Sudoku Remoto

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
Instancias de Entrega	**Entrega 1:** clase 4 (10/09/2019). **Entrega 2:** clase 6 (24/09/2019).
Temas de Repaso	● Uso de structs y typedef ● Uso de macros y archivos de cabecera ● Funciones para el manejo de Strings en C ● Funciones para el manejo de Sockets
Criterios de Evaluación	● Criterios de ejercicios anteriores ● Cumplimiento de la totalidad del enunciado del ejercicio ● Ausencia de variables globales ● Ausencia de funciones globales salvo los puntos de entrada al sistema (*main*) ● Correcta encapsulación en TDAs y separación en archivos ● Uso de interfaces para acceder a datos contenidos en TDAs ● Empleo de memoria dinámica de forma ordenada y moderada ● Acceso a información de archivos de forma ordenada y moderada

Índice

Introducción

Formato de Línea de Comandos
- Servidor
- Cliente

Códigos de Retorno

Entradas y Salidas Estándar
- Servidor
- Cliente
- Salidas estándar de errores
- Representación del tablero

Comandos
- PUT
- VERIFY
- RESET
- GET
- EXIT

Protocolo de comunicación
- Cliente a Servidor
- Servidor a Cliente

Ejemplo de Ejecución
- Archivo de Entrada
- Ejecución del Servidor
- Ejecución del Cliente
- Comando GET
- Comando PUT
- Comando VERIFY
- Comando RESET
- Comando EXIT

Sugerencias y Recomendaciones

Restricciones

Introducción

Se deberá implementar un sistema cliente-servidor para jugar al milenario Sudoku de manera remota. Para ello se desarrollará un solo programa que podrá ejecutarse en dos modos (cliente y servidor)
En el modo servidor deberá introducirse la lógica del juego, las reglas, el tablero y demás, mientras que el modo cliente servirá para que los jugadores puedan conectarse al sistema e interactuar con él.
Para simplificar el desarrollo, el servidor solamente aceptará una única conexión, atenderá todos sus pedidos hasta que se cierre remotamente, y terminará la ejecución ordenadamente, liberando todos sus recursos.
El Sudoku es un juego de ingenio que se juega en un tablero de 9x9 celdas. Este tablero se encuentra dividido en 9 sectores de 3x3 celdas cada uno, y consiste en lograr que cada sector, cada fila y cada columna del tablero contenga todos y cada uno de los números naturales del 1 al 9.
Cada instancia del juego inicia con celdas prefijadas que sirven como pistas, y que no pueden ser cambiadas durante el juego. El objetivo del juego es rellenar el resto de las celdas de una manera compatible con esas pistas fijas.

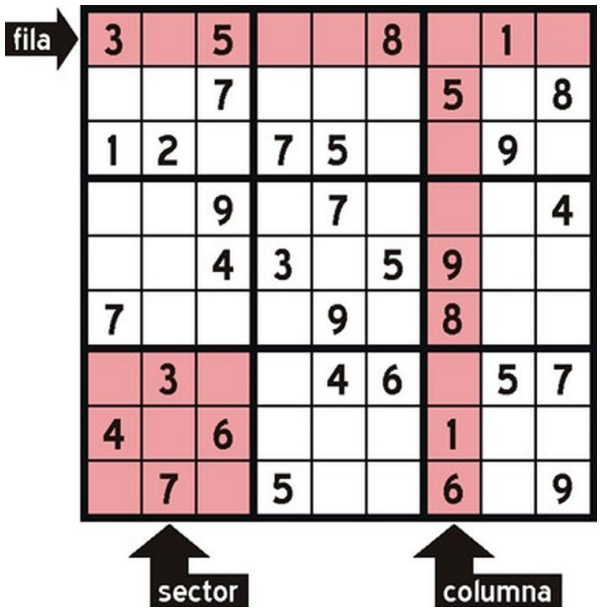

Imagen 1. Ejemplo de tablero con los nombres de cada conjunto de celdas.

