دانشگاه صنعتی امیرکبیر
دانشکده مهندسي برق

پروژه اول
درس سامانه های چند رسانه ای

استاد : جناب آقای دکتر شریفیان

مهشاد اکبری سریزدی _ ۹۹۲۳۰۹۳
آنوشا شریعتی - ۹۹۲۳۰۴۱

فرستنده (transmitter-Server)

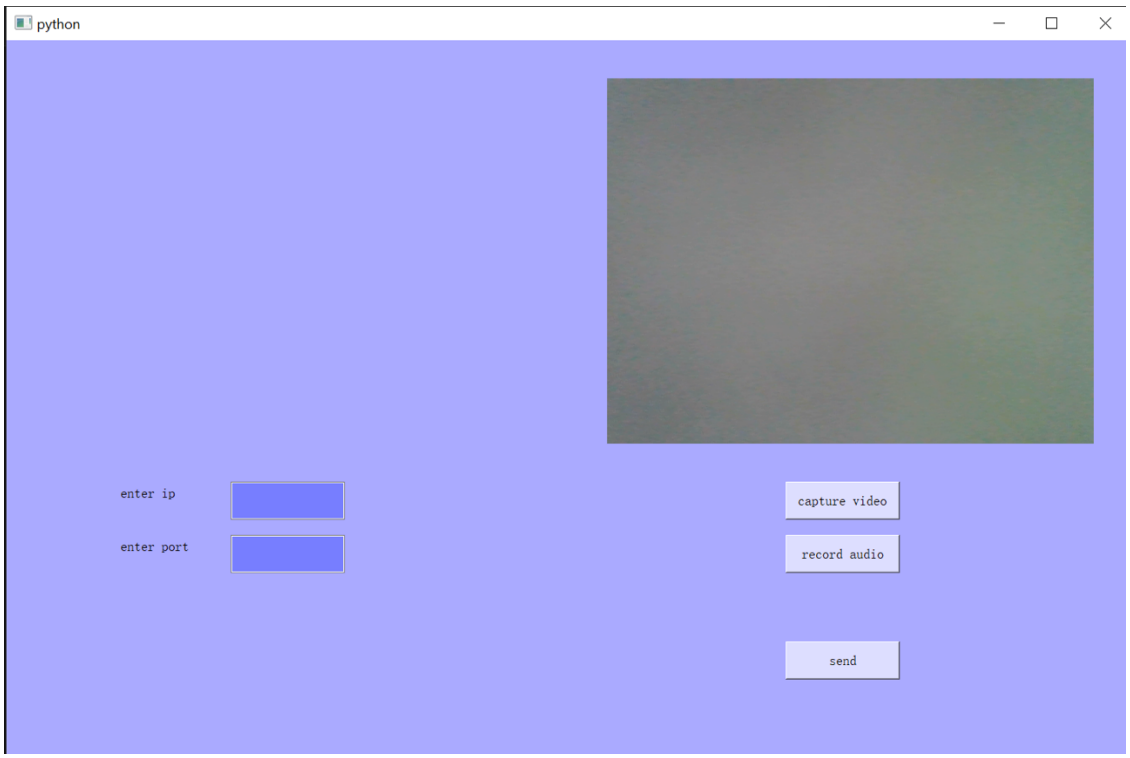
طراحی رابط کاربری (graphic user interface)

