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

IIC2333 — Sistemas Operativos y Redes — 1/2017

Tarea 4 - Battleship

Martes 23-Mayo-2017

Fecha de Entrega: Martes 6-Junio 2017, 23:59
Ayudantía: Viernes 26-Mayo 2017, 8:30
Composición: grupos de n personas, donde $n \leq 2$

El objetivo de esta tareas es construir una aplicación cliente-servidor de acuerdo a un protocolo de comunicación, de manera de poder conectar distintas implementaciones de clientes y servidores.

1. Descripción

Battleship es un juego que consiste en el enfrentamiento entre dos flotas navales, las cuales son establecidas por cada jugador al iniciarse una nueva partida.

Cada jugador dispone de un tablero de grilla de 10 por 10 cuadros, donde instalará tres naves que conformarán su flota. Este tablero no será conocido por su contrincante, por lo que las posiciones y características de sus naves, quedarán ocultas durante todo el juego.

Una vez conformada la flota, cada jugador tendrá un turno de forma alternada para enviar un torpedo a una posición del tablero enemigo, intentando de esta forma, destruir alguna nave de su contrincante.

El primer jugador que logre destruir el total de la flota enemiga, gana el juego.

2. Requerimientos generales

En esta tarea debe desarrollar un programa en C o C++ que implemente el juego *battleship*, considerando la implementación una interfaz de usuario apropiada (tablero, recepción de movimientos, despliegue de mensajes informativos), soporte de reglas del juego y comunicación a través de red para la interacción con su contrincante.

El programa deberá cumplir con lo siguientes requerimientos generales, presentados según su orden de ejecución:

2.1. Establecimiento de flota

Al iniciar el programa, se deberá solicitar al usuario que seleccione 3 naves de alguno de los tipos detallados en el cuadro 1, junto a sus respectivas posiciones dentro de un tablero de 10 x 10, y su orientación (vertical u horizontal). Considere el punto indicado por el usuario como el punto donde se ubicará la popa de la nave. Tome en consideración los largos de las naves y no permita traslapes con otras naves o ubicaciones fuera de los límites del tablero.

2.2. Establecimiento de conexión con el contrincante

Una vez que la flota ha sido creada, se deberá comenzar un procedimiento para establecer una conexión TCP con el contrincante. Para esto, su programa deberá tener la opción de comportarse tanto como cliente (inicia la conexión) o servidor (espera una conexión), según un parámetro recibido por línea de comandos. También deberá recibir como parámetro la dirección IP y puerto donde se establecerán las conexiones, como se especifica a continuación.

Cuadro 1: Formato general de mensajes

Tipo de nave	Id de nave	Largo
Carrier	1	5
Battleship	2	4
Cruiser	3	3
Submarine	4	3
Destroyer	2	2

Modo servidor

```
[user@host ~] ./battleship -l -i <ip_address> -p <tcp_port>
```

- `l`. Parámetro booleano para especificar que el proceso actuará en modo servidor (*listener*).
- `i <ip_address>`. Dirección IP local donde serán recibidas las conexiones entrantes. Si este parámetro no es proporcionado, o si se entrega la dirección 0.0.0.0, se debe asumir que las conexiones puede ser recibidas a través de cualquier interfaz (`ANY`).
- `p <tcp_port>`. Puerto TCP donde se recibirán las conexiones entrantes.

Modo cliente

```
[user@host ~] ./battleship -i <ip_address> -p <tcp_port>
```

- `i <ip_address>`. Dirección IP remota con la que se desea establecer la conexión.
- `p <port>`. Puerto TCP donde se establecerán las conexiones.

2.3. Inicio del juego

Una vez establecida la conexión con el contrincante, se deberán desplegar dos tableros de grilla de 10 x 10 cuadros, los cuales denominaremos: tablero de naves y tablero de torpedos enviados, tal como se muestra en la figura 1.

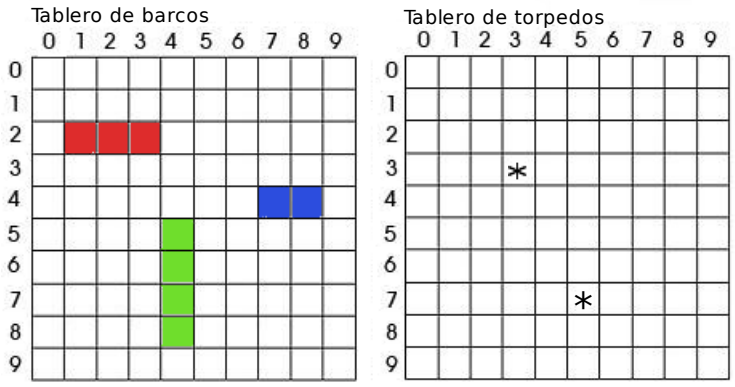

Figura 1: Tablero battleship

El tablero de naves deberá mostrar las naves que conforman la flota del usuario local y los torpedos que vayan siendo recibidos en el transcurso del juego. Por otra parte, el tablero de torpedos enviados será un registro de los ataques realizados por el usuario local, para así evitar disparar a zonas ya bombardeadas.

