KWANGWOON UNIVERSITY

Chap.3
Socket Introduction

KWANGWOON UNIVERSITY

Section

Section 3.2. Socket Address Structures

Section 3.3. Value-Result Arguments

Section 3.4. Byte Ordering Functions

Section 3.5. Byte Manipulation Functions

Section 3.6. inet_aton, inet_addr, and inet_ntoa Functions

Section 3.7. inet_pton and inet_ntop Functions

Section 3.8. sock_ntop and Related Functions

Section 3.9. readn, writen, and readline Functions

Socket Address Structures

IPv4 Socket Address Structure

```
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 */
};
```

Datatypes required by Posix.1g

Datatype	Description	Header
int8_t	Signed 8bit integer	<sys/types.h>
uint8_t	Unsigned 8bit integer	<sys/types.h>
int16_t	Signed 16bit integer	<sys/types.h>
uint16_t	Unsigned 16bit integer	<sys/types.h>
int32_t	Signed 32bit integer	<sys/types.h>
uint32_t	Unsigned 32bit integer	<sys/types.h>
Sa_family_t	Address family of socket address structure	<sys/socket.h>
Socklen_t	Length od socket address structure normally uint32_t	<sys/socket.h>
In_addr_t	Ipv4 address, normally uint32_t	<netinet/in.h>
In_port_t	TCP or UDP port, normally uint16_t	<netinet/in.h>

Generic Socket address structure

- Socket address structure는 함수에서 포인터 인자로서 사용. 따라서 이기종 protocol의 고유 structure를 handling 하기위해서 사용
- ANSI C : void*
- <sys/socket.h> : Generic Socket address structure

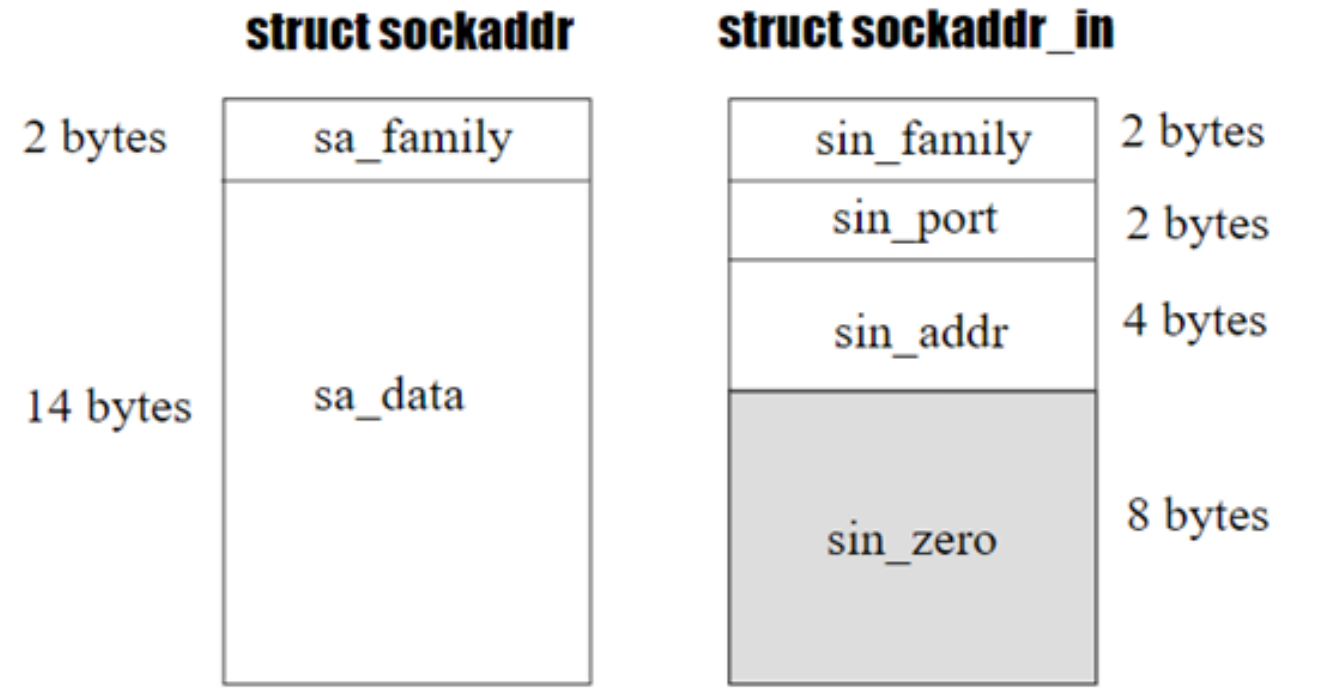
```
struct sockaddr {
    uint8_t   sa_len;
    sa_family_t   sa_family;
     /*address family: AF_xxx    value*/
    char   sa_data[14]; /*protocol specific address*/
}
```

