# CFL Kernel

| Integrante | Legajo |
| --- | --- |
| Bautista Canevaro | 62179 |
| Alejo Flores Lucey | 62622 |
| Nehuen Gabriel Llanos | 62511 |



## Descripción general

CFL Kernel es un pequeño sistema operativo desarrollado a partir de Pure64, en el marco de la materia Arquitectura de Computadoras.

Es posible interactuar con el sistema a través de una terminal, que permite ejecutar diversos comandos para verificar su funcionamiento.

Para su utilización es necesario tener acceso a un teclado; el mouse no es utilizado.

> ⚠️ CFL Kernel supone que se posee de un teclado con distrubución ANSI "United States".

# Compilación y ejecución del kernel

## Prerequisitos

Se necesitan tener instalados los siguientes paquetes para compilar y correr este proyecto:

- nasm
- qemu
- gcc
- make
- docker

## Opción 1

Por única vez, descargar la imagen de Docker:

```
$ docker pull agodio/itba-so:1.0
```

Luego, cada vez que se quiera compilar, ejecutar el siguiente comando en el directorio del proyecto:

```
$ docker run -v ${PWD}:/root --security-opt seccomp:unconfined -ti agodio/itba-so:1.0
```

Se iniciará el contenedor de Docker. Ahora, ejecutar los siguientes comandos:

```
$ cd root
$ cd Toolchain
$ make all
$ cd ..
$ make all
$ exit
```

Ahora habremos cerrado el contenedor de Docker. Solo falta ejecutar el kernel:

```
$ ./run.sh
```

### Opción 2

Primero, por única vez descargar la imagen de Docker:

```
$ docker pull agodio/itba-so:1.0
```

Luego, ejecutar el siguiente comandos en el directorio del proyecto:

```
$ docker run -d -v ${PWD}:/root --security-opt seccomp:unconfined -ti --name tpe agodio/itba-so:1.0
```

Luego, cada vez que se quiera compilar el proyecto, ejecutar los siguientes comandos en el directorio del proyecto:

```
$ ./a-compiletpe
$ ./run.sh
```

# Comandos disponibles

`help`

Programa que despliega en pantalla una lista de comandos válidos para introducir, junto a una pequeña descripción del mismo.

`divide_by_zero`

Programa que realiza una división por cero. El procesador lanzará una excepción de tipo 0. El objetivo del programa es verificar el funcionamiento de dicha excepción.

`invalid_opcode`

Programa que intenta ejecutar una instrucción inexistente. El procesador lanzará una excepción de tipo 6. El objetivo del programa es verificar el funcionamiento de dicha excepción.

`inforeg`

Programa que imprime a pantalla una lista de los registros de uso general junto al valor que almacenaban en el momento que se tomó el snapshot.

Para guardar los registros en un cierto momento y luego poder imprimirlos con el comando, apretar la tecla $F10$.

`printmem <dirección_de_memoria>`

Programa que realiza un vuelco de memoria de 32 bytes a la pantalla a partír de la dirección de memoria obtenida como parámetro.

La dirección de memoria debe cumplir ciertos requisitos para ser una dirección válida:

- Debe empezar con $0x$
- Debe ser un valor hexadecimal válido
- Debe ser menor a $0xFFFFFFF8$. Esto se debe a que ese es el límite del mapa de memoria de QEMU.

`datetime`

Programa que imprime en pantalla la fecha y hora actual. Dicha fecha y hora es desplegada en GMT-3.

`primes`

Programa que imprime en pantalla los números primos a partír del 1.

> !! Se desplegarán correctamente los números primos hasta que se supere el número $2.147.483.647$. Esto se debe a que ese es el valor máximo para un entero.

`fibonacci`

Programa que imprime en pantalla los números de la serie de Fibonacci.

> !! Se desplegarán correctamente los números de la serie de Fibonacci hasta que se supere el número $18.446.744.073.709.551.615$. Esto se debe a que ese es el valor máximo para un *unsigned long long*.

`clear`

Programa que limpia lo impreso en pantalla y coloca el cursor en la esquina superior izquierda.

# Ejecución de dos comandos en simultáneo

Todos los programas anteriormente descriptos pueden ejecutarse en simultáneo utilizando el siguiente comando: `<programa_1> | <programa_2>`.

Cuando se corra dicho comando, se entrará en el modo de pantalla dividida. A la izquierda, se ejecutará el programa <programa_1> y a la derecha se ejecutará el programa <programa_2>.

Para salir del modo de pantalla dividida, presionar la tecla $ESC$.

# Teclas para ejecutar ciertas acciones

Para ejecutar ciertas acciones que disrrumpen el funcionamiento normal de la terminal, se han asignado teclas especiales del teclado.

