Programación de Sistemas Distribuidos

Curso 2024/2025

Práctica 1

Diseño e implementación del juego conecta-4 en modo multijugador distribuido utilizando sockets

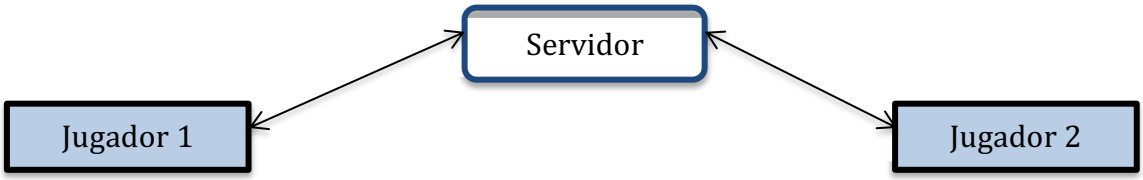
Esta práctica consiste en el diseño e implementación de un sistema de juegos on-line **utilizando sockets**. Concretamente, se pretende desarrollar tanto la parte cliente como el servidor para permitir partidas remotas del juego conecta-4 [1].

Para el desarrollo de esta práctica NO será necesario implementar la lógica del juego. El objetivo de la misma se centra en la parte encargada de las comunicaciones entre los clientes (jugadores) y el servidor.

La parte cliente será el programa que ejecuten los jugadores. Esta parte no se encarga, en ningún momento, de controlar la lógica del juego. Su función es mostrar por pantalla el estado del juego actual, esto es, el contenido del tablero. Además, recogerá los movimientos introducidos por el jugador.

La parte servidor controlará la lógica del juego (p.ej. controlar que no se realice un movimiento no válido, calcular cuándo se ha terminado la partida, etc…). Además, se encargará de sincronizar los movimientos de los jugadores involucrados en la partida.

Para que pueda dar comienzo una partida, será necesario que 2 clientes estén conectados al servidor. De lo contrario, la partida no podrá comenzar. El siguiente esquema muestra la arquitectura del juego.



En la comunicación entre cliente y servidor podemos definir 3 elementos que deberán transmitirse entre estas partes. El fichero `gameTypes.h` contiene los tipos utilizados para el desarrollo de la práctica. Seguidamente se describen los más relevantes.

- **Código**: Número que indica el estado de la partida, a través del cual, el cliente conocerá el turno del jugador y cuándo acaba la partida. Se recomienda utilizar el tipo `unsigned int`.
  - `TURN_MOVE`: Turno del jugador, puede realizar movimiento.
  - `TURN_WAIT`: El jugador debe esperar porque le toca mover al rival.
  - `GAMEOVER_WIN`: Jugador gana, fin de partida.
  - `GAMEOVER_DRAW`: Empate, fin de partida.
  - `GAMEOVER_LOSE`: Jugador pierde, fin de partida.

- **Mensaje**: Cadena de texto que contiene mensajes sobre el estado de la partida. Este mensaje se enviará en dos partes. La primera consiste en un número de 4 bytes (`unsigned int`) que contiene la longitud (en número de caracteres) del mensaje a enviar. La segunda parte consiste en el propio mensaje (`tString`), el cual está formado por una secuencia de caracteres.

- **Tablero** (`tBoard`): Tablero con el estado de la partida que consiste en un array de caracteres. Las casillas vacías se representan con el carácter `EMPTY_CELL`. Sin embargo, `PLAYER_1_CHIP` y `PLAYER_2_CHIP` representan los caracteres para mostrar las fichas del jugador 1 y jugador 2, respectivamente.

