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

IIC2333 — Sistemas Operativos y Redes — 1/2018

Tarea 4

Lunes 28 de Mayo de 2018

Fecha de Entrega: Viernes 8 de Junio de 2018 a las 23:59
Composición: grupos de n personas, donde $n \leq 2$

Objetivo

Esta tarea requiere que usted implemente un protocolo de comunicación entre un servidor y uno o más clientes, para coordinar un juego en línea. La tarea debe ser programada usando la API POSIX de *sockets* en el lenguaje C.

Descripción: *5-Card Draw Poker*

Uno de los juegos de apuestas más conocido es el Póker, que consiste en comparar cartas entre jugadores para ver quién tiene más puntaje. El ganador se lleva la suma de todas las *bet* (apuestas) realizadas en las rondas anteriores. Existen distintas variaciones del Póker, y nos centraremos en una de las más sencillas.

5-Card Draw Poker es una variante del Póker que consiste en que cada jugador recibe 5 cartas del mazo, de las cuales puede elegir cuál cambiar, para luego hacer una apuesta y finalmente ver quién tiene el puntaje mayor. Para efectos de la tarea, se simplifica el juego en lo siguiente:

1. Todos los jugadores comienzan con un *pot* (monto inicial) de $1000.
2. Al comienzo de cada partida, antes de que cada jugador reciba las 5 cartas, se debe hacer una apuesta mínima para poder jugar, de un monto de $10. Si un jugador no puede pagar la apuesta mínima, se considera como perdedor de la partida.
3. Una vez que los jugadores hayan pagado la apuesta mínima, se comienza la ronda, donde se reparten 5 cartas a cada jugador.
4. Cada jugador elige sus cartas a cambiar.
5. Luego de recibir las nuevas cartas se comienza a apostar. El primer jugador tiene la opción de hacer una *bet* de $0, $100, $200 $500.
6. El jugador 2 tiene la opción de hacer *FOLD* (retirarse de la ronda), igualar la apuesta del jugador 1 o hacer una *bet* mayor a la del jugador 1 (siempre considerando las 4 opciones de *bet* disponibles).
7. Si las *bet* coinciden, se procede a revisar qué jugador tiene el mayor puntaje y terminar la ronda.
8. Si la *bet* del jugador 2 es mayor a la del jugador 1, el jugador 1 tiene la opción de hacer *FOLD* (retirarse de la ronda) o igualar la *bet* del jugador 2.
9. Finalmente, se comparan las cartas y se revisa qué jugador es el ganador de la ronda. Al ganador se le envía una imagen de ganador, y al perder se le envía una imagen de perdedor. Para efectos de esta tarea, el cliente no necesita desplegar la imagen recibida, sin embargo sí es necesario que ésta sea recibida íntegramente.

Parte I - Cliente de juego

Esta parte consiste en construir un cliente de juego para *5-Card Draw Poker*. El cliente debe contener algún tipo de interfaz (en consola) que permita visualizar las cartas del jugador, su *pot* y la suma total de las *bet* hechas en la ronda. Además, una vez terminada la ronda, se deberán mostrar las cartas del contrincante, mostrar el ganador y el monto ganado. Al final de la partida, deberá recibir una imagen enviada por el servidor. El cliente se comporta como un *dumb-client*, es decir, depende totalmente de la conexión con el servidor para el procesamiento de la logística del juego.

Se debe implementar el protocolo estándar de esta tarea de tal modo que su cliente funcione comunicándose tanto con un servidor elaborado por usted como con uno elaborado por sus compañeros. Es decir, tiene que seguir exactamente el protocolo especificado en el enunciado. Si el servidor se desconecta repentinamente, el cliente debe ser capaz de controlar dicha conexión e informar al jugador sobre la desconexión.

Parte II - Servidor de juego

Para esta parte de la tarea, deberá implementar un servidor de juego que ofrezca la mediación de comunicación entre los clientes. El servidor es el encargado de procesar toda la logística del juego (controlar si los jugadores tienen suficiente *pot* para apostar, repartir cartas, ver quién ganó, etc.) y enviársela correctamente a los clientes.

Debe implementar el protocolo estándar de esta tarea de tal modo que sea posible que, tanto clientes elaborados por usted como por compañeros, puedan jugar sin inconvenientes. Si algún cliente interrumpe su conexión repentinamente, el servidor debe ser capaz de controlar dicha desconexión e informar al otro cliente conectado que ganó.

Protocolo

El protocolo que utilizará este juego considera red mensajes binarios que siguen un patrón estándar, específicamente:

Un mensaje en este protocolo contiene:

- *MessageType ID* (8 bits): Corresponde al tipo de paquete que se está enviando.
- *Payload Size* (8 bits): Corresponde al tamaño en *bytes* de la información (*Payload*) que se va a enviar.
- *Payload* (variable): Corresponde la información propiamente tal. Si en algunos paquetes no sale definido qué tipo de *Payload* enviar, o si el paquete en sí no necesita enviar *Payload*, el *Payload* debe tener valor *0*.

