Trabajo Práctico 2: `pthreads`
"Batalla Naval SOs"

Sistemas Operativos - Primer Cuatrimestre de 2016

Entrega: Miércoles 08 de junio a las 23:59

1. Introducción

Bienvenidos a una nueva transmisión de esta entretenida y sana materia que hemos dado en llamar *Sistemas Operativos*. El motivo que nos reune en esta ocasión es el de cooperar con nuestra compañía hermana, la PyME del entretenimiento *HaSObro*[1].

El último mes un consultor fue reclutado para analizar la difícil situación financiera de la empresa tras la caida del mercado interno. Tras recibir una escandalosa suma en concepto de honorarios, dicho analista le hizo saber a *HaSObro* que, "Dicho mal y pronto, su negocio está en las últimas. El juego de mesa *sha fue* y lo que se viene ahora es la *ueb tu point ou*[2]".

Desesperado, el CEO de *HaSObro* decidió lanzar al mercado una versión *web-social-online-multijugador-twitter-facebook* de su clásico juego de *Batalla Naval*. Rápidamente el CEO hizo uso de sus contactos para conseguir inversores suficientemente ingenuos para financiar el proyecto, y contrató a un grupo de programadores *monoproceso-teístas*. Sin embargo, durante el transcurso del desarrollo se hizo cada vez más notorio que era necesario adoptar las bondades del multiprocesamiento para que el juego pudiera hacerse realidad.

Ante tal atentado contra su fe, los *mono-programadores* abandonaron el proyecto, y a *HaSObro* no le quedó otra que recurrir a nosotros para que mejoremos su servidor de modo de que permita múltiples jugadores a la vez. Así fue como nosotros los comprometimos a ustedes a entregar un prototipo del servidor que permita **múltiples clientes jugando simultáneamente** sobre un mismo tablero antes del miércoles 08 de junio a las 23:59.

2. Las reglas del juego

Según la especificación (ampliamente informal) que nos entregaron, el juego consiste en dos equipos que pelean entre sí para destruirse sus barcos. Y cada equipo puede tener una cantidad ilimitada de jugadores (> 0).

El objetivo del juego es que cada equipo coloque sus barcos en un tablero compartido. Y una vez que todos los jugadores de ambos equipos dieron su conformidad con los barcos colocados, empieza la batalla.

[1] Ninguno de los docentes tiene acciones en esta empresa, ni en ninguna empresa de Panama
[2] Web 2.0

Los barcos se agregan por partes. Las partes del barco pueden agregarse en cualquier casillero **disponible** del tablero para conformar un barco entero (sucesiones de partes alineadas vertical u horizontalmente). Al formar un barco entero, las partes deberán ser colocadas en el tablero de forma adyacente (no es válido que queden "huecos" en medio de un barco a medida que se lo va construyendo).

Cuando un jugador termina de formar una barco, debe gritar **¡Barco Terminado!**. A partir de este momento, se considera que el barco está terminado y las partes del barco agregadas al tablero para que pueden ser usadas por los demás jugadores de su equipo.

¡Ojo! Las partes que fueron colocadas pero aún no son parte de un barco terminado (es decir, si el jugador puso una parte pero aún no gritó ¡Barco Terminado!) ocupan su lugar en el tablero momentáneamente para todo el equipo pero **no están confirmadas**. Sólo después de que el barco esté terminado los demás jugadores podrán servirse de dichas partes para utilizarlas también como partes de otro barco en construcción.

3. La implementación

3.1. Arquitectura del sistema

Por tratarse de una aplicación web, el sistema necesita atender peticiones HTTP que le hacen los *browsers* de los usuarios. Los responsables de atender a estos navegadores son los llamados servidores de *frontend* del sistema. Esta comunicación utiliza un dialecto particular que afortunadamente ya fue programado por los desarrolladores *monoproceso-teístas*. Un único servidor de *frontend* puede atender a varios *browsers* que deseen conectarse para jugar. Para implementar esta funcionalidad, el servidor de *frontend* inicia una conexión TCP por cada uno de los *browsers* que se conectan.

A su vez, los servidores de *frontend* se comunican con un único servidor de *backend* a través de conexiones TCP/IP. Este servidor de *backend* es el que nos atañe: la implementación actual sólo recibe una única conexión TCP, y por lo tanto sólo permite un jugador por vez. Los demás jugadores no logran conectarse al servicio, y de aquí se origina el problema de *HaSObro*.

