Analysis of TCP Communication Session

Course: Computer Network Lab (AV 341) **Experiment-4** **Date:** 20th February, 2018

Description: For this experiment, we need to create a communication session between two processes using Transmission Control Protocol (TCP). TCP is a *connection oriented* protocol which provides services including in-order delivery, flow-control, and congestion-control. To establish a connection, we need to use APIs such as *listen(), connect()*, and *accept()* in addition to *socket()* and *bind()*. The APIs *send()* and *recv()* are used to send and receive data in a TCP session. The sequence of API calls in both end of the connection is shown in Figure 1.

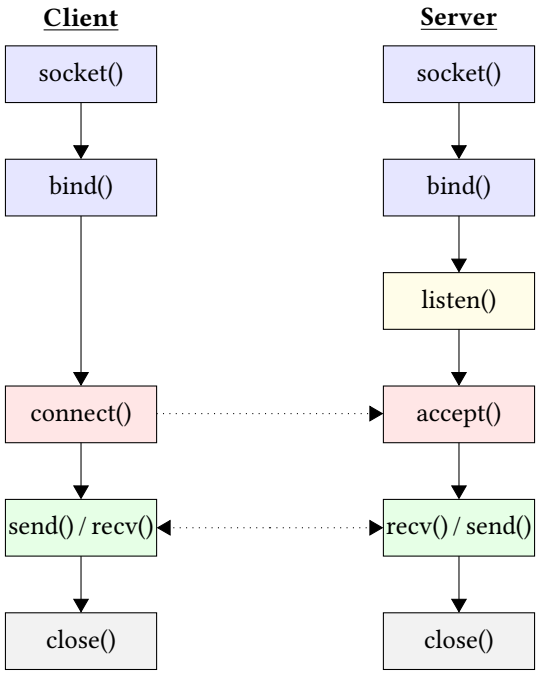

Figure 1: API call sequence in a TCP communication session.

The experiment can be realized using three phases:

Phase-1 Initiate a TCP session from client with an *echo* server.

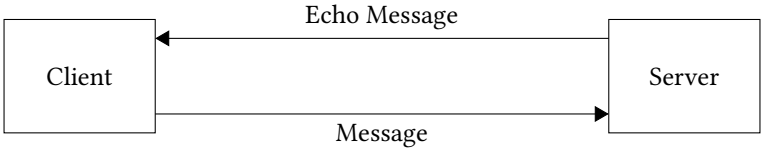

Figure 2: The set up for TCP communication session.

Phase-2 Send n packets consecutively from the client and check the number of *recv()* system calls, say k, required to receive n messages at the server, i.e., you need to observe whether $k < n, k = n$, or $k > n$.

Phase-3 Capture the entire communication session using Wireshark and study the sequence of messages transmitted between the processes in Phase-2.

Sample code for a TCP session between client and server processes is given in the following:

Server code

```c
#include<sys/socket.h>
#include<arpa/inet.h>
#include<stdio.h>
#include<string.h>

#define BUFFERSIZE 1024
#define PORT 5000

int main()
{
  int req_sock, conn_sock, bytes_recv, bytes_sent, bind_status;
  struct sockaddr_in s_server, s_client;
  int s_len = sizeof(s_server);
  char send_buf[BUFFERSIZE], recv_buf[BUFFERSIZE];

  // Creating requesting socket
  req_sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
  if(req_sock < 0)
    {
      printf("Socket creation failed. \n");
      return 1;
    }
  else
    {
      printf("Request socket creation successful with descriptor %d\n", req_sock);
    }

  // Bind the socket
  s_server.sin_family = AF_INET;
  s_server.sin_port = htons(PORT);
  s_server.sin_addr.s_addr = htonl(INADDR_ANY);

  bind_status = bind(req_sock, (struct sockaddr*)&s_server, sizeof(s_server));
  if(bind_status < 0)
    {
      printf("Socket binding failed.\n");
      return 1;
    }
  else
    {
      printf("Socket binding successful.\n");
    }

  // Enable req socket to accept connection requests
  listen(req_sock, 5);

  // Waiting for connection requests
  printf("Waiting for connection requests.\n");
  conn_sock = accept(req_sock, (struct sockaddr*)&s_client, &s_len);
  if(conn_sock == -1)
    {
      printf("Connection request rejected.\n");
      close(req_sock);
      return 1;
    }
  else
    {
      printf("Connection request accepted with socket %d\n", conn_sock);
    }

  bytes_recv = recv(conn_sock, recv_buf, sizeof(recv_buf), 0);
  recv_buf[bytes_recv] = '\0';

  printf("%d bytes received: %s\n", bytes_recv, recv_buf);
```

```
  strcpy(send_buf, recv_buf);
  bytes_sent = send(conn_sock, send_buf, strlen(send_buf), 0);
  printf("%d bytes sent: %s\n", bytes_sent, send_buf);

  bytes_recv = recv(conn_sock, recv_buf, sizeof(recv_buf), 0);
  printf("%d bytes received: %s\n", bytes_recv, recv_buf);


  close(req_sock);
  close(conn_sock);

  return 0;
}
```

Client code

```
#include<sys/socket.h>
#include<arpa/inet.h>
#include<stdio.h>
#include<string.h>

#define BUFFERSIZE 1024
#define SERVERADDR "127.0.0.1"
#define SERVERPORT 5000

int main()
{
  int sock, bytes_sent, bytes_recv, conn_status;
  struct sockaddr_in s_server;
  char send_buf[BUFFERSIZE], recv_buf[BUFFERSIZE];

  // Creating socket
  sock = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);
  if(sock < 0)
    {
      printf("Socket creation failed.\n");
      return 1;
    }
  else
    {
      printf("Socket creation successful with descriptor %d.\n", sock);
    }

  s_server.sin_family = AF_INET;
  s_server.sin_port = htons(SERVERPORT);
  inet_aton(SERVERADDR, &s_server.sin_addr);

  conn_status = connect(sock, (struct sockaddr*)&s_server, sizeof(s_server));
  if(conn_status == -1)
    {
      printf("Connection to server failed.\n");
      close(sock);
      return 1;
    }
  else
    {
      printf("Connected to server.\n");
    }

  // Reading message from keyboard and send
  printf("Enter the message: ");
  scanf("%s", send_buf);

  bytes_sent = send(sock, send_buf, strlen(send_buf), 0);
  printf("%d bytes sent: %s\n", bytes_sent, send_buf);
```

```
    // Waiting for receiving data
    bytes_recv = recv(sock, recv_buf, sizeof(recv_buf), 0);
    recv_buf[bytes_recv] = '\0';
    printf("%d bytes received: %s\n", bytes_recv, recv_buf);

    close(sock);

    return 0;
}
```