Arquitectura de Computadoras
Trabajo Práctico Especial
Informe

Integrantes:
Valentín Ruiz - 64046
Santiago Allegretti - 65531

Grupo 25 - 10 de junio de 2025

Objetivo

El objetivo de este trabajo fue implementar una parte del Kernel y la totalidad de Userland. El Kernel es booteable mediante el software-loader Pure64, el cual administra los recursos de Hardware de la computadora. También, se debe brindar una API por parte del Kernel que permita acceder a ciertas funcionalidades del mismo desde el User Space.

El Kernel interactúa directamente con el hardware mediante el uso de drivers e interrupciones, lo que facilita la comunicación con los periféricos, mientras que el User Space debe usar de intermediario al Kernel para comunicarse con estos dispositivos.

En el User Space, se creó una librería con ciertas funciones básicas, asemejándose en su implementación a la librería estándar de C en Linux. También, se implementó una consola de comandos similar a Shell que tiene ciertas funciones:

- Función para imprimir el manual
- Funciones que lanzan excepciones de división por cero y de código de operación inválido
- Función para imprimir la hora dependiendo la zona horaria
- Función para imprimir los valores de los registros
- Función para imprimir todos los argumentos que se le pasan
- Función que permite cambiar el tamaño de los caracteres
- Función que permite limpiar el buffer de pantalla
- Función para entrar al juego “Pongis Golf”
- Función de salida

Separación Kernel Space - User Space

El Kernel se encarga de administrar los recursos disponibles, además de brindarle al usuario herramientas para poder usarlos. El Kernel es el único que se comunica directamente con el hardware, y le brinda una API al usuario para que este pueda interactuar con el mismo.

La API consiste en múltiples System Calls, que se encuentran programadas en la IDT dentro del Kernel Space. Algunas de estas System Calls tienen un comportamiento similar a las de Linux, mientras que hay otras que se implementaron específicamente para darle al User Space medios para llevar a cabo las consignas solicitadas. Todas estas System Calls tienen un ID que las identifica, y reciben múltiples parámetros necesarios para llevar a cabo las acciones deseadas. Mediante el uso de estas System Calls, se pudo crear una librería estándar dentro del User Space, facilitando el uso del hardware desde fuera del Kernel.

System Calls

Se cargó la interrupción 80h en la IDT, la cual apunta a una rutina, la cual llama a una función que se encarga de determinar qué System Call es y llamar finalmente a su función correspondiente. En la siguiente tabla se pueden observar las System Calls que fueron implementadas:

Nombre	id	rbx	rcx	rdx
sys_read	0	uint64_t fd	uint8_t* buffer	uint64_t count
sys_write	4	uint64_t fd	uint8_t* buffer	uint64_t count
sys_ioctl	16	uint64_t cmd	-	uint64_t arg
sys_nanosleep	35	-	-	uint64_t ms
sys_exit	60	uint64_t exit_code	-	-
sys_time	96	-	-	int timezone
sys_clear_screen	100	-	-	uint64_t color
sys_get_screen_info	101	-	-	-
sys_draw_figure	102	uint64_t id	uint64_t* info	uint64_t color
sys_beep	103	uint64_t frec	-	uint64_t ms
sys_draw_text	104	uint64_t* xy	uint8_t* buffer	uint64_t* format
sys_toggle	105	uint8_t state	-	-
sys_read_by_keys	106	uint8_t* str	uint8_t* buffer	uint64_t count
sys_get_registers	335	-	uint64_t* buffer	-

Estas System Calls funcionan de la siguiente manera:

- sys_read: recibe un file descriptor *fd*, un *buffer* y la cantidad de caracteres a leer *count*. Guarda en *buffer* a lo sumo *count* caracteres. Retorna la cantidad de caracteres leídos.
- sys_write: recibe un file descriptor *fd*, un *buffer* y la cantidad de caracteres a escribir *count*. Escribe en pantalla según el file descriptor a lo sumo *count* caracteres de *buffer*.
- sys_ioctl: recibe un comando *cmd* y un argumento *arg*. En nuestra implementación en particular, el único comando válido es 0x4B01 y el argumento de 0 a 2. Con ese argumento, cambia el tamaño de la fuente.
- sys_nanosleep: recibe un tiempo *ms* expresado en milisegundos. Espera que transcurra esa cantidad de tiempo.