Formato de Línea de Comandos

Servidor

Para ejecutar el servidor, se deberá iniciar el programa indicando el modo correspondiente e indicar en qué puerto debe escuchar las conexiones:

```
./tp <modo> <puerto>
```

Un ejemplo sería:

```
./tp server 7777
```

Cliente

El cliente se ejecutará de una manera similar, pero también se deberá indicar el hostname del servidor:

```
./tp <modo> <hostname-servidor> <puerto>
```

Un ejemplo sería:

```
./tp client localhost 7777
```

Notar que no solamente se deben soportar direcciones IP, sino que se deben resolver nombres (usar **getaddrinfo**).

Códigos de Retorno

En ambos modos, se debe retornar un 0 ante una ejecución exitosa, o un 1 ante cualquier error que impida la correcta ejecución del programa.
Se considera una ejecución exitosa aquella que logra levantar ambos procesos, conectarlos mediante el uso de sockets, opcionalmente intercambiar algún mensaje, y liberar los recursos ordenadamente.

Entradas y Salidas Estándar

Servidor

- Entrada estándar: No se utilizará la entrada estándar en el servidor
- Salida estándar: No se verificará el contenido de la salida estándar, por lo que se recomienda utilizarla con fines de debugging en caso de que localmente el programa funcione bien, pero en Sercom tenga algún inconveniente.

Cliente

- Entrada estándar: Se utilizará para ingresar los comandos (ver sección de comandos más adelante).
- Salida estándar: Se deben imprimir las respuestas provenientes del servidor, o los errores especificados en cada comando (ver sección de comandos para mayor detalle)

Salidas estándar de errores

En caso de errores en la ejecución del programa se debe imprimir un error descriptivo del mismo. Si el error no se detalla a continuación, significa que no hay casos de prueba relacionados (y esto implica que si ocurren hay un problema en la implementación) .

- Error en la entrada de parámetros: Se deberá imprimir un mensaje con el uso del programa:
  Si el primer parámetro es "`server`", imprimir
  "`Uso: ./tp server <puerto>\n`"
  Si el primer parámetro es "`client`", imprimir
  "`Uso: ./tp client <host> <puerto>\n`"
  Si no hay parámetros o el mismo no es ninguno de los anteriores, imprimir
  "`Modo no soportado, el primer parámetro debe ser server o client\n`"

Representación del tablero

El servidor leerá el tablero a partir de un archivo llamado "board.txt" ubicado en la carpeta donde esté situada la consola en la que se ejecute, y será siempre el mismo durante toda la ejecución de dicho proceso. El archivo de texto "board.txt" constará de nueve líneas de números separados por espacios, donde cada número positivo representará el valor de una celda prefijada, y las celdas vacías serán representadas con un valor 0. Por ejemplo, si el estado inicial del sudoku es el visto en la introducción, el archivo "board.txt" tendría el siguiente contenido:

```
3 0 5 0 0 8 0 1 0
0 0 7 0 0 0 5 0 8
1 2 0 7 5 0 0 9 0
0 0 9 0 7 0 0 0 4
0 0 4 3 0 5 9 0 0
7 0 0 0 9 0 8 0 0
0 3 0 0 4 6 0 5 7
4 0 6 0 0 0 1 0 0
0 7 0 5 0 0 6 0 9
```

Siempre que se deba imprimir el tablero en salida estándar, se debe hacer de la siguiente manera:

```
U===========U===========U===========U
U X | X | X U X | X | X U X | X | X U
U---+---+---U---+---+---U---+---+---U
U X | X | X U X | X | X U X | X | X U
U---+---+---U---+---+---U---+---+---U
U X | X | X U X | X | X U X | X | X U
U===========U===========U===========U
U X | X | X U X | X | X U X | X | X U
U---+---+---U---+---+---U---+---+---U
U X | X | X U X | X | X U X | X | X U
U---+---+---U---+---+---U---+---+---U
U X | X | X U X | X | X U X | X | X U
U===========U===========U===========U
U X | X | X U X | X | X U X | X | X U
U---+---+---U---+---+---U---+---+---U
U X | X | X U X | X | X U X | X | X U
U---+---+---U---+---+---U---+---+---U
U X | X | X U X | X | X U X | X | X U
U===========U===========U===========U
```