Generic Socket address structure 사용예

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

```
struct sockaddr_in  serv;  /*IPv4 socket address structure*/
 /*  fill in serv{} */
bind(sockfd, (struct sockaddr *) &serv, sizeof(serv));
```

- 즉 이기종 protocol의 structure를 핸들링하기위한 일반 structure 포인터(type casting 주의)



A pointer to a struct sockaddr_in can be cast to a pointer to a struct sockaddr and vice-versa.

➔ Sockaddr 과 sockaddr_in 의 차이점

IPv6 Socket Address Structure

```
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 */
};
```

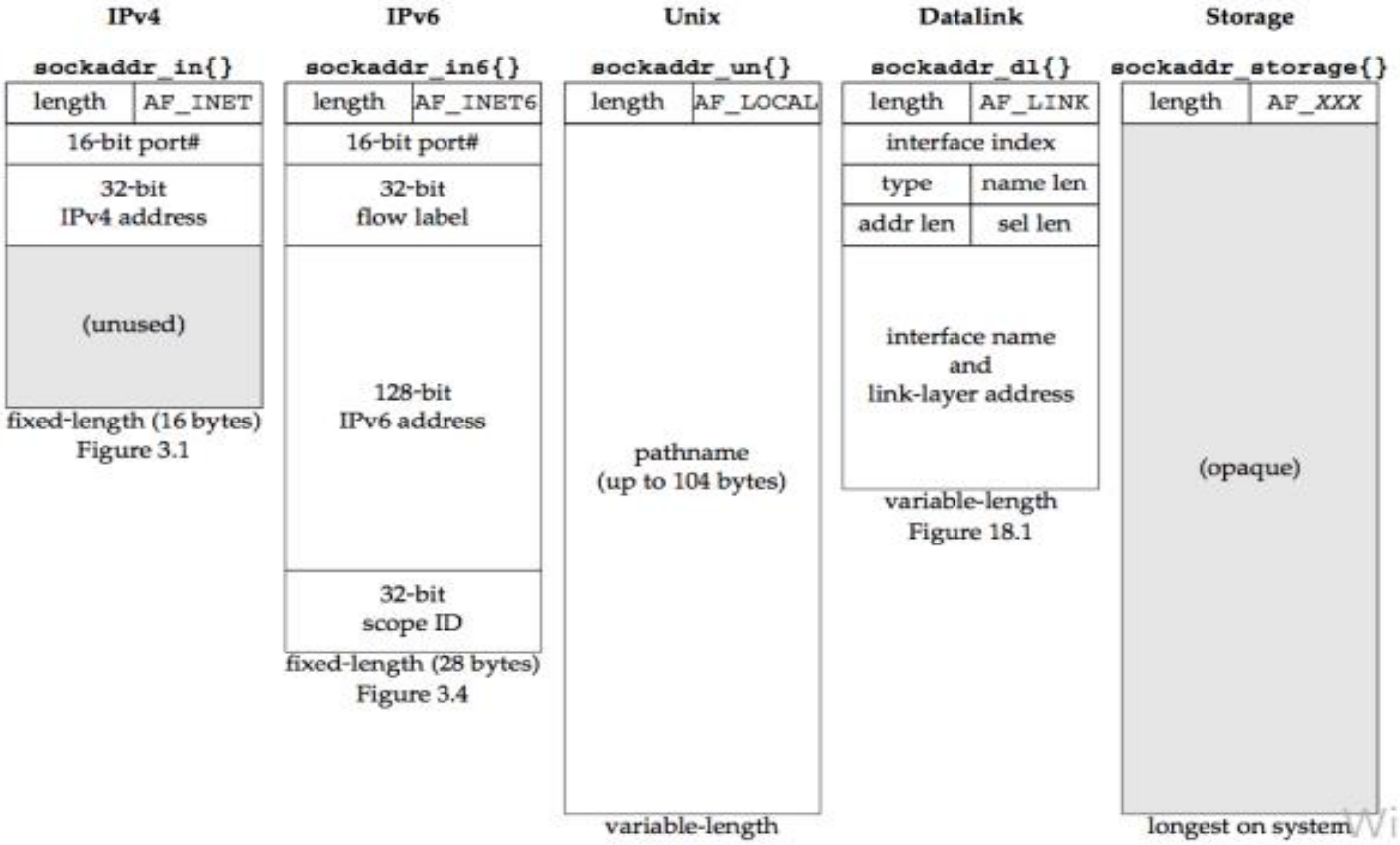
New Generic Socket Address Structure

```
struct sockaddr_storage {
  uint8_t      ss_len;       /* length of this struct (implementation dependent)
  sa_family_t  ss_family;    /* address family: AF_xxx value */
  /* implementation-dependent elements to provide:
   * a) alignment sufficient to fulfill the alignment requirements of
   *    all socket address types that the system supports.
   * b) enough storage to hold any type of socket address that the
   *    system supports.
   */
};
```