- sys_exit: recibe un valor para la salida *exit_code*. Finaliza la ejecución y retorna este valor.
- sys_time: recibe la zona horaria *timezone*. Retorna la hora del día en segundos.
- sys_clear_screen: recibe el color. Pinta la totalidad de la pantalla con ese color.
- sys_get_screen_info: retorna un uint64_t con el valor del ancho de pantalla en su parte alta, y el alto en su parte baja, ambos expresados en pixeles.
- sys_draw_figure: recibe un *id* que debe ser 0 para imprimir un rectángulo o 1 para imprimir un círculo, un arreglo *info* y el color. En el caso del rectángulo, el arreglo debe tener en su posición 0 el valor de x de la esquina superior izquierda, en su posición 1 el de y, en la posición 2 el ancho en píxeles y en la 3 el alto. En el caso del círculo, recibe lo mismo en las posiciones 0 y 1, representando la posición del centro del círculo, y en la posición 2 el radio del mismo, expresado también en píxeles.
- sys_beep: recibe un frecuencia *frec* y una duración *ms* expresada en milisegundos. Emite un sonido de frecuencia *frec* durante *ms* milisegundos.
- sys_draw_text: recibe un arreglo *xy*, un buffer y un arreglo *format*. El arreglo xy tiene el valor de x en la posición 0 y el de y en la posición 1. El arreglo format tiene el color de la letra en la posición 0 y el color del fondo en la posición 1. Imprime el texto de buffer en pantalla hasta encontrar un '\0'.
- sys_toggle: recibe un estado *state*. Los valores posibles de state son 0 y 1. En caso de ser 0, se apaga el cursor en pantalla, y lo enciende en caso de ser 1.
- sys_read_by_key: recibe un arreglo de chars *str*, un buffer y *count*. El arreglo str contiene las letras que se quieren filtrar, es decir, las únicas que se toman como válidas. El arreglo buffer se rellena con las teclas válidas que son presionadas en el teclado, en el orden en el que se presionaron. El parámetro *count* indica el tamaño del buffer.
- sys_get_registers: recibe un buffer. Se rellena el buffer con los valores de los registros en el orden rax, rbx, rcx, rdx, rbp, rsp, rsi, rdi, r8, r9, r10, r11, r12, r13, r14, r15, rip, rflags y cs.

  Siempre que se utiliza un color como parámetro, es con el formato 0x00RRGGBB.

Interrupciones

Las interrupciones que son tenidas en cuenta son el Timer Tick y la Keyboard Interrupt. Fueron cargadas en la IDT en las entradas 20h y 21h respectivamente, cada una con sus rutinas y funciones de atención. Se estableció la Master Mask del PIC en FCh, lo que permite que solo estas dos interrupciones lleguen al procesador.

Respecto al Timer Tick, se creó una variable en la que se acumula la cantidad de ticks que ocurrieron, lo cual fue útil para algunas funcionalidades. En cuanto a la interrupción de teclado, se creó un Keyboard Driver, el cual se encarga de capturar y procesar las teclas presionadas, que luego son utilizadas por otros módulos tanto dentro como fuera del Kernel.

Drivers

Keyboard Driver

Se implementó una matriz *scancodeToChar* la cual toma como fila la entrada del teclado y con 3 columnas, representando los valores “normales”, con el shift presionado y con el bloc. mayús. activado, en ese mismo orden. De esta forma, para conseguir el valor ASCII que debe ser representado, simplemente hay que acceder a la posición de la matriz que corresponda, de acuerdo a qué tecla se presionó y el estado del teclado.

Una vez que la tecla presionada fue transformada a ASCII, se analiza si se trata de alguna tecla especial o no. En caso de no serlo, se inserta en un buffer, donde se van guardando estos caracteres en orden y se imprime en pantalla en el lugar correspondiente. Si se tratara de un backspace (\b), se elimina el último carácter de pantalla y del buffer, y se retrocede el índice del mismo. Si la tecla presionada fuera un enter (\n), hace un salto de línea en pantalla y se cierra el buffer. Luego es el User Space quien se encarga de procesar el buffer y accionar al respecto.

Como se genera una interrupción tanto al presionar una tecla como al soltarla, se deben filtrar los casos de release. Para esto, se usa el valor 0x80 como máscara, ya que si el bit más significativo del scancode está encendido, significa que esa interrupción se generó al soltar una tecla. En estos casos, se analiza si esa tecla soltada fue alguno de los shifts y, en caso de serlo, se cambia el estado del teclado.

Video Driver

Se activó el modo video dentro del Bootloader, por lo que hubo que implementar una nueva forma para imprimir caracteres en pantalla, distinta a la que venía previamente implementada en naiveConsole.

Se utilizó ChatGPT y Claude.ai para la creación de arreglos que faciliten la impresión de caracteres en pantalla. Se creó un archivo fonts.h donde se declararon estos arreglos, uno para cada tamaño de fuente, siendo estos 8x16 y 12x24. A este tipo de arreglos se los conoce como bitmaps, donde cada elemento representa una línea de píxeles de determinada longitud. Cada bit de ese elemento representa si ese píxel correspondiente se debe pintar o no. La cantidad de bits de cada elemento de cada arreglo es igual al ancho de la font, mientras que la cantidad de elementos que representan un carácter es igual al alto en píxeles del mismo.

Se implementó también un scroll en el driver de video, que, al llegar al final de la pantalla, mueve los datos del frame buffer hacia atrás, generando que se mueva el contenido de la pantalla una línea hacia arriba.

