Instituto Politécnico Nacional

Unidad Profesional Interdisciplinaria en Ingeniería y Tecnologías Avanzadas

Asignatura: Programación Avanzada

Comunicación TCP/IP.

Integrantes:
Aboytes Arana Moises Antonio

Profesor:
Cruz Mora Jose Luis

18 de octubre de 2025

Índice

1. Objetivo

Crear aplicaciones que se comuniquen por TCP/IP.

2. Introducción

2.1. Hilos

Un hilo es una secuencia de instrucciones que puede ejecutarse de forma independiente dentro de un programa. Permite que una aplicación realice multitareas, ejecutando procesos de manera concurrente. Los hilos comparten el mismo espacio de memoria y recursos de su proceso principal, lo que facilita la comunicación y el intercambio de datos.

Pero, ¿qué diferencia hay entre un hilo y un proceso? Bueno, Mientras que un proceso es una instancia independiente de una aplicación con su propio espacio de memoria, un hilo es más ligero y comparte los recursos y la memoria de su proceso. En un solo proceso pueden existir múltiples hilos trabajando de manera concurrente, lo cual ahorra recursos y optimiza el rendimiento.

Algunas ventajas de ocupar hilos en los programas son:

- Mayor rendimiento: Los hilos pueden ejecutar tareas simultáneamente, aprovechando mejor los procesadores multinúcleo.
- Respuesta rápida: Mantienen la aplicación interactiva incluso cuando se ejecutan operaciones complejas, como cálculos intensivos o procesamiento de datos.
- Eficiencia en multitarea: Permiten manejar varias tareas dentro de una sola aplicación de forma eficiente.

En la mayoría de lenguajes de programación, puedes crear y gestionar hilos mediante APIs específicas o bibliotecas. Por ejemplo, en Python se usa el módulo threading.

Es importante señalar que un hilo no puede ejecutarse solo, requiere la supervisión de un proceso; por ello su ejecución está controlada por el contexto de un proceso donde se ejecuta.

2.1.1. Hilos demonio

Python proporciona dos tipos de hilos: subprocesos no daemon y daemon. De forma predeterminada, los subprocesos no son subprocesos de demonio. Estos subprocesos controlan tareas no críticas que pueden ser útiles para la aplicación, pero no la obstaculizan si fallan o se cancelan a mitad de la operación. Además, un hilo de demonio no tendrá control sobre cuándo se termina. El programa finalizará una vez que finalicen todos los subprocesos que no sean de demonio, incluso si todavía hay subprocesos de demonio ejecutándose en ese momento.
Los hilos Daemon tienen la prioridad más baja.

2.1.2. Multithreads

Un grupo de threads es un conjunto de "hilos de ejecución" que están corriendo todos dentro del mismo proceso. Dado que comparten todos la misma porción de memoria, pueden acceder a las mismas variables globales, los mismos descriptores de archivos, etc. Todos corren en paralelo. La ventaja de usar un grupo de threads en lugar de un programa normal en serie es que muchas operaciones pueden ser llevadas a cabo de forma paralela y, de esta forma, los eventos asociados a cada actividad pueden ser manejados inmediatamente tan pronto como llegan. Por otro lado, debido a que los threads dentro de un grupo comparten el mismo espacio de memoria, si uno de ellos corrompe el espacio de su memoria, los otros threads también sufrirán las consecuencias. Con un proceso, el sistema operativo normalmente protege a un proceso de otros y si un proceso corrompe su espacio de memoria los demás no se verán afectados.

Para crear varios hilos, es posible hacerlo con una lista y también es posible hacerlo con un ThreadPoolExecutor, disponible en la biblioteca concurrent.futures

2.2. Puertos

Ya sea un Arduino o un ESP32, no sólo pueden enviar señales sino que también pueden recibirlas con dos propósitos principales como son leer datos de sensores y recibir mensajes de otros dispositivos (shield, otro Arduino, PC, etc.). Las entradas se pueden clasificar en analógicas y digitales.

2.2.1. Entradas análogicas

Tomando como referencia Arduino, las entradas analógicas del modelo Uno son las correspondientes a los pines de A0 a A5. Se caracterizan por leer valores de tensión de 0 a 5 Voltios con una resolución de 1024 (10 bits). Si se divide 5 entre 1024, se tiene que ser capaz de detectar variaciones en el nivel de la señal de entrada de casi 5 mV.

