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

IIC2333 — Sistemas Operativos y Redes — 2/2020

Proyecto 2

Lunes 26 de Octubre

Fecha de Entrega: Miércoles 4 de Noviembre
Composición: grupos de n personas, donde $3 \leq n \leq 4$
Fecha de ayudantía: Miércoles 28 de Octubre

Índice

1. Objetivos

Este proyecto requiere que como grupo implementen un protocolo de comunicación entre un servidor y múltiples clientes para coordinar un juego en línea. El proyecto debe ser programado en el **lenguaje C**. La comunicación entre las partes debe hacerse través de la funcionalidad de *sockets* de la API **POSIX**.

Requisitos fundamentales:

1. Los clientes de este juego deberán comportarse como *dumb-clients*. Esto significa que actúan únicamente de intermediarios entre el usuario y el servidor. Este último es quien se encargará de computar **toda** la lógica.
2. Al finalizar el juego, debe aparecer un mensaje indicando la razón por la cual finalizó y se debe mostrar claramente al o los ganadores.
3. Debe existir obligatoriamente una interfaz por consola. En otras palabras, todos los efectos de las funcionalidades del juego deben verse reflejados en ella.

Nota: No se asignará el puntaje correspondiente para cada funcionalidad que no cumpla con estas tres restricciones.

2. Descripción: *Among Ruz*

El profesor Cristian Ruz, aburrido por los extensos meses de aislamiento por culpa de la pandemia, se le ha ocurrido una genial idea para entretenerse durante el encierro: un juego online original e innovador cuyas mecánicas y jugabilidad no se habían explorado nunca antes en todo el universo: ***Among Ruz***! Lamentablemente, el profesor se encuentra ocupado con todas sus obligaciones laborales, por lo que ha recurrido a ustedes, sus hábiles alumnos, expertos en redes, para que programen el juego.

Among Ruz se ambienta en la nave espacial ***R.U.Z. Enterprise*** en donde la tripulación realiza tranquilamente sus labores para mantener un viaje intergaláctico estable y sin percances. Una forma de vida alienígena hostíl, sin embargo, se ha infiltrado en la nave y ha adoptado la apariencia de uno de los tripulantes, y amenaza con eliminar a toda la tripulación. Es el deber de los tripulantes descubrir al impostor y expulsarlo de la nave antes de que este los elimine.

El juego en sí consiste de un chat de hasta 8 jugadores, es decir 8 clientes, conectados a un servidor central. Los jugadores tienen un rol que define sus objetivos para ganar, además de los comandos que pueden utilizar.

2.1. Conexión inicial e inicio del juego (10 puntos)

- **Conexión Inicial (3 puntos)** - El servidor en ejecución deberá esperar a que los clientes o jugadores se conecten para empezar el juego. El servidor debe estar preparado para mantener hasta 8 clientes conectados simultáneamente. Cuando un cliente logre conectarse al servidor, se le debe dar la bienvenida al juego e indicar el número de jugadores conectados, así como su color para ser identificado por el resto de los jugadores.
- **Colores (2 puntos)** - Para identificar a los jugadores, al conectarse al servidor se le otorga a cada jugador un color. Los colores disponibles son: Rojo, Naranja, Amarillo, Verde, Celeste, Azul, Violeta y Rosado. El color es solo el nombre. No es necesario que el texto se muestre en el color correspondiente en la consola.
- **Inicio del Juego (3 puntos)** - El comando `\start` puede ser usado por cualquiera de los jugadores para iniciar el juego. Esto solo ocurrirá si la cantidad de jugadores conectados es válida, de lo contrario se le indicará al jugador que la cantidad de jugadores conectados no es válida.
- **Cantidad Válida de Jugadores (2 puntos)** - Una cantidad de jugadores conectados válida para iniciar el juego son de 3 a 8 jugadores, ambos incluidos.

2.2. Roles y objetivos (5 puntos)

En ***Among Ruz*** existen 2 roles, cada uno con sus objetivos para ganar. Un jugador solo puede tener un rol durante toda la partida.

