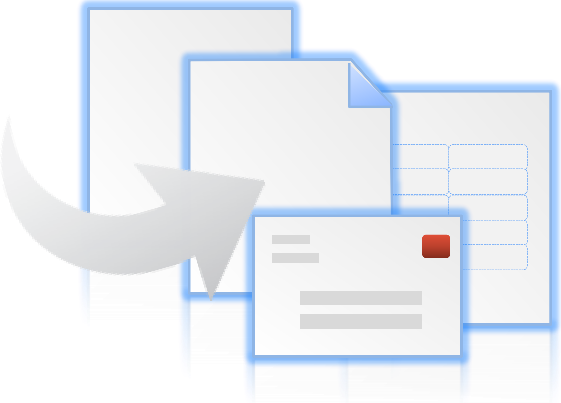
NETWORK PROGRAMMING

SOCKETS INTRODUCTION

BY

MR.PRASAD SAWANT



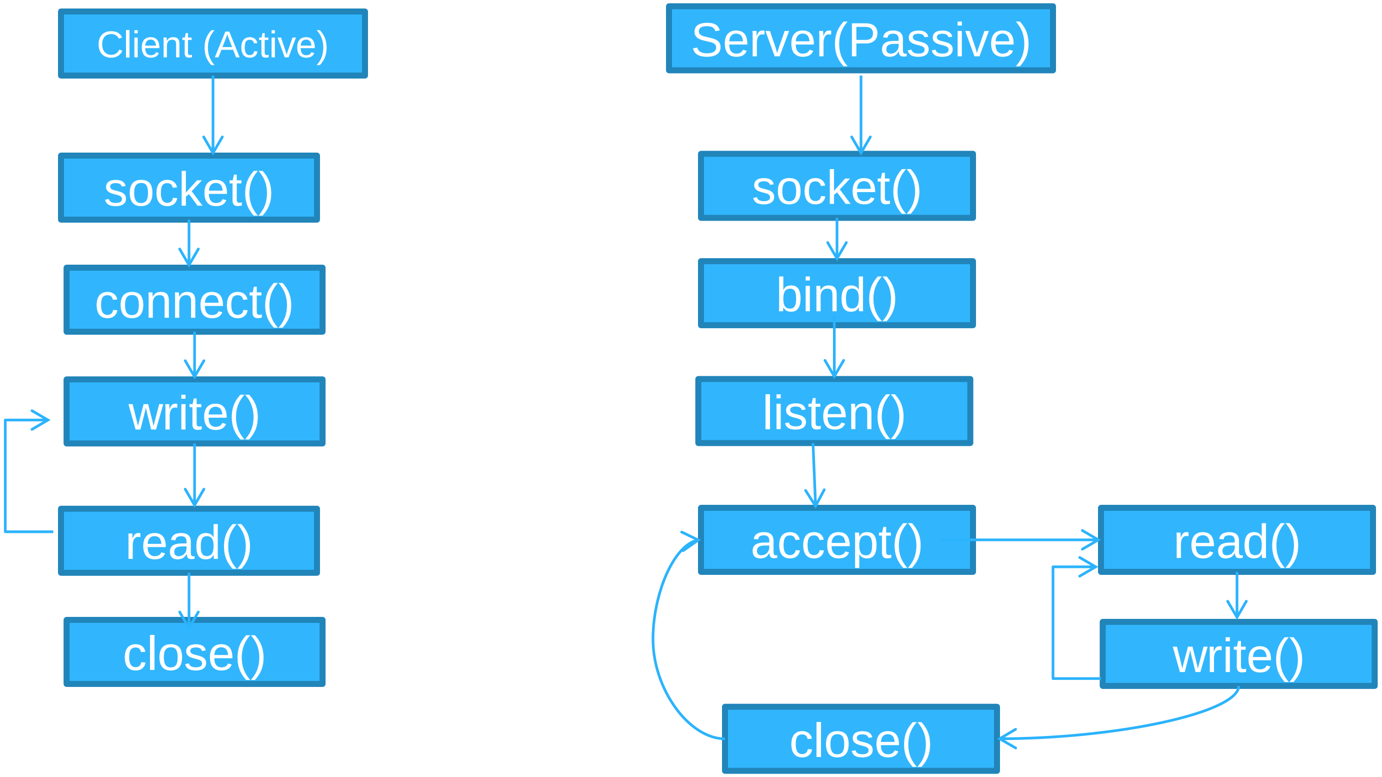
WHAT IS A SOCKET ?

