Lecture 8:Advanced Sockets

References for Lecture 8:

1) Unix Network Programming, W.R. Stevens, 1990,Prentice-Hall, Chapter 6.
2) Unix Network Programming, W.R. Stevens, 1998,Prentice-Hall, Volume 1, Chapter 3-4.

It is also possible to obtain the well-known address of a service or the name of a service on a specialized port.

```
#include <netdb.h>
struct servent *getservbyname(const char *servname, const char *portname);
-- Returns NULL on error. servname = “ftp” for example.
struct servent *getservbyport(int port, const char *portname);
-- returns NULL on error.
stuct servent{
        char    *s_name;      /* official server name*/
        char **s_aliases;    /* list of aliases */
        int        s_port;        /*port number – network byte order */
        char      s_proto;      /* protocol to use */
};
```

Socket Options

Like fcntl( ) for controlling file options, and msgctl/semctl/shmctl( ) for controlling message queue/semaphore/ shared memeory options, the following two functions are for controlling socket options.

#include <sys/socket.h>

int getsocketopt(int *sockfd*, int *level*, int *optname*, void **optval*, socklen_t **optlen*);

int setsocketopt(int *sockfd*, int *level*, int *optname*, const void **optval*, socklen_t *optlen*);

-- returns 0 if OK, -1 on error.

sockfd – an open socket descriptor;

level – who gets/sets the option: socket code, TCP/IP or XNS.

optname – predefined option name.

optval – pointer to the value to set or get. Most option values are integer type.

optlen – length of the option (size of the value), value-result for getsockopt( ); only useful for IP_OPTIONS.

An option can be either a flag (on/off) or a value that can be set or retrieved. Some options can find their places in TCP header or IP header such as TCP_MAXSEG and IP_TOS; some cannot such as TCP_NODELAY and SO_MTU. Flag options use 0 for off and a nonzero value for on. If *optval* has a value of zero after a call to getsockopt( ), that option is currently off. See Figure 6.14 [Stevens ed1:p314].

For TCP/IP, possible levels are:

SOL_SOCKET – for socket option,

IPPROTO_IP – for Ipv 4 option,

IPPROTO_Ipv6 – for Ipv6 option,

IPPROTO_ICMPv6 – for ICMP version6 option,

IPPROTO_TCP – for TCP option,

Socket level optons include:

SO_BROADCAST –f– enable/disable broadcasting. Datagrams only.

SO_DEBUG –f– used for TCP connection to return detailed information on packets

SO_ERROR –f– returns the "so_errno" (defined in <sys/socketvar.h>) value for a socket error. Same value is also stored in Unix errono variable.

SO_KEEPALIVE –f– when no data has been transmitted over a socket for 2 hours, a keepalive probe is sent. If no response is received after several probes are sent, the connection is closed. Used to detect abnomal termination.

SO_LINGER –v– determines whether any unsent data should be sent or discarded when a socket is closed. Close may block until data is sent. Most value options are integer type, but this one use

```
struct    <sys/socket.h>
struct linger { int l_onoff;      /* zero=off, nonzero=on */
                Int l_linger;    /* linger time in seconds */    }
```

SO_OOBINLINE –f– specifies that OOB data also be placed int eh normal input queue.

Ipv4 level options include:

IP_OPTIONS –v–set or fetch options in the IP header.

IP_TOS –v– specifies the type-of-service field in the IP header.

IP_TTL –v– set or fetch the TTL(time-to-live) field – maximum number of hopes.

TCP level options includes:

TCP_MAXSEG –v– returns the maximum segment size. The value is set when the connection is established.

TCP_KEEPALIVE –v– changes the keepalive interval for this connection.

TCP_NODELAY –f– prevents TCP for buffering data to create larger packets. Used for interactive application such as telnet.

#include < fcntl.h>

int fcntl(int *fd*, int *cmd*, int *arg*); /* See[Stvens ed 1: 41-43], here we only discuss socket-related *cmds**/

-- returns 0 if OK, -1 on error.

fd – an open socket descriptor;

cmd – operation to be performed on *fd.*

val – the value to set or get.

