Ahorcado

Ejercicio N° 1

Objetivos	● Buenas prácticas en programación de Tipos de Datos Abstractos (TDAs) ● Modularización de sistemas ● Correcto uso de recursos (memoria dinámica y archivos) ● Encapsulación y manejo de Sockets y Strings en C
Instancias de Entrega	**Entrega 1:** clase 4 (28/09/2021). **Entrega 2:** clase 6 (12/10/2021).
Criterios de Evaluación	● Criterios de ejercicios anteriores. ● Cumplimiento de la **totalidad** del enunciado del ejercicio incluyendo el **protocolo** de comunicación y/o el **formato** de los archivos y salidas. ● Correcta **encapsulación** en TDAs, ofreciendo una interfaz que **oculte** los detalles de implementación (por ejemplo que **no** haya un *get_fd()* que exponga el *file descriptor* del TDA Socket). ● **Ausencia** de variables globales y *static* y de funciones globales que no pertenezcan a un TDA salvo la función *main()*. Cualquier excepción debe estar *debidamente justificada* (por ejemplo la función *max()* que no tiene estado y no pertenece a un TDA concreto). ● Código **ordenado**, separado en archivos .c y .h, con **funciones cortas** y con la **documentación** pertinente. ● Empleo de memoria dinámica (*heap*) justificada. Siempre que se pueda **hacer uso del *stack*,** hacerlo antes que el del *heap*. Dejar el *heap* sólo para reservas grandes o cuyo valor sólo se conoce en *runtime* (o sea, es dinámico). Por ejemplo hacer un *malloc(4)* esta mal, seguramente un *char buf[4]* en el *stack* era suficiente. ● Acceso a información de archivos de forma ordenada y moderada

***El trabajo es personal**: debe ser de autoría completamente tuya. Cualquier forma de **plagio es inaceptable**: copia de otros trabajos, copias de ejemplos de internet o copias de tus trabajos anteriores (self-plagiarism).*
Si usas material de la cátedra deberás dejar en claro la fuente y dar crédito al autor (a la materia).

Índice

Introducción

Este ejercicio consiste en programar el clásico juego del ahorcado, **de manera distribuida**: Hay un servidor con la lógica del juego, y los clientes se conectan para iniciar una partida. El servidor puede atender a varios clientes, 1 a la vez.

Descripción

Reglas del Juego

El Ahorcado es un juego de 2 jugadores, donde uno piensa una palabra secreta, y el otro intenta adivinarla proponiendo letras. El jugador que piensa la palabra secreta debe brindar una información inicial al otro sobre cuántas letras tiene la palabra. Se cuenta con una cantidad de intentos **N**, que *irán disminuyendo a medida que el jugador propone una letra incorrecta*. La partida termina si el jugador logró descifrar la palabra, o si se quedó sin intentos.

Ayuda: Empezar por un TDA que represente una partida del ahorcado con una buena interfaz y testearla.

Partidas Online

En este trabajo, el servidor actuará del jugador que piensa la palabra secreta. Esta palabra proviene de un archivo de texto que se le pasa por parámetro. El servidor aceptará de a 1 cliente a la vez para jugar una partida por cada línea que lee del archivo. Una vez finalizado el archivo, el servidor termina ordenadamente. Al finalizar, se muestra una estadística de las partidas jugadas, indicando cuántas victorias tuvieron los clientes, y cuantas derrotas.

Interfaz de Usuario

El cliente mostrará la información del juego en tres líneas, donde se detalla los intentos restantes, la palabra adivinada hasta el momento, y una solicitud para ingresar la siguiente letra :

```
Palabra secreta: g___
Te quedan N intentos
Ingrese letra:
```

En este caso el cliente adivinó la primera letra de la palabra, compuesta de 4 letras, y le quedan todos los intentos.

Ayuda: No copy-pastear del PDF, a veces incluye caracteres invisibles indeseables.

