Título:

El juego del truco online

Ciclo Lectivo	2016		Curso	R1022

Integrantes:

Apellido y Nombres	Legajo	Dirección de correo	Usuario svn
Marinzalda, Federico	1433921	fmarinzalda@gmail.com	fmarinzalda
Chaves, Reina Celeste	1563531	reinachaves@frba.utn.edu.ar	celeste

Profesor titular: Ing. Marcelo Trujillo

Auxiliar/es y Docente/s:

Ing. Florencia Ferrigno
Sebastián Torres (ATP 2°)

Observaciones de entrega:

Índice

Descripción

El proyecto consta de una versión online para jugar al truco. En el mismo se encuentra un formulario de registración que pide nombre, apellido, mail, usuario y contraseña. Luego, para jugar el usuario registrado deberá loguearse para comenzar una nueva partida.
La visualización es una mesa, las cartas propias que son visibles al igual que las que se juegan y las del contrincante (no visibles), con los movimientos correspondientes de cartas por cada jugada y se verán las puntuaciones.

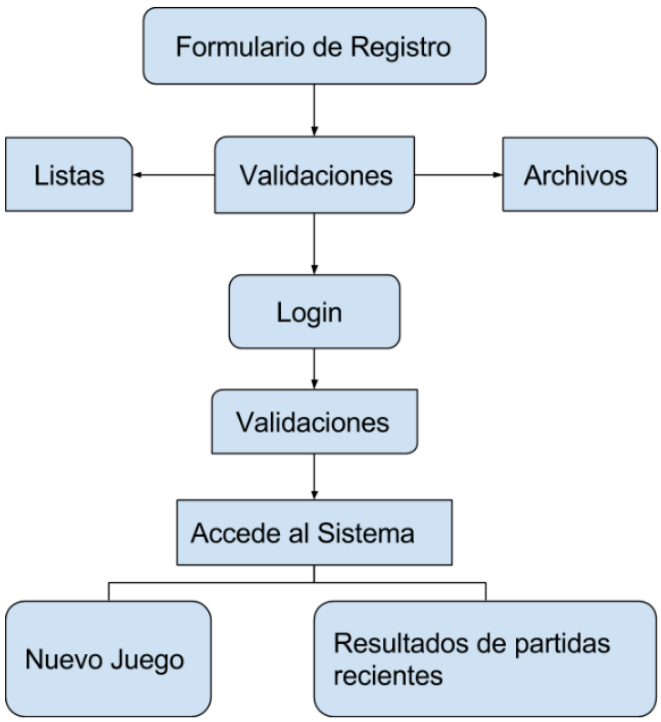
Objetivos

Aprender a usar manejo de strings, sockets, listas y archivos para el formulario de registración, login, jugar las partidas y los scores. Hacer las validaciones del formulario, logueo y movimientos de las cartas en las partidas.
Al momento de programar la parte gráfica de la mesa, las cartas con sus movimientos y el score, requiere de aprender el manejo de procesos y librerías gráficas tales como OpenCV y Allegro.

Temas involucrados

Tema	Si	No
Manejo de listas	Sí	
Manejo de archivos	Sí	
Manejo de sockets	Sí	
Manejo de procesos	Sí	
Manejo de strings	Sí	

Diagrama de bloques



El Proyecto consta de un formulario de Registro, el cual cuenta con los campos a llenar obligatoriamente de nombre, apellido, casilla de correo electrónico, usuario y contraseña. Cuando uno se registra se abre el archivo y se guarda al nuevo registrado en un nodo y luego se pasa ese nodo al archivo. Se valida los

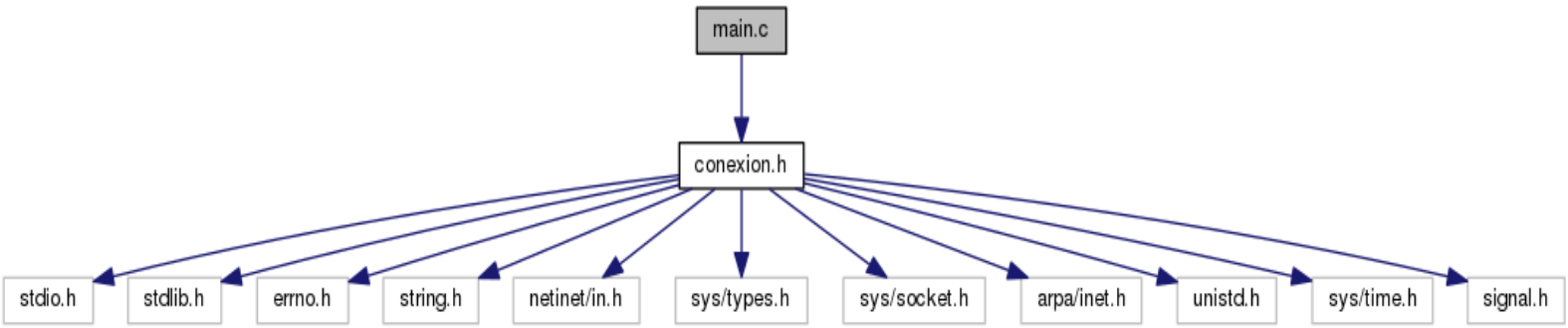
datos ingresados guardados en una lista, que los campos estén completos y luego se guardan usando archivos.
Luego puede ingresar usando la pantalla de logueo, que valida que los campos user y password tengan información, que el usuario exista abriendo el archivo para buscar el hombre de usuario (que coincida con alguno guardado en un nodo) y cuando lo encuentre compara la contraseña, validando así la existencia del usuario y coincidencia de la contraseña.
Si los datos son correctos se accede y aparece la opción de jugar una nueva partida o de ver los resultados de partidas recientes (si es que jugó alguna vez).
Al elegir nuevo juego, se visualiza una mesa con 3 cartas propias boca arriba y las 3 del oponente boca abajo, que se generan aleatoriamente y se guardan en un vector, junto a botones para hacer las distintas jugadas del truco, tales como el envido, flor, truco, vale cuatro, etc. Los botones se habilitan dependiendo de si en ese momento se puede hacer una jugada y cual. Se hace clic en alguno de ellos y luego en la carta a jugar para esa movida, aparece un cartel de esperar la movida del oponente.
Al finalizar la partida y durante la misma, aparece un botón para salir del juego, que muestra la puntuación que luego se guarda usando archivos.
Se crea un servidor y cliente con el uso de procesos.

Manejo de sockets:

Cliente:

El cliente será cualquier usuario que, teniendo los archivos específicos, podrá interactuar con el servidor solicitando realizar alguna de las opciones posibles y esperando una respuesta del server. La comunicación es half-duplex, ya que recién luego de recibir al atención solicitada a través del server, el cliente podrá realizar otra consulta.
Los archivos utilizados para el cliente son:

main.c:



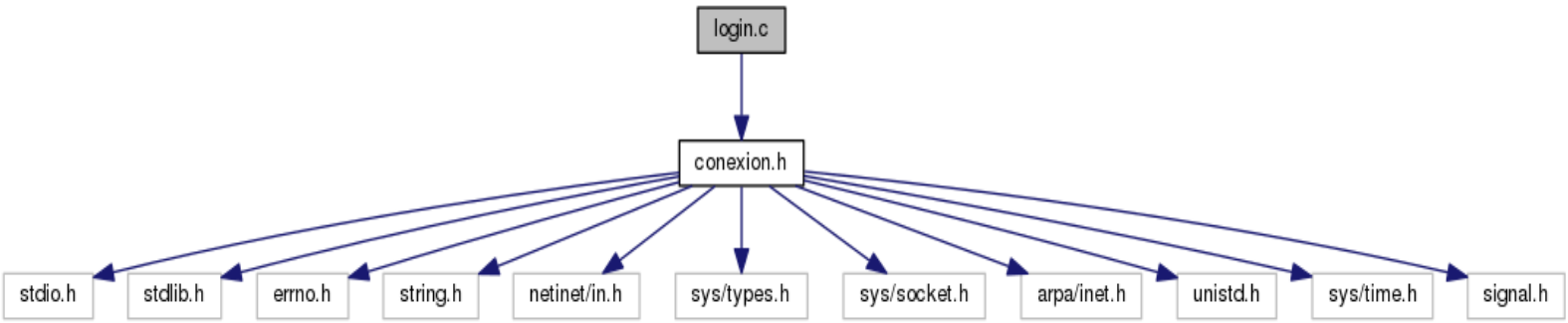
Archivo que contiene la función main, que es la función principal del cliente, la cual se encarga de verificar que el usuario haya presionado una tecla válida para realizar una solicitud al servidor (previamente para lograr la comunicación se debió haber abierto un socket; se explicará). Las opciones posibles son:

Opción	Tecla	Descripción
Registro al juego	R	Al querer registrarse, el servidor pedirá un nombre de usuario de hasta 32 caracteres y una contraseña de 8 a 16 caracteres. Opción válida solamente si el usuario no inició sesión
Inicio de sesión	I	Al querer iniciar sesión, el usuario debe ingresar su nombre y contraseña. Opción válida solamente si el usuario no inició sesión

Ver puntuaciones	P	El usuario puede consultar las 10 puntuaciones más altas y su puntuación (previamente debió iniciar sesión). Opción válida cuando el usuario no está ni en el log ni viendo puntuaciones
No ver más las puntuaciones	N	El usuario puede dejar de ver la tabla de posiciones. Opción válida solamente si el cliente está viendo las puntuaciones
Jugar a TrucoC	T	Al elegir esta opción el server le pide elegir modo de juego (S para jugar con flor y N para jugar sin flor) y cantidad de jugadores por equipo (de 1 a 3). Puede cancelar oprimiendo el 0 (cero). Al ingresar ambas opciones, se lo incorpora a un log. Al haber la suficiente cantidad de jugadores en el log, se inicia el juego. Opción válida cuando el jugador inició sesión
Salir del log	D	El cliente puede desconectarse se su log en caso que haya esperado un tiempo prolongado sin jugar. Opción válida cuando el jugador está en un log
Cerrar sesión y Salir	S	El cliente (habiendo o no iniciado sesión) puede oprimir esa tecla para cerrar el programa y desconectarse. Opción válida siempre

Luego de verificar que la opción sea válida (se trabaja con una variable opvalida que toma valores SI definida como 1 y NO definida como 0). En caso de serlo, se envía dicha opción al servidor el cual responderá; caso contrario, el cliente deberá presionar otra tecla. Esto se repetirá hasta que el cliente oprima la S para cerrar sesión y desconectarse,momento en el que se termina el programa del cliente.

login.c:



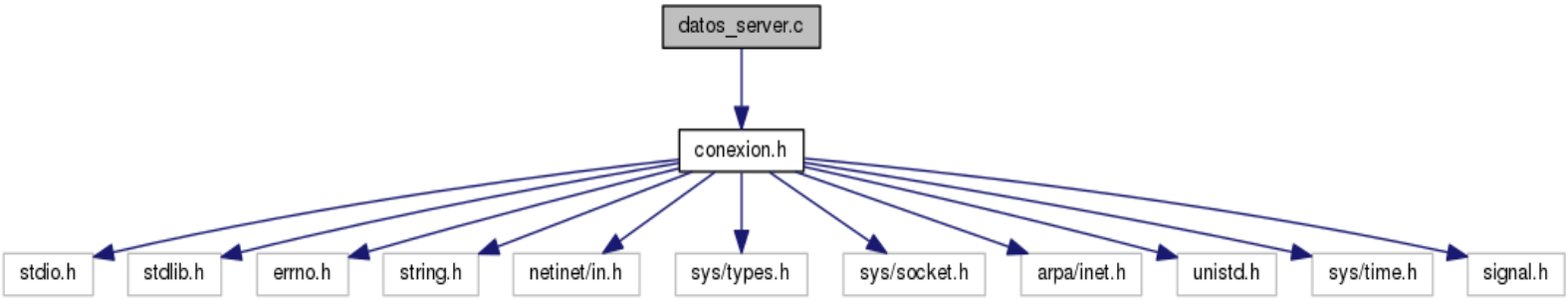
Archivo que contiene la función login, la cual es la primer función llamada por main y es la encargada de abrir el socket para lograr la comunicación con el servidor. Esta función a su vez llama a socket (su prototipo se encuentra en sys/socket.h) para abrir un socket del tipo SOCK_STREAM (su define está en arpa/inet.h) y protocolo TCP, a datos_server (se explicará más adelante) y a connect para iniciar la comunicación con el servidor.