Cmd:

- **fcntl(fd, F_GETOWN / F_SETOWN, arg):** get or set the associated process number (*arg* > 0) or the associated process group number (*arg* <0) in order to receive SIGIO or SIGURG.. Only available for terminals and sockets.
- **fcntl(fd, F_GETFL / F_SETFL, FNDELAY / FASYNC):** set or get file flag bits FNDELAY or FASYNC. FNDELAY affects accept, connect, read, write, recv, send, sendto and recvfrom. FASYNC enables the receipt of SIGIO.

Question: How many ways to set a nonblocking socket?

Asynchronous I/O

Process can wait for the kernel to send signal SIGIO when a specified descriptor is ready for I/O. 3 things to do:

1) Establish a handler for SIGIO by calling signal(SIGIO, ???);
2) Set PID or PGID for the descriptor to receive SIGIO by calling fcntl(fd, F_SETOWN, getpid());
3) Enable asynchronous I/O by calling fcntl(fd, F_SETFL,FASYNC).

```
/*    Copy standard input to standard output.     */
#define  BUFFSIZE   4096
main()
{    int       n;
     char      buff[BUFFSIZE];

     while ( (n = read(0, buff, BUFFSIZE)) > 0) write(1, buff, n);
}
```

```
/*   Copy standard input to standard output, using asynchronous I/O.    */
#include<signal.h>
#include<fcntl.h>
#define  BUFFSIZE   4096
int   sigflag;
main()
{    int       n;
     char      buff[BUFFSIZE];
     int       sigio_func();
     signal(SIGIO, sigio_func);                /*   Step 1: set up signal handler*/
     fcntl(0, F_SETOWN, getpid();              /*   Step 2: set descriptor's process ID*/
     fcntl(0, F_SETFL, FASYNC) ;               /*   Step 3: Enable Asynchronous I/O*/
     for ( ; ; ) {
          sigblock(sigmask(SIGIO));            /*   block signal SIGIO to avoid race condition */
          while (sigflag == 0)   sigpause(0);  /* release signals when waiting for a signal.
                                                  Note the difference between pause() and sigpause(0)*/
          /* We're here if (sigflag != 0).   Also, we know that the SIGIO signal is currently blocked.*/
          if ( (n = read(0, buff, BUFFSIZE)) > 0) write(1, buff, n) ;     /* not a loop structure */
         else if (n == 0)     exit(0);         /* EOF */
          sigflag = 0;                          /* turn off our flag */
          sigsetmask(0);                       /* and reenable signals */
     }
}

int sigio_func( )
{    sigflag = 1;        /* just set flag and return */
     /* the 4.3BSD signal facilities leave this handler enabled for any further SIGIO signals. */
}
```

Select( )

When a server (or client) has multiple connections, it can be difficult to guess which clients(or servers) have written data on a socket. One approach, called **polling**, is to use nonblocking recv( ) and loop through all the connections. This is inefficient. Another approach, using **fork( ),** is to fork a child process for each connections. This is also inefficient. A better option is to wait on all the connections simultaneously. This can be done using select( ) function.

```
#include <sys/select.h>
#include <sys/time.h>
int select (int maxfdp1, fd_set *readset, fd_set *writeset, fd_set *exceptset, const strut timeval *timeout);
-- returns # of ready descriptors, 0 if timeout occurs, -1 on error.
```

maxfdp1 – the maximum descriptor to test +1, the possible number of descriptors to test, ≤256.
readset – used to check which connections have data read.
writeset – used to check which connections have space for more output.
exceptset – used to check which connections have exceptions, such as OOB data.
timeout – specifies how long to block waiting for ready connction
There are three options;
  = 0 means the call is nonblocking. Used for polling connections.
  > 0 means the call times out after this amount of time if there are no ready connection during this time.
NULL means the call blocks until a connection is ready for I/O.

The format of the timeval structure is:

```
struct timeval {
   long tv_sec;     /*seconds*/
   long tv_usec;    /*microseconds*/
};
```

select( ) is used to determine which socket are ready for reading, writing, or exception handling. Use NULL for any fd_set that doesn't need to be checked.

The fd_set detatype typically uses one bit per socket fd. The appropriate method for using fd_set is to zero out all the bits and then set each one that is to be tested. The select( ) call modifies the *readset*, *writeset*, and *exceptset* variables by clearing the bits that are not ready for I/O. The user then tests each bit to see which are set and processes the corresponding sockets.