Cada 'X' será reemplazada por el número correspondiente si está presente, o un espacio si no lo está. Usando el mismo ejemplo que antes:

```
U===========U===========U===========U
U 3 |   | 5 U   |   | 8 U   | 1 |   U
U---+---+---U---+---+---U---+---+---U
U   |   | 7 U   |   |   U 5 |   | 8 U
U---+---+---U---+---+---U---+---+---U
U 1 | 2 |   U 7 | 5 |   U   | 9 |   U
U===========U===========U===========U
U   |   | 9 U   | 7 |   U   |   | 4 U
U---+---+---U---+---+---U---+---+---U
U   |   | 4 U 3 |   | 5 U 9 |   |   U
U---+---+---U---+---+---U---+---+---U
U 7 |   |   U   | 9 |   U 8 |   |   U
U===========U===========U===========U
U   | 3 |   U   | 4 | 6 U   | 5 | 7 U
U---+---+---U---+---+---U---+---+---U
U 4 |   | 6 U   |   |   U 1 |   |   U
U---+---+---U---+---+---U---+---+---U
U   | 7 |   U 5 |   |   U 6 |   | 9 U
U===========U===========U===========U
```

Comandos

PUT

Este comando sirve para ubicar un número en una posición del tablero. Se debe ejecutar cuando el cliente lee de entrada estándar un texto como el siguiente:

```
put <numero> in <fila>,<columna>
```

Por ejemplo:

```
put 4 in 9,9
```

Si el cliente lee de entrada estándar el texto anterior, debe enviar un mensaje de put (ver cómo en la sección del protocolo), indicando que quiere escribir un 4 en la celda de la fila 9 y columna 9.
Nota: Los índices de filas y columnas van del 1 al 9, y no del 0 al 8.

Posibles salidas:

- Error de índices: Si los índices no se encuentran en el rango válido, se debe imprimir el mensaje
  `"Error en los índices. Rango soportado: [1,9]\n"`
  En este caso no se debe enviar ningún mensaje al servidor.
- Error en el valor: Si el valor no está en el rango soportado, se debe imprimir un mensaje similar:
  `"Error en el valor ingresado. Rango soportado: [1,9]\n"`
  En este caso no se debe enviar ningún mensaje al servidor.
- Celda no modificable: Si la celda es una de las pistas prefijadas, no puede ser modificada durante la ejecución del programa, por lo que el servidor responderá con el siguiente mensaje, que deberá ser impreso en la salida estándar:
  `"La celda indicada no es modificable\n"`
- Si la celda era modificable, sin importar si ya tenía valor o no, el servidor debe modificar el valor al especificado y devolver una representación del tablero, que el cliente debe imprimir en salida estándar.

Nota: Con el objetivo de que no haya ambigüedades, en el caso de tener un comando put que tenga tanto errores de índices como de valor ingresado, se debe mostrar el mensaje correspondiente a los índices.

VERIFY

Este comando se utiliza para verificar si los valores actuales cumplen con todas las reglas.
Se debe ejecutar una verificación cuando el proceso cliente lee de entrada estándar lo siguiente:

```
verify
```

Posibles salidas:

- Si se están cumpliendo todas las reglas, se debe contestar con el texto "`OK\n`".
- Si hay algún número repetido en alguno de los tipos de conjunto, se debe devolver "`ERROR\n`".

RESET

El comando reset sirve para restablecer todas las celdas modificables, y volver al comienzo del juego.
Se debe ejecutar al leer:

```
reset
```

Siempre devuelve una representación del tablero, que debe ser mostrado en salida estándar.

