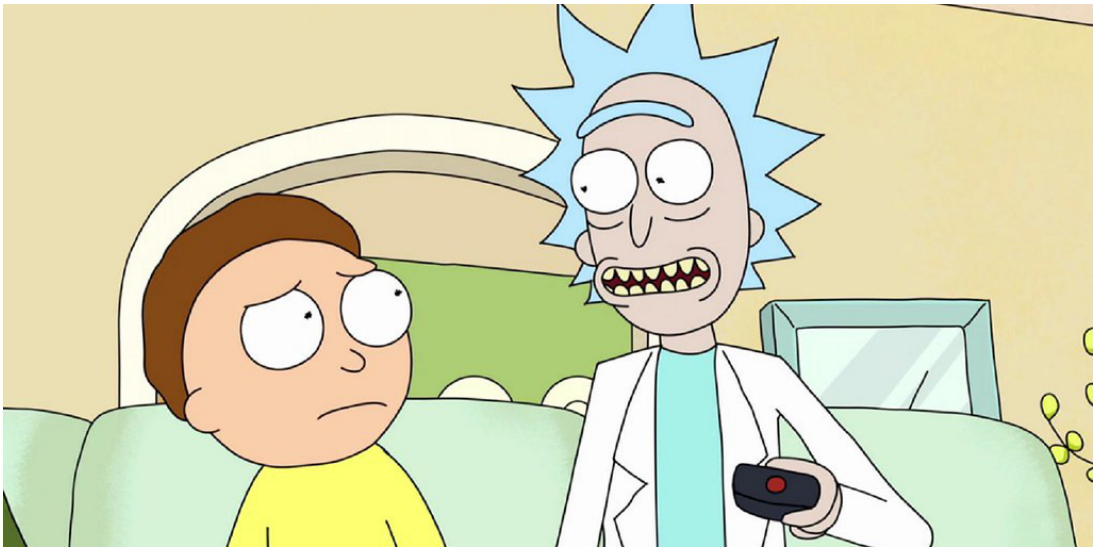
Trabajo Práctico 3

System Programming

Asimilación Cronenberg



0.1 Objetivo

Este trabajo práctico consiste en un conjunto de ejercicios en los que se aplican de forma gradual los conceptos de *System Programming* vistos en las clases teóricas y prácticas. Los ejercicios están inspirados en la serie *Rick y Morty*, siendo el sistema que desarrollarán un multiverso en "memoria", donde el Rick Sánchez de la dimensión C-137 trata de detener a su alter ego Rick de la dimensión D-248 para que no someta a los habitantes del mundo Cronenberg tomando sus mentes.

Se buscará construir un sistema mínimo que permita correr como máximo 25 tareas concurrentemente a nivel de usuario. Este sistema simulará un juego entre dos pares de tareas, `Rick` y `Morty`, sobre un mundo donde se ubicarán estas y otro conjunto de tareas, denominadas tareas Cronenberg (o habitantes Cronenberg). Las tareas serán ubicadas incialmente dentro del mundo de forma aleatoria. Una vez que comience el juego, las tareas pueden vivir o morir, pero una vez muertas no reviven más (suponemos que Rick no tiene más clones). Cada pares de jugadores Rick y Morty, tratará de tomar la mayor cantidad de mentes posible. El juego terminará cuando alguno de los Ricks tenga todas las mentes y el otro ninguna o alguno de los Ricks muera (ya sea porque su mente fue capturada o porque mataron a Rick). Además, como un Rick no puede estar sin su Morty (por el tema de la compensación), si su Morty llegará a estirar la pata, entonces el Rick en consecuencia también moriría. Los Ricks utilizarán su arma de portales, tanto para moverse por el mundo Cronenberg, como para capturar mentes.

Es importante recordar, que si bien este trabajo práctico esta ilustrado por un juego, los ejercicios proponen utilizar los mecanismos que posee el procesador para la programación desde el punto de vista del sistema operativo.

0.2 Introducción

Para este trabajo se utilizará como entorno de pruebas el programa *Bochs*. El mismo permite simular una computadora IBM-PC compatible desde el inicio, y realizar tareas de debugging. Todo el código provisto para la realización del presente trabajo está ideado para correr en *Bochs* de forma sencilla.

Una computadora al iniciar comienza con la ejecución del POST y el BIOS, el cual se encarga de reconocer el primer dispositivo de booteo. En este caso dispondremos de un *Floppy Disk* como dispositivo de booteo. En el primer sector de dicho *floppy*, se almacena el *boot-sector*. El BIOS se encarga de copiar a memoria 512 bytes del sector, a partir de la dirección `0x7C00`. Luego, se comienza a ejecutar el código a partir esta dirección. El boot-sector debe encontrar en el *floppy* el archivo `KERNEL.BIN` y copiarlo a memoria. Éste se copia a partir de la dirección `0x1200`, y luego se ejecuta a partir de esa misma dirección. En la figura 2 se presenta el mapa de organización de la memoria utilizada por el *kernel*.

Es importante tener en cuenta que el código del *boot-sector* se encarga exclusivamente de copiar el *kernel* y dar el control al mismo, es decir, no cambia el modo del procesador. El código del *boot-sector*, como así todo el esquema de trabajo para armar el *kernel* y correr tareas, es provisto por la cátedra.

Los archivos a utilizar como punto de partida para este trabajo práctico son los siguientes:

- `Makefile` - encargado de compilar y generar el *floppy disk*.
- `bochsrc` y `bochsdbg` - configuración para inicializar Bochs.
- `diskette.img` - la imagen del *floppy* que contiene el *boot-sector* preparado para cargar el *kernel*.
- `kernel.asm` - esquema básico del código para el *kernel*.
- `defines.h` y `colors.h` - constantes y definiciones.
- `gdt.h` y `gdt.c` - definición de la tabla de descriptores globales.
- `tss.h` y `tss.c` - definición de entradas de TSS.
- `idt.h` y `idt.c` - entradas para la IDT y funciones asociadas como `idt_init` para completar entradas en la IDT.
- `isr.h` y `isr.asm` - definiciones de las rutinas para atender interrupciones (*Interrupt Service Routines*).
- `sched.h` y `sched.c` - rutinas asociadas al *scheduler*.
- `mmu.h` y `mmu.c` - rutinas asociadas a la administración de memoria.
- `screen.h` y `screen.c` - rutinas para pintar la pantalla.
- `a20.asm` - rutinas para habilitar y deshabilitar A20.
- `print.mac` - macros útiles para imprimir por pantalla y transformar valores.
- `idle.asm` - código de la tarea `Idle`.
- `game.h` y `game.c` - implementación de los llamados al sistema y lógica del juego.
- `syscalls.h` - interfaz a utilizar en C para los llamados al sistema.
- `task.h` - función de posiciones "random" para tareas.
- `taskRickC137.c` y `taskMortyC137.c` - código de las tareas del jugador C-137.
- `taskRickD248.c` y `taskMortyD248.c` - código de las tareas del jugador D-248
- `taskCronenberg.c` - código de las tareas Cronenberg (son todas iguales).
- `i386.h` - funciones auxiliares para utilizar *assembly* desde C.
- `pic.c` y `pic.h` - funciones `pic_enable`, `pic_disable`, `pic_finish1` y `pic_reset`.

