Departamento de Computación,
Facultad de Ciencias Exactas y Naturales,
Universidad de Buenos Aires

Trabajo Práctico 3

Organización del Computador II

Segundo Cuatrimestre de 2013

Grupo: **Crema Americana/Persico**

Apellido y Nombre	LU	E-mail
Silvio Vileriño	106/12	svilerino@gmail.com
Esteban Rey	657/10	estebanlucianorey@gmail.com
Matias Chapresto	201/12	matiaschapresto@gmail.com

Índice

1. Introduccion

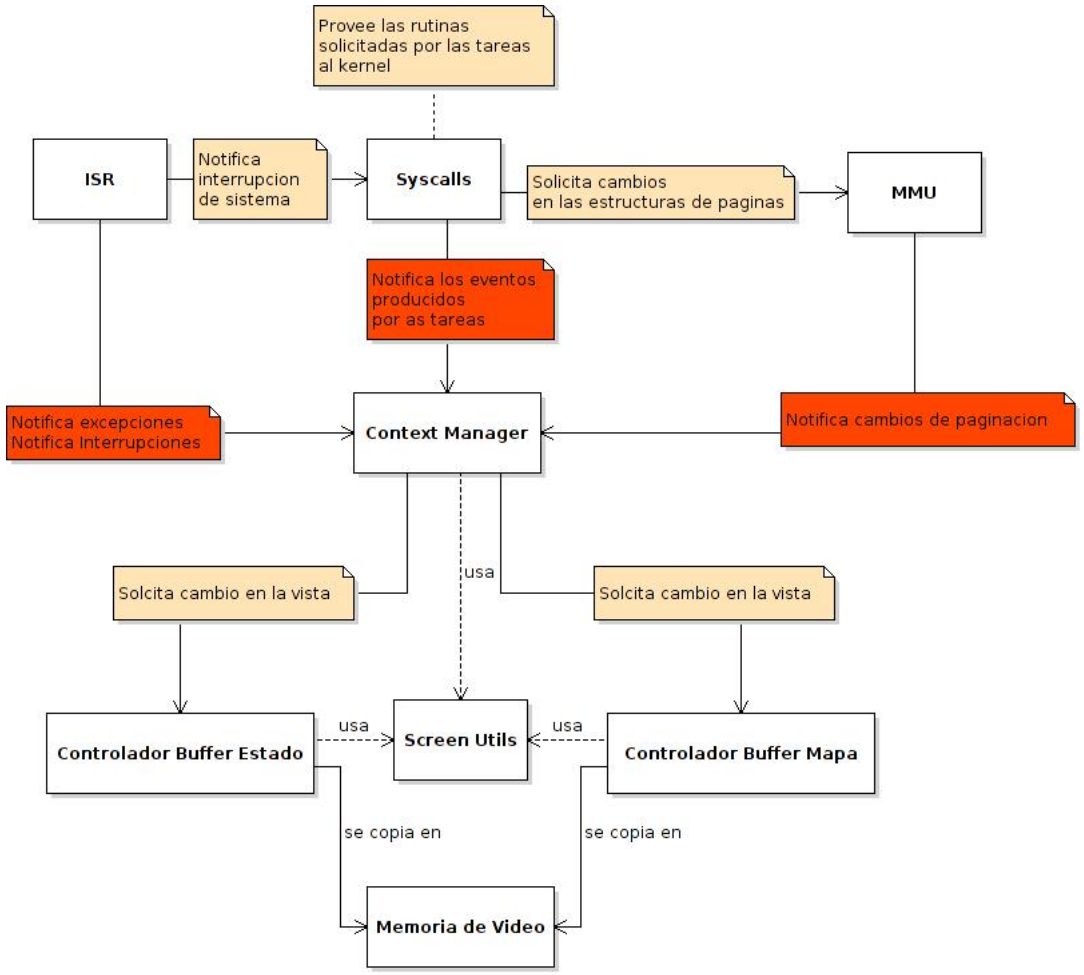
En el siguiente informe se describen las caracteristicas del TP entregado. Se separo en secciones con similitud a la secuencia de enunciados provistos por la catedra mas una seccion "Modularizacion de codigo"que indica como fue organizado el desarrollo por el grupo. La seccion Ïnterfaz con el usuario"describe el manejo de las pantallas mapa y estado y su interaccion con los demas modulos.