GET

Se utiliza para obtener el tablero, tal como está en el servidor.
Se debe ejecutar al leer:

```
get
```

EXIT

Para salir del programa cliente, se debe escribir el siguiente comando en entrada estándar:

```
exit
```

También se debe terminar la ejecución del programa al terminar de leer la entrada estándar (EOF).
No se debe enviar nada al servidor en estos casos.

Protocolo de comunicación

Cliente a Servidor

Con el objetivo de minimizar el uso del ancho de banda en la subida del cliente, se utilizará un protocolo binario para enviar comandos del cliente al servidor.

- Para el comando **PUT**, se enviarán 4 bytes: `[ 'P', fila, columna, número ]`
  - El primer byte es un caracter 'P'
  - El resto de los bytes números enteros sin signo de un byte (ver **uint8_t**).
- En el comando **VERIFY**, se envía un caracter 'V'
- En el comando **RESET**, se envía un caracter 'R'
- En el comando **GET**, se envía un caracter 'G'

Servidor a Cliente

El servidor siempre devolverá directamente el texto que deberá ser impreso en la salida estándar del cliente, ya sea un mensaje de error, un mensaje de aceptación, o la representación del tablero que fué descrita antes. Para ello enviará:

- Primero, el largo del texto, como un entero sin signo de 4 bytes, en big endian (ver **uint32_t, htonl** y **ntohl**).
- Y luego el texto. **Tener cuidado de enviar exactamente los bytes que se dijo en el largo**.

Ejemplo de Ejecución

A continuación se mostrará un ejemplo representativo de ejecución, que engloba el caso base del juego.

Archivo de Entrada

Consideremos el archivo “board.txt” que usamos antes como ejemplo:

```
3 0 5 0 0 8 0 1 0
0 0 7 0 0 0 5 0 8
1 2 0 7 5 0 0 9 0
0 0 9 0 7 0 0 0 4
0 0 4 3 0 5 9 0 0
7 0 0 0 9 0 8 0 0
0 3 0 0 4 6 0 5 7
4 0 6 0 0 0 1 0 0
0 7 0 5 0 0 6 0 9
```

En este archivo podemos ver las ubicaciones de cada una de las celdas prefijadas, y su valor. Todos los ceros serán celdas vacías que deberá llenar el jugador.

Ejecución del Servidor

El nombre del archivo de entrada es fijo, por lo que no necesitaremos usar ningún parámetro adicional en el servidor. Por lo tanto, se deberá ejecutar el mismo usando la línea de comandos antes especificada:

```
./tp server 7777
```

Al ejecutar el programa en modo server, se cargará el archivo del juego a las estructuras de datos correspondientes en memoria, y se dejará un socket listo para aceptar conexiones en el puerto **7777**.

Ejecución del Cliente

Una vez levantado el servidor, se iniciará el cliente. Se deberá analizar qué sucede si no está levantado el servidor previamente, y manejar este tipo de errores (en Sercom no habrá pruebas al respecto, por lo que la manera de manejar esos errores queda a criterio del programador).

```
./tp client 127.0.0.1 7777
```

Una vez levantado, este segundo proceso deberá establecer una conexión con el servidor que está escuchando en el puerto **7777** del host cuya IP es **127.0.0.1**, y empezar a leer comandos de la entrada estándar, hasta que el usuario cierre dicho flujo o utilice el comando de salida.

Comando GET

Una posible entrada, que probablemente sea muy útil antes de iniciar a ingresar números, es el comando para obtener el estado del tablero. En el presente ejemplo, si el usuario ingresa por entrada estándar el comando correspondiente:

```
get
```

entonces el cliente deberá enviar por el socket un mensaje de un único byte con el caracter ‘G’ (0x47 en hexa, 71 decimal, 01000111 en binario, etcétera). Notar que al ser **un único byte** no tiene sentido hablar de endianness.