Todos los archivos provistos por la cátedra **pueden** y **deben** ser modificados. Los mismos sirven como guía del trabajo y están armados de esa forma, es decir, que antes de utilizar cualquier parte del código **deben** entenderla y modificarla para que cumpla con las especificaciones de su propia implementación. Tengan en cuenta que el código provisto por la catedra puede **no** funcionar en todos los casos, y depende de del desarrollo de cada uno de los trabajos prácticos.

A continuación se da paso al enunciado, se recomienda leerlo en su totalidad antes de comenzar con los ejercicios. El núcleo de los ejercicios será explicado en las clases, dejando cuestiones cosméticas y de informe para que realicen por su cuenta.

0.3 Historia

Rick C-137 miraba uno de los canales de la televisión interdimensional en el que se presentaba una publicidad de puertas falsas reales. Mientras tanto Morty a su lado, trataba de entender el funcionamiento de una caja de Mr. Meeseeks. Suena el timbre y Summer atiende, era una ex novia de Rick, Unidad. Ella le explica a Rick que otro Rick D-248 había robado la formula para controlar mentes de Unidad, y que estaba en un mundo Cronenberg tratando de controlar las mentes de todos. Sin entender muy bien porque, y solo con la intensión de ayudar a Unidad, Rick se embarca al mundo Cronenberg para detener a Rick D-248.

0.4 Juego

El juego consiste en dos Rick intentando controlar las mentes de muchos habitantes del mundo Cronenberg. Obviamente, donde va un Rick, hay un Morty. Así que no solo vamos a tener a un par de Ricks, sino también a sus Mortys. El mundo Cronenberg va a ser modelado por un rectangulo de 80x40 celdas. En cada celda podrá ser habitada por un Rick, un Morty o un Cronenberg. Los Cronenbergs no son muy astutos, así que una vez que están en una celda, no suelen moverse a ningún lado. Sin embargo, nuestro "héroe", Rick Sanchez, y su alter ego, tienen un arma de portales que pueden usar para desplazarse a cualquier lugar del mundo Cronenbergs, incluso, dentro de las mentes de los habitantes del mundo Cronenberg.

El arma de portales, permite crear, como su nombre lo indica, un portal. El que utilice el arma puede tanto decidir atravesar o no el portal, como también ser o no acompañado por Morty. Aunque el arma de portales es potestad de Rick, cada 10 aventuras, Morty puede usar el arma una vez.

La figura 1 muestra un ejemplo de como se presentará el juego en pantalla. Lamentablemente no tenemos control de una placa de vídeo y estamos programando en ASM y C, por lo tanto los gráficos resultantes serán un poco más rústicos que los presentados en la figura. No obstante, se presenta toda la información necesaria para el juego, la posición de los Ricks y Mortys en el mundo, la posición de todas los habitantes Cronenberg y los portales creados por los Ricks. Además, sobre el lado inferior se observan dos contadores y 20 marcadores. Los marcadores indican el estado de cada uno de los habitantes Cronenberg, sí están en azul, significa que el Rick azul tomo sus mentes, sí figuran en rojo, implica que el otro Rick tomo sus mentes, y si están en marrón, corresponde a que siguen siendo los mismos habitantes Cronenberg de siempre. Por último, si presentan una cruz, quiere decir que no están más entre nosotros. Los contadores, por su parte, indican la cantidad habitantes Cronenberg asimilados por cada Rick.

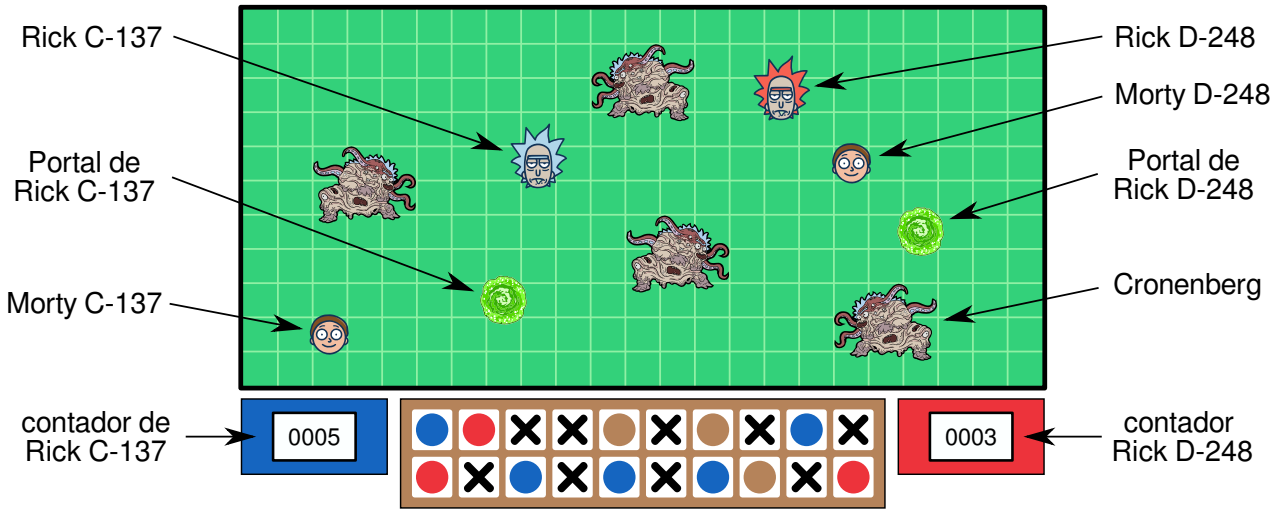


Figura 1: El Juego

Las tareas en sí, ya sean Ricks, Mortys o Cronenbergs ocupan exactamente 8KB de memoria, es decir, dos paginas. En este sentido, cada celda en el tablero que representa el mundo tiene 8KB, suficiente para albergar la memoria de una tarea cualquiera.

