Sockets

Socket connection

- A Socket is another method of inter-process communication. It allows us to develop the true distributed client/server applications to run either locally, or across networks.
- Socket connection is as phone call (client) to an operator, and operator put it through the correct department(server process), and then from there to the right person(server socket).
- A Socket is characterized by three attributes: Socket Domain, Socket Type and Protocol.
- A Socket domain is an abstraction that provides an addressing structure and a set of protocols. Sockets connect only with sockets in the same domain.

- Most popular domains include:

  `AF_UNIX`

  `UNIX internal(via the file system)`

  `AF_INET`

  `UNIX network sockets(TCP/IP networking)`

- Socket Type define the message transmission properties. Processes communicate only between sockets of the same type.

- Internet protocols provide two distinct types of service:

`SOCK_STREAM`	`A Stream socket`
`SOCK_DGRAM`	`A Datagram socket`

- A Stream socket provides two-way, sequenced, reliable, unduplicated flow of data(byte stream) with no record boundaries.

- A Stream socket guarantees the data not to be lost, duplicated or reordered without an indication of an error. Large messages are fragmented, transmitted and re-assembled.

- A Datagram socket supports a two-way flow of messages. The order of message is not guaranteed, it may get lost or duplicated, also record boundaries are preserved.

Socket Function Prototypes

1. Creating a Socket

   ```
   #include <sys/types.h>
   #include <sys/socket.h>

   int socket(int domain, int type, int protocol)
   ```

   protocol is determined by the socket domain and type. Normally not specified. The sytem select the default protocol that supports the specified socket type. The socket handle (a file descriptor) is returned.

   A remote process has no way to identify a socket until an address is bound to it. Communicating processes connect through addresses.

   In the UNIX domain, a connection is usually composed of one or two path names. In the Internet domain, a connection is composed of local and remote addresses and local and remote ports.

2. Naming a Socket

   ```
   #include <sys/types.h>
   #include <sys/socket.h>

   int bind(int sr_sd, struct sockaddr *address,
            size_t addr_length);
   ```

   To make the socket available for use by other processes, a server program must give the socket a name.

   bind()is called to bind a path or internet address to a socket. It returns 0 for success, -1 and set of errno for failure.

   Parameters: sd is the socket handle, the value returned by creating call. address is described by a structure “const struct sockaddr”. Details see below.

3. Socket address

   Each socket domain requires its own address format. In UNIX domain, the address is described as following structure:

```
#include <sys/un.h>

struct sockaddr_un{
sa_family_t  sun_family;
char         sun_path[];
};
```

In Internet domain, the address is specified with this structure:

```
#include <sys/netinet/in.h>

struct sockaddr_in{
short int           sin_family;
unsigned short int  sin_port;
struct in_addr      sin_addr;
};
```

The IP address structure is defined as,

```
#include <sys/netinet/in.h>

struct in_addr{
unsigned long int      s_addr;
};
```

4. Creating a Socket queue

```
#include <sys/socket.h>

int listen(int sr_sd, int backlog);
```

Server create a queue for storing the pending requests. It allows incoming connections to be held pending while server program is busy dealing with a previous client.

The size of the queue is set to “backlog”, the value of 5 is common used.

listen() returns 0 on success, -1 and a set of errno for failure.

5. Requesting a connections

```
#include <sys/socket.h>

int connect(int sd, struct sockaddr *address,
    size_t addr_length);
```

Client program requests a connection between an unnamed socket to the server and the server listen the socket.

Parameters: sd is the *client socket descriptor*. address is the *client socket address*.

6. Accepting a connections

```
#include <sys/socket.h>

int accept(int sd, struct sockaddr *address,
           size_t *addr_length);
```

   Once a server has created a socket, and named it, it can accept a request for connection by using accept() call.

   accept() creates a new socket to communication with client and return its descriptor. The new socket will the have a the same type as teh server listen socket.

   Parameter: sd is the *server socket descriptor*; address is the *calling client's address*.

