Trabajo Práctico 2: `pthreads`
"Locker Pocker"

Sistemas Operativos - Segundo Cuatrimestre de 2016

Entrega: Lunes 17 de octubre 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 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 *Poker*. Rápidamente el CEO hizo uso de sus contactos para conseguir inversores suficientemente ingenuos para financiar el proyecto, y contrató a un grupo de programadores *monoprocesoteístas*. 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.

2. Las reglas del juego

Según la especificación (ampliamente informal) que nos entregaron, el objetivo del juego es completar pokers sobre un tablero compartido mediante la sucesiva colocación de cartas sobre el mismo.

En cada jugada, al jugador le toca un sólo palo de cartas (diamantes, corazones, tréboles, picas) y puede agregar en el tablero tantas cartas del mismo palo desee en esa jugada.

[1] Ninguno de los docentes tiene acciones en esta empresa
[2] Web 2.0

Las cartas pueden agregarse en cualquier casillero **disponible** del tablero para conformar una jugada (sucesión de cartas contiguas alineadas vertical u horizontalmente).

Al formar una jugada, las cartas deberán ser colocadas en el tablero de forma adyacente (no es válido que queden "huecos" en medio de una jugada a medida que se la va construyendo).

Cuando un jugador termina de formar una jugada, debe gritar **¡Confirmo!**. A partir de este momento, se considera que la jugada está confirmada y las cartas agregadas al tablero pueden ser utilizadas por los demás jugadores para conformar sus propias pokers.

¡Ojo! Las cartas que fueron colocadas pero aún no son parte de una jugada terminada (es decir, si el jugador puso una carta pero aún no gritó ¡Confirmo!) ocupan su lugar en el tablero pero **no pueden ser utilizadas por los otros jugadores para formar sus jugadas**. Sólo después de que la jugada esté completa los demás jugadores podrán servirse de dichas cartas.

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*.

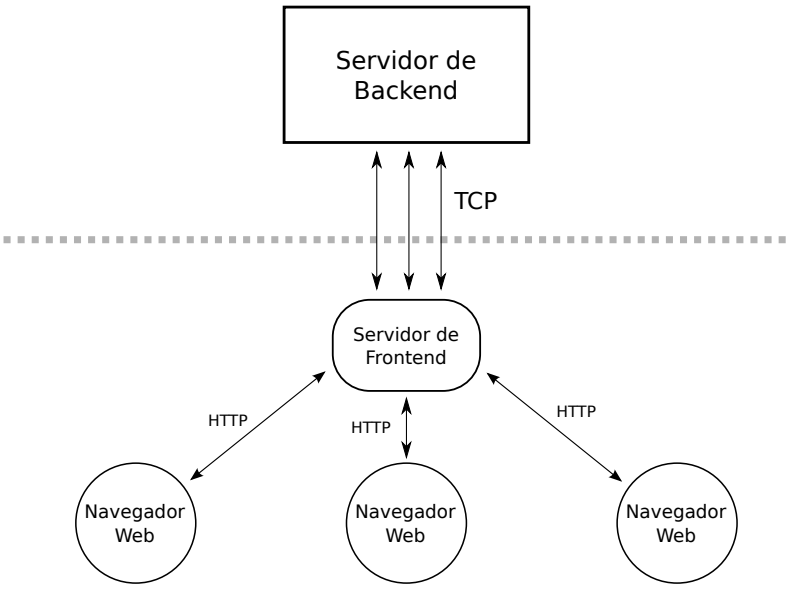

Figura 1: Arquitectura del sistema de SOScrabble

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 `SOY` *nombre*.
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*.

3.3.2. Gameplay

Durante esta fase del juego, el usuario envía cartas con sus respectivas posiciones y tipo al servidor e intenta construir pokers 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 `CARTA X Y P` indicando que desea colocar la carta de palo P 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. P es una letra mayúscula del alfabeto ASCII que representa el palo (D, C, P, T).
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 cartas de una jugada.
4. El *frontend* envía `CONFIRMO` para indicar que las cartas que acaba de enviar conforman su jugada.
5. El *backend* responde `OK` y a partir de esta respuesta, las cartas agregadas pueden ser utilizadas por cualquier otro jugador para constituir su jugada.
6. Dado el creciente espíritu olímpico, a nadie le interesa llevar la cuenta realmente de cuantas jugadas fueron póker y cuales no. Los jugadores, sólo juegan hasta el aburrimiento.

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

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 la palabra `STATUS` seguida de la lectura por filas de izquierda a derecha y de arriba hacia abajo del contenido del tablero, reemplazando los casilleros vacíos por guiones para completar un *string* de $N * M$ caracteres. Al hacer `UPDATE`, el cliente debe recibir el estado del tablero conteniendo **únicamente** las jugadas completas. Las cartas en curso (es decir, las de aquellos jugadores que aún no gritaron *¡Confirmo!*") no deben estar contenidas en la respuesta `STATUS`.

3.3.3. 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 las jugadas completadas se conservan en el tablero, mientras que las cartas que aún no forman jugadas son eliminadas del mismo.

3.3.4. Ejemplos

Veamos un ejemplo sencillo:

```
> SOY Fulano
< TABLERO 4 5
> CARTA 2 2 T
< OK
> CARTA 2 1 T
< OK
> CARTA 4 0 T
< ERROR
> CARTA 0 1 C
< OK
> CARTA 1 1 C
< OK
> CARTA 2 1 C
< OK
> CONFIRMO
< OK
> CARTA 1 0 T
< OK
> CARTA 1 2 T
< OK
> CONFIRMO
< OK
> CARTA 4 0 D
< OK
> CARTA 4 2 D
< ERROR
```

Explicación del protocolo:

1. SOY se registra al servidor de backend con el nombre *Fulano*.
2. El servidor retorna TABLERO indicando su ancho y alto.
3. El cliente envía CARTA con el la posición del tablero donde desea colocar cada carta y su palo.
4. El servidor le responde OK porque no surgen problemas al colocarlas.
5. El cliente envía CONFIRMO para confirmar las cartas de la jugada.
6. El servidor le responde OK indicando que la jugada quedó plasmada, y ahora puede continuar enviando cartas para otras jugadas.
7. Continúa enviando cartas y jugadas sin problema.
8. Luego de la última jugada, el cliente envia cartas a las posiciones $(4, 0)$ y $(4, 2)$
9. Como no existe una carta colocada en la posición $(4, 1)$, las cartas enviadas no son contíguas y el servidor le responde ERROR.

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`
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 la estructura *Read-Write Lock* representando la interfaz provista por la cátedra. El *Read-Write Lock* deberá ser **libre de inanición** y se podrá utilizar **únicamente** variables de condición de la liberería POSIX commo estructuras de sincronización para su implemenación (es decir, **no es válido usar ni semáforos ni mutex**). Además, deberán **implementar un test** de la estructura que involucre la creación de varios *threads* y muestre el funcionamiento esperado.
2. 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**. Para la primera sección, se deberá incluir el pseudocódigo de los algoritmos, las referencias que justiquen la implementación desarrollada, y la explicación de los tests desarrollados.

Para el segundo apartado, se deberá poner énfasis en los cambios realizados para la correcta sincronización del servidor. 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 cartas en casilleros distintos de forma **simultánea**.
- Permitir que varios clientes consulten el estado del tablero de forma **simultánea**.