También se implementaron otras funciones que permiten dibujar cosas en pantalla desde el User Space a través de System Calls. Entre estas se encuentran: print_text_in_screen, draw_rectangle, draw_circle y clear_screen. Además, se implementó una función extra, que también puede ser llamada indirectamente desde el User Space

mediante una System Call, que le devuelve al usuario las medidas de la pantalla, como ya se explicó anteriormente en el apartado de System Calls.

Sound Driver

Consta de la función play_sound, que se encarga de generar un sonido de una frecuencia y duración específicas mediante el altavoz de la computadora y el PIT. Primero, verifica si la frecuencia solicitada está dentro del rango 50 Hz - 20.000 Hz. Si no es así, utiliza un valor por defecto de 800 Hz para evitar valores inválidos. También, calcula el divisor necesario para el PIT a partir de la frecuencia estándar del temporizador y la frecuencia deseada.

La función lee el estado del puerto 61h, que corresponde al altavoz, para preservar su estado previo, y configura el PIT mediante los puertos 42h y 43h. Hace el pasaje de milisegundos a ticks del sistema, y espera el tiempo necesario teniendo las interrupciones activadas, deshabilitándolas una vez terminado.
Finalmente, restaura el estado anterior del puerto de control del altavoz, apagando el sonido.

Para lograr que el sonido funcionara correctamente en todos los dispositivos, se modificó *run.sh* de forma tal que reconociera el sistema operativo utilizado y configurara la variable *AUDIO_CONFIG* de acuerdo a este. La modificación de *run.sh* se realizó mediante el uso de ChatGPT y Claude.ai.

Excepciones

Se implementó una rutina de atención para cada excepción solicitada por la cátedra. En ambos casos se imprime un mensaje por pantalla que indica qué error se produjo e imprime los valores de los registros en pantalla.

Para esto, se decidió implementar una rutina que recibe como parámetro un buffer y devuelve dentro del mismo los valores de los registros, en el mismo orden que la System Call mencionada previamente. En ambos casos, los valores que se obtienen corresponden al contexto del User Space.

Se implementó una función *exceptionDispatcher* que es llamada desde Assembler y ejecuta la función correspondiente del manejo de cada excepción. Dentro del archivo asm, se hizo una macro *handler,* similar a la ya existente de las interrupciones, pero ahora para excepciones. Finalmente, con todo esto, se cargaron en la IDT las posiciones 0h y 6h, que corresponden a las excepciones de división por cero y código de operación inválido, respectivamente.

Shell

La consola de comandos se implementó con un loop infinito que imprime un prompt y levanta caracteres del buffer de teclado y los guarda en un buffer propio del User Space, hasta encontrar un '\n'. Una vez pasa esto, se procede con el procesamiento del buffer.

Se llama a una función con el buffer como parámetro, y dentro de esta se separa el buffer en tokens, usando el espacio como separador. Una vez hecho esto, se compara el primer token con todos los elementos de un arreglo de strings, que contiene los nombres de todas las funciones que pueden llamarse desde la Shell. En caso de no coincidir con ningún string del arreglo, se imprime en pantalla que el comando no fue encontrado y queda nuevamente a la espera del ingreso de datos por teclado.

Si coincide con un string del arreglo, se usa ese mismo índice para acceder a un arreglo de funciones paralelo y se llama a la función correspondiente, pasándole como parámetro la cantidad de argumentos y un arreglo de strings con todos los tokens previamente mencionados.

Estas funciones hacen uso de diversas funciones de la librería estándar, además de System Calls, ya que necesitan interactuar con el hardware para obtener información o bien para interactuar con los periféricos.

Pongis Golf

Para la implementación de este juego, primero se creó una librería con funciones útiles para la creación y el correcto funcionamiento del juego, donde esencialmente se hacen System Calls, pero que facilitaron el posterior desarrollo.

Como en este juego los objetos son todos círculos, se creó una estructura que tiene todos los datos necesarios para representarlos, a modo de objeto. Mediante ChatGPT y Claude.ai se crearon arreglos paralelos de frecuencias y duraciones para utilizar como sonidos dentro del juego, tanto al momento del inicio como al hacer un hoyo.

Para abrir el juego se debe escribir el comando *pongis* en el intérprete de comandos y pasarle por parámetro 1 o 2 dependiendo de la cantidad de jugadores que quiera tener.

Una vez iniciado el juego se reproduce una melodía y se queda esperando a que el usuario presione WASD o IJKL para el movimiento de Player 1 y Player 2 respectivamente. Al colisionar con la pelota, la empujan en la dirección del movimiento del jugador. Cada vez que la pelota entra en el hoyo, el último jugador que la empujó suma un punto y se pasa al siguiente nivel, reposicionando tanto a los jugadores como a la pelota y el hoyo. Así, al finalizar los 9 niveles, se muestran los puntajes y un mensaje indicando el ganador. El juego puede terminar en cualquier momento presionando la barra espaciadora.