



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

IIC2333 — Sistemas Operativos y Redes — 2/2021
Proyecto 2

Jueves 12 de mayo, 2022

Fecha de Entrega: Viernes 27 de mayo, 2022 hasta las 20:59

Fecha de ayudantía: Viernes 13 de mayo, 2022

Composición: grupos de 5 alumnos

Índice

1. Objetivos	2
2. Descripción: <i>Ruzzle</i>	2
2.1. Concepto General	2
2.2. Conexión inicial e inicio del juego (20 puntos)	2
2.3. <i>Ruzzle</i>	3
2.3.1. Flujo del juego (20 puntos)	3
2.3.2. Criterio para ganar	4
2.3.3. Fin del juego (10 puntos)	4
3. Implementación	5
3.1. Cliente	5
3.2. Servidor	5
3.3. Protocolo de comunicación (7 puntos)	5
3.4. Ejecución (3 puntos)	5
4. Bonus (¡hasta 15 décimas!)	6
4.1. <i>SSH Tunneling</i> (+5 décimas)	6
4.2. <i>Leaderboard</i> (+5 décimas)	6
4.3. <i>Descarga de Imagen</i> (+5 décimas)	6
5. <i>README</i> y formalidades	6
5.1. Grupos	6
5.2. Entrega	6
5.3. Otras consideraciones	7
6. Descuentos	7
6.1. Descuentos (¡hasta 20 décimas!)	7
6.2. SÚPER DESCUENTO (¡La mitad de la nota!)	7

1. Objetivos

Este proyecto requiere que como grupo implementen un protocolo de comunicación entre un servidor y múltiples clientes para coordinar un juego multijugador. 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. 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 restricciones.

2. Descripción: *Ruzdle*

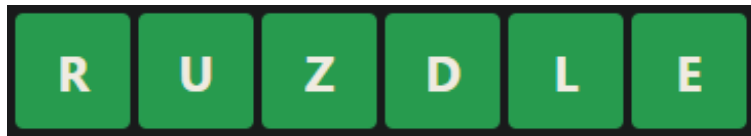


Figure 1: Ruzdle

El profesor Ruz después de muchos meses de jugar Wordle en la vasta solitaria de *internet* se empieza a aburrir, por lo que decide crear un Wordle con multijugador para así poder darle un *enhancing* a su ocio diario y quizás poder, quien sabe, entrenar a la futura generación de campeones chilenos de Wordle. Trágicamente esta muy ocupado jugando Wordle como para crear el programa por lo que le pide a maestros de la programación (ustedes) de interredes cibernéticas (redes) que lo hagan por él.

2.1. Concepto General

Deben programar una versión multijugador del juego de palabras **Wordle**. En un principio, los jugadores deben ser enviados a un **Lobby** donde esperan a que su partida sea configurada y que esta comience. Después, deben jugar a adivinar la misma palabra secreta en paralelo, con el objetivo de determinar un ganador.

2.2. Conexión inicial e inicio del juego (20 puntos)

- **Conexión inicial** - El servidor debe levantarse y esperar a que algún jugador se conecte. Una vez conectado el jugador, este debe enviarle su nombre al servidor y luego ser enviado al **Lobby**. El servidor siempre debe estar atento a alguna posible conexión de un nuevo jugador, sin dejar nunca de atender a los usuarios ya conectados.¹ Es importante que la atención de múltiples clientes sea robusta.
- **Lobby** - Al conectarse el primer cliente, este queda a cargo como **líder** del **Lobby**, lo que quiere decir que este cliente es quien puede configurar y dar inicio a la partida de **Ruzdle** cuando él desee. Conexiones posteriores de nuevos clientes también son enviados al **Lobby**, pero estos no pueden ni configurar la partida ni darle inicio. Se les debe notificar a todos los clientes cada vez que uno nuevo se conecte, junto con la cantidad de clientes conectados actualmente.

Previo a iniciar una partida, el cliente **líder** elige si jugar una palabra al azar, donde el servidor elige una palabra dentro del listado de posibles palabras de cinco letras para comenzar el juego, estas palabras se encuentran en el

¹ Se recomienda el uso de **threads**, **select()**, **poll()**, **epoll()** u otra alternativa de POSIX para hacer *multiplexing*, mientras sea razonable.

archivo `palabras.txt`. Alternativamente, el cliente puede decidir que palabra jugar enviándole al servidor el índice de la palabra, donde la primera línea del archivo corresponde a la palabra de índice 0. Una vez configurado esto, puede decirle al servidor que inicie la partida (queda a decisión de ustedes como hacer esto, se debe especificar en el `Readme.md`).