Para la realizacion del TP, al darse los temas para su desarrollo de forma gradual, quedo separado en distintas etapas: Pasaje a modo protegido configurando la GDT, establecimiento del sistema de paginacion, habilitacion las interrupciones, inicializacion de tareas y organizacion del scheduller. A todo esto se le suma la interfaz grafica, la cual requeria consumir de los datos obtenidos de las demas etapas. Como el desarrollo implico tocar constantemente todas las partes, decidimos escribir de la misma forma el informe, con excepcion de la interfaz grafica.

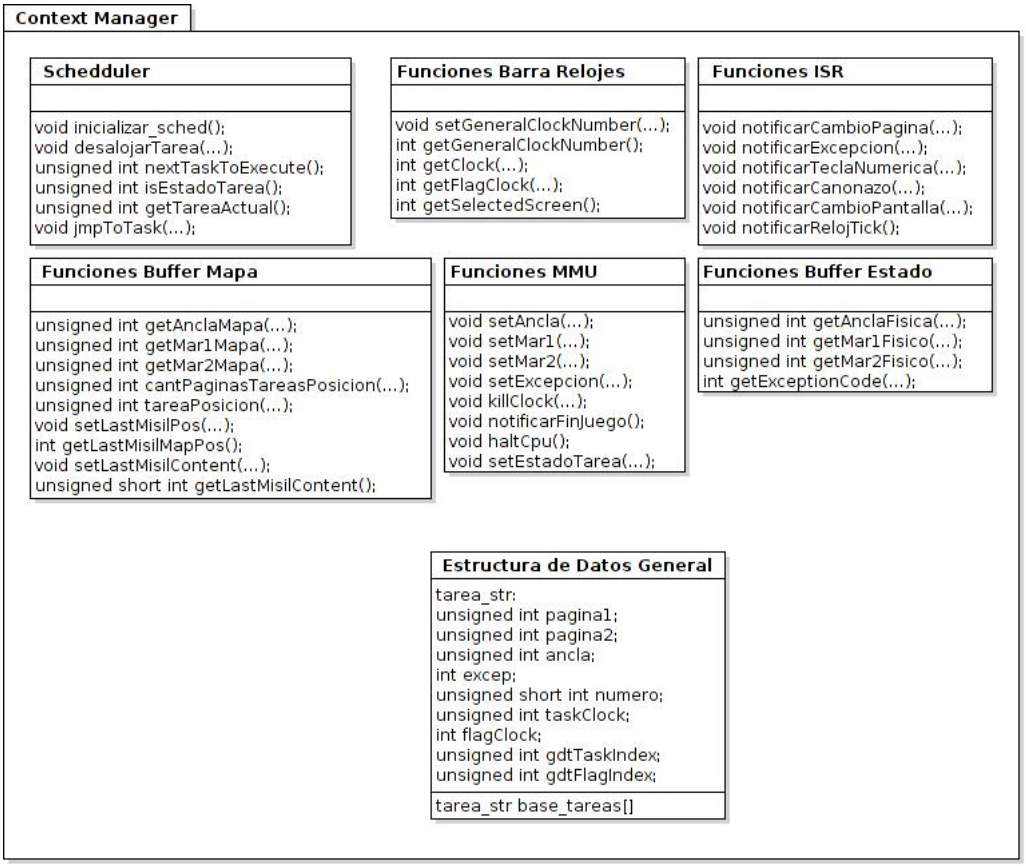
2. Desarrollo

2.1. Modularizacion del codigo

Como el codigo fue creciendo de a poco, notamos que, de equivocarnos en una seccion que era utilizada por otras, implicaba tocar todos los documentos una y otra vez, para resolver el error. Para solucionar esto optamos por aislar de alguna forma las secciones con una capa intermedia, la cual guardaria toda la informacion necesaria para el sistema y la expondria a cada seccion, solo el tipo de datos necesario. Por ejemplo, como tenemos 3 tipos de direcciones:fisica, virtual y de pantalla, centramos todas las conversiones entre estas tres en un modulo que llamamos Çontext Managerz cada seccion solo recibe el tipo de direccion que requiere, sin tener que incluir en el codigo las conversiones.



Context Manager y sus relaciones con las demas secciones. Imagen sin especificar las funciones internas



Funciones y estructuras del Context Manager

Siendo de esta forma un paso intermedio entre las partes, la secuencia de accion de cada funcionalidad del juego pasa por el mismo. En las siguientes secciones se mostrara como las distintas funcionalidades utilizan esta estructura.

2.2. Inicializacion y Segmentacion

Para iniciar el sistema se paso de modo real a modo protegido, para ello necesitamos establecer una GDT que nos delimite los distintos segmentos a usar por nuestro kernel.

