Socket Programming Assignment

EE4204/EE4204E/TEE4204

Sivaraman Vignesh (PhD Student)
Assoc. Prof. Mohan Gurusamy
Electrical and Computer Engineering Department
National University of Singapore

Sockets

- Application Programming Interface (API)
- Interface between an application and network
- Application creates a socket
- The interface defines the following operations
  - Create a socket
  - Attach a socket to the network
  - Send/receive messages through a socket
  - Close a socket

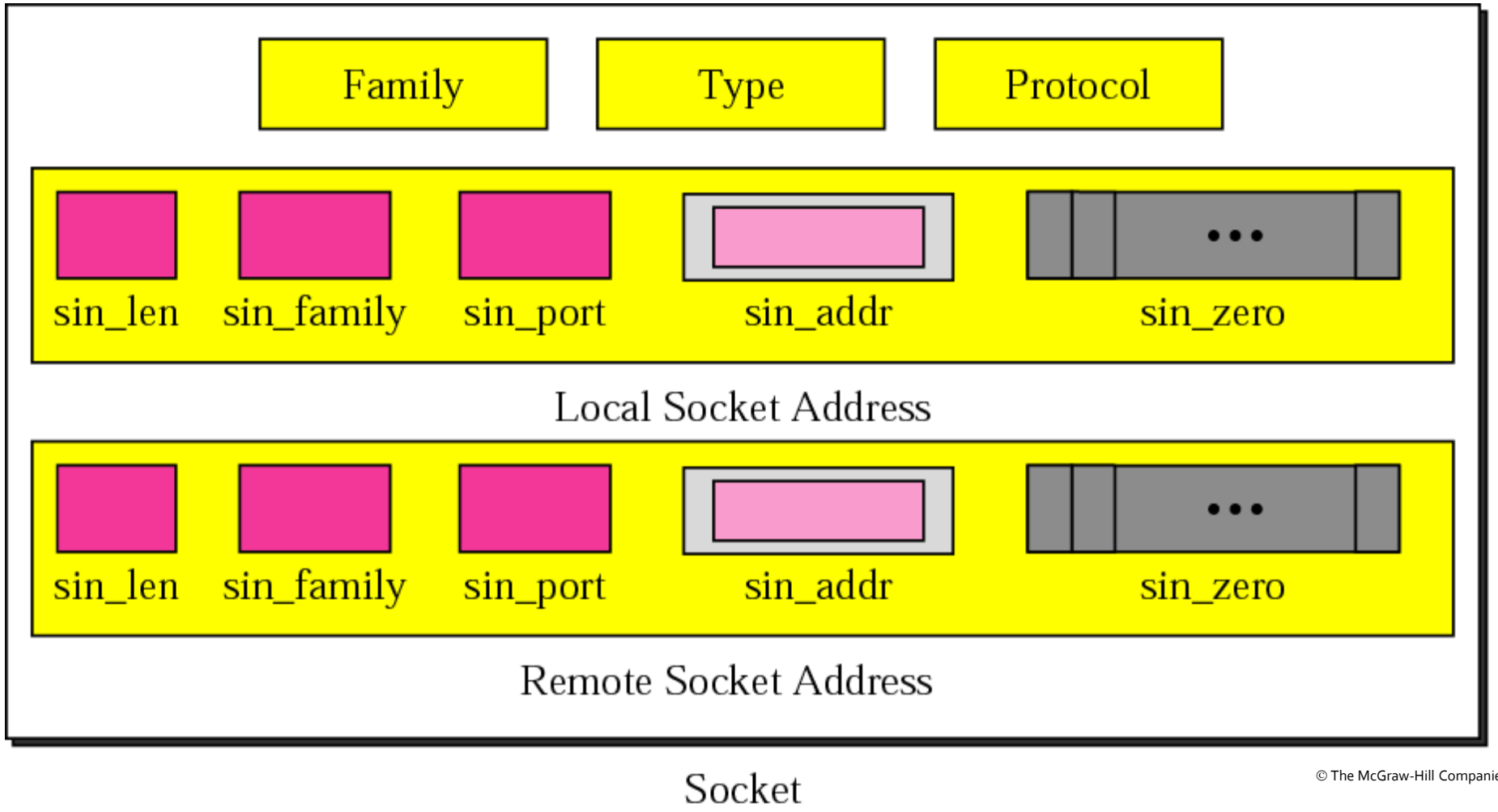
Socket – Family and Type

- Socket Family
  - PF_INET : Internet Family
  - PF_UNIX: Unix Pipe facility
- Socket Type
  - SOCK_STREAM: byte stream such as TCP
  - SOCK_DGRAM: message oriented service such as UDP

Example 1

- Develop a socket program in UNIX/Linux that uses (i) TCP as the transport protocol and (ii) UDP as the transport protocol for transferring a short message between a client and server. The client sends a string (input by the user) to the server and the server prints the string on the screen after receiving it.

