UNIVERSIDAD CATÓLICA ANDRÉS BELLO

FACULTAD DE INDENIERÍA

ESCUELA DE INGENIERIA INFORMÁTICA

Desarrollo de un gestor de tareas y un kernel para la arquitectura IA-32 que sirva de apoyo durante el proceso de docencia de los Sistemas Operativos.

TRABAJO ESPECIAL DE GRADO

Presentado ante la

UNIVERSIDAD CATÓLICA ANDRÉS BELLO

Como parte de los requisitos para optar al título de

INGENIERO EN INFORMÁTICA

REALIZADO POR — Ocanto España, Victor José
Rodríguez Boscán, Aneriz del Carmen

PROFESOR GUIA — Rodolfo Campos

FECHA — Caracas, 22 de Octubre de 2008

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

SINOPSIS

El presente Trabajo Especial de Grado describe la implementación de un gestor de tareas acompañado de un núcleo básico que da soporte al mismo.

El sistema en general es llamado Vianyx, creado para el apoyo en el área de la enseñanza de actividades prácticas y complemento de la teoría del curso de Sistemas Operativos.

Este sistema presenta una estructura simple frente a otros de mayor alcance debido a su enfoque educativo, sin embargo, está conformado por componentes esenciales que ayudan a su funcionamiento, tales como: módulos para el reconocimiento y tratamiento de interrupciones, controladores básicos de dispositivos como el reloj y el teclado, y un esquema para la planificación de los procesos.

Cuenta además con una plataforma educativa que permite tener acceso tanto al código y su documentación, como al tutorial, a las prácticas propuestas y sus posibles soluciones. Esta plataforma educativa permitirá la interacción entre los estudiantes de la materia y su profesor, facilitando así el estudio del curso y la comprensión del mismo.

Se espera que el sistema no sólo sea estudiado, sino que además sea ampliado por estudiantes.

INTRODUCCIÓN

Un Sistema Operativo es un programa, o conjunto de programas que trabajan coordinadamente para lograr ciertos objetivos -relacionados con los computadores-, entre ellos, principalmente, la asignación de tiempo del Procesador Central (CPU - por sus siglas en ingles *Central Processing Unit*) para planificar las diversas actividades, controlar los recursos de forma racional, distribuir la memoria entre los programas, controlar colas de entrada y salida, gestionar los datos desde y hacia los periféricos, controlar los errores como por ejemplo las excepciones y los fallos de la protección de memoria, informar acerca de las situaciones anómalas, manejo de archivos, procesos y facilitar su uso por medio de una interfaz con el usuario, entre otros.

Este tipo de programa es de naturaleza compleja, ya que la programación de los dispositivos de *hardware* es complicada, por lo que una de las tareas que debe realizar un Sistema Operativo es encapsular la programación de bajo nivel, añadiendo a éste mayor complejidad en el desarrollo.

La parte esencial de un Sistema Operativo es su núcleo (*kernel*), el cual descansa directamente sobre el *hardware* y proporciona una serie de servicios a las capas superiores del sistema. Sus tareas principales son: manejar interrupciones, asignar trabajo al procesador y proporcionar una vía de comunicación entre los diferentes programas. De todas estas funciones, asignar trabajo al procesador (conocido como planificación) es uno de los tópicos más estudiados de los Sistemas Operativos; parte fundamental que abarca desde el manejo de interrupciones,

hasta el concepto de procesos.

El planificador de procesos cumple una función de gran importancia en los Sistemas Operativos, ya que se encarga del área de control de dichos procesos, y a su vez, los procesos son uno de los conceptos fundamentales estudiados en la materia de Sistemas Operativos. Es por ello, que un conocimiento profundo del funcionamiento del planificador o gestor de tareas, es de vital importancia para el estudiante.

Este Trabajo Especial de Grado contempla la implementación de un gestor de tareas acompañado de un núcleo básico que da soporte al mismo, y se encuentra estructurado de la siguiente forma: planteamiento del problema, esta sección explica la problemática actual que motivo la creación de este sistema, junto con los objetivos propuestos, luego se encuentra el marco teórico, donde se explican los conceptos necesarios para la comprensión de este trabajo, seguido por la metodología, la cual describe la forma de trabajo que se siguió para el desarrollo, el cual se detalla en la sección que posee el mismo nombre, luego se encuentran los resultados, los cuales se detallan según los objetivos expuestos en el planteamiento del problema, finalmente se encuentran las conclusiones y las recomendaciones sobre el sistema desarrollado.

Además este documento se encuentra acompañado de un apartado de apéndices, el cual, consta de tres partes llamadas Apéndice A – Documentos de Visión y Diagrama de Clases, el cual contiene –como su nombre lo indica- los documentos de visión escritos para cada uno de los módulos del sistema junto con el diagrama de clases de toda la aplicación; Apéndice B – Casos de Prueba, el cual contiene las

pruebas realizadas a los módulos correspondientes de cada una de las iteraciones; y Apéndice C – Prácticas, contiene dos prácticas propuestas, una que describe una de las prácticas de la primera iteración acerca del módulo de video, y la otra referente a una de las prácticas de la segunda iteración referente al módulo de la planificación.

CAPITULO I. PLANTEAMIENTO DEL PROBLEMA

En general el estudio de los Sistemas Operativos tiene su base en planteamientos teóricos que son aplicables sobre cualquier arquitectura de *hardware* en particular, pero estos por si solos, no son suficientes para alcanzar un profundo entendimiento acerca del funcionamiento interno de los mismos. Este problema ha generado diversas iniciativas para complementar el estudio de este tipo de *software* y por lo general las soluciones propuestas se dividen en dos grandes grupos: unas basadas en un simulador y otras basadas en sistemas de código real, cada una tiene ventajas y desventajas y sirven a diversos propósitos.

Las primeras (basadas en simuladores) poseen una interfaz de programación sencilla y abstraen al estudiante de los conceptos más complejos de la implementación de un Sistema Operativo, permitiendo que se concentren los esfuerzos en reforzar las principales definiciones estudiadas en la teoría. Aunque los simuladores son una buena herramienta para estudiar los Sistemas Operativos, dejan un vació importante en los detalles del funcionamiento real de un sistema y en el porqué son tomadas ciertas decisiones de diseño, por esta razón, existe un segundo tipo de solución, los sistemas de código real. Estos cubren, desde el ámbito de la implementación a bajo nivel, hasta mostrar el funcionamiento real de un sistema sobre cierto tipo de hardware, permitiendo al estudiante no sólo conocer los conceptos teóricos, sino también dominar el complejo mundo de la programación de dispositivos y obtener un conocimiento profundo del funcionamiento interno de un computador.