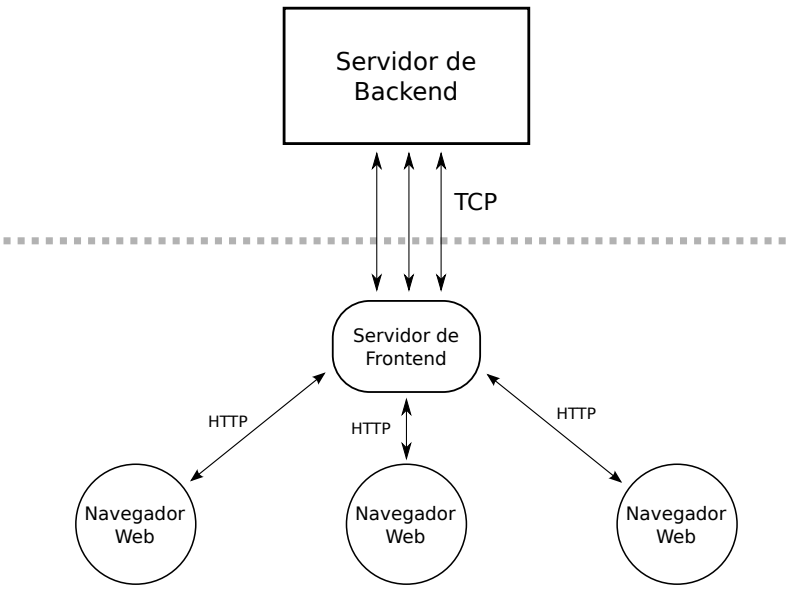

Figura 1: Arquitectura del sistema de Batalla Naval SOs

3.2. Comunicación cliente/servidor

Al iniciar el servidor de *backend*, se especifica el tamaño del tablero donde se jugará y el servidor espera a que se conecte algún cliente. La conexión a éste se realiza con un socket TCP utilizando el puerto 5481.

La conexión HTTP entre los clientes y los servidores de *frontend* ya fue resuelta anteriormente, por lo que sólo debemos preocuparnos por la conexión TCP entre los servidores de *frontend* y el de *backend*.

3.3. Protocolo de la comunicación entre *frontend* y *backend*

3.3.1. Handshake

Cuando un nuevo usuario desea comenzar a jugar, se produce la siguiente comunicación entre los servidores:

1. El *frontend* establece la conexión TCP con el *backend* en el puerto 5481 de este último.
2. El *frontend* registra al jugador enviando `EQUIPO` $nombre_{equipo}$.
3. El *backend* le responde `TABLERO N M` indicando que el tablero tiene N casilleros de ancho y M de alto.
4. A continuación, el cliente está listo para empezar a jugar y se pasa a la fase de *Gameplay*.
5. Sólo se debe aceptar dos equipos, pero una cantidad arbitraria de jugadores pueden registrarse en cada equipo enviando el nombre de su equipo.

3.3.2. Gameplay - Construcción de Barcos

Durante esta fase del juego, el usuario envía partes de un barco con sus respectivas posiciones al servidor e intenta construir barcos con ellas.

Dependiendo del estado del tablero, podría o no tener éxito. Es responsabilidad del *backend* comunicarle el resultado de sus acciones.

1. El *frontend* envía `PARTE_BARCO X Y` indicando que desea colocar una parte de un barco en la posición (X,Y) del tablero. Los números X e Y cumplen que $0 \leq X < N$ y $0 \leq Y < M$. La posición $(0, 0)$ corresponde a la esquina superior izquierda.
2. Si la posición (X,Y) se encontraba disponible, el *backend* responde `OK`. En caso contrario, responde `ERROR`.
3. Los dos pasos anteriores se repiten hasta colocar todas las partes de un barco.
4. El *frontend* envía `BARCO_TERMINADO` para indicar que las partes que acaba de enviar conforman un barco.
5. El *backend* responde `OK` y a partir de esta respuesta, las partes agregadas pueden también ser utilizadas por cualquier otro jugador para constituir sus propios barcos.

En caso de recibir una respuesta de error, el *frontend* debe comenzar de nuevo a construir el barco. Todas las partes colocadas desde el final del barco anterior (o desde que comenzó a jugar) son eliminadas del tablero.

Para empezar la batalla, todos los jugadores de todos los equipos deben enviar al servidor el comando `LISTO`.