Un mensaje tendrá de distintos significados de acuerdo a su ID. El contenido y uso del paquete se determina de acuerdo a la siguiente lista.

1. *Start Connection*: Cliente envía este paquete al Servidor.
2. *Connection Established*: Servidor responde con este paquete luego de recibir el *Start Connection* del cliente.
3. *Ask Nickname*: Servidor envía a cliente este paquete para preguntarle el nickname (nombre) del cliente que se acaba de conectar.
4. *Return Nickname*: Cliente responde a servidor este paquete con el nickname del cliente.
5. *Opponent Found*: Servidor envía a cliente este paquete indicando que se encontró otro cliente para comenzar el juego. El *payload* de este paquete es el nickname del contrincante.
6. *Initial Pot*: Servidor envía a cliente el *pot* inicial con el que comienza la partida.
7. *Game Start*: Servidor envía a cliente indicando que comenzó correctamente el juego.

8. *Start Round*: Servidor envía a cliente indicando que comenzó una nueva ronda del juego. El payload de este paquete es el *pot* actual del cliente.

9. *Initial Bet*: Servidor envía a cliente el monto de la apuesta inicial realizada por el cliente, que será siempre de $10.

10. *5-Cards*: Servidor envía a cliente las 5 cartas del cliente, según la tabla de ID's especificada en el enunciado. En el *payload* se deben ocupar 8 bits para representar el número de una carta y 8 bits para representar la pinta, obteniendo un total de 80 bits (10 *bytes*) (Revisar formato de envío de cartas).

11. *Who's First*: Servidor envía a cliente en el Payload un 1 o un 2. Si es 1, el cliente es el que apuesta primero. Si es 2, el cliente espera la apuesta del otro cliente.

12. *Get Cards to Change*: Servidor envía a cliente este paquete para preguntarle qué cartas son las que quiere cambiar.

13. *Return Cards to Change*: Cliente envía a Servidor este paquete con las cartas a cambiar. El formato es mismo que el paquete *5-Cards*.

14. *Get Bet*: Servidor envía a cliente las apuestas disponibles. El *payload* son los ID's de las *bet* disponibles.

15. *Return Bet*: Cliente envía a Servidor el ID de la apuesta elegida.

16. *Error Bet*: Servidor envía este paquete al cliente si la apuesta es incorrecta (por ejemplo, si la apuesta es mayor al monto del cliente).

17. *OK Bet*: Servidor envía a cliente este paquete confirmando la apuesta realizada.

18. *End Round*: Servidor envía a cliente informando que la ronda se terminó.

19. *Show Opponent Cards*: Servidor envía a cliente este paquete con las cartas del oponente.

20. *Winner/Loser*: Servidor envía a cliente un 1 si resultó ganador o un 2 si resultó perdedor.

21. *Update Pot*: Servidor envía a cliente el monto del *pot* resultante después de haber ganado, perdido o apostado.

22. *Game End*: Servidor envía a cliente este paquete informando que terminó el juego.

23. *Image*: Servidor envía a cliente paquete con la imagen correspondiente al ganador o perdedor. Esta imagen debe guardarse en el mismo directorio en donde se está ejecutando el código.

24. *Error Not Implemented*: Tanto el servidor como el cliente pueden responder este paquete si reciben algún ID que no tienen implementado.

Tablas de identificadores

Cuadro 1: Identificadores de mensajes

MessageType	**ID**
Start Connection	1
Connection Established	2
Ask Nickname	3
Return Nickname	4
Opponent Found	5
Initial Pot	6
Game Start	7
Start Round	8
Initial Bet	9
5-Cards	10
Who's First	11
Get Cards to Change	12
Return Cards to Change	13
Get Bet	14
Return Bet	15
Error Bet	16
OK Bet	17
End Round	18
Show Opponent Cards	19
Winner/Loser	20
Update Pot	21
Game End	22
Image	23
Error Not Implemented	24

Cuadro 2: Representación de Cartas

Carta	**Repr.**
A	1
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10
J	11
Q	12
K	13

Cuadro 3: Representación de pintas (Se recomienda usar los íconos de Unicode para mostrarlos en consola)

Pinta	Repr.	Unicode
♥	1	U+2665
♦	2	U+2666
♣	3	U+2663
♠	4	U+2660

Cuadro 4: Representación de apuestas

Apuesta	Repr.
FOLD	1
$0	2
$100	3
$200	4
$500	5

Nota: Tanto su cliente como su servidor no se pueden caer por recibir paquetes mal formados o de funciones que no implementen. En caso que no implementen alguna función, al menos debe ser capaz de manejar la recepción del paquete y tomar alguna acción. Las caídas de su programa debido a mal manejo originarán descuentos.