Prototipo:
int login (void)

Parámetros
[out] sockfd_cliente Socket del cliente que se conecta (toma valor -1 en caso de error)

Devuelve
Un número entero

datos_server.c:



Archivo que contiene la función datos_server, llamada por login para obtener la dirección IP y el puerto del servidor.
Esto se logra leyendo un archivo de texto (config.txt) en el cual se encuentran esos datos.

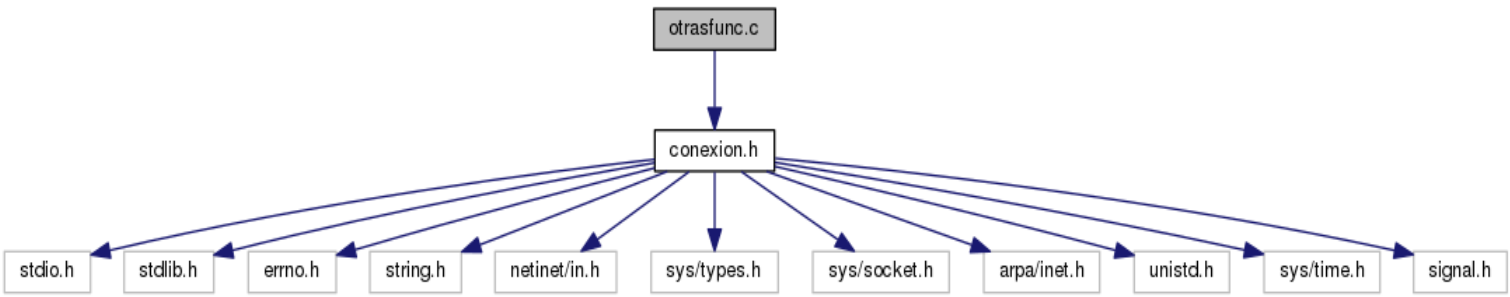
Prototipo:
void datos_server (struct datos * configuracion)

Función que carga del archivo config.txt los datos de ip y puerto del servidor.

Parámetros
[in] configuracion puntero a los datos del servidor.

La estructura datos contiene dos campos: un vector de chars donde se guarda la dirección IP y un entero donde se guarda el puerto.

otrasfunc.c



Archivo que contiene dos funciones de manejo de strings.

Una es myStrncmp, la cual realiza la comparación de los primeros n caracteres de dos cadenas.

Prototipo:
int myStrncmp (const char * s1, const char * s2, int n)

Función que recibe dos palabras y compara los primeros n caracteres.

Parámetros
[in] *s1 Palabra a comparar
[in] *s2 Palabra a comparar
[in] n Cantidad de caracteres a comparar (en caso que sea mayor a la longitud de la palabra s1, se compara esa cantidad de caracteres en vez de n)

Devuelve
Diferencia de códigos ASCII del primer caracter distinto entre s1 y s2, o la diferencia entre los ASCII del último caracter a comparar de cada uno como número entero

Resultados posibles
>0 –> s1 está antes de s2 en el alfabeto
==0 –> s1 y s2 están en el mismo orden alfabético
<0 --> s2 está antes de s1 en el alfabeto

La otra es my_strlen que calcula la longitud de una cadena de caracteres.

Prototipo:
int my_strlen (char * palabra)

Función que calcula la longitud de una palabra.

Parámetros
[in] *palabra Palabra cuya longitud quiere determinarse
[out] i Longitud de la palabra

Devuelve
Un número entero

Ambas son llamadas por main. El propósito es que el cliente interprete la respuesta recibida desde el server y a partir de allí realizar una acción. A modo de ejemplo se muestra esta parte de código del main:

```
case 'R':
    printf("Respuesta del servidor: %s\r\n",respuesta);
    while(myStrncmp(respuesta,"Ingrese su contraseña",my_strlen("Ingrese su contraseña")) && myStrncmp(respuesta,"Nombre de usuario existente. 5",my_strlen("Nombre de usuario existente. 5")) &&
    myStrncmp(respuesta,"Nombre muy largo. 5",my_strlen("Nombre muy largo. 5")))
    {
        fflush(stdin);
        scanf("%s",us.user);
        send(sockfd_cliente, us.user, strlen(us.user)+1, 0);
        nBytes = recv(sockfd_cliente,respuesta,MAX_REP,0);
        if(nBytes<=0){
            puts("Error en recv\r\n");
            close(sockfd_cliente);
            return(-1);
        }
        printf("Respuesta del servidor: %s\r\n",respuesta);
    }
    if(myStrncmp(respuesta,"Ingrese su contraseña",my_strlen("Ingrese su contraseña")))
    {
        break;
    }
```

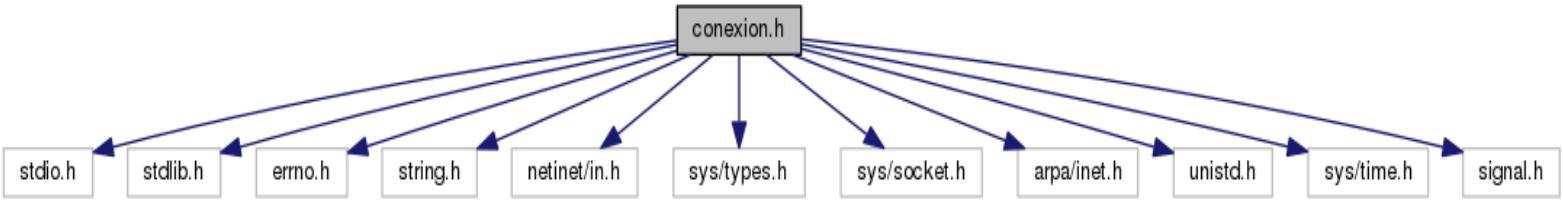
En el caso del registro, el while mostrado realiza una comparación de la cadena que empieza en la dirección guardada en respuesta (inicio de la cadena de la respuesta recibida del servidor) con 3 cadenas posibles:

- “Ingrese su contraseña”: Si la respuesta empieza con esta cadena, significa que el usuario ingresado es válido para el registro.
- “Nombre de usuario existente. 5”: Si la respuesta empieza así, significa que el quinto y último intento de escribir un nombre de usuario válido fue fallido ya que en el último intento se ingresó un nombre que ya existe en el archivo de registro del servidor.
- “Nombre muy largo. 5”: Si la respuesta empieza así, significa que el quinto y último intento de escribir un nombre de usuario válido fue fallido ya que en el último intento se ingresó un nombre de más de 32 caracteres.

El if siguiente verifica que la respuesta comience con “Ingrese su contraseña”. En dicho caso, se proseguirá con la contraseña, si no, se interrumpe el registro debido a la línea break; al ingresar por verdadero (se ingresa por verdadero si la myStrncmp devuelve distinto de 0).

conexion.h:

Archivo de cabecera que incluye los siguientes archivos de cabecera:



Por otra parte, allí están definidas varias constantes simbólicas y las estructuras:

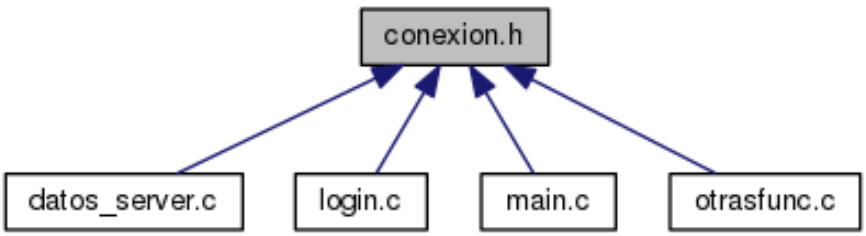
struct datos:
Campos de datos:
char ip [MAX_IP] (MAX_IP está definido como 14)
int puerto

Datos del servidor: Dirección IP (vector de chars) y Puerto (entero)

struct usuario:
Campos de datos:
char user [MAX_USER+1] (MAX_USER está definido como 40)
char password [MAX_PASS+1] (MAX_PASS está definido como 24)

Datos del usuario: Nombre de usuario y contraseña (ambos son vectores de chars)

Los prototipos de las funciones utilizadas y enumeradas al mencionar los demás archivos del cliente están en conexion.h



Otros archivos:

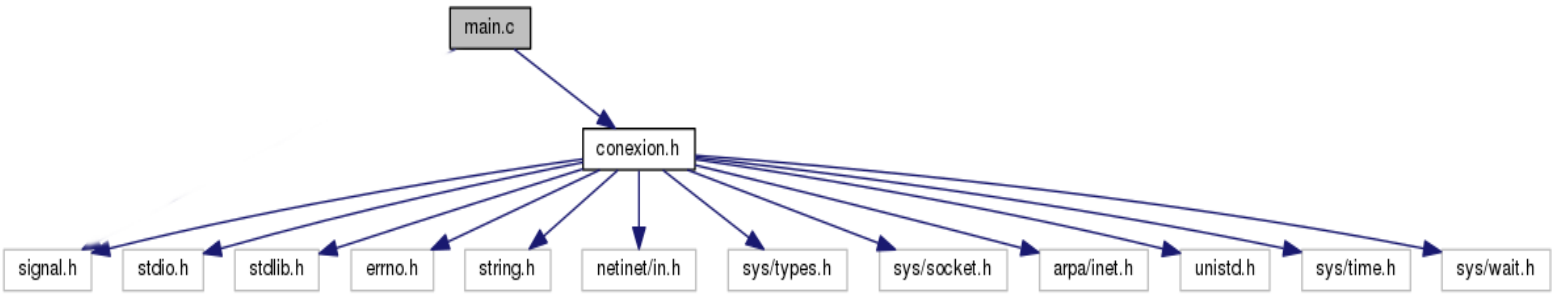
Previamente se mencionó el archivo de texto config.txt el cual se lee en la función datos_server para obtener IP y puerto del servidor. Otro archivo incorporado es el Makefile que funciona de la siguiente manera:

- Compilar con make new o make client
- Ejecutar o compilar y ejecutar en un paso con make run o simplemente make (luego de la ejecución se borrarán los archivos objeto y el ejecutable)
- Generar y abrir página web con documento Doxygen con make html
- Borrar web con make no_html
- Agregar archivos al repositorio con make git

Servidor:

