توسعه یک برنامه‌ی ساده برای ارتباط از طریق پروتکل UDP

تهیه و تنظیم: مبین خیری

شماره دانشجویی: 994421017

استاد راهنما: دکتر تاجبخش

چکیده:

هدفِ این گزارش، ارائه‌ی روشی برای طراحی یک برنامه جهت برقراری ارتباط در سطح شبکه و با پروتکل UDP می‌باشد. در این گزارش قصد داریم به طراحیِ برنامه‌ای بپردازیم که در واقع شامل دو نقطه‌ی اتصال یا Peer است.

عملکرد این برنامه بر اساسِ استانداردهای پروتکل شناخته‌شده‌ی UDP شکل گرفته. لازم به توضیح است که طراحی چنین برنامه‌ای با استفاده از اغلب زبان‌های برنامه‌نویسیِ موجود، مقدور است. این برنامه در واقع شامل دو قسمت مجزای Client و Server است.

طراحیِ هرکدام از این بخش‌ها، نیازمندی‌های ویژه‌ای را می‌طلبد که در طول این گزارش به آن‌ها خواهیم پرداخت. انتظار ما از برنامه‌ی مذکور این است که با استفاده از آن بتوان از طریقِ قسمتِ Client برای بخشِ Server پیامی ارسال نموده و محتوای آن را از سمتِ Server قابل مشاهده سازیم.

برای آشناییِ بیشتر با پروتکل UDP و تفاوت‌های آن با سایرِ پروتکل‌های موجود برای برقراریِ ارتباط در شبکه‌های کامپیوتری، ابتدا به معرفیِ کلی این پروتوکل می‌پردازیم:

پروتکل UDP چیست؟

UDP، سرواژه عبارت User Datagram Protocol، یک پروتکل ارتباطی است که برای کاربردهای حساس به زمان مانند VoD، پخش زنده و جست‌وجوی DNS در اینترنت استفاده می‌شود. UDP همانند TCP و سایر پروتکل‌های ارتباطی، پیام‌های برنامه‌های کاربردی را دریافت کرده و به تعدای بسته می‌شکند و سپس آن بسته‌ها را در شبکه به سمت مقصد ارسال می‌کند.

این پروتکل که در کنار پروتکل TCP از پرکاربردترین پروتکل‌های انتقال در بستر اینترنت به حساب می‌آید، به خاطر تشکیل ندادن اتصال قبل از انتقال داده سرعت ارتباطات را بسیار افزایش می‌دهد. همین سرعت بالای انتقال دلیل استفاده از UDP برای مصرف‌های حساس به زمان شده است. البته باید در نظر داشت که تشکیل ندادن اتصال به منظور افزایش سرعت انتقال داده‌ها، باعث خواهد شد که بسته‌ها در حین انتقال گم شوند و کیفیت تحت تاثیر این اتفاق قرار گیرد.

کاربردهای UDP

UDP برای کاربردهایی استفاده می‌شود که در آن از دست دادن بسته‌های یا به‌هم خوردن ترتیب آن‌ها اهمیت کم‌تری نسبت به صبر کردن برای رسیدن بسته‌ها دارد. به‌عنوان مثال برای ارسال صوت و فیلم آنلاین از این پروتکل استفاده می‌شود، چرا که از یک سو این کاربردها حساس به زمان هستند و نیاز به انتقال سریع داده‌ها بسیار پررنگ است و از سوی دیگر در طراحی آن‌ها قابلیت تحمل از دست دادن داده‌ها در نظر گرفته شده است. یک نمونه‌ی کاربردی دیگر از استفاده از پروتکل UDP، سیستم Voice over IP یا VoIP است. VoIP مبنای کار بسیاری از سیستم‌های تلفنی بر پایه‌ی اینترنت است که در آن، یک تماس تلفنی کم کیفیت ولی بدون تاخیر نسبت به یک تماس بسیار باکیفیت ولی با تاخیر زیاد مناسب‌تر است. به دلیلی مشابه، برای بازی‌های آنلاین نیز استفاده از UDP گزینه مناسبی است.