Operations on fd_sets should be performed using the following macros:

```
void FD_ZERO(fd_set *fdset);            /* clear all bits in fdset**/
void FD_SET(int fd, fd_set *dset);      /* turn on the bit for fd in fdset */
void FD_CLR(int fd, fd_set *fdset);     /* clear off the bits in fdset*/
int   FD_ISSET(int fd, fd_set *fdset);  /* test the bit for fd in fdset */
```

See <sys/types.h> for definitions of sd_set and FD_XXX macros.

Example1:

```
int i, n;
fd_set    fdvar;

FD_ZERO(&fdvar);    /* initilize the Set --- all bits off */
FD_SET(1, &fdvar);    /* turn on bit for fd 1 */
FD_SET(4, &fdvar);    /* turn on bit for fd 4 */
FD_SET(5, &fdvar);    /* turn on bit for fd 5 */

If ((n=select(6, &fdvar, NULL, NULL, NULL))<0) printf("Something wrong!\n");
   /* only want to check the readset.*/

for (i=0, i<6, i++) if (FD_ISSET(i, &fdvar)>0) handle(i); /* fd i had data for read, call handle(i) */
```

Example2:

```
#include"unp.h"
void str_cli(FILE *fp, int sockfd)
{     int               maxfdp1;
      fd_set          rset;
      char       sendline[MAXLINE], recvline[MAXLINE];

      FD_ZERO(&rset);
      for ( ; ; ) {
            FD_SET(fileno(fp), &rset);
            FD_SET(sockfd, &rset);
            maxfdp1 = max(fileno(fp), sockfd) + 1;
            Select(maxfdp1, &rset, NULL, NULL, NULL);

            if (FD_ISSET(sockfd, &rset)) {      /* socket is readable */
                  if (Readline(sockfd, recvline, MAXLINE) == 0)
                        err_quit("str_cli: server terminated prematurely");
                  fputs(recvline, stdout); }

            if (FD_ISSET(fileno(fp), &rset)) {    /* input is readable */
                  if (Fgets(sendline, MAXLINE, fp) == NULL)
                        return;           /* all done */
                  writen(sockfd, sendline, strlen(sendline));    }
      }
}
```

Notes: select( ) can be used for a more accurate timer than sleep( ).
select() can be used for waiting for a connection request.

Socket-related Signals:

1) SIGIO :

- sindicates that a socket is ready for asynchronous I/O as we have discussed.
- need to specify process ID or process group ID to receive the signal.
- Need to enable asynchronous I/O.

2) SIGURG:

- indicates urgent data is coming due to 1)OOB data or 2) control status information.
- need to specify process group ID to receive the signal,e.g., fcntl(sd,F_SETOWN, -getpgid( )).
- Use flag=MSG_URG to send and receive the OOB data.
- If O_OOBINLINE is set, we must use STOCATMARK ioctl to read OOB data.

```
    setsockopt(sd, SOL_SOCKET, SO_OOBINLINE, &seton, sizeof(seton)); /*let seton=1*/
  if ((n=ioctl(sd,STOCATMARK, &start)>0) read(sd, buf, n);    /*OOB data is in buf with n bytes.*/
```

3) SIGPIPE:

- indicates socket, pipe, or FIFO can never be written to.
- Sent only to the associated process,

Internet Superserver --- inetd

How many typical network servers?