Existen muchos sistemas de ambos tipos, que han sido creados para el estudio de

los conceptos principales de los Sistemas Operativos, algunos de ellos simuladores, como: "Minikernel", "Nachos" y "OSP", los cuales se encuentran bien documentados pero la mayoría (exceptuando a Minikernel) son en inglés, agregando así complejidad en su estudio por parte de estudiantes de habla hispana. Otros, como: "Xinu", "Linux", "Minix", son de código real, pero entre sus inconvenientes -para el estudio- tenemos que: el lenguaje de programación es complejo, algunos sistemas se encuentran escritos en lenguaje de ensamblador y esto complica su lectura y desarrollo; no están orientados a la enseñanza, aunque son sistemas de código abierto, su principal objetivo es la eficiencia en el manejo de los recursos -lo que hace en extremo difícil su estudio desde un enfoque educacional-; son muy grandes y desarrollados sin una metodología clara.

Lo mencionado anteriormente, produce que el estudiante pierda el interés en modificar los códigos fuentes para ver "qué pasaría si" y limita la curiosidad que éste pueda tener acerca del funcionamiento de los Sistemas Operativos, lo cual podría reprimir una gran oportunidad para el aprendizaje.

El mejor método, o si no, uno altamente efectivo de enseñanza, consistiría en colocar al estudiante en posición de prueba y despertar su curiosidad acerca del sistema, cómo es su funcionamiento, cómo puede cambiarse el desarrollo del mismo y qué podría producir este cambio, hasta llegar a observar cuáles serían las ventajas y desventajas de crear un Sistema Operativo desde otro punto de vista, y quizás hasta con otro tipo de metodología u otro tipo de algoritmos.

Por esta razón, es recomendable el uso de: una metodología clara, las mejores prácticas de programación para atacar este problema, el uso de la programación

orientada a objetos, patrones de diseño y reglas para documentación las cuales pueden ayudar a desarrollar un producto flexible, fácil de modificar, extender y entender. Esto permitirá a los estudiantes concentrarse en los aspectos teóricos de los Sistemas Operativos a la vez que conocen la implementación real de los mismos.

I.1 OBJETIVO GENERAL

Desarrollar un gestor de tareas y un *kernel* básico para la arquitectura IA-32, utilizando el paradigma de programación orientado a objetos, con la finalidad de proporcionar un ambiente de trabajo que sirva de apoyo durante la enseñanza de los principios básicos de los Sistemas Operativos.

I.2 OBJETIVOS ESPECÍFICOS

- Implantar un gestor de arranque.
- Desarrollo de un *kernel* básico, el cual será el responsable de cargar al planificador y proporcionar los servicios necesarios para que -dicho planificador- se ejecute.
- Desarrollar el Planificador (gestor de tareas) que se encargará de llevar el control de la ejecución de los procesos existentes en el sistema.
- Implementar el algoritmo de planificación tipo FIFO (*First In First Out*).
- Implementar el algoritmo de planificación tipo *Round Robin* (torneo).
- Desarrollar prácticas propuestas sobre el planificador del sistema.
- Establecer un ambiente de aprendizaje en línea sobre la herramienta.

I.3 JUSTIFICACIÓN

El estudio de los Sistemas Operativos es un tema de gran importancia y de mucha complejidad, donde el estudio meramente teórico no es suficiente; ante esta problemática han surgido muchas alternativas, desde el desarrollo de simuladores hasta códigos de Sistemas Operativos reales, cuya función principal es la enseñanza.

Colaborar con el estudio de los Sistemas Operativos es el principal objetivo de este proyecto, dando así un aporte a la investigación y desarrollo de *software* de bajo nivel, además de la idea de desarrollar una base que dé soporte a los interesados para ampliar la aplicación del mismo.

Aunque existen algunas iniciativas para la enseñanza de los Sistemas Operativos, la mayoría cubre el objetivo sólo de manera parcial o, algunas veces, se escapa del alcance de un curso de Sistemas Operativos, perdiendo así lo básico, esto los hace herramientas demasiado complejas. Para evitar estos problemas, se decidió cambiar el enfoque, actualmente la mayoría de los sistemas educacionales reales vienen como un producto completo y listo para usarse, pero la mayoría no son desarrollados bajo una metodología clara y abierta, ni muestran paso a paso, de manera incremental, el desarrollo del proyecto; por lo que se desea lograr un producto flexible, que no sólo sirva para el desarrollo de prácticas, sino que guíe en el desarrollo de un gestor de tareas desde las bases teóricas de la implementación, incluyendo la construcción de la base *kernel,* hasta las herramientas necesarias para mostrar cómo pueden aplicarse los conceptos fundamentales del desarrollo de *software* actuales en proyectos de este tipo.

Es por ello que el sistema será realizado bajo una metodología de desarrollo Orientado a Objetos, siguiendo las mejores prácticas de desarrollo de *software* como patrones de diseño; además el sistema estará acompañado de un tutorial que explica paso a paso cómo desarrollar un gestor de tareas sencillo; esto completaría la experiencia teórica, y además proporcionaría una división en módulos, para que el profesor del área pueda escoger la parte de su interés para ser estudiada durante el curso.

La documentación será tanto en código como en guías, para que aquellas personas que estén interesadas en el código puedan entender el funcionamiento del mismo y así tener más claro cómo puede modificarse y ampliarse; se tiene esta idea ya que otros sistemas operativos no están documentados y esto hace que el entenderlos sea una tarea difícil y sumamente complicada.

La razón por la que se desarrollará el gestor de tareas con una metodología Orientada a Objetos es porque ésta permite facilitar en ciertas medidas la comprensión de los conceptos, además los mismos podrán ser manejados de una manera más amplia, específica y coordinada, manteniendo mayor orden a nivel de código, ya que dicho gestor tiene fines educativos más que la eficiencia operacional.

