
Arquitectura de las Computadoras

Trabajo Practico Especial

Primer Cuatrimestre 2015

# Kernel Development

Autores:

*Braulio Sespede* - 51074

*Daniel Lobo* - 51171

*Mauricio Minestrelli* - 52015

**Resumen**

El objetivo del trabajo practico fue implementar un kernel que administre los recursos de hardware de una computadora y muestre caracteristicas del modo protegido de Intel.

10 de junio de 2015

# Índice

# 1. Introduccion

El trabajo practico especial consistio en la realizacion de un sistema booteable por *Pure64* para una computadora de una arquitectura Intel de 64 bits en Long Mode, que muestra algunas de las caracteristicas del modo protegido disponible en los microprocesadores Intel, y que utiliza recursos de hardware. El sistema implementado provee al usuario una shell que le permite ejecutar un conjunto reducido de comandos.

Una de las caracteristicas de este kernel es que administra los recursos de hardware y ademas provee una API para que aplicaciones de usuarios puedan utilizar estos recursos.

El presente informe tiene como finalidad explicar algunas de las decisiones tomadas en la realizacion del sistema booteable con respecto al disenio y funcionalidad. Ademas, se mencionaran problemas encontrados en el desarrollo del mismo. El informe esta dividido en dos grandes areas: espacio de usuario o *Userland* y el espacio de kernel o *Kernel Space*. El espacio de kernel interactua directamente con el hardware mediante drivers, mientras al mismo tiempo provee funciones al user space. Vale aclarar que deliberadamente se intento evitar que el espacio de usuario acceda directamente al hardware. Solo lo hace a traves del *Kernel Space*. En cambio, en *Userland* solo se puede acceder a dicho hardware mediante system calls (y unicamente a aquellas funcionalidades que el kernel le permita).

## 2. Kernel Space

Para hacer una separacion entre *User Space* y *Kernel Space*, se decidio implementar una interrupcion basada en la int 80h de Linux, de modo que ninguna funcion posea acceso directo y conocimiento de implementaciones especificas de hardware, direcciones de memoria, y variables de bajo nivel dependientes de la arquitectura.

El Kernel busca estar lo mas aislado posible de los programas de usuario. Para esto se realizo toda la interaccion entre ellos a traves de system calls.

La idea detras del disenio implementado fue la de observar al kernel como un gestor de hardware. Para interactuar con dicho hardware se modifico la IDT para que el kernel pueda ser quien atiende las rutinas de atencion de interrupciones de los distintos componentes.

Para esto se dividio su funcion en modulos basicos que proveen distintas funcionalidades.

Estos modulos intentan ser independientes uno de otro y finalmente se encuentran vinculados por el modulo principal del kernel *(kernel.c)* que gestiona el arranque, inicializa los dispositivos y luego delega la ejecucion lo que seria, dentro de un esquema verdaderamente dividido, la aplicacion de usuario.

# 3. Video

El driver de video funciona en modo texto y contiene las funciones que acceden directamente a la zona de memoria donde se encuentra la pantalla, y modifican el color y contenido a mostrar.

Se implemento un protector de pantalla que se activa despues de tiempo configurable determinado que por defecto es 30 segundos. La implementacion del mismo se hizo a traves de un vector de pantallas el cual nos permite alternar entre la terminal y tantas pantallas como se quieran para animar el protector.

Este driver cuenta con funciones para inicializar, formatear, scrollear y limpiar la pantalla, ademas de las relacionadas con el protector de pantalla antes mencionado.

# 4. Teclado

La rutina de atencion de interrupcion de teclado provee la funcionalidad de mapear el scancode recibido con los caracteres ascii correspondientes, teniendo en cuenta teclas especiales como lo son *Shift* y *CapsLock* las cuales activan flags de estado que sirven para obtener los caracteres correspondientes a cada estado.

El objetivo basico de este driver es colocar los caracteres traducidos en un buffer de teclado, el cual sera utilizado como entrada estandar para nuestra terminal basica.

Es importante aclarar que hubo algunas dificultades con algunos elementos como es la tecla shift, la cual funciona como un âcaps lockâ de caracteres especiales, y el ascii 6, el cual funciona como si el shift estuviera presionado.

# 5. Real Time Clock

El RTC se encarga de proveer las funcionalidades de manejo del reloj del sistema. Dicha informacion es tomada o modificada a traves de los puertos 0x70 y 0x71. Es importante aclarar que una vez inicializado el kernel se realiza un ajuste horario por unica vez para que se corresponda con el huso horario de Buenos Aires.

# 6. PIT

El PIT o programmable interval timer se encarga de realizar 18 interrupciones por segundo, el cual es responsable de manejar el tiempo entre activaciones del protector de pantalla.

# 7. API

Se definio una API con la cual el espacio de usuario accede a las funciones del kernel. Este acceso ha sido implementado a traves de la interrupcion de software 80h en diferentes funciones denominadas system calls.

Una vez que el Kernel recibe dicha interrupcion, procede a analizar de que system call se trata en la rutina de atencion de interrupciones de software. Finalmente, ejecuta la funcion deseada y retorna un valor correspondiente en caso de que fuese necesario.

En una capa superior, entre las system calls y el shell, se definio un set de funciones que utilizan esta API. Dichas funciones son un equivalente de las que provee la biblioteca estandar de entrada y salida de C, las cuales permiten tanto leer el buffer de teclado como escribir en pantalla.