Si otro cliente quiere conectarse al servidor mientras ya hay una partida en curso, el servidor **debe** aceptar dicha conexión, solo para enviarle un mensaje a dicho cliente, notificándole que lamentablemente ya hay una partida en curso, para luego expulsarlo y cerrar su conexión. Esta interacción debe ocurrir mediante comunicación cliente servidor, el cliente **no puede** responder sin comunicarse con el servidor primero.

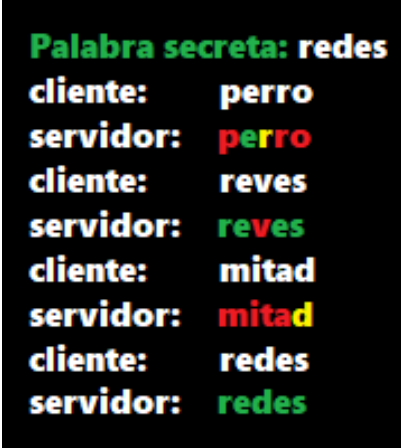
2.3. Ruzdle

En las partidas de **Ruzdle**, el servidor hace **TODO** el calculo y la lógica detrás del juego, el cliente solamente actúa como cliente *dumb*, de no respetar esto se obtiene puntaje mínimo en esta sección.

2.3.1. Flujo del juego (20 puntos)

Al iniciarse la partida, los clientes de manera **asíncrona** hacen intentos de adivinar la palabra secreta, para hacer un intento, un cliente debe enviar una palabra de **cinco letras**, que esté dentro del listado de palabras válidas al servidor. Estas palabras están en inglés y se encuentran en el archivo `palabras_validas.txt`. Una vez hecho el intento, el servidor debe enviarle al cliente todas las letras de la palabra recién enviadas que están en la palabra secreta y las letras que están en la palabra secreta y además están en la posición correcta, haciendo una distinción visible entre estas.

Ejemplo (referencia, no es necesario que sea igual)



```
Palabra secreta: redes
cliente:   perro
servidor:  perro
cliente:   reves
servidor:  reves
cliente:   mitad
servidor:  mitad
cliente:   redes
servidor:  redes
```

Figure 2: ejemplo redes

En este ejemplo la palabra secreta elegida por el servidor es 'redes', luego el cliente le pregunta si la palabra secreta es 'perro' y el servidor responde con la misma palabra pero marcando de color verde las letras que están en la palabra secreta y que además están en el lugar correcto, en este caso la letra 'e' es la única que cae en esta categoría. Por otra parte el servidor también marca las letras que están en la palabra secreta y en 'perro', pero que no están en la posición correcta, estas son marcadas en amarillo, para esta palabra se marcan las letras 'r' y 'r'. (**NOTA:** ojo que la palabra 'redes' tiene una sola 'r' y la palabra 'perro' tiene 2 'r's, el servidor debe devolver solamente la primera aparición de la 'r' marcada de color amarillo. Si devolviera las dos 'r' amarillas, daría a entender que la palabra secreta contiene al menos dos letras 'r', y no es el caso).

Pueden distinguir estas letras como sea mientras quede claro que letra está equivocada, cuál está en la palabra y cuál está en la posición correcta. Se debe explicar al principio de cada juego (en el cliente) cómo es este formato recién

descrito. Pueden investigar como utilizar colores en una terminal pero no es estrictamente necesario.

Una forma clara de distinguir entre letras es la siguiente:

```
cliente: perro
servidor:
      X-O-?-X-X
      P E R R O
```

A diferencia del Wordle convencional, en **Ruzzle** no hay límite de intentos para adivinar la palabra, sin embargo, para ganar, un cliente debe intentar adivinarla en el menor número de intentos posibles (ver sección 2.3.2). El juego termina cuando todos los clientes hayan adivinado la palabra secreta o se hayan rendido. Los jugadores que vayan terminando, entran a **modo espectador**, donde el **servidor** les va actualizando que jugadores ya han terminado y quienes no, junto con un contador de los intentos de todos. Notar que antes de entrar a **modo espectador**, los clientes no tienen como saber como van sus rivales.

Un jugador se rinde al escribir el comando `\exit`, así admite que no pudo completar la palabra y por lo tanto obtiene puntaje igual a 0 (ver sección 2.3.2).

Notar que `\exit` es un comando especial que se envía al servidor, por lo que se debe distinguir entre el manejo de una palabra normal y este comando. En este proyecto, no hay más comandos de este tipo.

