Estudiante: Adrian Peñaloza Ortega
DOC: 1005060452
Redes
Universidad de Pamplona
Pamplona - Colombia
2025

Documentación

Para server.py

```python
# server.py
import socket, threading

HOST = "0.0.0.0"
PORT = 5000
BACKLOG = 100
TIMEOUT_S = 30

ALLOWED_OPS = {"+", "-", "*", "/"}

def compute(op, a, b):
    if op not in ALLOWED_OPS:
        return None, ("BAD_OP", f"Operador no permitido: {op}")
    try:
        a = float(a)
        b = float(b)
    except ValueError:
        return None, ("BAD_NUM", "Operandos deben ser números (usa punto decimal).")
    if op == "/" and b == 0.0:
        return None, ("DIV_ZERO", "División por cero.")
    if op == "+":  res = a + b
    elif op == "-": res = a - b
    elif op == "*": res = a * b
    else:           res = a / b
    # Limita decimales para legibilidad
    return (f"{res:.10g}"), None

def process_line(msg: str) -> str:
    if msg == "PING":
        return "PONG"
    parts = msg.split()
    if len(parts) == 4 and parts[0] == "OP":
        _, op, op1, op2 = parts
        result, err = compute(op, op1, op2)
        if err:
            code, text = err
            return f"ERR {code} {text}"
        return f"OK {result}"
    return "ERR BAD_FORMAT Usa: OP <+|-|*|/> <num1> <num2> o PING"
```

```python
def handle_client(conn, addr):
    conn.settimeout(TIMEOUT_S)
    with conn:
        try:
            buf = b""
            while True:
                chunk = conn.recv(1024)
                if not chunk:
                    break
                buf += chunk
                while b"\n" in buf:
                    raw, buf = buf.split(b"\n", 1)
                    msg = raw.decode("utf-8", errors="replace").strip()
                    if not msg:
                        continue
                    print(f"[RX {addr}] {msg}")
                    reply = process_line(msg)
                    print(f"[TX {addr}] {reply}")
                    conn.sendall((reply + "\n").encode("utf-8"))
        except socket.timeout:
            print(f"[TIMEOUT] {addr}")
        except Exception as e:
            print(f"[ERROR {addr}] {e}")

def main():
    print(f"[BOOT] Calculadora TCP en {HOST}:{PORT} (F2+)")
    with socket.create_server((HOST, PORT), backlog=BACKLOG, reuse_port=True) as srv:
        while True:
            conn, addr = srv.accept()
            print(f"[ACCEPT] {addr}")
            threading.Thread(target=handle_client, args=(conn, addr), daemon=True).start()

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

```python
# server.py
import socket, threading

HOST = "0.0.0.0"
PORT = 5000
BACKLOG = 100
TIMEOUT_S = 30
ALLOWED_OPS = {"+", "-", "*", "/"}
```

socket: la librería de Python para hablar por red.

threading: para atender a varios clientes a la vez sin bloquear.

HOST = "0.0.0.0": escucha en todas las tarjetas de red.

PORT = 5000: numero de puerto para recibir las peticiones.

BACKLOG = 100: tamaño de la cola para clientes pendientes.

TIMEOUT_S = 30: tiempo maximo que esperan los clientes.

ALLOWED_OPS = {"+", "-", "*", "/"}: operadores validos para hacer la suma (+) resta (-) multiplicación (*) y división (/).

