



دانشگاه صنعتی اصفهان

دانشکده برق و کامپیوتر

آزمایشگاه سیستم عامل

جلسه ششم

برنامه نویسی socket

علی فانیان

زینب زالی

تابستان ۱۳۹۸



Socket چیست؟

- **socket** یکی از راههای ایجاد ارتباط بین دو فرایند است که اغلب از طریق **IP Address** و **Port Number** صورت می گیرد.
- معمولاً هنگامی از این روش استفاده می شود که دو فرایند ارتباط والد/فرزند نداشته باشند و یا دو فرایند قصد ارتباط روی شبکه را داشته باشند.
- سوکت ها **descriptor**هایی هستند که جهت ارتباط استفاده می شوند و ارسال پیام بین دو فرایند از طریق **read(send)** و **write(receive)** انجام می شود (مانند یک **file descriptor** که خواندن و نوشتن از طریق آن با توابع **read** و **write** انجام می گیرد). **socket** های تعریف شده در **POSIX** انواع متعددی دارند که از جمله پرکاربردترین آنها سوکت های **TCP, UDP** و **UNIX** می باشند.
- نوع سوکت استفاده شده، سرعت و کارایی آن را تعیین می کند. **UDP** و **TCP** پروتکل های مختلف لایه انتقال در شبکه هستند که جهت ارتباط بین دو فرایند از طریق شبکه استفاده می شوند. در مقابل سوکت نوع **UNIX** برای ارتباط بین دو فرایندی که روی یک ماشین قرار دارند استفاده می شود که سرعت بالاتری نسبت به دو سوکت قبلی دارد (سرعتی قابل قیاس با **PIPE**). در ادامه برنامه نویسی انواع **socket** های **TCP** و **UDP** شرح داده می شود.

مدل TCP

ارتباط بین دو فرایند با استفاده از **TCP** از پروتکل **TCP** در شبکه تبعیت می کند. در هر برنامه نویسی **TCP**، دو کد متفاوت **سرور** و **کلاینت** نیاز است. توابع موجود در سمت **سرور** اندکی با توابع موجود در سمت **کلاینت** متفاوت است. در ارتباط **TCP**، **سرور** فرایندی است که روی آدرس و شماره پورت مشخصی منتظر اتصال پروسس های **کلاینت** می ماند. هر فرایند **کلاینت** با آگاهی از آدرس **سرور**، به او متصل می شود. با اتصال هر فرایند **کلاینت** به **سرور**، ارتباط آن دو از طریق سوکت دنبال می شود.

