```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-


import cv2
import socket
import struct
import pickle
import sys
import pyaudio
import threading
import tkinter as tk
from tkinter import ttk, filedialog, messagebox
from PIL import Image, ImageTk
import wave
import time
import numpy as np
from queue import Queue, Empty
import psycopg2
import os


class AudioVideoStreamClient:
    def __init__(self, server_ip, video_port=9999, audio_port=9998):
        """
        初始化音视频流客户端

        Args:
            server_ip: 服务器 IP 地址
            video_port: 视频端口
            audio_port: 音频端口
        """
        self.server_ip = server_ip
        self.video_port = video_port
        self.audio_port = audio_port
        self.video_socket = None
        self.audio_socket = None
        self.payload_size = struct.calcsize("<L")
        self.data_buffer = b""

        # 音频参数
        self.audio = pyaudio.PyAudio()
        self.audio_stream = None
        self.running = False
```

```
        # 视频帧队列（用于线程间传递）
        self.frame_queue = Queue(maxsize=2)  # 限制队列大小避免内存占用过大

        # 录制相关
        self.recording = False
        self.video_writer = None
        self.audio_frames = []
        self.record_lock = threading.Lock()

        # GUI 相关
        self.root = None
        self.video_label = None
        self.current_frame = None
        self.is_fullscreen = False

        # 数据库连接参数
        self.db_config = {
            'database': 'record',
            'user': 'xlevon',
            'password': 'xlevon!2025',
            'host': '192.168.240.6',
            'port': '7654'
        }

    def connect(self):
        """连接到服务器"""
        try:
            # 连接视频流
            self.video_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            self.video_socket.setsockopt(socket.IPPROTO_TCP, socket.TCP_NODELAY, 1)
            self.video_socket.setsockopt(socket.SOL_SOCKET, socket.SO_RCVBUF, 65536)
            self.video_socket.connect((self.server_ip, self.video_port))
            print(f"✓ 视频流已连接: {self.server_ip}:{self.video_port}")

            # 连接音频流
            self.audio_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
            self.audio_socket.setsockopt(socket.IPPROTO_TCP, socket.TCP_NODELAY, 1)
            self.audio_socket.connect((self.server_ip, self.audio_port))
            print(f"✓ 音频流已连接: {self.server_ip}:{self.audio_port}")

            # 初始化音频播放
            self.audio_stream = self.audio.open(
                format=pyaudio.paInt16,
```

```
                channels=1,
                rate=16000,
                output=True,
                frames_per_buffer=1024
            )
            print(f"✓ 音频播放已初始化")

            return True
        except Exception as e:
            print(f"错误: 无法连接到服务器 - {e}")
            return False

    def receive_frame(self):
        """接收单个视频帧"""
        try:
            data = self.data_buffer

            # 接收消息大小（4 字节）
            while len(data) < self.payload_size:
                packet = self.video_socket.recv(4096)
                if not packet:
                    return None
                data += packet

            # 解析消息大小
            packed_msg_size = data[:self.payload_size]
            data = data[self.payload_size:]
            msg_size = struct.unpack("<L", packed_msg_size)[0]

            # 接收完整的帧数据
            while len(data) < msg_size:
                remaining = msg_size - len(data)
                chunk_size = min(remaining, 32768)
                packet = self.video_socket.recv(chunk_size)
                if not packet:
                    return None
                data += packet

            frame_data = data[:msg_size]
            self.data_buffer = data[msg_size:]

            # 反序列化和解码
            frame = pickle.loads(frame_data)
            frame = cv2.imdecode(frame, cv2.IMREAD_COLOR)
```

```
            return frame
        except socket.timeout:
            raise
        except Exception as e:
            print(f"接收帧错误: {e}")
            return None

def receive_audio(self):
    """音频接收线程"""
    self.audio_socket.settimeout(0.1)

    while self.running:
        try:
            # 接收音频数据大小
            size_data = b''
            while len(size_data) < 4:
                packet = self.audio_socket.recv(4 - len(size_data))
                if not packet:
                    return
                size_data += packet

            audio_size = struct.unpack("<L", size_data)[0]

            # 接收音频数据
            audio_data = b''
            while len(audio_data) < audio_size:
                remaining = audio_size - len(audio_data)
                packet = self.audio_socket.recv(min(remaining, 4096))
                if not packet:
                    return
                audio_data += packet

            # 录制音频
            if self.recording:
                with self.record_lock:
                    self.audio_frames.append(audio_data)

            # 播放音频
            if self.audio_stream:
                self.audio_stream.write(audio_data)

        except socket.timeout:
            continue
```

```
            except Exception as e:
                if self.running:
                    print(f"音频接收错误: {e}")
                break

    def receive_video(self):
        """视频接收线程"""
        self.video_socket.settimeout(0.1)

        while self.running:
            try:
                frame = self.receive_frame()
                if frame is not None:
                    # 将帧放入队列，如果队列满了则丢弃旧帧
                    try:
                        self.frame_queue.put(frame, block=False)
                    except:
                        # 队列满，丢弃最旧的帧
                        try:
                            self.frame_queue.get_nowait()
                            self.frame_queue.put(frame, block=False)
                        except:
                            pass
            except socket.timeout:
                continue
            except Exception as e:
                if self.running:
                    print(f"视频接收错误: {e}")
                break

    def start_recording(self):
        """开始录制"""
        if self.recording:
            return

        # 临时文件路径
        self.temp_video_path = f"temp_video_{int(time.time())}.avi"
        self.temp_audio_path = f"temp_audio_{int(time.time())}.wav"

        # 初始化视频写入器（使用当前帧的尺寸）
        if self.current_frame is not None:
            height, width = self.current_frame.shape[:2]
            fourcc = cv2.VideoWriter_fourcc(*'XVID')
            self.video_writer = cv2.VideoWriter(self.temp_video_path, fourcc, 30, (width,
```

```
height))

            self.audio_frames = []
            self.recording = True
            print("✓ 开始录制")

        def stop_recording(self):
            """停止录制并保存"""
            if not self.recording:
                return

            self.recording = False
            print("✓ 停止录制")

            # 关闭视频写入器
            if self.video_writer:
                self.video_writer.release()
                self.video_writer = None

            # 保存音频
            if self.audio_frames:
                with wave.open(self.temp_audio_path, 'wb') as wf:
                    wf.setnchannels(1)
                    wf.setsampwidth(2)  # 16-bit
                    wf.setframerate(16000)
                    wf.writeframes(b".join(self.audio_frames))

            # 弹出保存对话框
            self.root.after(100, self.show_save_dialog)

        def show_save_dialog(self):
            """显示保存对话框"""
            output_path = filedialog.asksaveasfilename(
                title="保存录制的视频",
                defaultextension=".mp4",
                filetypes=[("MP4 视频", "*.mp4"), ("AVI 视频", "*.avi"), ("所有文件", "*.*")]
            )

            if output_path:
                try:
                    import shutil
                    import os

                    # 生成文件名（不含扩展名）
```

```
                base_name = output_path.rsplit('.', 1)[0]
                output_ext = output_path.rsplit('.', 1)[1] if '.' in output_path else 'mp4'

                # 原始文件的目标路径
                video_output = base_name + '_video.avi'
                audio_output = base_name + '_audio.wav'

                # 先复制原始文件（保留副本）
                if hasattr(self, 'temp_video_path') and os.path.exists(self.temp_video_path):
                    shutil.copy2(self.temp_video_path, video_output)
                    print(f"✓ 已保存原始视频: {video_output}")