Una vez que las tareas aparecen en el mundo, no es posible identificar si son de universo C-237 o D-248. Por esta razón le deben indicar al gran hermano (osea el sistema) de que universo provienen.

Si no dicen nada, seguramente sean tareas Cronenberg (no son muy elocuentes).

El tablero que representa al mundo, corresponde a un área de memoria en donde se ubican todas las tareas. Cada vez que un Rick abre un portal y se mueve de un espacio a otro, literalmente está copiando su código fuente a otra celda dentro del mundo.

Como Rick salta entre portales puede que se le pierda su Morty. Para evitar estos problemas, Rick agrego un localizador de Mortys en su Morty. El sistema provee un acceso al localizador de Mortys que solo puede usar cada uno de los Ricks.

Los portales permitir ir a cualquier lugar del multiverso, en particular, a cualquier lugar del mundo Cronenberg. La forma de calibrar el arma de portales es por medio de desplazamientos `x` e `y` relativos a la posición actual del que use el arma de portales. Además los desplazamientos son modulares al tamaño del mundo (es decir, cuando se llega al limite, se comienza desde el lado opuesto).

El concepto de tomar las mentes es algo muy importante en este juego. Para una tarea, tomar su mente, es equivalente a remplazar su código fuente por uno nuevo. Cada vez que Rick o Morty se mueven por el mundo, pueden estar pisando el código de otra tarea con su propio código. Incluso, el usuario del arma de portales, puede decidir simplemente abrir el portal, esta acción se modela como mapear el espacio de la celda a la que apunta el arma de portales, dentro del espacio de memoria de la tarea que utilizo el arma de portales. Toda la información en la celda apuntada por el arma de portales, será accesible para la tarea que uso el arma. En este sentido, si está área, llegara a ser la mente (código) de algún Cronenberg vivo (o incluso del otro Rick o Morty), se podría modificarla de forma haga lo que uno quiera.

0.4.1 Tareas

Las tareas dispondrán de tres servicios, `usePortalGun`, `IamRick` y `whereIsMorty`. Estos se atenderán con un número de interrupción diferente para cada una. Los parámetros de cada servicio se describen a continuación:

- Syscall `usePortalGun` int `137`

Parámetros	Descripción
`in EAX=x`	Desplazamiento en `x`
`in EBX=y`	Desplazamiento en `y`
`in ECX=cross`	`0`: No cruzar, `1`: Cruzar
`in EDX=withMorty`	`0`: Sin Morty, `1`: Con Morty
`out EAX=worked`	`0`: No creado, `1`: Creado

  Genera un portal desde la posición de la tarea que llamo al servicio a la celda en el desplazamiento indicado por `x` e `y`. Si se indica `cross` en `1`, la tarea será copiada a destino. Si se indica `cross` en `0` la tarea no será copiada, pero serán mapeados los 8k del espacio destino, a partir de la dirección `0x08002000` de la tarea que llamo al servicio. A su vez, si se marca `withMorty` en `1`, la tarea morty asociada al tarea Rick que creo el portal se desplazará o mapeará el lugar destino considerando el desplazamiento indicado, pero esta vez desde donde este ubicada la tarea Morty.

  Este servicio no puede ser usado ilimitadamente. Los Ricks puede crear portales solamente una vez por cada tick de reloj, mientras que los Mortys puede crear portales cada 10 veces que Rick cree un portal. Obviamente, las tareas Cronenberg no tienen arma de portales, así que no pueden usar este servicio.

En el caso que se intente crear un portal pero ya se haya llamado al servicio en el presente tick de reloj, el servicio retornará un `0` en `EAX` indicando que el portal no fue creado. En el caso contrario, cuando el portal sea creado exitosamente, el servicio deberá retornar un `1`.

- Syscall `IamRick` int `138`

Parámetros	Descripción
`in AX=code`	Código de identificación de universos. `code=0xC137` para Rick C-137 `code=0xD248` para Rick D-248

Indica al sistema el universo del que proviene la tarea. Este servicio es utilizado tanto por Ricks como por Mortys para indicar de que universo provienen y así cuando capturan la mente de las tareas Cronenberg poder contarlas en el contador de mentes capturadas. Tener en cuenta que es el único servicio que las tareas Cronenberg pueden utilizar.

- Syscall `whereIsMorty` int `139`

Parámetros	Descripción
`out EAX=x`	Desplazamiento entre Rick y Morty en `x`
`out EBX=y`	Desplazamiento entre Rick y Morty en `y`

Retorna el desplazamiento que existe entre un Morty y su Rick. Este servicio solo puede ser llamado por Ricks para encontrar a sus Morty. El resultado de este servicio es cuantas celdas debe moverse Rick para llegar a la posición donde se encuentra su Morty. Las tareas Cronenberg no pueden utilizar este servicio.

Cualquiera sea el caso, ninguno de los servicios debe modificar ningún registro, a excepción de los indicados anteriormente.

0.4.2 Organización de la memoria

El primer MB de memoria física será organizado según indica la figura 2. En la misma se observa que a partir de la dirección `0x1200` se encuentra ubicado el *kernel*; inmediatamente después se ubica el código de las tareas Rick y Morty del universo C-137, seguidas por las tareas del universo D-248 y el código de la tarea Cronenberg. En último lugar, en la dirección `0x1A000` se encuentra ubicado el código de la tarea Idle. El resto del mapa muestra el rango para la pila del kernel, desde `0x26000` a `0x27000` y a continuación la tabla y directorio de paginas donde inicializar paginación para el kernel. La parte derecha de la figura muestra la memoria a partir de la dirección `0x1A0000`, donde se encuentra mapeada la memoria de vídeo y el código del `BIOS`.

En un rango más amplio, fuera del primer MB, memoria física estará dividida en tres áreas: *kernel*, *área libre kernel* y *mundo*. El área *kernel* corresponderá al primer MB de memoria, el *área libre kernel* a los siguientes 3 MB de memoria y el área *mundo* a los siguientes 25 MB.