```python
def compute(op, a, b):
    if op not in ALLOWED_OPS:
        return None, ("BAD_OP", f"Operador no permitido: {op}")
    try:
        a = float(a)
        b = float(b)
    except ValueError:
        return None, ("BAD_NUM", "Operandos deben ser números (usa punto decimal).")
    if op == "/" and b == 0.0:
        return None, ("DIV_ZERO", "División por cero.")
    if op == "+":  res = a + b
    elif op == "-": res = a - b
    elif op == "*": res = a * b
    else:           res = a / b
    # Limita decimales para legibilidad
    return (f"{res:.10g}"), None
```

Entradas: op (el operador como texto: + - * /), a y b (llegan como **texto**).

Salidas: una tupla:

- **Éxito:** ("resultado_como_texto", None)
- **Error:** (None, ("CODIGO", "Mensaje claro"))

Este formato evita reventar la conexión con excepciones; el servidor puede responder OK … o ERR … según lo que reciba.

if op not in ALLOWED_OPS:

return None, ("BAD_OP", f"Operador no permitido: {op}")

- Solo aceptamos **+ - * /**.
- Si te equivocas (por ejemplo x), devolvemos BAD_OP.

try:

a = float(a)

b = float(b)

except ValueError:

return None, ("BAD_NUM", "Operandos deben ser números (usa punto decimal).")

- lo que llega por red es texto; aquí lo pasamos a **float**.
- Si mandas letras o usas **coma** decimal (3,5), cae en BAD_NUM.

if op == "/" and b == 0.0:

return None, ("DIV_ZERO", "División por cero.")

- Evitamos que se reviente al dividir entre 0.

if op == "+": res = a + b

elif op == "-": res = a - b

elif op == "*": res = a * b

else: res = a / b

- Hacemos la operacion, dependiendo del caso que se este usando (suma, resta, multiplicación y division).

return (f"{res:.10g}"), None

- Le damos formato por si es un numero periódico (ejemplo: 1/3).

```python
def process_line(msg: str) -> str:
    if msg == "PING":
        return "PONG"
    parts = msg.split()
    if len(parts) == 4 and parts[0] == "OP":
        _, op, op1, op2 = parts
        result, err = compute(op, op1, op2)
        if err:
            code, text = err
            return f"ERR {code} {text}"
        return f"OK {result}"
    return "ERR BAD_FORMAT Usa: OP <+|-|*|/> <num1> <num2> o PING"
```

Entrada: cadena limpia, sin espacios en blanco, puede ser:

- “PING”
- “OP <operador> <numero1> <numero2>"

```
if msg == "PING":
    return "PONG"
```

Si entra “PING”, debe responder “PONG” (esto es para sanidad, veficiar que si responde algo sencillo).

```
parts = msg.split()
    if len(parts) == 4 and parts[0] == "OP":
        _, op, op1, op2 = parts
        result, err = compute(op, op1, op2)
        if err:
            code, text = err
            return f"ERR {code} {text}"
        return f"OK {result}"
    return "ERR BAD_FORMAT Usa: OP <+|-|*|/> <num1> <num2> o PING"
```

- Toma la entrada “OP <operador> <numero1> <numero2>" y le aplica un Split obtenido algo como ["OP", "<operador>", "<numero1>", "<numero2>"]
- Valida que el contenido sea correcto
- Si el contenido es correcto llama la función compute, y realiza el calculo de la operación

---

```python
def handle_client(conn, addr):
    conn.settimeout(TIMEOUT_S)
    with conn:
        try:
            buf = b""
            while True:
                chunk = conn.recv(1024)
                if not chunk:
                    break
                buf += chunk
                while b"\n" in buf:
                    raw, buf = buf.split(b"\n", 1)
                    msg = raw.decode("utf-8", errors="replace").strip()
                    if not msg:
                        continue
                    print(f"[RX {addr}] {msg}")
                    reply = process_line(msg)
                    print(f"[TX {addr}] {reply}")
                    conn.sendall((reply + "\n").encode("utf-8"))
        except socket.timeout:
            print(f"[TIMEOUT] {addr}")
        except Exception as e:
            print(f"[ERROR {addr}] {e}")
```

- **conn.settimeout(TIMEOUT_S)**: Si el cliente se queda callado más de TIMEOUT_S segundos, cerramos, esto con el objetivo de evitar conexiones zombis que se quedan abiertas para siempre.
- **with con ...**: Asegura cierre limpio del socket al terminar
- **buf = b""**: Búfer para ir armando líneas completas
- **chunk = conn.recv(1024)**: Lee hasta 1024 bytes del cliente
- **if not chunk**: Si el cliente cerró, salimos
- **buf += chunk**: Acumula en el búfer
- **raw, buf = buf.split(b"\n", 1)**: Toma una línea y deja el resto en buf
- **msg = raw.decode("utf-8", errors="replace").strip()**: Decodifica y limpia espacios/CR
- **if not msg**: Ignora líneas vacías
- **print(f"[RX {addr}] {msg}")**: Log de lo recibido
- **reply = process_line(msg)**: imterpreta y resuelve (PING/OP ...)
- **print(f"[TX {addr}] {reply}")**: Log de lo que respondeiremos
- **conn.sendall((reply + "\n").encode("utf-8"))**: Envía respuesta con salto de línea
- **print(f"[TIMEOUT] {addr}")**: El cliente no habló a tiempo
- **print(f"[ERROR {addr}] {e}")**: Cualquier otro error controlado en logs

Para client.py