2.3.2. Criterio para ganar

Cuando todos los jugadores hayan terminado, se calcula su puntaje a partir de la siguiente formula:

$$puntaje = \left\lfloor \frac{10000}{n \times (1 + \sum X_i)} \right\rfloor$$

Donde n es el número total de turnos utilizados para adivinar la palabra y $\sum X_i$ es la suma de la cantidad de letras incorrectas (no presentes en la palabra) obtenidas en cada turno. El ganador de la partida es el jugador que haya obtenido el mayor puntaje de la partida.

Tomando el ejemplo de la figura 2, en el ejemplo se tiene como palabra secreta 'redes'. En esta partida el cliente utiliza 4 turnos (n) hasta adivinar la palabra secreta y se equivoca en 8 letras ($\sum X_i$), la sumatoria de letras rojas que hay en la imagen). Reemplazando eso en la formula se tiene que:

$$puntaje = \left\lfloor \frac{10000}{4 \times (1 + 8)} \right\rfloor$$
$$puntaje = 277$$

2.3.3. Fin del juego (10 puntos)

El juego termina cuando todos los jugadores tienen un puntaje asociado a ellos, es decir, todos los jugadores han terminado, se han rendido o una combinación de estos casos. Una vez termina el juego se le pregunta al jugador si quiere volver al **Lobby**, registrar su puntaje (ver Bonus 4.2) o salir. Si se elige volver al Lobby, el nuevo líder será el primer jugador en volver al mismo.

3. Implementación

3.1. Cliente

El cliente debe recibir e imprimir los menús y/o mensajes correspondientes y debe enviar los inputs al servidor. El cliente no debe hacer ningún cálculo luego de hacer la conexión con el servidor.

3.2. Servidor

El servidor debe mediar la comunicación entre los clientes. El servidor es el encargado de procesar **toda** la lógica del juego y comunicación entre clientes. Cuando el cliente deba entregar algún tipo de *input*, es el servidor el encargado de pedirlo.

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

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.

A modo de ejemplo, consideren que queremos enviar el mensaje `Hola` con el ID 5. Si serializamos el Payload según [ASCII](#), su tamaño correspondería a cuatro *bytes*. El paquete completo se vería así:

```
00000101 00000100 01001000 01101111 01101100 01100001
      5           4           H           o           l           a
```

3.4. Ejecución (3 puntos)

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 numérico de IPv4, 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 15 décimas!)

4.1. *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 ejecutándose 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.

4.2. *Leaderboard* (+5 décimas)

Se deberá generar una tabla de puntajes (o leaderboard) en donde se listen de forma ordenada los 10 puntajes mas altos registrados por el servidor, las columnas de la tabla deben ser: Posición (1, 2, 3, ..., 9, 10), Puntaje obtenido, Nombre del jugador, Palabra del puntaje. La tabla de puntajes debe ser generada por el servidor y enviada a todos los clientes una vez hayan terminado la partida, esto mostrará en la terminal del cliente la tabla de puntajes.

Ejemplo:

1	10000	Ruz	Redes
2	500	Amongas	Farce
3	250	Joto	Ratio
4	200	Tomás	Table
5	150	Alvaro	Times
6	100	Lukas	Lukas
7	50	AAA	Zombi
8	10	Bromas	Blood
9	1	Gamer	Grape
10	0	Felipe	Gecko

4.3. *Descarga de Imagen* (+5 décimas)

Cuando un jugador gane una partida, el servidor le envía una imagen bonita para felicitarlo. Esta imagen debe quedar guardada en el mismo directorio donde se ejecuta el binario del cliente.

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 un directorio con nombre P2 (mayúscula la P) dentro del directorio de UNO de los integrantes del grupo en el servidor del curso. Además, deberán incluir:

- Un archivo `README.md` que indique quiénes son los autores del proyecto (**con sus respectivos números de alumno**), instrucciones para ejecutar y usar 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](#).

- Uno o dos archivos `Makefile` 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.

6. Descuentos

6.1. Descuentos (¡hasta 20 décimas!)

- 5 décimas por subir archivos binarios (programas compilados).
- 5 décimas por no incluir el/los `Makefiles`, o bien incluirlos y que no funcionen.
- 5 décimas por tener *memory leaks*. (Se recomienda fuertemente utilizar **Valgrind**).
- 5 décimas por la presencia archivos correspondientes a entregas del curso pasadas en el mismo directorio.

6.2. SÚPER DESCUENTO (¡La mitad de la nota!)

Si por cualquier motivo su proyecto no funciona ocupando sockets esto generará que cualquier puntaje que se haya obtenido se reduzca a la mitad. (¡Como hay puntaje por la conexión inicial esto significa que el 4 es imposible!)