                    # 插入视频记录到数据库
                    video_name = os.path.basename(video_output)
                    self.insert_file_record('.avi', video_name, os.path.abspath(video_output))

                if hasattr(self, 'temp_audio_path') and os.path.exists(self.temp_audio_path):
                    shutil.copy2(self.temp_audio_path, audio_output)
                    print(f"✓ 已保存原始音频: {audio_output}")

                    # 插入音频记录到数据库
                    audio_name = os.path.basename(audio_output)
                    self.insert_file_record('.wav', audio_name,
os.path.abspath(audio_output))

                # 尝试使用 ffmpeg 合并音视频
                success = self.merge_audio_video(output_path)

                if success:
                    # 插入合成视频记录到数据库
                    merged_name = os.path.basename(output_path)
                    merged_ext = '.' + output_path.rsplit('.', 1)[1] if '.' in output_path else
'.mp4'
                    self.insert_file_record(merged_ext, merged_name,
os.path.abspath(output_path))

                    messagebox.showinfo("保存成功",
                        f"✅ 合成视频: {output_path}\n\n"
                        f"   原始视频: {video_output}\n\n"
                        f"   原始音频: {audio_output}\n\n"
                        f"   已记录到数据库")
                    # 恢复状态显示
                    self.status_label.config(text="   已连接", foreground="green")
                else:
```

```
                    # 如果合并失败，只有原始文件
                    messagebox.showwarning("部分成功",
                        f"无法合并音视频（可能缺少 ffmpeg）\n\n"
                        f"    原始视频: {video_output}\n\n"
                        f"    原始音频: {audio_output}\n\n"
                        f"    已记录到数据库\n\n"
                        f"提示：安装 ffmpeg 后可自动合并音视频")
                    # 恢复状态显示
                    self.status_label.config(text="    已连接", foreground="green")

            except Exception as e:
                messagebox.showerror("保存失败", f"保存文件时出错: {e}")
                # 恢复状态显示
                self.status_label.config(text="    已连接", foreground="green")
        else:
            # 用户取消，删除临时文件
            import os
            try:
                if hasattr(self, 'temp_video_path'):
                    os.remove(self.temp_video_path)
                if hasattr(self, 'temp_audio_path'):
                    os.remove(self.temp_audio_path)
            except:
                pass
            # 恢复状态显示
            self.status_label.config(text="    已连接", foreground="green")

    def insert_file_record(self, file_type, file_name, file_path):
        """将文件信息插入数据库"""
        conn = None
        cur = None
        try:
            # 获取系统时间戳
            from datetime import datetime
            current_time = datetime.now()

            # 连接数据库
            conn = psycopg2.connect(**self.db_config)
            cur = conn.cursor()

            # 插入记录（使用系统时间戳）
            sql = """
                INSERT INTO record_address (file_type, file_name, file_path, created_at)
                VALUES (%s, %s, %s, %s)
```

```
                """
                cur.execute(sql, (file_type, file_name, file_path, current_time))
                conn.commit()

                print(f"✓ 数据库记录已添加: {file_name} ({file_type}) - {current_time.strftime('%Y-%m-%d %H:%M:%S')}")
                return True

            except Exception as e:
                print(f"⚠ 数据库插入失败: {e}")
                if conn:
                    conn.rollback()
                return False
            finally:
                if cur:
                    cur.close()
                if conn:
                    conn.close()

        def merge_audio_video(self, output_path):
            """合并音视频文件"""
            try:
                import subprocess
                import os

                # 检查是否有临时文件
                if not hasattr(self, 'temp_video_path') or not hasattr(self, 'temp_audio_path'):
                    return False

                if not os.path.exists(self.temp_video_path) or not os.path.exists(self.temp_audio_path):
                    return False

                # 尝试使用 ffmpeg 合并
                # 使用 -y 参数覆盖已存在的文件
                cmd = [
                    'ffmpeg',