- **Ruzmate (1 punto)** - Es un tripulante de la nave. Tiene como misión discutir con el resto de la tripualción para identificar al impostor y poder expulsarlo de la nave antes de que el impostor acabe con todos. Los **ruzmates** ganan cuando el impostor es expulsado de la nave.
- **Impostor (2 puntos)** - Es el alienígena infiltrado en la nave. Su misión es eliminar al resto de la tripulación antes de que lo expulsen de la nave. El **impostor** gana cuando quedan vivos solo él y otro tripulante.

En una partida solo habrá **un único impostor**. El resto de los jugadores serán **ruzmates**. El **impostor** es elegido **de forma aleatoria** por el servidor al iniciar la partida. Todos los jugadores deben ser notificados sobre cual es su rol al comienzo de la partida, de forma que **un jugador solo conozca su rol y no el del resto de los jugadores (2 puntos)**.

2.3. Chat (15 puntos)

Among Ruz funciona en base a un chat. Cualquier usuario que se conecte al servidor debe ser capaz de enviar un mensaje y este se le debe mostrar a todos los demás jugadores junto con el color de quien envió el mensaje **(5 puntos)**.

Además el servidor debe enviar un mensaje a todos los jugadores cuando ocurra lo siguiente:

- **Cuando un jugador se conecte (1 punto)**: se debe enviar un mensaje indicando que un jugador se ha conectado al servidor y el color del jugador.
- **Cuando un jugador es expulsado o eliminado (2 puntos)**: se debe enviar un mensaje indicando que un jugador fue expulsado o eliminado y el color del jugador. Además, si el jugador fue **expulsado**, se debe revelar su rol al resto de los jugadores. Si el jugador fue **eliminado no se revela su rol**.
- **Cuando un jugador utilice el comando `\start` (2 puntos)**: se debe enviar un mensaje a todos los jugadores indicando que la partida ha comenzando, junto con enviarle a cada jugador su rol, **solo si** la cantidad de jugadores conectados es válida.
- **Cuando un rol cumpla sus objetivos para ganar (3 puntos)**: se debe enviar un mensaje a todos los jugadores indicando que la partida ha terminado, junto con enviar el rol ganador de la partida. Usar el comando `\start` una vez terminada la partida **iniciará una nueva**, repartiendo nuevamente los roles entre todos los jugadores conectados, si la cantidad de jugadores conectados es válida.

Si un jugador fue expulsado de la nave, entonces no puede seguir participando del chat. El servidor debe seguir enviándole los mensajes de los demás jugadores, pero **debe ignorar cualquier mensaje enviado por usuarios expulsados**. La única excepción es el comando `\players`, el cual si puede ser utilizado por jugadores expulsados **(2 puntos)**.

2.4. Comandos (23 puntos)

Los comandos representan acciones que los jugadores pueden realizar durante el juego. Se activan enviando el mensaje correspondiente por el chat del juego. Un mensaje para activar un comando siempre será precedido por un *backslash* (\) de la siguiente forma:

```
\nombre_comando
```

Al enviar por el chat un mensaje que active un comando, **este no debe ser enviado al resto de los jugadores**, es decir, no aparece en el chat del resto de los jugadores como un mensaje de texto enviado por el jugador que activa el comando. Los comandos disponibles se detallan a continuación:

- `\start` **(1 punto)** - Inicia el juego si la cantidad de jugadores conectados es válida. Puede ser usado por cualquier jugador conectado. Al iniciar la partida se le debe notificar a todos los jugadores que el juego ha comenzado y ha cada jugador, su rol. Si la cantidad de jugadores conectados no es válida o si el juego ya comenzó, usar este comando solo imprime un mensaje al jugador que lo activa indicando que el comando no es válido y la razón.
- `\exit` **(3 puntos)** - El jugador que activa este comando sale del juego y deja de participar. Se le debe notificar a los jugadores que el jugador que activó el comando (representado por su color) ha salido del juego. Para efectos del juego, un jugador que sale usando este comando **se considera como expulsado**. Se debe verificar si el **impostor** o los **ruzmates** logran su objetivo si un jugador sale del juego mediante este comando.
- `\players` **(2 puntos)** - El jugador que activa este comando es informado sobre el estado de todos los jugadores: el color de los jugadores, si están vivos, eliminados o expulsados y su voto actual. **No se informa sobre el rol de los jugadores**.
- `\vote [color]` **(4 puntos)** - Al usar este comando, especificando un color, el jugador que usa el comando **crea o actualiza su voto**. A través de los votos los jugadores pueden ser expulsados de la nave: si al crear o actualizar un voto un color es **mayoría**, el jugador con este color es **expulsado** de la nave. Si un jugador usa este comando por segunda o más veces, **su voto se actualiza**, pudiendo así cambiar el color elegido en su voto. Si un jugador vota por un color que ya fue eliminado o expulsado, **no se crea ni actualiza su voto**, y debe ser notificado de que su voto no es válido y la razón. Todos los jugadores vivos pueden votar en cualquier momento, y solo el **último voto de cada jugador es válido para expulsar a alguien**. Cuando un jugador es expulsado de le debe indicar a todos los jugadores que el **color** y el **rol** del jugador expulsado.
- `\kill [color]` **(5 puntos)** - Al usar este comando, especificando un color, el impostor intenta eliminar al jugador con el color elegido. **Solo el impostor puede usar este comando**. Si un jugador **ruzmate** intenta usar este comando o si el color elegido no está en juego o ya fue eliminado, se le debe notificar al jugador que usa el comando que el comando no es válido y la razón. El resultado de usar este comando está definido por las siguientes probabilidades:
  - Eliminar (30 %) - El jugador con el color elegido es **eliminado** del juego. Se debe indicar en el chat a todos los jugadores que el jugador con el color elegido fue eliminado **sin revelar al impostor**.
  - Aviso privado (30 %) - El jugador con el color elegido **no es eliminado** y se entera de quien lo intentó eliminar. **Solo este jugador se entera** de que lo intentaron eliminar y quién lo intentó.
  - Aviso público (40 %) - El jugador con el color elegido **no es eliminado** y se avisa en el chat a todos los jugadores de que intentaron eliminar al jugador con el color elegido, **sin revelar al impostor**.

- `\spy [color]` **(4 puntos)** - Al usar este comando, especificando un color, el jugador que usa el comando espía al jugador con el color especificado y se entera del rol de este jugador. **Solo el jugador que usa el comando se entera del rol del jugador con el color elegido**. Además **solo los Ruzmates puede usar este comando** Si el color no está en juego, o bien, un impostor intenta utilizarlo, el comando no funciona y se le debe notificar al jugador que usa el comando que el comando no es válido y la razón. Este comando puede ser usado **una sola vez en todo el juego y entre todos los jugadores**, es decir, entre todos los Ruzmates solo uno podrá espiar a otro jugador una única vez. Si un jugador intenta usar este comando cuando ya fue usado (por el o por otro jugador), se le debe indicar que el comando ya fue usado.
- `\whisper [color] [mensaje]` **(4 puntos)** - Al usar este comando, especificando un color y un mensaje, el jugador que usa el comando le envía el mensaje especificado de forma privada al jugador con el color especificado. El jugador que usa este comando y el destinatario **son los únicos que pueden ver este mensaje en el chat**, y debe indicarse en este que el mensaje es privado. Si el color especificado no está en juego, el mensaje no se envía y debe notificarse al jugador que usa el comando que el comando no es válido y la razón.

2.5. Interfaz por consola

Se espera que para este proyecto desarrollen una interfaz por consola para interactuar con el juego. Se debe mostrar de forma clara el chat, resultados de comandos, mensajes de bienvenidas y otros elementos que estimen necesarios. A continuación se muestra un ejemplo del chat durante una partida:

```
[VERDE]: red is sus
[ROJO]: mentira!! yo espié a azul y el es el impostor D:
[SERVIDOR]: AZUL HA SIDO ELIMINADO
[CELESTE]: bruh...
[VIOLETA]: ok votemos rojo
[SERVIDOR]: ROJO HA SIDO EXPULSADO DE LA NAVE, ROJO ERA RUZMATE
[CELESTE]: :^)
[VIOLETA]: https://www.youtube.com/watch?v=GgYduvOD9Z0&ab_channel=BoladeNieve
```

Nota: Esto es solo un ejemplo. Ustedes son libres de decidir cómo van a mostrar el chat y la información necesaria, siempre y cuando sea de forma **clara**.

