Lecture #13 – Network Sockets II (Ch 16)

- send / recv

  "send" and "recv" are used to transmit data back and forth on stream sockets (can use read and write instead if we like).

```
int     send(int s, const void *msg, int len, int flags);
int     recv(int s, void *buf, int len, unsigned int flags);
```

  Example:

```
char    *msg = "Hello world";
int     len, num;

len = strlen(msg);
num = send(s, msg, len, 0);
```

  Example:

```
char    *msg;
int     num;

msg = (char *) malloc(MAXLINE+1);
num = recv(s, msg, MAXLINE, 0);
```

- sendto / recvfrom

  "sendto" and "recvfrom" are used to transmit data on datagram sockets (UDP).

```
int     sendto(int s, const void *msg, int len, unsigned int flags,
        const struct sockaddr *to, int tolen);

int     recvfrom(int s, void *buf, int len, unsigned int flags,
        struct sockaddr *from, int *fromlen);
```

  Note: You can also "connect" a datagram socket, and use send / recv instead. The socket will still be a datagram socket using UDP, and the socket interface will automatically add the destination and source information for you.

- getpeername

  int getpeername(int s, struct sockaddr *addr, int *addrlen);

  "getpeername" tells who is connected on the other end of the socket

- gethostname

  int gethostname(char *hostname, size_t size);

  "gethostname" returns the name of the computer we are currently running on

- DNS

  DNS (Domain Name Service) is a distributed database responsible for translating hostnames to IP addresses and vice versa.

  You can use "gethostbyname" to translate hostname to IP address:

```
struct hostent  *gethostbyname(const char *name);

struct hostent {
        char        *h_name;        // official name of host
        char        **h_aliases;    // array of aliases
        int         h_addrtype;     // AF_INET
        int         h_length;       // length of address in bytes
        char        **h_addr_list;  // zero terminated array of addresses
};

#define h_addr h_addr_list[0]
```

  Example:

```
int main(int argc, char **argv)
{
        struct hostent  *h;

        if (argc != 2)
        {
                fprintf(stderr, "usage: %s name\n", argv[0]);
                exit(1);
        }

        if ((h = gethostbyname(argv[1])) == NULL)
        {
                herror("gethostbyname");
```

```
                exit(1);
        }

        printf("Host name  : %s\n", h->h_name);
        printf("IP Address : %s\n", inet_ntoa(*((struct in_addr *)h->h_addr)));

        return 0;
}
```

- Datagram server example

```
int main(int argc, char **argv)
{
        int                     s;
        struct sockaddr_in      caddr, saddr;
        char                    buf[MAXLINE];
        int                     num, addr_len;

        if ((s = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
        {
                perror("socket");
                exit(1);
        }

        saddr.sin_family        = AF_INET;
        saddr.sin_addr.s_addr           = htonl(INADDR_ANY);
        saddr.sin_port          = htons(SERV_PORT);
        memset(&(saddr.sin_zero), '\0', 8);

        if (bind(s, (struct sockaddr *) &saddr, sizeof(saddr)) < 0)
        {
                perror("bind");
                exit(1);
        }

        addr_len = sizeof(struct sockaddr);

        if ((num = recvfrom(s, buf, MAXLINE-1, 0,
                (struct sockaddr *) &caddr, &addr_len)) < 0)
        {
                perror("recvfrom");
                exit(1);
        }

        printf("got packet from %s\n", inet_ntoa(caddr.sin_addr));
        printf("packet is %d bytes long\n", num);
```

```
        buf[num] = 0;
        printf("packet contains \"%s\"\n", buf);

        close(s);
        return(0);
}
```

- Datagram client example

```
int main(int argc, char **argv)
{
        int                     s;
        struct sockaddr_in      addr;
        struct hostent          *host;
        int                     num;

        if (argc != 3)
        {
                fprintf(stderr, "usage: %s hostname message\n", argv[0]);
                exit(1);
        }

        if ((host = gethostbyname(argv[1])) == NULL)
        {
                perror("gethostbyname");
                exit(1);
        }

        if ((s = socket(AF_INET, SOCK_DGRAM, 0)) < 0)
        {
                perror("socket");
                exit(1);
        }

        addr.sin_family         = AF_INET;
        bcopy(host->h_addr, (char *) &addr.sin_addr, host->h_length);
        addr.sin_port           = htons(SERV_PORT);
        memset(&(addr.sin_zero), '\0', 8);

        if ((num = sendto(s, argv[2], strlen(argv[2]), 0,
                (struct sockaddr *) &addr, sizeof(struct sockaddr))) < 0)
        {
                perror("sendto");
                exit(1);
        }
```

```
        printf("send %d bytes to %s\n", num, inet_ntoa(addr.sin_addr));

        close(s);
        return(0);
}
```

- UNIX Domain Sockets (client)

```
int main(int argc, char *argv[])
{
        int                     sockfd, servlen,n;
        struct sockaddr_un      serv_addr;
        char                    buffer[82];

        if (argc != 2)
        {
                fprintf(stderr, "usage: %s <socket path>\n", argv[0]);
                exit(1);
        }

        bzero((char *)&serv_addr,sizeof(serv_addr));
        serv_addr.sun_family = AF_UNIX;
        strcpy(serv_addr.sun_path, argv[1]);
        servlen = strlen(serv_addr.sun_path) + sizeof(serv_addr.sun_family);

        sockfd = socket(AF_UNIX, SOCK_STREAM,0);
        connect(sockfd, (struct sockaddr *) &serv_addr, servlen);

        printf("Please enter your message: ");
        bzero(buffer,82);
        fgets(buffer,80,stdin);

        write(sockfd, buffer, strlen(buffer));
        n = read(sockfd, buffer, 80);

        printf("The return message was [%s]\n", buffer);

        exit(0);
}
```

