BITS Pilani
Pilani | Dubai | Goa | Hyderabad

Sockets Overview

Srikanth Gunturu

Guest Faculty
BITS, WILP

In this segment

Sockets Overview

- Introduction
- Data Structures
- Library Calls
- General Operation
- Socket Example

Socket - Introduction

What is it ?

- Socket is an internal endpoint for sending/receiving data within a node on network
- Berkeley/POSIX sockets defined API for Inter Process Communication (IPC) within same host (BSD 4.2 – circa 1983)
- Early form of Middleware (limited to same host systems)
- Windows variant (WinSock) based on BSD Sockets.
- Treated similar to files in BSD/POSIX
- Maintained in File Descriptor table
- Supported protocols
  - TCP/IP – IPv4, IPv6
  - UDP

Socket – Data Structures

Socket Address

- Defined in *sys/socket.h*
- Address length is 8 bytes by default
- Family – Denotes the family of protocol (TCP, UDP)
- Address contains – host and port
- Socket Address family TCP/IP
  - IPv4 - sockaddr_in

```
sa_family_t         sin_family;
in_port_t           sin_port;
struct   in_addr    sin_addr;
```

  - IPv6 - sockaddr_in6

```
sa_family_t         sin6_family;
in_port_t           sin6_port;
uint32_t            sin6_flowinfo;
struct in6_addr     sin6_addr;
uint32_t            sin6_scope_id;
```

```
struct sockaddr
  {
    unsigned char  sa_len;    // length of address
    sa_family_t  sa_family;   // the address family
    char  sa_data[14];        // the address
  };
```

Socket – Library Calls

Socket APIs

socket —creates a descriptor for use in network communications

connect —connect to a remote peer (client)

write —send outgoing data across a connection

read —acquire incoming data from a connection

close —terminate communication and deallocate a descriptor

bind —bind a local IP address and protocol port to a socket

listen —set the socket listening on the given address and port for connections from the client and set the number of incoming connections from a client (backlog) that will be allowed in the listen queue at any one time

accept —accept the next incoming connection (server)

recv —receive the next incoming datagram

recvmsg —receive the next incoming datagram (variation of recv)

recvfrom —receive the next incoming datagram and record its source endpoint address

send —send an outgoing datagram

sendmsg —send an outgoing datagram (variation of send)

sendto —send an outgoing datagram, usually to a prerecorded endpoint address

shutdown —terminate a TCP connection in one or both directions

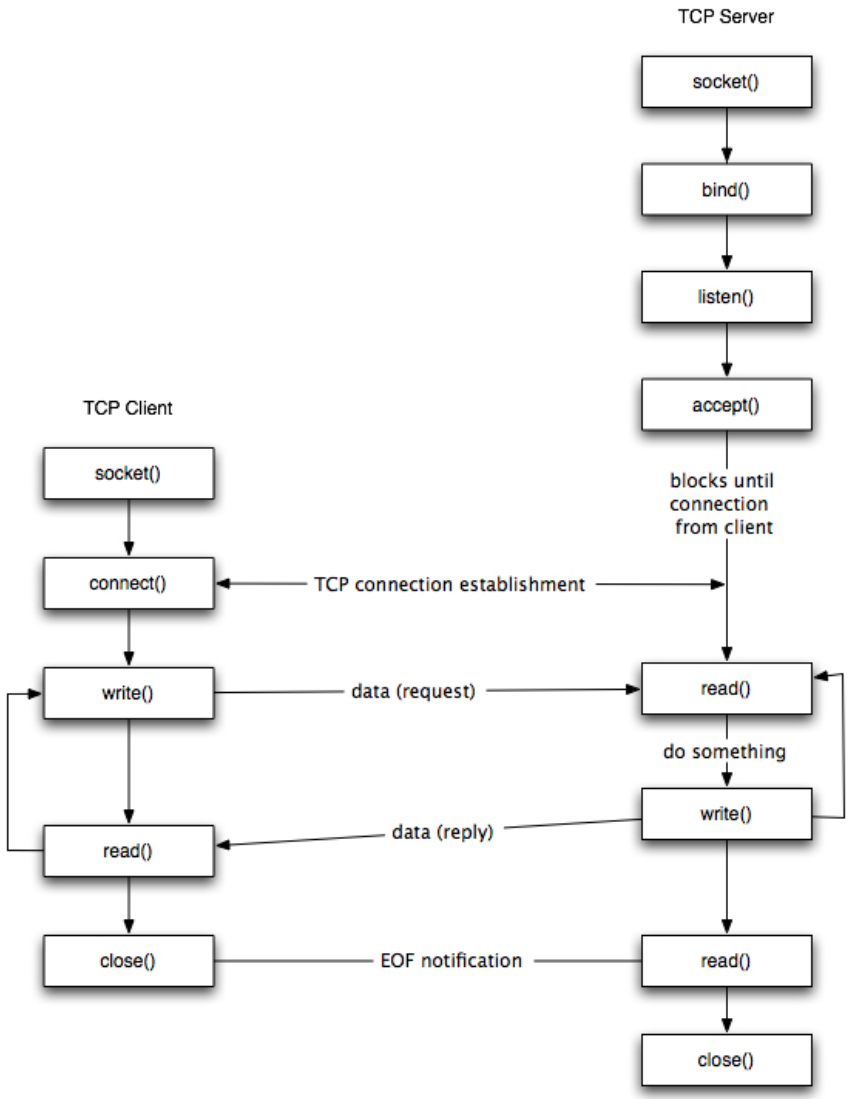
getpeername —after a connection arrives, obtain the remote machine's endpoint address from a socket

