Sistemas Operativos

Primer Cuatrimestre 2016

Trabajo Práctico II

Autores:

Jeremías Aagaard - 53219

Martín Leon Nagelberg - 56698

Daniel Lobo - 51171

Resumen

Construcción del Núcleo de un Sistema Operativo

11 de Julio de 2016

Índice

1. Introducción

El presente trabajo práctico fue comenzado a partir de un versión funcional del trabajo práctico de Arquitectura de Computadoras, el cual consta de un kernel monolítico de 64 bits con las siguientes funcionalidades:

- Manejo de interrupciones básico con system calls.
- Driver de teclado.
- Driver de video en modo texto.
- Binarios de Kernel Space y User Space.

2. System calls

El sistema provee system calls para que los procesos puedan interactuar con el Kernel. Aquí presentamos la lista:

- **sys read**: lee de un file descriptor.
- **sys write**: escribe a un file descriptor.
- **sys malloc**: devuelve un puntero a memoria libre que se encuentra dentro de la página asignada a cada proceso.
- **sys free**: libera memoria.
- **sys cls**: limpia la pantalla.
- **sys beep**: emite un sonido de beep. Se hace mediante la creación de un proceso 'beep' en kernel land.
- **sys play note**: permite utilizar el PCSpeaker para emitir sonidos a determinada frecuencia.
- **sys sleep**: duerme un proceso por una determinada cantidad de tiempo.
- **sys create process**: crea un proceso a partir de un puntero a función y lo encola en el scheduler. Devuelve el pid del mismo.
- **sys kill**: mata a un determinado proceso.
- **sys wait pid**: duerme al proceso que la invoca hasta que termina su hijo.
- **sys get pid**: devuelve el pid del proceso.
- **sys list ps**: imprime en pantalla la lista de procesos corriendo en el momento.
- **sys list ipcs**: imprime en pantalla la lista de ipcs abiertos en el momento.

- **sys mute**: silencia el PCSpeaker.
- **sys draw frect**: carga en el buffer auxiliar de video los pixels necesarios para formar un rectángulo relleno.
- **sys draw char**: carga en el buffer auxilia de video los pixels necesarios para formar un caracter.
- **sys draw circle**: carga en el buffer auxiliar de video los pixels necesarios para formar un círculo relleno.
- **sys draw rect**: carga en el buffer auxiliar de video los pixels necesarios para formar un rectángulo sin relleno.
- **sys flush**: copia al puntero de video vesa todo lo que se encuentra en el buffer auxiliar de video.
- **sys opipe**: abre un pipe.
- **sys cpipe**: cierra un pipe.
- **sys wpipe**: escribe en un pipe.
- **sys rpipe**: lee de un pipe.

3. Physical Memory Manager

El kernel cuenta con un bitmap en donde cada bit de éste representa una página física de 4 KiB. A su vez, esta página, representada mediante una estructura, contiene un bitmap que permite dividirla en bloques de 8B para poder allocar memoria a la disposición del programador.

Se recomienda ver el archivo *memmanager.c* de la carpeta *drivers* del Kernel Space.

4. Procesos, Context Switching y Scheduling

El sistema cuenta con multitasking preeemptivo en una cantidad variable de procesos. Para esto se implementó un mecanismo que permite suspender la ejecución de un proceso y continuar la ejecución de otro.

El context switching implementado en este trabajo práctico es externo a los procesos en sí, esto significa que ellos actúan independientemente al mecanismo que se utilizó para ello. El algoritmo de selección utilizado es el Round-Robin; es decir, no contamos con ningún tipo de prioridad a la hora de darle el cpu a un proceso. El archivo *scheduler.c* se encuentra en la carpeta drivers del Kernel Space. Los procesos en el Scheduler de este sistema operativo tienen los siguientes estados: dormido, activo, pausado, en espera y desconocido.

Nunca hay más de un proceso ejecutándose al mismo tiempo. Todo proceso que se encuentra en estado *dormido* es ignorado por el Scheduler. Los procesos se encuentran en estado "pausado" esperando a que el scheduler los seleccione para darles el procesador.

Cada proceso tiene los siguientes atributos:

- Un nombre de longitud de 128 char.
- PID
- PID del padre.
- Estado.
- Tiempo dormido.
- Una página donde se encuentran el heap y el stack.
- Un stack propio de tamaño fijo.
- Un stack de Kernel
- Un punto de entrada.
- Una página del stack de usuario.
- Una página del stack de Kernel.