1. If any socket address structures that the system supports have alignment requirements, sockaddr_storage provides the strictest alignment requirement

2. The sockaddr_storage is large enough to contain any socket address structure that the system supports.

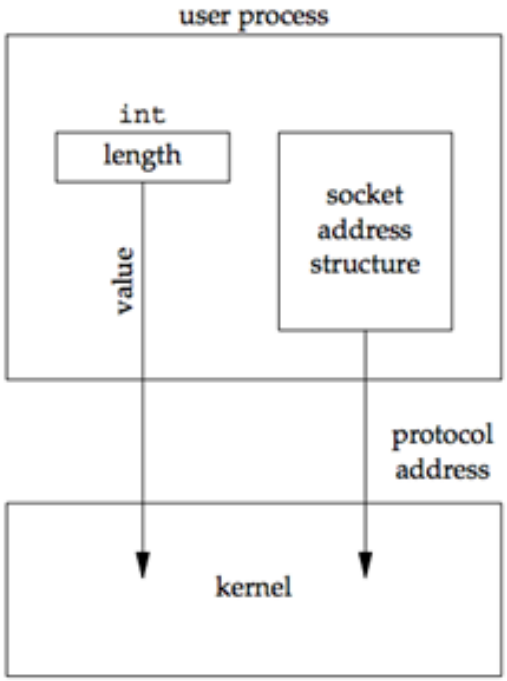
Comparison of Socket Address Structures



Value-Result Arguments

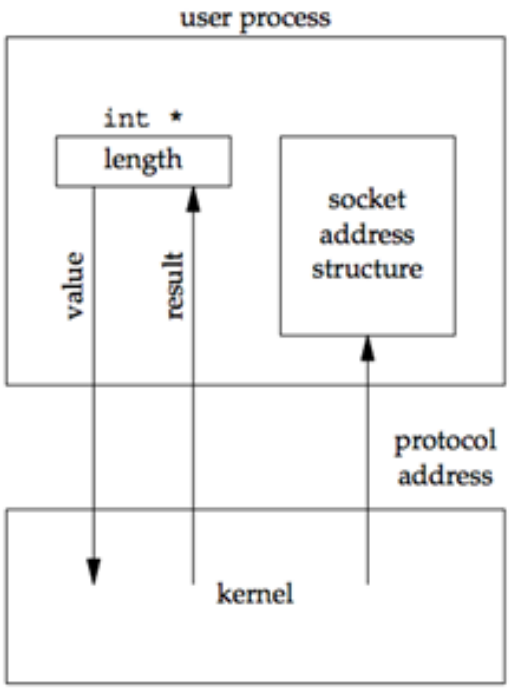
Value-Result Argument

- socket address structure는 socket function 들에 인자로 넘겨질때 , 항상 포인터로 넘겨진다.(passed by reference.)
- socket address structure 가 process 에서 kernel로 가는지 그 역인지에 따라 방법이 다름



->프로세스에서 커널로 전달

Bind, connect, sendto



->커널에서 프로세스로 전달

Accept, recvfrom, getsockname, getpeername

- Process to kernel

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

- Kernel to process

```
struct  sockaddr_un   cli
/* unix domain */

socklen_t    len;
len = sizeof(cli);
getpeername(unixfd, (SA*)&cli,&len);
/* len값은 변할수 있다. */
```