getsockopt —obtain the current options for a socket

setsockopt —change the options for a socket

Socket – General Operation

Lifecycle



Socket – Example

Server code

```
int main()
{
int server_socket_fd, client_conn_fd;
int client_addr_size;
struct sockaddr_in server_addr, client_addr;
int port_number;
char message_from_client[256];
char message_from_server_to_client[256];
int client_message_length, server_message_length;
// Set the port number
port_number = 1132;
// Create the socket for the server to listen on
server_socket_fd = socket( AF_INET,
SOCK_STREAM,
0
);
if (server_socket_fd < 0)
PrintError("ERROR opening socket");
// clear out the server address to make sure no problems
with binding
// if the address is clear then it won't think the address
has
// already been used by another socket
bzero( (char *) &server_addr, sizeof(server_addr));
// set the sockaddr_in struct appropriately
// note that this assumes IPv4
server_addr.sin_family = AF_INET;
server_addr.sin_addr.s_addr = INADDR_ANY;
server_addr.sin_port = htons(port_number);
// Bind the socket descriptor to the address
if (bind( server_socket_fd,
(struct sockaddr *) &server_addr,
sizeof(server_addr))
<0)
PrintError("binding to socket failed");
// Start the server listening on the socket
// limit the number of connections in the listen queue to 3
// (this is the backlog)
listen(server_socket_fd, 3);
while (1)
{
// clear out the client address to make sure no problems with
accepting
// on this address
// if the address is clear then it won't think the address has
// already been used by another socket
bzero( (char *) &client_addr, sizeof(client_addr));
```

Socket – Example

Server code

```
// accept a connection from a client
// on the open socket
client_addr_size = sizeof(client_addr);
client_conn_fd = accept( server_socket_fd,
(struct sockaddr *) &client_addr,
(socklen_t *) &client_addr_size
);
if (client_conn_fd < 0)
PrintError("the accept failed");
// Clear out the character array to store the client message
// to make sure there's no garbage in it
bzero(message_from_client, 256);
// read the message from the client
client_message_length = read( client_conn_fd,
message_from_client,
255
);
if (client_message_length < 0)
PrintError(" unable to read from socket" );
cout << " message is"  << message_from_client << endl;
```

```
// Now write a message back to the client
// Doing a little mixed mode C++ strings and char buffers
// just to show you how
string mystring;
mystring = " Server received from client, then echoed back to client:" ;
mystring += message_from_client;
bzero(message_from_server_to_client, 256);
mystring.copy(message_from_server_to_client, mystring.length() );
server_message_length = write( client_conn_fd,
(char *) message_from_server_to_client,
strlen(message_from_server_to_client)
);
if (server_message_length < 0)
PrintError(" unable to write to socket" );
sleep(5); // let the client close first, avoids socket address reuse issues
close(client_conn_fd);
} // end while (1)
close(server_socket_fd); // this isn't reached because we used while (1)
// but with a different while loop test condition
// this would be important
return 0;
}
```