در این بخش در ابتدا کتابخانه PYQT5 را اضافه کرده و به کمک دستورات زیر به طراحی رابط کاربری پرداختیم. برای انتخاب متنوع تر رنگ و فونت و... از برنامه QT DESIGNER استفاده کرده و فایل UI را به فایل PY تبدیل کرده و با اعمال تغییرات مناسب در کد به خروجی دلخواه که در زیر آمده است رسیدیم.

```
from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtCore import Qt,QTimer
from PyQt5.QtGui import QFont, QImage ,QPixmap
from PyQt5.QtWidgets import QMainWindow,QApplication,QWidget,QPushButton ,
QLCDNumber ,QFrame , QLineEdit ,QLabel, QMessageBox

if __name__ == "__main__":
    app = QApplication(sys.argv)
    MainWindow = QWidget()
    MainWindow.setObjectName("MainWindow")
    MainWindow.resize(1480, 1000)
    MainWindow.setStyleSheet("background-color: rgb(170, 170, 255);")

    #audio record button
    audio_btn = QPushButton("record audio",MainWindow)
    audio_btn.setGeometry(QtCore.QRect(1025, 650 , 150, 50))
    audio_btn.setStyleSheet("background-color: rgb(221, 222, 255);")
    audio_btn.setObjectName("audio_btn")
    #video capture button
    video_btn = QPushButton("capture video",MainWindow)
    video_btn.setGeometry(QtCore.QRect(1025, 580, 150, 50))
    video_btn.setStyleSheet("background-color: rgb(221, 222, 255);")
    video_btn.setObjectName("video_btn")
    #send button
    send_btn = QtWidgets.QPushButton("send",MainWindow)
    send_btn.setGeometry(QtCore.QRect(1025, 790, 150, 50))
    send_btn.setStyleSheet("background-color: rgb(221, 222, 255);")
    send_btn.setObjectName("send_btn")
    #ip text
    ip_text = QLineEdit(MainWindow)
    ip_text.setGeometry(QtCore.QRect(295, 580, 150, 50))
     ip_text.setStyleSheet("background-color:  rgb(119,  126,  255);  color:
rgb(255, 255, 255);")
    ip_text.setObjectName("ip_text")

    #port text
```

```
    port_text = QLineEdit(MainWindow)
    port_text.setGeometry(QtCore.QRect(295, 650, 150, 50))
     port_text.setStyleSheet("background-color: rgb(119, 126, 255); color:
rgb(255, 255, 255);")
    port_text.setObjectName("port_text")
    #ip label
    enter_ip = QLabel("enter ip",MainWindow)
    enter_ip.setGeometry(QtCore.QRect(150,585, 100, 20))
    enter_ip.setObjectName("enter_ip")
    #port label
    enter_port = QLabel("enter port",MainWindow)
    enter_port.setGeometry(QtCore.QRect(150, 655, 100, 20))
    enter_port.setObjectName("enter_port")
```

```
#show window
    MainWindow.show()
    sys.exit(app.exec_())
```



اشتراک گذاری وبکم:

برای این قسمت نیاز داشتیم کتابخانه CV2 را اضافه کنیم تا بتوانیم از دستورات زیر استفاده کنیم. در ابتدای کد تابع آپدیت فریم را تعریف کرده تا در هر لحظه یک فریم از وبکم را کپچر کرده و در متغیر فریم بریزد وبا اعمال تغییراتی روی آن خروجی را به عنوان لیبل ذخیره کند. سپس در قسمت اصلی برنامه لیبل تعریف شده را با تنظیم سایز و مختصات در پنجره اصلی نمایش میدهیم. برای آپدیت شدن هرلحظه فریم از تایمر استفاده میکنیم و طبق دستور زیر با تمام شدن تایمر تابع فریم آپدیت که در بالا توضیح داده شد صدا میشود.

```
import cv2
def frame_update():
    global label
    cap = cv2.VideoCapture(0)
    ret, frame = cap.read()
    while ret:
        ret, frame = cap.read()
         # Convert the frame to QImage and then to QPixmap to display in
QLabel
        rgb_image = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        h, w, ch = rgb_image.shape
        bytes_per_line = ch * w
        convert_to_Qt_format = QImage(rgb_image.data, w, h, bytes_per_line,
QImage.Format_RGB888)
        p = convert_to_Qt_format.scaled(640, 480, Qt.KeepAspectRatio)
        pixmap = QPixmap.fromImage(p)
        label.setPixmap(pixmap)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
    cap.release()
```

```
    # Label to display the camera frame
    label = QLabel(MainWindow)
    label.resize(640, 480)
    label.move(790, 50)

    # Start video capture and update QLabel
    timer = QTimer(MainWindow)
    timer.start(1000//30)
    timer.timeout.connect(frame_update)
```