- UNIX Domain Sockets (server)

```
int main(int argc, char *argv[])
{
        int                     sockfd, newsockfd, servlen, clilen, n;
        struct sockaddr_un      cli_addr, serv_addr;
```

```
        char            buf[80];

        if (argc != 2)
        {
                fprintf(stderr, "usage: %s <socket path>\n", argv[0]);
                exit(1);
        }

        sockfd = socket(AF_UNIX, SOCK_STREAM, 0);

        bzero((char *) &serv_addr, sizeof(serv_addr));
        serv_addr.sun_family = AF_UNIX;
        strcpy(serv_addr.sun_path, argv[1]);
        servlen = strlen(serv_addr.sun_path) + sizeof(serv_addr.sun_family);

        bind(sockfd,(struct sockaddr *)&serv_addr,servlen);
        listen(sockfd, 5);

        clilen = sizeof(cli_addr);

        if ((newsockfd = accept(sockfd,(struct sockaddr *)&cli_addr, &clilen)) < 0)
                error("accepting");

        n = read(newsockfd, buf, 80);

        printf("A connection has been established %d\n", n);
        write(1, buf, n);
        write(newsockfd, "I got your message\n", 19);

        return 0;
}
```

Advanced IPC (Chapter 17)

- Stream Pipes

  A steam pipe is just a bi-directional (full-duplex pipe).

  Note: definitions of s_pipe:

  SVR4:

```
int s_pipe(int fd[2])
{
        return (pipe(fd));
}
```

  4.3+ BSD:

```
int s_pipe(int fd[2])
{
        return (socketpair(AF_UNIX, SOCK_STREAM, 0, fd));
}
```

- Passing File Descriptors

  The ability to pass an open file descriptor between processes is powerful. This allows us to have a server handle all of the details about opening and negotiating a connection, and that complexity can be hidden from the client (child).

  When we pass an open file descriptor from one process to another, we want the passing process and the receiving process to share the same file table entry (see Figure 15.4 on page 480).

  We will define the following functions (implementations to follow):

```
int     send_fd(int fd, int filedes);
int     recv_fd(int fd);
```

  "send_fd" sends the descriptor "filedes" across the stream pipe "fd"
  "recv_fd" is called by the client to receive a descriptor

  Note: I am omitting the "send_err" function and offering simpler implementations

  Protocol specs:

  "send_fd" sends two bytes of 0, followed by the descriptor
  "recv_fd" reads two bytes to make sure they are 0. It then reads the descriptor.

SVR4:

```
int send_fd(int clifd, int fd)
{
        char    buf[2];

        buf[0] = 0;
        buf[1] = 0;

        if (write(clifd, buf, 2) != 2)
                return(-1);

        if (ioctl(clifd, I_SENDFD, fd) < 0)
                return(-1);

        return(0);
}
```

When receiving a descriptor, the 3rd arg is a ptr to struct strrecvfd:

```
        struct strrecvfd {
                int     fd;
                uid_t   uid;
                gid_t   gid;
                Char    fill[8];
        };

int recv_fd(int servfd)
{
        int                     nread,;
        char                    buf[2];
        struct strrecvfd        recvfd;

        if ((nread = read(servfd, buf, 2)) != 2)
                return(-1)

        if (ioctl(servfd, I_RECVFD, &recvfd) < 0)
                return(-1)

        return(recvfd.fd);
}
```

4.3BSD:

To send and receive descriptors, we use "sendmsg" and "recvmsg" which use the structure msghdr as defined in <sys/socket.h>.

```
struct msghdr  {
        caddr_t         msg_name;       /* optional address */
        int             msg_namelen;    /* size of address */
        struct iovec    msg_iov;        /* scatter / gather array */
        int             msg_iovlen;     /* number of elements in array */
        caddr_t         msg_accrights;  /* access rights */
        caddr_t         accrightslen;   /* size of access rights */
};

int send_fd(int clifd, int fd)
{
        struct  iovec   iov[1];
        struct msghdr   msg;
        char            buf[2];

        iov[0].iov_base         = buf;
        iov[0].iov_len          = 2;

        msg.msg_iov             = iov;
        msg.msg_iovlen          = 1;
        msg.msg_name            = NULL;
        msg.msgnamelen          = 0;

        msg.msg_accrights       = (caddr_t) & fd;
        msg.msg_accrightslen= sizeof(int);

        buf[0] = 0;
        buf[1] = 0;

        if (sendmsg(clifd, &msg, 0) != 2)
                return(-1);

        return(0);
}
```

```
int recv_fd (int servfd)
{
        int             nread;
        char            buf[2];
        struct iovec    iov[1];
        struct msghdr   msg;
        iov[0].iov_base = buf;
        iov[0].iov_len          = sizeof(buf);
        msg.msg_iov             = iov;
        msg.msg_iovlen          = 1;
        msg.msg_name            = NULL;
        msg.msg_namelen         = 0;
        msg.msg_accrights       = (caddr_t) &newfd;
        msg.msgaccrightslen     = sizeof(int);

        if ((nread = recvmsg(servfd, &msg, 0)) < 0)
                return(-1);

        return(newfd);
}
```