*Ayuda: La palabra secreta puede tener una longitud variable, y viene dado por un archivo que se verá a continuación. Se recomienda el uso de la función **getline** para leer dicha palabra.*

*Ayuda: El cliente puede ingresar más de un caracter, si esto sucede, el cliente **procesa secuencialmente***

cada letra ingresada, como si el cliente hubiese ingresado de a una letra a la vez.

Una vez que la partida finaliza, el servidor envía un último mensaje, indicando cuál era la palabra secreta y los intentos restantes. El cliente usará estos datos para imprimir los siguientes mensajes de finalización:

- “Ganaste!!”, si el jugador adivinó la palabra.
- “Perdiste! La palabra secreta era: **<palabra secreta>**”, si el jugador no adivinó la palabra.

Repositorio de palabras

El servidor recibirá por **entrada línea de comandos** la dirección al archivo de repositorio de palabras. Este archivo **de texto** contiene una palabra en cada línea, de *longitud variable*. El servidor irá leyendo de una línea a la vez, y aceptará un cliente. Cuando termina de leer el archivo, el servidor finaliza su ejecución, imprimiendo antes el resumen de las partidas jugadas. Se asume que el repositorio de palabras se compone de **líneas con una sola palabra**, en **lowercase**. No es necesario chequear la consistencia del archivo.

Un ejemplo de este archivo puede ser el siguiente:

```
gato
casa
habitual
casero

```

*Ayuda: El repositorio de palabras termina con un salto de línea siempre. Leer la documentación de **getline**.*

En este caso, el servidor **podrá atender hasta 4 clientes** antes de su finalización.

Formato de Línea de Comandos

La entrega de este trabajo se compone de dos ejecutables, “client” y “server”. El servidor se ejecuta de la siguiente manera:

```
./server <port> <N> <path al repositorio de palabras>
```

Ayuda: ¿Qué pasa si le paso un N inválido?

Y el cliente de esta otra manera:

```
./client <host> <port>
```

Códigos de Retorno

Tanto cliente como servidor retornarán un 0 en caso de que la ejecución haya sido exitosa, o un 1 si hubo un error en los parámetros.

Se permite al alumno decidir sobre el uso de otros códigos de retorno para otros tipos de error que detecte. Sin embargo, se asegura que los casos de prueba no forzará ese tipo de errores, sino que usarán “caminos felices”.

Entrada y Salida Estándar

Server

El servidor no espera ningún ingreso por la entrada estándar, ni tampoco imprimirá nada en la salida estándar de error. Sin embargo, por la salida estándar se imprimirá un resumen de las estadísticas de las partidas jugadas de la siguiente forma.

```
Resumen:
      Victorias: <cantidad de victorias de los clientes>
      Derrotas: <cantidad de derrotas de los clientes>
```

Notar que la segunda y tercer línea tiene una tabulación al principio y un salto de línea al final

```
Resumen:
\tVictorias: <cantidad de victorias de los clientes>\n
\tDerrotas: <cantidad de derrotas de los clientes>\n
```

Client

El cliente recibirá las **propuestas de letras** por **entrada estándar**, y mostrará las respuestas del servidor por **salida estándar**.

El cliente no utilizará la **salida estándar de error**.

Protocolo de Comunicación

El cliente enviará mensajes al servidor de tamaño un byte, con la letra propuesta para adivinar la palabra en formato ‘ascii’.

Ayuda: se asegura que los tests del sercom siempre usará caracteres ascii correspondiente a letras en minúsculas (de la ‘a’ a la ‘z’) y ***el caracter especial ‘enter’****. Queda a cargo del alumno qué hacer cuando el cliente ingresa un caracter “inválido”.*

Por ejemplo, si el usuario ingresa la letra ‘a’, el cliente enviará un byte al servidor, de valor **0x61**. El servidor recibirá este comando, lo procesa y le contesta al cliente de la siguiente forma:

- El primer byte se compone de dos datos, el primero es un flag de terminación de partida, será el **bit mas significativo** puesto en 0 si la partida no terminó, o en 1 si la partida terminó. Los otros 7 bits corresponden a la cantidad de intentos restantes que le queda al cliente para adivinar la palabra.
- Luego, se envían **2 bytes en formato big endian**, que representa el largo del string del mensaje.
- Todos los bytes del mensaje (la palabra secreta parcialmente adivinada, o la palabra secreta cuando finaliza la partida) **sin contar el ‘\0’**.