Recordar que se supone que se posee de un teclado con distrubución ANSI "United States".

| Acción | Tecla asociada |
|---|---|
| Cancelar la ejecución de un programa que se está ejecutando en la pantalla completa | F1 |
| Pausar y reanudar la ejecución de un programa que se está ejecutando en la pantalla completa | F2 |
| Pausar y reanudar la ejecución del programa que se está ejecutando en la parte izquierda en la pantalla dividida | F3 |
| Pausar y reanudar la ejecución del programa que se está ejecutando en la parte derecha en la pantalla dividida | F4 |
| Salir del modo de pantalla dividida y cancelar la ejecución de ambos programas que se están ejecutando | ESC |
| Guardar el contenido de los registros de uso general para luego imprimirlos usando el comando `inforeg` | F10 |

# System Calls implementadas

Se debe generar una interrupción del tipo 80 para ejecutar la system call deseada.

Los registros que se detallan a continuación deben poseer los parámetros para la ejecución de la system call.

En *RAX* se indica qué system call se desea ejecutar.

El valor de retorno de la system call se obtendrá en *RAX*.

| System Call | RAX | RDI | RSI | RDX |
|---|---|---|---|---|
| Read | 0x00 | `int fd` | `char * buf` | `int count` |
| Write | 0x01 | `int fd` | `char * buf` | `int count` |
| Clear Screen | 0x02 | | | |
| Seconds Elapsed | 0x03 | | | |
| System Datetime | 0x04 | `uint64_t * info` | | |
| Print Byte From Memory | 0x05 | `int fd` | `uint8_t * address` | |
| Start Split Screen | 0x06 | | | |

| System Call | RAX | RDI | RSI | RDX |
|---|---|---|---|---|
| Start Unique Screen | 0x07 | | | |
| Load Process | 0x08 | `uint64_t function` | `arguments * args_function` | |
| Hibernate Process | 0x09 | `int pid` | | |
| Get Inforeg Registers | 0x0A | `uint64_t ** results` | | |

## Read

Recibe un file descriptor de donde leer, un vector de caracteres donde se guardarán los caracteres leídos y la cantidad de bytes a leer.

Retorna la cantidad de bytes leídos.

> 🗣️ El primer argumento (File Descriptor) se incluye para seguir el estándar de Linux. Sin embargo, en este kernel sólo se puede leer de la entrada estándar y eso es lo que ocurrirá sin importar el valor del parámetro

## Write

Recibe un file descriptor a donde escribir, un vector de caracteres con los caracteres a escribir y la cantidad de bytes a escribir.

Retorna la cantidad de bytes que se escribieron.

Los file descriptor disponibles son:

| | |
|---|---|
| 1 | Standart Output |
| 2 | Standard Error |
| 3 | Left Output |
| 4 | Left Error |
| 5 | Right Output |
| 6 | Right Error |

## Clear Screen

No recibe parámetros.

Retorna 1.

## Seconds Elapsed

No recibe parámetros.

Retorna la cantidad de segundos que pasaron desde que se inició el kernel.

## System Datetime

Recibe como parámetro un vector de `uint64_t` donde e guardarán todas las componentes de la fecha y la hora. Dicho vector debe ser de `6 * SIZEOF(uint64_t)`.

Retorna 1.

## Print Byte From Memory

Recibe un file descriptor a donde escribir y la dirección de memoria de la cual se desea imprimir su contenido.

Retorna 1.

Los file descriptor disponibles son:

| | |
|---|---|
| 1 | Standart Output |
| 2 | Standard Error |
| 3 | Left Output |
| 4 | Left Error |
| 5 | Right Output |
| 6 | Right Error |

## Start Split Screen

No recibe argumentos.

Retorna 1.

## Start Unique Screen

No recibe argumentos.

Retorna 1.

## Load Process

Recibe un puntero a la función que se desea cargar como proceso y un puntero a una estructura con los argumentos de dicha función (que pueden ser un string y/o un entero).

Retorna el ID del proceso creado ó -1 si ocurrió algún error.

La estructura es la siguiente:

```
// Rellenar con -1 si el argumento no es utilizado.
typedef struct {
  int integer;
  char * string;
} arguments;
```

## Hibernate Process

Recibe el ID del proceso a pausar.

Retorna 0 si hubo algún error ó 1 si se pausó el proceso.

## Get Inforeg Registers

Recibe un vector de `uint64_t *` donde se guardará el puntero al vector que contiene el valor de los registros. Dicho vector debe ser de `1 * SIZEOF(uint64_t *)`.

Retorna 1 si se han guardado los registros y 0 en caso contrario.