Además de ello, la programación orientada a objetos junto con las mejores prácticas de diseño permite crear un *software* flexible y ampliable, ya que posee características que permiten al desarrollador aumentar su productividad. Dichas características se basan en los pilares de la programación orientada a objetos: herencia, encapsulamiento y polimorfismo, estos permitirán la reutilización de

código, la separación de módulos de forma más limpia y la facilidad de ampliar la funcionalidad del sistema base.

Este sistema estará dirigido, principalmente, a estudiantes e interesados en el área, por lo que será necesario que sus usuarios cuenten con medianos conocimientos de programación, principalmente, en leguaje C y C++, así como también cuenten con un conocimiento básico de los dispositivos de *hardware*. Además, serán necesarios ciertos conocimientos sobre las herramientas de programación y compilación del Sistema Operativo "Linux" (make, gcc, etc), ya que con estas herramientas se realizará el desarrollo y posteriores prácticas sobre el sistema (el cual será escrito y compilado usando las herramientas que "Linux" provee).

Al ser un sistema educacional, se desean hacer guías por capítulos o módulos que lo acompañen y así poder separar la creación y desarrollo del gestor de tareas. Esto puede dar facilidad a las personas que lo estudian, como a las que lo enseñan, porque pueden enfocarse en aquellos puntos que deseen con prácticas más sencillas y explicaciones teóricas, sin tener que perder el hilo de la última clase o el último cambio.

Con este sistema se busca obtener una herramienta propia, en español y de fácil comprensión para aquellos interesados en el estudio de los Sistemas Operativos.

I.4 ALCANCE

El sistema tendrá un enfoque educacional, por lo que abarca los principios básicos de los sistemas operativos como el sector de arranque, el manejo de interrupciones, los procesos y su planificación.

Las prácticas propuestas cubrirán hasta el tema de la planificación, la interfaz proporcionada al usuario será en modo texto, a través de una consola y será un sistema monousuario.

El ambiente de aprendizaje en línea contendrá tutoriales que describan los pasos para desarrollar un núcleo básico, enunciado de las prácticas y respuestas a preguntas frecuentes.

Estará orientado a estudiantes del área de informática en general, que posean conocimientos básicos sobre la programación orientada a objetos, lenguaje de programación C++ y lenguaje de ensamblador. El sistema incluirá una serie de recursos en línea para promover la enseñanza, a saber: tutoriales, preguntas frecuentes y cuestionarios.

I.5 LIMITACIONES

- El sistema estará orientado al entendimiento del código, no a la eficiencia en ejecución.
- El sistema será específico para la arquitectura Intel 32 bits (IA-32).
- No se incluirán características más avanzadas de los Sistemas Operativos, como interfaz gráfica de usuario, sistema de archivos ni manejo de memoria.

CAPÍTULO II. MARCO TEÓRICO

A continuación se presentan los conceptos más importantes involucrados en el desarrollo del tema:

II.1 SISTEMA OPERATIVO:

Es un programa que controla la ejecución de los programas de aplicación y que actúa como interfaz entre las aplicaciones de usuario y al *hardware* del computador éste a su vez tiene tres objetivos: comodidad, eficiencia y capacidad de evolución. [3]

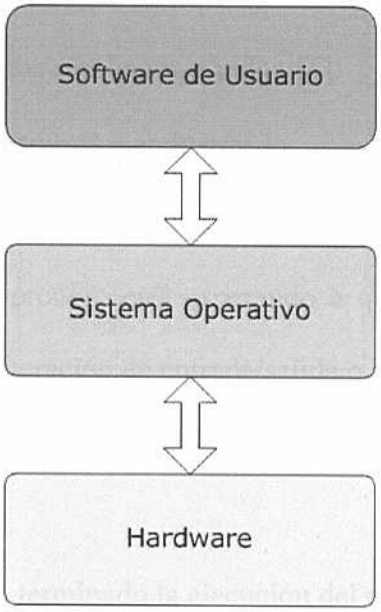

Figura 1 - Capas del Sistema Operativo

Fuente: Stallings, W., "Sistemas Operativos"

II.1.1 NÚCLEO (*KERNEL*):

Es aquel que incluye las funciones utilizadas con más frecuencia en el sistema operativo y, en un momento dado, puede incluir otras partes del sistema operativo que estén en uso. [3]

II.1.2 PROCESO:

Programa en ejecución que incluye los valores actuales de: contador de programa, registros y variables. Un proceso es controlado y planificado por el Sistema Operativo. [4]

II.1.2.1 ESTADOS DE UN PROCESO:

A medida que se ejecuta un proceso, el proceso va cambiando de estado. El estado de un proceso se define, en parte, según la actividad actual de dicho proceso. Cada proceso puede estar en uno de los siguientes estados[1][6]:

- **Nuevo:** El proceso está siendo creado.
- **En ejecución:** Se están ejecutando las instrucciones.
- **En espera:** El proceso está esperando a que se produzca un suceso (como la terminación de una operación de entrada/salida o la recepción de una señal).
- **Preparado:** El proceso está a la espera de que le asignen a un procesador.
- **Terminado:** Ha terminado la ejecución del proceso.

---

[1] Estos nombres varían de un sistema a otro dependiendo del modelo de estado.

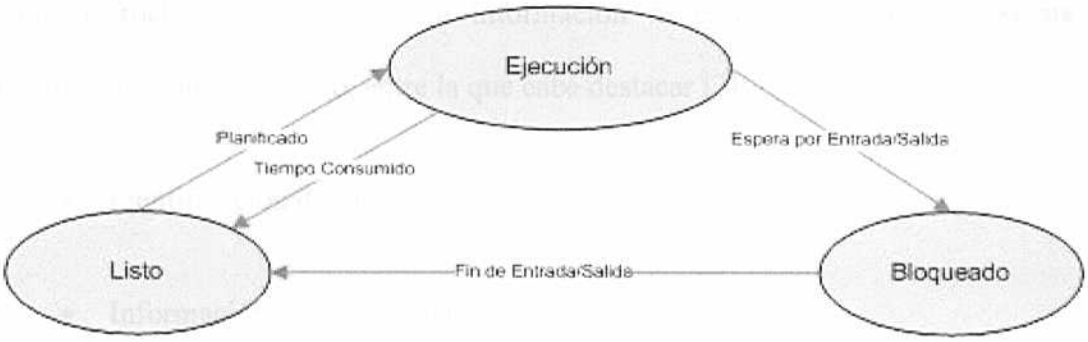

Figura 2 - Modelo de 3 estados

Fuente: Stallings, W., "Sistemas Operativos"

