udp Escuela de Informática y Telecomunicaciones
FACULTAD DE INGENIERÍA

Serpientes sobre hardware

Proyecto 2

Sistemas Operativos

1. Introducción

En este proyecto ustedes escribirán un conjunto de librerías que funcionaran como drivers de la computadora. Ustedes deberán utilizar estos drivers para construir un juego (*snake*) que se ejecute sobre ellos y el hardware mismo. No existirá un sistema operativo entre la aplicación y el hardware, de tal forma que ustedes estarán en control de los dispositivos del computador.

El primer driver será un manejador de gráficos. Éste será responsable de imprimir en la pantalla (pueden imprimir caracteres así como píxeles). Sin este driver será difícil el poder obtener retroalimentación de lo que sucede en la aplicación.

El segundo driver es un manejador de teclado. Este será responsable de escuchar cuando una tecla es presionada, y de realizar una tarea asociada a ella. Por ejemplo, una de las tareas a implementar será el retornar el carácter de la tecla que se presiono, o el estado de ésta (presionada o liberada). Note que el teclado funciona a través de interrupciones, y deberá de poder detener la ejecución de la aplicación para poder manejar el evento.

Para el desarrollo de la este proyecto se les provee con varios ejemplos del uso de aplicaciones que inician desde el hardware que realizan tareas sencillas. Éstas no serán el proyecto final, pero les servirán de partida para poder crear sus librerías, y posterior juego *snakes*. Note que para poder realizar dicho juego, ustedes deben de ser capaces de imprimir en pantalla la serpiente y el fondo. Así mismo, deberán de generar niveles, y dificultades para hacer el juego entretenido. Se evaluará la funcionalidad del juego que presenten.

1.1. Objetivos

Este proyecto es más sencillo que los futuros trabajos sobre un kernel real. Sin embargo, éste sembrará las semillas a varios conceptos fundamentales que les facilitará el trabajo futuro. Concéntrense en entender las ideas fundamentales, y en producir código modularizado, que sea entendido por otros, bien documentado, y que este versionado.

Al final de este proyecto ustedes deberían de:

- Escribir código limpio en `C`. A muchas personas les gusta el lenguaje `C` ya que da libertad al programador (punteros, casting, etc.). Sin embargo, es muy sencillo el tropezar con nuestros pies. Desarrollar (y mantener) un sistema consistente de definición de variables, comentarios, y separación de la funcionalidad es esencial.

  Note que debe de apegarse a `C`, y no `C++`. El escribir un kernel en `C++` es probablemente más difícil de lo que parece, porque se tiene que implementar versiones seguras a hilos (o al menos consciente de interrupciones) de `new` y `delete`. Adicionalmente, usted deberá implementar otras piezas del código de ejecución de `C++`. Aunque estas características no se utilizarán en el presente proyecto, son habilidades que es mejor desarrollarlas ahora para los proyectos futuros, y no empezar a trabajarlas cuando esos proyectos lleguen. De tal manera que el iniciar a familiarizarse con `C` es una buena forma de iniciar los proyectos.

- Escribir pseudocódigo (o diseñar antes de programar). Para la programación de sistemas es muy importante el pensar estructuras de datos cruciales y algoritmos antes de tiempo, ya que se convierten en primitivas para el resto del programa.

- Comentar. Es importante el incluir comentarios para que alguien más pueda ver y mantener el código tan rápidamente como quien lo escribió sin tener que ver los detalles. Debe de comentarse el código utilizando doxygen, que es similar a javadoc pero para `C`.

- Utilizar herramientas comunes de desarrollo, e.g., `gcc` o `ld`.

Adicionalmente a estas metas primarias, otros detalles de implementación como instalar software, aprender a diagnosticar problemas de compatibilidad entre herramientas, decisiones sobre usabilidad, y sobre que camino seguir cuando se nos presentan varias opciones serán desarrollados durante este proyecto.

