PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA
DEPARTAMENTO DE CIENCIA DE LA COMPUTACIÓN

IIC2333 — Sistemas Operativos y Redes — 2/2022

Proyecto 2

Jueves 20 de octubre, 2022

Fecha de Entrega: Lunes 7 de noviembre, 2022 hasta las 20:59
Fecha de ayudantía: Viernes 21 de octubre, 2022
Composición: grupos de 6 alumnos

Índice

1. Objetivos

Este proyecto requiere que como grupo implementen un protocolo de comunicación entre un servidor y múltiples clientes para coordinar un juego multijugador. El proyecto debe ser programado en el **lenguaje C**. La comunicación entre las partes debe hacerse través de la funcionalidad de *sockets* de la API **POSIX**.

Requisitos fundamentales:

1. Los clientes de este juego deberán comportarse como *dumb-clients*. Esto significa que actúan únicamente de intermediarios entre el usuario y el servidor. Este último es quien se encargará de computar **toda** la lógica.
2. Debe existir obligatoriamente una interfaz **por consola**. Todos los efectos de las funcionalidades del juego deben verse reflejados en ella.

No se asignará el puntaje correspondiente para cada funcionalidad que no cumpla con estas restricciones.

2. Descripción: *RuzleShip*

Figure 1: RuzleShip

El profesor Ruz decidió construir un nuevo laboratorio de computación en la mitad del mar. después de muchos meses notó que era vulnerable a ataques enemigos, por lo que decide crear un simulador para entrenar a los profesores y alumnos del DCC. Este simulador es **Ruzleships**. Necesita que sea multijugador, para así poder competir contra los mejores jugadores de Chile y del mundo. Trágicamente está muy ocupado supervisando la construcción del laboratorio como para crear el programa, por lo que le pide a maestros de la programación (ustedes) de interredes cibernéticas (redes) que lo hagan por él.

2.1. Concepto General

Deben programar una versión multijugador del juego Battleship. Donde se combate 1 v 1 (1 persona contra 1 persona) basado en turnos, cuyo fin es eliminar todos los barcos del contrincante antes de que este elimine los tuyos.

2.2. Conexión inicial e inicio del juego (20 puntos)

- **Conexión inicial** - El servidor debe iniciarse y esperar a que algún jugador se conecte. Una vez conectado el jugador, este debe enviarle su nombre al servidor y luego ser enviado al **Lobby**. El servidor siempre debe estar atento a alguna posible conexión de un nuevo jugador, sin dejar nunca de atender a los usuarios ya conectados.[1] Es importante que la atención de múltiples clientes sea robusta.
- **Lobby** - Dentro del **Lobby** se tendrán distintas **Salas** que permitirán a los jugadores que entren a esta esperar a un contrincante. El cliente debe ser capaz de pedirle al servidor un listado de las **Salas** disponibles junto con la cantidad de jugadores presentes en ella. Además el cliente debe poder ingresar a una **Sala** si es que esta tiene al menos un cupo disponible para el jugador.
- **Salas** - Cada **Sala** tiene un cupo máximo de 2 personas para entrar. En caso de estar llena se le debe avisar al cliente que no puede ingresar y devolverlo al **Lobby**. Dentro de la **Sala** se le debe dar las opciones de **jugar** y **salir de la sala** a ambos jugadores que estén en la sala. Una vez que ambos jugadores en la **Sala** estén listos para **jugar**, comienza el juego.

2.3. Ruzleship

En las partidas de **Ruzleship**, el servidor hace **TODO** el calculo y la lógica detrás del juego. El cliente solamente actúa como cliente *dumb*. De no respetar esto se obtiene puntaje mínimo en esta sección.

2.3.1. Fase de preparación (10 puntos)

Al iniciarse la partida, ambos jugadores deben colocar sus barcos en una grilla de 5x5, donde el eje horizontal está representado con las primeras letras del abecedario (A-E) y el eje vertical con números del 1 al 5 (Figura 2). Una vez colocados todos los barcos en la grilla se le pregunta al jugador si está seguro de las posiciones que ha elegido para cada barco. En caso de no estar seguro, se deben borrar todos los barcos y permitirle al jugador colocarlos todos de nuevo, en caso contrario se asume que el jugador está listo para comenzar la partida.

