Computer Networks Lab
IT-553

Dept. of Comp Sc & Engg. (IT Block)
National Institute of Technology,
Durgapur

Objective

- To understand socket Preliminaries.
- To write, execute programs using Socket.
- To understand the use of client/server architecture in application development.
- To understand TCP and UDP based sockets and their differences.
- To understand Peer to Peer communication using socket.

SOCKET(s)

Socket (or *Communication Socket*) : Definition

- *Sockets* allow communication between two different processes on the same or different machines.
- it's a way to talk to other computers using standard Unix file descriptors.
- To a programmer, a socket looks and behaves much like a low-level file descriptor. This is because commands such as read() and write() work with sockets in the same way they do with files and pipes.

Socket : Utilities

A Socket is used in a client-server application framework. A server is a process that performs some functions on request from a client. The application-level protocols like FTP, SMTP, and POP3 make use of sockets to establish connection between client and server and then for exchanging data.

- Define an “end- point” for communication
- Initiate and accept a connection

Examples

- File transfer apps (FTP), Web browsers
- (HTTP), Email (SMTP/ POP3), etc…

Types of Sockets #1

- Different types of sockets :

- Stream socket :(*a. k.* a. connection- oriented socket)
  - It provides reliable, connected networking service
  - Error free; no out- of- order packets (uses TCP)
  - applications: telnet/ ssh, http, …

- Datagram socket :(*a. k.* a. connectionless socket)
  - It provides unreliable, best- effort networking service
  - Packets may be lost; may arrive out of order (uses UDP)
  - applications: streaming audio/ video (realplayer), …

Types of Sockets #2

Raw Sockets: Raw sockets are not intended for the general user; they have been provided mainly for those interested in developing new communication protocols, or for gaining access to some of the more cryptic facilities of an existing protocol.

Sequenced Packet Sockets: This interface is provided only as a part of the Network Systems (NS) socket abstraction, and is very important in most serious NS applications. Sequenced-packet sockets allow the user to manipulate the Sequence Packet Protocol (SPP) or Internet Datagram Protocol (IDP) headers on a packet or a group of packets, either by writing a prototype header along with whatever data is to be sent, or by specifying a default header to be used with all outgoing data, and allows the user to receive the headers on incoming packets.

This is obsoleted after invention of stream and datagram sockets.

Client and Server

(works with a **request-reply** protocol)

Client Process :
This is the process, which typically makes a request for information. After getting the response, this process may terminate or may do some other processing.
Example: Internet Browser works as a client application, which sends a request to the Web Server to get one HTML webpage.

Server Process
This is the process which takes a request from the clients. After getting a request from the client, this process will perform the required processing, gather the requested information, and send it to the requestor client. Once done, it becomes ready to serve another client. Server processes are always alert and ready to serve incoming requests.
Example: Web Server keeps waiting for requests from Internet Browsers and as soon as it gets any request from a browser, it picks up a requested HTML page and sends it back to that Browser.

Types of servers

Types of Server
There are two types of servers you can have:

Iterative Server: This is the simplest form of server where a server process serves one client and after completing the first request, it takes request from another client. Meanwhile, another client keeps waiting.

Concurrent Servers: This type of server runs multiple concurrent processes to serve many requests at a time because one process may take longer and another client cannot wait for so long. The simplest way to write a concurrent server under Unix is to *fork* a child process to handle each client separately.