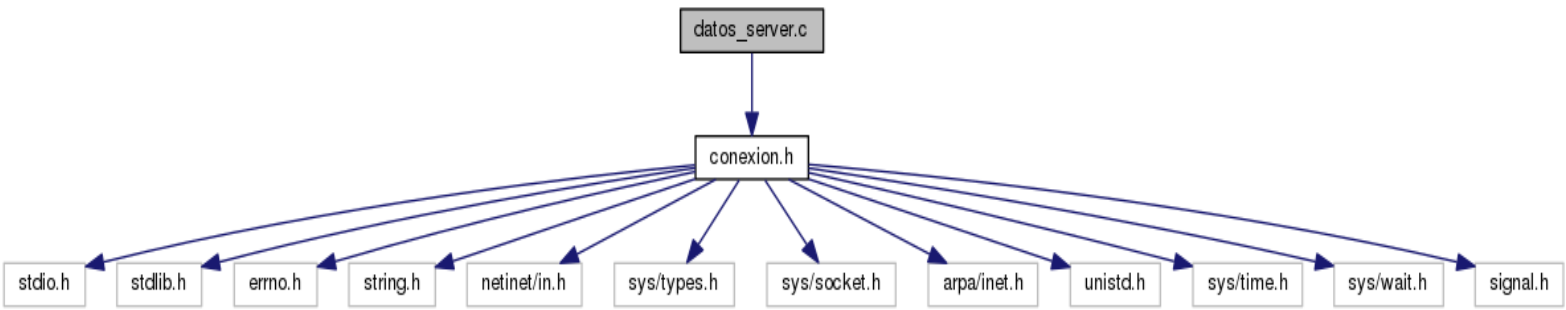
El servidor se encarga de esperar la conexión de un cliente, atenderlo luego de aceptar una conexión y realizar el manejo de archivos y listas, agregando y eliminando registros de usuarios en distintos archivos binarios. A continuación se enumeran los archivos utilizados:

main.c:



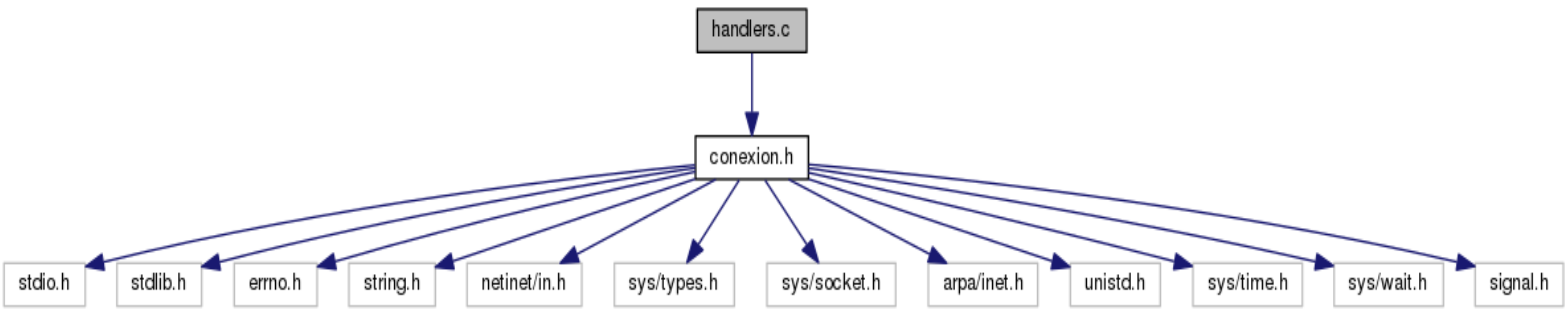
Archivo que contiene la función main. Dicha función llama a la función datos_server (se detallará más adelante) para obtener IP y puerto a reservar para el servidor, luego a socket para abrir el socket encargado de esperar la conexión de un cliente (a este socket se lo denomina en el código como listener) y a bind para reservar el puerto.
Luego, main llama a la función bloqueante accept de la cual se sale al llegar la conexión de un cliente (accept devuelve un nuevo socket que se guarda en el código en la variable cliente). En ese momento se crea un proceso hijo (se detallará en el item Manejo de procesos), que atenderá al cliente, mientras que el padre vuelve a esperar a otro cliente.

datos_server.c:



Archivo que contiene la función datos_server. La misma es idéntica a la función homónima utilizada en la parte del cliente.

handlers.c:



Archivo que contiene dos funciones con las acciones a realizar ante la llegada de una interrupción.

Una de ellas se denomina siginthandler que trapea la señal de interrupción SIGINT. En caso de llegar dicha interrupción, la misma se encarga de cerrar los sockets listener y cliente, así como algún archivo que se haya abierto o liberar recursos pedidos por memoria dinámica a través de malloc (en ambos casos se verifica que el file pointer y el puntero al inicio de la lista, respectivamente, no sean igual a NULL; de serlo, no hace falta realizar el cierre del archivo o la liberación de los recursos según sea el caso).

Prototipo:
void siginthandler (int signal)

Función (handler) que se ejecuta al llegar la señal de interrupción.

Parámetros
[in] signal Número de la señal de interrupción (SIGINT)

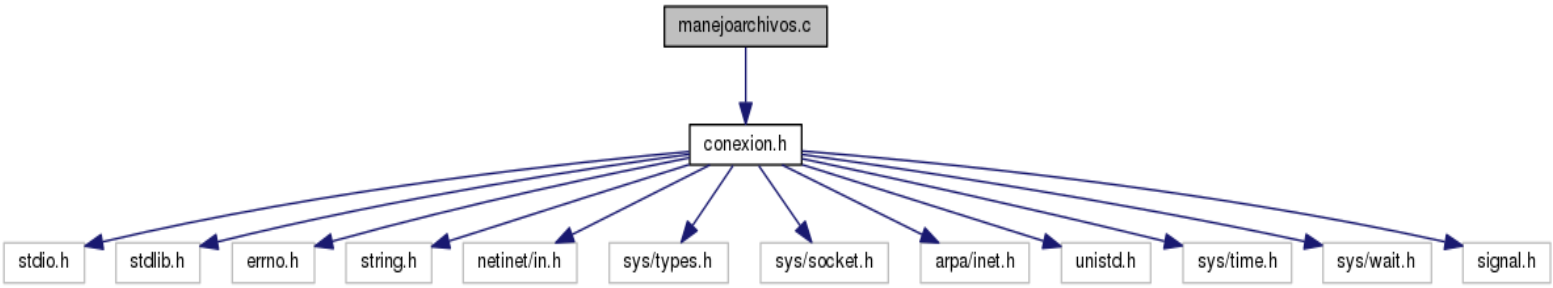
El otro handler que se encuentra en este archivo es sichldhandler. La misma se ejecuta al llegar la señal de muerte de un proceso hijo (SIGCHLD), y se encargar de cerrar el socket cliente y libera los recursos del proceso hijo.

Prototipo:
void sigchldhandler (int signal)

Función (handler) que se ejecuta al llegar la señal de muerte de proceso hijo.

Parámetros
[in] signal Número de la señal de muerte de proceso hijo (SIGCHLD)

manejoarchivos.c:



Archivo con las siguientes funciones:

int comprobarnombre (char *nom)
Función que comprueba el nombre de usuario entre todos los registrados. Para obtener los usuarios registrados para realizar esta comprobación se carga el archivo binario usuariosregistrados
Parámetros
[in] *nom Nombre del usuario

Devuelve
Un número entero

Valores posibles:

0 en caso de no encontrar el nombre de usuario en el archivo (es el resultado esperado si un cliente quiere registrarse)
ERR_NOM (definido como -1 en el archivo de cabecera conexion.h) si se encontró ese nombre en el registro (es el resultado esperado si un cliente quiere iniciar sesión)
Mayor a 0 en caso de error

int comprobarpass (DATO *usu)
Función que comprueba la contraseña del usuario. Para ello carga el archivo usuariosregistrados en el cual cada registro contiene la contraseña de cada usuario
Parámetros
[in] *usu Puntero a DATO que guarda la dirección de comienzo de los datos del usuario
[out] res Resultado de la comprobación

Devuelve
Un número entero

Valores posibles:

PASS_OK (definido como 10 en el archivo de cabecera conexion.h) si la contraseña es correcta
ERR_PASS (definido como -1 en el archivo de cabecera conexion.h) si la contraseña es incorrecta

int validarpass (char *pass)

Función que verifica que la contraseña ingresada sea válida (debe tener mayúsculas, minúsculas y números y debe ser de entre 8 y 16 caracteres)

Parámetros

[in] *pass Contraseña a verificar

Devuelve

Un número entero

Valores posibles:

0 en caso de exito

Mayor a 0 en caso de error (contraseña no válida)

int registrar (DATO *usu)

Función que registra a un nuevo usuario al archivo. Para ello abre el archivo binario usuariosregistrados en modo a+ para incorporar los datos al final de dicho archivo.

Parámetros

[in] *usu Puntero a DATO que guarda la dirección de comienzo de los datos del usuario

Devuelve

0 en caso de éxito, distinto de 0 en caso de error

int iniciarsesion (DATO *usu)

Función que se encarga de realizar y registrar el inicio de sesión de un usuario. Para ello abre el archivo binario ipensesion en modo a+ para incorporar los datos al final de dicho archivo.

Parámetros

[in] *usu Puntero a DATO que guarda la dirección de comienzo de los datos del usuario

Devuelve

0 en caso de éxito, distinto de 0 en caso de error

int ingresolog (DATO *usu, char opflor, int opjug)

Función que se encarga de realizar y registrar el modo de juego elegido por el usuario y registrarlo en el log correspondiente. Para ello abre uno de los archivos binarios de log en modo a+ para incorporar los datos al final de dicho archivo.

Parámetros

[in] *usu Puntero a DATO que guarda la dirección de comienzo de los datos del usuario

[in] opflor Modo de juego elegido (S para jugar con flor, N para jugar sin flor)

[in] opjug Cantidad de jugadores por equipo elegida (de 1 a 3)

Devuelve

0 en caso de éxito, distinto de 0 en caso de error

Ejemplo: Si opflor es igual a S y opjug es igual a 2, el usuario ingresará al log trucoS2 en el que se encuentran aquellos que quieren jugar con flor en un partido de 2 contra 2

void salidalog (DATO *usu)

Función que se encarga de realizar la salida del log del usuario. Para ello se cargan todos los archivos de log en modo lectura de a uno por vez y se borra el registro del usuario correspondiente (un usuario debe estar logueado en un solo log). Luego se lo abre en modo escritura para volver a guardar los nodos no borrados.

Parámetros

[in] *usu Puntero a DATO que guarda la dirección de comienzo de los datos del usuario

void cerrarsesion (DATO *usu)

Función que se encarga de realizar el cierre de sesión del usuario. Para ello se carga el archivo ipensesion en modo lectura para borrar el nodo del usuario correspondiente y volver a guardar aquellos no borrados en ducho archivo (previamente se debió abrir en modo escritura).

Parámetros

[in] *usu Puntero a DATO que guarda la dirección de comienzo de los datos del usuario

int puntuaciones (DATO *usu, char *respuesta)

Función que se encarga de preparar la tabla de posiciones para informarla al usuario. Carga el archivo usuariosregistrados, lo ordena por el campo puntos de mayor a menor y muestra los primeros 10 y la puntuación del usuario que inicio sesión.

Parámetros

[in] *usu Puntero a DATO que guarda la dirección de comienzo de los datos del usuario
[in] *respuesta Dirección de comienzo del buffer en el que se guardará la respuesta

Devuelve

0 en caso de éxito, distinto de 0 en caso de error

void mydectostr (int num, char *buf)

Función que convierte un entero en una cadena de caracteres. Es llamada por puntuaciones para convertir los puntos acumulados por un usuario a string para que dicha cadena sea concatenada a la respuesta.

Parámetros

[in] num Número a convertir
[in] *buf Dirección de comienzo del buffer en el que se guardará la conversión

void stringReverse (char *palabra)

Función que recibe la palabra y la invierte. Llamada en mydectostr. Es necesaria ya que, debido a como se arma la cadena en dicha función, si no se la invierte se muestra el número al revés.

