl (uint32_t net32bitvalue);
    - Both return value in host byte order

h – host, n – network, s – short, l – long

Program - Determine the host byte order


```
include “unp.h”
int main(int argc, char **argv) {
        union {
                short s;
                char c[sizeof(short)];
        } un;
        un.s = 0x0102;          //the hexadecimal number
        if (sizeof(short)==2)       {
                if (un.c[0]==1 && un.c[1]==2)
                        printf (“\n Big Endian”);
                else if (un.c[0]==2 && un.c[1]==1)
                        printf (“\n Little Endian”);
                else
                        printf (“\n Unknown order”);
        }
        else
                printf (“\n Size of (short) = %d”, sizeof(short));
                exit(0);        //success
}
```

Byte ordering functions

- Header file is ***<string.h>***
  - bzero ()
  - bcopy ()
  - bcmp ()
  - memcpy ()
  - memset ()
  - memcmp ()

bzero()

- Set specified number of bytes to 0 in the destination

```
void bzero (void *dest, size_t nbytes);
```

- Use to initialise a socket address structure to 0

bcopy()

- Moves specified number of bytes from source to destination

void bcopy (const void *src, void *dest, size_t nbytes);

bcmp()

- Compares two arbitrary byte strings

int bcmp (const void *ptr1, const void *ptr2, size_t nbytes);

- Return value :
  - 0 if two byte strings are identical
  - Nonzero if not identical

memset()

- Set specified number of bytes to value c in destination

```
void *memset (void *dest, int c, size_t len);
```

memcpy()

- Same as bcopy() but two pointer arguments are swapped

```
void *memcpy (void *dest, const void *src, size_t nbytes);
```

memcmp()

- Compares two arbitrary strings

int memcmp (const void *ptr1, const void *ptr2, size_t nbytes);

- Return value:
  - 0 if identical
  - >0 if ptr1 > ptr2
  - <0 if ptr1 < ptr2

Example Program - bzero()


```
#include "unp.h"
int main(int argc, char *argv[])
{
        char s[10];
        int i;
        if (argc!=2)
        {
                printf("Usage :<command> <string>\n");
                exit(0);
        }
```

Example Program - bzero()


```
        strcpy(s,argv[1]);
        printf("Original string : %s\n", s);
        bzero(s,1);
        printf("After bcopy() : %s\n\n",s);
        exit(1); //exit(1);
}
```

Example Program - bcopy()


```
#include "unp.h"
int main(int argc, char *argv[])
{
        const int SOURCE_BUFF_SIZE = 10;
        const int DEST_BUFF_SIZE = 3;
        char s[10]={'a','b','c','d','e'};
        char d[3]={'x','y','z'};

        int i = DEST_BUFF_SIZE*sizeof(char);
        printf("Before copy\n");
        printf("Source = %s\n", s);
        printf("Destination = %s\n\n",d);
```

Example Program - bcopy()


```
        bcopy(s,d,DEST_BUFF_SIZE*sizeof(char));
        //unit no of integers copied

        printf("After copy\n");
        printf("Source = %s\n", s);
        printf("Destination = %s\n\n", d);

        exit(1); //success
}
```

Example Program - memset()


```
#include "unp.h"
int main(int argc, char *argv[]){
        const int SOURCE_BUFF_SIZE = 10;
        const int DEST_BUFF_SIZE = 3;
        char s[10]={'a','b','c','d','e'};
        int i = DEST_BUFF_SIZE*sizeof(char);
        printf("Before setting\n");
        printf("String = %s\n", s);
        memset(s, '*', i);
        printf("After setting\n");
        printf("String = %s\n", s);
}
```