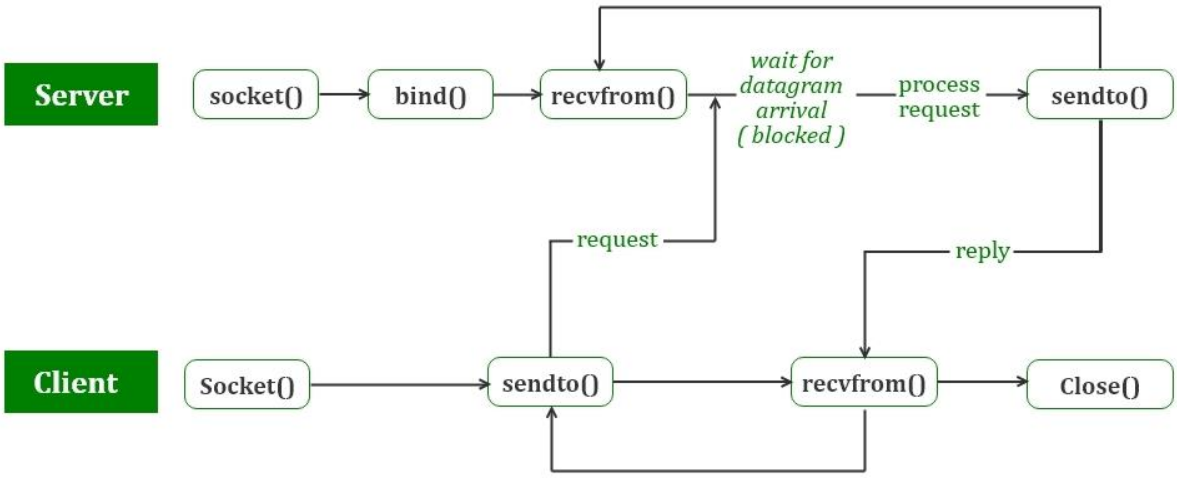
تفاوت پروتکل UDP و TCP

UDP یک روش استاندارد انتقال داده بین دو دستگاه در شبکه است. این پروتکل مکانیزم انتقال را بسیار ساده می‌کند؛ چرا که بدون ایجاد اتصال و فرایندی زمان‌گیر مانند Handshake در TCP، انتقال اطلاعات آغاز می‌شود. از سوی دیگر، در UDP الزامی برای حفظ ترتیب بسته‌ها و بررسی صحت بسته‌های دریافت شده وجود ندارد. این موارد در کنار هم باعث می‌شوند تا انتقال یک فایل یکسان در UDP نسبت به TCP با سرعت بیش‌تری انجام شود.

از طرف دیگر، UDP در مقابل حمله‌های منع سرویس توزیع شده یا DDoS آسیب‌پذیر است. از آن‌جایی که در UDP نیازی به Handshake نیست، حمله‌کنندگان می‌توانند یک سرور قربانی را با ترافیک UDP مورد هجوم قرار دهند. درحالی‌که در پروتکل TCP چون باید در ابتدا Handshake برقرار شود، سرور به‌راحتی می‌تواند به ارتباط های مشکوک پاسخ نداده و از ترافیک ناخواسته پرهیز کند. در یک حمله DDoS مبتنی بر UDP، حجم بسیار زیادی ترافیک UDP به پورت‌های گوناگونی از سرور ارسال خواهند شد که منابع پردازشی سرور را برای پاسخ‌دهی به آن‌ها اشغال می‌کند.

