

# Arquitectura de Computadoras

## Trabajo Práctico Especial



Integrantes:

- Prado Torres, Macarena      legajo: 59.711
- Rodríguez, Martina      legajo: 59.036
- Tarantino Pedrosa, Ana      legajo: 59.147

Fecha de Entrega: 08 de Noviembre de 2020

## Introducción

En este trabajo práctico especial se implementó un pequeño sistema operativo. En todo momento se priorizó la separación entre Kernel y Userland pensando en preservar la seguridad del sistema. La comunicación entre ellas se efectúa mediante llamados a *systemCalls* programadas en Kernel.

## Kernel

Al correr el programa en un primer momento, se ejecuta el main del archivo kernel.c. Este se encarga de llamar a las funciones *load\_idt* e *initializeVideo* explicadas más adelante. Otra función dentro del main muy importante es *initExceptionHandler* que se va a encargar de preservar registros para que una vez que ocurran excepciones el sistema pueda recuperarse. Por último, se llama a *sampleCodeModule*. Esta se ocupa de ejecutar la función *startShell*, que da inicio a la terminal.

## Drivers

En esta sección se dispone de los drivers de teclado y de video. Dado que el trabajo requiere de una interfaz visual, se empezó por el driver de video, en el cual se utiliza la función *initializeVideo* para inicializar la pantalla con los valores que consideramos default. Dicha función es accedida desde el main en kernel.c . A su vez, en este driver se pueden encontrar las funciones requeridas para la impresión de caracteres, como *printChar* o *printCharOnScreen*. Ambas utilizan la función *charMap* de font.c para obtener el mapa de pixeles que luego se imprimirán en *drawPixel*. La diferencia entre estas dos funciones que permiten imprimir caracteres es que, en el caso de *printCharOnScreen*, permite también imprimir en una posición cualquiera de la pantalla. Por último, se agregaron funcionalidades como *deleteChar*, *newLine*, *clearScreen* y *clearSpace* para el borrado y espaciado en la pantalla, además de la función *scrollScreen* para un manejo más dinámico de la misma.

Por otro lado, en el driver de teclado se crearon las funciones necesarias para el manejo de las interrupciones de hardware de teclado. Tal es el caso de *keyboard\_handler* donde, a partir del código de *get\_key* y *pressed\_key* en keyboard.asm , y haciendo uso de una matriz representativa del teclado en inglés, se añadía al buffer el carácter correspondiente. Una particularidad de esta función es que tiene en consideración el uso de teclas como shift, caps locked y control. Esta última tecla se agregó principalmente para la función *updateRegisters*, que se encarga de ir actualizando el valor de los registros (Ctrl+r) dentro del programa. Esta función es útil para el buen funcionamiento de los comandos de la shell en los cuales se pide mostrar en pantalla los registros en cierto momento del programa. Otra función implementada en este archivo fue *getChar* que, como su nombre lo indica, le solicita al usuario un carácter y este se obtiene buscándolo en una posición específica en el buffer.

## IDT

En este apartado se trató todo lo referido a las interrupciones. En *idtLoader* se conectaron las interrupciones con sus descriptores. De esta manera, al recibirse una interrupción, la misma es atendida por su función correspondiente. Se tratan dos tipos de excepciones, división por cero y código de operación inválido, e interrupciones como la del teclado y la de timer tick.

En *exceptions.c*, la función *exceptionDispatcher* se encarga de imprimir el tipo de excepción generada y el estado de los registros en ese momento. Asimismo, se ocupa de setear el *RIP* y *RSP* para que se pueda retornar al programa una vez lanzada cualquiera de estas dos excepciones.

Por su parte, *irqDispatcher* se responsabiliza del manejo de las interrupciones del teclado (interrupción 21) y del timer (interrupción 20). El mismo, en caso de que ocurra la interrupción 20 llama al *timer\_handler* el cual suma un tick a los ticks totales y, en caso de la otra interrupción, llama al *keyboard\_handler* mencionado previamente.