Parámetros

[in] *palabra Palabra a invertir

void OrdenarLista (NODO *h)

Ordena todos los nodos de una lista.

Parámetros

[in] *h Recibe dirección del primer nodo de la lista

int CargarArchivo (char *ruta, NODO **h, NODO **last)

Función que lee un archivo y forma una lista.

Parámetros

[in] *ruta Nombre del archivo donde se guardan los datos
[in] **h Inicio de la lista
[in] **last Fin de la lista

Devuelve

0 en caso de éxito, distinto de 0 en caso de error

int GuardarLista (NODO **p, char *ruta)

Guarda todos los nodos de la lista a un archivo.

Parámetros

[in] **p Recibe dirección del primer nodo de la lista
[in] *ruta Nombre del archivo donde se guardan los datos

void BuscarAgregarNodoAlFinal (NODO **h, NODO *aux, NODO **last)

Función que busca un nodo y lo agrega si no lo encuentra.

Parámetros

[in] **h Dirección de inicio de la lista
[in] *aux Dirección del nodo a agregar/buscar
[in] **last Dirección de fin de la lista

int BuscarNodo (NODO **h, NODO *aux, NODO **last, int instruccion)

Función que busca un nodo y lo agrega si no lo encuentra.

Parámetros

[in] **h Dirección de inicio de la lista
[in] *aux Dirección del nodo a buscar

[in] **last Dirección de fin de la lista
[in] instruccion Acción a realizar al encontrar el nodo (ej.: BUSCARPASS, definido como 4 en conexión.h, para verificar contraseña)

void AgregarNodoAlFinal (NODO **h, NODO *aux, NODO **last)
Función que busca un nodo y lo agrega si no lo encuentra. Llamada por BuscarAgregarNodoAlFinal si no se encontró el nodo.
Parámetros
[in] **h Dirección de inicio de la lista
[in] *aux Dirección del nodo a agregar/buscar
[in] **last Dirección de fin de la lista

void BorrarLista (NODO **p)
Función que borra toda una lista.
Parámetros
[in] **p Dirección de inicio de la lista

int my_strlen (char *palabra)
Idéntica a aquella explicada en el archivo otrasfunc.c de la parte del cliente.

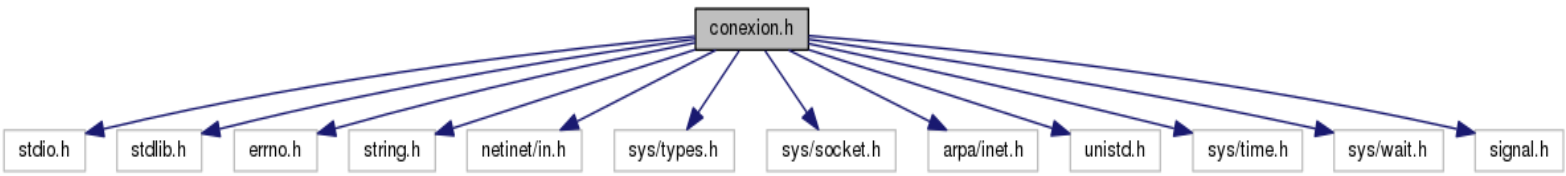
int myStrncpy (char *dest, char *origen, int n)
Función que copia los primeros n caracteres de una palabra a otro string.
Parámetros
[in] *dest Dirección del string al que se envía la palabra
[in] *origen Dirección del string del que proviene la palabra
[in] n Cantidad de caracteres a copiar
[out] -1 En caso de error en la copia
[out] 0 En caso de copia realizada correctamente

Devuelve
Devuelve número entero

int myStrncmp (const char *s1, const char *s2, int n)
Idéntica a aquella explicada en el archivo otrasfunc.c de la parte del cliente.

conexion.h:

Archivo de cabecera que incluye los siguientes archivos de cabecera:



Por otra parte, allí están definidas varias constantes simbólicas y las estructuras:

struct datos:
Campos de datos:
char ip [MAX_IP] (MAX_IP está definido como 14)
int puerto

Datos del servidor: Dirección IP (vector de chars) y Puerto (entero)

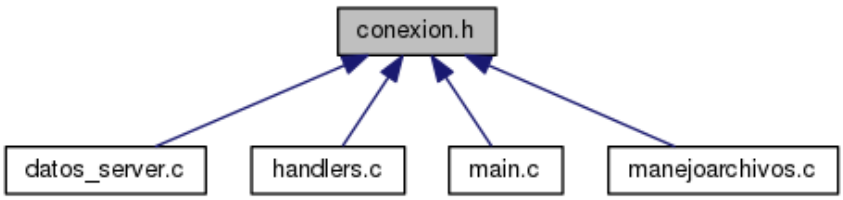
struct usuario (estructura definida con el alias DATO):
Campos de datos:
char user [MAX_USER+1] (MAX_USER está definido como 40)
char password [MAX_PASS+1] (MAX_PASS está definido como 24)
int puntos

Datos del usuario: Nombre de usuario, contraseña (ambos son vectores de chars) y puntos acumulados (entero)

struct nodo (estructura definida con el alias NODO):
Campos de datos:
DATO dato
struct nodo * sgte

Creación del nodo para ser utilizado en listas: Dato (Estructura usuario) y dirección del siguiente nodo de la lista (Puntero a la estructura nodo)

Los prototipos de las funciones utilizadas y enumeradas al mencionar los demás archivos del servidor están en conexion.h



Otros archivos:

Otros archivos con los que interactúa el servidor son:

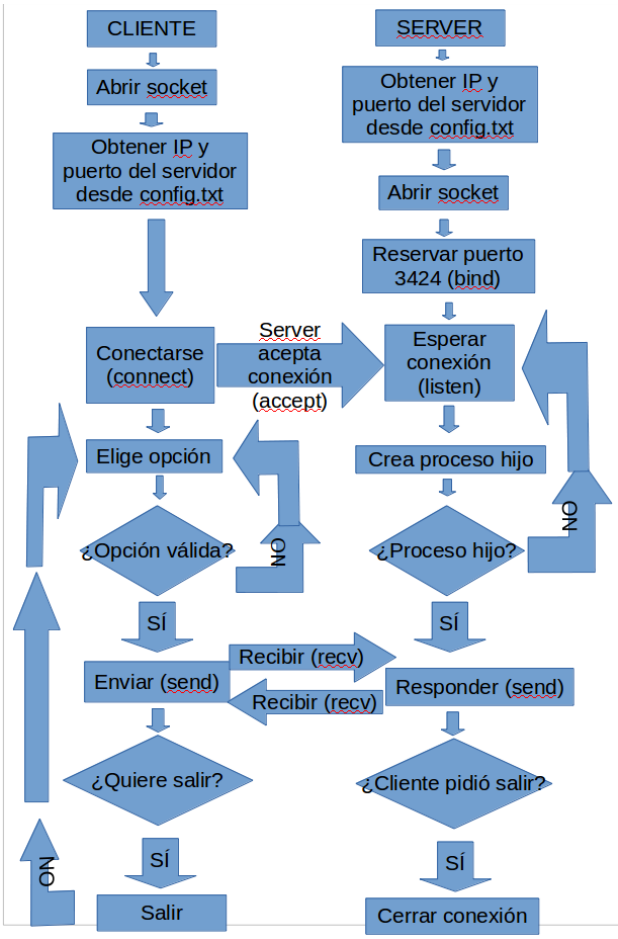
- config.txt: Contiene los datos del servidor (IP:127.0.0.1; Puerto: 3424).
- usuariosregistrados: Archivo binario que contiene los datos (nombre, contraseña y puntos) de los usuarios registrados.
- ipensesion: Archivo binario que guarda los usuarios en sesión.
- Archivos de log: Contiene a los usuarios que uieren jugar. Los archivos son:
  - trucoS1: Con flor, 1 contra 1
  - trucoN1: Sin flor, 1 contra 1
  - trucoS2: Con flor, 2 contra 2
  - trucoN2: Sin flor, 2 contra 2
  - trucoS3: Con flor, 3 contra 3
  - trucoN3: Sin flor, 3 contra 3

Por otra parte existe un Makefile para realizar las siguientes instrucciones:

- Compilar con make new o make server
- Ejecutar o compilar y ejecutar en un paso con make run o simplemente make (luego de la ejecución se borrarán los archivos objeto y el ejecutable)
- Generar y abrir página web con documento Doxygen con make html
- Borrar web con make no_html
- Agregar archivos al repositorio con make git

Diagrama de bloques:

En el siguiente esquema puede observarse la comunicación entre el cliente y el servidor.

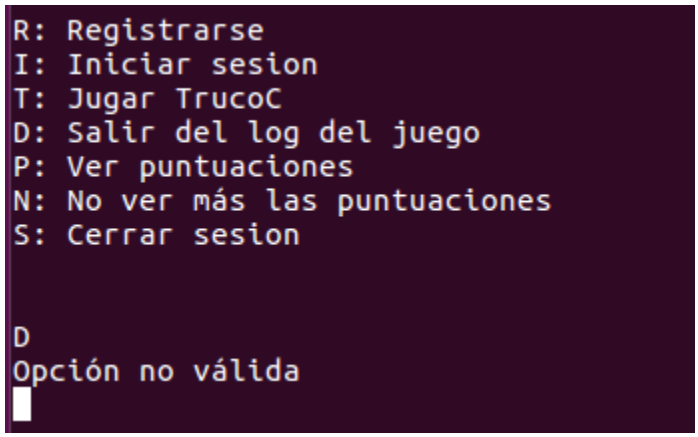


Capturas:

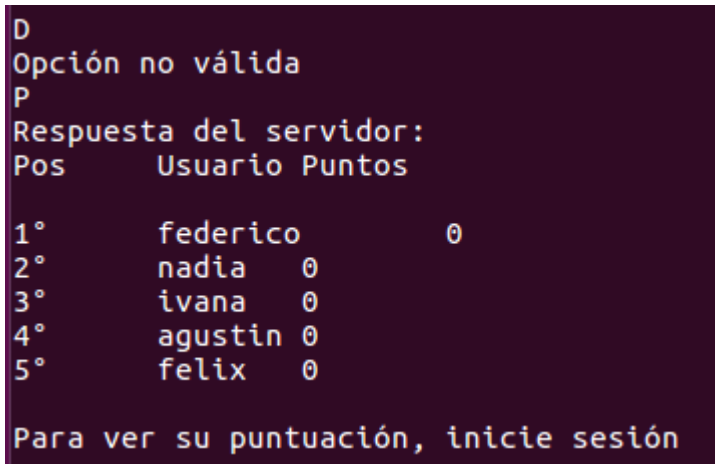
Las siguientes imágenes muestra la ejecución del cliente, detallando la respuesta que recibe de acuerdo a cada solicitud.

```
federico@federico-MS-7641:~/Escritorio/gittruco/truco/cliente$ ./client
Datos del server
ip:"127.0.0.1"
puerto:"3424"
Cliente conectado exitosamente
Respuesta del servidor: Conexion aceptada

R: Registrarse
I: Iniciar sesion
T: Jugar TrucoC
D: Salir del log del juego
P: Ver puntuaciones
N: No ver más las puntuaciones
S: Cerrar sesion
```

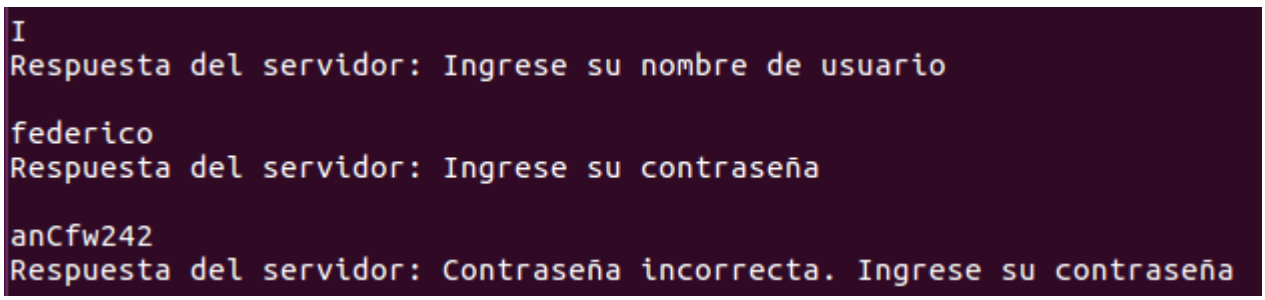
Primera respuesta del servidor que recibe el cliente al conectarse. Allí se ven las opciones que puede elegir.