Cuando el servidor recibe ese mensaje, debe contestar con un texto que represente al tablero, pre-concatenando el largo de dicho texto. Este largo debe ser un entero de cuatro bytes, sin signo, y en big-endian. En el caso del ejemplo, el texto a devolver es el siguiente:

```
U===========U===========U===========U
U 3 |   | 5 U   |   | 8 U   | 1 |   U
U---+---+---U---+---+---U---+---+---U
U   |   | 7 U   |   |   U 5 |   | 8 U
U---+---+---U---+---+---U---+---+---U
U 1 | 2 |   U 7 | 5 |   U   | 9 |   U
U===========U===========U===========U
U   |   | 9 U   | 7 |   U   |   | 4 U
U---+---+---U---+---+---U---+---+---U
U   |   | 4 U 3 |   | 5 U 9 |   |   U
U---+---+---U---+---+---U---+---+---U
U 7 |   |   U   | 9 |   U 8 |   |   U
U===========U===========U===========U
U   | 3 |   U   | 4 | 6 U   | 5 | 7 U
U---+---+---U---+---+---U---+---+---U
U 4 |   | 6 U   |   |   U 1 |   |   U
U---+---+---U---+---+---U---+---+---U
U   | 7 |   U 5 |   |   U 6 |   | 9 U
U===========U===========U===========U
```

Como son 19 líneas de 37 caracteres, más sus respectivos saltos de línea, tenemos un total de 722 caracteres a enviar cuando enviamos un tablero. El 722 en hexa es 0x02D2, por lo que antes de enviar esos 722 bytes, se deberán enviar los bytes 0x00, 0x00, 0x02 y 0xD2.

Comando PUT

Una vez que el usuario vio impreso el tablero del sudoku, querrá empezar a jugar, ingresando uno a uno los números en donde crea conveniente. En el caso del ejemplo podemos ver que en la fila 5, columna 8 (recordar que los índices van de 1 a 9), se debe colocar un 7, ya que inspeccionando otras filas y columnas se puede deducir que dicho número no puede estar presente en ninguna otra celda del sector.

Para que esto suceda, el usuario debe ingresar por entrada estándar:

```
put 7 in 5,8
```

Esto provocará que se envíe un mensaje de 4 bytes al servidor:

1) El caracter ‘P’
2) La fila: un 5 en este caso, representado en un entero sin signo de un solo byte (ver **uint8_t** en **stdint.h**)
3) La columna: un 8, representado como el 5 (otra vez, aquí no importa el endianness).
4) El número: un 7

Como esta entrada no tiene ningún problema, el servidor interpretará el comando, agregará un 7 en la celda correspondiente, y devolverá al cliente una representación del nuevo estado del tablero.

```
U===========U===========U===========U
U 3 |   | 5 U   |   | 8 U   | 1 |   U
U---+---+---U---+---+---U---+---+---U
U   |   | 7 U   |   |   U 5 |   | 8 U
U---+---+---U---+---+---U---+---+---U
U 1 | 2 |   U 7 | 5 |   U   | 9 |   U
U===========U===========U===========U
U   |   | 9 U   | 7 |   U   |   | 4 U
U---+---+---U---+---+---U---+---+---U
U   |   | 4 U 3 |   | 5 U 9 | 7 |   U
U---+---+---U---+---+---U---+---+---U
U 7 |   |   U   | 9 |   U 8 |   |   U
U===========U===========U===========U
U   | 3 |   U   | 4 | 6 U   | 5 | 7 U
U---+---+---U---+---+---U---+---+---U
U 4 |   | 6 U   |   |   U 1 |   |   U
U---+---+---U---+---+---U---+---+---U
U   | 7 |   U 5 |   |   U 6 |   | 9 U
U===========U===========U===========U
```

Nota: el color rojo del 7 es para evidenciar el cambio, no se debe imprimir en colores en la consola.