2.2.2. Entradas digitales

Las entradas digitales son las mismas que las salidas digitales, es decir, los pines que van del 1 al 13. Se diferencian de las analógicas porque éstas son capaces de "entender" sólo dos niveles de señal, LOW o valores cercanos a 0 V y HIGH o valores cercanos a 5 V. Puede parecer una desventaja pero en realidad puede ser todo lo contrario. Y no sólo porque a veces únicamente se necesite saber dos estados (interruptor, pulsador, sensor de presencia, final de carrera,etc) sino porque así es capaz de leer señales de pulsos digitales.

2.3. Microcontroladores

No está de más hacer un breve repaso a dos de los microcontroladores más usados y útiles en la actualidad, mencionando sus principales características y ventajas de uso.

2.3.1. Arduino

Arduino es una plataforma famosa por su simplicidad y facilidad de uso, lo que la hace perfecta para principiantes y aficionados. Con un ecosistema completo que incluye tanto hardware

(varias placas) como software (Arduino IDE), proporciona todo lo necesario para comenzar.
Sus características principales son:

- Variedad de Tableros: Arduino ofrece una amplia gama de placas, desde la amigable para principiantes Arduino Uno hasta modelos más avanzados como el Arduino Mega y el Arduino Due.
- Facilidad de uso: El Arduino IDE es fácil de usar, y el amplio soporte de la comunidad facilita encontrar tutoriales y ayuda para la resolución de problemas.
- E/S Analógica y Digital: Las placas Arduino incluyen una mezcla de pines de E/S analógicos y digitales, lo que permite la compatibilidad con varios sensores, actuadores y otros componentes.

Entre sus principales ventajas se tienen:

- Fácil para principiantes: Fácil de aprender y usar, con una gran comunidad y abundantes recursos de aprendizaje.
- Amplia gama de placas: Las opciones están disponibles para adaptarse a diferentes niveles de complejidad y requisitos de potencia.

2.3.2. ESP32

El ESP32 es un microcontrolador increíble con capacidades integradas de Wi-Fi y Bluetooth. Desarrollado por el talentoso equipo de Espressif Systems, este pequeño gigante se ha convertido rápidamente en una opción popular para proyectos de IoT (Internet of Things) debido a sus excelentes características de conectividad y rendimiento confiable.
Sus características principales son:

- Conectividad: Una de las características destacadas del ESP32 es su Wi-Fi y Bluetooth integrados (compatible con Classic y BLE), lo que lo hace ideal para proyectos que requieren comunicación inalámbrica.
- Memoria: Ofrece hasta 520 KB de RAM y hasta 4 MB de memoria flash, soportando aplicaciones complejas con facilidad.

Entre sus principales ventajas se tienen:

- Alto Rendimiento: El procesador de doble núcleo y la amplia memoria le permiten manejar tareas exigentes de manera eficiente.
- Conectividad Versátil: El Wi-Fi y Bluetooth integrados amplían significativamente sus posibles casos de uso.

2.4. Comunicación TCP/IP

El protocolo TCP/IP (Transmission Control Protocol / Internet Protocol) es uno de los modelos de comunicación más utilizados en sistemas distribuidos debido a su confiabilidad, escalabilidad y capacidad de operar sobre redes locales y remotas. TCP/IP proporciona un mecanismo de comunicación orientado a conexión, el cual garantiza la entrega ordenada de los datos, así como la detección y corrección de errores durante la transmisión.

En el presente proyecto, la comunicación TCP/IP se utiliza para establecer un enlace cliente-servidor entre una computadora personal y un microcontrolador ESP32. A través de esta conexión, se envían comandos de control desde la interfaz gráfica hacia el microcontrolador y, de manera simultánea, se reciben datos provenientes de sensores, como la temperatura ambiental. Este esquema permite una comunicación bidireccional en tiempo real, fundamental para aplicaciones de automatización y monitoreo.