7. Closing a Socket

```
#include <sys/socket.h>

int  close(sd);
```

Applications

socket on file system

1. A local client

```
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <unistd.h>
#include <stdio.h>

int main()
{
  int sockfd; //socket descriptor
  int len; //len of address

  struct sockaddr_un address;
  int result; //result from connection

  char ch = 'A';
/* client create a socket*/
  sockfd = socket(AF_UNIX, SOCK_STREAM, 0);

/*associate the socket as agree with server*/
  address.sun_family = AF_UNIX;
  strcpy(address.sun_path, "server_socket");
  len= sizeof(address);

/* request a connection to server */
  result= connect(sockfd, (struct sockaddr *)
                   &address, len);
  if(result == -1){
    perror("oops, client 1");
    exit(1);
  }
/*now write to the server */
  write(sockfd, &ch, 1);
  read(sockfd, &ch, 1);
  printf("char from server = %c\n", ch);
  close(sockfd);
  exit(0);
}
```

2. A local server

```
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <unistd.h>
#include <stdio.h>

int main()
{
  int server_sockfd;
  int client_sockfd;
  int server_len;
  int client_len;

  struct sockaddr_un server_address;
  struct sockaddr_un client_address;

/* server remove any existing socket*/
  unlink("server_socket");

/* server create an new socket*/
server_sockfd = socket(AF_UNIX,SOCK_STREAM,0);
```

```
/* Name the socket, it available to client*/
  server_address.sun_family= AF_UNIX;
  strcpy(server_address.sun_path,
                 "server_socket");
  server_len = sizeof(server_address);
  bind(server_sockfd, (struct sockaddr *)
             &server_address, server_len);

/* create a socket queue wait for client*/
  listen(server_sockfd, 5);
  while(1){
    char ch;
    printf("server waiting\n");

    /* accept a client*/
    client_len= sizeof(client_address);
    client_sockfd= accept(server_sockfd,
                     (struct sockaddr *)
                      &client_address,
                            &client_len);

    /*read and write to client*/
    read(client_sockfd, &ch, 1);
    ch ++;
    write(client_sockfd, &ch, 1);
    close(client_sockfd);
  }
}
```

socket on network

1. Network client

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <unistd.h>
#include <stdio.h>

int main()
{
  int sockfd; //socket descriptor
  int len; //len of address
  struct sockaddr_in address;
  int result; //result from connection

  char ch = 'A';

/* client create a socket*/
  sockfd = socket(AF_INET, SOCK_STREAM, 0);

/*associate the socket as agree with server*/
  address.sin_family = AF_INET;
/*system choose default IP address loopback*/
  address.sin_addr.s_addr= htonl(INADDR_ANY);
  address.sin_port= 9734;
  len= sizeof(address);

/* request a connection to server */
  result= connect(sockfd, (struct sockaddr *)
                  &address, len);
  if(result == -1){
    perror("oops, client 1");
    exit(1);
  }
/*now write to the server */
```

```
    write(sockfd, &ch, 1);
    read(sockfd, &ch, 1);
    printf("char from server = %c\n", ch);
    close(sockfd);
    exit(0);
  }
```

2. Network server

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <unistd.h>
#include <stdio.h>

int main()
{
  int server_sockfd;
  int client_sockfd;
  int server_len;
  int client_len;
  struct sockaddr_in server_address;
  struct sockaddr_in client_address;

/* server create an new socket*/
server_sockfd = socket(AF_INET,SOCK_STREAM,0);

/* Name the socket, it available to client*/
  server_address.sin_family= AF_INET;
/*allow the connection from any IP address*/
  server_address.sin_addr.s_addr
                    = htonl(INADDR_ANY);
  server_address.sin_port= 9734;
  server_len = sizeof(server_address);
  bind(server_sockfd, (struct sockaddr *)
            &server_address, server_len);

/* create a socket queue wait for client*/
  listen(server_sockfd, 5);
  while(1){
    char ch;
    printf("server waiting\n");

    /* accept a client*/
```

```
        client_len= sizeof(client_address);
        client_sockfd= accept(server_sockfd,
                        (struct sockaddr *)
                         &client_address,
                              &client_len);

        /*read and write to client*/
        read(client_sockfd, &ch, 1);
        ch ++;
        write(client_sockfd, &ch, 1);
        close(client_sockfd);
      }
    }
```

Multiple Clients

So far, we've seen the client and server communication through the file system or across the network.

We would like to have several clients simultanously connecting to server. How to do?

After the server socket is created, the server program calls fork(), then a second copy of server is generated, also the open socket is inherited.