3. Implementación

3.1. Cliente

El cliente debe tener algún tipo de interfaz en consola que permita visualizar el chat, resultados de comandos, mensajes del servidor y enviar mensajes y comandos. Siéntase libres de diseñar esto como quieran, siempre que se entienda el flujo del juego y se cumplan con los requisitos mínimos de este.

Otro punto importante, que fue mencionado anteriormente, es que el cliente debe comportarse como un *dumb-client*, es decir, que solamente actúa de intermediario entre el usuario y el servidor y no es un participante activo de la lógica del programa.

3.2. Servidor

El servidor que ustedes implementarán debe mediar la comunicación entre los clientes. El servidor es el encargado de procesar **toda** la lógica del juego (recibir y procesar *inputs* de usuario, mantener el estado de cada jugador, gestionar el uso de comandos, revisar si se cumple una mayoría de votos para expulsar a alguien, evaluar quien es/son el ganador/los ganadores, etc.).

3.3. Protocolo de comunicación (7 puntos)

Todos los mensajes enviados, tanto de parte del servidor, como de parte del cliente, deberán seguir el siguiente formato:

- `ID` (1 *byte*): Indica el tipo de paquete. **(1 puntos)**
- `PayloadSize` (1 *byte*): Corresponde al tamaño en *bytes* del `Payload` (entre 0 y 255). **(2 puntos)**
- `Payload` (`PayloadSize` *bytes*): Es el mensaje propiamente tal. En caso de que no se requiera, el `PayloadSize` será `0` y el `Payload` estará vacío. **(2 puntos)**

Tienen total libertad para implementar los paquetes que estimen necesarios. No obstante, deberán documentarlos todos en su `README.md`, explicitando el `ID` y el formato de `Payload`, junto con una breve descripción de cada uno. **(2 puntos)**

A modo de ejemplo, consideren que quiero enviar el mensaje `Hola` con el `ID` 5. Si serializo el `Payload` según **ASCII**, su tamaño correspondería a cuatro *bytes*. El paquete completo se vería así:

```
00000101 00000100 01001000 01101111 01101100 01100001
```

Si necesitan inspiración, los invitamos a revisar los enunciados de semestres anteriores, en los que encontrarán algunos ejemplos de paquetes que podrían llegar a ser útiles (luego de algunas modificaciones) en este proyecto.

3.4. Ejecución

Los clientes y el servidor deberán ejecutarse de la siguiente manera, respectivamente:

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

Donde `<ip_address>` corresponde a la **dirección IP** del servidor (en formato *human-readable*, por ejemplo, `172.16.254.1`) y `<tcp_port>` al puerto TCP a través del cual el servidor recibirá nuevas conexiones.

El proyecto solo será corregido si cumple con esta modalidad de ejecución.

4. Bonus (¡hasta 11 décimas!)

Para que está sección sea tomada en consideración, es importante que indiquen en su `README.md` cuáles son los bonus que implementaron y una breve descripción de su funcionamiento. **Es requisito que la nota mínima de su proyecto sea 4.0 sin bonus para que estos sean considerados**.

4.1. Chat Fantasma (spooky)(+2 décimas)

Cuando se juega con Chat fantasma los jugadores expulsados podrán hablar entre ellos. Cualquier mensaje enviado por un jugador expulsado debe ser visible por todos los demás jugadores expulsados, pero NO deben ser visibles por lo jugadores que aún no han sido expulsados. Se debe indicar en la interfaz del jugador que se está comunicando por el chat fantasma.

4.2. Más Impostores (+4 décimas)

Cuando se juega con más impostores las reglas del juego cambian, en específico ya no habrá un solo impostor, si no que habrá un número variable de impostores.