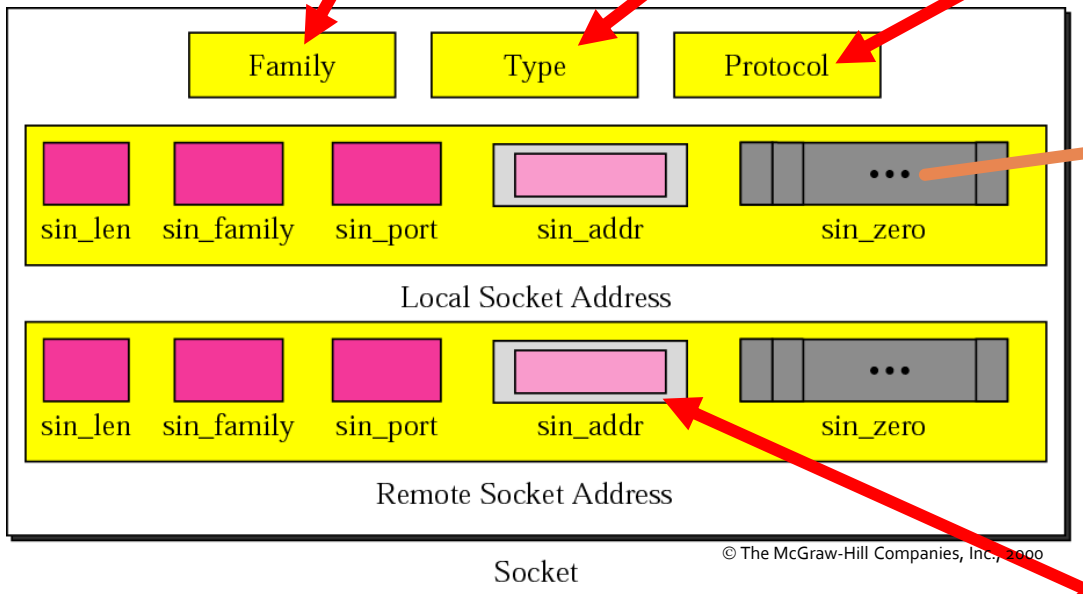
Socket structure



UDP Client

- Create a socket

```
28      sockfd = socket(AF_INET, SOCK_DGRAM, 0);
```



Chosen by the system

```
69      sendto(sockfd, &sends, strlen(sends), 0, addr, addrlen);
```

UDP Client

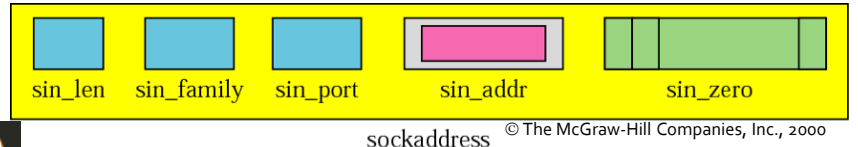
- Get the server address

Has information like name, aliases, address type etc

```
14      struct hostent *sh;
```

```
23      if ((sh=gethostbyname(argv[1]))==NULL)
```

```
35      addrs = (struct in_addr **)sh->h_addr_list;
```




```
49      ser_addr.sin_family = AF_INET;
50      ser_addr.sin_port = htons(MYUDP_PORT);
51      memcpy(&(ser_addr.sin_addr.s_addr), *addrs, sizeof(struct in_addr));
52      bzero(&(ser_addr.sin_zero), 8);
```

- Send data to the server

```
69      sendto(sockfd, &sends, strlen(sends), 0, addr, addrlen);
```

Server address(remote socket)

