# 📖 Guía de Ejercicios Previos - Práctica 5

## ⚠️ IMPORTANTE

**Debes completar estos ejercicios ANTES de hacer la práctica 5**. Son fundamentales para entender cómo funciona la comunicación mediante sockets.

---

## 🎯 Ejercicio 1: Sistema Cliente-Servidor Básico

### Objetivo

Aprender los fundamentos de la comunicación mediante sockets con un modelo simple donde:

- Un **servidor** escucha en un puerto
- Un **cliente** se conecta al servidor
- Intercambian mensajes de forma síncrona

### Archivos Involucrados

- `Cliente.cpp` - Proceso cliente
- `Servidor.cpp` - Proceso servidor (atiende a UN solo cliente)
- `Socket/Socket.cpp` y `Socket/Socket.hpp` - Librería de sockets

### Funcionalidad

El cliente envía frases al servidor, y el servidor cuenta cuántas vocales hay en cada frase.

### Compilación

```bash
make ejercicios
```

Esto genera los ejecutables:

- `Cliente`
- `Servidor`
- `ServidorMulticliente`

---

# 🔧 Ejercicio 1.1: Ejecución en Local

### Paso 1: Iniciar el Servidor

```bash
# Terminal 1
./Servidor 3000
```

**Salida esperada:**

```
Servidor escuchando en puerto 3000
Bind realizado correctamente
Esperando conexión de un cliente...
```

### Paso 2: Conectar un Cliente

```bash
# Terminal 2
./Cliente localhost 3000
```

**Salida esperada del cliente:**

```
Intentando conectar a localhost:3000
Conectado al servidor correctamente
Escribe frases para contar vocales ('END OF SERVICE' para terminar)
Frase para contar las vocales:
```

**Salida esperada del servidor:**

```
Cliente conectado correctamente
```

### Paso 3: Enviar Mensajes

**En el cliente, escribe:**

```
Frase para contar las vocales: Hola mundo
```

**Salida del cliente:**

```
Mensaje enviado: 'Hola mundo'
Numero de vocales: 4
```

**Salida del servidor:**

```
Mensaje recibido: 'Hola mundo' (10 bytes)
Mensaje enviado: '4' (1 bytes)
```

**Paso 4: Probar con Más Mensajes**

```
Frase para contar las vocales: Programacion de sistemas concurrentes
Mensaje enviado: 'Programacion de sistemas concurrentes'
Numero de vocales: 15

Frase para contar las vocales: Universidad de Zaragoza
Mensaje enviado: 'Universidad de Zaragoza'
Numero de vocales: 10
```

**Paso 5: Finalizar**

```
Frase para contar las vocales: END OF SERVICE
Bye bye
```

**El servidor también termina:**

```
Mensaje de finalización recibido
Cerrando cliente: 4
Cerrando servidor: 3
Bye bye
```

---

## 🌐 Ejercicio 1.2: Ejecución en Red (Distribuido)

Este ejercicio requiere **DOS ordenadores** en la misma red.

**Máquina A (Servidor)**

**Paso 1: Obtener la IP**

bash

```
# En Linux
ip addr show
# Busca algo como: inet 192.168.1.100/24

# En hendrix
host hendrix01.cps.unizar.es
# Ejemplo: 155.210.154.205
```

**Paso 2: Iniciar el Servidor**

```bash
./Servidor 3000
```

**Máquina B (Cliente)**

**Conectar usando la IP del servidor:**

```bash
./Cliente 192.168.1.100 3000
# o
./Cliente 155.210.154.205 3000
```

**Pruebas a Realizar**

1. **Enviar mensajes desde cliente remoto**
   - Verifica que el servidor recibe y responde correctamente
2. **Prueba con compañeros**
   - Pide a un compañero que ejecute el cliente contra tu servidor
   - O viceversa
3. **Verifica errores comunes:**
   - ¿Qué pasa si el servidor no está iniciado?
   - ¿Qué pasa si usas una IP incorrecta?
   - ¿Qué pasa si el puerto está ocupado?