El uso de TCP/IP facilita además la implementación de un servidor intermedio que actúa como puente entre distintos dispositivos, permitiendo que el sistema pueda ampliarse para incluir múltiples clientes o nodos adicionales sin modificar de manera significativa la arquitectura general. De esta forma, TCP/IP se convierte en una solución robusta y flexible para el desarrollo de sistemas de control remoto y casas inteligentes.

2.5. Programa

2.5.1. Mi SmartHome

Con esta aplicación podremos controlar nuestros dispositivos inteligentes del hogar a través de una GUI desde una computadora. Se deberá tener mínimo 4 dispositivos distintos, como focos, timbres o alarmas, cerraduras eléctricas (puede usar motores o solenoides electromagnéticos), monitoreo de temperatura, monitoreo de lluvia, sensores de movimiento, cámaras de vigilancia, entre otros. Los dispositivos inteligentes deben conectarse a internet y recibir y enviar información a través de sockets.

3. Desarrollo

3.1. Programa. Mi SmartHome

Se desarrolló un sistema de casa inteligente que contó con los siguientes componentes: un sensor de temperatura, un servomotor para abrir y cerrar la puerta, leds cuya función es simular focos y un buzer que hizo de alarma, esta se activa en dos casos, uno es al forzar la puerta y el segundo cuando la temperatura rebase 35 grados centígrados, el sistema se centró principalmente en dos componentes de software: el servidor de comunicación y la aplicación principal con interfaz gráfica, los cuales se describen a continuación.

3.1.1. Servidor de Comunicación

El servidor funciona como un intermediario entre la interfaz gráfica y el ESP32. Su función principal es aceptar múltiples conexiones TCP y reenviar mensajes entre los clientes conectados, permitiendo que la computadora y el dispositivo embebido se comuniquen de manera indirecta.

Cada cliente que se conecta al servidor se identifica mediante un mensaje inicial que indica su tipo (PC o ESP32). Esta identificación permite al servidor distinguir el origen de los mensajes y reenviarlos correctamente al destinatario correspondiente.

El servidor permanece escuchando en un puerto específico y maneja cada conexión de manera concurrente, asegurando que la comunicación sea continua y estable durante la ejecución del sistema.

```python
import socket
import threading
import traceback
import time

class Cliente:
    def __init__(self, conn, addr):
        self.conn = conn
        self.addr = addr
        self.tipo = None

clientes = []
lock = threading.Lock()

# ENVÍO GENERAL A TIPOS DE CLIENTES
def enviar_a(tipo, message):
    with lock:
        copia = list(clientes)

    for c in copia:
        if c.tipo == tipo:
            try:
                c.conn.send((message + "\n").encode())
            except:
                try:
                    c.conn.close()
                except:
                    pass
                with lock:
                    if c in clientes:
                        clientes.remove(c)


# MANEJO DE CLIENTE
def manejar_cliente(cliente: Cliente):
    conn = cliente.conn
    addr = cliente.addr

    print(f"[+] Nueva conexión: {addr}")
    try:
        conn.send(b"<identify>\n")
```

Figura 1: Código del servidor TCP utilizado en el sistema

3.1.2. Interfaz Gráfica Principal

La aplicación principal fue desarrollada en QT Designer y se exporto a Python utilizando la biblioteca PySide6, lo que permitió crear una interfaz gráfica intuitiva y funcional. Desde esta

interfaz, el usuario puede interactuar con los distintos elementos de la casa inteligente.

La interfaz se encarga de establecer la conexión con el servidor TCP y mantener una comunicación constante para el envío de comandos y la recepción de datos provenientes del ESP32.

Figura 2: Interfaz grafica diseñada en Qt Designer.

3.1.3. Gestión de Comunicación

Para evitar bloqueos en la interfaz gráfica, la recepción de mensajes se implementó en un hilo independiente. Este hilo escucha continuamente los datos recibidos desde el servidor y emite señales hacia la interfaz principal cuando se recibe información relevante, como lecturas de temperatura.