UDP Server

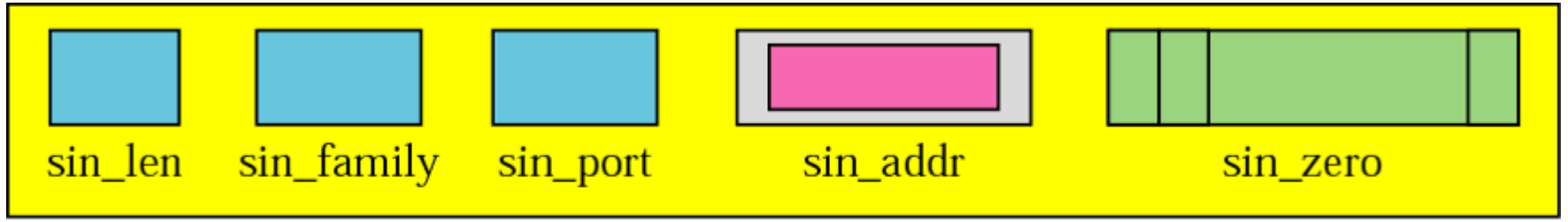
- Creating socket is same as the client

```
13      if ((sockfd = socket(AF_INET, SOCK_DGRAM, 0)) == -1) {
```

- Values of the other tuples

```
18      my_addr.sin_family = AF_INET;
19      my_addr.sin_port = htons(MYUDP_PORT);
20      my_addr.sin_addr.s_addr = INADDR_ANY;
21      bzero(&(my_addr.sin_zero), 8);
```

In case the system has multiple IPs



sockaddress

UDP Server

- Bind function: Associates the socket with the given local address

```
22      if (bind(sockfd, (struct sockaddr *) &my_addr, sizeof(struct sockaddr)) == -1)
```

- Receive from client

```
42      if ((n=recvfrom(sockfd, &recvs, MAXSIZE, 0, (struct sockaddr *)&addr, &len)) == -1)
```

Client address (remote socket)

- Close the socket (both at server and client)

```
30        close(sockfd);
```

headsock.h (Ex 1)

```
14 #define MYTCP_PORT 4950
15 #define MYUDP_PORT 5350
16 #define MAXSIZE 50
```

Example 1 (TCP): Client

Server address (from command line)

```
21      sh = gethostbyname(argv[1]);
```

```
40      addrs = (struct in_addr **)sh->h_addr_list;                    //get the se
41      sockfd = socket(AF_INET, SOCK_STREAM, 0);                       //create t
42      if (sockfd <0)
43      {
44          printf("error in socket");
45          exit(1);
46      }
47      ser_addr.sin_family = AF_INET;
48      ser_addr.sin_port = htons(MYTCP_PORT);
49      memcpy(&(ser_addr.sin_addr.s_addr), *addrs, sizeof(struct in_addr));
50      bzero(&(ser_addr.sin_zero), 8);
51      ret = connect(sockfd, (struct sockaddr *)&ser_addr, sizeof(struct sockaddr));
```

Creating a socket and connecting to it

Example 1 (TCP): Client

```
64 void str_cli(FILE *fp, int sockfd)
65 {
66     char sends[MAXSIZE];
67
68     printf("Please input a string (less than 50 character):\n");
69     if (fgets(sends, MAXSIZE, fp) == NULL) {
70         printf("error input\n");
71     }
72     send(sockfd, sends, strlen(sends), 0);      //send the string
73     printf("send out!!\n");
74 }
```

Client application to send a string to the server

Example 1 (TCP): Server

```
20      sockfd = socket(AF_INET, SOCK_STREAM, 0);           //create socket
21      if (sockfd <0)
22      {
23          printf("error in socket!");
24          exit(1);
25      }
26
27      my_addr.sin_family = AF_INET;
28      my_addr.sin_port = htons(MYTCP_PORT);               //port number
29      my_addr.sin_addr.s_addr = htonl(INADDR_ANY);
30      bzero(&(my_addr.sin_zero), 8);
31      ret = bind(sockfd, (struct sockaddr *) &my_addr, sizeof(struct sockaddr));
```

Create a socket at the server

Example 1 (TCP): Server

```
38      ret = listen(sockfd, BACKLOG);
```

Informs the OS, that socket is ready for requests

Connect to an incoming request

```
49        con_fd = accept(sockfd, (struct sockaddr *)&their_addr, &sin_size);
```

```
74      if ((n= recv(sockfd, &recvs, MAXSIZE, 0))==-1)
75      {
76          printf("receiving error!\n");
77          return;
78      }
79      recvs[n] = '\0';
80      printf("the received string:\n%s\n", recvs);
```

Server application to receive a string from a client

Example 2

- Develop a TCP-based client-server socket program for transferring a large message. Here, the message transmitted from the client to server is read from a large file. The entire message is sent by the client as a single data-unit. After receiving the file, the server sends an ACK message to the receiver. Verify if the file has been sent completely and correctly by comparing the received file with the original file ("*diff*" command could be used). Measure the message transfer time and throughput.