2. Detalles

2.1. Arquitectura

Dada la disponibilidad, bajo costo, y la amplia existencia de la arquitectura `x86`, se ha escogido a ésta como la plataforma para el desarrollo del proyecto. Como su creador, Intel Corporation ha proveido mucha de la documentación que se utilizará en este proyecto.

Noten que el objetivo del proyecto no es el aprender las idiosincrasias de la arquitectura `x86` o memorizar la documentación de Intel. Sin embargo, será necesario que ustedes se familiaricen con la manera de hacer las cosas en las `x86`, y con la nomenclatura de Intel. Adicionalmente, es bueno el distinguir que hay otras diversas maneras de poder resolver y atacar los mismos problemas.

2.2. Archivos

Durante el desarrollo de su proyecto tenga en cuenta que es una buena práctica el dividir su código entre las cabeceras (`.h`) que definen el comportamiento de las funciones, y la implementación (`.c` y `.S`).

Se les recomienda que en la creación de cabeceras utilicen `#define`s para poder mantener la inclusión de éstas de manera única. Y que generen un buen diseño en el desarrollo del proyecto manteniendo sus archivos de implementación y definición separados.

2.3. Programación de kernel `x86`

En este proyecto ustedes tendrán una nueva experiencia dentro del ámbito de programación. Primero, como programadores de kernel, estarán sujetos a varias restricciones que no aparecen en la programación a nivel de usuario. Segundo, ustedes podrán observar el hardware en el que se ejecutan los programas, y podrán manipularlo, tanto los registros como los dispositivos, y estructuras del procesador.

Este primer proyecto es aún más sencillo que la implementación de un kernel real. Ya que no nos preocuparemos por la administración de recursos (no compartiremos los dispositivos), ni por almacenar información sobre la ejecución de otros procesos debido al cambio de contexto. Sin embargo, nos dará la experiencia suficiente para poder entender los primeros pasos en el desarrollo de kernel, y proveerá la fundación para el entendimiento del funcionamiento del kernel.

2.4. Niveles de privilegios

La arquitectura `x86` utiliza niveles de privilegio para poder defender el kernel contra procesos de usuario salvajes más fácilmente. Existen cuatro niveles de privilegio que van desde cero hasta tres. Siendo cero el nivel con más privilegios, y tres el nivel con menos privilegios. En cualquier momento, el procesador está ejecutando uno de estos cuatro niveles.

Como ustedes se estarán imaginando, el código que escribirán se ejecutará en el nivel de privilegio cero, mientras que el código que están acostumbrados a escribir se ejecuta en nivel de privilegio tres. En este proyecto no utilizaremos los niveles del uno al tres. Dada la manera en que los niveles de privilegio son explicados en la Guía de Programación del Sistema de Intel ([1], Volumen 3, Sección 5.5) se les refiere a estos como anillos (por si encuentran la notación en otra parte).

El procesador revisa el nivel de privilegio en distintas circunstancias, e.g., cuando uno intenta ejecutar una instrucción privilegiada. Éstas son instrucciones que modifican los registros de control o cambian como se ejecuta el procesador (una lista de estas instrucciones esta en la Sección 5.9, Volumen 3 [1]). Si se intenta ejecutar estas instrucciones en un nivel distinto, un error ocurrirá.

2.5. Comunicación con los dispositivos

Hay dos maneras de comunicarse con los dispositivos en la arquitectura `x86`. La primera es enviar bytes al puerto I/O del dispositivo. La segunda es a través de un dispositivo mapeado en la memoria.

La mayoría de los dispositivos en la arquitectura `x86` son accesados a través de puertos I/O. Estos puertos son controlados a través de un sistema especial de hardware que tiene acceso a los datos, direcciones y lineas de control del procesador. Se pueden utilizar estas instrucciones especiales para leer y escribir de y hacia los puertos. De tal forma, nosotros podemos utilizar los puertos I/O sin infringir el espacio de memoria del kernel o de usuario. Esto se debe a que estas instrucciones especiales le dicen al hardware que esta referencia de memoria es en realidad un puerto de entrada y salida, y no una posición real de memoria. Para más información sobre los puertos de entrada y salida, consulten la Sección 10, Volumen 1 [1].