                    '-y',  # 覆盖输出文件
                    '-i', self.temp_video_path,  # 视频输入
                    '-i', self.temp_audio_path,  # 音频输入
                    '-c:v', 'copy',              # 复制视频流（不重新编码）
                    '-c:a', 'aac',               # 音频编码为 AAC
                    '-strict', 'experimental',
                    output_path
```

```
            ]

            # 执行 ffmpeg 命令
            result = subprocess.run(cmd, capture_output=True, text=True, timeout=30)

            if result.returncode == 0:
                # 成功合并，清理临时文件
                try:
                    os.remove(self.temp_video_path)
                    os.remove(self.temp_audio_path)
                    print(f"✓ 已合成视频: {output_path}")
                except:
                    pass
                return True
            else:
                print(f"ffmpeg 错误: {result.stderr}")
                # 合并失败也清理临时文件（因为已经复制保存了）
                try:
                    os.remove(self.temp_video_path)
                    os.remove(self.temp_audio_path)
                except:
                    pass
                return False

        except FileNotFoundError:
            # ffmpeg 未安装，清理临时文件
            print("未找到 ffmpeg，无法合并音视频")
            try:
                os.remove(self.temp_video_path)
                os.remove(self.temp_audio_path)
            except:
                pass
            return False
        except subprocess.TimeoutExpired:
            print("ffmpeg 合并超时")
            try:
                os.remove(self.temp_video_path)
                os.remove(self.temp_audio_path)
            except:
                pass
            return False
        except Exception as e:
            print(f"合并音视频时出错: {e}")
            try:
```

```
                os.remove(self.temp_video_path)
                os.remove(self.temp_audio_path)
            except:
                pass
            return False

    def toggle_fullscreen(self):
        """切换全屏模式"""
        self.is_fullscreen = not self.is_fullscreen
        self.root.attributes('-fullscreen', self.is_fullscreen)

        if self.is_fullscreen:
            # 全屏时隐藏控制面板
            self.control_frame.pack_forget()
        else:
            # 退出全屏时恢复控制面板到顶部
            self.control_frame.pack_forget()  # 先移除
            self.control_frame.pack(side=tk.TOP, fill=tk.X, before=self.video_label)  # 重新
放到视频上方

    def update_video_display(self):
        """更新视频显示（从队列获取帧）"""
        try:
            # 从队列获取帧
            try:
                frame = self.frame_queue.get_nowait()

                if frame is not None:
                    self.current_frame = frame

                    # 录制视频帧
                    if self.recording and self.video_writer:
                        self.video_writer.write(frame)

                    # 转换为 RGB 并调整大小以适应窗口
                    frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

                    # 获取视频标签的尺寸
                    label_width = self.video_label.winfo_width()
                    label_height = self.video_label.winfo_height()

                    if label_width > 1 and label_height > 1:
                        # 计算缩放比例，保持纵横比
                        frame_height, frame_width = frame_rgb.shape[:2]
```

```
                        scale = min(label_width / frame_width, label_height / frame_height)
                        new_width = int(frame_width * scale)
                        new_height = int(frame_height * scale)

                        frame_resized = cv2.resize(frame_rgb, (new_width, new_height))

                        # 转换为 PIL Image
                        img = Image.fromarray(frame_resized)
                        imgtk = ImageTk.PhotoImage(image=img)

                        self.video_label.imgtk = imgtk
                        self.video_label.configure(image=imgtk)

                except Empty:
                    # 队列为空，跳过此次更新
                    pass

        except Exception as e:
            print(f"显示错误: {e}")

        # 继续更新
        if self.running:
            self.root.after(10, self.update_video_display)

    def create_gui(self):
        """创建 GUI 界面"""
        self.root = tk.Tk()
        self.root.title(f"音视频接收端 - {self.server_ip}")