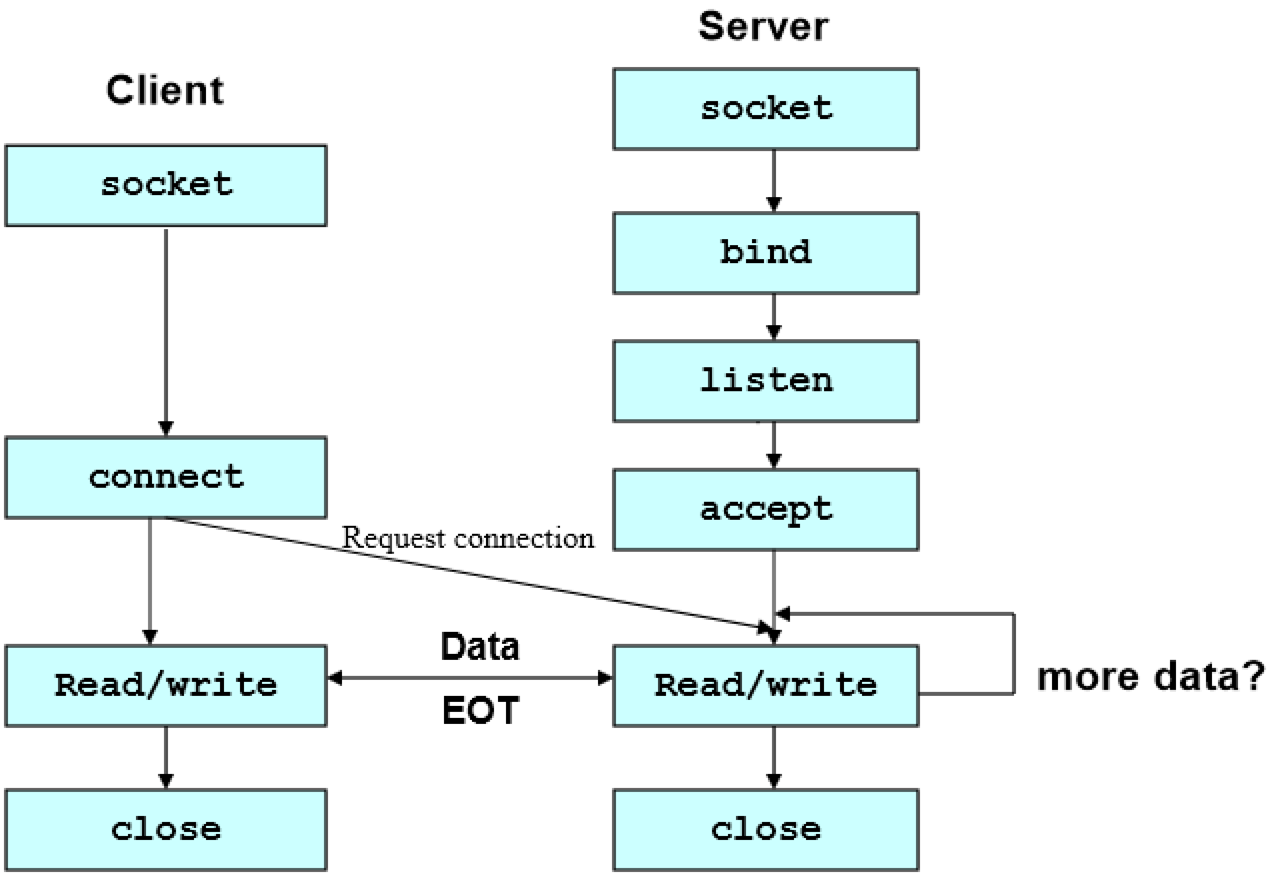
A *socket* is an abstraction for network communication, just as a file is an abstraction for file system communication

Primary Socket Functions

OPERATION	**EXPLANATION**
Open	Prepare for input or output operations.
Close	Stop previous operations and return resources.
Read	Get data and place in application memory.
Write	Put data from application memory and send control.
Control (ioctl)	Set options such as buffer sizes and connection behavior.

USING SOCKETS




Server
Client
socket
socket
bind
listen
connect
accept
Request connection
Data
Read/write
Read/write
more data?
EOT
close
close

BERKELEY SOCKETS

Berkeley sockets (or **BSD sockets**) is a computing library with an application programming interface (API) for internet sockets and Unix domain sockets, used for inter-process communication (IPC).

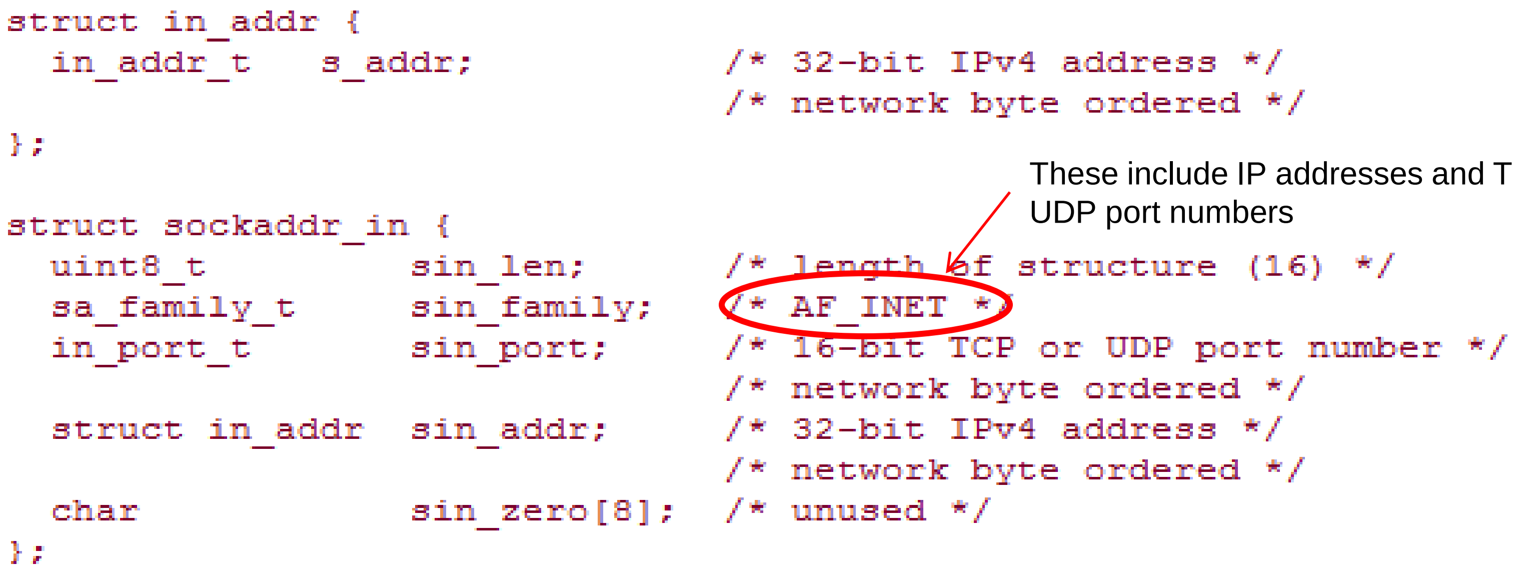
BSD VS POSIX

POSIX "**P**ortable **O**perating **S**ystem **I**nterface",

Action	BSD	POSIX
Conversion from text address to packed address	inet_aton	inet_pton
Conversion from packed address to text address	inet_ntoa	inet_ntop
Forward lookup for host name/service	gethostbyname, gethostbyaddr, getservbyname, getservbyport	getaddrinfo
Reverse lookup for host name/service	gethostbyaddr, getservbyport	getnameinfo