Byte Ordering Functions

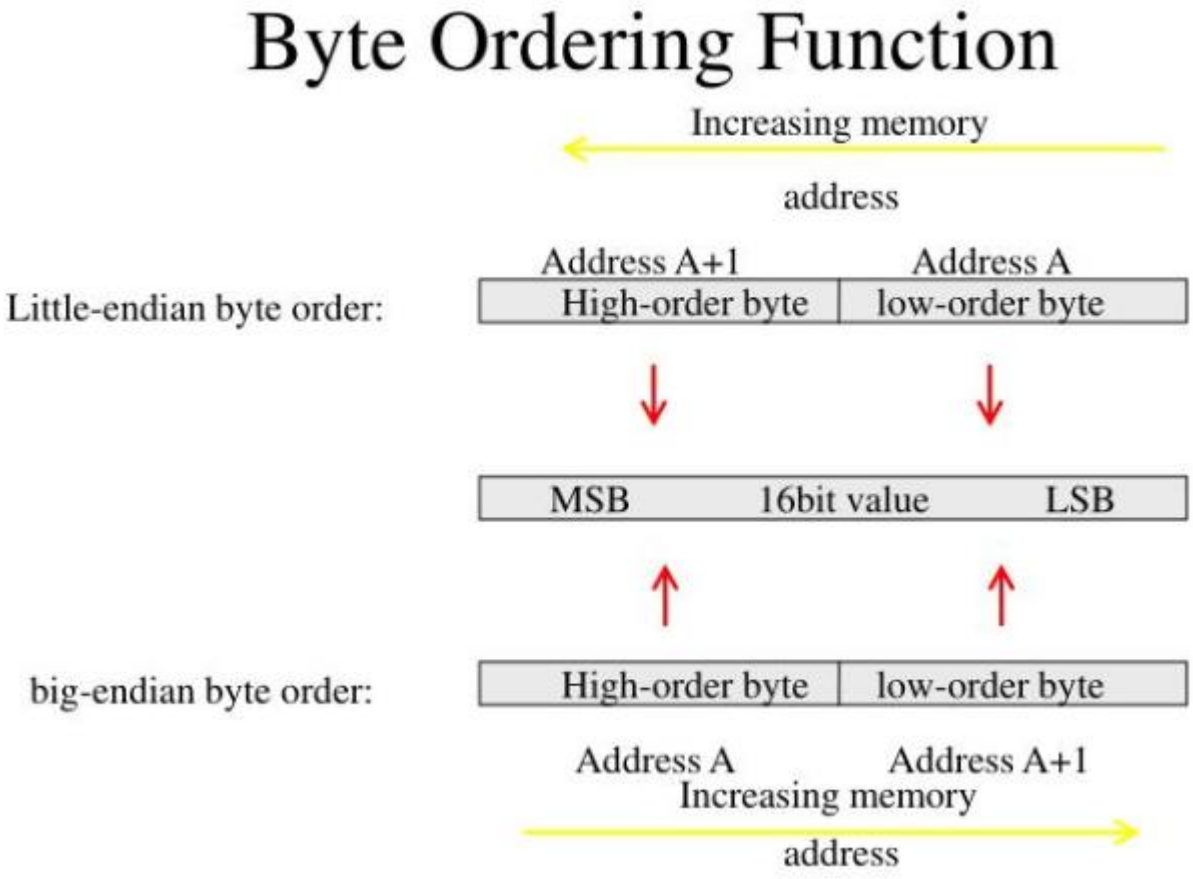


Figure3.9 determine host byte order

```
#include  "unp.h"
int  main(int argc, char **argv)
{
    union {short  s;
           char   c[sizeof(short)];    } un;
    un.s = 0x0102;
    printf("%s: ", CPU_VENDOR_OS);
    if (sizeof(short) == 2) {
        if (un.c[0] == 1 && un.c[1] == 2)        printf("big-endian\n");
          else if (un.c[0] == 2 && un.c[1] == 1)   printf("little-endian\n");
          else  printf("unknown\n");
    } else    printf("sizeof(short) = %d\n", sizeof(short));
    exit(0);
}
```

•network program시 항상 byte ordering에 유의해야함
•Internet protocol은 big-endian byte를 사용(i384계열은 little-endian byte)
•즉 host byte order와 network byte order간의 byte order가 다르면conversion필요

•#include<netinet/in.h>

```
uint16_t    htons  (uint16_t    host16bitvalue);
uint32_t    htonl  (uint32_t    host32bitvalue);
```

Return : value in network byte order

```
uint16_t    ntohs  (uint16_t    net16bitvalue);
uint32_t    ntohl  (uint32_t    net32bitvalue);
```

Return : value in host byte order

h:host n:network s:short(16bit) l:long(32bit)
@host byte order 와 network byte order가 같은 시스템에서 위함수들은 null macro로 정의되있음

Byte Manipulation Functions

Byte Manipulation Function

- #include <strings.h>

  void bzero(void *dest, size_t nbytes);