Por conveniencia se muestra un ejemplo simple en el uso de las interrupciones para capturar la presión de una tecla. En general, las funciones `in` leen desde un puerto I/O, mientras que las funciones `out` escriben en él. La letra después del nombre indica el tamaño de los datos que se envían. Por ejemplo, `b` simboliza un byte, `w` una palabra (word o short), `l` un long o int. Para este proyecto, probablemente utilizarán los dispositivos primitivos de I/O de 8-bit (que datan de los '80). De tal forma, utilizarán las funciones `inb()` y `outb()`. Note que esto no le impide utilizar dispositivos más avanzados si así lo considera.

Existen dispositivos que son accesados a través de leer y escribir direcciones particulares en la memoria tradicional. Estos dispositivos son denominados como entradas y salidas mapeadas en memoria. Notemos que este tipo de memoria es parte del espacio de memoria regular y debe ser, entonces, manejada cuidadosamente. El hardware para desplegar video utiliza ambos: I/O mapeados en memoria y puertos.

2.6. Interrupciones de hardware

Un programador de kernel utiliza los puertos de entrada y salida o mapeos en la memoria para enviar comandos a los dispositivos (que varían en tamaño desde un byte hasta long). Sin embargo, ¿cómo se comunican los dispositivos con el kernel (o en este caso, nuestra aplicación)? Cuando un paquete llega a una interfaz de red, o cuando el usuario presiona una tecla en el teclado, o cuando mueve el mouse, o cualquier otro evento que ocurre en un dispositivo de hardware, ese dispositivo necesita una manera de avisar que un evento ha ocurrido. Una manera es que el kernel pregunte a los dispositivos si tienen algo que reportar (ya sea periodica o constantemente). Esto es conocido como entradas y salidas *encuestadas*. Sin embargo, este proceso gasta mucho tiempo de CPU, y existe un mecanismo mejor. Ese mecanismo es la interrupción de hardware.

Cuando un dispositivo quiere desplegar una interrupción de hardware, éste se comunica con uno de dos controladores de interrupciones programables (PICs) a través del envío de señales en las lineas de solicitud de interrupciones (IRQ). Los PICs son responsables de la serialización de las interrupciones (tomando posiblemente interrupciones simultaneas y ordenandolas), y entonces comunican estas interrupciones al procesador a través de las lineas de control especiales. Los PICs le dicen al procesador que una interrupción de hardware ha ocurrido y en que linea de solicitud ocurrió para que el procesador sepa como manejar dicha interrupción. Como son los dispositivos asignados a cada linea de solicitud de interrupciones es complicado, pero existen convenciones que son utilizadas para estas asignaciones. Por ejemplo, en el PIC 1 el timer está en la posición 0, y el teclado en la 2.

Una vez que el procesador obtiene la interrupción desde el PIC necesita saber como responder. Para esto, el procesador lee una estructura de datos llamada la tabla de descriptores de interrupción (IDT). Existe un descriptor en esta tabla para cada interrupción. Un descriptor contiene información variada acerca de como resolver la interrupción, aún más importante donde está localizado el manejador de la interrupción. El manejador de la interrupción es una porción de código (entiendase una función) que el autor del driver del dispositivo escribe para que sea ejecutada cuando el dispositivo emite una interrupción. La IDT también almacena el nivel de privilegio que se necesita para realizar la llamada al manejador (DPL —descriptor privilege level) y el segmento para utilizar mientras se ejecuta el manejador (el que determina el nivel de privilegio para ejecutarse). Una vez el procesador localiza la entrada apropiada en la IDT, este salva alguna información acerca de lo que está pasando antes de que la interrupción ocurriera, entonces inicia la ejecución en la dirección de memoria del manejador de la interrupción. Una vez que el manejador de la interrupción se ha ejecutado, el procesador utiliza la información que salvo antes para poder restaurar la tarea que se estaba ejecutando. Note que los números de las IRQ son índices de la IDT. Los PICs tienen la habilidad de mapear sus lineas de IRQ a cualquier entrada en la IDT.