SOCKET ADDRESS STRUCTURE

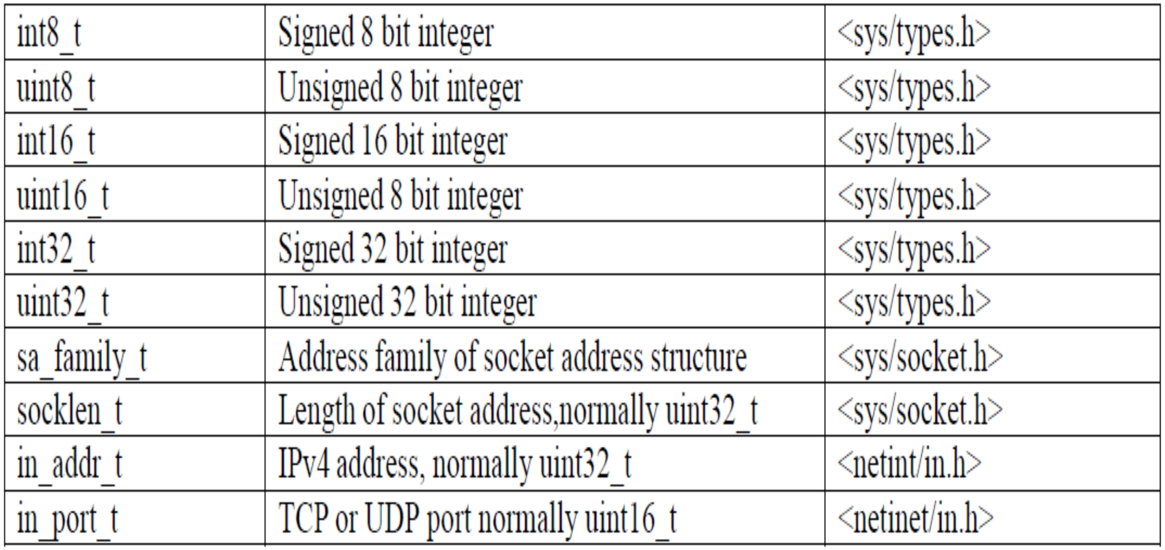
An IPv4 socket address structure, commonly called an "Internet socket address structure," is named sockaddr_in and is defined by including the <netinet/in.h>



```
struct in_addr {
  in_addr_t   s_addr;           /* 32-bit IPv4 address */
                                /* network byte ordered */
};

struct sockaddr_in {
  uint8_t         sin_len;      /* length of structure (16) */
  sa_family_t     sin_family;   /* AF_INET */
  in_port_t       sin_port;     /* 16-bit TCP or UDP port number */
                                /* network byte ordered */
  struct in_addr  sin_addr;     /* 32-bit IPv4 address */
                                /* network byte ordered */
  char            sin_zero[8];  /* unused */
};
```

These include IP addresses and TCP and UDP port numbers

VARIOUS DATA TYPES THAT ARE COMMONLY USED ARE LISTED BELOW:



int8_t	Signed 8 bit integer	<sys/types.h>
uint8_t	Unsigned 8 bit integer	<sys/types.h>
int16_t	Signed 16 bit integer	<sys/types.h>
uint16_t	Unsigned 8 bit integer	<sys/types.h>
int32_t	Signed 32 bit integer	<sys/types.h>
uint32_t	Unsigned 32 bit integer	<sys/types.h>
sa_family_t	Address family of socket address structure	<sys/socket.h>
socklen_t	Length of socket address,normally uint32_t	<sys/socket.h>
in_addr_t	IPv4 address, normally uint32_t	<netint/in.h>
in_port_t	TCP or UDP port normally uint16_t	<netinet/in.h>

SOCKET API FUNCTIONS

bind() is typically used on the server side, and associates a socket with a socket address structure, i.e. a specified local port number and IP address.

connect() is used on the client side, and assigns a free local port number to a socket. In case of a TCP socket, it causes an attempt to establish a new TCP connection.

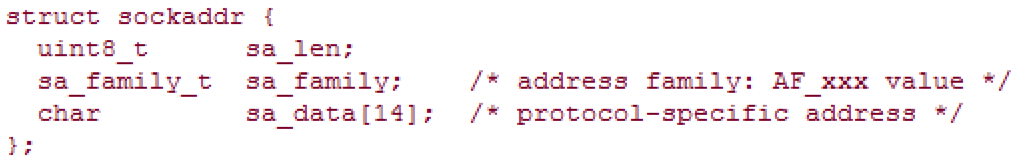
sendto() is used for sending and receiving data to/from a remote socket.

sendmsg() function sends a message through a connection-mode or connectionless-mode socket

SOCKET ADDRESS STRUCTURE

- Four socket functions – **bind(), connect(), sendto(), sendmsg()** -pass socket address structures from application to kernal. All invoke **sockargs() .**
- This function copies socket address structures and explicitly set the **sin_len** member to the size of the structure that was passed.
- The other socket functions that pass socket address to the application from kernal **accept(), recvfrom(), recvmsg(), getpeername() and getsockname()** all set the **sin_len** member before returning to the process.
- *sin_port*, ***sin_family and sin_addr*** are the only required for Posix.1g. ***sin_zero*** is implemented to keep the structure length to 16 byte.

GENERIC SOCKET ADDRESS STRUCTURE

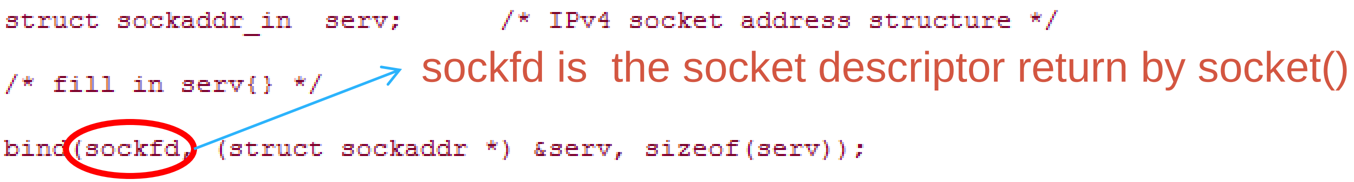


```
struct sockaddr {
  uint8_t      sa_len;
  sa_family_t  sa_family;    /* address family: AF_xxx value */
  char         sa_data[14];  /* protocol-specific address */
};
```

