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

IIC2333 — Sistemas Operativos y Redes — 2/2017

Tarea 4

Jueves 18-Octubre-2017

Fecha de Entrega: Jueves 2-Noviembre-2017 a las 23:59
Composición: grupos de n personas, donde $n \leq 2$

Esta tarea requiere que usted implemente un protocolo de comunicación entre dos programas, un servidor y uno o más clientes, para coordinar un juego en línea. La tarea debe ser programada en C.

Parte I. Cliente de juego - `ajedrez`

Esta parte consiste en realizar un cliente de juego para ajedrez. Deberá tener algún tipo de interfaz para el jugador actual y permitir que vea el tablero, haga su jugada y vea las jugadas del contrincante. El cliente deberá comunicarse exclusivamente con un servidor que podrá ser configurado al momento de ejecutar, y recibirá datos desde este servidor. Debe considerar que el servidor ofrece, además del servicio de mediación de comunicación, un servicio de $matchmaking$ donde puede consultar por jugadores esperando contrincante y solicitarles un juego. 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.

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 y el $matchmaking$. El servidor debe soportar la conexión de un número arbitrario de clientes, incluso cuando existan partidas en proceso, y poder interactuar con todos ellos. 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.

Protocolo

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

- ID de transacción (8 bits): Corresponde al tipo de mensaje que se está enviando.
- Número de *bytes* de *payload* (8 bits): Corresponde al tamaño en *bytes* de la información que se va a enviar.
- *payload* (Variable): Corresponde la información propiamente tal.

Un paquete puede usarse de distintas maneras de acuerdo a su ID. El contenido y uso del paquete se determina de acuerdo a la siguiente lista.

1. *Heartbeat* - Servidor envía un paquete con este ID y como *payload* un *string* aleatorio de 1 *byte*. El cliente debe responder con un paquete con el mismo ID, y en su *payload* el primer *byte* debe contener el *string* aleatorio enviado por el servidor y en los siguientes 4 *byte* debe contener el *timestamp* de la respuesta, formateado como Unix epoch. Se espera que se envíe un paquete de heartbeat cada 10 segundos, salvo que se permita configurar. Por defecto, considere un timeout de 30 segundos.
2. *Matchmaking*. Cliente envía como *payload* un *string* que contenga un *nickname* al servidor para registrarse como disponible. El *string* debe estar codificado en UTF-8. Servidor responde asignando un ID único de 2 *byte* como *payload*.

3. *Matchmaking List*. Cliente envía este paquete sin *payload*. El servidor responde codificando en el *payload* los clientes que actualmente esperan una partida, usando un formato de primeros 4 *byte* el número de clientes en espera; luego, por cada cliente, 2 *byte* con su ID, 1 *byte* para el número de *byte* del *nickname* y los siguientes *byte* el *nickname* del usuario.

4. *Match Request*. Cliente envía al servidor como *payload* el ID del cliente con el que desea jugar. El servidor responde con un `0` si el cliente no acepta el juego y un `1` si acepta.

5. *Match Request*. Servidor envía a un cliente como *payload* el ID del cliente que desea jugar y, concatenado, el *nickname* del jugador. El cliente debe responder como *payload* `0` si no acepta jugar y `1` si acepta.

6. Chat. Cliente en su *payload* debe especificar en los primeros 2 *byte* el ID de destino de su mensaje, donde el ID `0` corresponde a todos los clientes en espera. Los siguientes *byte* serán su mensaje codificado en UTF-8

7. Game-start. El servidor envía a los clientes que jugarán, un *payload* que contiene el ID de cada jugador seguido de 1 *byte* en `0` para el jugador blanco y `1` para el jugador negro. Clientes deben responder con su ID y color.

8. *Move*. El cliente envía paquete con 1 *byte* para columna de origen, 1 *byte* fila de origen, 1 *byte* columna de destino, 1 *byte* fila de destino y 1 *byte* identificador de pieza. Servidor recibe paquete y reenvía a cliente, quién debe responder con `0` (OK) o `1` (error). Esta respuesta debe ser reenviada al originador de la movida.