Formato de envío

Veamos un ejemplo: después de que el servidor reparta las cartas a un cliente, va a querer enviárselas a través del paquete *5-Cards*, con ID *10*. Supongamos que las cartas son:

- 10 de corazón
- 2 de diamante
- 5 de trébol
- 7 de picas
- Q de corazón

Usando la tabla de identificación de cartas y de pinta, el *payload* tendrá el siguiente formato (el color rojo representa los IDs de las pintas):

$$10\,1\,2\,2\,5\,3\,7\,4\,12\,1$$

Luego, cada número presentado anteriormente tiene que ser pasado a su representación binaria:

$$\underbrace{10}_{00001010}\ \underbrace{1}_{00000001}\ \underbrace{2}_{00000010}\ \underbrace{2}_{00000010}\ \underbrace{5}_{00000101}\ \underbrace{3}_{00000011}\ \underbrace{7}_{00000111}\ \underbrace{4}_{00000100}\ \underbrace{12}_{00001100}\ \underbrace{1}_{00000001}$$

Finalmente, el servidor arma el paquete a enviar al cliente, asignándole el ID *10* (ya que va a enviar el paquete *5-Cards*) y con *Payload Size* de *10 bytes* (ya que son 10 números, cada uno de 1 bit):

$$\underbrace{00001010}_{\textit{ID: 10}}\ \underbrace{00001010}_{\textit{Payload Size: 10}}\ \underbrace{0000101000000001000000100000001000000101000000110000011100000100000110000000001}_{\textit{Payload: 10 1 2 2 5 3 7 4 12 1}}$$

El cliente, al recibir este paquete, primero interpretará el primer *byte* para saber qué paquete recibió. Luego, interpretará el segundo *byte*, que es el *Payload Size* para saber el tamaño del *Payload*. Y, por último, interpetará el *Payload*. Sería incorrecto que el cliente interprete el *Payload* como un solo número (si seguimos el ejemplo, el cliente interpretaría *472422562564187517122 57*), sino que tiene que ser capaz de dividir los bits de tal forma de interpretar correctamente el mensaje. Este procedimiento es equivalente a los otros paquetes del protocolo.

Formato de Ejecución

Tanto el servidor como el cliente deben ejecutarse de la siguiente manera:

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

donde:

- `-i <ip-address>` es la dirección IP que va a ocupar el servidor para iniciar, o la dirección IP en la cual el cliente se va a conectar.
- `-p <tcp-port>` es el puerto TCP donde se establecerán las conexiones.

Para que el cliente se conecte correctamente a la IP y puerto, es necesario que primero se inicie el servidor. Por ejemplo:

```
$ ./server -i 127.0.0.1 -p 8888
```

El servidor iniciará la conexión con esos parámetros. Luego, el cliente tendrá que usar el mismo IP y puerto para establecer la conexión:

```
$ ./client -i 127.0.0.1 -p 8888
```

README y Formalidades

Deberá incluir un archivo `README` que indique quiénes son los autores de la tarea (**con sus respectivos números de alumno**), cuáles fueron las principales decisiones de diseño para construir el programa, qué supuestos adicionales ocuparon, y cualquier información que considere necesarias para facilitar la corrección de su tarea. Se sugiere utilizar formato **markdown**.

A cada alumno se le asignó un nombre de usuario y una contraseña para el servidor del curso (`iic2333.ing.puc.cl`). Para entregar su tarea usted deberá crear una carpeta llamada `T4` y subir su tarea a esa carpeta. En su carpeta `T4` **solo debe incluir código fuente** necesario para compilar su tarea, además del README y un `Makefile`. **NO debe incluir archivos binarios (será penalizado)**. Se revisará el contenido de dicha carpeta el día Viernes 8 de Junio de 2018 a las 23:59.

- Puede ser realizada en forma individual, o en grupos de 2 personas. En cualquier caso, recuerde indicar en el `README` los autores de la tarea con sus respectivos números de alumno.
- La parte I de esta tarea debe estar en el directorio `/T4/client`, y la parte II de esta tarea en el directorio `/T4/server`. Cada una de las partes debe compilar utilizando el comando `make` en los directorios señalados anteriormente.

Evaluación de Funcionalidades

Cada funcionalidad se evaluará en una escala de logro de 4 valores, ponderado según la siguiente información:

- 10 % Inicio y término Conexión.
- 5 % Nickname de jugadores.
- 20 % Logística de apuestas.
- 20 % Puntajes correctos de cartas y combinaciones.
- 20 % Actualización de *bet* (puntaje).
- 10 % Envío de imagen.
- 10 % Readme y Formalidades. Esto incluye cumplir las normas de la sección formalidades.

- 5 %. Manejo de memoria. *valgrind* reporta en su código 0 *leaks* y 0 errores de memoria, considerando que los programas funcionen correctamente.

El no respeto de las formalidades o un código extremadamente desordenado podría originar descuentos adicionales. Se recomienda modularizar, utilizar funciones y ocupar nombres de variables explicativos. En el caso de no entregar en la carpeta especificada la tarea **no** se corregirá.

Preguntas

Cualquier duda preguntar a través del foro.