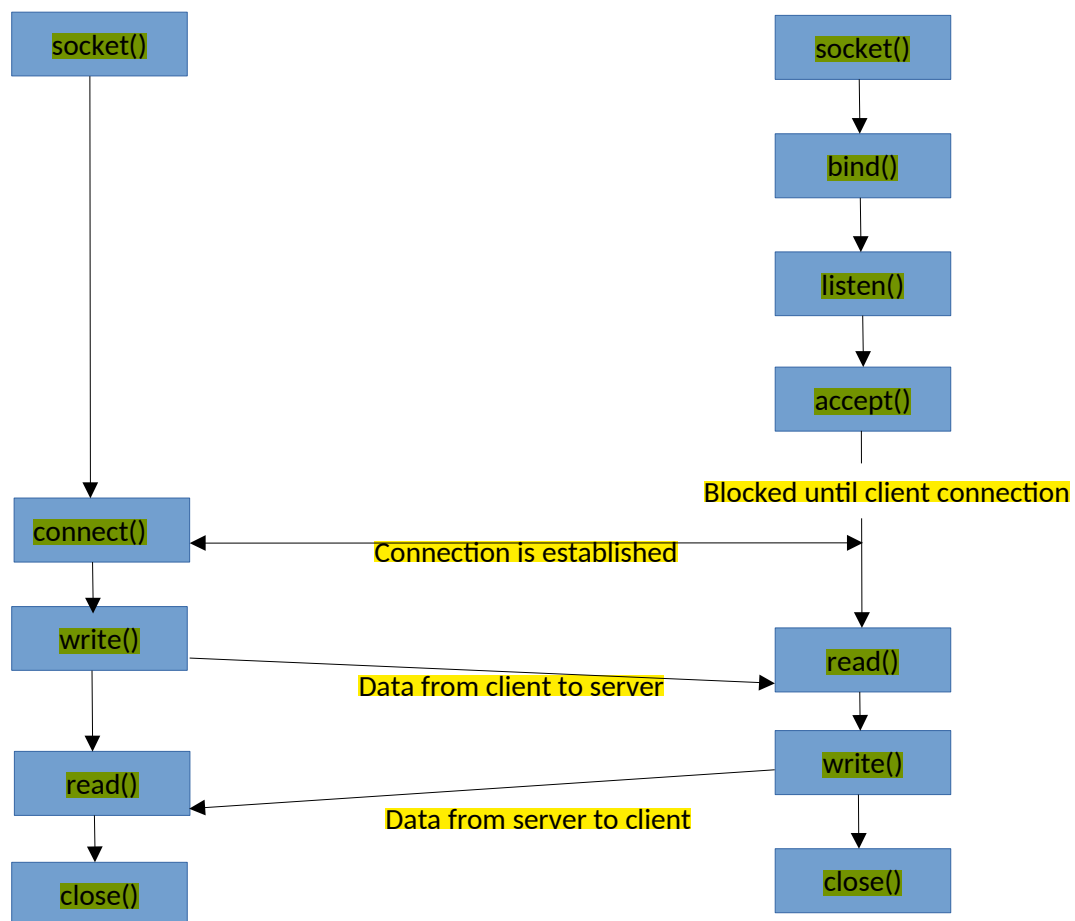
شکل ۱ مراحل مختلف ایجاد ارتباط **TCP** را نشان می دهد که به شرح زیر است:

- اولین قدم در دو طرف ارتباط، ساخت یک سوکت از نوع **SOCK_STREAM** است.
- **سرور** باید آدرس (آدرس **IP** و شماره **port**) خود را مشخص کند تا اتصال **کلاینت**ها به آن آدرس میسر شود. این کار از طریق تابع **bind** انجام می شود. در این مرحله **bind** کردن آدرس به سوکت **کلاینت** در کد **کلاینت** ضروری نیست (هرچند امکان پذیر است). اگر سوکتی مستقیماً توسط برنامه نویس **bind** نشود از آدرس **local** یا پیش فرض کامپیوتر موردنظر و یک شماره پورت آزاد دلخواه استفاده خواهد کرد. بنابراین اگر برای شما مهم هست که چه شماره پورتی استفاده می شود حتماً سوکت موردنظر را باید **bind** کنید. در اینجا هم چون آدرس و شماره پورت **سرور** در برنامه **کلاینت** مورد نیاز است باید آن را در **سرور** **bind** کرد.
- سپس **سرور** روی سوکتی که ساخته است اصطلاحاً شروع به گوش دادن (**listen**) می کند یعنی اعلام می کند که آماده درخواست اتصالهای **کلاینت**ها است. با فراخوانی این تابع **سرور** آماده می شود که اتصالات **کلاینت**ها را دریافت کند. این اتصالات تا شروع رسیدگی به آنها که توسط تابع **accept** انجام می گیرد در یک صف نگهداری می شود. طول این صف را می توان در آرگومان ورودی دوم تابع **listen** مشخص کرد.
- در طرف مقابل، **کلاینت** با تابع **connect** به آدرسی که برای **سرور** مشخص شده متصل می شود.