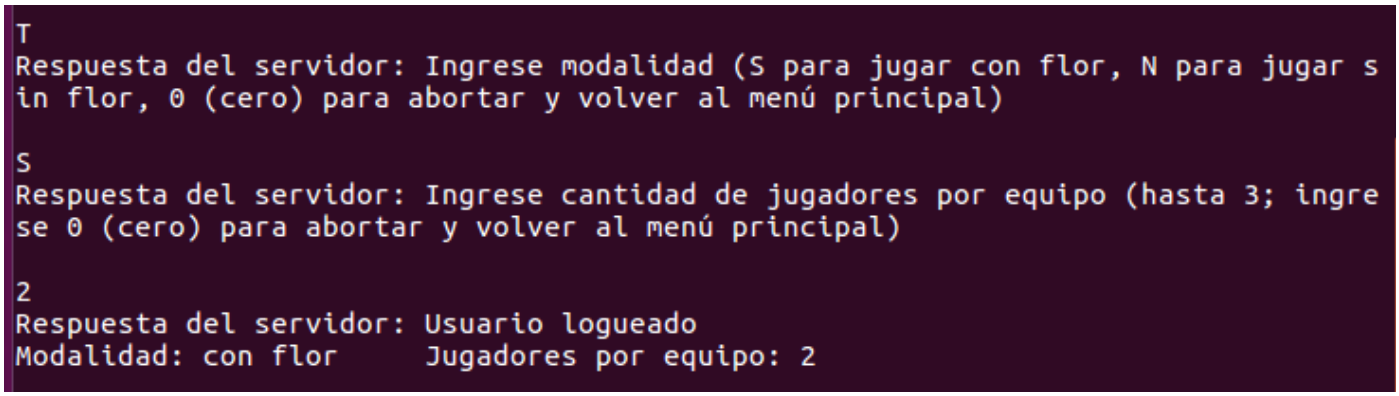
El cliente no puede desconectarse de un log si no inició sesión. La respuesta "Opción no válida" se imprime desde el mismo código del cliente encargado de realizar esta validación.



Respuesta que recibe un cliente que no inició sesión que quiere ver la tabla de posiciones.



Cliente quiere iniciar sesión. Luego ingresa un nombre de usuario, el servidor encuentra ese nombre en el archivo usuariosregistrados y le pide ingresar contraseña. El usuario luego ingresa una contraseña incorrecta. El servidor se da cuenta de ello al abrir nuevamente usuariosregistrados como sólo lectura.

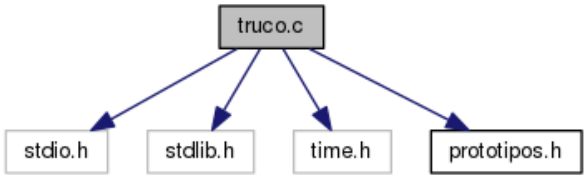


Luego de iniciar sesión correctamente, el usuario pide jugar al truco. El servidor solicita que elija modalidad y cantidad de jugadores por equipo.

El juego propiamente dicho:

TrucoC es un programa que permite a cualquier persona que tenga los códigos necesarios juegar al truco en su computadora. Puede elegir jugar con o sin flor, así como escoger la cantidad de jugadores que desea que haya en cada equipo (pueden ser 1, 2 ó 3; en ese último caso, se jugará bajo la modalidad pica-pica).
Para poder acceder al juego, el cliente primero debe ingresar al log correspondiente según modo de juego y cantidad de jugadores por equipo (en el apartado Cliente del ítem Manejo de sockets se explica como debe proceder el usuario). En el momento que el servidor abra un archivo de log y verifique que hay la cantidad suficiente de jugadores, se inicia el juego.
El objetivo para ganar es que un equipo (ya sea de 1 ,2 ó 3 personas) sume 30 puntos para ganar. Luego de finalizado el juego, cada jugador sumará 5 puntos en la tabla de posiciones, mientras que los integrantes del equipo ganador obtendrán otros 20 cada uno.
A modo de prueba, la jugabilidad se codificó por separado al servidor y al cliente. En el ítem Dificultades se detallará por qué estas partes no están unidas.
Las reglas de juego de TrucoC se basan en libro “El truco” de Juan Carlos Ortega.
Los archivos utilizados para el funcionamiento de esta parte del programa son:

truco.c:



Contiene la función main, en la que se llaman funciones que preparan el juego y funciones que moderan la partida.
En este caso la función main tiene argumentos de recepción.

Prototipo:
int main (int argc, char ** argv)

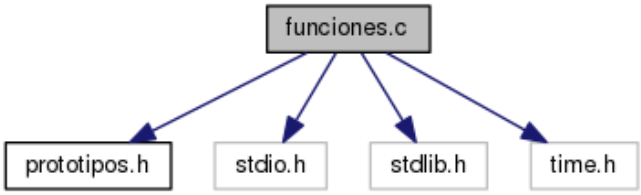
Función principal del juego.

Se inicia el juego si el segundo argumento es S o N y el tercero es 1, 2 ó 3
Luego de iniciado, por cada mano se generan aleatoriamente las cartas por jugador y comienza el juego desde el jugador que es mano
Si juegan 3 por equipo, habrá pica-pica a partir que un equipo consiga 5 puntos y hasta que un equipo consiga 25

Parámetros
[in] argc Cantidad de argumentos. Deben ser 3 para jugar: el nombre del programa (./truco), el modo de juego (S para jugar con flor o N para jugar sin flor) y la cantidad de jugadores por equipo (de 1 a 3)
[in] argv Valor de los argumentos

Si no se ingresan exactamente 3 argumentos (incluyendo el nombre del programa) o bien si los argumentos del modo de juego o la cantidad de jugadores toman valores que no corresponden, el programa se interrumpirá.
Si los argumentos son correctos, se prepara el juego, inicializando cartas, eligiendo aleatoriamente las cartas con las que se va a jugar, y luego se inicia la mano. Habrá juego hasta que uno de los equipos llegue a 30 puntos luego de terminar una mano. En caso que ambos lleguen a esa cantidad y queden empatados, se jugará otra mano hasta tener un ganador.

funciones.c:



Archivo con todas las funciones de preparación del juego como las de moderación. Ellas son las siguientes:

void configurarcartas (baraja_t(*cartas)[10])

Función que completa una matriz con códigos de cartas.

Parámetros

[in] **cartas Puntero a puntero a baraja_t que contiene la dirección de comienzo de la matriz de cartas

Esta función de preparación les asocia un código numérico a cada carta. Ese código permite a la PC interpretar la carta jugada. Con esta función se arma la matriz cartas:

	Carta	1	2	3	4	5	6	7	10	11	12
Palo	Fila\Col	0	1	2	3	4	5	6	7	8	9
Basto	0	81	82	83	84	85	86	87	90	91	92
Copa	1	121	122	123	124	125	126	127	130	131	132
Espada	2	161	162	163	164	165	166	167	170	171	172
Oro	3	201	202	203	204	205	206	207	210	211	212

void calcularvalorescartas (baraja_t(*cartas)[10], unsigned char(*valorcartaparaprimera)[10], unsigned char(*valorcartaparasegunda)[10])

Función que completa matrices con códigos de valores de las cartas para el envido y para el truco.

Parámetros

[in] **cartas Puntero a puntero a baraja_t que contiene la dirección de comienzo de la matriz de cartas

[in] **valorcaraparaprimera Puntero a puntero a unsigned int que contiene la dirección de comienzo de la matriz con el valor de cada carta para el envido o la flor

[in] **valorcaraparasegunda Puntero a puntero a unsigned int que contiene la dirección de comienzo de la matriz con el valor de cada carta para el truco

A partir de la matriz cartas se asignan valores a las matrices valorcartaparaprimera, que contiene el valor de cada carta para el envido y la flor, y valorcartaparasegunda, que contiene el valor de las cartas para el truco. Se empieza con el 0 para las de menor valor y se suma 1 progresivamente según la importancia de la carta para cada caso.

Matriz valorcartaparaprimera:

	Carta	1	2	3	4	5	6	7	10	11	12
Palo	Fila\Col	0	1	2	3	4	5	6	7	8	9
Basto	0	1	2	3	4	5	6	7	0	0	0
Copa	1	1	2	3	4	5	6	7	0	0	0
Espada	2	1	2	3	4	5	6	7	0	0	0
Oro	3	1	2	3	4	5	6	7	0	0	0

Matriz valorcartaparasegunda:

	Carta	1	2	3	4	5	6	7	10	11	12
Palo	Fila\Col	0	1	2	3	4	5	6	7	8	9
Basto	0	12	8	9	0	1	2	3	4	5	6
Copa	1	7	8	9	0	1	2	3	4	5	6
Espada	2	13	8	9	0	1	2	11	4	5	6
Oro	3	7	8	9	0	1	2	10	4	5	6

void repartircartas (baraja_t(*cartas)[10], baraja_t(*cartasrepartidas)[3], unsigned char mano, unsigned char jugadoresporequipo)

Función que reparte 3 cartas a cada jugador. La primera va a quien es mano.

Parámetros

[in] **cartas Puntero a puntero a baraja_t que contiene la dirección de comienzo de la matriz de cartas

[in] **cartasrepartidas Puntero a puntero a baraja_t que contiene la dirección de comienzo de la matriz con los códigos de las cartas repartidas para los jugadores

[in] mano Indica qué jugador es mano (ese jugador recibe la primer carta)

[in] jugadoresporequipo Indica cuántos jugadores hay por equipo

Se completa con valores aleatorios la matriz cartasrepartidas. Los equipos están conformados así:

Equipo 1	Equipo 2
Jugador 1	Jugador 2
Jugador 3	Jugador 4
Jugador 5	Jugador 6

Si el Jugador 2 es mano, la primera carta irá para él. Luego, si hay más de un jugador por equipo, la segunda irá al Jugador 3 y así hasta que cada jugador tenga 3 naipes.

void calculotanto (baraja_t(*cartas)[10], baraja_t(*cartasrepartidas)[3], unsigned char(*valorcartaparaprimera)[10], char *tantoenvido, char *tantoflor, unsigned char jugadoresporequipo, unsigned char hayflor)

Función que calcula los tantos que tiene cada jugador para el envido o la flor.