Noten que los manejadores de interrupción que escriban deben de ejecutarse tan rápido como posible ara que el procesador se libere y pueda recibir otras interrupciones. Cuando una interrupción es enviada por el PIC, éste no envía otra interrupción de la misma fuente hasta que obtiene una respuesta del puerto correspondiente. En particular, un manejador de interrupciones nunca debe de bloquearse. La mayoría de los manejadores de interrupciones simplemente toman nota del trabajo que debe ser hecho como resultado de la interrupción, limpian la interrupción, y terminan, dejando el trabajo que debe ser realizado para otro momento. Noten que puede ser posible para un manejador de interrupciones ser interrumpido por otro manejador de interrupciones diferente, mientras que no compartan estructuras de datos.

2.7. Trampas y excepciones

Las interrupciones son enviadas por el hardware de manera asíncrona al flujo de instrucciones. Si un programa desea salirse de su ejecución normal para invocar un servicio del kernel (por ejemplo), puede entrar en una trampa si emite una instrucción de trampa. Noten que en la arquitectura `x86` se refieren a éstas como interrupciones de software. La instrucción es `INT n`, que causa al procesador el ejecutar el `n`-ésimo manejador en la IDT.

Adicionalmente a las interrupciones de hardware y software, la IDT también contiene la información acerca de los manejadores de excepciones. Las excepciones son condiciones en el procesador que son usualmente no intencionales, y que necesitan ser manejadas. Dependiendo del tipo de diseño que realicen en su aplicación, puede que necesiten utilizar las excepciones para controlar el comportamiento de su aplicación.

2.8. Drivers

Ustedes deberán de utilizar el detalle de las secciones anteriores para diseñar (al menos) dos drivers que manejen la pantalla y el teclado para poder proveer una capa de comunicación con el hardware para la aplicación que necesitan generar.

Para esto deben de analizar las interrupciones a su disposición para poder consumirlas. Se les recomienda el escribir stubs (porciones de código cortas) en

assembler para comunicarse con los dispositivos a bajo nivel, y posteriormente consumir dichas funciones en su código en `C`. Deben de documentar las decisiones de diseño en éstas librerías (ver detalles en la Sección 3.4).

2.9. Aplicación Booteable

Uno de los requisitos es que su aplicación pueda ser ejecutada desde una imagen de disco en un emulador (nosotros nos enfocaremos en dos en particular bochs o qemu). Para esto, se debe de generar una firma dentro de la aplicación que está creando que le diga al BIOS que existe un dispositivo de almacenamiento que contiene código que puede ser ejecutado. Para esto existen varias opciones, una es el colocar la firma `0xaa55` en un byte específico al inicio del programa, e iniciar el código de ejecución en una posición específica también (ver ejemplos proporcionados). Otra opción es el uso de un bootloader (e.g., grub) para que carge su aplicación cuando bootea el dispositivo.

La calificación se realizará sobre máquinas virtuales (que en algunos casos son más fáciles de utilizar durante la depuración del programa). Debe de documentar las pruebas que realizó sobre las distintas plataformas, y explicar la robustez de su código. Para los intrépidos existirá una bonificación si la imagen que genera puede ser booteada en cualquier hardware, y no solamente en los distintos emuladores.

2.10. Serpientes

El juego que deben implementar utilizando sus librerías es el famoso juego snake. Dentro de este deberán de (al menos) detectar el movimiento de las teclas de movimiento (flechas), y permitir al usuario pausar el juego (utilizando la techa de espacio) y resumirlo (utilizando la misma tecla). El juego debe de permitir el progreso para el usuario (llevando un punteo, niveles, dificultades, etc.), manteniendo la usabilidad y estabilidad del mismo.