ترکیب چهار فیلد هدر UDP: پورت مبدا، پورت مقصد، طول UDP و جمع کنترل

برای پیاده‌سازیِ خصوصیاتِ برشمرده‌شده‌ی برنامه‌ی مذکور در اینجا زبانِ C را انتخاب کرده‌ایم.
پیاده‌سازیِ این برنامه به کمکِ کتابخانه‌هایی انجام شده که اصالتا در ساختارِ زبانِ C گنجانده شده‌اند.
برای طراحیِ این سیستم ارتباطی، مشابهِ دیاگرامِ زیر عمل خواهیم کرد:



فرآیندِ طراحیِ هرکدام از قسمت‌های برنامه را می‌توان به صورتِ زیر به مراحلِ مختلفی تجزیه نمود:

سرورِ UDP:

1. ساختِ یک سوکتِ UDP
2. اتصالِ سوکتِ به آدرسِ سرور
3. صبر کردن برای دریافت بسته‌های دیاگرام از سمتِ کاربر یا Client
4. پردازشِ بسته‌های دریافتی و ارسالِ پاسخِ مناسب به درخواستِ کاربر
5. بازگشت به مرحله‌ی سوم

سرور Client:

1. ساختِ یک سوکتِ UDP
2. ارسال یک پیام برای سرور
3. صبر کردن برای دریافتِ پاسخِ مناسب از سوی سرور
4. پردازشِ پاسخِ دریافتی از سمتِ سرور و در صورتِ لزوم: بازگشت به مرحله‌ی دوم
5. بستنِ سوکت‌ها و پایان دادن به اتصال

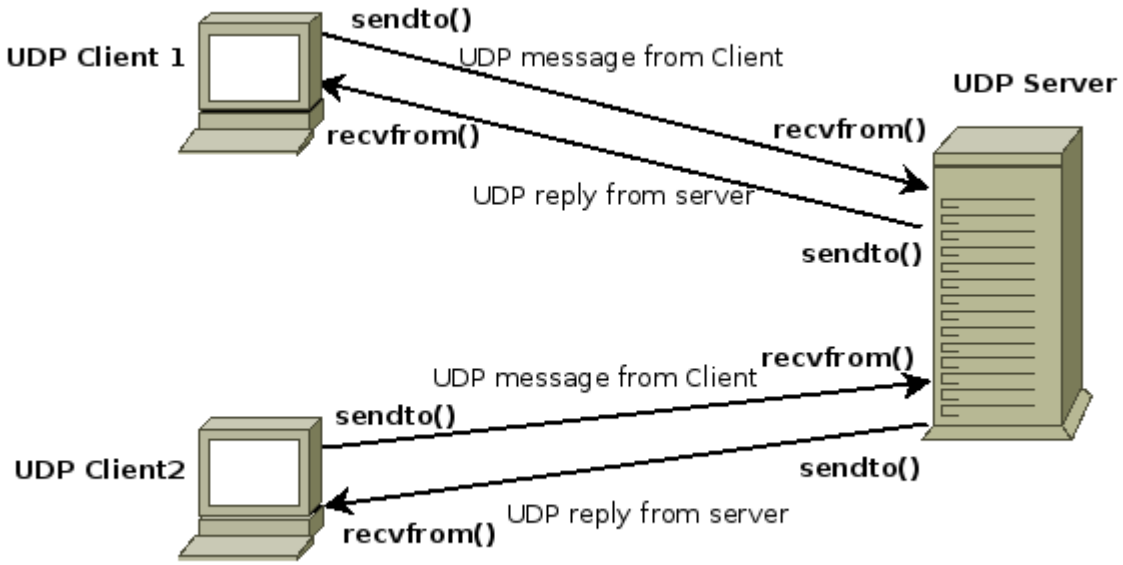
تصویرِ زیر پیاده‌سازیِ بخش مربوط به سرور را نشان می‌دهد:

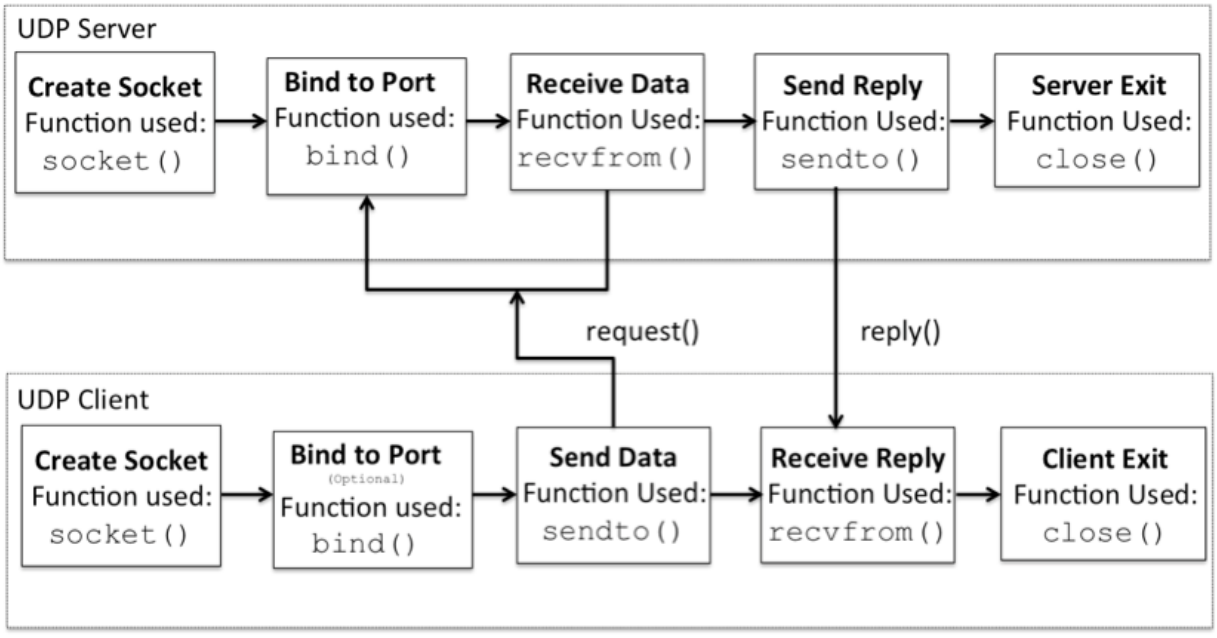
```c
// Server side implementation of UDP client-server model
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>

#define PORT     8080
#define MAXLINE 1024

// Driver code
int main() {
    int sockfd;
    char buffer[MAXLINE];
    char *hello = "Hello from server";
    struct sockaddr_in servaddr, cliaddr;

    // Creating socket file descriptor
    if ( (sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0 ) {
        perror("socket creation failed");
        exit(EXIT_FAILURE);
    }

    memset(&servaddr, 0, sizeof(servaddr));
    memset(&cliaddr, 0, sizeof(cliaddr));

    // Filling server information
    servaddr.sin_family = AF_INET; // IPv4
    servaddr.sin_addr.s_addr = INADDR_ANY;
    servaddr.sin_port = htons(PORT);

    // Bind the socket with the server address
    if ( bind(sockfd, (const struct sockaddr *)&servaddr,
            sizeof(servaddr)) < 0 )
    {
        perror("bind failed");
        exit(EXIT_FAILURE);
    }

    int len, n;

    len = sizeof(cliaddr); //len is value/result

    n = recvfrom(sockfd, (char *)buffer, MAXLINE,
                MSG_WAITALL, ( struct sockaddr *) &cliaddr,
                &len);
    buffer[n] = '\0';
    printf("Client : %s\n", buffer);
    sendto(sockfd, (const char *)hello, strlen(hello),
        MSG_CONFIRM, (const struct sockaddr *) &cliaddr,
            len);
    printf("Hello message sent.\n");

    return 0;
}
```