La administración del área libre de memoria será muy básica. Se tendrá un contador de páginas a partir del cual se solicitará una nueva página. Este contador se aumentará siempre que se requiera usar una nueva página de memoria, y nunca se liberarán las páginas pedidas.

Las páginas del *área libre kernel* serán utilizadas para datos del kernel: directorios de páginas, tablas de páginas y pilas de nivel cero. Las páginas del *mundo* corresponderán a toda la memoria del mundo Cronenberg, y serán utilizadas para almacenar los códigos, datos y pila de las tareas vivas.

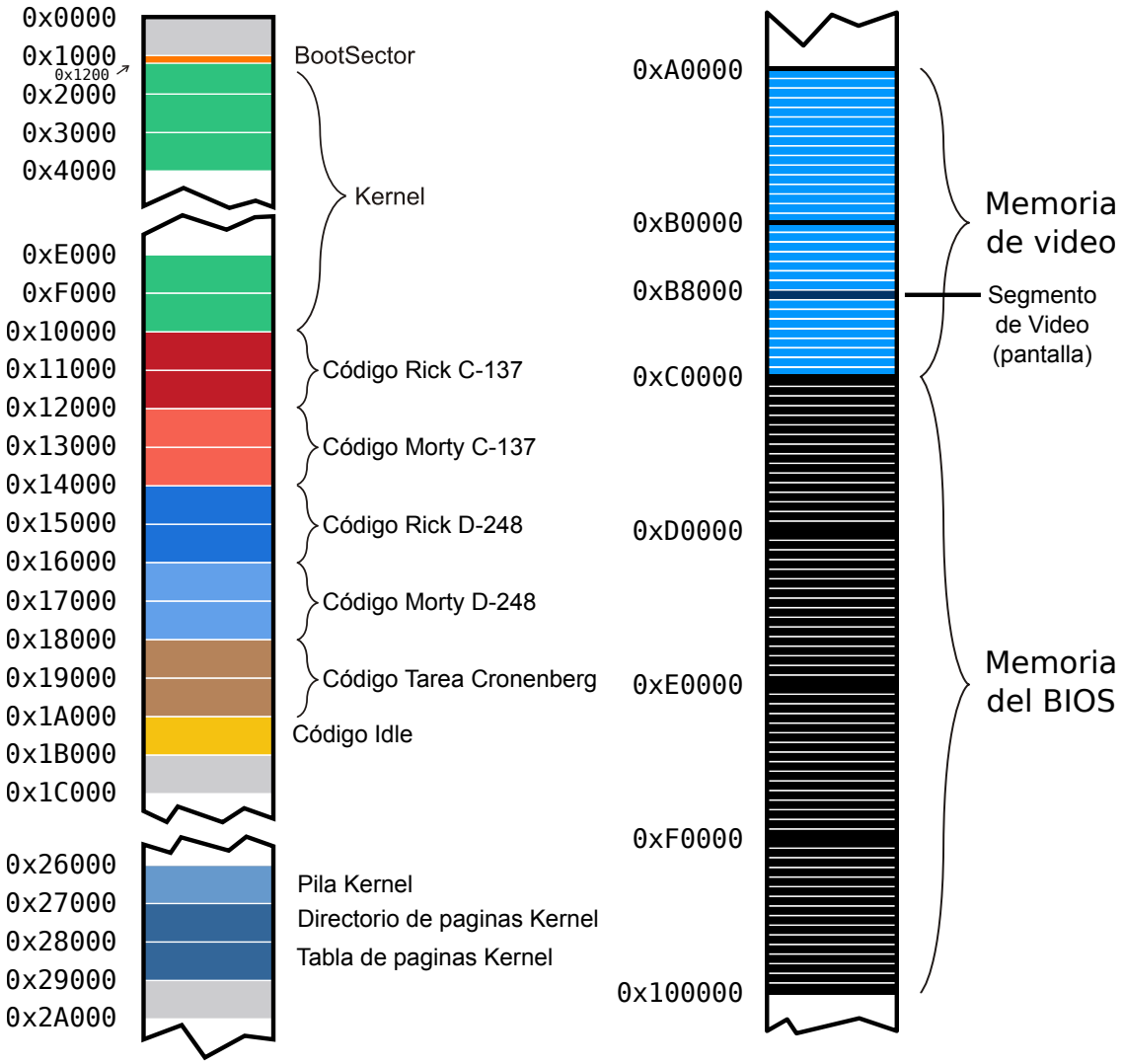


Figura 2: Mapa de la organización de la memoria física del *kernel*

La memoria virtual de cada una de las tareas tiene mapeado el *kernel* y el *área libre kernel* con *identity mapping* en nivel 0. Sin embargo, el área *mundo* no está mapeada. Esto obliga al *kernel* a mapear el área del *mundo*, cada vez que quiera escribir en el. No obstante, el *kernel* puede escribir en cualquier posición del *área libre kernel* desde cualquier tarea sin tener que mapearla.

El código, pila y datos de las tareas estará compartido en un área de 8 KB. La copia original del mismo se encuentra almacenada a continuación del kernel según indica la figura 2. Para construir una nueva tarea se debe realizar una copia de este código al área de memoria del *mundo*. El código de las tareas por su parte estará mapeado en nivel 3 con permisos de lectura/escritura según indica la figura 3.

0.4.3 Scheduler

El sistema va a correr tareas de forma concurrente, durante un tiempo fijo denominado *quantum*. Este será determinado por un *tick* de reloj. Para lograr este comportamiento se va a contar con un *scheduler* minimal que encargará de desalojar una tarea del procesador para intercambiarla por otra en intervalos regulares de tiempo.