Server –> High Level View

Create a socket

Bind the socket

Listen for connections

Accept new client connections

Read/write to client connections

Shutdown connection

Client –> High Level View

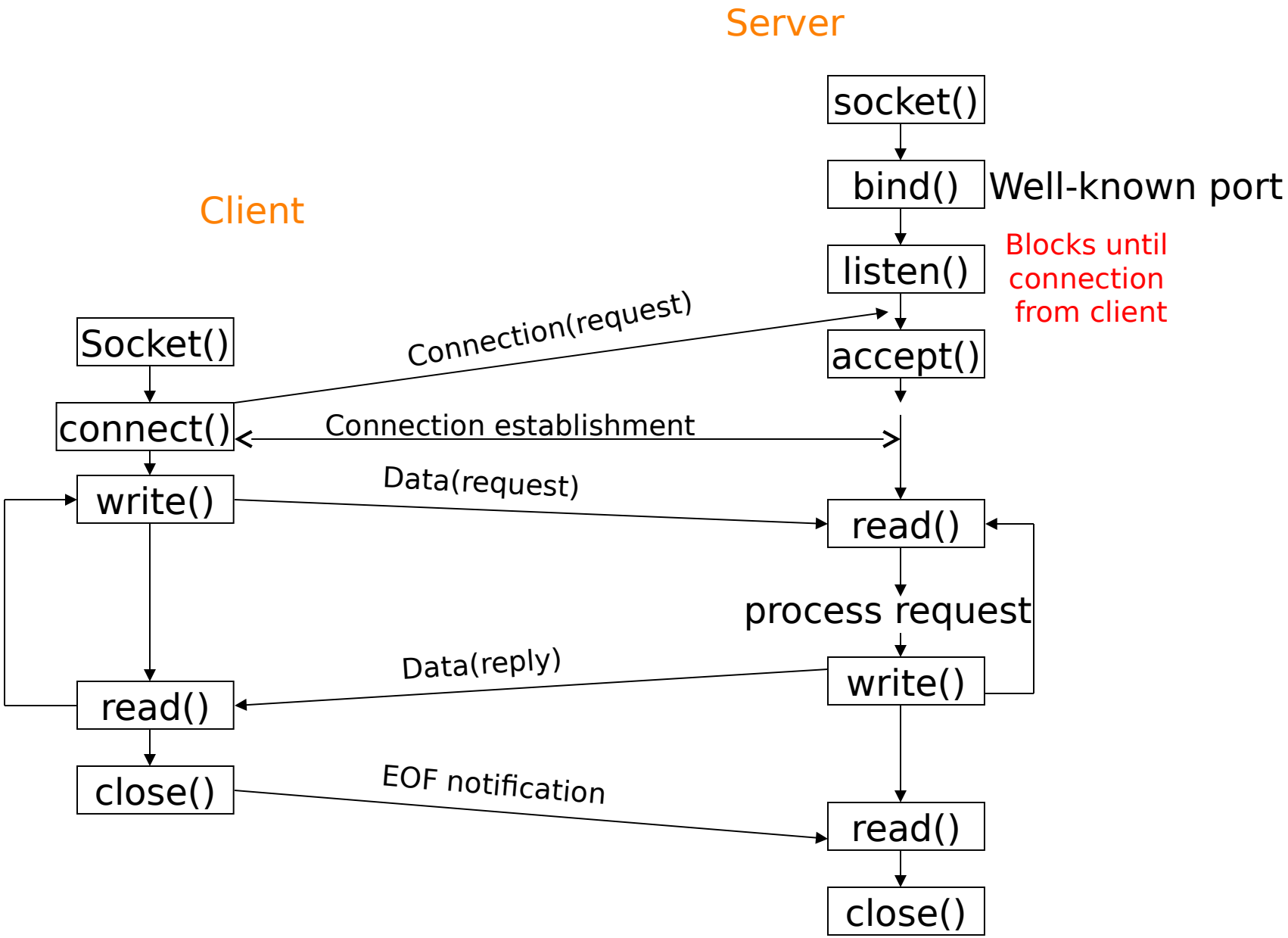
Create a socket

Setup the server address

Connect to the server

Read/write data

Shutdown connection


Server
socket()
bind()
Well-known port
listen()
Blocks until connection from client
accept()
Client
Socket()
connect()
Connection(request)
Connection establishment
write()
Data(request)
read()
process request
write()
Data(reply)
read()
close()
EOF notification
read()
close()

A Simple Server :

```
# server.py
import socket
import time
# create a socket object
serversocket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# get local machine name
host = socket.gethostname()
port = 9999
# bind to the port
serversocket.bind((host, port))
# queue up to 5 requests
serversocket.listen(5)
while True:
    # establish a connection
    clientsocket,addr = serversocket.accept()
    print("Got a connection from %s" % str(addr))
    currentTime = time.ctime(time.time()) + "\r\n"
    clientsocket.send(currentTime.encode('ascii'))
    clientsocket.close()
```

A Simple Client :

```
# client.py
import socket
# create a socket object
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# get local machine name
host = socket.gethostname()
port = 9999
# connection to hostname on the port.
s.connect((host, port))
# Receive no more than 1024 bytes
tm = s.recv(1024)
s.close()
print("The time got from the server is %s" % tm.decode('ascii'))
```

Assignments

(Must submit before next lab session)

1. Compare server and client sockets.
2. Compare DGRAM, STREAM sockets.
3. Implement a simple ‘hi-hello’ socket with suitable program(s) in python.