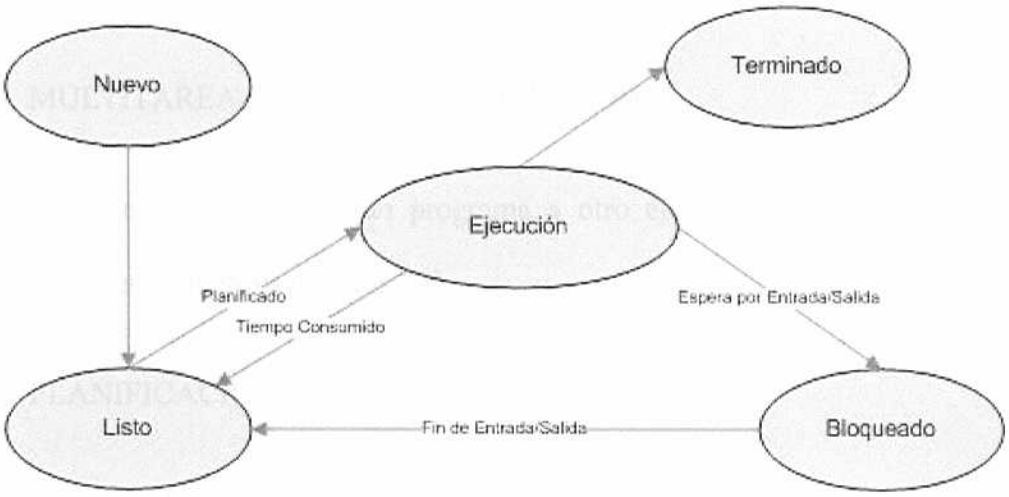

Figura 3 - Modelo de 5 estados

Fuente: Stallings, W., "Sistemas Operativos"

II.1.2.2 BLOQUE DE CONTROL DE PROCESO (BCP):

Es una estructura que contiene la información necesaria para que el Sistema Operativo controle al proceso, entre la que cabe destacar [3]:

- Identificación del proceso.
- Información del estado del procesador.
- Información de control del proceso.

II.1.2.3 IMAGEN DEL PROCESO:

Se define como imagen del proceso a la colección de programas, datos, pila y atributos. [3]

II.1.3 MULTITAREA O MULTIPROGRAMACIÓN:

La multitarea es el cambio de un programa a otro ejecutando cada uno durante decenas o centenas de milisegundos. [4]

II.1.4 PLANIFICACIÓN A CORTO PLAZO:

Se conoce como planificación a corto plazo como la selección de uno de los procesos que se encuentran en la cola de procesos preparados para ejecución. [6]

Su principal objetivo es repartir el tiempo del procesador de forma que se optimicen uno o más elementos del comportamiento del sistema. [3]

II.1.5 PLANIFICADOR:

El planificador es la parte del Sistema Operativo que se encarga de elegir el próximo proceso a ejecutar siguiendo una estrategia de selección la cual es denominada algoritmo de planificación. [3]

II.1.6 REPARTIDOR O ACTIVADOR (*DISPATCHER*):

Es el módulo que proporciona el control de la Unidad Central de Procesamiento, CPU (por sus siglas del inglés *Central Processor Unit*) a los procesos seleccionados por el planificador a corto plazo. Esta función implica lo siguiente:

- Cambio de contexto.
- Cambio al modo usuario.
- Salto a la posición correcta dentro del programa de usuario para reiniciar dicho programa. [6]

II.1.7 ALGORITMO FIFO DE PLANIFICACIÓN (*FIRST IN FIRST OUT*):

Estrategia de selección donde se ejecuta cada proceso en el orden en que se encuentra en la cola de ejecución. Primero en entrar primero en Salir, FIFO (por sus siglas en ingles *First In First Out*). [1]

En este esquema no se plantea expulsión por lo que el proceso se ejecuta hasta que realiza una llamada bloqueante al Sistema Operativo. [5]

Ventajas:

- Es simple de escribir y fácil de comprender.[6]
- Rinde mejor con procesos largos que con procesos cortos.[3]

Desventajas:

- El tiempo medio de espera es a menudo bastante largo.[6]
- Tiende a favorecer a los procesos con carga de CPU frente a los que tienen carga de entrada/salida.[3]
- No es una alternativa atractiva para un esquema de monoprocesador (lo mejor es combinarlo con otro esquema).[3]

II.1.8 TURNO ROTATORIO O TORNEO (*ROUND ROBIN*):

Es un algoritmo de planificación en el cual a cada proceso se le asigna un intervalo de tiempo, llamado **cuanto (*quamtum*),** durante el cual se le permite ejecutarse. Si el proceso todavía se está ejecutando al expirar su cuanto, el Sistema Operativo se apropia del CPU y se lo da a otro proceso. Si el proceso se bloquea o termina antes de terminar el cuanto, la conmutación de CPU naturalmente se efectúa cuando el proceso se bloquee. [4]

Ventajas:

- Es particularmente efectivo en sistemas de carácter general y de tiempo compartido o procesos de transacciones. [3]

Desventajas:

- Los procesos con carga de procesador tienden a recibir una porción desigual de tiempo de procesador, lo que origina un rendimiento pobre de los procesos con carga de entrada/salida, un mal aprovechamiento de los dispositivos de entrada/salida y un incremento de la variabilidad del tiempo de respuesta.[3]

II.1.9 PLANIFICACIÓN POR PRIORIDADES:

Es una política de planificación en la que se selecciona para ejecutar el proceso en estado de listo que tenga la máxima prioridad. Cuando las prioridades son fijas puede surgir el problema de la inanición para evitar este problema se puede añadir un mecanismo de **envejecimiento**[2]. [5]

II.1.10 INTERRUPCIONES:

Mecanismo mediante el cual otros módulos (e/s entrada salida y memoria) pueden interrumpir la ejecución normal del procesador para requerir sus servicios. [3]

[2] Encargado de aumentar la prioridad a los procesos que tengan un determinado tiempo esperando a ser ejecutados.

II.2 IA32:

IA32 es la arquitectura de procesadores de 32 bits de Intel (Intel Architecture 32). Son los procesadores más usados en los ordenadores personales (PC).