Socket – Example

Client code

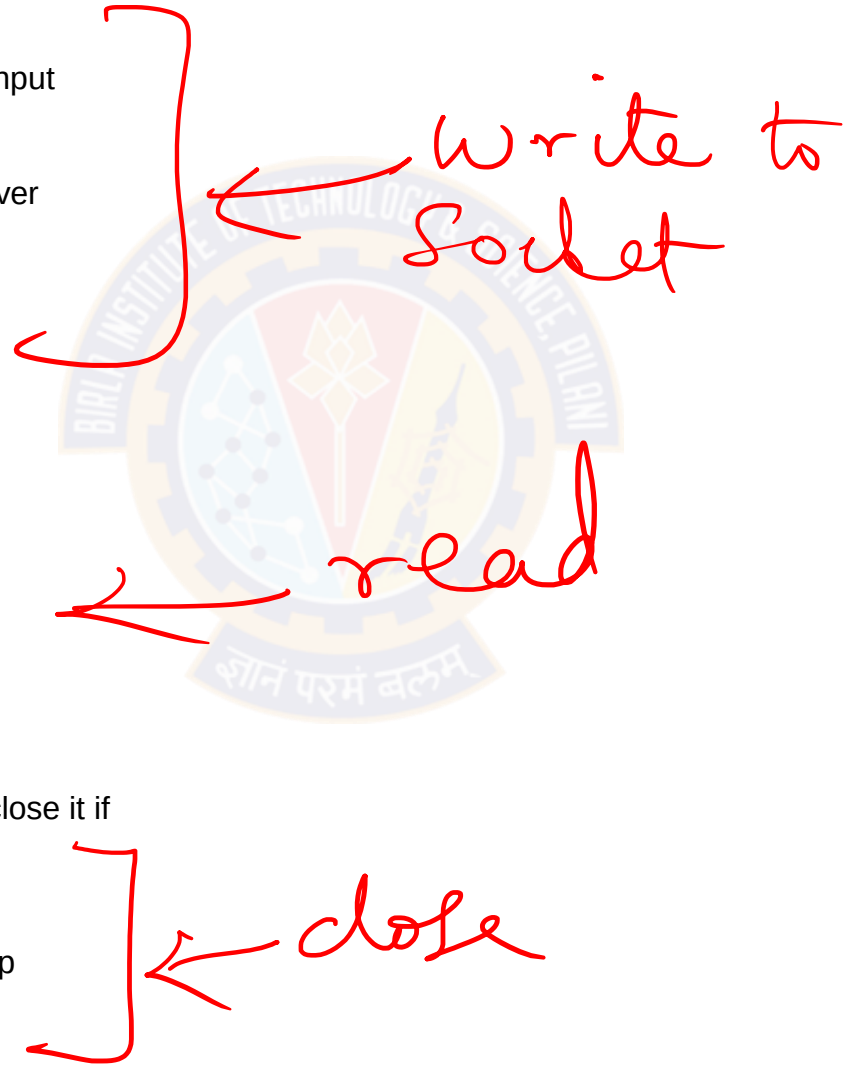
```
int main()
{
int socket_fd;
struct sockaddr_in serv_addr;
int port_number;
char * IP_address;
char client_message[256];
int message_result;
// Set the IP address (IPv4)
IP_address = new char [sizeof(“ 127.0.0.1” )];
strcpy(IP_address, “ 127.0.0.1” ); // could instead have
copied “ localhost”
 // Set the port number
port_number = 1132;
// Create the socket
socket_fd = socket( AF_INET,
SOCK_STREAM,
0
);
if (socket_fd < 0)
PrintError(“ ERROR opening socket” );
// clear out the server address to make sure no problems
with binding
bzero((char *) &serv_addr, sizeof(serv_addr));
 // set the sockaddr_in struct appropriately
serv_addr.sin_family = AF_INET;
serv_addr.sin_port = htons(port_number);
// Use the inet_pton function to convert the IP address to
// binary
if (inet_pton( AF_INET,
IP_address,
&serv_addr.sin_addr
)
< 0
)
PrintError(“ Unable to convert IP address to binary to put in
serv_addr” );
// Connect to the server
if (connect( socket_fd,
(struct sockaddr *)
&serv_addr,
sizeof(serv_addr)
)
<0
)
PrintError(“ unable to connect to server” );
 cout << “ Enter message to send to server: ” ;
bzero(client_message,256);
```

IPv4

Socket – Example

Client code

```
string mystring;
getline(cin, mystring); // read the line from standard input
cout << endl;
strcpy(client_message,mystring.c_str());
// Write the message to the socket to send to the server
message_result = write( socket_fd,
client_message,
strlen(client_message)
);
if (message_result < 0)
PrintError("unable to write to socket");
// Read the return message from the server
bzero(client_message,256);
message_result = read( socket_fd,
client_message,
255
);
if (message_result < 0)
PrintError("unable to read from socket");
cout << client_message << endl;
// close(socket_fd); // commented out because only close it if
you
// don't want to do more calling the server from
// a run of the client
delete [] IP_address; // return the memory to the heap
return 0;
}
```



Thank You!

In our next session:

Early Middleware Technologies