        self.root.geometry("900x650")
        self.root.minsize(600, 400)

        # 控制面板（先创建，放在顶部）
        self.control_frame = ttk.Frame(self.root, padding="10")
        self.control_frame.pack(side=tk.TOP, fill=tk.X)

        # 录制按钮
        self.record_btn = ttk.Button(
            self.control_frame,
            text="● 开始录制",
            command=self.toggle_recording,
            width=15
        )
        self.record_btn.pack(side=tk.LEFT, padx=5)
```

```
        # 全屏按钮
        fullscreen_btn = ttk.Button(
            self.control_frame,
            text="⛶ 全屏/退出全屏",
            command=self.toggle_fullscreen,
            width=15
        )
        fullscreen_btn.pack(side=tk.LEFT, padx=5)

        # 状态标签
        self.status_label = ttk.Label(
            self.control_frame,
            text="○ 未连接",
            font=('Arial', 10, 'bold')
        )
        self.status_label.pack(side=tk.LEFT, padx=20)

        # 退出按钮
        quit_btn = ttk.Button(
            self.control_frame,
            text="✕ 退出",
            command=self.quit_app,
            width=10
        )
        quit_btn.pack(side=tk.RIGHT, padx=5)

        # 视频显示区域（放在控制面板下方）
        self.video_label = tk.Label(self.root, bg='black')
        self.video_label.pack(fill=tk.BOTH, expand=True, padx=5, pady=5)

        # 绑定 ESC 键退出全屏
        self.root.bind('<Escape>', lambda e: self.toggle_fullscreen() if self.is_fullscreen else
None)

        # 窗口关闭事件
        self.root.protocol("WM_DELETE_WINDOW", self.quit_app)

    def toggle_recording(self):
        """切换录制状态"""
        if self.recording:
            self.record_btn.config(text="● 开始录制")
            self.status_label.config(text="⧗ 正在保存...", foreground="orange")
            self.stop_recording()
        else:
```

```
            self.record_btn.config(text="■ 停止录制")
            self.status_label.config(text="    录制中", foreground="red")
            self.start_recording()

    def quit_app(self):
        """退出应用"""
        if self.recording:
            if messagebox.askyesno("确认退出", "正在录制中，确定要退出吗？\n 录制的内容将不会保存。"):
                self.recording = False
            else:
                return

        self.running = False
        if self.root:
            self.root.quit()

    def run(self):
        """运行音视频显示（GUI 模式）"""
        if not self.connect():
            return

        print("\n 正在启动 GUI 界面...")

        self.running = True

        # 创建 GUI
        self.create_gui()

        # 更新状态
        self.status_label.config(text="    已连接", foreground="green")

        # 启动视频接收线程
        video_thread = threading.Thread(target=self.receive_video, daemon=True)
        video_thread.start()
        print("✓ 视频接收线程已启动")

        # 启动音频接收线程
        audio_thread = threading.Thread(target=self.receive_audio, daemon=True)
        audio_thread.start()
        print("✓ 音频接收线程已启动")

        # 启动视频更新（GUI 线程）
        self.root.after(100, self.update_video_display)
```

```
        # 启动 GUI 主循环
        try:
            self.root.mainloop()
        except KeyboardInterrupt:
            print("\n 检测到键盘中断...")
        finally:
            self.running = False
            video_thread.join(timeout=1)
            audio_thread.join(timeout=1)
            self.cleanup()

    def cleanup(self):
        """清理资源"""
        if self.audio_stream:
            try:
                self.audio_stream.stop_stream()
                self.audio_stream.close()
            except:
                pass
        if self.audio:
            self.audio.terminate()
        if self.video_socket:
            self.video_socket.close()
        if self.audio_socket:
            self.audio_socket.close()
        print("✓ 连接已关闭")


def main():
    print("=" * 60)
    print("音视频流接收端（客户端）")
    print("=" * 60)

    # 固定配置（直接修改这里）
    server_ip = "192.168.240.6"   # 服务器 IP
    video_port = 9999             # 视频端口
    audio_port = 9998             # 音频端口

    print(f"\n 连接配置:")
    print(f"   服务器 IP: {server_ip}")
    print(f"   视频端口: {video_port}")
    print(f"   音频端口: {audio_port}")
    print()
```

```
    client = AudioVideoStreamClient(server_ip, video_port, audio_port)
    client.run()


if __name__ == "__main__":
    main()
```