Esta gama de procesadores comenzó con el 80386, conocido luego popularmente como 386 o x86 para denominar a toda la gama. Los procesadores de Intel que siguieron y mantuvieron la compatibilidad son: 486, Pentium (ó 586), Pentium II (ó 686), Pentium III y Pentium IV.

La novedad de estos procesadores con respecto a sus predecesores es que incluyen gestión de memoria avanzada (segmentación, paginación, soporte de memoria virtual), unidad de punto flotante, y a partir del Pentium MMX, soporte para operaciones matriciales complejas muy usadas en aplicaciones gráficas y multimedia. [7]

II.3 C++:

Lenguaje de programación creado por Bjarne Stroustrup el cual es un superconjunto del lenguaje C, ampliado para dar soporte a la programación orientada a objetos. [1]

II.4 PROGRAMACIÓN ORIENTADA A OBJETOS:

Paradigma de programación cuya unidad base es el objeto, siendo este la agrupación de datos y operaciones que pueden realizarse sobre ellos. La programación orientada

a objetos posee tres pilares fundamentales, herencia, encapsulamiento, polimorfismo. [1]

II.5 GESTOR DE ARRANQUE:

Un gestor de arranque es un programa que se carga en el momento de arrancar el computador y que permite elegir que Sistema Operativo, de entre los que haya instalados en el disco duro, se quiere cargar.

Cuando el gestor de arranque toma el control puede solicitar al usuario alguna información necesaria sobre qué sistema cargar o cómo cargarlo. [12]

II.6 GRUB:

GRand Unified Bootloader (GRUB) es un gestor de arranque múltiple que se usa comúnmente para iniciar dos o más Sistemas Operativos instalados en un mismo ordenador. Es responsable de cargar y transferir el control al núcleo del sistema operativo de *software*. Fue diseñado e implementado por Erich Stefan Boleyn. [11]

II.7 PRUEBAS DE UNIDAD:

La prueba de unidad se concentra en el esfuerzo de verificación de la unidad más pequeña del diseño del software: el componente o módulo de software. Tomando como guía la descripción del diseño al nivel de componentes, se prueban importantes caminos de control para descubrir errores dentro de los límites del módulo. El alcance

restringido que se ha determinado para las pruebas de unidad limita la relativa complejidad de las pruebas y los errores que éstas descubren. Las pruebas de unidad se concentran en la lógica del procesamiento interno y en las estructuras de datos dentro de los límites de un componente. Este tipo de prueba se puede aplicar en paralelo a varios componentes. [2]

II.8 BOCHS:

Es un emulador de código abierto para PC, fue escrito en C++ por Kevin Lawton y es capaz de ejecutarse en múltiples plataformas y arquitecturas. Su última versión es la 2.3.6 lanzada el 24 de Diciembre de 2007, se encuentra bajo licencia GPL.

Bochs posee la capacidad de emular un CPU con arquitectura Intel x86, dispositivos comunes de E/S, y un BIOS. Actualmente, Bochs puede ser compilado para emular un 386, 486 Pentium, Pentium II, Pentium III, Pentium 4 o una CPU con arquitectura x86, de 64 Bits, incluyendo instrucciones adicionales como las MMX, SSEx y 3DNow!. [9]

II.9 DOKEOS:

Es un entorno de aprendizaje en línea y una aplicación de administración de contenidos de cursos y una herramienta de colaboración. Es *software* libre está bajo la licencia GNU GPL y su desarrollo es internacional y colaborativo.

También está certificado por la OSI[3] y puede ser usado como un sistema de gestión de contenido (CMS[4], por sus siglas en inglés) para educación y educadores. Esta característica para administrar contenidos incluye distribución de contenidos, calendario, proceso de entrenamiento, chat en texto, audio y video, administración de pruebas y guardado de registros. Hasta el 2007, estaba traducido en 34 idiomas y es usado por más de mil organizaciones. [14]

[3] La Open Source Initiative (OSI) es una organización dedicada a la promoción del código abierto

[4] Un Sistema de gestión de contenidos (*Content Management System* en inglés, abreviado CMS) es un programa que permite crear una estructura de soporte (framework) para la creación y administración de contenidos por parte de los participantes principalmente en páginas web.

CAPÍTULO III. METODOLOGÍA

Para el desarrollo del presente Trabajo Especial de Grado, se decidió utilizar la metodología que aplica la empresa Softclear de Venezuela C.A.[1] para el desarrollo de proyectos de software, llamada "Metodología Iterativa Incremental".

Esta empresa adaptó dicha metodología a través de la experiencia obtenida en proyectos y de esta forma generaron un estándar, que es utilizado para el desarrollo de los proyectos que allí se realizan.

Este tipo de metodología plantea cada recorrido, ciclo análisis/diseño/desarrollo/pruebas, como la refinación gradual de las decisiones estratégicas y tácticas, logrando así la reutilización de versiones desarrolladas en iteraciones previas -proceso incremental-. Esto permite manejar cada cambio, incremento o modificación en cada iteración del desarrollo.

La metodología utilizada contempla las siguientes fases -que pueden solaparse-, y que se llevaron a cabo en incrementos -módulos- de forma iterativa -cíclica-. Cada iteración esta conformada por las mismas:

1. Análisis y diseño. El objetivo final de esta fase es obtener el Documento de Análisis y Diseño.

2. Desarrollo. Durante esta fase cada uno de los Desarrolladores de Software se encarga de implementar los módulos que le corresponden y de realizar pruebas de unidad. En caso de que las pruebas -automatizadas o no- sobre las piezas de *software* desarrolladas arrojen resultados desfavorables, se llevarían a cabo tantas

[1] Esta empresa es del tutor académico.

iteraciones como el equipo de desarrollo crea conveniente.

3. Pruebas sobre los entornos de desarrollo y de pruebas. En esta etapa se chequean los módulos, implementados y pasados a prueba por los desarrolladores, dentro del entorno de desarrollo, y por último, sobre el entorno de pruebas.

De igual manera deben ejecutarse pruebas funcionales del sistema, para lo cual se realizan listas de funcionalidades y comportamientos esperados por parte de los componentes del sistema.

4. Pruebas en entorno de producción. Una vez que el sistema haya sido colocado en producción, se valida el comportamiento del sistema, a través de las pruebas elaboradas durante la fase 3.