Por último, se encuentra *syscallDispatcher* que maneja la interrupción 80. Según el parámetro recibido en *rax* se ejecuta la rutina correspondiente (se asume que se reciben los demás parámetros correspondientes, en el orden adecuado). Las syscalls que se decidieron incorporar para ofrecer al usuario desde Kernel fueron las siguientes:

1. **READ**: Llama a *getChar* el cual espera una interrupción de teclado y retorna dicho caracter.
2. **WRITE**: Llama a *syscallWrite* encargada de escribir en pantalla. Está syscall es utilizada por funciones como *printf*, *putChar*, *println*, entre otras.
3. **SCREEN**: Inicializa la pantalla.
4. **CLEAR**: Pinta la pantalla de negro.
5. **REGISTERS**: Retorna un puntero con el estado de los registros.
6. **TIMERTC**: Devuelve la cantidad de segundos desde que se inició el sistema.
7. **DRAW**: Dibuja un pixel en pantalla.
8. **CLEARSPACE**: Pinta una determinada porción de la pantalla (recibiendo por parámetros dónde empieza y termina en las coordenadas x e y) de algún color pasado por parámetro.
9. **WRITEONSCREEN**: Permite escribir en algún lugar específico de la pantalla. Recibe el puntero a *char* a imprimir, su longitud, la posición en x e y, el color de fuente y fondo.
10. **TICKSELAPSED**: Retorna la cantidad de ticks que pasaron desde que se inició el sistema.
11. **CHARINTERRUPT**: Es similar a *READ* solo que en vez de esperar que se ingrese un caracter por teclado mediante un while, solo se hace una llamada a *hlt*. Esta interrupción se hizo a partir de que se necesitaba imprimir un cronómetro para el

ajedrez. Por ende, el ciclo que se realizaba en *READ* ahora lo puede controlar el usuario.

## ASM

Se dispone de tres archivos en lenguaje Assembler dentro de Kernel. El primero, *interrupts.asm* está enfocado en las interrupciones. El mismo incluye los handlers para las interrupciones. El segundo es *keyboard.asm*, este contiene las funciones *get\_key*, que obtiene el scan code de la tecla presionada, y *pressed\_key*, que indica si una tecla está siendo presionada en el momento. Finalmente, *lib.asm* tiene distintas funciones como *cpuVendor* (provista por la cátedra) que ofrece información acerca del CPU y su marca, *getRTC*, encargada de devolver el *real time clock* y *getRSP* la cual, como indica su nombre, devuelve el *rsp*. Se agregaron las macros *popaq* y *pushaq* las cuales se usan para preservar todos los registros.

## Libraries

Se desarrollaron cuatro librerías con las funciones necesarias para utilizar dentro de Kernel. Una de ellas es *font* (provista por la cátedra) la cual contiene mapa de bits con todos los caracteres y una función para poder acceder a cada uno de ellos. La librería *prints* contiene aquellas funciones que se utilicen para imprimir en pantalla. Hay dos funciones que se destacan. Una es *syscallWrite*, la misma recibe un *char \** a imprimir, su longitud y los colores de fuente y fondo e imprime en pantalla donde está el cursor ubicado. Por otro lado, está *syscallWriteOnCurrentPos* que es similar a la anterior, solo que recibe las coordenadas en x e y donde se quiere comenzar a escribir. Esta función fue hecha ya que es recurrente que se necesite imprimir en otro lugar de la pantalla que difiere de donde está el cursor. Nos pareció apropiada ya que la otra alternativa encontrada fue darle la posibilidad al usuario de fijar el cursor en otra posición cada vez que se quiera escribir en otro lado y que luego se encargue de volver el cursor a su posición original. Sin embargo, esta última manera dependía de que el usuario recuerde devolver el cursor a su posición original.

La librería *lib* fue provista por la cátedra y no sufrió cambios. Contiene la función *memset* que recibe un puntero, un caracter c y una longitud. Se encarga de copiar en el puntero el caracter recibido la cantidad de veces que indica la longitud. Por otro lado, *memcpy* recibe un puntero destino, uno fuente y una longitud. Copia n (lo que indique la longitud) caracteres del puntero fuente al puntero destino.