Ejemplo (referencia, no es necesario que sea igual)

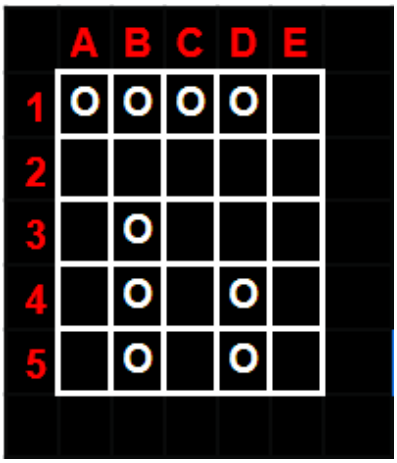

Figure 2: preparación de barcos

En la Figura 2 se ilustra una grilla de 5x5 con los 3 barcos colocados correctamente. Ninguno de ellos está superpuesto con otro y todos están dentro de la grilla de 5x5.

[1] Se recomienda el uso de threads, select(), poll(), epoll() u otra alternativa de POSIX para hacer *multiplexing*, mientras sea razonable.

HINT: A modo de recomendación, pueden colocar un barco ingresando las 2 coordenadas de inicio y fin del barco, es decir, siguiendo con el ejemplo de la Figura 2 el barco de largo 4 se puede colocar en esas posición ingresando las coordenadas A1 y luego D1 o en el sentido contrario (D1 y luego A1), luego solamente es necesario comprobar que las celdas entre A1 y D1 estén libres y que el barco esté solamente 1 columna o 1 fila (no esté diagonal).
Cada jugador tendrá a su disposición 3 barcos, cada uno de distinto tamaño. El **primero** es de tamaño **1x2**, el **segundo** es de tamaño **1x3** y el **tercero** es de tamaño **1x4**. Todo barco puede ser colocado de forma horizontal o forma vertical en el tablero, es decir, las dimensiones del **primer** barco pueden ser **1x2** o **2x1** dependiendo de como quiere colocarlo el jugador. También se debe tener en cuenta que 2 barcos no pueden colocarse en la misma posición en el tablero ni tampoco superponerse.

2.3.2. Fase de juego (10 puntos)

Una vez terminada la **Fase de preparación** comienza la **Fase de juego**. El primer jugador en entrar a la sala es quien tiene el primer turno.

Los jugadores en su turno deben elegir 1 coordenada (ejemplo, C2) para lanzar un misil al tablero del enemigo. En caso de que el jugador le haya dado a un barco se le debe notificar que golpeó un barco. En caso de que este haya golpeado todas las celdas de un barco se le debe notificar que hundió el barco. Por último si el misil no cae en ningún barco se le debe notificar al jugador que su misil falló.

Todo lo descrito anteriormente debe verse representado en una interfaz para el jugador. Se le debe presentar una grilla de 5x5 donde se muestren sus barcos y el estado en que están (golpeados o sanos), y también se debe presentar una grilla de 5x5 donde se muestren los misiles que ha lanzado al enemigo y si ha habido contacto con un barco o no. (Figura 3)

Figure 3: Ejemplo de batalla

En la Figura 3 se muestran 2 tableros: el tablero del enemigo y el tablero propio. En el tablero enemigo se marca con un `X` los misiles que hemos lanzado y que han colisionado con algún barco; y se marca con un − (guión) los misiles que fueron lanzados pero no colisionaron con algún barco. En el tablero propio se marca con `X` las celdas de los barcos propios golpeados, y con − los misiles recibidos que no han golpeados barcos; también se muestra con `O` las celdas de los barcos propios que no han sido golpeadas.

2.3.3. Fin del juego (10 puntos)

El juego termina cuando un jugador ya no tiene barcos flotando. El perdedor es quien haya perdido todos sus barcos y el ganador es quien que todavía tenga barcos. Una vez terminado el juego se le pregunta al jugador si quiere volver al **Lobby** o salir del juego. El cliente no debe desconectarse sin preguntar esto antes.

3. Implementación

3.1. Cliente

El cliente debe recibir e imprimir los menús y/o mensajes correspondientes y debe enviar los *input* al servidor. El cliente no debe hacer ningún cálculo luego de hacer la conexión con el servidor.

3.2. Servidor

El servidor debe mediar la comunicación entre los clientes. El servidor es el encargado de procesar **toda** la lógica del juego y comunicación entre clientes. Cuando el cliente deba entregar algún tipo de *input*, es el servidor el encargado de pedírselo.

3.3. Protocolo de comunicación (7 puntos)

Todos los mensajes enviados, tanto de parte del servidor, como de parte del cliente, deberán seguir el siguiente formato:

- `ID` (1 *byte*): Indica el tipo de paquete.
- `PayloadSize` (1 *byte*): Corresponde al tamaño en *bytes* del `Payload` (entre 0 y 255).
- `Payload` (`PayloadSize` *bytes*): Es el mensaje propiamente tal. En caso de que no se requiera, el `PayloadSize` será `0` y el `Payload` estará vacío.

Tienen total libertad para implementar los paquetes que estimen necesarios. No obstante, **deberán documentarlos** todos en su `README.md`, explicitando el `ID` y el formato de `Payload`, junto con una breve descripción de cada uno.