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <signal.h>
#include <unistd.h>
#include <stdio.h>

int main()
{
  int server_sockfd;
  int client_sockfd;
  int server_len;
  int client_len;
  pid_t pid;

  struct sockaddr_in server_address;
  struct sockaddr_in client_address;
```

```
/* server create an new socket*/
server_sockfd = socket(AF_INET,SOCK_STREAM,0);

/* Name the socket, it available to client*/
  server_address.sin_family= AF_INET;
/*allow the connection from any IP address*/
  server_address.sin_addr.s_addr
                    = htonl(INADDR_ANY);
  server_address.sin_port= 9734;
  server_len = sizeof(server_address);
  bind(server_sockfd, (struct sockaddr *)
             &server_address, server_len);

/* create a socket queue wait for client*/
  listen(server_sockfd, 5);
/*don't wait for child to complete*/
  signal(SIGCHLD, SIG_IGN);
  while(1){
    char ch;
    printf("server waiting\n");

    /* accept a client*/
    client_len= sizeof(client_address);
    client_sockfd= accept(server_sockfd,
                     (struct sockaddr *)
                      &client_address,
                            &client_len);

/*create a process for this client*/
    pid = fork();

    if(pid==0){ // this is child
    /*read and write to client*/
    read(client_sockfd, &ch, 1);
    sleep(3); //3 secd delay
    ch ++;
    write(client_sockfd, &ch, 1);
    close(client_sockfd);
    exit(0);
    }
    else{ //this is parent
/* our work for this client is finished*/
      close(client_sockfd);
    }
  }
}
```

Network Information

For general server and client, we shall be able to consult the network information to determine address and ports to use.

Given a computer's name, we can determine the IP address by calling host database functions that resolve the address for us.

Structures returned by functions

1. Structure hostent

   ```
   #include <netdb.h>

   struct hostent *gethostbyaddr(void *addr,
                                 size_t len, int type);

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

   The structure of hostent is defined as followed:

```
struct hostent{
     char *h_name;     //name of the host
     char **h_aliases  //list of nicknames
     int  h_addrtype;  //address type
     int h_length;     //length(byte) of addr
     char **h_addr_list; // list of address
};
```

2. Structure servent

```
#include <netdb.h>

struct servent *getservbyname(void *addr,
                          const char *proto);

struct servent *getservbyport(int port,
                         const char *proto);
```

The *proto* parameter specifies the protocol to be used to connect to the service, either *tcp* for stream socket or *udp* for datagrams.

The structure *servent* is defined as followed:

```
struct servent{
     char *s_name;  //name of server
     char **s_aliases  //list of nicknames
     int s_port;       // IP port number
     char *s_proto;    // tcp or udp
};
```

3. Convert address to printable string

```
#include <arpa/inet.h>

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

Function prototype

```
#indude <unistd.h>

int gethostname(char *name, int namelength);
```

This function writes the name of the current host in to the string given by*name*. The *namelenght* indicates the length of the string*name*.

gethostname returns 0 on success and -1 on error.

Example: host infomation

```
#include <netinet/in.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <netdb.h>
#include <stdio.h>

int main(int argc, char *argv[])
{
   char *host;       //host name
  char **names;    //list of nicknames
  char **addrs;    //list of address

  struct hostent *hostinfo;

  /* set the host to the argument*/
  if(argc == 1){
    char myname[256];
   /*use this computer's name*/
    gethostname(myname, 255);
    host = myname;
  }
```

```
else /*use the given name*/
  host= argv[1];

/* call gethostbyname */
hostinfo = gethostbyname(host);
if(!hostinfo){
  printf("can't get info for host %s\n", host);
  exit(1);
}

printf("Results for host %s\n", host);
printf("Name: %s\n", hostinfo -> h_name);
printf("Aliases: ");
names= hostinfo -> h_aliases;

while(*names){
  printf("%s", *names);
  names++;
}
printf("\n");

/*warning if the host isn't an IP host*/
if(hostinfo -> h_addrtype != AF_INET){
    printf("not a IP host!\n");
    exit(1);
  }

  /*else display the IP address*/
  addrs= hostinfo -> h_addr_list;
  while(*addrs){
    printf("%s", inet_ntoa(*(struct in_addr *)
                                           *addrs));
    addrs++;
  }
  printf("\n");
  exit(0);
}
```

THE END