Sabemos que cada proceso debería tener su propio árbol, para que cada uno piense que tiene disponible toda la memoria del sistema.

Además, se tomó como referencia lo implementado en la parte de scheduling de Wyrm, que es un microkernel de sistema operativo.

5. Memory Protection

Utilizando el *page allocator* desarrollado y el sistema de de paginación de 64 bits de Intel, se implementó *"identity mapping"*.

Cabe destacar que se aplicó un identity mapping de 4GB porque la dirección de video que nos daba el driver de VESA era aproximadamente de 3GB y medio.

6. IPCs

Se tomó la decisión de usar pipes para la comunicación entre procesos. La implementación de los mismos se puede encontrar en la carpeta *Kernel/drivers*, en los archivos *pipe.c* y *semaphore.c*

Se consideró la implementación de semáforos y pipes mostrados en ejemplos de la cátedra anterior y también las ideas de este tutorial. Para probar su funcionamiento se puede ejecutar la aplicación de usuario *ipcs* que se encuentra implementada en la carpeta *Userland/game*.

7. Driver de Sonido

Se utilizó el driver de PC Speaker implementado en el trabajo práctico anterior.

Se puede encontrar el driver de audio en la carpeta *Kernel/drivers*, en el archivo *audio.c*

8. Driver de Teclado

Se utilizó el driver de teclado implementado en el trabajo práctico anterior.

La implementación se puede encontrar en la carpeta *Kernel/drivers*, en el archivo *keyboard.c*

9. Driver de Video-Texto

Si bien originalmente se había considerado usar el driver de video implementado en el trabajo práctico anterior, luego de investigar y no poder agregar la posibilidad de desactivar el modo video XVGA una vez puesto, decidimos migrar los drivers de video-texto a VESA.

10. Driver de Video-Imagen

Se implementó un driver VBE2 que soporta una resolución de 1024x768 pixeles con colores de 32 bits, siguiendo el estándar VESA. Además para todas las funciones de video, se utilizó una técnica de double buffering para optimizar. Se tomaron como referencia los siguientes links:

- https://goo.gl/vIhNx1
- http://wiki.osdev.org/Bochs VBE Extensions

La imeplemantación se puede encontrar en la carpeta *Kernel/drivers*, en el archivo *vesa-video.c*

11. Aplicaciones de User Space

Con el objetivo de mostrar las funcionalidades implementadas, en user space se cuenta con las aplicaciones listadas a continuación.

- **ps**: muestra la lista de procesos con sus propiedades, PID, nombre y estado.
- **ipcs**: muestra la lista de estructuras de IPC creadas en el sistema.
- **help**: muestra una lista con todos los comandos disponibles.
- **game**: corre una versión del famoso juego Snake.
- **beep**: hace un ruido sonoro con el objetivo de testear si funciona el Driver de Sonido.
- **clear**: limpia la pantalla.
- **test ipc**: corre dos procesos que se intercomunican mediante pipes.

12. Juego implementado: Snake

Se implementó el juego Snake, el cuál puede jugarse con el comando *game*. El mismo es una aplicación de usuario, por lo que puede encontrarse en la carpeta *Userland/game*.

Con las teclas **W, S, D, A** del teclado se pueden obtener los movimientos de **arriba, abajo, derecha, izquierda** respectivamente y con la **Q** salir del mismo. Si bien intentamos optimizar el video al máximo, como por ejemplo utilizando double buffering o flusheando al puntero de video de a 64 bits, podemos notar la baja cantidad de fps por el alto grado de parpadeo. No se han tenido en cuenta las colisiones con la propia serpiente, sino únicamente contra los bordes. Para ganar tenemos que lograr capturar 10 puntos.

13. Instrucciones para correr el sistema operativo

13.1. Preparar el entorno

Se recomienda tener los siguientes paquetes instalados antes de compilar el *Toolchain* y el *Kernel*:

nasm qemu gcc make

13.2. Compilar el Toolchain

Correr los siguientes comandos en la raíz del proyecto.

cd Toolchain
make all

13.3. Compilar el Kernel

Correr el siguiente comando en la raíz del proyecto.

make all

13.4. Correr el Kernel

Correr el siguiente comando en la raíz del proyecto.

./run.sh