The socket functions are then defined as taking a pointer to the generic socket address structure, as shown here in the ANSI C function prototype for the bind function:

int bind(int, struct sockaddr *, socklen_t);

This requires that any calls to these functions must cast the pointer to the protocol-specific socket address structure to be a pointer to a generic socket address structure. For example,

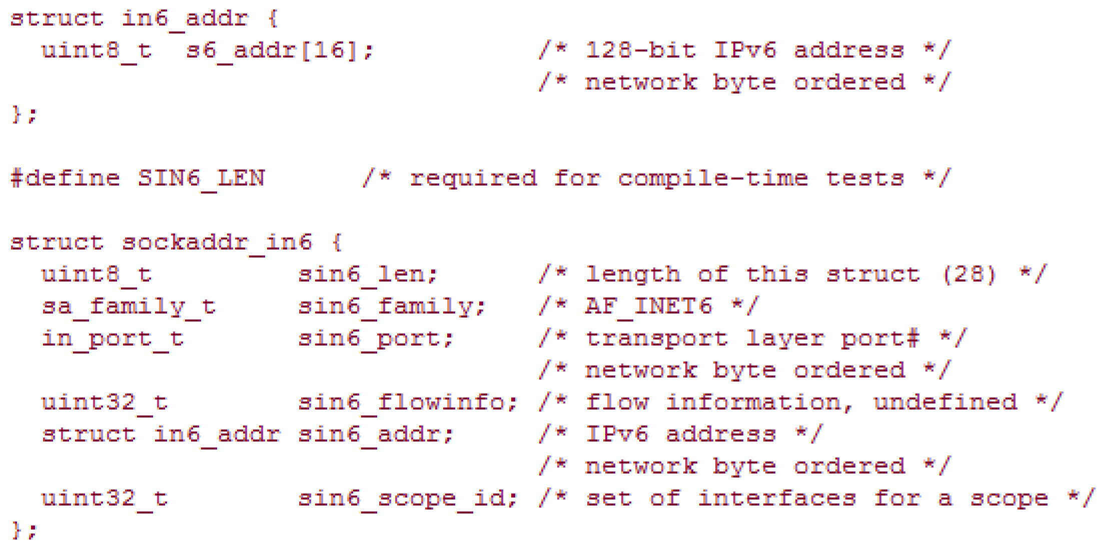


```
struct sockaddr_in  serv;      /* IPv4 socket address structure */

/* fill in serv{} */

bind(sockfd, (struct sockaddr *) &serv, sizeof(serv));
```

sockfd is the socket descriptor return by socket()

IPV6 SOCKET ADDRESS STRUCTURE

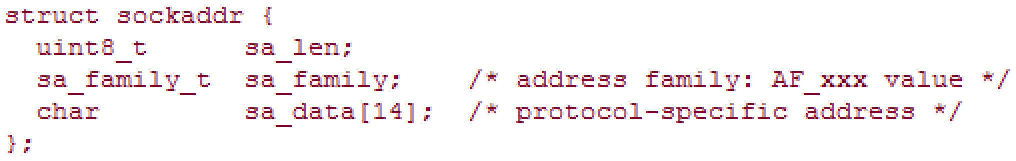
The IPv6 socket address is defined by including the <netinet/in.h> header



```
struct in6_addr {
  uint8_t  s6_addr[16];          /* 128-bit IPv6 address */
                                 /* network byte ordered */
};

#define SIN6_LEN      /* required for compile-time tests */

struct sockaddr_in6 {
  uint8_t         sin6_len;      /* length of this struct (28) */
  sa_family_t     sin6_family;   /* AF_INET6 */
  in_port_t       sin6_port;     /* transport layer port# */
                                 /* network byte ordered */
  uint32_t        sin6_flowinfo; /* flow information, undefined */
  struct in6_addr sin6_addr;     /* IPv6 address */
                                 /* network byte ordered */
  uint32_t        sin6_scope_id; /* set of interfaces for a scope */
};
```

Value-result Arguments

- When a socket address structure is passed to any socket function, it is always passed by reference. That is, a pointer to the structure is passed. The length of the structure is also passed as an argument. But the way in which the length is passed depends on which direction the structure is being passed: from the process to the kernel, or vice versa.
- Three functions, bind, connect, and sendto, pass a socket address structure from the process to the kernel. One argument to these three functions is the pointer to the socket address structure and another argument is the integer size of the structure.

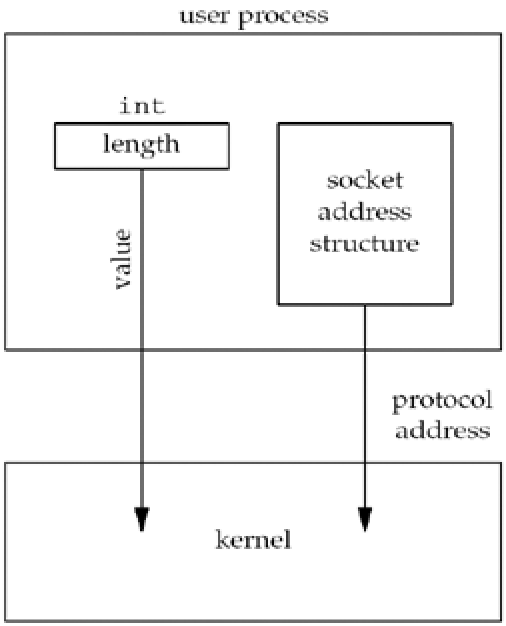


```
struct sockaddr {
  uint8_t      sa_len;
  sa_family_t  sa_family;    /* address family: AF_xxx value */
  char         sa_data[14];  /* protocol-specific address */
};
```

- cont

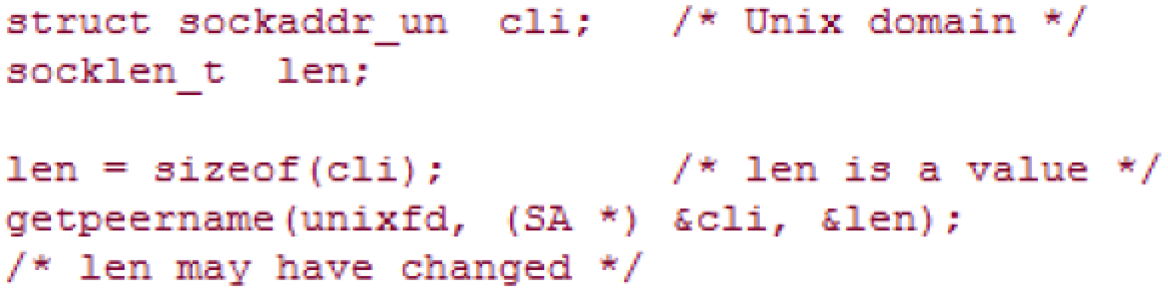
Value-result Arguments

Socket address structure passed from process to kernel.