Por enunciado se pedian 5 descriptores, 1 de video, codigo de nivel kernel, datos de nivel kernel, codigo de nivel usuario y datos de nivel usuario. Se definio por cada entrada un indice (en orden del 18 al 22) y un descriptor:

índice	Descripción	Tipo	Límite	Base	AVL	DPL	P	S	D/B	G	L
18	Level 0 Code	Execute-Only, accessed	0xFFFF	0	No	0	1	0	32 bits	4k	32 bits
19	Level 0 Data	Read/Write	0xFFFF	0	No	0	1	1	32 bits	4k	32 bits
20	Level 3 Code	Execute-Only, accessed	0xFFFF	0	Si	1	1	0	32 bits	4k	32 bits
21	Level 3 Data	Read/Write	0xFFFF	0	Si	1	1	1	32 bits	4k	32 bits
22	Video	Read/Write	0x0F9F	0x8	No	0	1	1	32 bits	1B	32 bits

Organizacion de la GDT: primeras entradas.

Una vez completa, se incluyo en el codigo del kernel la instruccion LGDT para cargar en el procesador la direccion de memoria de la misma. Paso seguido seteamos el bit 0 del registro CR0 en 1 para pasar al modo protegido y efectuamos un salto para hacer que el procesador se setee en el segmento de codigo de kernel, indice 18 de la GDT con la instruccion JMP 0x90:protected_mode. Se utiliza el valor 0x90 para saltar al segmento correspondiente ya que el indice del segmento al que necesitamos ir (codigo de kernel) es el 18_10. Como el RPL es 0 de kernel y estoy accediendo a una entrada de la GDT, los 3 primeros bits del selector son 0, equivalente de multiplicar por 8: 18_10 x 8_10 = 0x90

Despues seteamos los registros selectores a los segmentos correspondientes e inicializamos la pila del kernel en la posicion 0x27000, pasandole este valor al registro ESP y EBP.

```
1  BITS 16
2  start:
3      ; Deshabilitar interrupciones
4      cli
5
6      ; habilitar A20
7      call habilitar_A20
8
9      ;desaparecer cursor en pantalla
10     mov BL, 0
11     dec BL
12     mov BH, 0
13     dec BH
14     set_cursor
15
16     ; cargar la GDT
17     lgdt [GDT_DESC]
18
19     mov EAX, CR0
20     OR EAX, 1
21     mov CR0, EAX
22     ; pasar a modo protegido
23     jmp 0x90:protected_mode
24
25 BITS 32;modo de programacion en 32 bits(compila en 32 bits)
26 protected_mode:
27     ;cargo los selectores de segmento
28     xor eax, eax
29     mov ax, 10110000b
30     mov fs, ax
31     mov ax, 10011000b
32     mov ds, ax
33     mov es, ax
34     mov gs, ax
35     mov ss, ax
36     ;inicializo la pila
37     mov esp, 0x27000
38     mov ebp, esp
```

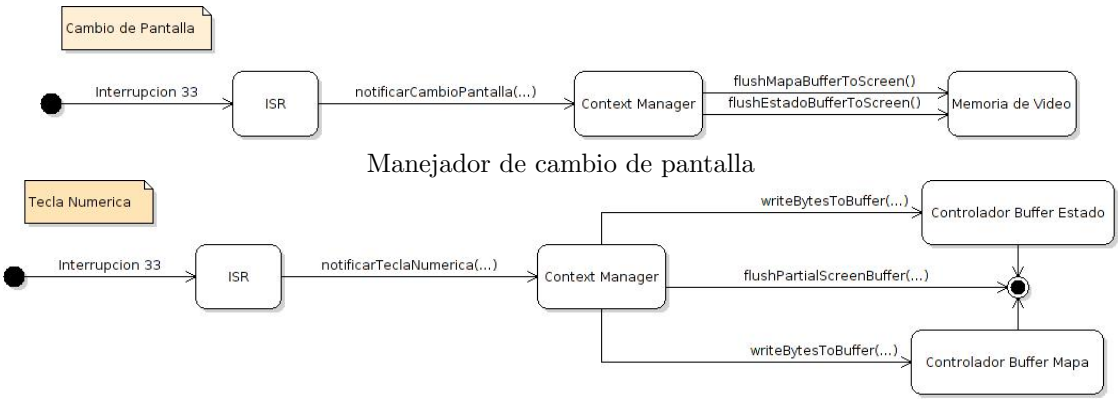
2.3. Descriptor de Interrupciones y Handlers