```python
# client.py
import socket, sys

def ask_float(prompt):
    while True:
        txt = input(prompt).strip().replace(",", ".")
        try:
            return float(txt)
        except ValueError:
            print("Valor inválido. Usa números y punto decimal (ej: 3.5).")

def ask_op():
    while True:
        op = input("Operación (+, -, *, /): ").strip()
        if op in {"+", "-", "*", "/"}:
            return op
        print("Operación inválida.")

def main():
    if len(sys.argv) < 2:
        print("Uso: python client.py <IP_SERVIDOR> [PUERTO]")
        sys.exit(1)
    host = sys.argv[1]
    port = int(sys.argv[2]) if len(sys.argv) >= 3 else 5000

    a = ask_float("Primer número: ")
    b = ask_float("Segundo número: ")
    op = ask_op()

    msg = f"OP {op} {a} {b}\n"
    try:
        with socket.create_connection((host, port), timeout=5) as s:
            s.sendall(msg.encode("utf-8"))
            data = s.recv(1024).decode("utf-8", errors="replace").strip()
            if data.startswith("OK "):
                print("Resultado:", data[3:].strip())
            elif data.startswith("ERR "):
                print("Error del servidor:", data[4:].strip())
            else:
                print("Respuesta desconocida:", data)
    except TimeoutError:
        print("Tiempo de conexión agotado.")
    except OSError as e:
        print("Error de conexión:", e)

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

```python
def ask_float(prompt):
    while True:
        txt = input(prompt).strip().replace(",", ".")
        try:
            return float(txt)
        except ValueError:
            print("Valor inválido. Usa números y punto decimal (ej: 3.5).")
```

- convierte una coma a un punto, si es posible, si no se lanza una excepción.

```python
def ask_op():
    while True:
        op = input("Operación (+, -, *, /): ").strip()
        if op in {"+", "-", "*", "/"}:
            return op
        print("Operación inválida.")
```

- verifica que el operador sea valido

```python
def main():
    # Uso básico: python client.py <IP_SERVIDOR> [PUERTO]
    if len(sys.argv) < 2:
        print("Uso: python client.py <IP_SERVIDOR> [PUERTO]")
        sys.exit(1)
    host = sys.argv[1]
    port = int(sys.argv[2]) if len(sys.argv) >= 3 else 5000  # Por defecto 5000

    # Pide datos al usuario
    a = ask_float("Primer número: ")
    b = ask_float("Segundo número: ")
    op = ask_op()

    # Arma el mensaje del protocolo (línea que termina en \n)
    msg = f"OP {op} {a} {b}\n"

    try:
        # Abre conexión TCP al servidor (con timeout cortico)
        with socket.create_connection((host, port), timeout=5) as s:
            s.sendall(msg.encode("utf-8"))                      # Envía la operación
            data = s.recv(1024).decode("utf-8", errors="replace").strip()  # Lee respuesta
            # Interpreta respuesta del servidor
            if data.startswith("OK "):
                print("Resultado:", data[3:].strip())
            elif data.startswith("ERR "):
                print("Error del servidor:", data[4:].strip())
            else:
                print("Respuesta desconocida:", data)
        # El with cierra la conexión automáticamente al terminar
    except TimeoutError:
        print("Tiempo de conexión agotado.")
    except OSError as e:
        print("Error de conexión:", e)
```