1. Cuando un jugador envíe el comando `LISTO` por primera vez, recibirá la respuesta `OK`. Si vuelve a enviarlo, recibirá `ERROR`.
2. Una vez que un jugador ha enviado `LISTO`, deberá recibir `ERROR` si intenta agregar nuevos barcos.
3. Una vez que todos hayan enviado `LISTO`, no se podrá agregar nuevos barcos y comenzará la batalla.

3.3.3. Gameplay - Batalla

Durante esta fase del juego, el usuario envía bombas con sus respectivas posiciones al servidor e intenta destruir los barcos del otro equipo.

Dependiendo del estado del tablero, podría o no tener éxito. Es responsabilidad del *backend* comunicarle el resultado de sus acciones.

1. El *frontend* envía `BOMBA X Y` indicando que desea lanzar una bomba en la posición (X,Y) del tablero. Los números X e Y cumplen que $0 \leq X < N$ y $0 \leq Y < M$. La posición $(0, 0)$ corresponde a la esquina superior izquierda.
2. Si en la posición (X,Y) había una parte de un barco rival, el *backend* responde `GOLPE` si fue la primera vez que se lanzó una bomba en esa posición.
3. Si en la posición (X,Y) había una parte de un barco rival ya golpeado por una bomba, el *backend* responde `ESTABA_GOLPEADO`.
4. Si en la posición (X,Y) no había nada, responde `OK`.

3.3.4. Estado de juego

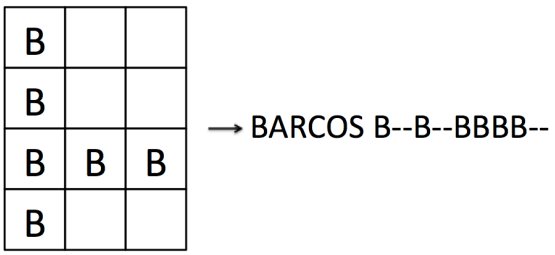

Figura 2: Formato de la respuesta a `UPDATE`

Al momento de comenzar a jugar, o después de recibir una confirmación del tipo OK o ERROR, el *frontend* puede en todo momento enviar `UPDATE` al *backend* para recibir una actualización del estado actual del juego.

El formato de la respuesta que le devuelve consiste en una palabra que indica el estado del juego: `BARCOS` o `BATALLA` seguida de la lectura por filas de izquierda a derecha y de arriba hacia abajo del contenido del tablero.

En el caso de estar en el momento de construcción de barcos (`BARCOS`), las **B** representan las partes de barco del equipo y los guiones **-**los casilleros vacíos.

En el caso de estar en el momento de batalla (`BATALLA`), los ***** representan las partes de barcos dañadas del equipo rival y los guiones **-** los casilleros vacíos.

Al hacer `UPDATE`, el cliente debe recibir el estado del tablero conteniendo **únicamente** los barcos terminados. Los barcos en curso (es decir, las de aquellos jugadores que aún no gritaron "*¡Barco Terminado!*") no deben estar contenidas en la respuesta `BARCOS`.

3.3.5. Finalización

Cuando una conexión del servidor de *frontend* se pierde o se cierra, o si se produce un error en la conexión durante el transcurso del juego, el sistema debe garantizar que los barcos se conservan en el tablero, mientras que los barcos que aún no fueron terminados serán eliminados de este.

3.3.6. Ejemplo

```
> EQUIPO Azul
< TABLERO 4 5
> PARTE_BARCO 2 2
< OK
> PARTE_BARCO 2 1
< OK
> PARTE_BARCO 4 0
< ERROR
> PARTE_BARCO 0 1
< OK
> PARTE_BARCO 1 1
< OK
> PARTE_BARCO 2 1
< OK
> BARCO_TERMINADO
< OK
> PARTE_BARCO 1 0
< OK
> PARTE_BARCO 1 2
< OK
> BARCO_TERMINADO
< OK
> PARTE_BARCO 4 0
< OK
> PARTE_BARCO 4 2
< ERROR
> BOMBA 2 2
< ERROR
> LISTO
< OK
> LISTO
< ERROR
> BOMBA 2 2
< ERROR
> BOMBA 2 2
< OK
> PARTE_BARCO 4 2
< ERROR
> BOMBA 4 2
< GOLPE
> BOMBA 4 2
< ESTABA_GOLPEADO
```

Explicación del protocolo:

1. EQUIPO registra al jugador en servidor de backend para el equipo *Azul*.
2. El servidor retorna TABLERO indicando su ancho y alto.
3. El cliente envía PARTE_BARCO con la posición del tablero donde desea colocar cada parte.
4. El servidor le responde OK porque no surgen problemas al colocarlas.
5. El cliente envía BARCO_TERMINADO para confirmar el barco.
6. El servidor le responde OK indicando que el barco quedó plasmado, y ahora puede continuar enviando partes para otro barco.
7. Continúa enviando partes y terminando barcos sin problema.
8. Luego del último barco, el cliente envia partes a las posiciones $(4, 0)$ y $(4, 2)$
9. Como no existe una parte colocada en la posición $(4, 1)$, las partes enviadas no son contíguas y el servidor le responde ERROR.
10. Al enviar el mensaje BOMBA por primera vez, todavía estábamos en la parte de armar barcos, por lo que el servidor contesta ERROR.
11. Al enviar LISTO, recibe OK la primera vez. Y luego, ERROR si vuelva a intentarlo.
12. Cuando envía BOMBA después del LISTO, falla una vez porque el resto de los clientes todavía no había enviado LISTO.
13. Cuando envía BOMBA más tarde, ya es aceptada y recibe una respuesta dependiendo de si golpeó un barco o no.

3.4. Utilización del código provisto

El código que dejaron tras de sí los programadores anteriores está disponible para descargar en la página de la materia.

Para utilizarlo, se deben realizar los siguientes pasos:

1. Compilar el servidor de *backend*: `make`

2. Iniciar el servidor de *backend*: `./backend-mono/backend 4 5`
3. Iniciar el servidor de *frontend*: `python ./frontend/frontend.py` `(no hacerlo desde dentro de la carpeta frontend)`
4. Acceder con un browser a `http://localhost:5482` (¡Ojo! En caso de que se esté usando un *proxy*, habrá que poner una excepción para `localhost`)

Para iniciar múltiples clientes, alcanza con abrir nuevas ventanas o pestañas del navegador y acceder nuevamente a la dirección indicada.

4. Entregable

La entrega del trabajo es realizará de manera electrónica enviando un mail a la dirección **sisopdc@gmail.com** un mail cuyo *subject* debe decir:

[TP2]: Entrega TP Pthreads

y cuyo cuerpo debe contener los datos de cada integrante:

$Apellido_1$, $Nombre_1$, LU_1, $Correo\ Electrónico_1$
$Apellido_2$, $Nombre_2$, LU_2, $Correo\ Electrónico_2$
$Apellido_3$, $Nombre_3$, LU_3, $Correo\ Electrónico_3$

El trabajo consta de dos apartados:

1. En primer lugar, deberán implementar un *Read-Write Lock* **libre de inanición** utilizando **únicamente** Variables de Condición POSIX (es decir, **no es válido usar semáforos**) y respetando la interfaz provista en los archivos `backend-multi/RWLock.h` y `backend-multi/RWLock.cpp`.
2. Deberán, a su vez, implementar un test para su implementación de *Read-Write Locks* (`RWLockTest`) que involucre la creación de varios *threads* lectores y escritores donde cada uno de ellos trate de hacer un *lock* sobre un mismo recurso y se vea que no haya *deadlocks* ni inanición (archivo `backend-multi/RWLockTest.cpp`).
3. En segundo lugar, deberán implementar el servidor de *backend multithreaded* inspirándose en el código provisto y lo desarrollado en el punto anterior.

En la entrega se deberán adjuntar únicamente:

- El documento del informe (en PDF).
- El código fuente **completo y con Makefiles** del servidor de *backend*.

El informe deberá ser de caracter **breve** e incluir el pseudocódigo de los algoritmos que se ejecutan en el servidor de *backend* frente a cada petición de un cliente, poniendo énfasis en las primitivas de sincronización al estilo del primer parcial de la materia. Si fuera necesario, puede ser buena idea incluir una explicación del funcionamiento del servidor en lenguaje natural. Cualquier decisión de diseño que hayan tomado deberá ser incluida aquí.

La implementación que realicen del servidor de *backend* debe estar libre de condiciones de carrera y presentar la funcionalidad descripta arriba a cada uno de los clientes. A su vez, debe:

- Permitir que múltiples clientes se conecten al *backend* de forma **simultánea**.
- Permitir que todos los jugadores coloquen partes de barcos o bombas en casilleros distintos de forma **simultánea**.
- Permitir que varios clientes consulten el estado del tablero de forma **simultánea**.