*RTCTime* es la encargada de devolver los segundos que pasaron desde que inició el sistema. Contiene una única función *getTime* que dentro de ella hace una llamada a *getRTC*, desarrollada en *libasm* para obtener el *real time clock*. Esta función recibe un *time\_type* descriptor el cual es un enum que define constantes correspondientes al año, mes, día, día de la semana, horas, minutos y segundos. Devuelve la información correspondiente al descriptor ingresado.

Finalmente, la última biblioteca es la de *strings*. Ofrece una función *strlen* que calcula la longitud de un *char \**, y las funciones *uintToBase* y *uintToBaseHexa*, la primera provista por la cátedra, y la segunda con un pequeño agregado a la primera, que convierten valores de tipo

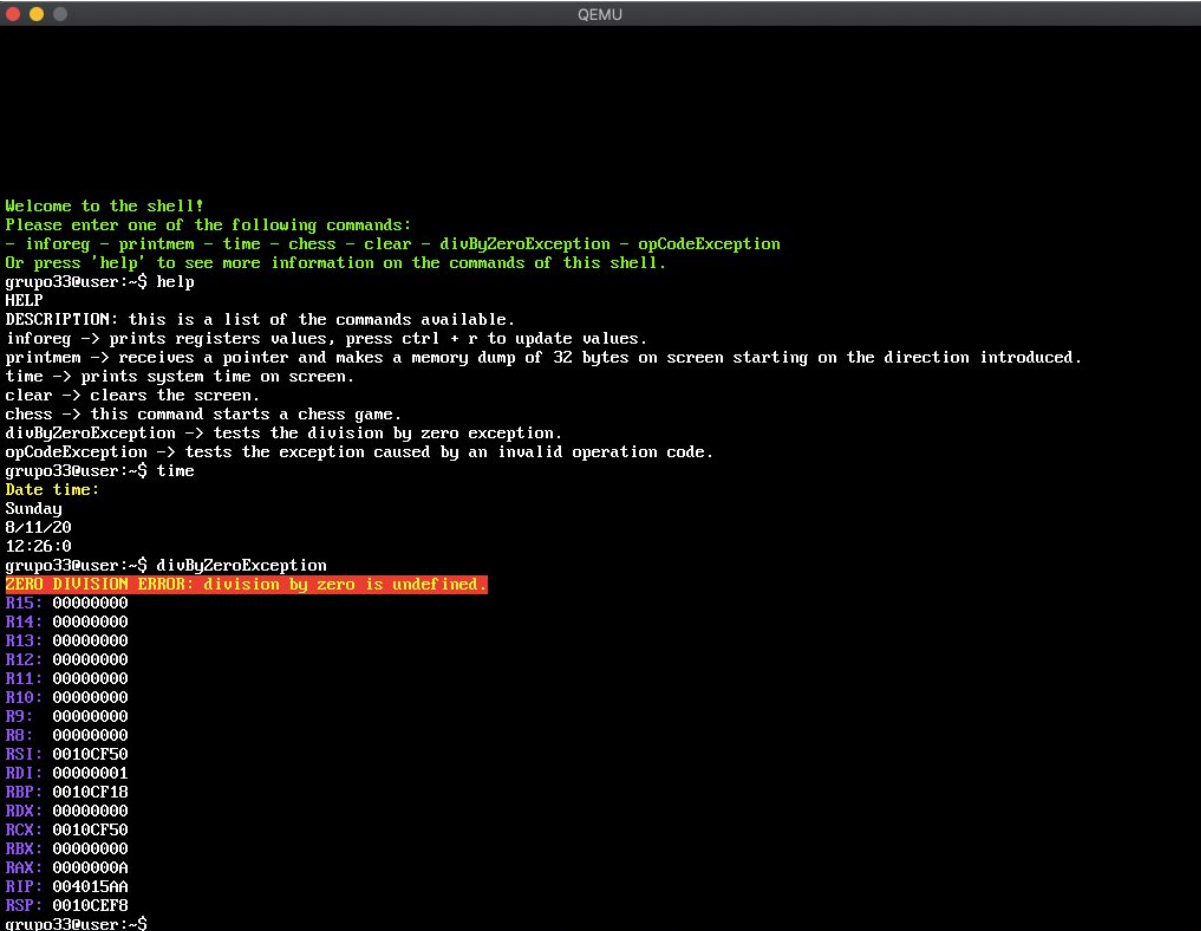
`uint64_t` en la base ingresada en el caso de la primera, y un `uint64_t` a hexa en el caso de la segunda.