  /* dest의 nbyte만큼을 0으로 만듬: socket address structure를 0으로 초기화 할때 사용 */

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

  /* src로 부터 nbytes만큼을 dest로 copy */

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

  /* return 0 if equal, nonzero if unequal */

- #include <string.h>

```
void  *memset(void  *dest, int  c, size_t  len);
/* dest에서 len만큼의 byte를  변수 c의 값으로 설정 */

void  *memcpy(void  *dest, const void *src, size_t  nbytes);
/*  src의 nbytes만큼을 dest로 copy  */
int  memcmp(const void  *ptr1, const  void *ptr2, size_t nbytes);
/* ptr1 < ptr2 : less than 0
   ptr1 > ptr2 : greater than 0
    ptr1 = ptr2 :  than 0
   */
```

inet_aton, inet_addr, and inet_ntoa Functions

inet_aton,inet_addr, inet_ntoa function

```
#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 */
```

inet_aton:
strptr이 가리키는 C문자열을 32bit network byte order의 이진값으로 변환
값은 addrptr 포인터를 이용해 저장, 정상이면 1 반환 or 0 반환

inet_addr:
결과값으로32bit network byte order의 이진값을 반환.
문제점: 오류가 있을 때 32bit가 모두 1인 상수 INADDR_NONE을 반환함,
따라서 IPv4의 브로드 캐스트 주소(255.255.255.255)를 처리하지 못한다.

inet_ntoa:
32bit network byte order IPv4주소를 점으로 구분된 십진수 문자열로 변환
이 함수는 구조체를 함수의 인수로 취하는데, 구조에 대한 포인터가 아니라
구조 자체를 인수로서 취한다

inet_pton and inet_ntop Functions

inet_pton,inet_ntop function

- IPv4,IPv6 모두에 대해 address converting
- p : presentation(string) n : numeric(binary)