## 7.1. Librerias del sistema

Se implementaron las librerias *stdio.h*, *ctype.h*, *stddef.h*, *stdlib.h* y *string.h* lo mas parecido posible a los estandares POSIX. Varias funciones fueron omitidas por ser consideradas de poca utilidad para el trabajo actual, pero tambien se incluyeron funciones con utiles que no necesariamente se usaron en otros lugares.

### 7.1.1. string.h

Las implementaciones de string.h responden en general al estandar POSIX y funcionan tal como uno esperaria que funcionen en cualquier sistema Unix. La unica funcion no estandar es trim, que permite eliminar espacios en blanco al principio y al final de una cadena de caracteres. Tener en cuenta que la funcion cambia la cadena, no devuelve una nueva.

Se listan los headers a continuacion:

```
char * strcpy(char *dest, const char *src);
char * strncpy(char *dest, const char *src, uint64t n);
char * strcat(char *dest, const char *src);
uint64t strlen(const char *s);
int strcmp(const char *s1, const char *s2);
int strncmp(const char *s1, const char *s2, uint64t n);
char * strrev(char *str);
char * strtoupper(char *str);
char * strtolower(char *str);
char * strchr(const char *s, int c);
char * strrchr(const char *s, int c);
void* memcpy(void *dest, const void *src, uint64t n);
int memcmp(const void *dest, const void *src, uint64t n);
void * memset(void *s, int c, uint64t n);
```

```
void trim(char *str);
```

#### 7.1.2. ctype.h

Las funciones de *ctype.h* suelen implementarse como macros en varias librerias estandar. En nuestro caso por simplicidad, fueron implementadas como funciones, pero se las podria optimizar de mejor manera en caso de ser necesario. Otra posibilidad es tomar ventaja de las directivas para funciones inline del compilador, que no son estandar.

Se listan los headers a continuacion:

```
int isdigit(int c);
int isupper(int c);
int islower(int c);
int toupper(int c);
int tolower(int c);
int isalpha(int c);
int isalnum(int c);
int iscntrl(int c);
int isspace(int c);
int isprint(int c);
int isxdigit(int c);
int isvowel(int c);
```

#### 7.1.3. stdlib.h

Las semilla de srand se define inicialmente de manera est atica, pero se puede usar la funcion time para obtener una semilla aleatoria con la hora de sistema.

Se listan los headers a continuacion:

```
int rand(void);
void srand(unsigned int seed);
int atof(const char *nptr);
int atoi(const char *nptr);
long atol(const char *nptr);
char * itoa(int value, char *str, int base);
char * utoa(unsigned int value, char *str, int base);
int abs(int x);
long labs(long x);
```

#### 7.1.4. stdio.h

La implementacion realizada de esta libreria contiene las siguientes funciones. Las unicas funciones que no bloquean a la espera de datos son read y write, con lo cual se uso la instruccioon hlt del procesador para bloquear todas las

demas. Esto nos permitio tener la flexibilidad de hacer entrada de datos no bloqueando, llamando directamente a las primitivas.

Se listan los headers a continuacion:

```
int putc(int ch, FILE *stream);
int getc(FILE *stream);
int putchar(int ch);
int getchar(void);
int puts(const char *str);
char * gets(char *s, int size);
char * fgets(char *s, int size, FILE *stream);
int printf(const char *fmt, ...);
int sprintf(char *s, const char *fmt, ...);
int fprintf(FILE *f, const char *fmt, ...);
int vsprintf(char *str, const char *fmt, valist ap);
int vfprintf(FILE *f, const char *fmt, valist ap);
int write(int fd, void *buf, int count);
int read(int fd, void *buf, int count);
```

# 8. Shell

El shell o interprete de comandos cuenta con su propio buffer y almacena todos los caracteres que se van leyendo de la entrada estandar, mostrandolos a traves de la salida estandar de manera apropiada. El objetivo del shell es interpretar el contenido del buffer y ejecutar algunos de los comandos cuando el usuario lo decida.

Se decidio manejar el caso de buffer de shell lleno cortando la lectura de la entrada estandar. Se definio un tamanio de buffer de 256 caracteres.

Los comandos interpretados por nuestra shell son:

- help: muestra los distintos comandos disponibles.
- prtc: muestra en salida estÃ¡ndar el horario del sistem.
- srtc: cambia el horario del sistema (formato: srtc H M S).
- sstest: lanza el protector de pantallas.
- sstime: define la frecuencia del protector de pantalla en segundos.
- echo: repite los argumentos recibidos en consola.

# 9. Conclusion

Nos parecio muy util y prolijo separar las funcionalidades en espacio de usuario o *UserLand* y espacio de Kernel o *Kernel Space* e implementar la interrupcion de software que se transforma en una interfaz de comunicacion entre ambos espacios.

Considerar el correcto manejo de la system calls fue algo que desconociamos antes de este proyecto y ahora es un factor que consideramos fundamental al trabajar con funcionalidades de hardware o kernel mismo.

Finalmente, fue interesante explorar tutoriales y foros tales como *osdev.org*, los cuales nos fueron de mucha utilidad para tomar decisiones y para tener como referencia. Muchos de estos sitios eran desconocidos por nosotros antes de este trabajo practico y creemos que puede llegar a ser de utilidad en un futuro cuando cursemos materias correlativas a esta.

El trabajo practico especial permitio reforzar los conceptos aprendidos durante la materia y ganar una vision mas global acerca de los componentes involucrados en la creacion de un sistema booteable bajo la arquitectura Intel.