Parámetros

[in] **cartas Puntero a puntero a baraja_t que contiene la dirección de comienzo de la matriz de cartas

[in] **cartasrepartidas Puntero a puntero a baraja_t que contiene la dirección de comienzo de la matriz con los códigos de las cartas repartidas para los jugadores

[in] **valorcaraparaprimera Puntero a puntero a unsigned int que contiene la dirección de comienzo de la matriz con el valor de cada carta para el envido o la flor

[in] *tantoenvido Puntero a char que contiene la dirección de comienzo de un vector con los tantos de cada jugador para el envido (si un jugador tiene flor, habrá un -1 en su elemento correspondiente)

[in] *tantoflor Puntero a char que contiene la dirección de comienzo de un vector con los tantos de cada jugador para la flor (si un jugador no tiene flor, habrá un -1 en su elemento correspondiente)

[in] jugadoresporequipo Indica cuántos jugadores hay por equipo

[in] hayflor Indica si se juega con flor o no

Se completan las matrices tantoenvido y tantoflor. Si un jugador no tiene flor, se lo completa con -1. Si se juega sin flor, sólo se calcula el tanto para el envido.

La importancia de esto está en que cada jugador verá cada tanto y en caso de jugarse el envido o la flor, en caso de un mal canto (decir una cantidad distinta a la que se tiene) o negar una flor, su equipo sufrirá una penalización.

void inicializaracciones (unsigned char(*acciones)[6], unsigned char jugadoresporequipo, unsigned char hayflor)

Función que inicializa la matriz de acciones.

Parámetros

[in] **acciones Puntero a puntero a unsigned char que contiene la dirección de comienzo de la matriz de acciones posibles para cada jugador

[in] jugadoresporequipo Indica cuántos jugadores hay por equipo

[in] hayflor Indica si se juega con flor o no

Se completa la matriz acciones que se irá actualizando según como transcurra la mano. Cada jugador puede:

- Jugar carta
- Cantar envido
- Cantar real envido

- Cantar falta envido
- Cantar flor (si se juega con flor)
- Cantar contraflor
- Cantar contraflor al resto
- Cantar con flor me achico
- Cantar truco
- Cantar retruco
- Cantar vale cuatro
- Cantar quiero
- Cantar no quiero
- Decir tanto del envido o flor
- Enviar un mensaje a todos los jugadores
- Enviar un mensaje a los compañeros de equipo
- Irse al mazo
- No hacer nada y pasar turno

El juego informa al jugador que posee el turno cuáles de estas acciones puede escoger, asociándole un número a cada una. Si ingresa un número incorrecto, se informa que se eligió una opción no válida y el jugador ingresará otro número hasta que el mismo corresponda a algo que él pueda realizar.

void inicializarmano (char *tantoenvido, char *tantoflor, char *penalizacionflor, char *cantobienflor, char *cartasjugadastruco, char *jugcantoenvido, unsigned char *mazo, unsigned char(*ordencartasjugadas)[3], unsigned char jugadoresporequipo)

Función que inicializa la mano a jugar.

Parámetros

[in] *tantoenvido Puntero a char que contiene la dirección de comienzo de un vector con los tantos de cada jugador para el envido (si un jugador tiene flor, habrá un -1 en su elemento correspondiente)

[in] *tantoflor Puntero a char que contiene la dirección de comienzo de un vector con los tantos de cada jugador para la flor (si un jugador no tiene flor, habrá un -1 en su elemento correspondiente)

[in] *penalizacionflor Puntero a char que contiene la dirección de comienzo de un vector con los flags de penalización de cada jugador por flor mal cantada

[in] *cantobienflor Puntero a char que contiene la dirección de comienzo de un vector con los flags que indican si un jugador decidió bien en cantar o no una flor

[in] *cartasjugadastruco Puntero a char que contiene la dirección de comienzo de un vector con las cartas jugadas por cada jugador en una baza

[in] *jugcantoenvido Puntero a char que contiene la dirección de comienzo de un vector con los flags que indican si un jugador cantó en el envido

[in] *mazo Puntero a unsigned char que contiene la dirección de comienzo de un vector con los flags que indican si un jugador se fue al mazo

[in] **ordencartasjugadas Puntero a puntero unsigned char que contiene la dirección de comienzo de una matriz con los daatos de las cartas jugadas por cada jugador y en qué orden (la primer carta jugada estará en la posición 0)

[in] jugadoresporequipo Indica cuántos jugadores hay por equipo

Esta función inicializa varias matrices para lograr un correcto funcionamiento del programa.

void informarcartas (unsigned char mostrarcartas, unsigned char(*cartasrepartidas)[3], unsigned char hayflor, unsigned char mano, char *tantoenvido, char *tantoflor)

Función que informa las cartas y los tantos de un jugador.

Parámetros

[in] mostrarcartas Código del jugador cuyas cartas y tanto se van a informar

[in] **cartasrepartidas Puntero a puntero unsigned char que contiene la dirección de comienzo de una matriz con los daatos de las cartas repartidas a cada jugador

[in] hayflor Indica si se juega con flor o no

[in] mano Código del jugador que es mano

[in] *tantoenvido Puntero a char que contiene la dirección de comienzo de un vector con los tantos de cada jugador para el envido (si un jugador tiene flor, habrá un -1 en su elemento correspondiente)

[in] *tantoflor Puntero a char que contiene la dirección de comienzo de un vector con los tantos de cada jugador para la flor (si un jugador no tiene flor, habrá un -1 en su elemento correspondiente)

Informa por pantalla los naipes repartidos a cada jugador y el tanto para el envido y la flor.

void juegamano (baraja_t(*cartas)[10], baraja_t(*cartasrepartidas)[3], unsigned char(*acciones)[6], unsigned char mano, char *tantoenvido, char *tantoflor, char *penalizacionflor, char *cantobienflor, char *cartasjugadastruco, char *jugcantoenvido, unsigned char *mazo, unsigned char(*ordencartasjugadas)[3], unsigned char(*valorcartaparasegunda)[10], unsigned char jugadoresporequipo, unsigned char hayflor, unsigned char estilo, unsigned char *eqganaprimera, unsigned char *eqganasegunda, unsigned char *floresnegadaseq1, unsigned char *floresnegadaseq2, unsigned char *puntosprimera, unsigned char *puntossegunda, unsigned char puntoseq1, unsigned char puntoseq2)

Función que modera la mano a jugar.

Parámetros

[in] **cartas Puntero a puntero a baraja_t que contiene la dirección de comienzo de la matriz de cartas

[in] **cartasrepartidas Puntero a puntero a baraja_t que contiene la dirección de comienzo de la matriz con los códigos de las cartas repartidas para los jugadores

[in] **acciones Puntero a puntero a unsigned char que contiene la dirección de comienzo de la matriz de acciones posibles para cada jugador

[in] mano Código del jugador que es mano

[in] *tantoenvido Puntero a char que contiene la dirección de comienzo de un vector con los tantos de cada jugador para el envido (si un jugador tiene flor, habrá un -1 en su elemento correspondiente)

[in] *tantoflor Puntero a char que contiene la dirección de comienzo de un vector con los tantos de cada jugador para la flor (si un jugador no tiene flor, habrá un -1 en su elemento correspondiente)

[in] *penalizacionflor Puntero a char que contiene la dirección de comienzo de un vector con los flags de penalización de cada jugador por flor mal cantada

[in] *cantobienflor Puntero a char que contiene la dirección de comienzo de un vector con los flags que indican si un jugador decidió bien en cantar o no una flor

[in] *cartasjugadastruco Puntero a char que contiene la dirección de comienzo de un vector con las cartas jugadas por cada jugador en una baza

[in] *jugcantoenvido Puntero a char que contiene la dirección de comienzo de un vector con los flags que indican si un jugador cantó en el envido

[in] *mazo Puntero a unsigned char que contiene la dirección de comienzo de un vector con los flags que indican si un jugador se fue al mazo

[in] **ordencartasjugadas Puntero a puntero unsigned char que contiene la dirección de comienzo de una matriz con los daatos de las cartas jugadas por cada jugador y en qué orden (la primer carta jugada estará en la posición 0)

[in] jugadoresporequipo Indica cuántos jugadores hay por equipo

[in] hayflor Indica si se juega con flor o no

[in] estilo Indica si se juega una mano redonda o un pica-pica

[in] *eqganaprimera Puntero con el dato del equipo que ganó la primera

[in] *eqganasegunda Puntero con el dato del equipo que ganó la segunda

[in] *floresnegadaseq1 Puntero con el dato de las flores negadas o mal cantadas por el equipo 1

[in] *floresnegadaseq2 Puntero con el dato de las flores negadas o mal cantadas por el equipo 2

[in] *puntosprimera Puntero con el dato de los puntos obtenidos en la primera

[in] *puntossegunda Puntero con el dato de los puntos obtenidos en la segunda

[in] puntoseq1 Puntos obtenidos por el equipo 1

[in] puntoseq2 Puntos obtenidos por el equipo 2

Esta función administra los turnos, actualiza matrices y calcula quien ganó la primera y quién la segunda, así como los puntos obtenidos por cada equipo.

void informarmano (unsigned char eqganaprimera, unsigned char eqganasegunda, unsigned char floresnegadaseq1, unsigned char floresnegadaseq2, unsigned char puntosprimera, unsigned char puntossegunda, unsigned char *puntoseq1, unsigned char *puntoseq2)

Función que informa el resultado de la mano redonda jugada.

Parámetros

[in] eqganaprimera Equipo que ganó la primera

[in] eqganasegunda Equipo que ganó la segunda

[in] floresnegadaseq1 Flores negadas o mal cantadas por el equipo 1

[in] floresnegadaseq2 Flores negadas o mal cantadas por el equipo 2

[in] puntosprimera Puntos obtenidos en la primera
[in] puntossegunda Puntos obtenidos en la segunda
[in] *puntoseq1 Puntero con el dato de los puntos del equipo 1
[in] *puntoseq2 Puntero con el dato de los puntos del equipo 2

Se muestran los puntos obtenidos por cada equipo y el resultado parcial o final.

void informarpicapica (unsigned char puntoseq1picapica, unsigned char puntoseq2picapica, unsigned char *puntoseq1, unsigned char *puntoseq2)

Función que informa el resultado de la mano jugada en estilo pica-pica.

Parámetros

[in] puntoseq1picapica Puntos obtenidos por el equipo 1 en el pica-pica
[in] puntoseq2picapica Puntos obtenidos por el equipo 2 en el pica-pica
[in] *puntoseq1 Puntero con el dato de los puntos del equipo 1
[in] *puntoseq2 Puntero con el dato de los puntos del equipo 2

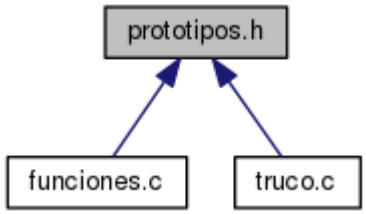
Luego de los 3 pica-pica de una mano, se informa cuantos puntos obtuvo el equipo ganador de la mano. Esos puntos se obtienen de la diferencia entre lo acumulado por los jugadores del equipo 1 y lo sumado por el equipo 2 en dicha mano.

int myStrlen (const char *palabra)

Función idéntica a my_strlen usada en las partes del servidor y del cliente.

prototipos.h:

Los prototipos de las funciones utilizadas y enumeradas al mencionar los demás archivos del cliente están en prototipos.h



El archivo posee constantes simbólicas y la definición de la variable baraja_t.

unsigned char baraja_t

Tipo de dato creado para la generación de códigos numéricos asociados a cada carta.