Comando VERIFY

En cualquier momento de la ejecución, se podrá enviar un comando de verificación, que sirve para chequear que no haya ningún conjunto de celdas que incumpla la regla de no repetir números.
En el caso de nuestro ejemplo, vemos que si en este momento el usuario ingresa el comando:

```
verify
```

entonces se deberá enviar un mensaje de un único byte, el código ascii de la ‘V’, que al ser interpretado por el servidor provocará que se verifiquen las reglas y se devuelva el texto “OK\n” al cliente. Notar que en este

caso el texto tiene 3 bytes, y que dicho 3 se deberá pre-concatenar al mensaje como un entero de 4 bytes, sin signo, y en big-endian.

Comando RESET

Si el usuario quiere volver a comenzar el juego, puede ingresar el comando de reinicio, que borrará todos los números **que hayan sido ingresados por él**, no pueden borrarse los números prefijados en el archivo de entrada. Si en el estado actual del ejemplo (solamente agregamos el 7) el usuario ingresa

```
reset
```

entonces se deberá enviar la letra 'R' y el servidor deberá restablecer cada celda no prefijada, terminando por devolver la representación del tablero original:

```
U===========U===========U===========U
U 3 |   | 5 U   |   | 8 U   | 1 |   U
U---+---+---U---+---+---U---+---+---U
U   |   | 7 U   |   |   U 5 |   | 8 U
U---+---+---U---+---+---U---+---+---U
U 1 | 2 |   U 7 | 5 |   U   | 9 |   U
U===========U===========U===========U
U   |   | 9 U   | 7 |   U   |   | 4 U
U---+---+---U---+---+---U---+---+---U
U   |   | 4 U 3 |   | 5 U 9 |   |   U
U---+---+---U---+---+---U---+---+---U
U 7 |   |   U   | 9 |   U 8 |   |   U
U===========U===========U===========U
U   | 3 |   U   | 4 | 6 U   | 5 | 7 U
U---+---+---U---+---+---U---+---+---U
U 4 |   | 6 U   |   |   U 1 |   |   U
U---+---+---U---+---+---U---+---+---U
U   | 7 |   U 5 |   |   U 6 |   | 9 U
U===========U===========U===========U
```

Comando EXIT

Cuando el usuario no desea seguir jugando, ya sea porque ganó o se rindió, puede ingresar el comando:

```
exit
```

En este caso no se debe enviar ningún mensaje al servidor (analizar cómo es que el servidor se entera entonces de que no debe seguir atendiendo al cliente) y debe cerrarse ordenadamente, evitando cualquier tipo de leak.
Otra manera de terminar con la ejecución del programa es con un fin de archivo (buscar ctrl+d para las pruebas manuales).

Sugerencias y Recomendaciones

- En caso de notar algún defecto en el enunciado o en las pruebas, por favor avisar lo antes posible para que sea corregido de la manera más inmediata.
- Encarar el trabajo de manera modular, en el orden que resulte más cómodo, pero no arrancar programando funciones demasiado extensas después recortarlas, ya que esa manera de encarar los problemas conduce a mucho retrabajo y bugs.
- Hacer un paneo de las preguntas de otros cuatrimestres antes de empezar a trabajar, ya que hay respuestas muy útiles para facilitar el trabajo.
- **No programar demás**, hacer módulos con responsabilidades claras y cohesivas, y evitar el acoplamiento. Definir las interfaces antes de las implementaciones.
- Las páginas de manual de unix son especialmente útiles para programar con sockets. (Leer **man send** y **man getaddrinfo**, por ejemplo).

Restricciones

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

1. El sistema debe desarrollarse en ISO C (C99).
2. Está prohibido el uso de variables y funciones globales. La función **main** no puede contener lógica del negocio, solamente debe servir como punto de entrada del flujo del programa.
3. Las funciones de más de 15 líneas requieren una justificación clara de su extensión.
4. El informe debe contener al menos un diagrama que represente alguna parte del diseño. No hace falta que sea UML, pero sí que sea descriptivo.
5. El protocolo de comunicación es obligatorio, no sugerido.