

INFORME

Trabajo Práctico Especial Segundo Cuatrimestre 2017 Arquitecturas de Computadoras

- **Integrantes del grupo:** Lóránt Mikolás (57347), Johnathan Katan Piñeiro (56653) y Matías Heimann(57503).
- **Fecha de vencimiento de la entrega:** 14 de noviembre a las 23:59hs.

Introducción

El objetivo de este informe es presentar las decisiones tomadas a lo largo de la implementación de un kernel que administra los recursos de hardware de una computadora y ponga en evidencia las características del modo protegido de intel.

También se mostrarán los resultados obtenidos y el diseño elegido. Por último las fuentes utilizadas para la búsqueda de información y soluciones a algunos de los problemas encontrados.

Plataforma

El trabajo fue construido utilizando como base el Pure64 (provisto por la cátedra). Luego utilizando tanto lenguaje c como assembler se añadieron diversos componentes, con interacción

Se debieron implementar tanto interrupciones de software como de hardware para el correcto funcionamiento del trabajo. De esta manera el usuario a través de funciones en *user space* puede acceder a funcionalidad del *kernel*.

Para más información sobre cómo utilizar la shell o los programas y comandos proveídos leer el manual de usuario entregado.

Requerimientos e Implementación

User Space

En primer lugar, en cuanto a al set de funciones definidos en user space (equivalente a la biblioteca estándar de C), se definieron las siguientes funciones:

- **printf** : se debe tener en cuenta que no puede recibir formatos para la impresión, como por ejemplo `%d`.
- **strlen**: simplemente cuenta la cantidad de caracteres de un string.
- **scanf**: al igual que printf, nuestra implementación de scanf no puede recibir formatos para el guardado de los datos leídos.
- **getChar**: es equivalente a la función `getchar()` de c.
- **putchar**: es equivalente a la función `putchar()` de c.
- **printInt**: sirve para imprimir enteros a la salida estándar.
- **countDigits**: cuenta los dígitos de un entero pasado como parámetro.
- **parseNumber**: transforma un número escrito en caracteres ASCII al double correspondiente.
- **isNumeric**: devuelve 1 si recibe un número y 0 si no.
- **clear**: limpia la pantalla.
- **strcmp**: compara dos strings.
- **startsWith**: verifica si el string en el primer parámetro comienza con el string pasado como segundo parámetro.

Se debe tener en cuenta que `putchar` como `getChar` llaman a funciones de assembler que generan interrupciones de software (int 80h).

En segundo lugar, hicimos una librería gráfica (con funciones de video) y una librería para hacer gráficos de funciones lineales o parabólicas. La librería gráfica incluye las siguientes funciones:

- **xres**: retorna la resolución horizontal de la pantalla.
- **yres**: retorna la resolución vertical de la pantalla.
- **paintPixelAt**: pinta un pixel de color blanco en las coordenadas indicadas.

Estas tres funciones a través de interrupciones de software también hace uso del driver de video que se encuentra en kernel space.

Luego para graficar las funciones lineales y parabólicas implementamos un programa que puede ser abierto desde la shell. El usuario puede agregar gráficas, ingresando los coeficientes del polinomio de grado máximo 2. La librería implementada para el programa cuenta con las funciones:

- **cuadratic**: que grafica el polinomio con los coeficientes correspondientes.
- **paintAxis**: dibuja los ejes x e y en la pantalla.

En tercer lugar, la shell fue programada en el main de user space. La shell imprime un encabezado con el nombre del programa y luego un *prompt* para ingreso de nuevos comandos en la línea inferior. Cuando no entran más líneas en la pantalla se desplazan automáticamente todas las líneas de información hacia arriba (elimina la línea más antigua de ser necesario). Los comandos proveídos por la shell son:

- **echo**: para imprimir por terminal un string.
- **clear**: limpia la pantalla y la pone en negro.
- **divide by cero**: genera la excepción de división por cero.
- **overflow**: genera la excepción de overflow.
- **invalid opcode**: genera la excepción de opcode inválido.
- **time**: muestra la información proveída por el real time clock. Presenta [la fecha actual y la hora actual](#).
- **help**: muestra los comandos disponibles.
- **graph**: ejecuta un programa para graficar funciones.

Si se ingresa un comando inválido se imprime el mensaje correspondiente.

Kernel Space

En cuarto lugar, con lo que respecta a las excepciones manejadas por el *kernel*, se implementaron tres: división por cero, overflow y código de operación inválido.

Se debe tener en cuenta que se provee de comandos para generar dichas excepciones desde user-space. Dichos comandos pueden ser consultados en el manual de usuario entregado.