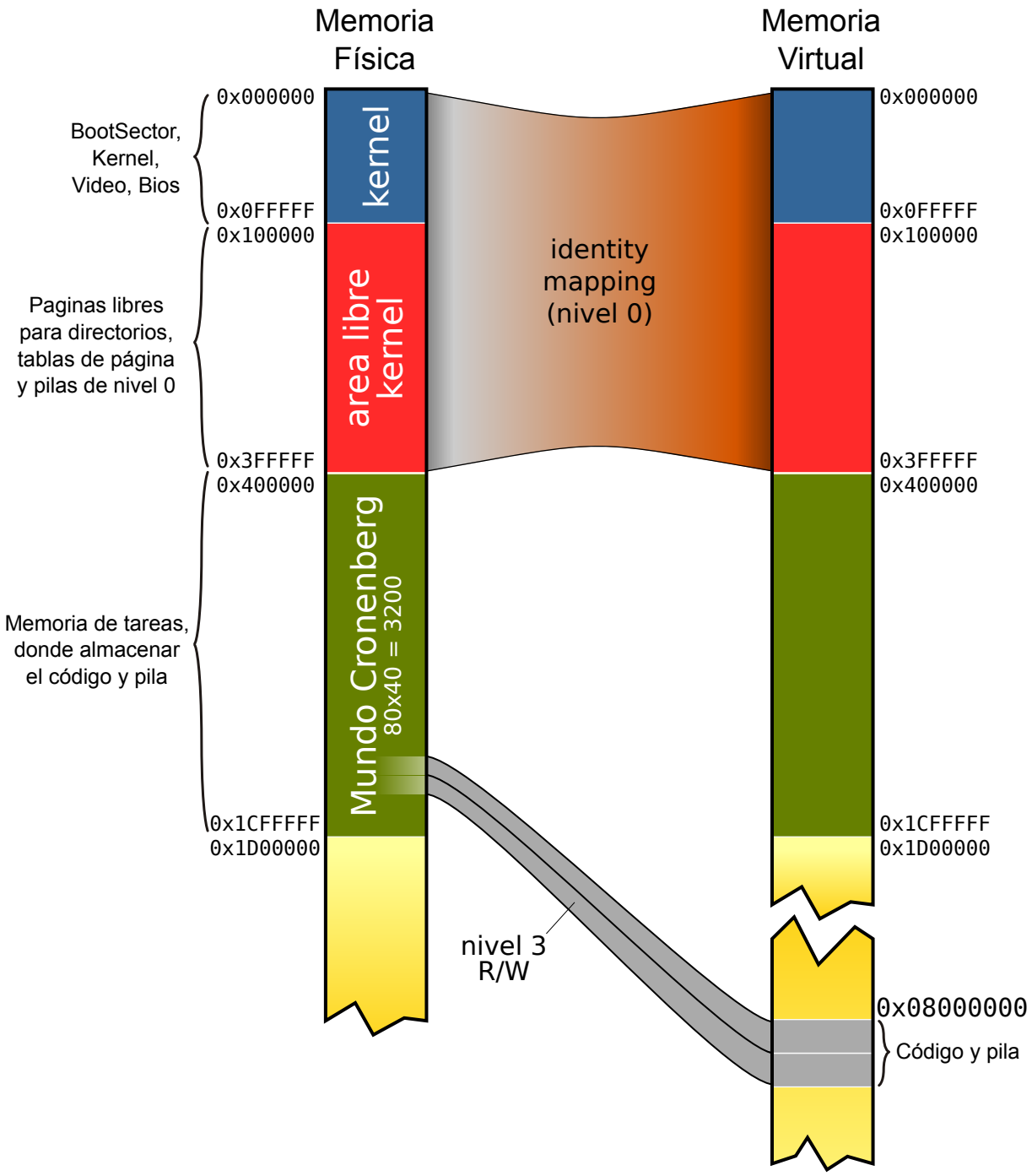


Figura 3: Mapa de memoria de la tarea

Las tareas serán ejecutadas una a continuación de la otra. Este proceso se repetirá indefinidamente. En el caso de que no existan tareas a ejecutar, se ejecutará la tarea `Idle`. Esta es una tarea distinguida dentro del sistema que no realiza ninguna acción. El orden de ejecución de las tareas puede ser cualquiera, pero se debe garantizar que todas las tareas sean ejecutadas al menos una vez.

Las tareas pueden generar problemas de cualquier tipo, por esta razón se debe contar con un mecanismo que permita desalojarlas para que no puedan correr nunca más. Este mecanismo debe poder ser utilizado en cualquier contexto (durante una excepción, un acceso inválido a memoria, un error de protección general, etc.) o durante una interrupción ocasionada por haber llamado de forma incorrecta a un servicio. Cualquier acción que realice una tarea de forma incorrecta será penada con

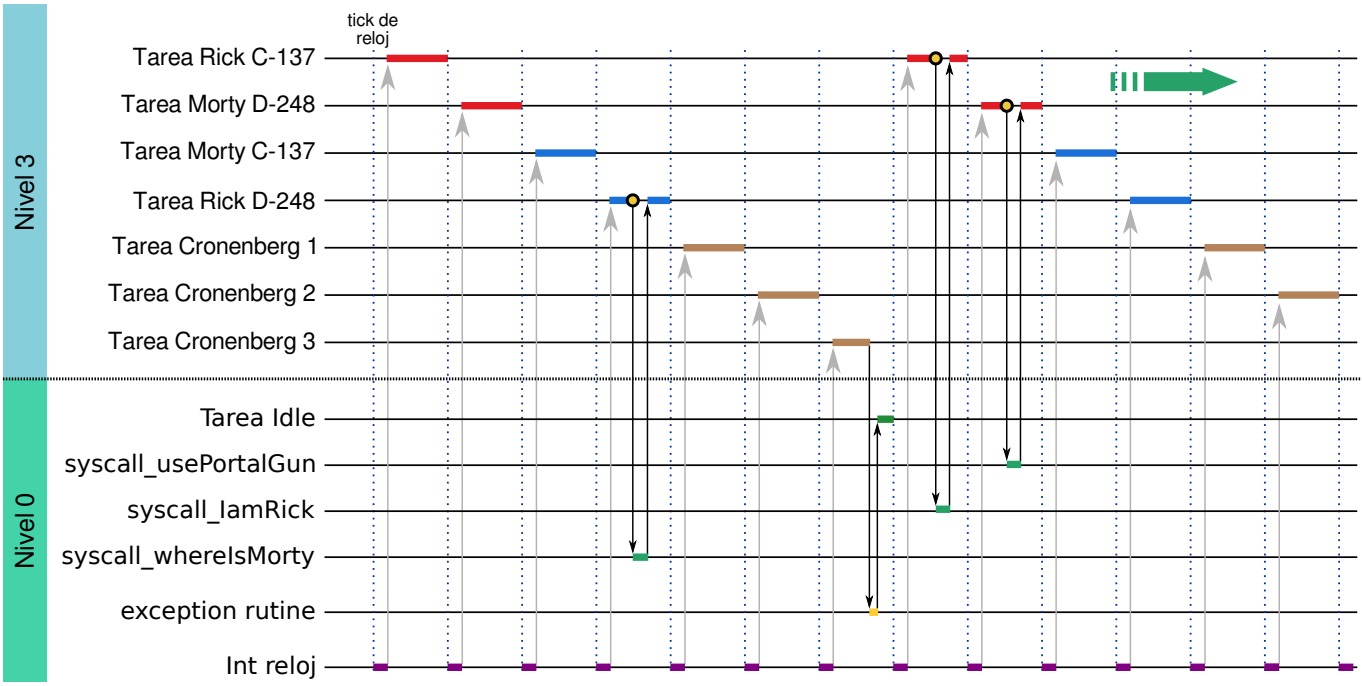


Figura 4: Ejemplo de funcionamiento del *Scheduler*

el desalojo de dicha tarea del sistema (esto en el contexto del juego equivale a que la tarea muera).