Value-result Arguments

- Four functions, accept, recvfrom, getsockname, and getpeername, pass a socket address structure from the kernel to the process .
- Two of the arguments to these four functions are the pointer to the socket address structure along with a pointer to an integer containing the size of the structure



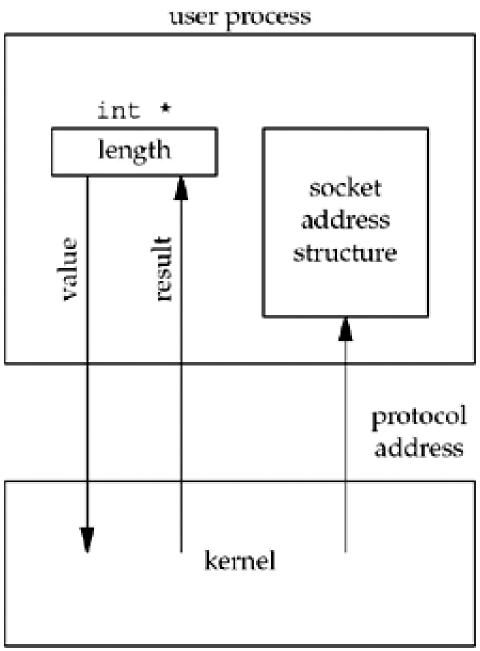
```
struct sockaddr_un  cli;    /* Unix domain */
socklen_t  len;

len = sizeof(cli);          /* len is a value */
getpeername(unixfd, (SA *) &cli, &len);
/* len may have changed */
```

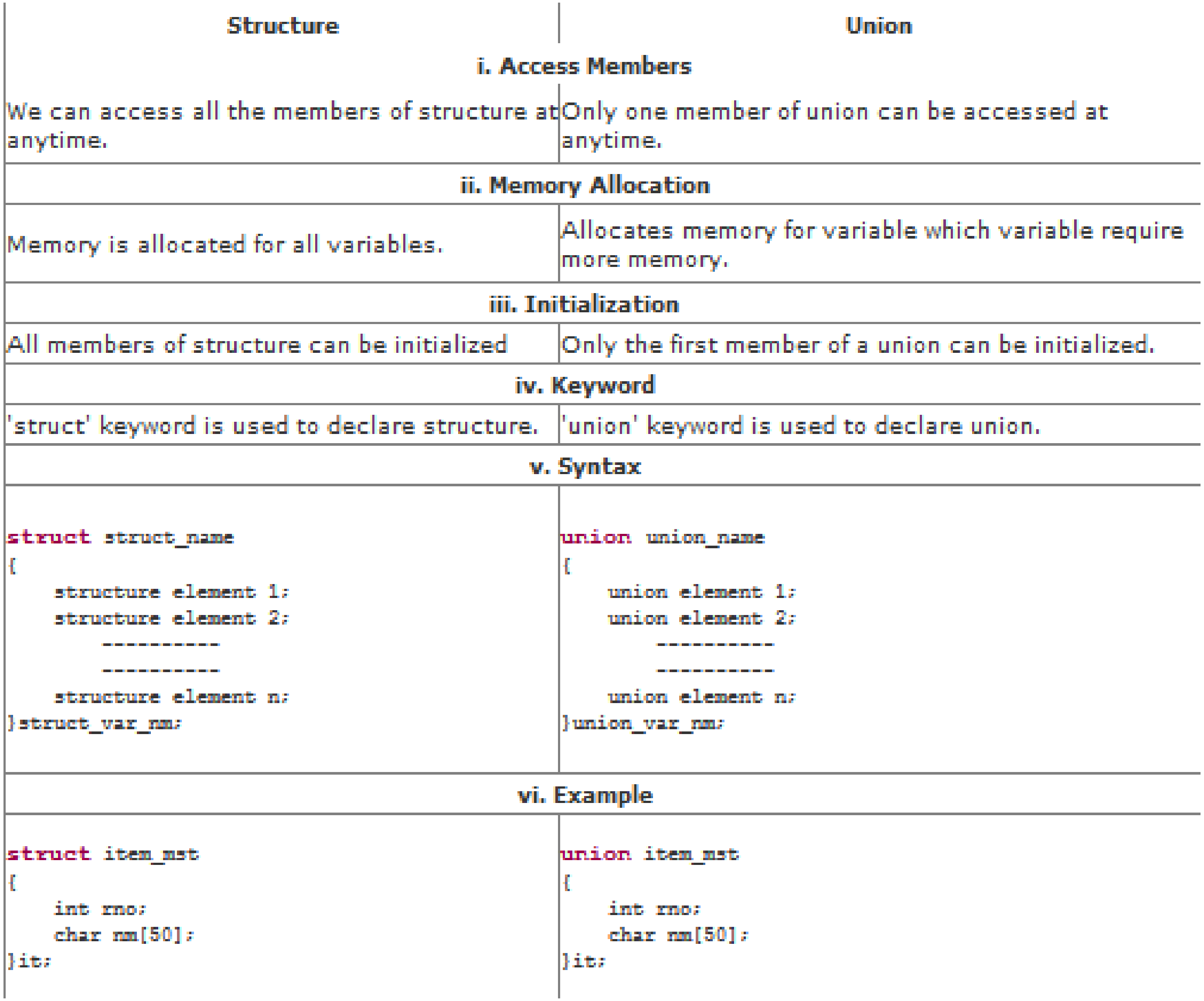
- Cont.

Value-result Arguments

- The reason that the size changes from an integer to be a pointer to an integer is because the size is both a value when the function is called and a result when the function returns
- This type of argument is called a value-result argument

Socket address structure passed from kernel to process.