Dentro del progreso del juego ustedes deberán de crear niveles aleatorios (desde niveles sencillos hasta más difíciles) que creen obstáculos para el jugador. Por ejemplo, deberán de generar paredes las cuales al chocar con ellas decrementan las vidas (oportunidades) del usuario. Dichas paredes deberán de cambiar en forma y dificultad al momento de ir avanzando de nivel.

Al igual que el juego clásico, se generarán piezas comestibles dentro del tablero de juego de manera aleatoria para que el jugador pueda tomarlas. Al comer, la serpiente crecerá un espacio. Y de igual manera que en el original, si el jugador choca la serpiente consigo misma perderá una oportunidad.

Otra funcionalidad dentro de la aplicación es el mantener un registro de los mejores 10 jugadores y sus punteos máximos. Al terminar el programa no es necesario que este registro sea almacenado.

La manera de avanzar será a través del punteo ganado. Cada vez que se coma una pieza, se ganarán 10 puntos, cada 100 puntos se puede avanzar un nivel, y aumentar la dificultad.

En caso de ambigüedad puede guiarse por el juego común, o bien documentar sus decisiones de diseño del juego en el informe.

3. Instrucciones

3.1. Trabajo a realizar

1. Este proyecto se desarrollará en parejas.

2. Lean detalladamente los ejemplos de código proporcionados con este enunciado. Muchas de las respuestas a las preguntas comunes están en el código fuente.

3. Probablemente, se necesitará alguna información que no está disponible en el framework de `C`. Entonces, usted necesitará el utilizar una pieza de código o dos de assembler `x86`. Se recomienda encarecidamente el hacer esto a través de una función de `C` que se llame en un archivo `.S` (note la `S` en mayúscula) en lugar de utilizar las facilidades en linea `asm()`. Cualquiera de las dos formas puede funcionar, pero en la práctica es mucho más sencillo el escribir código con `asm()` que funcione con una versión del programa o una versión en particular del compilador, pero que dejará de funcionar misteriosamente. Adicionalmente, el poblar el código de `C` con llamadas `asm()` lo hace extremadamente difícil de portar de una plataforma a otra.

   El código de soporte incluye un archivo de muestra `.S` (`add_one.S`), y usted puede encontrar el detalle de la función `asm()` en la sección Assembler Instructions with C expression Operands de la documentación de `gcc`. Si aún desea utilizar `asm()`, note que debe utilizar la versión "complicada" para que el programa sea correcto. Así su programa le comunicará su intención al compilador.

   En términos de hacer que `make` construya los archivos `.S`, note que éstos (`.S`) son isomorfos a los archivos `.c` en el sentido que `make` contiene las reglas por defecto para poder construirlos en `.o`.

4. Es importante que lean la documentación de la arquitectura `x86`, así como información sobre el booteo del kernel y de aplicaciones antes de empezar a programar. Diseñen una solución y un plan de trabajo para poder realizar el proyecto.

5. Deben de generar un `makefile` que se pueda ejecutar en cualquier computadora y que genere la imagen final de su aplicación. Este `makefile` debe de crear los distintos objetos de su proyecto, y ser modular para la compilación.

6. Comentar es una parte importante de escribir código. Para esto, utilice doxygen para documentar su código, y se recomienda el seguir la guía de doxygen en el sitio del curso sobre como documentar. Siga las buenas prácticas del proyecto anterior.

3.2. Entregables

El código fuente que generen deberá de tener la siguiente estructura:

- `codigo`: carpeta que contiene todos los archivos fuente que definen su proyecto.
- `informe`: el código fuente para generar el informe que presenta.