## Userland

### ASM

En el archivo `libasm.asm` se creó una de las funciones más importantes de este programa, *syscalls*. Esta es la encargada de la comunicación entre Kernel y Userland. Se utilizó la misma convención que Linux para el pasaje de parámetros. En `rax` se ingresa el valor de la `syscall` a ejecutar y luego en el resto de los registros para pasar otros parámetros. Al momento de preservar los registros a utilizar, no se incluyó a `rax` para poder devolver el valor de las funciones. Para ver más en particular sobre cada `syscall` ver el **Apéndice**.

### Shell



```
QEMU

Welcome to the shell!
Please enter one of the following commands:
- inforeg - printmem - time - chess - clear - divByZeroException - opCodeException
Or press 'help' to see more information on the commands of this shell.
grupo33@user:~$ help
HELP
DESCRIPTION: this is a list of the commands available.
inforeg -> prints registers values, press ctrl + r to update values.
printmem -> receives a pointer and makes a memory dump of 32 bytes on screen starting on the direction introduced.
time -> prints system time on screen.
clear -> clears the screen.
chess -> this command starts a chess game.
divByZeroException -> tests the division by zero exception.
opCodeException -> tests the exception caused by an invalid operation code.
grupo33@user:~$ time
Date time:
Sunday
8/11/20
12:26:0
grupo33@user:~$ divByZeroException
ZERO DIVISION ERROR: division by zero is undefined.
R15: 00000000
R14: 00000000
R13: 00000000
R12: 00000000
R11: 00000000
R10: 00000000
R9: 00000000
R8: 00000000
RSI: 0010CF50
RDI: 00000001
RBP: 0010CF18
RDX: 00000000
RCX: 0010CF50
RBX: 00000000
RAX: 0000000A
RIP: 004015AA
RSP: 0010CFB8
grupo33@user:~$
```

En esta parte del código se empieza con una función *startShell* encargada en un primer momento de imprimir un mensaje para ayudar al usuario dentro de la terminal. Luego se utiliza un ciclo `while(1)` que su propósito es que el programa no finalice ante la primera instrucción dada. Dentro de esa función, se llamó a *getChar* y luego se prosiguió a analizar

ese caracter ingresado. *analyzeChar* tiene en su interior un switch con tres diferentes casos: ‘\n’, ‘\b’, default. El primero de estos significaba que el usuario había ingresado un enter, por lo cual se debía analizar lo que este ingreso previo al ‘\n’ para saber de qué comando se trataba y si era o no válido. Para lograr esto, se invoca a *findCommand*. El segundo, indicaba que el usuario tenía intención de borrar texto escrito en la shell, por lo cual se prosiguió a llamar a una función llamada *removeChar* que se encargaba de eso. Por último, en el caso default, simplemente se empezó a agregar al buffer el char ingresado.