Junto con esto, el usuario que tenga el turno de disparar, deberá recibir un mensaje solicitando las coordenadas de su próximo torpedo. En cambio su contrincante, no tendrá la opción de ingresar coordenadas hasta que llegue su turno. La implementación de este comportamiento se encuentra definido más adelante en la sección Protocolo.

3. Reglas del juego

A continuación se listan algunas reglas generales del juego

- Cada usuario poseerá 3 naves de cualquier tipo, de entre los ya especificados
- Los turnos de juego deben ser alternados
- Una nave es considerada destruida, si todas las posiciones de su longitud son alcanzadas por un torpedo
- Gana el usuario que logre primero destruir toda la flota enemiga
- Si se detecta la desconexión del contrincante, el juego deberá terminar y el presente usuario será considerado ganador.

4. Protocolo

Cada uno de sus procesos debe interactuar con el otro usando un protocolo de comunicación. Siguiendo este protocolo debe ser posible también interactuar con un cliente o servidor implementado por otro grupo.

A continuación se define el protocolo de comunicación.

4.1. Secuencia de inicio de partida

Una vez que ambos usuarios han definido sus flotas y se ha establecido la conexión entre sus interfaces, cada proceso debe enviar a su contrincante, un mensaje de inicio denominado *Start*. Este mensaje deberá transportar como parámetro, un número entero entre 0 y 2.147.483.647 calculado de forma aleatoria, el cual permitirá definir que usuario empezará la partida.

Al recibir el mensaje, cada proceso deberá comparar el número recibido con el calculado localmente (y enviado). Según esto, el criterio de partida será que el usuario que calculó el menor número aleatorio, tendrá derecho al primer movimiento (lanzamiento de moneda, versión digital). Si por cosas del destino, estos números tienen el mismo valor, deberá obtenerse otro número aleatorio[1], y reenviar el mensaje Start hasta que se obtengan número distintos entre procesos.

La Figura 2 muestra un diagrama que representa los procesos A y B en su etapa de inicialización de partida.

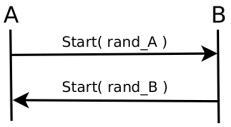

Figura 2: Secuencia de inicio de partida

4.2. Actualización de jugadas y alternancia de turnos

Cada vez que un jugador ingrese la posición de su torpedo, además de marcarlo en el tablero de torpedos, deberá enviar un mensaje denominado *NewMove* a su oponente, transportando como parámetros, las coordenadas (x, y) ingresadas por el usuario.

Dado que los turnos entre jugadas se van dando de forma alternada entre jugadores, el intercambio de estos mensajes también se dará de forma alternada tal como lo muestra la figura 3.

Luego de enviado este mensaje, el proceso no deberá permitir el ingreso de una nueva coordenada hasta la recepción del siguiente mensaje *NewMove*.

[1] Y jugar un Kino, ¿por qué no?

Adicionalmente, cada vez que una nave sea destruida, el usuario que sufrió la pérdida deberá enviar el mensaje *Destroyed* a su oponente, transportando con él, el parámetro *remaining*, el cual deberá ser un número entero que informe acerca de la cantidad de naves disponibles luego de la destrucción. Esta situación es mostrada por la figura 3, y destacada con una flecha roja (A destruyó una nave de B).

El primer jugador en quedar sin naves disponibles (todas destruidas), perderá el juego. En ese instante, el perdedor enviará el último mensaje *Destroyed* con el valor 0 en el parámetro *remaining*, y terminará la ejecución del proceso imprimiendo un mensaje indicando que el juego fue perdido. A su vez, el proceso que recibe el mensaje *Destroyed* con *remaining* igual a 0, deberá imprimir un mensaje anunciando que el juego fue ganado, y el proceso deberá terminar.