Para un manejo prolijo de la información generada por las excepciones, se decidió implementar una estructura en C que represente el stackframe al momento de provocar una excepción. De esta manera, en los handlers de las excepciones, se inicializa una variable

de tipo puntero a esta estructura, y se apunta al registro RSP. Esto permite un acceso a la información del error que se encuentra en el stack de manera más clara y prolija, usando los campos de la estructura.

Es importante aclarar el inconveniente que se tuvo al provocar una excepción de overflow. Se intentó generar esta excepción realizando una división válida donde el resultado es más grande que el registro donde se va a guardar. Al provocar la excepción de overflow, en todos los casos se ejecutaba el handler de la excepción de división por cero. Al verificar que la implementación de las excepciones y de los handlers era correcta, se consultó a la cátedra acerca de este problema y aclararon que estaban al tanto ya que tampoco pudieron generar esta excepción. Es por esta razón que cuando se intenta generar una excepción de overflow, se ejecuta el handler de la excepción de división por cero.

En quinto lugar en lo que respecta al *kernel space* algunas de los agregados al template de la cátedra son: **driver de video**, **driver de rtc**, **driver de teclado**, **dispatcher de syscalls**, **driver de texto** e **irq dispatcher**, entre otros.

En cuanto al driver de texto, vale la pena mencionar que si bien se configuró al sistema para que bootee en modo video, el desarrollo del proyecto comenzó en modo texto. Por lo tanto, se hicieron un gran número de funciones para la impresión de texto y manejo de la salida estándar en modo texto que luego no fueron usadas.

El driver de teclado, además de caracteres alfanuméricos soporta la utilización de la tecla *shift* izquierda y derecha para el uso de mayúsculas o caracteres como el paréntesis. Para esto se utilizaron dos tablas distintas que recibían el *scanCode* y retornaban el carácter correspondiente. Además tiene un *buffer* de 128 chars.

En cuanto al modo video, se debe aclarar que funciona más lento que el modo texto. Incluye funcionalidad para: dibujar un pixel, limpiar la pantalla, y soporta la impresión de distintos tipos de caracteres (enteros, números en hexa, etc.). Para la impresión de las letras se utilizó el *font bitmap* proveído por la cátedra. Por último para acceder a la información del modo video se implementó una estructura que accedía a los diferentes campos definidos en el archivo *sysvar.asm* (como por ejemplo resolución horizontal) .

Para finalizar, en el *irq dispatcher* se soporta la *int 20h* y la *int 21h*, correspondientes al driver del timer y el teclado respectivamente.

Syscalls de User space a Kernel Space

Las syscalls implementadas son:

- **write**: escribe en entrada estándar.
- **read**: lee de entrada estándar.
- **clearScreen**: limpia la pantalla.
- **paintPixel**: pinta un píxel en las coordenadas indicadas.
- **getResolutionY**: devuelve la resolución vertical.
- **getResolutionX**: devuelve la resolución horizontal.
- **printRTCInfo**: imprime la información del *real time clock*.

Para más información acerca del pasaje de parámetros consultar el manual de usuario.