[1] https://es.wikipedia.org/wiki/Conecta_4

```
typedef char tBoard [BOARD_WIDTH * BOARD_HEIGHT];
```

El fichero `game.h` contiene las cabeceras de los subprogramas encargados de la lógica del juego. Además, este fichero contiene una descripción detallada de los parámetros de entrada y salida de cada uno, los cuales podrán ser invocados desde el programa servidor.

```
void initBoard (tBoard board);
```

Inicializa un tablero. Debe ser invocado antes de comenzar una partida.

```
tMove checkMove (tBoard board, unsigned int column);
```

Comprueba el movimiento de un jugador. El movimiento será el número [0 - `BOARD_WIDTH`) de la columna (`column`) donde insertar la ficha del jugador.

```
void insertChip (tBoard board, tPlayer player, unsigned int column);
```

Introduce una ficha en la columna indicada por el jugador, actualizando el tablero.

```
int checkWinner (tBoard board, tPlayer player);
```

Comprueba si el jugador actual es el ganador. Este subprograma deberá invocarse después de haber realizado el movimiento.

```
int isBoardFull (tBoard board);
```

Comprueba si el tablero está lleno. Esto es, ya no es posible realizar más movimientos.

1.- Implementación del servidor

El servidor se ejecutará recibiendo como parámetro, únicamente, el puerto donde realizará la escucha de las conexiones de los clientes.

Seguidamente, establecerá conexión con dos clientes (jugadores), asignando un socket a cada una de estas dos conexiones. A partir de ese momento, el servidor ya podrá comunicarse con los dos jugadores.

El fichero `serverGame.h` contiene las cabeceras de los subprogramas que deberán implementarse en el fichero `serverGame.c`. Algunos ya están implementados con el objetivo de facilitar el desarrollo de la práctica.

Sin embargo, otros deberéis implementarlos, como por ejemplo:

```
void sendMessageToPlayer (int socketClient, char* message);
```

```
void receiveMessageFromPlayer (int socketClient, char* message);
```

que se encargan de enviar/recibir un mensaje a/del jugador, respectivamente. El primer parámetro es el socket a través del cual se realiza la comunicación con el jugador correspondiente. Puesto que los mensajes se transmiten muy a menudo con los

jugadores, estos subprogramas ayudan a reducir el código fuente y aumentan la claridad del programa.

El servidor deberá ser capaz de mantener varias partidas simultáneamente. Para ello, será necesario el uso de *threads*. La estructura `tThreadArgs`, definida en el fichero `gameTypes.h`, puede utilizarse para gestionar la comunicación de cada partida. Con el fin de simplificar el desarrollo de esta parte, no será necesario limitar el número de conexiones con los clientes

Además, el servidor almacenará al vencedor y perdedor de las tres últimas partidas. En los empates no se almacena ninguna información. Cada vez que se finalice una partida, se actualizará la lista y se volcará el contenido de la misma a un fichero de texto, que contendrá exactamente la información almacenada en la lista.

2.- Implementación del cliente

La parte cliente se ejecuta recibiendo dos parámetros, la IP del servidor y el puerto donde éste permanece escuchando conexiones de los jugadores.

Cada cliente realizará una única conexión con el servidor, la cual se deberá cerrar una vez finalice la partida. Además, el cliente no controlará en ningún momento el estado del tablero. De esta forma, la lógica del juego se gestionará completamente en el servidor. La única comprobación que se realizará en el cliente, es que el movimiento del jugador sea un número entre el [0 - `BOARD_WIDTH`). Sin embargo, aspectos tales como comprobar si la columna introducida por el jugador está llena y no puede realizar el movimiento, se controlará en el servidor.

De forma similar al fichero `serverGame.h`, el fichero `clientGame.h` contiene las cabeceras de las funciones utilizadas por la parte cliente. El fichero `clientGame.c` deberá contener la implementación de estas funciones.

3.- Estructura de una partida

Una partida entre dos jugadores se compone de dos fases. La primera se corresponde con el intercambio de nombres, mientras que la segunda se corresponde con el desarrollo del juego. En la primera fase, cada jugador envía al servidor su nombre y, seguidamente, recibe el nombre del rival. El siguiente esquema detalla el paso de mensajes en esta fase.