Structure	Union
i. Access Members	
We can access all the members of structure at anytime.	Only one member of union can be accessed at anytime.
ii. Memory Allocation	
Memory is allocated for all variables.	Allocates memory for variable which variable require more memory.
iii. Initialization	
All members of structure can be initialized	Only the first member of a union can be initialized.
iv. Keyword	
'struct' keyword is used to declare structure.	'union' keyword is used to declare union.
v. Syntax	

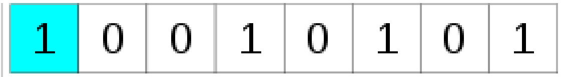
`struct struct_name`
`{`
`    structure element 1;`
`    structure element 2;`
`        ----------`
`        ----------`
`    structure element n;`
`}struct_var_nm;`
	`union union_name`
	`{`
	`    union element 1;`
	`    union element 2;`
	`        ----------`
	`        ----------`
	`    union element n;`
	`}union_var_nm;`

vi. Example	

`struct item_mst`
`{`
`    int rno;`
`    char nm[50];`
`}it;`
	`union item_mst`
	`{`
	`    int rno;`
	`    char nm[50];`
	`}it;`

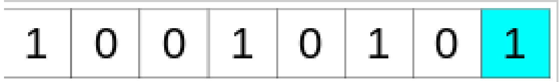
MSB & LSB

- **Most significant bit** (**msb** or **MSB**, also called the **high-order bit**) is the bit position in a binary number having the greatest value.
- Decimal 128



1	0	0	1	0	1	0	1

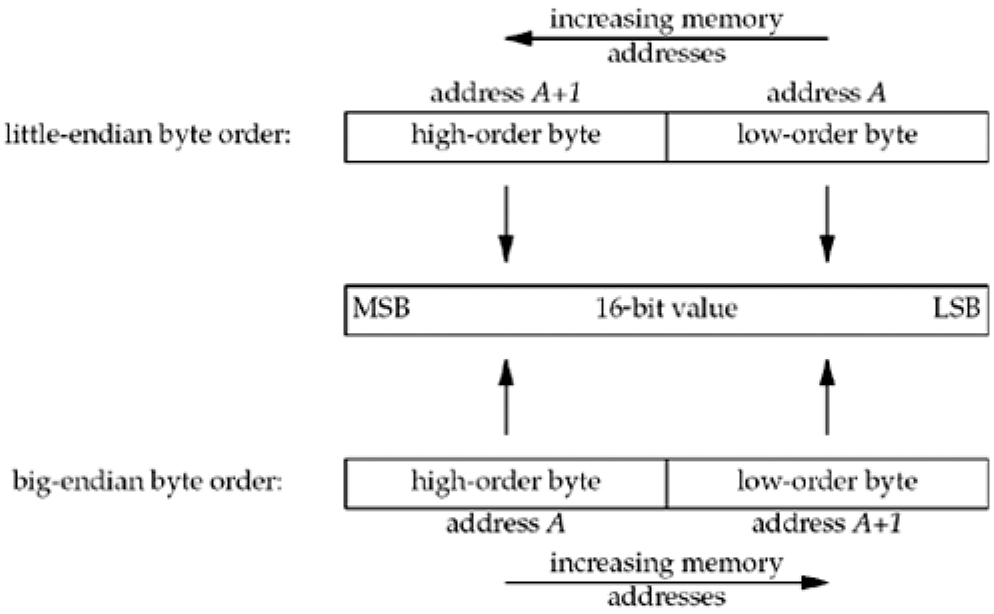
- **Least significant bit** (**lsb**) is the bit position in a binary integer giving the units value
- Decimal 149



1	0	0	1	0	1	0	1

Byte Order Functions

. **Little-endian byte order and big-endian byte order for a 16-bit integer.**



increasing memory addresses

little-endian byte order:

address *A*+1	address *A*
high-order byte	low-order byte

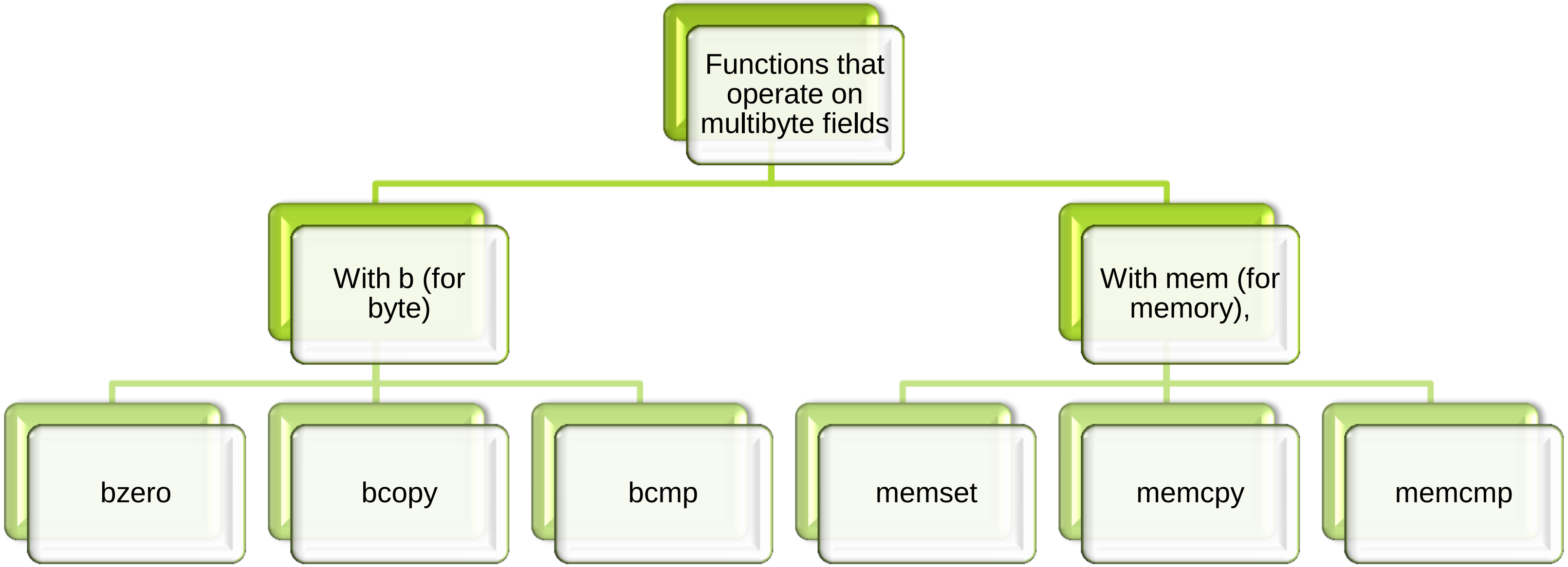
MSB	16-bit value	LSB

big-endian byte order:

high-order byte	low-order byte
address *A*	address *A*+1

increasing memory addresses

Byte Manipulation Functions



Functions that operate on multibyte fields

With b (for byte)