ضبط شدن صدا :

برای ضبط شدن صدا نیاز به اضافه کردن کتابخانه SD داشتیم. سپس با تعریف تابع زیر ویس 30 ثانیه ای را با اسم دلخواه ذخیره میکنیم. سپس در برنامه اصلی با فشرده شدن دکمه ریکرود آدیو این تابع اجرا میشود.

```
import os
import sounddevice as sd
from scipy.io.wavfile import write
def audio_record():
    fs = 44100  # Sample rate
    seconds = 30  # Duration of recording
    myrecording = sd.rec(int(seconds * fs), samplerate=fs, channels=2)
    sd.wait()
    write('audio.wav', fs, myrecording)
    QMessageBox.information(None,"audio","your voice is saved
successfully")
```

```
audio_btn.clicked.connect(audio_record)
```

گرفتن تصویر :

برای گرفتن تصویر نیاز به اضافه کردن کتابخانه CV2 داشتیم. سپس با تعریف تابع زیر تصویر گرفته شده را با اسم دلخواه ذخیره میکنیم. سپس در برنامه اصلی با فشرده شدن دکمه کپچر ویدیو این تابع اجرا میشود.

```
def video_capture():
    cap=cv2.VideoCapture(0)
    ret,frame = cap.read()
    cv2.imwrite('photo.jpg', frame)
    cap.release()
    QMessageBox.information(None,"video","your photo is captured
successfully")
```

```
video_btn.clicked.connect(video_capture)
```

گرفتن ورودی ip ,port :

برای گرفتن مقدار آی پی و پورت توابع زیر را تعریف کرده و سپس در کد اصلی با وارد شدن این مقادیر در تکست باکس توابع بالا اجرا میشوند.

```
def ip_entered():
    ip_val = ip_text.text()

def port_entered():
    port_val = int(port_text.text())
```

```
    ip_text.returnPressed.connect(ip_entered)
    port_text.returnPressed.connect(port_entered)
```

اتصال TCP :

برای نوشتن کد این بخش کتابخانه های زیر را به برنامه مان اضافه می کنیم .

```
1    import sys
2    import socket
3    import os
```

از آنجایی که میخواهیم صوت و تصویر را بین
این دو جا به جا کنیم ’ برای هر کدام از صوت و تصویر دو تابع جداگانه هم در بخش سرور وهم در بخش کلاینت ایجاد کردیم .

ابتدا به بخش ارسال فایل صوتی می پردازیم ’ یک تابع با نام audio_server ایجاد کردیم و در آن کد های مربوط به برقراری ارتباط و ارسال صوت را به صورت زیر نوشتیم .

```
def audio_server():
    host = ip_val  # Get the host IP from the GUI
    port = port_val  # Get the port number from the GUI
```

در این تابع ابتدا مقدار port و host را با استفاده از توابعی که برای بخش ui سیستم طراحی کردیم تا فرد بتواند آدرس port و host ای که میخواهد با آن در ارتباط باشد را وارد کند ’ میگیریم .

```
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

در این خط درواقع در حال فراخوانی ارتباط tcp می باشیم و درواقع این خط تقریبا نقش ثابتی در هر کد tcp دارد . در این جا یک سوکت فراخوانی کردیم .

```
   server_socket.bind((host, port))
    server_socket.listen(1)
```

حال درواقع سوکت فراخوانی شده را مقید به port و host ثابتی که در ابتدا گرفتیم می کند.خط بعد درواقع سوکت سرور منتظر است برای ارتباط گرفتن کاربر.مقدار ۱ به این معناست که لیست کاربران قابل قبول یکی می باشد و اگر دیگر کاربری برای متصل شدن به سرور تلاش کند ممکن نیست و احتمالا خطا دریافت میکند.