تصویرِ زیر پیاده‌سازیِ بخش مربوط به کلاینت را نشان می‌دهد:

```c
// 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>

#define PORT     8080
#define MAXLINE 1024

// Driver code
int main() {
    int sockfd;
    char buffer[MAXLINE];
    char *hello = "Hello from client";
    struct sockaddr_in    servaddr;

    // Creating socket file descriptor
    if ( (sockfd = socket(AF_INET, SOCK_DGRAM, 0)) < 0 ) {
        perror("socket creation failed");
        exit(EXIT_FAILURE);
    }

    memset(&servaddr, 0, sizeof(servaddr));

    // Filling server information
    servaddr.sin_family = AF_INET;
    servaddr.sin_port = htons(PORT);
    servaddr.sin_addr.s_addr = INADDR_ANY;

    int n, len;

    sendto(sockfd, (const char *)hello, strlen(hello),
        MSG_CONFIRM, (const struct sockaddr *) &servaddr,
            sizeof(servaddr));
    printf("Hello message sent.\n");

    n = recvfrom(sockfd, (char *)buffer, MAXLINE,
                MSG_WAITALL, (struct sockaddr *) &servaddr,
                &len);
    buffer[n] = '\0';
    printf("Server : %s\n", buffer);

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

تصاویرِ زیر نحوه‌ی ارتباطِ اجزای مختلفِ یک شبکه در این پروتکل را نشان می‌دهند:





برنامه‌ای که در قسمتِ قبل توسعه داده شد، تنها قابلیت تبادلِ پیام‌های متنی را دارد و خروجیِ آن در ترمینالِ سیستم‌عامل به شکلِ زیر خواهد بود:

```
$ ./server
Client : Hello from client
Hello message sent.
```

```
$ ./client
Hello message sent.
Server : Hello from server
```

طراحیِ برنامه‌ی مشابه با زبان برنامه‌نویسیِ پایتون، پیچیدگی‌های بسیار کمتری دارد و همزمان این امکان را به ما می‌دهد که امکان انتخابِ IP های مقصد، پورت‌های مربوطه و محتوای پیام را توسطِ کامندلاین یا ترمینال، از کاربرانِ احتمالی مهیا کنیم.

در این پیاده‌سازی از کتابخانه‌ی pyfiglet استفاده شده که به ما کمک می‌کند تا رابطِ کاربری برنامه‌هایی که از کامندلاین استفاده می‌کنند را با به‌کارگیریِ المان‌های زیبایی، طراحی کنیم.

با پیاده‌سازیِ ویژگی‌هایی که در سطرهای بالا به معرفیِ آن‌ها پرداختیم، در واقع نسخه‌ی مینیمالِ یک مسنجر یا پیامرسان را طراحی کرده‌ایم.

در ادامه تصویر کدهای مربوط به این برنامه‌ی چت را مشاهده می‌کنید. ضمن اینکه همچون گذشته، این برنامه هم برای برقراری ارتباط از پروتکل UDP استفاده می‌کند.

تصویر کد مربوط به این برنامه در صفحه‌ی بعد موجود است.

استفاده از برنامه‌های Putty و نیز Wireshark، امکانِ آنالیز کردنِ بسته‌های ارسالی و دریافتی را برای ما میسر خواهند کرد.

پایان.

```python
import os
from pyfiglet import Figlet

os.system("clear")
pyf = Figlet(font='puffy')
a = pyf.renderText("UDP Chat App with Multi-Threading")
os.system("tput setaf 3")
print(a)

# importing modules

import socket
import os
import time
import threading
import sys

# AF_INET = Network Address Family : IPv4
# SOCK_DGRAM = DataGram Socket : UDP

# Function for receiving
def receiver():
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    s.bind((ip_sender, port_sender)) #binding the IP address and port
number
    while True:
        msg = s.recvfrom(1024)
        print("\n"+msg[0].decode())
        if "exit" in msg[0].decode() or "bye" in msg[0].decode():
            sys.exit()

#Function for sending
def sender():
    s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    text = "hello"
    while True:
        if "bye" in text or "exit" in text or "finish" in text:
            exit()
        else:
            text = input(f'{name}:')
            text = name+":"+text
            s.sendto(text.encode(), (ip_receiver, port_receiver))

print("Initializing....")
ip_receiver = input("\nEnter the IP of reciever: ")
port_receiver = int(input("\nEnter the port of the reciever: "))
ip_sender = input("\nEnter the IP of your system : ")
port_sender = int(input("\nEnter the port of your system: "))
name = input("Enter your name: ")


print("Waiting for client....")
time.sleep(1)
print("Connection established....")

# Using Multi-threading
send = threading.Thread(target=sender)
receive = threading.Thread(target=receiver)

send.start()
receive.start()
```