Otros archivos:

Se incluye un Makefile con las siguientes instrucciones:

- Compilar con make new o make truco
- Ejecutar o compilar y ejecutar en un paso con make run o simplemente make (luego de la ejecución se borrarán los archivos objeto y el ejecutable)
- Generar y abrir página web con documento Doxygen con make html
- Borrar web con make no_html
- Agregar archivos al repositorio con make git

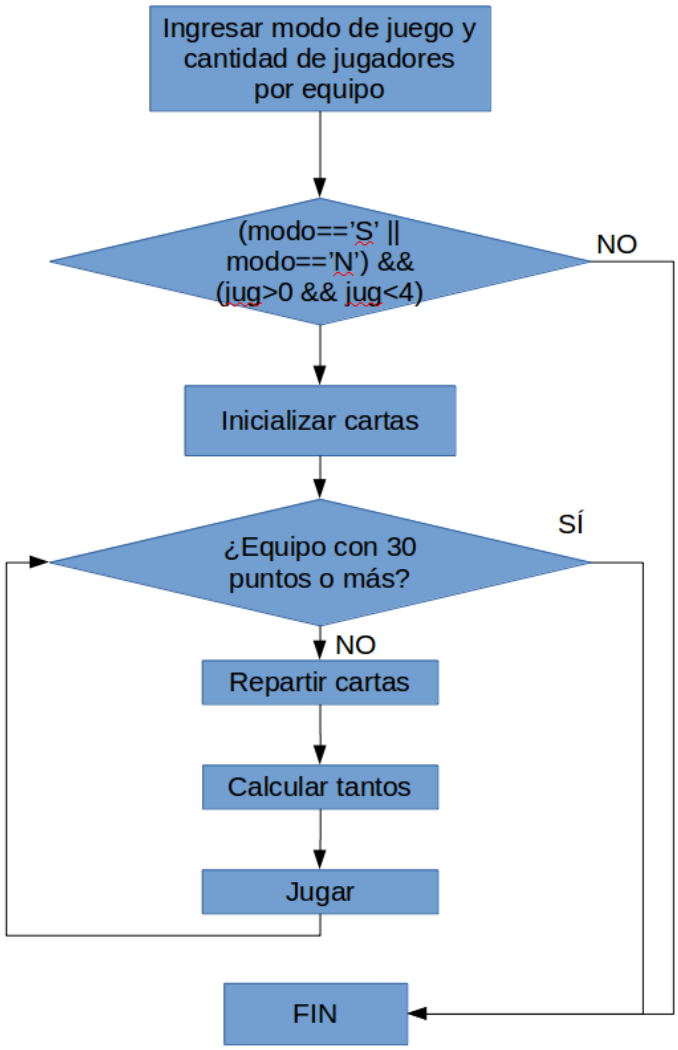
La regla run para crear el archivo ejecutable incluye una línea previa a realizar la ejecución. La misma es:

```
@while [ -z "$$mod" ]; do \
read -r -p "Ingrese modo de juego (presione S para jugar con flor o N para jugar sin flor): " mod; \
done ;  \
[ $$mod = "S" ] || [ $$mod = "N" ] || (echo "Opción incorrecta\nBorrando archivos objeto ..."; rm -f *.o; echo "Archivos objeto eliminados\n"; killall make;); \
while [ -z "$$jug" ]; do \
read -r -p "Ingrese cantidad de jugadores por equipo (hasta 3): " jug; \
done ; \
[ $$jug = "1" ] || [ $$jug = "2" ] || [ $$jug = "3" ] || (echo "Cantidad de jugadores incorrecta\nBorrando archivos objeto ..."; rm -f *.o; echo "Archivos objeto eliminados\n"; killall make;); \
echo "Puede jugar"; \
./$(ARCHIVO) $$mod $$jug
```

Al ejecutar esta regla, se le pedirá al usuario ingresar modo de juego (S o N). Si ingresa una tecla no válida, make se interrumpe. Caso contrario, deberá ingresar la cantidad de jugadores por equipo. En caso de elegir 1, 2 ó 3, se le permitirá jugar y se abre el programa enviando 3 argumentos por línea de comando (el nombre del programa, el modo de juego y la cantidad de jugadores). Si elige otro valor distinto, provocará la interrupción de make.

Diagrama de bloque:

A continuación se resume con el siguiente diagrama los pasos a seguir para jugar:



Capturas:

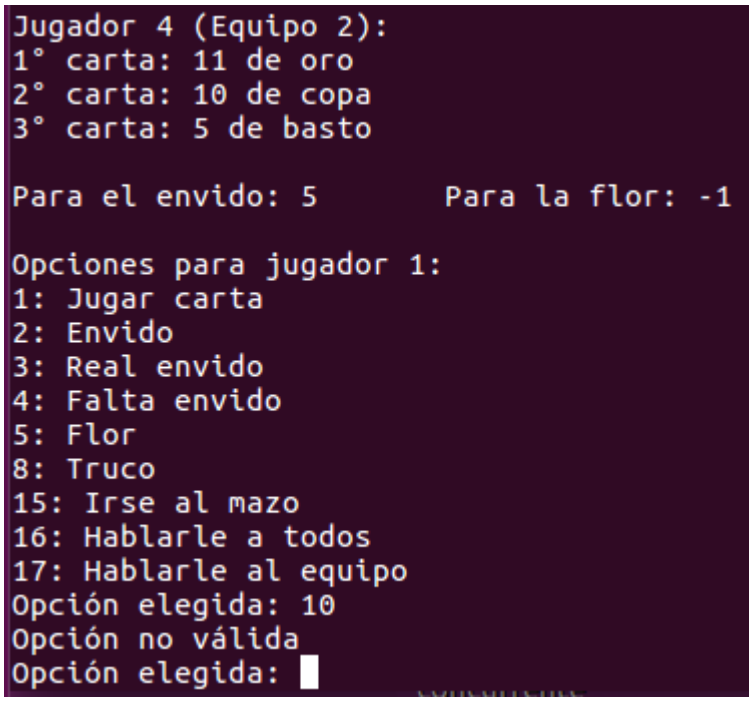
Mediante las siguientes capturas se muestra el funcionamiento del programa del juego. La ejecución se realizó con la sentencia make.

```
federico@federico-MS-7641:~/Escritorio/gittruco/truco$ make

Compilando archivos .c ...
gcc -Wall -o truco.o -c truco.c

Compilando archivos .c ...
gcc -Wall -o funciones.o -c funciones.c

Se crearon los archivos objeto truco.o funciones.o
Linkeando archivos objeto ...
gcc -Wall -o truco truco.o funciones.o

Se creó el archivo ejecutable truco

Ingrese modo de juego (presione S para jugar con flor o N para jugar sin flor):
T
Opción incorrecta
Borrando archivos objeto ...
Archivos objeto eliminados

Makefile:15: fallo en las instrucciones para el objetivo 'run'
make: *** [run] Terminado
Terminado
```

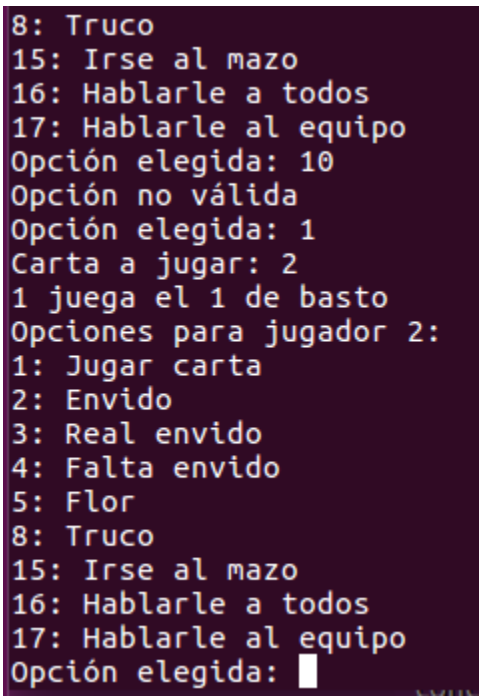
Interrupción de la ejecución del juego debido a elegir una opción no válida para el modo de juego.

```
Compilando archivos .c ...
gcc -Wall -o truco.o -c truco.c

Compilando archivos .c ...
gcc -Wall -o funciones.o -c funciones.c

Se crearon los archivos objeto truco.o funciones.o
Linkeando archivos objeto ...
gcc -Wall -o truco truco.o funciones.o

Se creó el archivo ejecutable truco

Ingrese modo de juego (presione S para jugar con flor o N para jugar sin flor):
S
Ingrese cantidad de jugadores por equipo (hasta 3): 7
Cantidad de jugadores incorrecta
Borrando archivos objeto ...
Archivos objeto eliminados

Makefile:15: fallo en las instrucciones para el objetivo 'run'
make: *** [run] Terminado
Terminado
```

Interrupción de la ejecución del juego debido a elegir una cantidad incorrecta de jugadores por equipo.



Habiendo elegido jugar con flor en un partido de dos contra dos, a cada jugador se les muestra sus cartas y el tanto para el envido y la flor. Aquí se muestra las opciones para el Jugador 1 que tiene el turno. Al elegir un código de acción incorrecto, se le pide ingresar otro número.



El Jugador 1 elige jugar una de sus cartas (opción 1). Luego decide jugar la segunda carta que se le repartió, en este caso, el 1 de basto.

Manejo de archivos y listas:

En el ítem Manejo de sockets se mencionaron los archivos de texto y binarios utilizados tanto para el servidor como para el cliente. Explicaremos como se trabaja con cada uno de ellos.

config.txt:

Tanto para el servidor como para el cliente, para completar los datos de dirección IP y puerto del servidor en la estructura sockaddr_in, en cada parte se lee un archivo config.txt. De dicho archivo de texto se obtiene lo siguiente:

- IP: 127.0.0.1
- Puerto: 3424

El archivo se abre en modo solo lectura.

usuariosregistrados:

Archivo binario que guarda nombre, contraseña y puntos de todos los jugadores registrados. El servidor es el encargado de administrar este archivo. Se abre con la función registrar en modo a+ para agregar al final del archivo un nuevo registro. También es abierto en modo lectura en varias funciones para distintas situaciones, por ejemplo en comprobarnombre para verificar que un usuario nuevo no haya escogido un nombre igual al de otro ya incorporado.

ipensesion:

Archivo binario que guarda los datos de los usuarios que tienen sesión abierta en el momento. Es administrado por el servidor, que lo abre en modo a+ con la función iniciarsesion y en modo lectura en cerrarsesion para eliminar el registro de un usuario que se desconecta, así como por comprobarnombre para verificar que un mismo usuario no pueda tener dos sesiones abiertas al mismo tiempo.

trucoS1:

Archivo binario de log administrado por el servidor. Se abre con iniciolog en modo a+ y por salidalog primero en modo lectura para identificar registros a borrar y luego en modo escritura para guardar aquellos usuarios que permanecerán en el log esperando jugar.
Aquí se registran los clientes que quieren jugar con flor en un partido de uno contra uno. Cuando haya dos jugadores, ellos competirán entre sí y se los borra del log.

trucoN1:

Id. trucoS1 pero para jugar sin flor.

trucoS2:

Id. trucoS1 pero para jugar dos contra dos. Se abre un nuevo juego al haber cuatro jugadores en el log.

trucoN2:

Id. trucoN2 pero para jugar sin flor.

trucoS3:

Id. trucoS2 pero para jugar tres contra tres. Se abre un nuevo juego al haber seis clientes en el archivo.

trucoN3:

Id. trucoS3 pero para jugar sin flor.