A modo de ejemplo, consideren que queremos enviar el mensaje `Hola` con el `ID` 5. Si serializamos el `Payload` según **ASCII**, su tamaño correspondería a cuatro *bytes*. El paquete completo se vería así:

```
00000101 00000100 01001000 01101111 01101100 01100001
   5        4         H        o        l        a
```

3.4. Ejecución (3 puntos)

Los clientes y el servidor deberán ejecutarse de la siguiente manera, respectivamente:

```
$ ./server -i <ip_address> -p <tcp_port>
$ ./client -i <ip_address> -p <tcp_port>
```

Donde `<ip_address>` corresponde a la **dirección IP** del servidor (en formato numérico de IPv4, por ejemplo, `172.16.254.1`) y `<tcp_port>` al puerto TCP a través del cual el servidor recibirá nuevas conexiones.

El proyecto solo será corregido si cumple con esta modalidad de ejecución.

4. Bonus (¡hasta 15 décimas!)

4.1. *SSH Tunneling* (+5 décimas)

Su programa debe poder funcionar de forma completamente distribuida. Esto es, con el servidor ejecutando en `iic2333.ing.puc.cl`, y clientes ejecutándose en lugares distintos (como sus domicilios, por ejemplo). El servidor `iic2333.ing.puc.cl` posee abierto solamente los puertos 22 y 80.

Este desafío deberá ser resuelto haciendo uso de **SSH Tunneling** para poder conectarse con un puerto local de `iic2333.ing.puc.cl`, un mecanismo cuyo funcionamiento deberán investigar por cuenta propia. Se recomienda revisar **este link** como punto de partida.

4.2. *Descarga de Imagen* (+5 décimas)

Cuando un jugador gane una partida, el servidor le envía una imagen bonita para felicitarlo. Esta imagen debe quedar guardada en el mismo directorio donde se ejecuta el binario del cliente. El bonus **DEBE** respetar el protocolo de comunicación y la imagen debe ser de más de 255 bytes, de no ser así no se otorgará puntaje.

4.3. *Desconexión y Re-conexión* (+5 décimas)

Cuando un jugador se cae, es decir se termina su conexión de manera inesperada, por ejemplo se cierra la terminal que corre el cliente, este puede volver a ingresar, puede regresar a la sala en que estaba y se le restaura su estado de juego antes de desconectarse.

El jugador que queda "vivo" debe ser informado cuando su rival se ha desconectado y puede decidir esperarlo o salir. El método para autenticar al jugador que se re-conecta queda a juicio del alumno pero necesita estar en su `README.md`

5. *README* y formalidades

5.1. Grupos

Está permitido que los grupos cambien respecto al proyecto anterior.

5.2. Entrega

Su proyecto debe encontrarse en un directorio con nombre P2 (mayúscula la P) dentro del directorio de UNO de los integrantes del grupo en el servidor del curso. Además, deberán incluir:

- Un archivo `README.md` que indique quiénes son los autores del proyecto (**con sus respectivos números de alumno**), instrucciones para ejecutar y usar el programa, descripción de los paquetes utilizados en la comunicación entre cliente y servidor, cuáles fueron las principales decisiones de diseño para construir el programa, cuáles son las principales funciones del programa, **qué supuestos adicionales ocuparon, y cualquier información que consideren necesaria para facilitar la corrección de su tarea**. Se recomienda usar formato **Markdown**.
- Uno o dos archivos `Makefile` que compilen su programa en dos ejecutables llamados `server` y `client`, correspondientes al servidor y al cliente, respectivamente.

5.3. Otras consideraciones

- Este proyecto **debe** ser programado usando el lenguaje C. No se aceptarán desarrollos en otros lenguajes de programación.
- No respetar las formalidades o un código extremadamente desordenado podría originar descuentos adicionales. Se recomienda modularizar, utilizar funciones y ocupar nombres de variables explicativos.

6. Descuentos

6.1. Descuentos (¡hasta 20 décimas!)

- 5 décimas por subir archivos binarios (programas compilados).
- 5 décimas por no incluir el/los `Makefile`s, o bien incluirlos y que no funcionen.
- 5 décimas por tener *memory leaks*. (Se recomienda fuertemente utilizar **Valgrind**).
- 5 décimas por la presencia archivos correspondientes a entregas del curso pasadas en el mismo directorio.

6.2. SÚPER DESCUENTO (¡La mitad de la nota!)

Si por cualquier motivo su proyecto no funciona ocupando sockets esto generará que cualquier puntaje que se haya obtenido se reduzca a la mitad. (¡Como hay puntaje por la conexión inicial esto significa que el 4 es imposible!)