El comando `\start` debe funcionar de la siguiente manera: `\start [impostores]`, donde impostores puede tomar el valor 1 o 2. La partida deberá seleccionar tantos jugadores para que sean impostores como indica la variable `impostores`. En el caso de 1 impostor, la cantidad válida de jugadores se mantiene igual, pero si se intenta iniciar una partida con 2 impostores, la cantidad válida de jugadores es de 5 a 8 (ambos incluidos)

Cuando un **impostor** active el comando `\players` se le deberá indicar además que jugadores son impostores, de ese modo podrá reconocer a sus compañeros.

Si se juega con 2 impostores, los Ruzmates ganan si logran expulsar a ambos impostores de la nave. Por otro lado, los impostores ganan si quedan vivos la misma cantidad de impostores que Ruzmates, o bien, quedan vivos más impostores que Ruzmates.

4.3. *SSH Tunneling* (+5 décimas)

Su programa debe poder funcionar de forma completamente distribuida. Esto es, con el servidor ejecutando en `iic2333.ing.puc.cl`, y clientes ejecutandose en lugares distintos (como sus domicilios, por ejemplo). El servidor `iic2333.ing.puc.cl` posee abierto solamente los puertos 22 y 80.

Este desafío deberá ser resuelto haciendo uso de **SSH Tunneling** para poder conectarse con un puerto local de `iic2333.ing.puc.cl`, un mecanismo cuyo funcionamiento deberán investigar por cuenta propia. Se recomienda revisar **este link** como punto de partida.

Notar que para la conexión necesitarán un puerto en el servidor. El puerto de su grupo deberá ser el puerto:

```
9000 + (Num en la lista del curso de cualquier integrante)
```

Por ejemplo, un grupo con el alumno 76 en la lista del curso ocupará el puerto 9076. Se puede ocupar el número de cualquier integrante del grupo en la lista. La lista la pueden encontrar en el siguiente **link**.

5. *README* y formalidades

5.1. Grupos

Está permitido que los grupos cambien respecto al proyecto anterior.

5.2. Entrega

Su proyecto debe encontrarse en la *master branch* de su repositorio en Github al momento de la entrega. Si realizan el bonus de ***SSH Tunneling***, también deberán subir su proyecto al servidor del curso (`iic2333.ing.puc.cl`).

Además, deberán incluir:

- Un archivo `README.md` que indique quiénes son los autores del proyecto (**con sus respectivos números de alumno**), instrucciones para ejecutar el programa, descripción de los paquetes utilizados en la comunicación entre cliente y servidor, cuáles fueron las principales decisiones de diseño para construir el programa, cuáles son las principales funciones del programa, qué supuestos adicionales ocuparon, y cualquier información que consideren necesaria para facilitar la corrección de su tarea. Se recomienda usar formato **Markdown**.
- Un o dos archivos `Makefiles` que compilen su programa en dos ejecutables llamados `server` y `client`, correspondientes al servidor y al cliente, respectivamente.

5.3. Otras consideraciones

- Este proyecto **debe** ser programado usando el lenguaje C. No se aceptarán desarrollos en otros lenguajes de programación.
- No respetar las formalidades o un código extremadamente desordenado podría originar descuentos adicionales. Se recomienda modularizar, utilizar funciones y ocupar nombres de variables explicativos.
- Si bien no será evaluado, se recomienda el uso de **Valgrind**.

6. Descuentos (¡hasta 20 décimas!)

- 5 décimas por subir archivos binarios (programas compilados).
- 10 décimas por no incluir el/los `Makefiles`, o bien incluirlos y que no funcionen.
- 5 décimas por la presencia archivos correspondientes a entregas del curso pasadas en la misma *branch*.

7. Nota final y atraso

La nota final del proyecto entregado a tiempo se calcula de la siguiente manera:

$$N = 1 + \frac{\sum_i p_i}{10} + b - d$$

Se puede hacer entrega del proyecto con un máximo de cuatro días de atraso. La fórmula a seguir es la siguiente:

$$N_{atraso} = \text{mín}(N; 7 + b - d - 0{,}75 \cdot a)$$

Siendo d el descuento total, p_i el puntaje obtenido en el ítem i, b el bonus total, a la cantidad de días de atraso y $\text{mín}(x; y)$ la función que retorna el valor mas pequeño entre x e y.