- #include<arpa/inet.h>
  int inet_pton(int family, const char *strptr, void *addrptr);
  /* return: 1 if OK, 0 if input not a valid presentation format, -1 onerror */
  /* string 을 binary값으로 */

  const char *inet_ntop(int family, const void *addrptr, char *strpt, size_t len);
  /* return : pointer to result if OK, NULL onerror */
  /* len : size of the destination */
  /* binary 값을 string값으로 */

sock_ntop and Related Functions

- **Inet_ntop** -> 호출자가 구조체의 형태와 address family를 알아야 한다 (프로토콜에 의존적)

- **Sock_ntop** -> socket address 구조를 가리키는 포인터를 받아 구조체 field를 살핀 후에 주소의 presentation 형식을 반환하는 적절한 함수를 호출

```
#include "unp.h"

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

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

Sock_ntop definition

```
char *
sock_ntop(const struct sockaddr *sa, socklen_t salen)
{
    char        portstr[8];
    static char str[128];       /* Unix domain is largest */

    switch (sa->sa_family) {
    case AF_INET: {
        struct sockaddr_in  *sin = (struct sockaddr_in *) sa;

        if (inet_ntop(AF_INET, &sin->sin_addr, str, sizeof(str)) == NULL)
            return(NULL);
        if (ntohs(sin->sin_port) != 0) {
            snprintf(portstr, sizeof(portstr), ":%d", ntohs(sin->sin_port));
            strcat(str, portstr);
        }
        return(str);
    }
  /* ... */
```

Related Functions

sock_bind_wild: binds the wildcard address and an ephemeral port to a socket.

sock_cmp_addr: compares the address portion of two socket address structures.

sock_cmp_port: compares the port number of two socket address structures.

sock_get_port: returns just the port number.

sock_ntop_host: converts just the host portion of a socket address structure to presentation format (not the port number)

sock_set_addr: sets just the address portion of a socket address structure to the value pointed to by *ptr*.

sock_set_port: sets just the port number of a socket address structure.

sock_set_wild: sets the address portion of a socket address structure to wildcard

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 maxline);
```

All return: number of bytes read or written, -1 on error

#Readn definition

```
 1 #include     "unp.h"

 2 ssize_t                           /* Read "n" bytes from a descriptor. */
 3 readn(int fd, void *vptr, size_t n)
 4 {
 5     size_t  nleft;
 6     ssize_t nread;
 7     char   *ptr;

 8     ptr = vptr;
 9     nleft = n;
10     while (nleft > 0) {
11         if ( (nread = read(fd, ptr, nleft)) < 0) {
12             if (errno == EINTR)
13                 nread = 0;      /* and call read() again */
14             else
15                 return (-1);
16         } else if (nread == 0)
17             break;              /* EOF */

18         nleft -= nread;
19         ptr += nread;
```

#Readline definition

```
 2 static int read_cnt;
 3 static char *read_ptr;
 4 static char read_buf[MAXLINE];

 5 static ssize_t
 6 my_read(int fd, char *ptr)
 7 {

 8     if (read_cnt <= 0) {
 9       again:
10         if ( (read_cnt = read(fd, read_buf, sizeof(read_buf))) < 0) {
11             if (errno == EINTR)
12                 goto again;
13             return (-1);
14         } else if (read_cnt == 0)
15             return (0);
16         read_ptr = read_buf;
17     }

18     read_cnt--;
19     *ptr = *read_ptr++;
20     return (1);
21 }
```

#Readline definition

```
22 ssize_t
23 readline(int fd, void *vptr, size_t maxlen)
24 {
25     ssize_t n, rc;
26     char    c, *ptr;

27     ptr = vptr;
28     for (n = 1; n < maxlen; n++) {
29         if ( (rc = my_read(fd, &c)) == 1) {
30             *ptr++ = c;
31             if (c  == '\n')
32                 break;          /* newline is stored, like fgets() */
33         } else if (rc == 0) {
34             *ptr = 0;
35             return (n - 1);     /* EOF, n - 1 bytes were read */
36         } else
37             return (-1);        /* error, errno set by read() */
38     }

39     *ptr  = 0;                  /* null terminate like fgets() */
40     return (n);
41 }
```