```
    file_to_send = 'audio.wav'  # Check this path and update it accordingly
    if not os.path.exists(file_to_send):
        print("File does not exist. Exiting...")
        return

    while True:
        conn, addr = server_socket.accept()
```

```
        print(f"Connected by {addr}")

        with open(file_to_send, 'rb') as f:
            data = f.read()
            size = len(data)
            conn.sendall(f"{size}\n".encode('utf-8'))  # Size followed by a
newline character
            conn.sendall(data)

        conn.close()
        print("Audio file sent and connection closed.")
        break
```

سرور ابتدا بررسی می‌کند که فایل مورد نظر وجود دارد یا خیر و سپس یک حلقه شروع می‌کنیم که این به این معنی است که سرور به طور مداوم برای پذیرش درخواست‌های جدید اتصال اجرا می‌شود، مگر اینکه با دستوری از حلقه خارج شود . پس از برقراری اتصال، فایل را می‌خواند و اندازه فایل را به کلاینت ارسال می‌کند در نهایت کل داده‌های فایل را به کلاینت ارسال می‌کند. پس از ارسال، اتصال را می‌بندد و از حلقه خارج می شود.
حال به بخش ارسال تصویر می پردازیم .

```
def picture_server():

    host = ip_val
    port = port_val

    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.bind((host, port))
    server_socket.listen(1)
    print(f"Picture Server started at {host} on port {port}")
```

ابتدای کار همانند بخش قبل می باشد و بعد از گرفتن پورت و آی پی یک سوکت برای اتصال tcp میسازیم و سپس سرور به کلاینت گوش میکند .

```
    while True:
        conn, addr = server_socket.accept()
        print(f"Connected by {addr}")

        with open('photo.jpg', 'rb') as f:
            data = f.read()
            size = len(data)
            conn.sendall(str(size).encode('utf-8'))
            conn.sendall(data)

        conn.close()
        print("Picture sent and connection closed.")
```

```
        break  # close server after sending file
```

سرور منتظر می‌ماند تا یک کلاینت به آن متصل شود. تابع accept() استفاده می‌شود تا درخواست اتصال کلاینت را بپذیرد. هنگامی که اتصال برقرار می‌شود، اطلاعات اتصال (شامل آدرس کلاینت) را چاپ می‌کند.فایل تصویری (photo.jpg) باز شده و محتوای آن خوانده می‌شود. سپس، اندازه فایل محاسبه شده و به کلاینت ارسال می‌شود. پس از ارسال اندازه، کل داده‌های تصویری نیز به کلاینت ارسال می‌شوند. این برای این است که کلاینت بداند چه مقدار داده باید دریافت کند و اینکه تمام داده‌ها دریافت شده باشند.پس از ارسال داده‌ها، اتصال با کلاینت بسته شده و یک پیام چاپ می‌شود که نشان می‌دهد تصویر ارسال شده و اتصال بسته شده است. سپس با استفاده از دستور break سرور پس از ارسال فایل توقف می‌کند ومنتظر اتصالات جدید نمی‌ماند.

گیرنده (receiver-Client)

طراحی رابط کاربری (graphic user interface)

طراحی رابط کاربری در گیرنده هم مانند فرستنده انجام شد با این تفاوت گه دکمه هایی برای پخش کردن صدا و نمایش دادن تصویر دارد.

```
if __name__ == "__main__":
    app = QApplication(sys.argv)
    MainWindow = QWidget()
    MainWindow.setObjectName("MainWindow")
    MainWindow.resize(1480, 1000)
    MainWindow.setStyleSheet("background-color: rgb(170, 170, 255);")
    #audio play button
    audio_btn = QPushButton("play audio",MainWindow)
    audio_btn.setGeometry(QtCore.QRect(1025, 650 , 150, 50))
    audio_btn.setStyleSheet("background-color: rgb(221, 222, 255);")
    audio_btn.setObjectName("audio_btn")
    #video capture button
    video_btn = QPushButton("show picture",MainWindow)
    video_btn.setGeometry(QtCore.QRect(1025, 580, 150, 50))
    video_btn.setStyleSheet("background-color: rgb(221, 222, 255);")
    video_btn.setObjectName("video_btn")