---

## 👥 Ejercicio 2: Servidor Multicliente

**Objetivo**

Aprender a manejar **múltiples clientes simultáneamente** usando threads. Cada cliente es atendido por un thread independiente.

**Diferencias con Ejercicio 1**

- **Ejercicio 1**: Servidor atiende a UN solo cliente y luego termina
- **Ejercicio 2**: Servidor atiende a MÚLTIPLES clientes concurrentemente

**Archivos Involucrados**

- `Cliente.cpp` - Mismo cliente del ejercicio 1

- `ServidorMulticliente.cpp` - Servidor mejorado con threads

---

## 🔧 Ejercicio 2.1: Ejecución Local con Múltiples Clientes

### Paso 1: Iniciar el Servidor

```bash
# Terminal 1
./ServidorMulticliente 3000
```

### Salida esperada:

```
Servidor multicliente escuchando en puerto 3000
Bind realizado correctamente
Servidor esperando clientes...
(El servidor se cerrará automáticamente después de 60 segundos sin actividad)
```

### Paso 2: Conectar Primer Cliente

```bash
# Terminal 2
./Cliente localhost 3000
```

### Salida del servidor:

```
Nuevo cliente 1 aceptado
Thread para cliente 1 iniciado
```

### Paso 3: Conectar Segundo Cliente (SIN cerrar el primero)

```bash
# Terminal 3
./Cliente localhost 3000
```

### Salida del servidor:

```
Nuevo cliente 2 aceptado
Thread para cliente 2 iniciado
```

### Paso 4: Conectar Tercer Cliente

```bash
```

```
# Terminal 4
./Cliente localhost 3000
```

**Salida del servidor:**

```
Nuevo cliente 3 aceptado
Thread para cliente 3 iniciado
```

**Paso 5: Enviar Mensajes desde Diferentes Clientes**

**Cliente 1:**

```
Frase para contar las vocales: Cliente numero uno
```

**Servidor muestra:**

```
Cliente 1 - Mensaje recibido: 'Cliente numero uno'
Cliente 1 - Respuesta enviada: 8
```

**Cliente 2 (simultáneamente):**

```
Frase para contar las vocales: Cliente numero dos
```

**Servidor muestra:**

```
Cliente 2 - Mensaje recibido: 'Cliente numero dos'
Cliente 2 - Respuesta enviada: 8
```

**Cliente 3:**

```
Frase para contar las vocales: Tercer cliente conectado
```

**Servidor muestra:**

```
Cliente 3 - Mensaje recibido: 'Tercer cliente conectado'
Cliente 3 - Respuesta enviada: 9
```

**Observaciones Importantes**

1. **Los clientes pueden enviar mensajes en cualquier orden**
2. **El servidor responde a todos concurrentemente**

3. **Cada cliente tiene su propio thread en el servidor**
4. **Los mensajes no se mezclan gracias a los threads**

---

## 🌐 Ejercicio 2.2: Prueba Distribuida con Compañeros

### Escenario

- **1 Servidor** en una máquina
- **Múltiples Clientes** en diferentes máquinas

### Máquina del Servidor

```bash
# Obtener IP
ip addr
# Ejemplo: 192.168.1.100

./ServidorMulticliente 3000
```

### Máquinas de los Clientes

**Estudiante 1:**

```bash
./Cliente 192.168.1.100 3000
```

**Estudiante 2 (diferente ordenador):**

```bash
./Cliente 192.168.1.100 3000
```

**Estudiante 3 (diferente ordenador):**

```bash
./Cliente 192.168.1.100 3000
```

### Actividad Conjunta

1. **Todos envían mensajes simultáneamente**
2. **Observa en el servidor cómo se gestionan múltiples clientes**
3. **Algunos clientes se desconectan (END OF SERVICE)**