La documentación es generada de forma intensiva durante la fase de análisis y diseño, contempla: alcance, limitaciones, riesgos, funcionalidades, descripciones, clasificación de módulos, modelo de clases, entre otros. Además, durante la fase de desarrollo se genera documentación en el código fuente de la aplicación.

El presente Trabajo Especial de Grado, consta de un desarrollo de software el cual fue elaborado en tres iteraciones, cada una de ellas contemplando varios incrementos, a saber:

1. **Desarrollo de un núcleo con características básicas**. Esta primera iteración contempla cuatro incrementos (módulos). El conjunto de ellos permite tener las actividades básicas de un núcleo, mostrar mensajes por pantalla, leer caracteres de teclado y manejar interrupciones, de hardware (teclado, video) y las excepciones.

2. **Desarrollo de un planificador básico de procesos**. Esta segunda iteración contempla cuatro incrementos, adicionales a los cuatro incrementos de la primera iteración, donde se unen todas las funcionalidades permitiendo que el sistema pueda reconocer el ciclo de reloj, presentar la multitarea, cambiar de contexto y planificar procesos.

3. **Establecimiento de la plataforma educativa.** Esta tercera iteración abarca el proceso de elección de la herramienta Web a utilizar, el establecimiento de la estructura del curso en línea, la distribución de los documentos complementarios para el aprendizaje, las prácticas propuesta sobre el sistema y sus posibles soluciones.

CAPÍTULO IV. DESARROLLO

IV.1 ITERACIÓN 1: DESARROLLO DE UN NÚCLEO CON CARACTERÍSTICAS BÁSICAS.

Para desarrollar el gestor de tareas fue necesario crear primero la base sobre la cual éste se ejecutaría, dicha base sería un núcleo con funciones básicas para que el gestor de tareas o planificador pudiera funcionar. Algunas de las funciones más importantes del núcleo, son [3]:

- Creación y ejecución de programas.
- Administración del tiempo de procesador que los procesos utilizan.
- Planificación de la ejecución de los procesos.
- Gestión del acceso a los periféricos/elementos de nuestro computador de una manera cómoda.

Tomando en consideración los puntos anteriormente expuestos, se comenzó el desarrollo de los módulos (incrementos) principales que cubrirían las necesidades subyacentes.

El desarrollo fue dividido siguiendo la metodología seleccionada, no solo para facilitar el desarrollo, sino además para facilitar su posterior comprensión y modificación.

IV.1.1 ANÁLISIS Y DISEÑO.

En esta fase se realizaron los documentos de visión para los módulos de arranque, vídeo, interrupciones y teclado (contenidos en el Apéndice A – Documentos de Visión y Diagrama de Clases). Cada uno de estos documentos contiene la definición, los casos de uso, los requerimientos funcionales y no funcionales y los diagramas de casos de uso para dichos módulos. Además, contienen la descripción del elemento a desarrollar y una breve definición sobre su funcionalidad, junto con el diagrama de clases del sistema.

Para obtener los requerimientos, casos de usos, y otros elementos que conforman el documento de visión, se realizaron entrevistas con expertos en la materia. Específicamente, el Profesor Dario León del área de Sistemas Operativos, ayudó a conocer con mayor detalle los temas básicos en la materia, los tópicos principales según sus niveles de complejidad para los alumnos y aquellos puntos necesarios para el aprendizaje.

IV.1.2 DESARROLLO.

Para el desarrollo de este proyecto se utilizó Linux SUSE la versión 10.0, junto a las herramientas de programación y compilación descritas más adelante, sobre un computador Intel Pentium IV de 2.40 GHz con 512 Mb de memoria RAM.

El primer paso fue establecer el entorno de desarrollo y las pruebas. Para el cual se debieron descargar las aplicaciones necesarias, instalarlas, configurarlas y probarlas.

Las aplicaciones necesarias fueron:

- Simulador de hardware: como simulador se utilizó Bochs, el cual se instaló descargando un paquete `rpm` *(red hat package manager)* y la configuración del mismo mediante el archivo `bochs.src`; indicando la cantidad de memoria simulada y los dispositivos que se requerían.
- Compilador del lenguaje de ensamblador: se utilizó NASM, su instalación consistió en la descarga de un paquete `rpm`. No necesitó configuración especial.
- Editor de lenguaje C++: se utilizó Kate[1] el cual viene con la distribución SUSE de Linux.
- Compilador de C++: se utilizó g++ el cual fue instalado usando el cd de instalación de la distribución SUSE de Linux.
- El enlazador (linker): se utilizó ld[2] y su instalación fue igual a la del Compilador C++.

IV.1.2.1 MÓDULO 1: ARRANQUE

El gestor de arranque del Sistema Operativo no forma parte del mismo, pero es de vital importancia, ya que sin él no sería posible ejecutar el núcleo, debido a que es éste quien establece el ambiente mínimo necesario para que pueda funcionar dicho núcleo.

[1] Es un editor de textos para KDE. Kate significa KDE Advanced Text Editor, es decir Editor de textos avanzado para KDE.

[2] Es un enlazador para las plataformas tipo Unix. Se incluye como parte de una colección de herramientas de programación para la manipulación del código objeto.

Para la realización de este Trabajo Especial de Grado se ha utilizado GRUB como gestor de arranque, debido a que es una herramienta configurable y fácil de adaptar al sistema, además de la capacidad para reconocer múltiples formatos ejecutables y sistemas de archivos.

Para obtener una imagen de GRUB en el disco extraíble (*floppy*) fue necesario escribir en el primer y segundo sector del disco los archivos `stage1` y `stage2` del directorio de GRUB. A continuación se muestran los pasos seguidos para realizar la configuración:

```
# cd /usr/lib/grub/i386-pc
# dd if=stage1 of=/dev/fd0 bs=512 count=1
1+0 records in
1+0 records out
# dd if=stage2 of=/dev/fd0 bs=512 seek=1
153+1 records in
153+1 records outz
```

Una vez instalado el gestor de arranque, fue necesario generar su configuración indicando el núcleo a cargar, el nombre a ser mostrado para el usuario y la dirección en el dispositivo físico del archivo binario ejecutable que conforma al núcleo.

Para ello se realizó lo siguiente:

En el directorio "boot/grub/", del disco extraíble que contiene a GRUB, se creó un archivo llamado `menu.lst`, el cual se puede abrir con cualquier editor de texto, donde se escribió lo siguiente:

```
title    Vianyx
root     (fd0)
kernel   /boot/Vianyx.bin
```

Donde:

`title`: indica el nombre que aparecerá en la lista de opciones de GRUB.