    #connect button
    connect_btn = QPushButton("connect",MainWindow)
    connect_btn.setGeometry(QtCore.QRect(295, 720, 150, 50))
    connect_btn.setStyleSheet("background-color: rgb(221, 222, 255);")
    connect_btn.setObjectName("connect_btn")
```

```
    #ip text
    ip_text = QLineEdit(MainWindow)
    ip_text.setGeometry(QtCore.QRect(295, 580, 150, 50))
    ip_text.setStyleSheet("background-color: rgb(119, 126, 255); color:
rgb(255, 255, 255);")
    ip_text.setObjectName("ip_text")
    #port text
    port_text = QLineEdit(MainWindow)
    port_text.setGeometry(QtCore.QRect(295, 650, 150, 50))
    port_text.setStyleSheet("background-color: rgb(119, 126, 255); color:
rgb(255, 255, 255);")
    port_text.setObjectName("port_text")
    #ip label
    enter_ip = QLabel("enter ip",MainWindow)
    enter_ip.setGeometry(QtCore.QRect(150,585, 100, 20))
    enter_ip.setObjectName("enter_ip")
    #port label
    enter_port = QLabel("enter port",MainWindow)
    enter_port.setGeometry(QtCore.QRect(150, 655, 100, 20))
    enter_port.setObjectName("enter_port")
```

پخش شدن صدا :

برای پخش شدن صدا کتابخانه playsound را اضافه کردیم و با دستور زیر فایل دریافت شده از آدرس زیر پخش میشود. همچنین با اضافه کردن دستور زیر در کد اصلی با فشردن دکمه پلی آدیو تابع تعریف شده اجرا میشود.

```
from playsound import playsound
from pydub import AudioSegment
from pydub.playback import play
def audio_play():
    playsound('/Users/My/Desktop/receiver/audio.wav')
    QMessageBox.information(None,"audio","this was your audio ^-^")
```

```
audio_btn.clicked.connect(audio_play)
```

نمایش تصویر :

برای پخش شدن نمایش تصویر کتابخانه PIL را اضافه کردیم و با دستور زیر فایل دریافت شده از آدرس زیر باز میشود. همچنین با اضافه کردن دستور زیر در کد اصلی با فشردن دکمه شو پیکچر تابع تعریف شده اجرا میشود.

```
from PIL import Image
def picture_show():
    image = cv2.imread('/Users/My/Desktop/receiver/photo.jpg')
    cv2.imshow("photo",image)
    cv2.waitKey(0)
    QMessageBox.information(None,"photo","this is your photo ^-^")
```

```
video_btn.clicked.connect(picture_show)
```

گرفتن ورودی ip ,port :

همانند کد فرستنده برای گیرنده هم برای گرفتن مقدار آی پی و پورت توابع زیر را تعریف کرده و سپس در کد اصلی با وارد شدن این مقادیر در تکست باکس توابع بالا اجرا میشوند.

```
def ip_entered():
    ip_val = ip_text.text()
def port_entered():
    port_val = int(port_text.text())

    ip_text.returnPressed.connect(ip_entered)
    port_text.returnPressed.connect(port_entered)
```

اتصال TCP :

برای نوشتن کد این بخش کتابخانه های زیر را به برنامه مان اضافه می کنیم .

```
import sys
import socket
import os
```

ابتدا به تابع تعریف شده برای صوت میپردازیم:

```
def audio_client():
    host = ip_val
    port = port_val

    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client_socket.connect((host, port))
```

همانند بخش سرور، این بارهم باید کلاینت port و host سروری که قصد دارد به آن متصل شود را دریافت کند و یک سوکت برای اتصال tcp ایجاد کند.
خط آخر خط دستور برقراری اتصال به سرور مشخص شده توسط host و port را به سوکت می‌دهد. اگر سرور در دسترس باشد ، اتصال برقرار خواهد شد و داده‌ها می‌توانند بین کلاینت و سرور منتقل شوند.