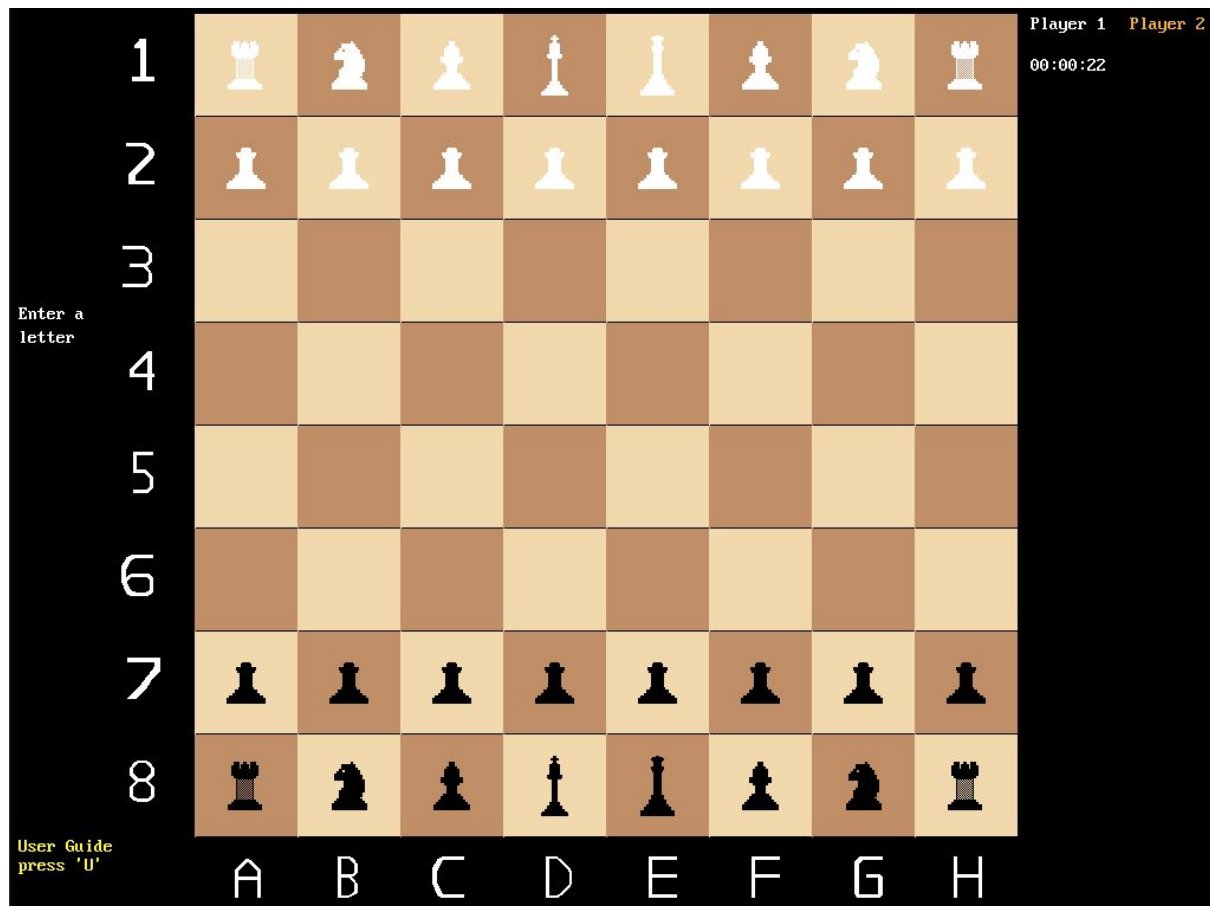
Por su parte, *findCommand* se encargó mediante la función de *strtok* de separar por tokens el buffer que contenía todo lo ingresado por el usuario. Estos tokens se guardaron en un vector y esto sirvió para lograr poder comparar el primer argumento de este array con un array previamente definido con todos los comandos válidos en esta shell. Si estos coincidían, se llamaba a la función *applyCommand* la cual realiza un switch dependiendo del comando obtenido. Los comandos son los siguientes:

- **inforeg** se encarga de obtener el estado de los registros en un momento determinado para luego imprimirlos.
- **printmem** recibe como argumento un puntero y realiza en memoria un volcado de memoria de 32 bytes a partir de la dirección recibida como argumento.
- **time** devuelve la hora y la fecha utilizando funcionalidades implementadas en Kernel.
- **help** es una guía que se encarga de informarle al usuario todos los comandos que tiene disponibles en esta shell.
- **clear** hace que toda la pantalla se limpie.
- **divisionByZero** causa una excepción de división por cero.
- **opCodeException** causa una excepción de código de operación invalido.
- **chess** despliega un juego de ajedrez de interacción humano-humano.