Example 2: Client

- The TCP connection process is same as Example 1

```
62      if((fp = fopen ("myfile.txt","r+t")) == NULL)
```

Open the file in read mode

```
89      fseek (fp , 0 , SEEK_END);
90      *len= lsize = ftell (fp);
91      rewind (fp);
92      printf("The file length is %d bytes\n", (int)lsize);
```

Determine the length of the file

```
96      fread (sends.data,1,lsize,fp);
```

Read the file into the buffer

```
102       sends.len = lsize;
103       sends.num = 0;
104       n=send(sockfd, &sends, (sends.len+HEADLEN), 0);
```

Send the file to the server

Example 2: Client

```
110     if ((n=recv(sockfd, &acks, 2, 0)) == -1) {
111         printf("error receiving data\n");
112         exit(1);
113     }
```

Wait for the ack

```
114     if ((acks.len == 0) && (acks.num == 1))          //if it is ACK
115     {
116         gettimeofday(&recvt, NULL);
117         tv_sub(&recvt, &sendt);
118         time_inv += (recvt.tv_sec)*1000.0 + (recvt.tv_usec)/1000.0;
119         return(time_inv);
120     }
```

Check if it is an Ack and calculate the time taken

Example 2: Server

```
76      while(ci < lsize)
77      {
78          if ((n= recv(sockfd, &recvs, MAXSIZE, 0))==-1)
79          {
80              printf("receiving error!\n");
81              return;
82          }
83          else printf("%d data received\n", n);
84          if (ci == 0) {
85              lsize = recvs.len;
86              memcpy(buf, recvs.data, (n-HEADLEN));
87              ci += n-HEADLEN;
88          }
89          else {
90              memcpy((buf+ci), &recvs, n);
91              ci += n;
92          }
93      }
```

Receive the whole file from client

Example 2: Server

```
94      ack.len = 0;
95      ack.num = 1;
96  //  memcpy(buf, recvs.data, re
97      send(sockfd, &ack, 2, 0);
```

Send ack to client after receiving the whole file

```
106     fwrite (buf , 1 , lsize, fp);
107     fclose(fp);
```

Save the file from buffer to a file

headsock.h (Ex 2)

```
16 #define MYTCP_PORT 4950
17 #define MYUDP_PORT 5350
18 #define MAXSIZE 30008
19 #define MAXLEN 30000
20 #define MAXINT 0x7fffffff
21 #define BUFSIZE 31000
22 #define N 1
23 #define HEADLEN 8
```

```
32 struct ack_so
33 {
34 uint8_t num;
35 uint8_t len;
36 };
```

```
25 struct pack_so
26 {
27 uint32_t num;
28 uint32_t len;
29 char data[MAXLEN];
30 };
```

Example 3

- Develop a TCP-based client-server socket program for transferring a large message. Here, the message transmitted from the client to server is read from a large file. The message is split into short data-units which are sent one by one without waiting for any acknowledgement between transmissions of two successive data-units. Verify if the file has been sent completely and correctly by comparing the received file with the original file. Measure the message transfer time and throughput for various sizes of data-units.

Example 3: Client sending file in data units

```
105     while(ci<= lsize)
106     {
107         if ((lsize+1-ci) <= DATALEN)
108             slen = lsize+1-ci;
109         else
110             slen = DATALEN;
111         memcpy(sends, (buf+ci), slen);
112         n = send(sockfd, &sends, slen, 0);
113         if(n == -1) {
114             printf("send error!");
115             exit(1);
116         }
117         ci += slen;
118     }
```

Sending file in data units of size DATALEN

Client waits for Ack

```
83      struct ack_so ack;
```

```
119     if ((n= recv(sockfd, &ack, 2, 0))==-1)
120     {
121         printf("error when receiving\n");
122         exit(1);
123     }
124     if (ack.num != 1|| ack.len != 0)
125         printf("error in transmission\n");
```

Example 3: Server receiving file in data units

```
82      while(!end)
83      {
84          if ((n= recv(sockfd, &recvs, DATALEN, 0))==-1)
85          {
86              printf("error when receiving\n");
87              exit(1);
88          }
89          if (recvs[n-1] == '\0')
90          {
91              end = 1;
92              n --;
93          }
94          memcpy((buf+lseek), recvs, n);
95          lseek += n;
96      }
```