Previo a cualquier configuracion o funcion a llamar dentro de nuestro kernel, se llamo a la instruccion CLI, la cual nos deshabilito las interrupciones ("saltando a modo protegido"). Una vez deshabilitadas y en modo protegido, establecimos la tabla de interrupciones con la instruccion LIDT. A esta instruccion le pasamos la tabla ya configurada, con las interrupciones posibles: tanto las excepciones como las interrupciones. Cada entrada fue configurada para ser ejecutada en el segmento de codigo de kernel (selector 0x90), ya que el debe ser el responsable de ellas. Las rutinas de atencion, configuradas en el offset de cada descriptor de la IDT, notifican al context manager el evento producido, para que este actualice los datos en los distintos modulos: sea pantalla o mmu.

index	Descripción	Gate Descriptor			
		P	DPL	Tipo	D
0	Div x 0	1	00	Trap	1
2..8	Varios	1	00	Trap	1
10..14	Varios	1	00	Trap	1
16..19	Varios	1	00	Trap	1
32	Clock	1	00	Interrupt	1
33	Teclado	1	00	Interrupt	1
80	Syscall	1	11	Interrupt	1
102	Syscall bandera	1	11	Interrupt	1

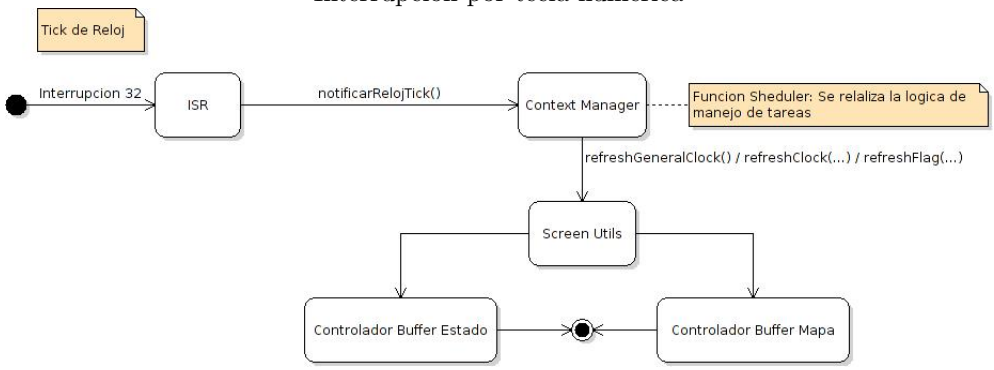
Descriptores de la IDT.

Los campos de offset se completaron con la direccion de los handlers de atencion a las interrupciones, en donde dependiendo del tipo, se recurre a distintos modulos para resolver el evento producido.