## Libraries

En esta sección se implementan aquellas funciones que consideramos propias de una librería estándar para nuestro Userland. Las diferenciamos en cuatro librerías: lib, prints, strings y timeRTC. En el caso de lib, todas las funciones fueron provistas por la cátedra a excepción de *getRegisters*, cuyo objetivo es realizar una system call para obtener el estado de los registros. Por otro lado, en timeRTC están las funciones de manejo del tiempo, como *getTime* y *ticks\_elapsed* con sus system calls correspondientes. En tercer lugar, en prints.c se realiza el manejo de todas las funciones que imprimen en pantalla, en adición a la impresión del cronómetro y las funciones de limpieza de pantalla y de área delimitada. Cabe mencionar que para el cronómetro se decidió que la impresión sea en formato hh:mm:ss. Por último, en strings.c se encuentran todas las funciones de manejo de caracteres como *getChar*, *strlen*, *stringcmp*, *scanf*, entre otras.

## Chess



En esta carpeta se trató de mantener la funcionalidad del ajedrez en un archivo solo mientras que la parte gráfica y el archivo de fuentes se trataron por separado. En *chess\_piece* se puede observar un manejo de las fuentes similar a *font.c* de Kernel. Por otra parte, en *graphics.c* se trabajó la parte visual del programa, que consta de la impresión del tablero con sus colores correspondientes, de los ejes de coordenadas y de las piezas. Para ello se hizo uso de una matriz que representa los 64 casilleros del tablero y constantes numéricas que representan a cada una de las piezas, siendo los números positivos las piezas blancas y los números negativos las piezas negras. Para más información sobre las piezas, ver apéndice.

El ajedrez se implementó con una función principal denominada *playChess* la cual permite inicializar el juego o continuar un juego guardado, imprime la parte gráfica y comienza el juego. En primer lugar, se brindan varias posibilidades de uso dentro del juego con algunas teclas especiales (ver apéndice) que permiten, por ejemplo, acceder al registro de jugadas o acceder a una guía para el usuario o rotar el tablero. Luego, se recibe por entrada estándar las coordenadas de la posición inicial de la pieza a mover (Ejemplo, “a2” para mover el peón blanco de más a la izquierda”) seguido de la posición final (“a4”). En caso de ingresar una coordenada inválida en cualquiera de los casos inicial o final, se le pedirá al usuario que

ingrese una coordenada válida. Luego, se analizará si el movimiento a realizar es un movimiento permitido dentro de las reglas del ajedrez. Para ello se utilizaron distintas funciones dependiendo de la pieza a mover, cada una con el nombre correspondiente a la pieza. Finalmente, se realizará la jugada y será el turno del siguiente jugador.

Cabe mencionar algunos detalles propios del juego. Por un lado, se cuenta con dos cronómetros, uno para cada jugador. Al momento que la diferencia de tiempos entre ambos jugadores supera el minuto, perderá el jugador que se haya excedido de tiempo. Por otro lado, el juego tiene en consideración jugadas especiales como el peón al paso, el enroque y el cambio de pieza (ver Leyes del Ajedrez en Referencias). A su vez, se implementó la función de jaque, que permite saber si hay una pieza atacando al rey. Por último, en los registros de jugadas se respetó la Notación Algebraica del ajedrez en el idioma inglés. Se optó por utilizar el idioma inglés en el juego ya que nuestro teclado corresponde al teclado inglés.

## **Ventajas y desventajas**

En este trabajo se procuró sacar el mayor provecho a la separación en módulos por lo que se decidió utilizar una única función para el manejo de las system calls y así proteger el sistema operativo. Además, se separaron aquellas funciones que se consideraron propias de un sistema operativo de aquellas propias del uso cotidiano para evitar realizar múltiples llamadas a funciones del Kernel. A su vez, se eligió trabajar mayoritariamente en el lenguaje C por su portabilidad y simplicidad.