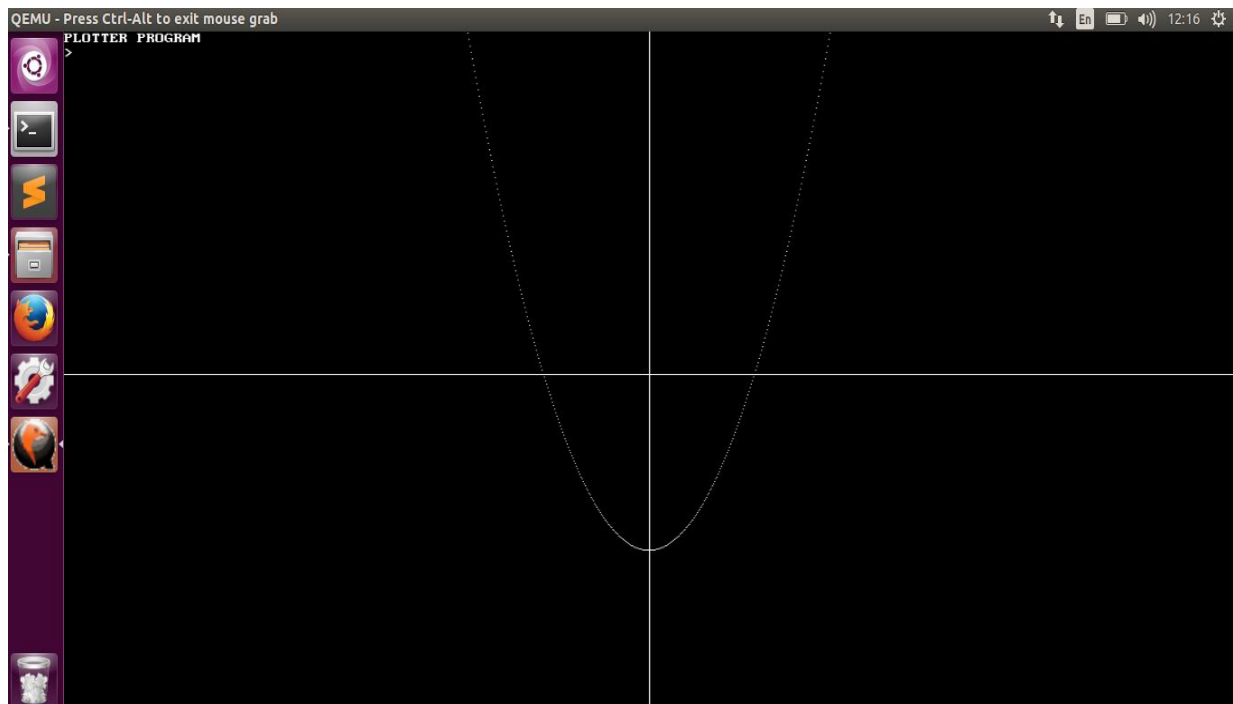
Resultados

Antes que nada vale la pena mencionar que a lo largo del desarrollo el trabajo fue probado sobre QEMU para acelerar las diferentes pruebas.

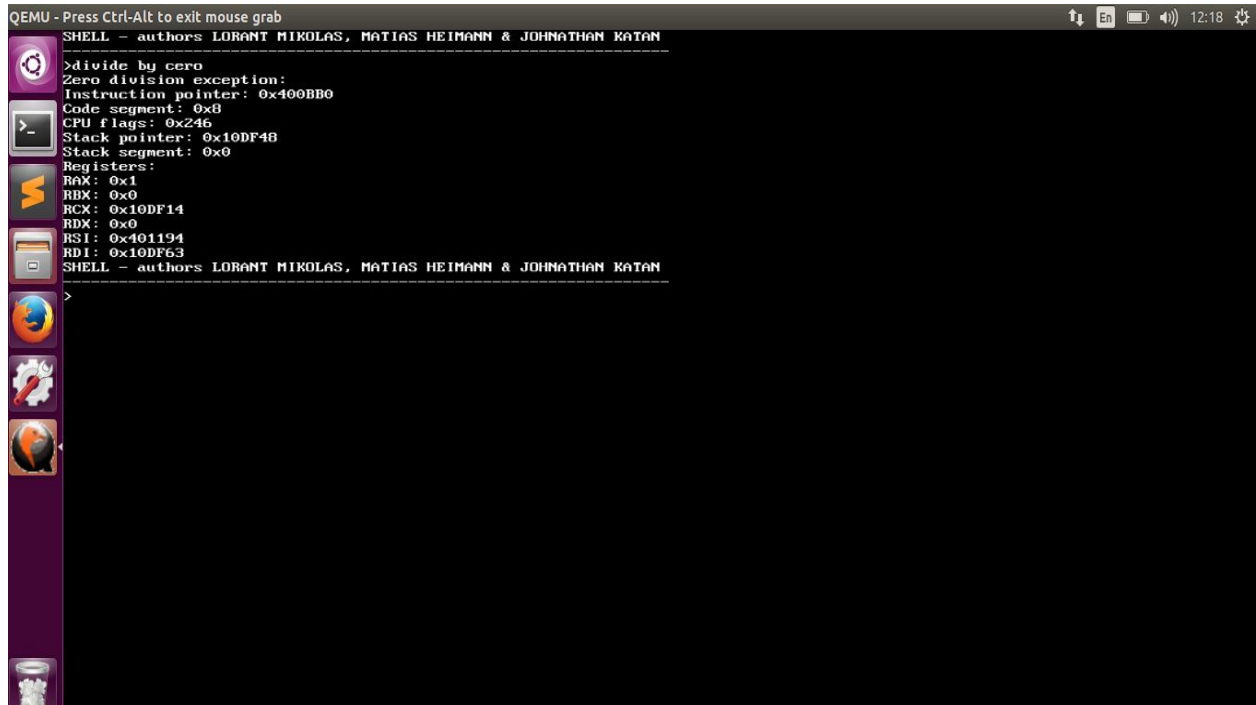
Debido a que no se contó con la computadora real en el laboratorio C, el trabajo no pudo ser preparado para funcionar en la máquina requerida. Sin embargo, el tp compila sin errores ni *warnings* y además funciona correctamente sobre QEMU.