El uso de un socket con tiempo de espera permitió que el hilo se cerrara correctamente al finalizar la aplicación, evitando errores comunes relacionados con hilos activos al cerrar la interfaz.

```python
class ReceiverThread(QThread):
    received = Signal(str)

    def __init__(self, sock):
        super().__init__()
        self.sock = sock
        self.running = True

    def run(self):
        buffer = b""
        while self.running:
            try:
                data = self.sock.recv(4096)
                if not data:
                    break

                buffer += data
                while b"\n" in buffer:
                    line, buffer = buffer.split(b"\n", 1)
                    msg = line.decode(errors="ignore").strip()
                    if msg:
                        self.received.emit(msg)

            except socket.timeout:
                continue
            except:
                break

    def stop(self):
        self.running = False
        self.quit()
        self.wait()

```

Figura 3: Implementación del hilo receptor para la comunicación TCP

3.1.4. Monitoreo de Temperatura

La interfaz gráfica muestra en tiempo real la temperatura medida por el sensor conectado al ESP32. Cada vez que se recibe un mensaje con el valor de temperatura, este se procesa y se actualiza el indicador visual correspondiente.

Cuando la temperatura supera un umbral predefinido, el sistema activa una alerta visual y sonora, notificando al usuario sobre una posible condición de riesgo.


```
// ================== TEMPERATURA ==================
unsigned long now = millis();
if (now - lastSend > 2000) {
  lastSend = now;
  float t = dht.readTemperature();

  if (!isnan(t)) {
    client.print(String("<esp><temp>") + t + "\n");

    if (t > 27 && !buzzerActivo) {
      buzzerActivo = true;
      digitalWrite(buzzer, HIGH);
      client.print("<esp><alarm>TEMP\n");
    }
  }
}
```

Figura 4: Codigo y visualización de la temperatura

3.1.5. Control de Iluminación

La interfaz permite encender y apagar focos de distintas áreas de la casa de manera individual o general. Para ello, se utilizan ventanas emergentes que permiten al usuario seleccionar la habitación deseada.

Los comandos seleccionados se envían al servidor, el cual los retransmite al ESP32 para ejecutar la acción física correspondiente.

3.1.6. Control de Puerta Automática

El sistema incluye el control de una puerta automática, cuyo estado se representa visualmente mediante una barra de progreso. Esta barra simula el tiempo de apertura y cierre de la puerta, proporcionando retroalimentación visual al usuario.

La lógica implementada evita movimientos innecesarios, ya que la puerta solo responde a comandos válidos según su estado actual.

```
156     # ================ PUERTA ================
157     def abrir_puerta(self):
158         if self.modo_seguro:
159             self.mostrar_alarma("Modo seguro activo: acceso denegado")
160             self.enviar("BUZZER_FORZADO")
161             return
162 
163         if self.puerta_abierta or self.animando:
164             return
165 
166         self.animando = True
167         self.progress_value = 0
168         self.ui.Display.setText("Abriendo puerta...")
169         self.enviar("PUERTA_ABRIR")
170         self.progressTimer.start(30)
171 
172     def cerrar_puerta(self):
173         if not self.puerta_abierta or self.animando:
174             return
175 
176         self.animando = True
177         self.progress_value = 100
178         self.ui.Display.setText("Cerrando puerta...")
179         self.enviar("PUERTA_CERRAR")
180         self.progressTimer.start(30)
181 
182     def animar_puerta(self):
183         paso = 100 / 50
184 
185         if self.ui.Display.text().startswith("Abriendo"):
186             self.progress_value += paso
187             if self.progress_value >= 100:
188                 self.progress_value = 100
189                 self.puerta_abierta = True
190                 self.animando = False
191                 self.progressTimer.stop()
192         else:
193             self.progress_value -= paso
194             if self.progress_value <= 0:
195                 self.progress_value = 0
196                 self.puerta_abierta = False
197                 self.animando = False
198                 self.progressTimer.stop()
199 
200         self.ui.Progreso_puerta.setValue(int(self.progress_value))
```

Figura 5: Código implementado para la puerta

3.1.7. Modo Seguro

El modo seguro es una función diseñada para restringir el acceso cuando el sistema se encuentra en un estado de alerta. Al activarse:

- Se bloquea la apertura de la puerta.
- Se genera una alerta visual y sonora si se intenta acceder.
- Si la puerta está abierta, se cierra automáticamente.

```
202     # ================= MODO SEGURO =================
203     def activar_modo_seguro(self):
204         self.modo_seguro = True
205 
206         # cerrar puerta si estaba abierta
207         if self.puerta_abierta and not self.animando:
208             self.animando = True
209             self.progress_value = 100
210             self.ui.Display.setText("errando puerta por modo seguro...")
211             self.enviar("PUERTA_CERRAR")
212             self.progressTimer.start(30)
213         else:
214             self.ui.Display.setText(" Modo seguro ACTIVADO")
215 
216     def desactivar_modo_seguro(self):
217         self.modo_seguro = False
218         self.ui.Display.setText("Modo seguro DESACTIVADO")
```

Figura 6: Código implementado para el modo seguro

4. Conclusion

4.1. Aboytes Arana Moises Antonio

Esta práctica me permitió reforzar conceptos fundamentales de programación avanzada, comunicación entre procesos, manejo de interfaces gráficas y sistemas embebidos, demostrando la importancia de integrar diferentes tecnologías para el desarrollo de soluciones completas y funcionales en aplicaciones de automatización y control, además se logró un sistema, integrando software y hardware mediante una arquitectura distribuida basada en comunicación TCP/IP. La correcta interacción entre la interfaz gráfica, el servidor y los dispositivos embebidos permitió el control y monitoreo de distintos elementos del sistema. La importancia de una interfaz gráfica fácil de entender para su uso, facilitó la interacción del usuario con el sistema, permitiendo enviar comandos de manera intuitiva y visualizar información relevante en tiempo real. Asimismo, la implementación de hilos y temporizadores garantizó un comportamiento responsivo de la aplicación, evitando bloqueos y asegurando una correcta gestión de los procesos concurrentes. Por otro lado, la comunicación en red permitió separar las responsabilidades del sistema, delegando el control físico de los dispositivos a la unidad embebida, mientras que el servidor

actuó como intermediario para el intercambio de información. Esta separación contribuyó a una arquitectura más ordenada, escalable y fácil de mantener.

Bibliografía

[1] “Hilo en computación: ¿Qué es? — Lenovo México”. Accedido el 16 de noviembre de 2025. [En línea]. Disponible en: https://www.lenovo.com/mx/es/glosario/hilo/

[2] “Hilos”. Unidades de Apoyo para el Aprendizaje - CUAED - UNAM. Accedido el 16 de noviembre de 2025. [En línea]. Disponible en: https://repositoriouapa.cuaed.unam.mx/repositorio/moodle/pluginfile.php/3084/mod_resource/content/1/UAPA-Hilos/index.html

[3] “Python - Daemon Threads”. Free Tutorials on Technical and Non Technical Subjects. Accedido el 16 de noviembre de 2025. [En línea]. Disponible en: https://www.tutorialspoint.com/python/python_daemon_threads.htm

[4] FCC - Facultad de Ciencias de la Computación BUAP. Accedido el 16 de noviembre de 2025. [En línea]. Disponible en: https://www.cs.buap.mx/~mceron/cap2_dis.pdf

[5] “Multithread”. UNIVERSIDAD AUTONOMA DEL ESTADO DE HIDALGO. Accedido el 16 de noviembre de 2025. [En línea]. Disponible en: https://sakuhina.tripod.com/id11.html

[6] J. L. Cruz Mora. “Hilos”. Classroom. Accedido el 16 de noviembre de 2025. [En línea]. Disponible en: https://drive.google.com/file/d/11tPujCUpnpZ1VgUVSwQlkLP6nF2X8_Qo/view?hl=es

[7] Guillermo Perez. “Tutorial Arduino: Entradas AnalÃ³gicas y Digitales — OpenWebinars”. OpenWebinars.net. Accedido el 16 de noviembre de 2025. [En línea]. Disponible en: https://openwebinars.net/blog/tutorial-arduino-entradas-analogicas-y-digitales/

[8] “ESP32 vs Arduino vs Raspberry Pi Pico: ¿Cuál es mejor?” OpenELAB Technology Ltd. Accedido el 16 de noviembre de 2025. [En línea]. Disponible en: https://openelab.io/es/blogs/learn/esp32-vs-arduino-vs-raspberry-pi-pico-which-is-better?srsltid=AfmB\OopgVyK4bXqL88VSd21CFQPMQ65MAcvXtbDMBlXHLA3IHOBfT70z