With mem (for memory),

bzero

bcopy

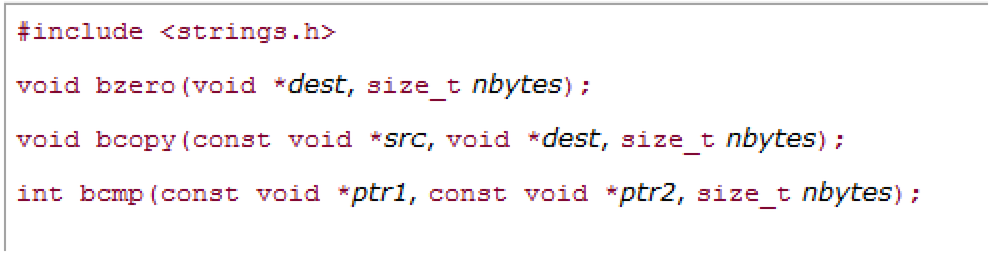
bcmp

memset

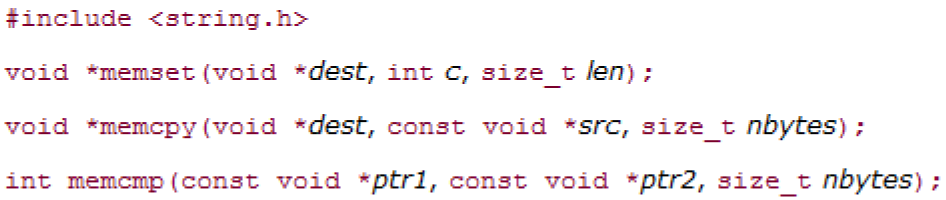
memcpy

memcmp

Byte Manipulation Functions



```
#include <strings.h>

void bzero(void *dest, size_t nbytes);

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

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



```
#include <string.h>

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

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

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

Address conversion functions

- inet_aton, inet_ntoa, and inet_addr convert an IPv4 address from a dotted-decimal string (e.g., "206.168.112.96") to its 32-bit network byte ordered binary value. You will probably encounter these functions in lots of existing code.
- #include <arpa/inet.h>

```
int inet_aton(const char *strptr, struct in_addr *addrptr);
```

Returns: 1 if string was valid, 0 on error

```
in_addr_t inet_addr(const char *strptr);
```

Returns: 32-bit binary network byte ordered IPv4 address;
INADDR_NONE if error

```
char * inet_ntoa(struct in_addr inaddr);
```

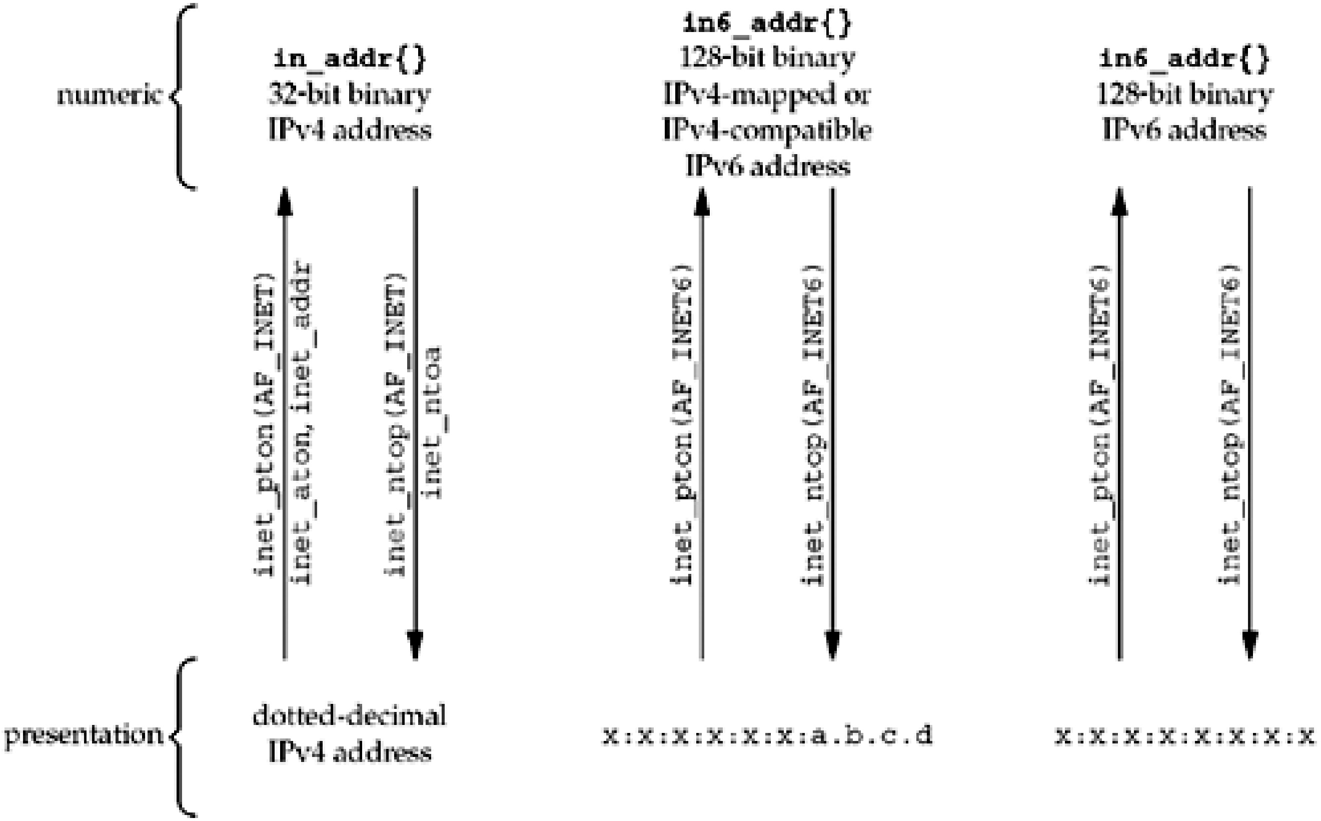
Returns: pointer to dotted-decimal string