9. *Disconnect*. Cliente envía paquete sin *payload* para notificar su salida. Servidor debe responder con `0` como *payload*.

10. *Contrincant-Disconnect*. Servidor envía paquete a cliente con *payload* ID del contrincante que se ha desconectado, cliente debe responder con `0`.

11. *Error*. Servidor responde esta intención cuando una transacción produce un error al cliente.

12. *GameSync*. Cliente envía tablero completo codificado, siguiente un *byte* en `0` o `1` para el color de pieza, 1 *byte* para columna, 1 *byte* para fila y 1 *byte* para tipo de pieza.

13. *BoardSetAuthortity*. Servidor envía tablero completo codificado de forma igual a un paquete *GameSync*. Clientes deben aceptarlo.

14. *ServerInfo*. Cliente envía a servidor un paquete sin *payload*, servidor responde con número de clientes conectados (1 byte), número de clientes en espera (1 byte), número de juegos en curso (1 byte), versión de servidor de la forma X.Y.Z donde X,Y,Z son 1 byte cada uno e ID de implementación (4 byte).

15. *PacketSupport*. El programa que recibe este paquete, debe responder con una lista con los IDs de los paquetes que soporta correctamente. Cada ID debe ocupar 1 byte.

16. *GameRequest*. El servidor envía un paquete con este ID y sin payload a un cliente. Cliente debe ejecutar un GameSync a continuación.

Cuadro 1: Identificadores de mensajes

MessageType	**Id**
Heartbeat	1
Matchmaking	2
MatchmakingList	3
Matchrequest (Inivitación a jugar)	4
Matchrequest (Invitar a jugar)	5
Chat	6
GameStart	7
Move	8
Disconnect	9
Contrincant-Disconnect	10
Error	11
GameSync	12
BoardSetAuthority	13
ServerInfo	14
PacketSupport	15
GameRequest	16

Cuadro 2: Identificadores de piezas

ChessPieces	**Id**
Rey	1
Reina	2
Torre	3
Alfil	4
Caballo	5
Peón	6

Cuadro 3: Códigos ASCII para las piezas

ChessPiece	**ASCII Pieza Negra**	**ASCII pieza Blanca**
Rey	U+265A	U+2654
Reina	U+265B	U+2655
Torre	U+265C	U+2656
Alfil	U+265D	U+2657
Caballo	U+265E	U+2658
Peón	U+265F	U+2659

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.

Bonus: Lobby y Chat

Podrán optar por un bonus de un 15 % adicional aquellos alumnos que implementen este bonus.

Que el servidor haga $matchmaking$ instantáneamente después de recibir una conexión de un cliente es poco natural e incómodo para el usuario. Por lo tanto, para ser más realistas con los juegos existentes hoy en día, se pide que,

en vez de que servidor realice un $matchmaking$ inmediatamente después de recibir alguna conexión, exista un Lobby donde los clientes puedan ingresar al momento de realizar la conexión. Este Lobby debe contar con una sala de Chat, donde los clientes conectados pueden interactuar y conversar. Una vez que alguno de los clientes esté listo para jugar, debe poder ser capaz de enviarle esa señal al servidor. El servidor recibirá dicha solicitud, y esperará a que otro cliente quiera jugar, y luego hacer el $matchmaking$.

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` en el directorio `Entregas` de su carpeta personal 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 Jueves 2-Noviembre-2017 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 `Entregas/T4/client`, y la parte II de esta tarea en el directorio `Entregas/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 nota entre 1.0 y 7.0, ponderado según la siguiente información:

- 5 % Heartbeat.
- 10 % Inicio Conexión.
- 10 % Término Conexión.
- 5 % Nickname de jugadores.
- 20 % ptos. Juego y Movimientos.
- 10 % Sincronización de tablero.
- 10 % Manejo de errores.
- 5 % Server Info.
- 10 % Package Support.
- 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.
- BONUS: 15 % Chat y Lobby.

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.