Por ejemplo, supongamos que la palabra secreta es ‘casa’, y que solamente se adivinó la letra ‘a’. Si la partida no terminó, y se tienen todos los intentos (donde se configuró que los intentos son 5), el servidor mandará (la representamos en hexa):

```
05 00 04 5f 61 5f 61
```

Es decir, del primer byte, 0x05 (0000 0101 en binario) tiene el bit más significativo en cero, por lo que la partida no terminó. Por otro lado, los 7 bits restantes indican que quedan 5 intentos. Luego viene el tamaño del mensaje, en forma de entero de dos bytes en big endian. Finalmente, se envía la palabra parcialmente adivinada. El string que representa esos cuatro bytes corresponde a “_a_a”.

Una vez finalizada la partida, el último mensaje del servidor será (considerando que los intentos restantes siguen siendo 5).

```
85 00 04 63 61 73 61
```

Ayuda: el servidor pareciera no indicar nunca si el cliente ganó o no la partida. Esto se puede deducir de la información enviada ¿cómo?

Ejemplos de Ejecución

Vamos a ver un ejemplo completo de cómo debería comportarse nuestro sistema.

Primero, vamos a ejecutar el servidor en el puerto 7777:

```
./server 7777 5 words.txt
```

Supongamos que el repositorio de palabras cuenta con la siguiente información

```
casa
gato

```

Luego se ejecuta el cliente

```
./client localhost 7777
```

El cliente se conecta con el servidor, y este le envía el mensaje inicial de la partida.

```
05 00 04 5f 5f 5f 5f
```

Es decir: la partida sigue activa, quedan 5 intentos, la palabra está compuesta de 4 caracteres y son todos '_' porque todavía no se adivinó nada.

El cliente recibirá este mensaje e imprimirá el mensaje inicial.

```
Palabra secreta: ____
Te quedan 5 intentos
Ingrese letra:
```

Supongamos ahora que el usuario teclea la letra 'c' y presiona enter. El cliente recibe esa letra por la entrada estándar, y la envía al servidor en formato ascii.

```
63
```

El servidor recibe este mensaje, lo procesa, y, como la letra 'c' se encuentra en la palabra secreta, no reduce la cantidad de intentos. Le contesta al cliente con la información actualizada

```
05 00 04 63 5f 5f 5f
```

Y el cliente imprime lo siguiente:

```
Palabra secreta: c___
Te quedan 5 intentos
Ingrese letra:
```

Ahora el cliente ingresa la la letra 'o', enviando al servidor el siguiente paquete

```
6f
```

El servidor lo recibe, y como esa letra no se encuentra en la palabra secreta, decrementa en 1 la cantidad de intentos. La respuesta que envía al cliente será

```
04 00 04 63 5f 5f 5f
```

Y el cliente imprime

```
Palabra secreta: c___
Te quedan 4 intentos
Ingrese letra:
```

Supongamos ahora que el cliente ingresa la letra ‘a’, enviando el ascii 0x61, el servidor lo procesa y devuelve

```
04 00 04 63 61 5f 61
```

Y en el cliente se imprime

```
Palabra secreta: ca_a
Te quedan 4 intentos
Ingrese letra:
```

Finalmente, el cliente envía la letra ‘s’, de ascii 0x73, y el servidor responde de la siguiente forma

```
84 00 04 63 61 73 61
```

Es decir, la partida terminó, quedaron 4 intentos, el mensaje consta de 4 bytes con la palabra secreta decodificada ‘casa’. El cliente imprimirá un último mensaje

```
Ganaste!!
```

El cliente terminará ordenadamente su ejecución. Por otro lado, como el servidor tiene más palabras en su repositorio, espera que se conecte otro cliente para jugar con la palabra secreta ‘gato’. Supongamos que el cliente se conecta, y envía siempre la letra ‘c’ (0x63) que no se encuentra en la palabra secreta. El servidor irá disminuyendo el contador de intentos y una vez que ese contador llegue a cero, terminará la partida.