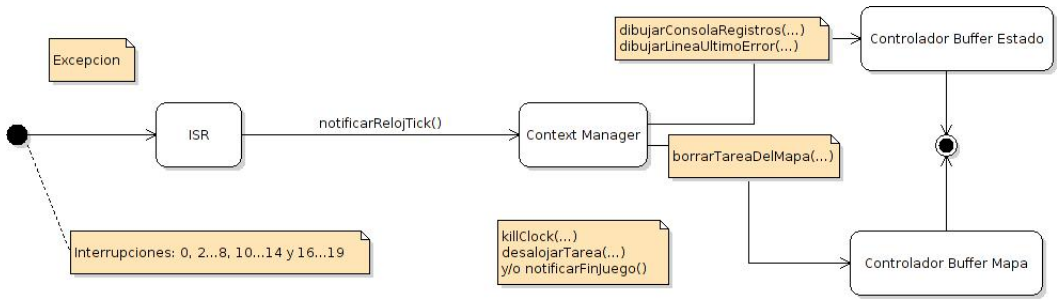


Manejador de cambio de pantalla

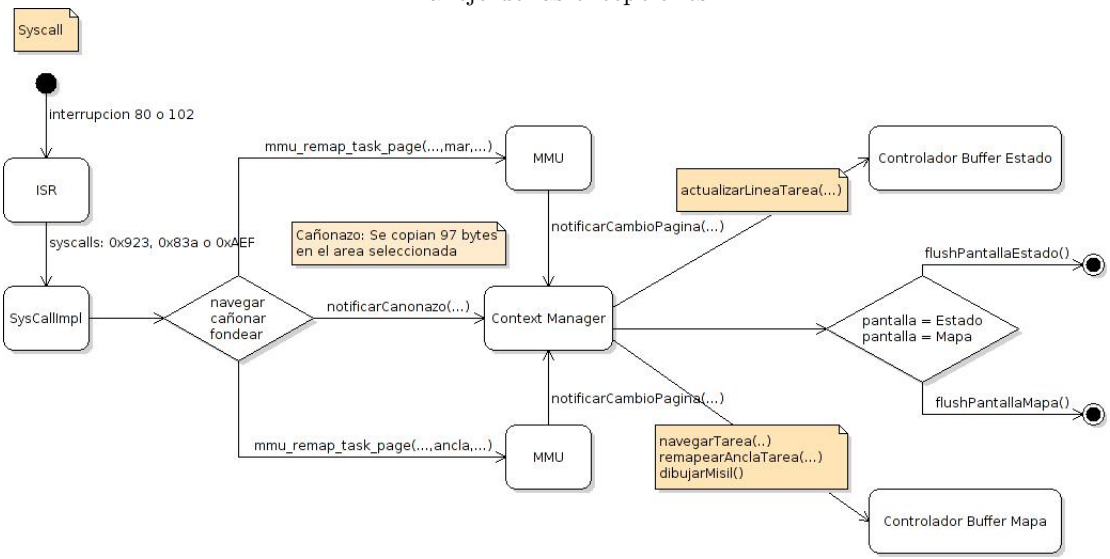
Interrupcion por tecla numérica



Interrupcion del clock



Manejo de las excepciónes



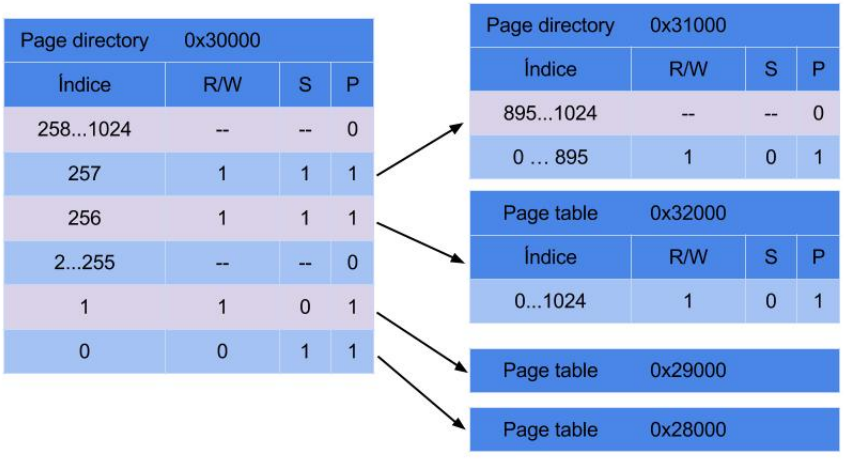
Llamadas al sistema

2.4. Paginacion

Para activar la paginacion es necesario que armemos el sistema de directorios y paginas del kernel. Como se requirio un identity mapping para el kernel, de los primeros 7.5Mb de memoria, utilizamos entradas en el directorio. La primer entrada posee todas las posiciones de la tabla inicializadas en Present, marcando asi los primeros 4Mb. La segunda entrada, sin embargo solo debe mapear 3.5Mb, con lo cual solo se necesitan 896 entradas mas marcadas como presentes.

Distribucion de paginas del kernel y posicion de los directorios.

Para el paginado de las tareas se requerian por enunciado 3 paginas: 2 para la tarea que debian mapearse en el mar (arriba del primer giga) y una tercera, el ancla, que debia apuntar al espacio de memoria del kernel (no modificable). La estructura del directorio y de las tablas se pusieron contiguas a las del kernel, en su area de memoria. Para ubicar las tareas a partir de la direccion virtual 0x40000000, se utilizo el indice 256 de sus directorios y cada tabla apuntando de a 4Mb: 0x40000000, 40001000 y 40002000. Tambien fue necesario inicializar 2 entradas en el directorio de cada tarea con identity mapping a la memoria del kernel, para que este pudiese ejecutarse en el contexto de la tarea. Estas entradas poseen acceso solo por el kernel y permiten la ejecucion de los handlers de interrupciones.



Mapeo generico de una tarea.

Como las tareas son capaces de "moverse"se puso a disposicion de la mmu funciones para remapear las paginas propias. Se paso de forma mas directa la informacion soblre las paginas al Context Manager para poder reflejar el movimiento de las paginas en la pantalla del mapa y en los indicadores de la pantalla de estado.