- ممکن است کلاینت‌های متعددی به سرور متصل شوند. تابع `accept` در سرور با هربار فراخوانی، منتظر اتصال یکی از کلاینت‌ها می‌ماند و به محض اتصال یک کلاینت، یک سوکت جدید برای آن کلاینت `return` می‌کند. از این پس ارتباط بین سرور و هر کلاینت از طریق این سوکت انجام می‌شود. پس در برنامه‌نویسی سرور یک سوکت برای سرور نیاز داریم که در ابتدا تعریف شد و سپس به ازای هر اتصال کلاینت یک سوکت جدید برای کلاینت مورد نظر خواهیم داشت.
 - اگر کلاینت به درستی به سرور متصل شود تابع `connect` باید بدون مشکل اجرا شده صفر برگرداند در غیر این صورت یک عدد منفی برمی‌گرداند.
 - از این پس ارسال و دریافت پیام بین کلاینت و سرور از طریق توابع `read` و `write` قابل انجام است. برای استفاده از این توابع، فرآیند کلاینت از سوکت خود و فرآیند سرور از سوکت دوم بازگشتی از تابع `accept` استفاده می‌کند.
- دقت کنید که اگر `connect` در کد کلاینت قبل از اجرای `listen` در کد سرور، اجرا شود کلاینت به سرور متصل نخواهد شد زیرا سروری هنوز وجود ندارد. پس بدیهی است که کد سرور باید در حال اجرا باشد تا بتوان کد کلاینت را با موفقیت اجرا کرد.

در ادامه کدهای نمونه کلاینت و سرور در شکل های ۲ و ۳ آمده است.



شکل ۱- مراحل ارتباط کلاینت و سرور TCP



```
/*
example of TCP client, the client frequently sends messages to the server
*/

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

#define SERVER_PORT 6000

int main(){
    int client_socket;
    char buffer [256];

    //making the server address record with a recognized server IP and port
    struct sockaddr_in server;
    server.sin_family=AF_INET;
    server.sin_port=htons(SERVER_PORT);
    server.sin_addr.s_addr=inet_addr("127.0.0.1");

    //making socket family = AF_INET, type = SOCK_STREAM , protocol = TCP
    client_socket = socket ( AF_INET , SOCK_STREAM , IPPROTO_TCP );

    //connecting to the server
    if (connect ( client_socket , (struct sockaddr * ) &server , sizeof(server))==0)
        printf("Client is connected to the server!\n");
    else
        printf("Error in connecting to the server\n");
    while(1) {
        bzero (buffer ,256);
        fgets(buffer, 256,stdin);
        write(client_socket , buffer , 256);
        printf("msg is sent to the server!\n");
    }
    return 0;
}
```

شکل ۲- کد کلاینت TCP



```
/*  
example of TCP server  
the server listens to incoming clients and lets maximum 5 client requests are  
queued before acception. Then the server accepts one client socket and frequently  
receives messages from it.  
*/  
#include <stdio.h>  
#include <string.h>  
#include <unistd.h>  
#include <sys/types.h>  
#include <sys/socket.h>  
#include <arpa/inet.h>  
#define SERVER_PORT 6000  
  
int main(){  
    char buffer [256];  
    int server_socket , client_socket ;  
    // server_address = explicit address of server  
    // client_address = client information  
    struct sockaddr_in server_address , client_address ;  
  
    //filling the server address record  
    server_address.sin_family=AF_INET; //IPv4  
    server_address.sin_port=htons(SERVER_PORT);  
    server_address.sin_addr.s_addr=inet_addr("127.0.0.1");  
    //server_address.sin_addr.s_addr=INADDR_ANY;  
  
    //making socket family = AF_INET, type = SOCK_STREAM , protocol = TCP  
    server_socket = socket ( AF_INET , SOCK_STREAM , IPPROTO_TCP );  
  
    //binding socket to the server address  
    bind ( server_socket , (struct sockaddr * ) & server_address ,  
    sizeof(server_address)) ;  
  
    //listening to incoming requests from clients with backlog of 5 clients  
    listen (server_socket , 5);  
    int clientsize=sizeof(client_address);  
    if ((client_socket = accept ( server_socket , (struct sockaddr * ) &  
client_address , &(clientsize)))>=0)  
        printf("A connection from a client is recieved\n");  
    else  
        printf("Error in accepting the connection from the client\n ");  
    while(1)  
    {  
        bzero(buffer,256);  
        read(client_socket , buffer , 256) ;  
        printf ("%s\n", buffer);  
    }  
    return 0;  
}
```