Dentro de la separación de funciones, cabe mencionar la decisión tomada para la ubicación de ciertas funciones. Tal es el caso de las funciones de impresión en una posición particular de la pantalla, las cuales consideramos de una utilidad inmensurable y una ventaja considerable para un sistema operativo, por lo tanto, se decidió que dichas funciones pertenecían a Kernel. Por otro lado, para el manejo del Timer, utilizado en el ajedrez, se decidió optar por una nueva versión del getChar donde el ciclo quede a cargo del usuario. De esta manera, es el usuario quien decide cómo imprimir el contador y si quiere agregar más funcionalidades dentro. Otra opción hubiese sido que el contador se imprimiera desde kernel pero esto no permitiría modificarlo en su función o cambiar el formato de impresión.

En el juego del ajedrez, una desventaja es que no se ven las jugadas ingresadas en tiempo real, por lo que no se puede borrar un caracter ingresado en caso de equivocarse. Esto no se incorporó para que no haya confusión con la impresión de los logs. En cambio, se priorizó hacer hincapié en la validez de las jugadas realizadas.

Para finalizar, una ventaja de nuestro juego es que es muy informativo y cuenta con su propia guía de usuario.



## Referencias

Linux System Call Table for x86 64

[https://blog.rchapman.org/posts/Linux\\_System\\_Call\\_Table\\_for\\_x86\\_64/](https://blog.rchapman.org/posts/Linux_System_Call_Table_for_x86_64/)

Leyes del Ajedrez

[https://es.wikipedia.org/wiki/Leyes\\_del\\_ajedrez#Captura\\_al\\_paso\\_del\\_pe%C3%B3n](https://es.wikipedia.org/wiki/Leyes_del_ajedrez#Captura_al_paso_del_pe%C3%B3n)

Notación Algebraica

[https://es.wikipedia.org/wiki/Notaci%C3%B3n\\_algebraica](https://es.wikipedia.org/wiki/Notaci%C3%B3n_algebraica)

Opcode Exception

[https://mudongliang.github.io/x86/html/file\\_module\\_x86\\_id\\_318.html](https://mudongliang.github.io/x86/html/file_module_x86_id_318.html)

Keyboard Scan Codes

[https://wiki.osdev.org/PS/2\\_Keyboard](https://wiki.osdev.org/PS/2_Keyboard)

Real Time Clock

<https://wiki.osdev.org/CMOS>

## Apéndice

- Tabla de uso de los Registros para las Syscalls:

%rax	System call	%rdi	%rsi	%rdx	%r10	%r8	%r9
0	Read	-	-	-	-	-	-
1	Write	char *str	uint_64 length	uint_64 f_color	uint_64 bg_color	-	-
2	Screen	-	-	int f_color	int bg_color	-	-
3	Clear	-	-	-	-	-	-
4	Registers	-	-	-	-	-	-
5	TimeRTC	time_type descriptor	-	-	-	-	-
6	Draw	-	-	int color	-	int x	int y
7	ClearSpace	uint_32 startx	uint_32 starty	uint_32 endx	uint_64 bg_color	uint_32 endy	-

8	WriteOnScreen	char *str	uint_64 length	uint_64 f_color	uint_64 bg_color	-	-
9	TicksElapsed	-	-	-	-	-	-
10	CharInterrupt	-	-	-	-	-	-

- Números representativos de las piezas en la matriz que se utilizó de guía para el tablero:

<b>Pieza</b>	<b>En inglés</b>	<b>Blancas</b>	<b>Negras</b>
Torre	Rook	2	-2
Caballo	Knight	3	-3
Alfil	Bishop	4	-4
Reina	Queen	5	-5
Rey	King	6	-6
Peón	Pawn	10 al 17	-10 al -17

- Teclas especiales para el Ajedrez:

<b>Caracter</b>	<b>Función</b>
‘N’ o ‘n’	Inicia un juego nuevo
‘C’ o ‘c’	Continuar el juego anterior
‘S’ o ‘s’	“Spin”, permite rotar el tablero
‘X’ o ‘x’	“Exit”, permite salir del juego
‘L’ o ‘l’	“Logs”, registro de jugadas
‘U’ o ‘u’	“User Guide”, guía de uso tanto del juego como de teclas especiales