Un punto fundamental en el diseño del *scheduler* es que debe proveer una funcionalidad para intercambiar cualquier tarea por la tarea `Idle`. Este mecanismo será utilizado al momento de matar a una tarea, ya que la tarea `Idle` será la encargada de completar el *quantum* actual. La tarea `Idle` se ejecutará por el resto del *quantum* de la tarea desalojada, hasta que nuevamente se realice un intercambio de tareas por la próxima que corresponda.

En la figura 4 se presenta un ejemplo con tres tareas Cronenberg, y las tareas Rick y Morty de ambos universos. Se puede observar como las tareas Rick y Morty llaman a diferentes servicios. Además el caso particular de una tarea Cronenberg, que comente un error y es desalojada para no correr nunca más, continuando por ese *quantum* la tarea `Idle`.

0.4.4 Modo debug

El sistema deberá responder a una tecla especial en el teclado, la cual activará y desactivará el modo de debugging. La tecla para tal propósito es la "y". En este modo se deberá mostrar en pantalla la primera excepción capturada por el procesador junto con un detalle de todo el estado del procesador como muestra la figura 5. Una vez impresa en pantalla esta excepción, el juego se detendrá hasta presionar nuevamente la tecla "y" que mantendrá el modo debug pero borrará la información presentada en pantalla por la excepción. La forma de detener el juego será instantánea. Al retomar el juego se esperará hasta el próximo ciclo de reloj en el que se decidirá cuál es la próxima tarea a ser ejecutada. Se recomienda hacer una copia de la pantalla antes de mostrar el cartel con la información de la tarea.

0.4.5 Pantalla

La pantalla presentará el *mundo* en forma de tablero de 80×40. En este tablero se indicará la posición de todas las tareas, la cantidad de mentes caputaradas actuales y el estado de todas las tareas

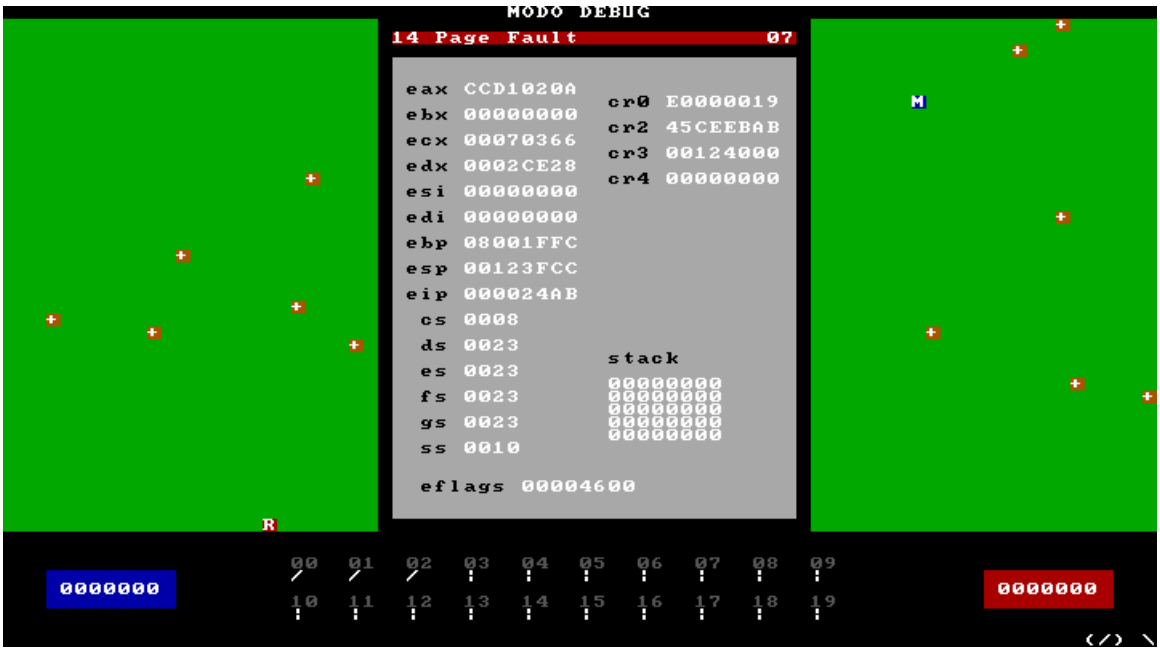


Figura 5: Pantalla de ejemplo de error

Cronenberg. Además se deberá indicar qué tareas están corriendo y cuáles lugares están libres, dado que las tareas ya murieron.

La figura 6 muestra una imagen ejemplo de la pantalla indicando cuáles datos deben presentarse como mínimo. Se recomienda implementar funciones auxiliares que permitan imprimir datos en pantalla de forma cómoda. No es necesario respetar la forma exacta de presentar estos datos en pantalla. Se puede modificar la forma, no así los datos en cuestión.