El lunes 13 de noviembre el trabajo fue probado en una computadora proveída por la cátedra obteniendo resultados inesperados y erróneos al bootear en modo video.

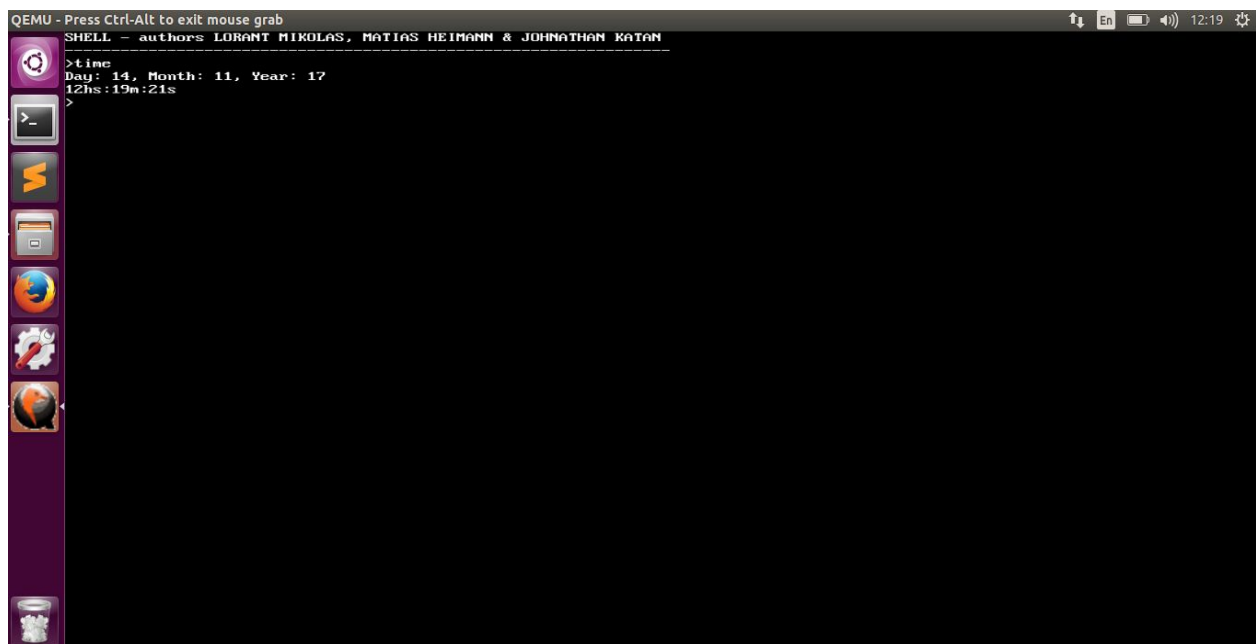
A continuación se presentan imágenes mostrando algunas de las funcionalidades del trabajo práctico:



Ejemplo del programa para graficar funciones



Ejemplo de la excepción de división por cero



Ejemplo del comando time

Conclusiones

En conclusión, este trabajo presenta una marcada división entre Kernel Space y User Space, y cómo estos interactúan entre sí mediante system calls. Algunas de las interacciones que se pueden apreciar en el trabajo se realizan cuando se hace uso del teclado, el RTC o el modo video en la pantalla.

También, la implementación de system calls de manera organizada permitió desarrollar correctamente distintas librerías en user space (como la librería gráfica o la librería estándar), que a su vez permitieron que el desarrollo de los programas de user space (como la shell o la graficadora) fuera más prolijo, claro y cómodo.

Por último, el desarrollo de este trabajo fue una excelente oportunidad para aplicar todos los conocimientos adquiridos en la teoría, y entenderlos de forma práctica. Por ejemplo, el uso del idt para cargar las funciones, el uso de la irq y el uso de interrupciones tanto para el teclado o para el timer tick.

Anexo

Hipervínculos de interés y referencias

- Base del trabajo práctico:
<https://bitbucket.org/RowDaBoat/x64barebones/wiki/Home>
- Referencia para driver de teclado:
<http://www.osdever.net/bkerndev/Docs/keyboard.htm>
- Referencia para manejo de excepciones:
<https://os.phil-opp.com/better-exception-messages/>
- Referencia para el manejo de la estructura con la información del modo video:
<http://www.delorie.com/djgpp/doc/ug/graphics/vesa.html>
- Referencia para el driver del real time clock:
<http://wiki.osdev.org/CMOS>