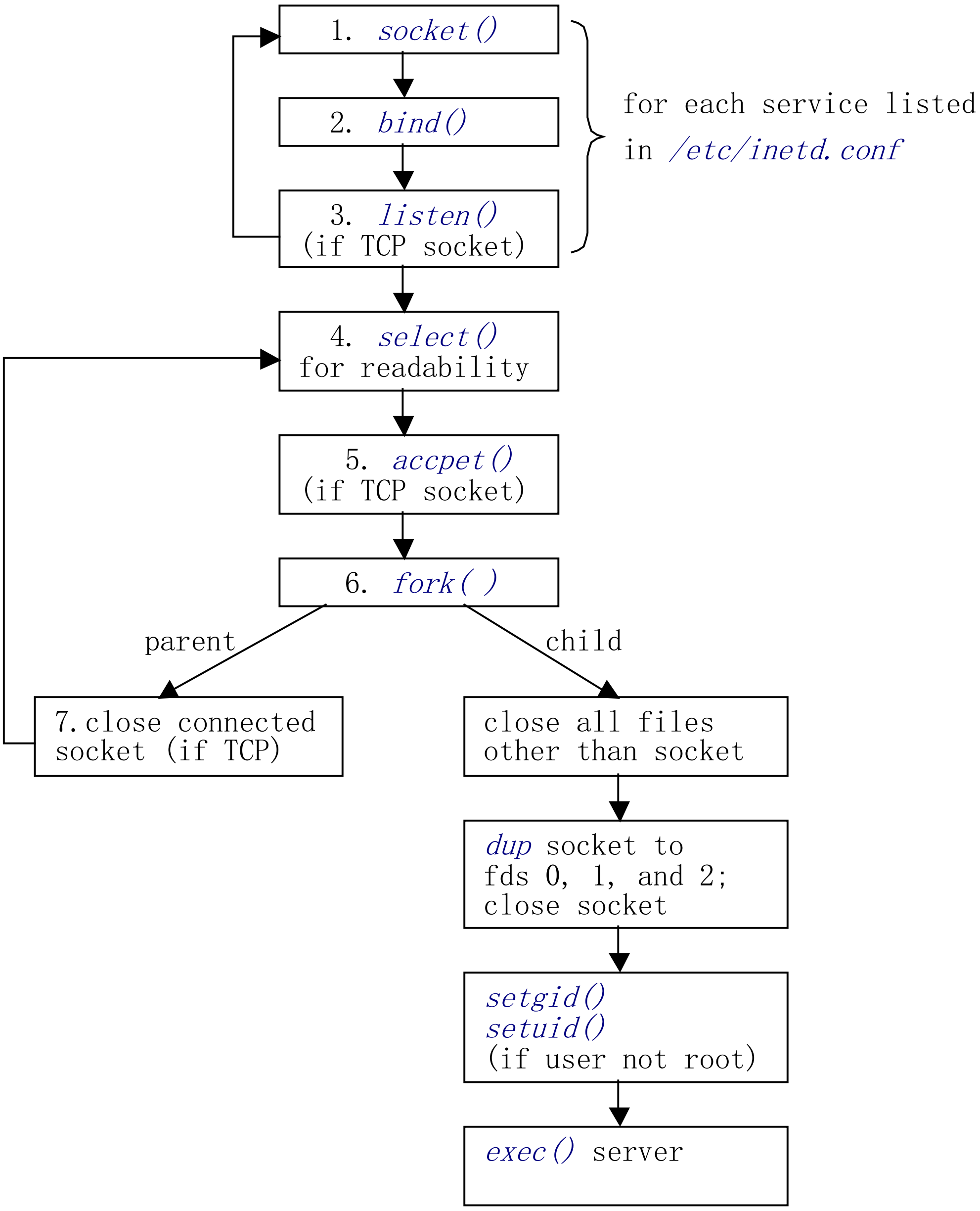
- telnet, ftp, tftp, remote login, remote shell
- started from /etc/rc
- did the same startup tasks: socket, bind, listen, accept, fork, …

How to use select( ) to combine them into one daemon?

- 4.3 BSD supersever: inetd
- reduce the number of processes
- simplify the writing of daemon processes since they have the same startup tasks and skeleton daemon tasks (see Lecture 1 for skeleton daemon).

Flow chart of inetd (version2: section 12.5 or version1:section 6.16)

1) read /etc/inetd.conf to create one socket for each service in the file.
2) read /etc/services to bind well-known port numbers to each service.
3) Listen() only for TCP.
4) Select() can be used for connect requests that arrives at the socket for reading.
5) If it is TCP request, call accept().
6) Fork a child process to handle the request
   6.1) close all files except socket
   6.2) dup2(sd,0), dup2(sd,1), and dup2(sd, 2).
   6.3) login program: a superuser can become any user. Must in the order of setgid() first and then setuid().
   6.4) exec() to execute server_program accordingly.
7) Parent goes up to accept next request without wait.



Steps performed by inetd