Cliente

```
Palabra secreta: ____
Te quedan 5 intentos
Ingrese letra:
```

```
63
```

Servidor

```
04 00 04 5f 5f 5f 5f
```

Cliente

```
Palabra secreta: ____
Te quedan 4 intentos
Ingrese letra:
```

```
63
```

Servidor

```
03 00 04 5f 5f 5f 5f
```

Cliente

```
Palabra secreta: ____
Te quedan 3 intentos
Ingrese letra:
```

```
63
```

Servidor

```
02 00 04 5f 5f 5f 5f
```

Cliente

```
Palabra secreta: ____
Te quedan 2 intentos
Ingrese letra:
```

```
63
```

Servidor

```
01 00 04 5f 5f 5f 5f
```

Cliente

```
Palabra secreta: ____
Te quedan 1 intentos
Ingrese letra:
```

Finalmente, el cliente vuelve a enviar el ascii 0x63, el servidor finalizará la partida y le contesta al cliente el siguiente mensaje antes de finalizar la conexión.

```
80 00 04 67 61 74 70
```

Y el cliente imprimirá un último mensaje.

```
Perdiste! La palabra secreta era: 'gato'
```

El cliente terminará su ejecución ordenadamente, y el servidor, al no tener más palabras en su repositorio, imprime el resumen de resultados

```
Resumen:
      Victorias: 1
      Derrotas: 1
```

Y luego cierra ordenadamente.

Recomendaciones

Los siguientes lineamientos son claves para acelerar el proceso de desarrollo sin pérdida de calidad:

1. Tener siempre a mano las páginas de manual, y al abrirlas leerlas con mucha atención. Sobre todo para las funciones que no hayas usado nunca: no cuesta nada escribir '*man send*' o '*man recv*' en la consola y leer *bien* qué dice cada una sobre los valores del último parámetro o sobre los valores de retorno.
2. **¡Repasar los temas de la clase!** Los videos, las diapositivas, los handouts, las guias, los ejemplos, los tutoriales.
3. **¡Programar por bloques!** No programes todo el TP y esperes que funcione. Debuggear un TP completo es más difícil que probar y debuggear sus partes por separado.
   Dividir el TP en bloques, codearlos, testearlos por separada y luego ir construyendo hacia arriba.
   ¡Si programas así, podés hasta hacerles tests fáciles antes de entregar!. Teniendo cada bloque, es fácil armar un bloque de más alto nivel que los use. La **separación en TDAs** es crucial.
4. **Escribí código claro**, sin saltos en niveles de abstracción, y que puedas leer entendiendo qué está pasando. Si editás el código "*hasta que funciona*" y cuando funcionó lo dejás así, **buscá la explicación de por qué anduvo**.
5. **Documentá** a medida que desarrolles. No gastes tiempo en documentar funciones triviales, documenta las funciones o métodos y los TDAs más importantes. En el informe también **escribí claro**, en impersonal, y evitando opiniones.
6. No te compliques la vida con diseños complejos. **Cuanto más fácil sea tu diseño, mejor**.

Restricciones

La siguiente es una lista de restricciones técnicas exigidas por el cliente:

1. El sistema debe desarrollarse en C (C11) con el estándar POSIX 2008.
2. Está prohibido el uso de variables globales.
3. Está prohibido el uso de funciones globales.
4. Todo socket utilizado en este TP debe ser **bloqueante** (es el comportamiento por defecto).
5. No se puede cargar todo el archivo del repositorio de palabras en memoria.

6. Seguir las *ayudas*

Referencias

[1] Ahorcado: https://es.wikipedia.org/wiki/Ahorcado_(juego)
[2] getline: https://man7.org/linux/man-pages/man3/getline.3.html
[3] htons: https://linux.die.net/man/3/htons