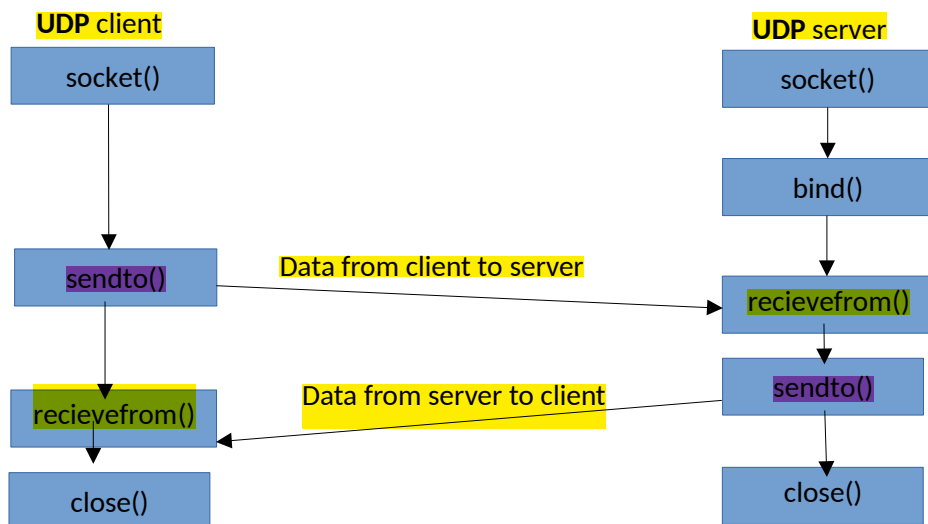
شکل ۳- کد سرور TCP



مدل UDP

اگر بخواهیم ارتباط بین دو فرآیند مبتنی بر پروتکل UDP در شبکه باشد، از سوکت نوع UDP استفاده می‌کنیم. در این ارتباط، برقراری یک جلسه ارتباط اولیه مانند TCP وجود ندارد و ارسال هر پیام بین هر دو فرآیند به صورت مستقل انجام می‌شود. مطابق شکل ۵، در ارتباط UDP هم می‌توان دو طرف server و client را تصور کرد. اما در اینجا کلاینت و سرور با مفهوم و روش ارتباطی که در TCP داشتیم وجود ندارد. لفظ کلاینت صرفاً جهت مشخص کردن فرآیندی که ارتباط را شروع می‌کند، یعنی اولین پیام را می‌فرستد استفاده می‌شود، همچنین لفظ سرور صرفاً جهت مشخص کردن فرآیندی که اولین پیام را از یک فرآیند مشخص دریافت می‌کند استفاده می‌شود. فرآیند کلاینت باید آدرس سرور را برای فرستادن اولین پیام بداند و فرآیند سرور با دریافت اولین پیام از یک کلاینت می‌تواند با استفاده از آدرس کلاینت که از طریق دریافت پیام بدست آمده، ارتباط را با او ادامه دهد. مراحل کار مطابق با شکل ۴ به صورت زیر است:

- در قدم اول هر دو فرآیند، یک سوکت از نوع Datagram می‌سازند.
 - اغلب ارسال و دریافت پیام در UDP، توسط توابع `sendto` و `recvfrom` انجام می‌شود. تابع `sendto` برای ارسال پیام، نیاز به آدرس فرآیند گیرنده دارد. پس آرگومان موردنظر باید قبلاً به درستی پر شده باشد. اما تابع `recvfrom` از یک آدرس نامشخص پیام دریافت می‌کند که پس از دریافت پیام، اطلاعات این آدرس در آرگومان پنجم و ششم تابع، قابل بازیابی است.
 - از آنجایی که در مدل کلاینت-سرور، کلاینت اولین پیام را می‌فرستد، نیاز به آدرس فرآیندی که پیام را برای او می‌فرستد یعنی سرور دارد. بنابراین در UDP نیز مانند TCP، برنامه سرور باید سوکت ساخته شده را به آدرس مشخصی `bind` کند. در صورتی که `bind` کردن آدرس کلاینت به سوکتش در کد کلاینت ضروری نیست. اما سرور مطابق مرحله بعد، می‌تواند آدرس کلاینت را بدست آورد.
 - همان‌طور که بیان شد، پس از اینکه یک پیام از سمت فرآیندی دریافت شد، آدرس آن فرآیند توسط آرگومان‌های `recvfrom` قابل بازیابی است. بنابراین سرور، با استفاده از این آدرس می‌تواند پیامی را برای فرآیند کلاینت (فرستنده) ارسال کند.
- کدهای مربوط به کلاینت و سرور UDP در شکل ۵ و ۶ آمده است.



شکل ۴: مراحل ارتباط سرور و کلاینت UDP



```
// Client side implementation of UDP client-server model

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>

int main() {
    int client_socket;
    char buffer[256];
    char hello_msg[256];
    struct sockaddr_in server_address, client_address;

    // Creating a socket file descriptor
    if ( (client_socket = socket(AF_INET, SOCK_DGRAM, 0)) < 0 ){
        printf("error: socket creation failed\n");
        return -1;
    }

    //filling the server address record
    server_address.sin_family = AF_INET;
    server_address.sin_port = htons(6000);
    server_address.sin_addr.s_addr=inet_addr("127.0.0.1");

    sprintf(hello_msg, "Hello from client");

    //send a message from the known server address
    sendto(client_socket, (const char *)hello_msg, strlen(hello_msg), MSG_CONFIRM,
    (const struct sockaddr *) &server_address, sizeof(server_address));
    printf("Hello message is sent.\n");

    int n, server_address_len;

    //receive a message from the known server address
    n = recvfrom(client_socket, (char *)buffer, 255, MSG_WAITALL, (struct sockaddr
*) &server_address, &server_address_len);
    buffer[n] = '\0';
    printf("A message from the server : %s\n", buffer);

    close(client_socket);
    return 0;
}
```

شکل ۵- کد کلاینت UDP



```
// Server side implementation of UDP client-server model
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>

int main() {
    int server_socket;
    char buffer[256];
    char hello_msg[256];
    struct sockaddr_in server_address, client_address;

    // Creating a socket file descriptor
    if ( (server_socket = socket(AF_INET, SOCK_DGRAM, 0)) < 0 ) {
        printf("error: socket creation failed\n");
        return -1;
    }

    // Filling server information
    server_address.sin_family = AF_INET;
    server_address.sin_addr.s_addr=inet_addr("127.0.0.1");
    server_address.sin_port = htons(60000);

    // Bind the socket with the server address
    if ( bind(server_socket, (const struct sockaddr *)&server_address,
        sizeof(server_address)) < 0 )
    {
        printf("error: bind failed\n");
        return (-1);
    }

    int client_address_len = sizeof(client_address);
    int n;
    n = recvfrom(server_socket, (char *)buffer, 255, MSG_WAITALL, ( struct sockaddr
*) &client_address, &client_address_len);
    buffer[n] = '\0';
    printf("A message from a client: %s\n" , buffer);

    sprintf(hello_msg, "Hello from server");
    sendto(server_socket, (const char *)hello_msg, strlen(hello_msg),MSG_CONFIRM,
(const struct sockaddr *) &client_address, client_address_len);
    printf("Hello message sent to the client.\n");

    return 0;
}
```

شکل ۶- کد سرور UDP