4. **Nuevos clientes se conectan**
5. **El servidor sigue funcionando**

---

## 🧪 Experimentos Adicionales

**Experimento 1: ¿Qué pasa si un cliente se desconecta bruscamente?**

**Terminal Cliente:**

```bash
./Cliente localhost 3000
# Presiona Ctrl+C para matar el proceso
```

**Observa el servidor** - Debe detectar el error y cerrar el thread de ese cliente.

**Experimento 2: ¿Cuántos clientes puede manejar?**

```bash
# Conectar 10 clientes simultáneamente
for i in {1..10}; do
    ./Cliente localhost 3000 &
done
```

**Experimento 3: Timeout del Servidor**

**El servidor se cierra después de 60 segundos sin recibir nuevas conexiones.**

1. Inicia el servidor
2. NO conectes ningún cliente
3. Espera 60 segundos
4. El servidor debe cerrarse automáticamente

**Experimento 4: Puerto Ocupado**

**Terminal 1:**

```bash
./ServidorMulticliente 3000
```

**Terminal 2 (sin cerrar Terminal 1):**

```bash
```

```
./ServidorMulticliente 3000
```

**Resultado esperado:**

```
Error en el bind
```

**Solución:**

- Usa un puerto diferente: `./ServidorMulticliente 3001`
- O espera a que el primer servidor termine

---

## 📋 Checklist de Competencias

Antes de pasar a la Práctica 5, debes ser capaz de:

### Ejercicio 1

- [ ] Compilar Cliente y Servidor
- [ ] Ejecutar servidor en un puerto específico
- [ ] Conectar cliente a localhost
- [ ] Conectar cliente a IP remota
- [ ] Enviar y recibir mensajes
- [ ] Entender el flujo: Connect → Send → Receive → Close
- [ ] Manejar el mensaje de finalización

### Ejercicio 2

- [ ] Ejecutar servidor multicliente
- [ ] Conectar múltiples clientes simultáneamente
- [ ] Entender cómo se usan threads para cada cliente
- [ ] Ver mensajes de múltiples clientes intercalados
- [ ] Entender Accept en bucle
- [ ] Comprender la gestión de threads con vector<thread>

### Conceptos Clave

- [ ] Sockets TCP y comunicación síncrona
- [ ] Bind, Listen, Accept, Connect
- [ ] Send y Receive síncronos
- [ ] Threads para concurrencia
- [ ] IP y puertos
- [ ] localhost vs IP remota
- [ ] Manejo de errores en red

---

## 🚀 ¿Listo para la Práctica 5?

Si has completado y entendido estos ejercicios, estás preparado para:

1. **ServidorTareas**: Similar a ServidorMulticliente pero con buffer de tareas
2. **ServidorMatriz**: Similar a ServidorMulticliente pero con matriz de resultados
3. **Controlador**: Similar a Cliente pero se conecta a DOS servidores

La Práctica 5 combina estos conceptos con:

- Monitores de la Práctica 4
- Comunicación con múltiples servidores
- Sincronización compleja
- Protocolo de finalización coordinado

---

## ❓ Preguntas Frecuentes

**P: ¿Por qué el cliente no se conecta?** R: Verifica que:

- El servidor esté ejecutándose
- La IP sea correcta
- El puerto sea el mismo
- No haya firewall bloqueando

**P: ¿Por qué dice "Error en el bind"?** R: El puerto ya está en uso. Usa otro puerto o cierra el proceso que lo está usando.

**P: ¿Cómo sé mi IP?** R: `ip addr`, `ifconfig`, o `host nombre_maquina`

**P: ¿Qué significa "localhost"?** R: Es 127.0.0.1, la IP de la propia máquina (loopback).

**P: ¿Puedo usar otro puerto?** R: Sí, cualquier puerto > 1024 que no esté ocupado.

**P: ¿Por qué el servidor multicliente se cierra solo?** R: Tiene un timeout de 60 segundos para facilitar las pruebas. Puedes modificar el código para eliminar esta funcionalidad.

---

## 📚 Recursos Adicionales

- Manual de sockets POSIX
- Documentación de threads en C++11

- Práctica 4 (monitores y sincronización)
- Apuntes de teoría sobre cliente-servidor

---

¡Buena suerte con los ejercicios! Una vez completados, estarás listo para la Práctica 5. 🎓