```
# Assuming the first message is the size of the audio file
    size = int(client_socket.recv(1024).decode())
    data = b''
    while len(data) < size:
        packet = client_socket.recv(4096)
        if not packet:
            break
        data += packet

    # Save the received audio file
    with open('/Users/My/Desktop/receiver/audio.wav', 'wb') as f:
        f.write(data)

    client_socket.close()
    QMessageBox.information(None, "Audio", "Audio file received and saved
successfully.")
```

ابتدا از سوکت 1024 بایت داده دریافت می‌کند که شامل اندازه فایل صوتی (به صورت یک رشته) است. داده دریافتی دیکود شده و به عدد صحیح تبدیل می‌شود تا مشخص شود چه مقدار داده باید در ادامه دریافت شود. سپس در حلقه‌ای داده‌ها را از سوکت دریافت می‌کند تا زمانی که تمام داده‌های فایل براساس اندازه اعلام شده

دریافت شود. برای هربار، 4096 بایت داده دریافت میکند و به متغیر data اضافه می‌شود. اگر داده‌ای دریافت نشود از حلقه خارج می شود و این پروسه پایان می یابد . سپس فایل صوتی دریافت شده را در مسیر مشخص شده ذخیره می‌کند. از wb استفاده می‌کنیم تا فایل به صورت باینری نوشته شود. بعد از این اتصال سوکت بسته می‌شود و یک پنجره پیام نمایش داده می‌شود که اطلاع می‌دهد فایل صوتی با موفقیت دریافت و ذخیره شده است.

حال به بخش ارسال تصویر می پردازیم :

```
def picture_client():
    host = ip_val
    port = port_val

    client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    client_socket.connect((host, port))
```

بخش ابتدایی آن همانند بخش های قبل می باشد و یک سوکت برای اتصال ایجاد می کند و به پورت و هاست سروری که میخواهد اتصال پیدا کند وصل می شود .

```
size = int(client_socket.recv(1024).decode())
    data = b''
    while len(data) < size:
        packet = client_socket.recv(4096)
        if not packet:
            break
        data += packet

    # Save the received image file
    with open('/Users/My/Desktop/receiver/photo.jpg', 'wb') as f:
        f.write(data)

    client_socket.close()
    QMessageBox.information(None, "Picture", "Picture file received and saved
successfully.")
```

این کد هم مانند ارسال فایل صوتی می باشد با این تفاوت که تصویر ارسال میشود . مقدار داده ای که باید دریافت شود مشخص می شود و سپس یک حلقه شروع می‌شود که در آن داده‌ها به تدریج از سرور دریافت می‌شوند تا زمانی که کل داده‌های فایل بر اساس اندازه اعلام شده دریافت شود. اگر دیگر داده‌ای دریافت نشود حلقه متوقف می‌شود.پس از دریافت داده‌ها، فایل تصویری در مسیر مشخص شده ذخیره می‌شود. در نهایت، اتصال سوکت بسته می‌شود و یک پیغام نمایش داده می‌شود که اطلاع می‌دهد فایل تصویری با موفقیت دریافت و ذخیره شده است.

: Threading

```
def tcp_connect(): #for send button

    threading.Thread(target=audio_client, daemon=True).start()

    threading.Thread(target=picture_client, daemon=True).start()
```

این تابع برای وقتی است که کاربر بر روی دکمه ارسال کلیک می‌کند، دو threadرا راه‌اندازی می‌کند تا دو تابع مجزا را به صورت موازی اجرا کند. این بخش درواقع برای جلوگیری از کرش کردن برنامه استفاده می شود .
یک thread برای audio_client میباشد و دیگری برای picture_client.

```
connect_btn.clicked.connect(tcp_connect)
```

درواقع هنگامی که روی دکمه connect تعریف شده در رابط کاربری کلیک شود این به تابع tcp_connect که بالاتر آورده شده است متصل می شود و درواقع اجرای برنامه را به صورت موازی برای گرفتن صوت و تصویر انجام می دهد .