`root`: indica el disco extraíble.

`kernel`: se refiere a la ruta (relativa a la raíz) donde se encuentra el sistema operativo.

Luego de tener lista la configuración del cargador, se procedió a codificar la base del núcleo, la cual constaría de un archivo escrito en lenguaje de ensamblador con las directivas necesarias para que dicho cargador pueda reconocer y cargar al núcleo en memoria.

```
;##########################################################################
MULTIBOOT_PAGE_ALIGN    equ 1<<0   ; Cargar el núcleo y sus módulos alineados a 4kb (1 pagina)
MULTIBOOT_MEMORY_INFO   equ 1<<1   ; Proveer al núcleo con información de memoria

MULTIBOOT_HEADER_MAGIC equ 0x1BADB002
MULTIBOOT_HEADER_FLAGS equ MULTIBOOT_PAGE_ALIGN | MULTIBOOT_MEMORY_INFO
CHECKSUM               equ -(MULTIBOOT_HEADER_MAGIC + MULTIBOOT_HEADER_FLAGS)


align 4
dd MULTIBOOT_HEADER_MAGIC
dd MULTIBOOT_HEADER_FLAGS
dd CHECKSUM

;Fin de la cabecera
;##########################################################################
```

Figura 4 – start.asm (Sección de la cabecera para GRUB).

Fuente: Elaboración propia.

En la figura anterior (Figura 4) podemos observar parte del archivo `start.asm` en el cual se encuentran las directivas mencionadas anteriormente.

La codificación de esta base involucró la creación de archivos de configuración para indicar la forma de compilación y enlace de los ejecutables, dicha

configuración se realizó mediante archivos conocidos como `makefile`[3]. Durante esta primera fase se obtuvo la base para el resto del desarrollo.

IV.1.2.2 CREACIÓN DE LA FUNCIÓN PRINCIPAL

Aunque la base del núcleo fue escrita en lenguaje de ensamblador, éste o su mayor parte están escritos en C++, por lo que sería necesario crear una forma de pasar el control a funciones escritas en dicho lenguaje. Por ello se creó un "puente" que comunicaría a ambos lenguajes (agregado en el archivo `start.asm`). Para esta primera fase se incluyó la capacidad de mostrar mensajes en la pantalla del computador. Los pasos seguidos fueron los siguientes:

a) Se creó un archivo llamado `Main.cpp` en el directorio `/kernel`, para lo cual se agregó la llamada a la función main en el archivo `start.asm` como se muestra a continuación:

[3] Archivos que sirven para automatizar operaciones de compilación y configuración entre otras.

```
;##############################################################################
MULTIBOOT_PAGE_ALIGN    equ 1<<0   ; Cargar el núcleo y sus módulos alineados a 4kb (1 pagina)
MULTIBOOT_MEMORY_INFO   equ 1<<1   ; Proveer al núcleo con información de memoria

MULTIBOOT_HEADER_MAGIC equ 0x1BADB002
MULTIBOOT_HEADER_FLAGS equ MULTIBOOT_PAGE_ALIGN | MULTIBOOT_MEMORY_INFO
CHECKSUM               equ -(MULTIBOOT_HEADER_MAGIC + MULTIBOOT_HEADER_FLAGS)



align 4
dd MULTIBOOT_HEADER_MAGIC
dd MULTIBOOT_HEADER_FLAGS
dd CHECKSUM

;Fin de la cabecera
;##############################################################################
[BITS 32] ; Indica código para modo protegido

[global _start] ;#Punto de Entrada del núcleo
[extern main] ;#Esta es la función principal del núcleo en leguaje c++
[extern __cxa_finalize] ;#Función para soportar objetos globales debe ser llamada luego de main
_start:

push ebx      ;#Colocamos en la pila un puntero a la estructura multiboot para pasarla al núcleo
call main
call __cxa_finalize

ret ;
```

Figura 5 – start.asm

Fuente: Elaboración propia.

Esta figura muestra el código escrito en lenguaje de ensamblador y la forma en la que se comunica con la función `main` de C++, donde se pueden apreciar detalles importantes, a saber:

- La directiva `[extern main]` : permite al compilador (g++) conocer el hecho de que dicha función se encuentra en un archivo externo y que será enlazada posteriormente por el enlazador o linker (ld).

- La directiva `push ebx`: GRUB puede pasar información que podría resultar de gran utilidad para el núcleo. Esta información incluye la cantidad de memoria en el sistema y la línea de comandos usada para

cargar el núcleo, entre otras. La forma en la que se puede pasar dicha información a la función `main` es a través de la "pila", realizando un `push` previo a la llamada de la función.

Las funciones C++ se encargarán de recuperar el valor indicado de forma transparente para el programador. Además, fue necesario crear una representación de dicha estructura en C++; esta clase se llama `multibootinfo_t`.

- La llamada a la función principal de C++: Se realizó con la instrucción `call main`. Esta instrucción permite realizar la llamada a la función principal `main` y de esta forma se logró el paso del lenguaje de ensamblador a C++.

b) La configuración para la compilación y posterior enlace de los archivos generados se realizó utilizando archivos `makefile`.

```
# Makefile para generar la entrada al núcleo

# El Nasm
NASM = nasm

all: Start.o

Start.o : Start.asm
    $(NASM)  -f elf  -o ../Binario/$@ $<
```

Figura 6 – Makefile para generar el núcleo base.

Fuente: Elaboración propia.

En la figura anterior (Figura 6) se pueden observar las instrucciones necesarias para compilar el archivo `start.asm`, utilizando el compilador NASM.

C++ es un lenguaje que cuenta con ciertas funciones que se ofrecen para simplificar algunas tareas comunes, como por ejemplo, funciones para el manejo de cadenas de caracteres, manejo de memoria, entre otras. Algunas de estas funciones, se encuentran en la librería estándar y otras forman parte de la especificación del lenguaje, estas son soportadas por cada Sistema Operativo en el que se compila el código C++, es decir, existe una implementación específica para cada sistema en el que se ejecuta dicho código. Además, cada compilador puede variar en la forma como soporta algunas características del lenguaje.