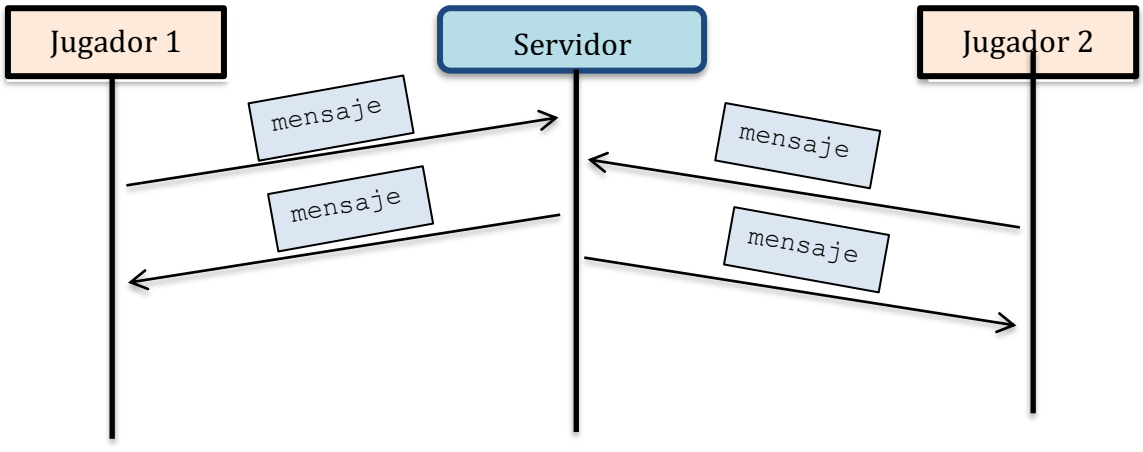


Ilustración 1. Fase 1: Intercambio de nombres

Cada mensaje, tal y como se comentó anteriormente, se compone de un número de 4 bytes (`unsigned int`) y una cadena de caracteres que contiene el texto con información sobre el estado de la partida.

Las siguientes capturas de pantalla detallan la información mostrada, al ejecutar la primera fase de la partida, en cada una de las partes.

Servidor

```
Albertos-MacBook-Pro:PSD_Prac1_solucionSockets cana$ ./serverGame 50001
Player 1 is connected!
Player 2 is connected!
Name of player 1 received: Homer Simpson
Name of player 2 received: Spiderman
```

Jugador 1

```
Albertos-MacBook-Pro:PSD_Prac1_solucionSockets cana$ ./clientGame 192.168.1.132 50001
Connection established with server!
Enter player name:Homer Simpson
You are playing against Spiderman
Game starts!
```

Jugador 2

```
Albertos-MacBook-Pro:PSD_Prac1_solucionSockets cana$ ./clientGame 192.168.1.132 50001
Connection established with server!
Enter player name:Spiderman
You are playing against Homer Simpson
Game starts!
```

A continuación, el servidor deberá calcular – aleatoriamente - quién de los dos jugadores empieza la partida. Una vez calculado el orden de los jugadores, da comienzo la segunda fase, que se corresponde con el desarrollo del juego hasta que la partida finaliza. A partir de este punto, cada vez que el servidor envía información al cliente, esta información estará formada por:

- Código: `unsigned int`.
- Mensaje: `unsigned int` indicando la longitud del mensaje y el propio mensaje.
- Tablero: `tBoard`.

Sin embargo, el cliente sólo enviará el movimiento realizado a través de un número de 4 bytes (`unsigned int`). Este número deberá ser, obligatoriamente, un número en el intervalo [0 - `BOARD_WIDTH`).

Supongamos que el servidor calcula que el jugador 1 empieza la partida. La información que le enviará al jugador 1 será:

- Code: TURN_MOVE
- Msg: 30 – Its your turn. You play with:o
- Board: (tablero vacío)

Mientras que al jugador 2 le llegará la siguiente información:

- Code: TURN_WAIT
- Msg: 55 – Your rival is thinking... please, wait! You play with:x
- Board: (tablero vacío)