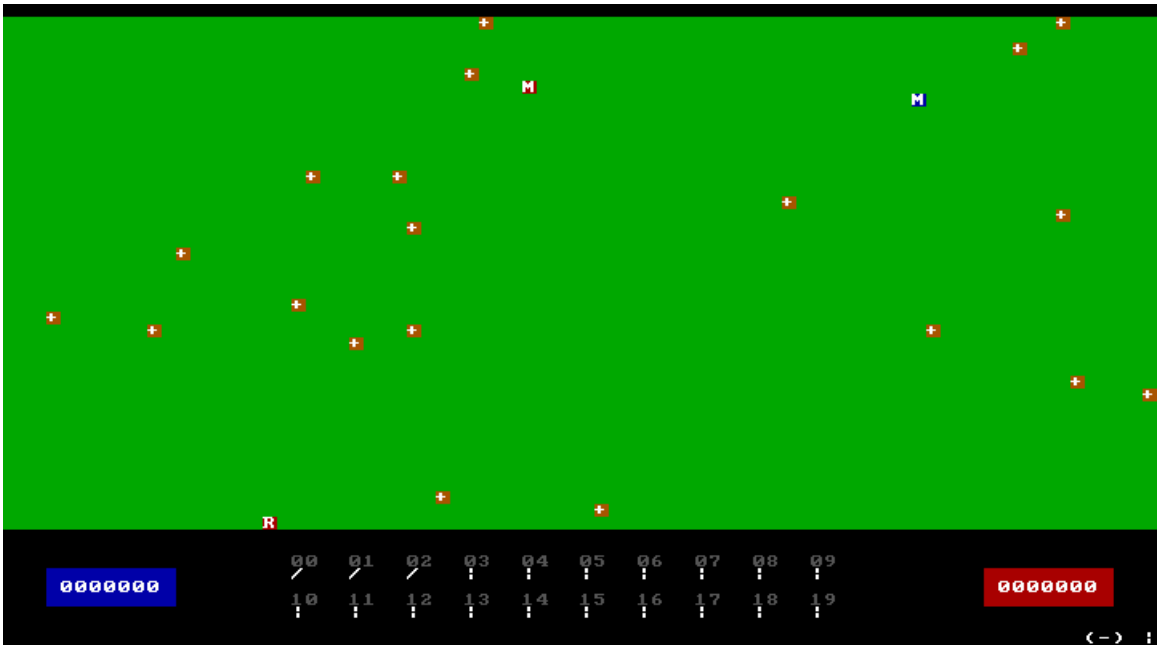


Figura 6: Pantalla de ejemplo

0.5 Ejercicios

Ejercicio 1

a) Completar la Tabla de Descriptores Globales (`GDT`) con 4 segmentos, dos para código de nivel 0 y 3; y otros dos para datos de nivel 0 y 3. Estos segmentos deben direccionar los primeros 137MB de memoria. En la *gdt*, por restricción del trabajo práctico, las primeras 7 posiciones se consideran utilizadas y por ende no deben utilizarlas. El primer índice que deben usar para declarar los segmentos, es el 8 (contando desde cero).

b) Completar el código necesario para pasar a modo protegido y setear la pila del *kernel* en la dirección `0x27000` (es decir, en la base de la pila).

c) Declarar un segmento adicional que describa el área de la pantalla en memoria que pueda ser utilizado sólo por el *kernel*.

d) Escribir una rutina que se encargue de limpiar la pantalla y pintar [1] el área del tablero con algún color de fondo, junto con las barras de los jugadores según indica la sección 0.4.5. Para este ejercicio se debe escribir en la pantalla usando el segmento declarado en el punto anterior. Es muy importante tener en cuenta que para los próximos ejercicios se accederá a la memoria de video por medio del segmento de datos.

Nota: La `GDT` es un arreglo de `gdt_entry` declarado sólo una vez como `gdt`. El descriptor de la `GDT` en el código se llama `GDT_DESC`.

Ejercicio 2

a) Completar las entradas necesarias en la `IDT` para asociar diferentes rutinas a todas las excepciones del procesador. Cada rutina de excepción debe indicar en pantalla qué problema se produjo e interrumpir la ejecución. Posteriormente se modificarán estas rutinas para que se continúe la ejecución, resolviendo el problema y desalojando a la tarea que lo produjo.

b) Hacer lo necesario para que el procesador utilice la IDT creada anteriormente. Generar una excepción para probarla.

Nota: La `IDT` es un arreglo de `idt_entry` declarado sólo una vez como `idt`. El descriptor de la `IDT` en el código se llama `IDT_DESC`. Para inicializar la `IDT` se debe invocar la función `idt_init`.

[1] http://wiki.osdev.org/Text_UI

Ejercicio 3

a) Completar las entradas necesarias en la `IDT` para asociar una rutina a la interrupción del reloj y otra a la interrupción de teclado. Además crear tres entradas adicionales para las interrupciones de software `137`, `138` y `139`.

b) Escribir la rutina asociada a la interrupción del reloj, para que por cada *tick* llame a la función `nextClock`. La misma se encarga de mostrar cada vez que se llame, la animación de un cursor rotando en la esquina inferior derecha de la pantalla. La función `nextClock` está definida en `isr.asm`.

c) Escribir la rutina asociada a la interrupción de teclado de forma que si se presiona cualquiera de `0` a `9`, se presente la misma en la esquina superior derecha de la pantalla.

d) Escribir las rutinas asociadas a las interrupciones `137`, `138` y `139` para que modifique el valor de `eax` por $0x42$, $0x43$ y $0x44$, respectivamente. Posteriormente este comportamiento va a ser modificado para atender cada uno de los servicios del sistema.

Ejercicio 4

a) Escribir las rutinas encargadas de inicializar el directorio y tablas de páginas para el *kernel* (`mmu_initKernelDir`). Se debe generar un directorio de páginas que mapee, usando *identity mapping*, las direcciones `0x00000000` a `0x003FFFFF`, como ilustra la figura 3. Además, esta función debe inicializar el directorio de páginas en la dirección `0x27000` y las tablas de páginas según muestra la figura 2.

b) Completar el código necesario para activar paginación.

c) Escribir una rutina que imprima el número de libreta de todos los integrantes del grupo en la pantalla.

Ejercicio 5

a) Escribir una rutina (`mmu_init`) que se encargue de inicializar las estructuras necesarias para administrar la memoria en el área libre de kernel.

b) Escribir dos rutinas encargadas de mapear y desmapear páginas de memoria.

I- `mmu_mapPage(uint32_t cr3, uint32_t virtual, uint32_t phy)`
Permite mapear la página física correspondiente a `phy` en la dirección virtual `virtual` utilizando `cr3`.