Address conversion functions

- **inet_pton** **and** **inet_ntop** **Functions**
- These two functions are new with the IPv6 and work with both IPv4 and IPv6 addresses.
- The letter **p** and **n** stands for **presentation and numeric**. Presentation format for an address is often
- ASCII string and the numeric format is the binary value that goes into a socket address structure

```
#include <arpa/inet.h>
int inet_pton(int family, const char *strptr, void *addrptr);
                        Returns: 1 if OK, 0 if input not a valid presentation format, -1 on error
const char * inet_ntop(int family, const void *addrptr, char *strptr, size_t len);
                                                 Returns: pointer to result if OK, NULL on error
```

Summary Of Address Conversion Functions.



WILDCARD MASK

- A wildcard mask can be thought of as an inverted subnet mask.
- For example, a subnet mask of 255.255.255.0 (binary equivalent = 11111111.11111111.11111111.00000000) inverts to a wildcard mask of 0.0.0.255.

EPHEMERAL PORT

An **ephemeral port** is a short-lived transport protocol port for Internet Protocol (IP) communications allocated automatically from a predefined range by the TCP/IP software.

tcp_ephemeral_low = 32768

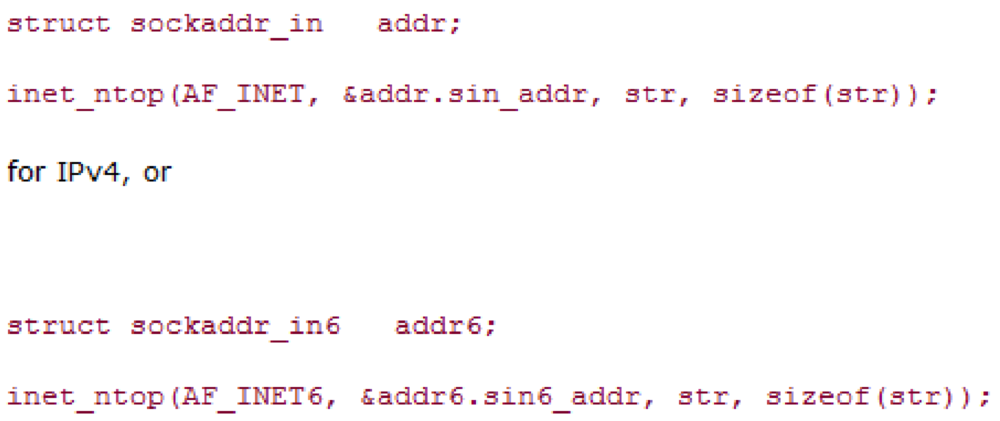
tcp_ephemeral_high = 65535

udp_ephemeral_low = 32768

udp_ephemeral_high = 65535

sock_ntop And Related Functions

- A basic problem with inet_ntop is that it requires the caller to pass a pointer to a binary address. This address is normally contained in a socket address structure, requiring the caller to know the format of the structure and the address family.



```
struct sockaddr_in   addr;

inet_ntop(AF_INET, &addr.sin_addr, str, sizeof(str));
```

for IPv4, or

```
struct sockaddr_in6   addr6;

inet_ntop(AF_INET6, &addr6.sin6_addr, str, sizeof(str));
```

sock_ntop And Related Functions

#include "unp.h"

char *sock_ntop(const struct sockaddr *sockaddr, socklen_t addrlen);

Returns: non-null pointer if OK, NULL on error

sock_ntop And Related Functions

int sock_bind_wild(int sockfd, int family);

Returns: 0 if OK, -1 on error

int sock_cmp_addr(const struct sockaddr *sockaddr1,

const struct sockaddr *sockaddr2, socklen_t addrlen);

Returns: 0 if addresses are of the same family and ports are equal, else nonzero

int sock_cmp_port(const struct sockaddr *sockaddr1,

const struct sockaddr *sockaddr2, socklen_t addrlen);

Returns: 0 if addresses are of the same family and ports are equal, else nonzero

sock_ntop And Related Functions

```
int sock_get_port(const struct sockaddr *sockaddr, socklen_t addrlen);
```

Returns: non-negative port number for IPv4 or IPv6 address, else -1

```
char *sock_ntop_host(const struct sockaddr *sockaddr, socklen_t addrlen);
```

Returns: non-null pointer if OK, NULL on error

```
void sock_set_addr(const struct sockaddr *sockaddr, socklen_t addrlen, void *ptr);
```

```
void sock_set_port(const struct sockaddr *sockaddr, socklen_t addrlen, int port);
```

```
void sock_set_wild(struct sockaddr *sockaddr, socklen_t addrlen);
```

readn, writen, and readline functions

```
#include "unp.h"
ssize_t readn(int filedes, void *buff, size_t nbytes);
ssize_t writen(int filedes, const void *buff, size_t nbytes);
ssize_t readline(int filedes, void *buff, size_t maxlen);
                              All return: number of bytes read or written, –1 on error
```

Assignment # 2

1. Define IPv4 Socket Address Structure
2. Define IPv6 Socket Address Structure
3. What are generic socket address structure?
4. What are Byte Order Function?
5. What are Byte Manipulation Functions?
6. What are Address Conversion functions?
7. Write a note on sock_ntop and Related Functions
8. Dead Line 16 Augest 2013

UNIT 2 END

Prof .Prasad Sawant

sawant_cs@yahoo.com

http://prasadsawant.wordpress.com

http://www.facebook.com/dprasadsawant96k

https://twitter.com/mePrasadSawant