Estas carpetas deberán de ser comprimidas en un solo archivo, y subido al sitio del curso junto con un `PDF` del informe, y un `PDF` de la documentación del código generada en Doxygen (ver Sección 3.5 sobre el uso de Doxygen en su código). El archivo comprimido deberá ser nombrado utilizando los RUTs de los integrantes siguiendo el patrón `rut-rut`, donde `rut` es el RUT de los integrantes sin puntos ni guión. Por ejemplo, si los integrantes tienen RUT `1234-5` y `6789-0` el archivo comprimido deberá llamarse `12345-67890`.

3.3. Fecha de entrega

La fecha de entrega es el día 01 de mayo. En horario de clase deberán de entregar el informe impreso. Y deberán subir los archivos al sitio del curso antes de la entrega impresa. Noten que la entrega digital de los documentos cierra al inicio de la hora de la clase, por lo que deberán subir sus archivos antes de la clase.

Se les recomienda el subir los archivos antes de la hora límite para evitar problemas con el servidor. No se considerarán correos que se envíen horas antes de la fecha límite de la entrega con problemas subiendo los documentos.

Se les recuerda nuevamente que el curso tiene una política de *tolerancia cero* para las entregas tardías.

3.4. Sobre el informe

- Deben preparar un informe de **máximo 5 páginas** de contenido (éstas no incluyen las figuras o referencias) utilizando `IEEEtran.cls` y la guía entregada en clase.
- El informe debe de contener al menos:
  - Resumen
  - Introducción (explicando la idea general de sus drivers, e introduciendo las tecnologías que utilizó para su solución)
  - Explicación del diseño e implementación de los drivers para la pantalla y para el uso del teclado. Explique decisiones de diseño, y compromisos al realizar la librería. No es un copy-paste del código, sino una explicación de la función y su diseño y desarrollo. Para revisar el código está la documentación que entrega generada por Doxygen.
  - Explicación del diseño e implementación de su juego. Explique el uso de sus librerías, y la lógica detrás de la implementación.
  - Discusión de las pruebas realizadas. Explique los resultados de las distintas pruebas que realizó y como se aseguró de que el código hace lo que se supone que hace. Y lo más importante, lo que garantiza que funcionará (y compilará) en otra computadora.
  - Conclusión
- **Si se detecta plagio ustedes obtendrán un uno en el proyecto**, y se dará aviso a las autoridades correspondientes para que se tomen las sanciones del caso.

3.5. Notas de interés y restricciones

- Su informe debe de estar realizado siguiendo la guía de documentos técnicos discutida en clase. Recuerde que se penalizará cada falta de ortografía y de redacción con 1 % de la nota final hasta un 20 %.
- No se aceptarán entregas fuera de la hora de entrega. Esto incluye copia digital fuera del plazo, o copia impresa fuera del horario de clase. Además, no se considerará una entrega completa sino existe la entrega digital y la física.
- Para evitar problemas al subir sus archivos (por ejemplo, problemas con el tamaño de la entrega, o disponibilidad del servidor) se les recomienda el subir los archivos antes de la hora de entrega. Noten que la hora de entrega no es la hora para empezar a subir los archivos, sino un límite para tener un punto de corte. Programen su entrega y háganla antes de la fecha límite.
- En caso se sobrepase el límite de páginas estipulado, se evaluará lo presentado hasta el límite solicitado. Para evitar problemas, sigan las instrucciones y sean concisos.