II- `mmu_unmapPage(uint32_t cr3, uint32_t virtual)`
Borra el mapeo creado en la dirección virtual `virtual` utilizando `cr3`.

c) Escribir una rutina (`mmu_initTaskDir`) encargada de inicializar un directorio de páginas y tablas de páginas para una tarea, respetando la figura 3. La rutina debe mapear las páginas del

mundo Cronenberg donde aleatoriamente[2] estará ubicada la tarea a partir de la dirección virtual 0x08000000(128MB). Luego, debe copiar en está área el código de la tarea. Sugerencia: agregar a esta función todos los parámetros que considere necesarios.

d) A modo de prueba, construir un mapa de memoria para tareas e intercambiarlo con el del *kernel*, luego cambiar el color del fondo del primer caracter de la pantalla y volver a la normalidad. Este item no debe estar implementado en la solución final.

Nota: Por construcción del *kernel*, las direcciones de los mapas de memoria (`page directory` y `page table`) están mapeadas con *identity mapping*. En los ejercicios en donde se modifica el directorio o tabla de páginas, se debe que llamar a la función `tlbflush` para que se invalide la *cache* de traducción de direcciones.

Ejercicio 6

a) Definir las entradas en la `GDT` que considere necesarias para ser usadas como descriptores de `TSS`. Mínimamente, una para ser utilizada por la *tarea inicial* y otra para la tarea Idle.

b) Completar la entrada de la `TSS` de la tarea Idle con la información de la tarea `Idle`. Esta información se encuentra en el archivo `tss.c`. La tarea `Idle` se encuentra en la dirección `0x0001A000`. La pila se alojará en la misma dirección que la pila del kernel y será mapeada con *identity mapping*. Esta tarea ocupa 1 página de 4KB y debe ser mapeada con *identity mapping*. Además, la misma debe compartir el mismo `CR3` que el *kernel*.

c) Completar la entrada de la `GDT` correspondiente a la *tarea inicial*.

d) Completar la entrada de la `GDT` correspondiente a la tarea `Idle`.

e) Escribir el código necesario para ejecutar la tarea `Idle`, es decir, saltar intercambiando las `TSS`, entre la *tarea inicial* y la tarea `Idle`.

f) Construir una función que complete una `TSS` con los datos correspondientes a una tarea. Esta función será utilizada más adelante para crear una tarea. El código de las tareas se encuentra a partir de la dirección `0x00010000` ocupando una página de 8kb cada una según indica la figura 2. Para la dirección de la pila se debe utilizar el mismo espacio de la tarea, la misma crecerá desde la base del código de la tarea. Para el mapa de memoria se debe construir uno nuevo utilizando la función `mmu_initTaskDir`. Además, tener en cuenta que cada tarea utilizará una pila distinta de nivel 0, para esto se debe pedir una nueva página del área libre de kernel a tal fin.

Nota: En `tss.c` están definidas las `tss` como estructuras `TSS`. Trabajar en `tss.c` y `kernel.asm` .

[2] No vamos a utilizar un generador de números aleatorios. Las posiciones iniciales de todas las tareas estarán declaradas en un arreglo estático en memoria.

Ejercicio 7

a) Construir una función para inicializar las estructuras de datos del *scheduler*.

b) Crear la función `sched_nextTask()` que devuelve el índice en la `GDT` de la próxima tarea a ser ejecutada. Construir la rutina de forma devuelva una tarea por vez según se explica en la sección 0.4.3.

c) Modificar las rutinas de interrupciones `0x137`, `0x138` y `0x139`, para que implemente los distintos servicios del sistema según se indica en la sección 0.4.1.

d) Modificar el código necesario para que se realice el intercambio de tareas por cada ciclo de reloj. El intercambio se realizará según indique la función `sched_nextTask()`.

e) Modificar las rutinas de excepciones del procesador para que desalojen a la tarea que estaba corriendo y ejecuten la próxima.

f) Implementar el mecanismo de debugging explicado en la sección 0.4.4 que indicará en pantalla la razón del desalojo de una tarea.

Ejercicio 8 (optativo)

a) Crear un par de tareas Rick y Morty que respete las restricciones del trabajo práctico, de no hacerlo no podrán ser ejecutados en el sistema implementado por la cátedra.

   Debe cumplir:

   - No ocupar más de 8 KB (tener en cuenta la pila)
   - Tener como punto de entrada la dirección cero
   - Estar compilado para correr desde la dirección `0x08000000`
   - Utilizar los servicios del sistema correctamente

   Explicar en pocas palabras la estrategia utilizada.

b) Si consideran que su tarea pueden hacer algo más que completar el primer ítem de este ejercicio, y se atreven a enfrentarse en batalla interdimensional de conquistación de mentes, entonces pueden enviar el **binario** de sus tareas a la lista de docentes indicando el nombre de la tarea.

   Se realizará una competencia a fin de cuatrimestre con premios a definir para los primeros puestos.

0.6 Entrega

Este trabajo práctico esta diseñado para ser resuelto de forma gradual. Dentro del archivo `kernel.asm` se encuentran comentarios que muestran las funcionalidades que deben implementarse para resolver cada ejercicio. También deberán completar el resto de los archivos según corresponda.

A diferencia de los trabajos prácticos anteriores, en este trabajo está permitido modificar cualquiera de los archivos proporcionados por la cátedra, o incluso tomar libertades a la hora de implementar la solución; siempre que el producto final resuelva el ejercicio y cumpla con el enunciado. Parte de código

con el que trabajen está programado en ASM y parte en C, decidir qué se utilizará para desarrollar la solución, es parte del trabajo.

Se deberá entregar un informe que describa **detalladamente** la implementación de cada uno de los fragmentos de código que fueron construidos para completar el kernel. Se espera que el informe tenga el detalle suficiente como para entender el código implementado en la solución entregada. Considerar al informe como un documento que ayuda al docente corrector en la tarea de entender su trabajo práctico. En el caso que se requiera código adicional también debe estar descripto en el informe. Cualquier cambio en el código que proporcionamos también deberá estar documentado. Se deberán utilizar tanto pseudocódigos como esquemas gráficos, o cualquier otro recurso pertinente que permita explicar la resolución. Además se deberá entregar en soporte digital el código e informe; incluyendo todos los archivos que hagan falta para compilar y correr el trabajo en Bochs.

La fecha de entrega de este trabajo es **14/07**. Deberá ser entregado a través de un repositorio GIT almacenado en `https://git.exactas.uba.ar` respetando el protocolo enviado por mail y publicado en la página web de la materia. El informe de este trabajo debe ser incluido dentro del repositorio en formato `PDF`.

Ante cualquier problema con la entrega, comunicarse por mail a la lista de docentes.