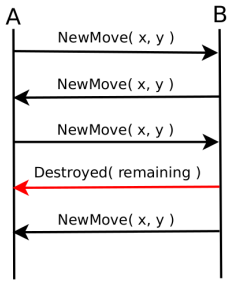

Figura 3: Actualización de nueva jugada

5. Mensajería

Todos los mensajes implementados son formados por campos de 8 bits (*buffer* de `chars`), teniendo como base común, los primeros dos campos que corresponderán al largo y tipo de mensaje respectivamente, como se muestra en el cuadro 2.

Cuadro 2: Formato general de mensajes

Lenght	Message Type
8 bits	8bits

El largo del mensaje transportado por el campo *length*, debe ser incluido dentro de la cuenta.
Los valores numéricos de los identificadores de mensajes que deben ser soportados, se muestran en el cuadro 3.

Cuadro 3: Identificadores de mensajes

Message Type	Id
Start	11
New Move	12
Destroyed	13

Por otra parte, los cuadros 4, 5 y 6 presentan los formatos de parámetros específicos para cada mensaje en particular.

Cuadro 4: Formato de mensaje Start

Lenght	Message Type	rand number
8 bits	8bits	8 bits

Cuadro 5: Formato de mensaje New Move

Lenght	Message Type	x	y
8 bits	8bits	8 bits	8 bits

Cuadro 6: Formato de mensaje Destroyed

Lenght	Message Type	remaining
8 bits	8bits	8 bits

6. Consideraciones de diseño

A continuación se detallan algunas consideraciones de diseño que pueden ser relevantes en la implementación:

- Su programa en modo servidor, deberá permitir la conexión de solo un usuario remoto. Luego de establecida, no se debe permitir la conexión desde otros clientes. Si otro cliente intenta conectarse, su servidor debe seguir funcionando normalmente. Puede ignorar el mensaje del nuevo cliente, o bien responder que no acepta la conexión.
- Si se detecta la desconexión del oponente, el juego deberá ser considerado como ganado, imprimiéndose el correspondiente mensaje de triunfo y terminando el proceso.
- Si bien para la recepción de comandos de usuario debe utilizarse la consola, los tableros pueden ser desplegados directamente sobre ella, o bien, utilizando la interfaz gráfica de linux. Algunas ideas para esto: tablero ASCII en consola, tablero gráfico basado en GNU plot, Qt, etc.
- La función de recepción de mensajes debe comenzar por leer el primer *byte* para determinar el largo del mensaje, y luego, entrar en un *loop* de lectura para rescatar los siguientes `n` bytes, donde `n`, corresponde al valor obtenido en la primera lectura. Esto debido a que no está garantizada la recepción de mensajes independientes aunque hayan sido realizados en envíos separados. Es decir, existe la posibilidad de que lleguen dos mensajes pegados, o bien, uno seccionado.

7. Evaluación

- 30 % Cumplimiento de requerimientos básicos del juego
- 10 % Calidad de interfaz gráfica (independiente del método elegido)
- 30 % Correcta implementación del protocolo
- 20 % Procedimiento de establecimiento de conexiones y reusabilidad de direcciones
- 10 % Correcta implementación de función de lectura de *socket* según solicitud de enunciado
- **10 % BONUS:** Implementación de mecanismo de continuación de partidas al detectarse una desconexión. Ambos procesos deben continuar en el estado donde quedó la partida hasta antes de la desconexión, incluso si la causa fue la caída de uno de los procesos. Si ambos procesos caen, la partida comienza desde cero. Para esto puede agregar las modificaciones que considere necesarias a la línea de comandos o a la interfaz.

8. Entrega

La tarea será entregada mediante `git` en un repositorio **privado** que ustedes deberán crear. Se revisará el contenido de la rama `master` al día Martes 6 de Junio de 2017, 23:59.

- Puede ser realizada en forma individual, o en grupos de 2 personas.
- Debe incluir un README con todas las consideraciones de diseño que sean relevantes al momento de revisar su tarea. Debe incluir al menos, instrucciones de compilación y ejecución.
- **NO** agregar código binario (ejecutables) al repositorio. Puede agregar imágenes si las necesita.
- Su tarea deberá compilar utilizando el comando `make` en la raíz de su repositorio, y generar los ejecutables necesarios.
- Su repositorio **DEBE ser privado**, de lo contrario calificará como copia (alguien les podría haber copiado). Esta restricción es fundamental. Si no la cumple **no** se revisará su tarea.
- La entrega será automatizada, basta que **registren** su grupo y repositorio mediante un formulario que habilitaremos para ello (no por email).
- Como el repositorio debe ser privado, tendrá que permitir acceso especial al curso, autorizando al servidor de tareas acceder al contenido. Para esto basta registrar la llave pública del curso en las **Deployment Keys** de su repositorio.
- Lo mejor es que use Bitbucket.