- El código que generen debe de estar documentado utilizando Doxygen. Se recomienda el leer la guía sobre puntos a tener en cuenta al documentar código en el sitio del curso.
- Al documentar en Doxygen deben de utilizar la opción `JAVADOC`, y las opciones afines para la documentación del código.
- Noten que Doxygen permite exportar la documentación a varios formatos, si le es más fácil el evaluar con otro formato mientras desarrolla no hay ningún problema, siempre y cuando el documento final esté generado en LaTeX y se entregue en `PDF`.
- El código fuente que entreguen debe de contener solo archivos que no son generables. Es decir, incluyan archivos fuente de código (`.cpp`, `.h`, `.py`, `.tex`, etc.), pero no incluyan librerías que se generan (`.o`, `.so`, `.dll`), ejecutables, archivos auxiliares, o similares.
- Dada la complejidad del código que puede generar en este proyecto, se recomienda el entregar un archivo `README` comprensivo, que contenga información necesaria para la compilación de su código. Por ejemplo, dependencias que deben existir en la computadora que compila, versione de las distintas herramientas (gcc, ld, nasm, etc.), e instrucciones detalladas para la compilación y ejecución de su aplicación.
- Se recomienda que utilicen un manejador de versiones para mantener su código (por ejemplo, git).
- Usted debe de hacerse responsable de que su código compile, enlace (link), y se ejecute correctamente en cualquier ambiente (se calificará en un ambiente Unix con los emuladores `bochs` y `qemu`).

4. Tips

- Este proyecto es mucho más complejo que el proyecto anterior. Por lo tanto, se les recomienda el empezar al momento de recibir este enunciado. Lean detenidamente los requisitos que se le plantean. Varios requisitos que parecen sencillos implican varios pasos y pre-requisitos que pueden ser complicados en si mismos. Busque información en otras fuentes que complemente las ideas y conceptos que no estén claros, y consulte tanto al ayudante como al profesor oportunamente. El mayor problema del fracaso en este tipo de proyectos es subestimar al proyecto e iniciar un par de días antes. No lo haga.

Tabla 1: Ponderación del proyecto.

	Informe	**Práctica**	**Subtotal**
Resumen	2		2
Introducción	4		4
Diseño librería de la pantalla	8	5	13
Diseño librería del teclado	8	5	13
Diseño del juego	8		8
Funcionamiento de la aplicación		8	8
Creación de niveles		5	5
Animación de la serpiente		5	5
Manejo de colisiones (muerte)		6	6
Administración de niveles (manejo de punteos, vidas, registro jugadores)		6	6
Usabilidad		5	5
Explicación de pruebas	6	5	11
Conclusiones	4		4
Doxygen		10	10
Total	40	60	100

- Revisen los capítulos del libro, así como los apuntes de clase, y las diapositivas para poder entender los conceptos que se plantean. En general, deberá de leer cada documento más de una vez.
- Si bien el informarse es importante, no se paralicen por la cantidad de información. Realicen un mapa de las características que tiene que tener la aplicación, así como de las tareas necesarias para poder resolver cada una de ellas. Creen junto con cada tarea, un conjunto de habilidades que poseen y otras que deben adquirir (por ejemplo, conocimientos sobre x tema, o como ocupar y herramienta). Generen un calendario que les permita medir su avance y las habilidades que han adquirido. Además, este calendario les permitirá el poder balancear la carga de trabajo con otras cosas que tengan que realizar dentro del semestre, por ejemplo, estudiar los temas que siguen avanzando en la clase de Sistemas Operativos, u otros proyectos de la carrera.
- Establezcan metas iniciales. Por ejemplo, el poder compilar una porción de código que despliegue cierto texto en el medio de la pantalla con cierto color, y que este programa se pueda ejecu-

tar desde la imagen que crearon en un emulador. Esta tarea puede demorar más de lo que creen.

5. Ponderación

El proyecto está ponderado 60 % de práctica y 40 % del informe (esto incluye escritura, estructuración de ideas, redacción, etc.) que será evaluado a través del informe escrito, de la documentación entregada en doxygen, y del código entregado. La evaluación será general y se evaluarán los aspectos teóricos, prácticos, y la presentación (a través del informe) en conjunto. El detalle de la ponderación está en la Tabla 1.

Referencias

[1] Intel Corporation, *Intel® 64 and IA-32 Architectures Software Developer's Manual*, December 2015. [Online]. Available: http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-manual-325462.pdf