2.5. Tareas: Descriptores y contexto

Al saltar por primera vez a la tarea idle, nos aseguramos de tener inicializadas todas las TSS de las distintas tareas (2 por tarea, uno para la tarea en si y la otra para la bandera). La tarea inicial, desde donde se salta por primera vez a la idle, tambien debe tener un espacio pare el contexto de ejecucion, en donde el procesador guardara el estado del kernel para luego pasar a la ejecucion de tareas. La tarea idle debe ser mapeada sobre la memoria del kernel y compartir su CR3; ya que de no ser asi, y ubicarla en el mar con las demas, corremos el riesgo de que sea corrompida por otra tarea mediante un cañonazo, comprometendo la estabilidad del sistema.

Tarea	eip	esp 0	ssp 0	cr3	eflags	esp/ebp	cs	ss	ds	fs	gs	rpl
init	0x00000000	0x0	0x0	0x27000	0x00000202	0x0	0x0	0x0	0x0	0x0	0x0	kernel
idle	0x40000000	0x0	0x0	0x27000	0x00000202	0x2B000	GDT:Level 0 Code	GDT:Level 0 Data	GDT:Level 0 Data	GDT:Level 0 Data	GDT:Level 0 Data	kernel
task n	0x40000000	a partir de 0x00050000	GDT:Level 0 Data	0x30000 + offset tarea	0x00000202	0x40001C00	GDT:Level 3 Code	GDT:Level 3 Data	GDT:Level 3 Data	GDT:Level 3 Data	GDT:Level 3 Data	user
flag n	0x40000000	task esp 0 + 0x1000	GDT:Level 0 Data	0x30000 + offset	0x00000202	0x40001C00	GDT:Level 3 Code	GDT:Level 3 Data	GDT:Level 3 Data	GDT:Level 3 Data	GDT:Level 3 Data	user

Mapeo en memoria del contexto de ejecucion de la tarea Idle y de las tareas.

Para que el procesador ubique la TSS de la tarea, incluimos en la GDT, por cada una, 2 entradas (tarea y bandera). Estas entradas son Descriptores de tarea, y le seteamos el limite minimo para las TSS que es de 0x67.

índice	Descripción	Tipo	Límite	Base	AVL	DPL	P	S	D/B	G	L
23	init	Execute-Only, accessed	0x67	&TSS init	No	0	1	0	32 bits	4k	32 bits
24	idle	Execute-Only, accessed	0x67	&TSS idle	No	0	1	0	32 bits	4k	32 bits
25+2*n	task	Execute-Only, accessed	0x67	&TSS task	No	0	1	0	32 bits	4k	32 bits
25+2n+1	flag	Execute-Only, accessed	0x67	&TSS flag	No	0	1	0	32 bits	4k	32 bits

Valores de las entradas de la GDT para la tarea Idle, Tareas y Banderas.

2.6. Interfaz con el usuario

Las 2 pantallas del sistema muestran todos los datos utilizados por el mismo y los movimientos de las tareas por el mar y sus anclas por la tierra. Esta parte de la funcionalidad del kernel solo conoce sobre posiciones de mapa, caracterizada por la cantidad de memoria de cada buffer de pantalla. La adecuacion de las posiciones se llevan a cabo en el Context Manager, el cual traduce las direcciones fisicas y virtuales en datos para mostrar en los distintos sectores de la pantalla.

Las pantallas tienen en comun 2 secciones: el nombre de grupo en la parte superior, y la barra de relojes en la inferior. El nombre de grupo es seteado permanentemente al inicio del sistema, mientras que, por otro lado, los relojes son actualizados cada vez que una tarea corre.

Todos los elementos dinamicos de las pantallas fuenron pensados para actualizarse por eventos de forma individual, para no tener que copiar todo el buffer de cada pantalla, cada vez que un reloj se movia. Cada sector responde a un evento el cual es delegado a traves del Context Manager a las secciones de manejo de Buffers.

2.7. Schedduler

Para el manejo de los tiempos de las tareas, utilizamos las funcionalidades del Context Manager, incluidas en el las funcionalidades de un shedduler. Dentro del mismo se calcula la proxima tarea a ejecutar mediante una lista circular (el espiritu de una lista circular), la cual se resetea a la tarea idle cada vez que una tarea solicita una syscall (comportamiento segun el enunciado). Esta lista chequea cada vez que va a dar tiempo a una tarea, que la misma no posea una excepcion, las cuales de producirse, marcan en la estructura interna del Context Manager un flag permanente de error tanto para la tarea como para la bandera de la misma.