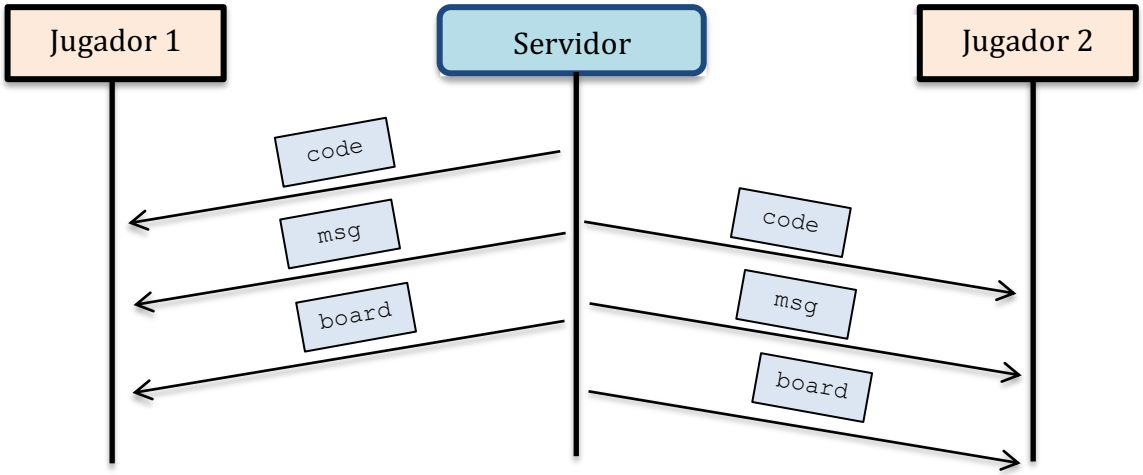


Ilustración 2. Fase 2: Desarrollo del juego

Una vez el servidor envía la información inicial a los jugadores, cada pantalla muestra lo siguiente:

Servidor

```
Albertos-MacBook-Pro:PSD_Prac1_solucionSockets cana$ ./serverGame 50001
Player 1 is connected!
Player 2 is connected!
Name of player 1 received: Homer Simpson
Name of player 2 received: Spiderman
```

Jugador 1

```
Albertos-MacBook-Pro:PSD_Prac1_solucionSockets cana$ ./clientGame 192.168.1.132 50001
Connection established with server!
Enter player name:Homer Simpson
You are playing against Spiderman
Game starts!


Its your turn. You play with:o

  0   1   2   3   4   5   6
|---|---|---|---|---|---|---|
|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
|---------------------------|


Enter a move [0-6]:
```

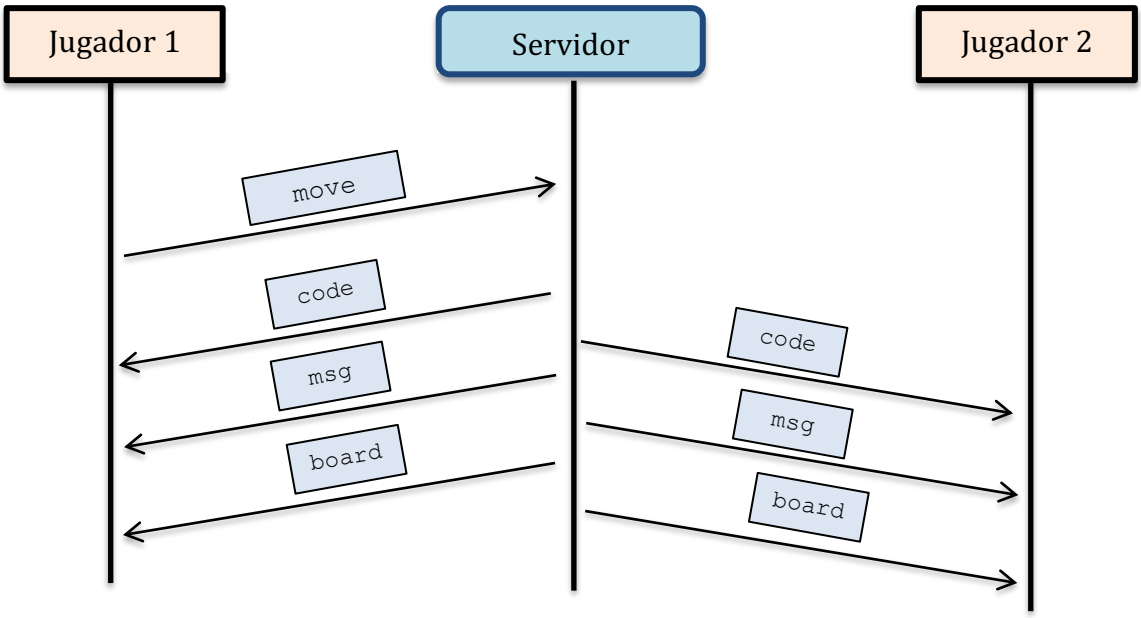
Jugador 2

```
Albertos-MacBook-Pro:PSD_Prac1_solucionSockets cana$ ./clientGame 192.168.1.132 50001
Connection established with server!
Enter player name:Spiderman
You are playing against Homer Simpson
Game starts!


Your rival is thinking... please, wait! You play with:x

  0   1   2   3   4   5   6
|---|---|---|---|---|---|---|
|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
|   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|
|   |   |   |   |   |   |   |
|---------------------------|
```