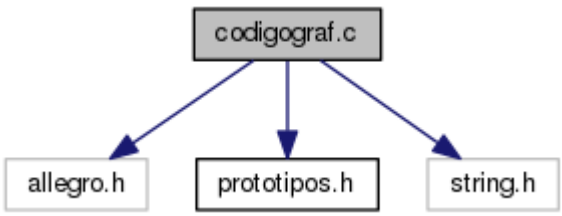
Manejo de procesos:

En la función main del servidor, el server crea un proceso hijo el cual atiende a un cliente en particular. Para dicha creación se llama a la función fork inmediatamente después de aceptar la conexión de un cliente. Luego se verifica si el proceso que se ejecuta es el padre o el hijo (cabe recordar que luego de ejecutar fork, el proceso hijo copia por valor la variables del proceso padre y el código de ese proceso).
En caso de ser el proceso padre, se procede a cerrar el socket abierto por la función accept y mantiene abierto aquél abierto por la función socket que se queda esperando a que otro cliente se conecte.
En tanto, el proceso hijo cierra el socket general, quedándose con aquél abierto por accept que será utilizado para comunicarse exclusivamente con un socket en particular. A este mismo le llegarán las solicitudes de dicho cliente para que el proceso hijo prepare la respuesta a ser enviada y realizar oras acciones, tal como modificar el contenido de archivos.

Interfaz gráfica:

Todo juego requiere una interfaz visual intuitiva y práctica para que el jugador sepa que pasos debe seguir para jugar, registrarse o realizar cualquier otra acción que le brinde el programa.
Para ello, se diseñó una versión del cliente que contempla la interacción del usuario con el servidor a través de una ventana que se abre al ejecutar el programa.
La ventana fue desarrollada utilizando funciones de la librería Allegro 4.2 para la programación de videojuegos.
Los archivos que componen esta parte del proyecto son:

codigograf.c:



En este archivo se encuentran las siguientes funciones:

int main (void)
Función principal de la interfaz gráfica.
Se crea una ventana a partir de imagenes bmp/tga, con la incorporación de texto según cada submenú.

void abort_on_error (const char * message)
Función utilizada para mostrar un mensaje de error y abortar el programa.
Parámetros
[in] message Mensaje a mostrar

void closebuttonhandler (void)
Handler para permitir cerrar ventana presionando el botón cerrar (x) de la ventana.
Este handler es trapeado por un puntero a función declarado por Allegro denominado set_close_button_callback.

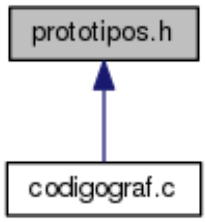
int my_strlen (char * palabra)
Idéntica a aquella explicada en el archivo otrasfunc.c de la parte del cliente.

int myStrncmp (const char * s1, const char * s2, int n)
Idéntica a aquella explicada en el archivo manejoarchivos.c de la parte del servidor.

int myStrncpy (char * dest, char * origen, int n)
Idéntica a aquella explicada en el archivo manejoarchivos.c de la parte del servidor.

prototipos.h:

Los prototipos de las funciones utilizadas y enumeradas al mencionar los demás archivos del cliente están en prototipos.h



Por otra parte, allí están definidas varias constantes simbólicas y las estructuras:

struct usuario (estructura definida con el alias DATO):
Campos de datos:
char user [MAX_USER+1] (MAX_USER está definido como 40)
char password [MAX_PASS+1] (MAX_PASS está definido como 24)

int puntos

Datos del usuario: Nombre de usuario, contraseña (ambos son vectores de chars) y puntos acumulados (entero)

Otros archivos:

Se incluye un Makefile con las siguientes instrucciones:

- Compilar con make new o make clienteui
- Ejecutar o compilar y ejecutar en un paso con make run o simplemente make (luego de la ejecución se borrarán los archivos objeto y el ejecutable)
- Generar y abrir página web con documento Doxygen con make html
- Borrar web con make no_html
- Agregar archivos al repositorio con make git

Por otra parte el programa carga todas las imágenes .bmp y .tga utilizadas para las pantallas de cada menú y submenú.

Imagen portada.tga utilizada como ventana inicial del juego.

En cada submenú el programa mostrará un texto que puede ser una información o una instrucción a seguir según lo solicitado por el usuario. Para ello se debe agregar un texto sobre una imagen que sirve como plantilla. De modo de mostrar ese texto, primero se debe cargar una fuente en formato .pcx.

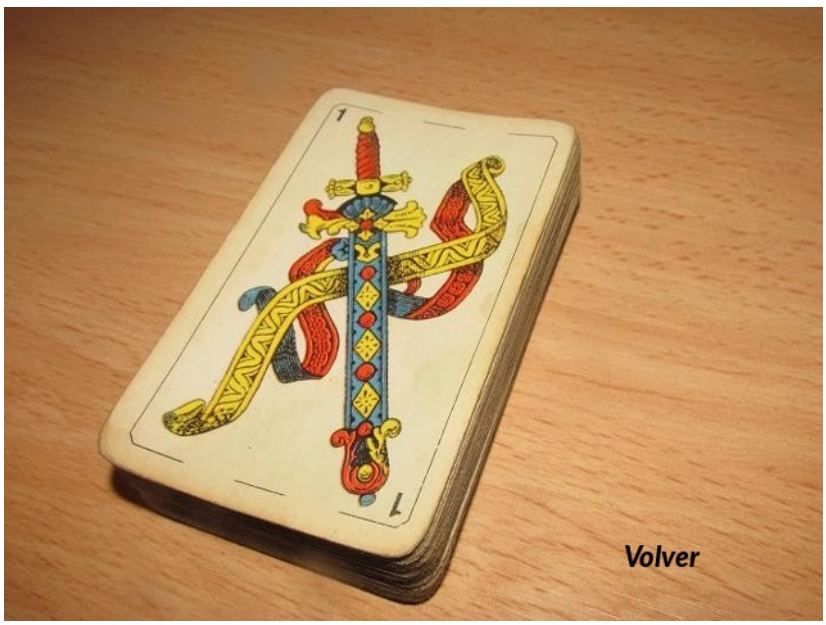


Imagen portadavolver1.tga que sirve como plantilla para los submenús.

Muestra de caracteres de la fuente Trebuchet MS, cursiva, 20, en formato .pcx

Funciones, variables y estructuras de Allegro utilizadas:

allegro_init:

Prototipo:
int allegro_init();
Función que inicializa la biblioteca Allegro (se linkea en forma dinámica).

END_OF_MAIN:

Macro:
END_OF_MAIN();
Se la coloca luego de la llave de cierre de la función main. Debe ser utilizada para que el código Unix obtenga una copia de los valores de argv[] y poder ubicar archivos de configuración y traducción.

allegro_error:

extern char allegro_error[ALLEGRO_ERROR_SIZE];
Guarda el último mensaje de error de Allegro.

allegro_message:

Prototipo:
void allegro_message(const char *text_format, ...);
Muestra un mensaje usando el formato de printf.

install_mouse:

Prototipo:
int install_mouse();
Para poder utilizar el mouse en la interfaz gráfica.

install_keyboard:

Prototipo:
int install_keyboard();
Para poder utilizar el teclado en la interfaz gráfica.

key:

extern volatile char key[KEY_MAX];
Array de flags que contiene el estado de una tecla (1 si está presionada o 0 si no lo está).

keypressed:

Prototipo:
int keypressed();
Determina si hay teclas presionadas guardadas en el buffer de teclado para leer.

readkey:

Prototipo:
int readkey();
Lee teclas del buffer de teclado.

clear_keybuf:

Prototipo:
void clear_keybuf();
Limpia el buffer de teclado.

FONT:

Estructura que guarda los datos de una fuente.

font:

extern FONT *font;
Puntero global que apunta a una fuente predeterminada por Allegro, pero que puede guardar cualquier otra fuente que se cargue.

load_font:

Prototipo:
FONT *load_font(const char *filename, RGB *pal, void *param);
Carga la fuente cuya ruta se envía a filename (la fuente debe estar en formato pcx). Actúa como malloc, asignando recursos para el almacenamiento de la fuente.

destroy_font:

Prototipo:
void destroy_font(FONT *f);
Libera recursos asignados a la fuente (actúa como free).

BITMAP:

Estructura que guarda los datos de una imagen.

create_bitmap:

Prototipo:
BITMAP *create_bitmap(int width, int height);
Creo un mapa de bits en memoria de ancho=width y alto=height. Actúa como malloc, asignando memoria para este mapa de bits.

load_bitmap:

Prototipo:
BITMAP *load_bitmap(const char *filename, RGB *pal);
Carga un mapa de bits desde un archivo .bmp, .lbm, .pcx, o .tga. Actúa como malloc, al asignar los recursos asociados a este mapa de bits.

blit:

Prototipo:
void blit(BITMAP *source, BITMAP *dest, int source_x, int source_y, int dest_x, int dest_y, int width, int height);

Pega una porción rectangular de una imagen en otra. Si source_x y source_y son 0, se pega todo el mapa de bit source, cuya esquina superior se colocará en la coordenada (dest_x;dest_y) de dest.

masked_blit:

Prototipo:
void masked_blit(BITMAP *source, BITMAP *dest, int source_x, int source_y, int dest_x, int dest_y, int width, int height);
Como blit(), pero salta los píxeles transparentes (cero en modos de 256 colores, y rosa fucsia para modos truecolor). La imagen origen (source) debe ser un bitmap de memoria o un sub-bitmap, y las regiones de origen y destino no pueden superponerse.

textout_centre_ex:

Prototipo:
void textout_centre_ex(BITMAP *bmp, const FONT *f, const char *s, int x, int y, int color, int bg);
Actúa como puts, imprimiendo una cadena de caracteres en un mapa de bits a y píxeles del borde superior, obteniendo un bloque de texto cuyo centro se ubica a x píxeles del borde izquierdo.

textout_justify_ex:

Prototipo:
void textout_justify_ex(BITMAP *bmp, const FONT *f, const char *s, int x1, int x2, int y, int diff, int color, int bg);
Actúa como puts, imprimiendo una cadena de caracteres en un mapa de bits a y píxeles del borde superior, obteniendo un bloque de texto que arranca a x1 píxeles del borde izquierdo y finaliza a x2 píxeles de dicho borde. La justificación de texto se logra si diff es mayor o igual a la diferencia entre x2 y x1, caso contrario, el texto se imprimirá alineado a la izquierda desde x1.

Dificultades:

Se mencionó en el ítem El juego propiamente dicho que la parte del juego de truco está separada del servidor/cliente. Esto se debe a que el servidor se encuentra en período de prueba para poder lograr que se comunique con dos clientes al mismo tiempo. Actualmente, si llega las conexión de dos clientes, el servidor solamente atiende al primero y deja en espera al segundo hasta que el primero se desconecte.
En cuanto a la interfaz gráfica, se mencionó que la misma serviría como una interfaz visual entre el usuario y las respuestas que da el servidor. Debido a la gran complejidad para imprimir la respuesta del server en la ventana, esto no pudo completarse.

Conclusiones:

La realización del juego TrucoC con los objetivos planteados en la introducción demanda un tiempo prolongado, sobre todo para la implementación de la interfaz gráfica.
La parte más complicada y larga para armar fue el desarrollo del juego en sí, debido a que se deben contemplar gran cantidad de situaciones que pueden ocurrir durante la partida.
Gracias a los conocimientos adquiridos, se pudo avanzar en un servidor y un cliente acordes para que cada usuario realice las consultas que comúnmente puede hacer en cualquier otro juego, tal como iniciar sesión, ver puntuaciones.
En nuestra opinión, la parte central del trabajo está en el servidor debido a que es el encargado de administrar todo el juego y los archivos necesarios para el correcto funcionamiento de TrucoC.

Bibliografía:

Manual de Allegro 4.2 (en inglés): https://www.allegro.cc/manual/4/
“El truco” de Juan Carlos Ortega: http://www.latitud-argentina.com/blog/images/Truco-criollo.pdf