Receiving file in data units of size DATALEN

Server sending Ack and saving the file

```
72      struct ack_so ack;
97      ack.num = 1;
98      ack.len = 0;
99      if ((n = send(sockfd, &ack, 2, 0))==-1)
```

Send ack to client

```
109     fwrite (buf , 1 , lseek , fp);
110     fclose(fp);
```

Save the file from buffer to a file

headsock.h (Ex 3)

```
17 #define MYTCP_PORT 4950
18 #define MYUDP_PORT 5350
19 #define DATALEN 500
20 #define BUFSIZE 60000
21 #define PACKLEN 508
22 #define HEADLEN 8
```

```
31 struct ack_so
32 {
33 uint8_t num;
34 uint8_t len;
35 };
```

```
24 struct pack_so
25 {
26 uint32_t num;
27 uint32_t len;
28 char data[DATALEN];
29 };
```

Sample Assignment problem

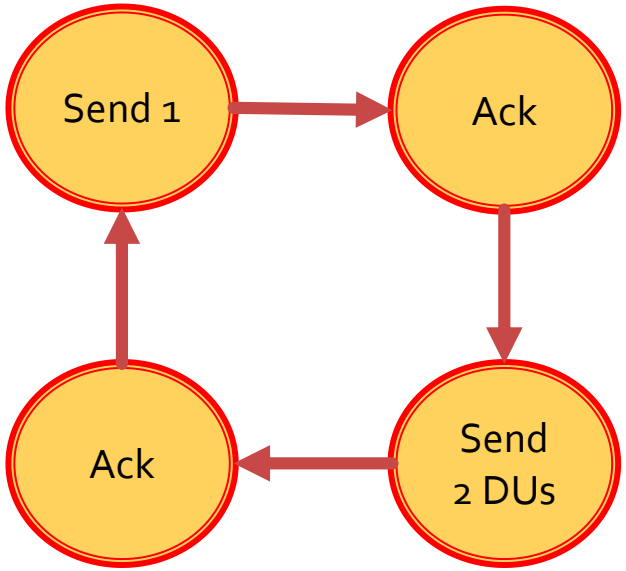
- Develop a UDP-based client-server socket program for transferring a large message. Here, the message transmitted from the client to server is read from a large file. The message is split into short data-units (DUs) which are sent and acknowledged in alternating batches of size 1 and 2 DUs. The sender sends one DU, waits for an acknowledgment (ACK); sends two DUs, waits for an acknowledgement; and repeats the above procedure until the entire file is sent. The receiver sends an ACK after receiving a DU; sends next ACK after receiving two DUs; and repeats the above procedure.

Sample Assignment problem (contd.)

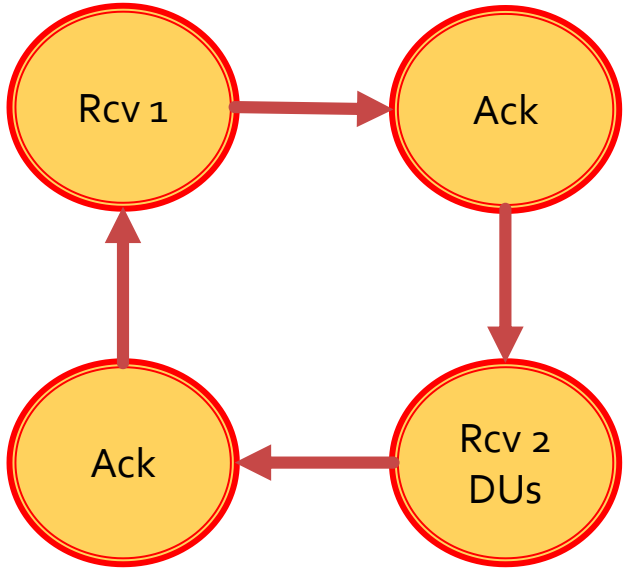
- Verify if the file has been sent completely and correctly by comparing the received file with the original file. Measure the message transfer time and throughput for various sizes of data-units and compare it with the stop-and-wait protocol where the batch size is always fixed to be 1. Choose appropriate values for "data unit size" and measure the performance. Repeat the experiment several times and plot the average values in a report with a brief description of results, assumptions made, etc.

Sample Assignment problem (contd.)

- Send a file from client to server using UDP.
- Server acks the data units in alternating batches of 1 and 2 data units (Dus).
- Compare the performance with Stop and Wait.



Client states



Server states

Connectionless iterative communication using UDP