En este punto, tal y como puede apreciarse en las capturas de pantalla, es el turno del jugador 1 para realizar un movimiento. Supongamos que introduce un 3. El siguiente esquema muestra las comunicaciones realizadas:



Es importante destacar que una vez el jugador ha realizado un movimiento, el servidor envía información a las dos partes, ya que **el tablero se actualiza en el servidor, no en el cliente**. De esta forma, por cada movimiento realizado, el servidor envía (código + mensaje + tablero) a cada cliente.

La parte cliente no necesita manipular (ni debe) el tablero. Una vez reciba la información del servidor, deberá invocar

```
void printBoard (tBoard board, char* message);
```

para que se imprima por pantalla tanto el estado actual del tablero como el mensaje con la información de la partida.

Una vez procesado el movimiento por el servidor, se realizan las siguientes comprobaciones:

- Si el movimiento es correcto
  - Es fin de partida
    - Si gana el jugador actual, se transmite el estado de la partida a ambos jugadores.
    - Si hay empate, se transmite el estado de la partida a ambos jugadores.
  - No es fin de partida
    - Se cambia el turno del jugador actual.
    - Se envía la información correspondiente **a cada jugador**.
    - Se espera el movimiento del jugador actual.
- Si el movimiento no es correcto (columna llena):
  - Se pide al jugador actual que realice un movimiento.

4.- Consideraciones de implementación

Para copiar una cadena de caracteres en otra se puede utilizar

```
strncpy (strDst, strSrc, STRING_LENGTH);
```

de forma que se copian los primeros `STRING_LENGTH` bytes de la cadena `strSrc` en la cadena `strDst`.

Para generar una cadena de caracteres a partir de otra, añadiendo el valor de variables se puede utilizar

```
sprintf (message, "Its your turn. You play with:%c\n", PLAYER_1_CHIP);
```

de forma que en `message` se copiaría la cadena `Its your turn. You play with:o`. Además, se pueden añadir más variables, de forma similar a como se imprimen mensajes por pantalla utilizando `printf`.

La inicialización de una cadena de caracteres se puede llevar a cabo utilizando

```
memset(str, chr, STRING_LENGTH);
```

de forma que en `str` se escriben `STRING_LENGTH` caracteres con el valor `chr`.

5. Ficheros a entregar

Los ficheros necesarios para la realización de esta práctica se encuentran en el fichero `PSD_ Prac1_Sockets.zip`. Para desarrollar la práctica se deberán modificar, **únicamente,** los ficheros `clientGame.h`, `clientGame.c`, `serverGame.h`, `serverGame.c` y `gameTypes.h`. Se deberá entregar, además, un fichero llamado `autores.txt` que contenga el nombre completo de los integrantes del grupo.

La entrega de esta práctica se llevará a cabo mediante un **único fichero comprimido** en formato `zip`. Es importante matizar que la práctica entregada debe contener los ficheros necesarios para realizar la compilación, tanto del cliente como del servidor.

En caso de que cualquiera de las partes entregadas no compile, se tendrá en cuenta la penalización correspondiente.

NO se permite incluir nuevos ficheros para el desarrollo de este apartado.

6. Plazo de entrega

La práctica debe entregarse a través del Campus Virtual **antes del día 8 de octubre de 2024, a las 18:00 horas**. La defensa de la práctica se realizará en la clase de laboratorio del día **8 de octubre de 2024**.

No se recogerá ninguna práctica que no haya sido enviada a través del Campus Virtual o esté entregada fuera del plazo `indicado`**.**

Se van a perseguir las copias y plagios de prácticas, aplicando con rigor la normativa vigente.