Se encuentra fuera del alcance de este desarrollo realizar una implementación completa de la librería, no obstante el compilador utilizado intentará hacer referencia a dicha librería, y algunas funciones resultan necesarias (como las funciones virtuales puras), es por ello que se deben realizar operaciones de configuración que eviten la inclusión de aquellas funciones que no se utilizaran, y crear un soporte para las que serán necesarias. A continuación se detallan las funciones que el compilador intentará utilizar:

- Las funciones de la librería estándar: funciones para el manejo de cadenas de caracteres, funciones matemáticas, entre otras.
- Las funciones de construcción y destrucción de objetos (`builting`): `new`, `delete`.
- Soporte a variables globales, funciones virtuales puras y excepciones.
- Las funciones de información de tipo en tiempo de ejecución (RTTI *Run Time Type Information*): como el operador `dynamic_cast ()`.

En el proceso normal de creación de un ejecutable, el compilador (en éste caso GCC) intentaría utilizar las librerías predeterminadas, las cuales como se explicó anteriormente, no estarían disponibles, por lo que fue necesario cambiar la configuración de dicho compilador. Para lograrlo, se utilizaron las siguientes opciones al compilar:

- `-nostdlib`: evita la inclusión de la librería estándar.
- `-fno-builtin`: evita la inclusión de las funciones de construcción (`new-delete`).
- `-fno-rtti`: evita la inclusión de las funciones de RTTI.
- `-fno-exceptions`: evita la inclusión de las funciones para manejo de excepciones.

Al compilar los fuentes con estas opciones se obtuvieron los siguientes resultados:

Funciones disponibles:

- Variables globales y funciones virtuales puras: se utilizó una implementación abierta para el compilador g++ versión 4.0.2 ó superior, esta implementación se encuentra en el archivo `Soporte.cpp`, que ayuda a crear variables globales y funciones virtuales puras, las excepciones no se abarcan en el desarrollo del sistema.

Funciones no disponibles:

- Funciones de la librería estándar, las cuales pueden ser portadas durante el desarrollo.
- Funciones de construcción y destrucción de objetos (`new` y `delete`)
- RTTI (*RunTime Type Information*)
- Manejo de excepciones

Al conocer las limitaciones y las formas de compensarlas, se creó el archivo `makefile`, con las indicaciones específicas para evitar que se intentaran utilizar las funciones de la librería estándar. El archivo se muestra a continuación:

```
# Makefile para generar el núcleo

# Comando para borrar archivos
REMOVE = rm -f

# El compilador de C y C++
GCC = gcc
G++ = g++

# El linker
LD  = ld

# Nombre del código objeto generado
OBJS = Start.o Soporte.o  Video.o Main.o

# Nombre del ejecutable
KERNELFN = Vianyx.bin
CPPFLAGS = -Wall -nostdlib -fno-builtin -fno-rtti -fno-exceptions -O -I.

# Link
all: $(KERNELFN)

$(KERNELFN) : Main.o Soporte.o Arranque.o Controladores.o
    cd Binario/; $(LD)  -o $@ $(OBJS)  -Ttext 0x100000; cp $@ /media/floppy/boot/; cd ../../;bochs

PruebasUnitarias: Main2.o Soporte.o Arranque.o Controladores.o
    cd PruebasUnitarias/; make;
    cd Binario/; $(LD)  -o Vianyx.bin $(OBJS) Pruebas.o -Ttext 0x100000; cp Vianyx.bin /media
    /floppy/boot/;

Arranque.o:
    cd Arranque/; make;

Controladores.o:
    cd Controladores/; make;

Main.o: Main.cpp
    $(G++) $(CPPFLAGS) -c -o Binario/$@ $<

Soporte.o: Soporte.cpp
    $(G++) $(CPPFLAGS) -c -o Binario/$@ $<

Main2.o: Main.cpp
    $(G++) $(CPPFLAGS) -DTEST -c -o Binario/Main.o $<



# Limpiar
clean:
    cd Binario/; $(REMOVE) $(OBJS)
```

Figura 7 - makefile para compilar y enlazar todas las partes del núcleo.

Fuente: Elaboración propia.

La definición general de la estructura de un programa ejecutable, indica que estos poseen al menos tres secciones bien definidas, a saber: una para los datos, otra para el código y otra para la pila. El núcleo, al no ser más que otro programa,

necesitaba definir dichas secciones y el compilador define por defecto algunas de estas, pero aún así, fue necesario establecer manualmente un valor para el registro que definiría la pila, esto se realizó en la función `start` (ver Figura 8).

```
mov esp,stackKernel  ;#Establecemos la pila del núcleo (Recordar que esp apunta al comienzo de la pila )

push ebx     ;#Colocamos en la pila un puntero a la estructura multiboot para pasarla al núcleo
call main
call __cxa_finalize
hlt
;# Sección bss es para data no inicializada allí colocamos la pila
section .bss

resb 100000h
stackKernel:
```

Figura 8 – start.asm (Sección para reservar la pila).

Fuente: Elaboración propia.

En la figura anterior se puede observar la forma en la que se definió la pila para el núcleo, se reservaron `0x100000`[4] bytes para dicha pila y se colocó una etiqueta al final del espacio reservado, esto debido a que la pila crece hacia direcciones decrecientes de memoria, así, al iniciar el núcleo, sólo fue necesario establecer el valor del registro `esp` al fin del espacio reservado. Esto se logró con la instrucción `mov esp, stackKernel`.

[4] Los valores con el prefijo 0x se encuentran en hexadecimal.

IV.1.2.3 MÓDULO 2: VÍDEO

Para desarrollar el módulo de vídeo, se procedieron a implementar aquellas funciones necesarias (plasmadas en el documento de visión del módulo) dentro de una clase específica. Luego se procedió a instanciar un objeto de esa clase con alcance global dentro del proyecto, esto motivado por el hecho de que la aplicación (el núcleo) necesitaba mostrar información acerca de su estado y mensajes independientemente del módulo.

La clase `Video` consta de una función llamada `escribir`, la cual es sobrecargada para permitir la impresión en pantalla de diferentes tipos de datos. En el modo de vídeo, conocido como modo texto, la pantalla se divide en filas de 25 caracteres y columnas de 80 caracteres, con 2 bytes por carácter que aparecen en la pantalla. El primer byte es el valor ASCII del carácter propio, y el segundo es el atributo.

El dispositivo provee una zona de memoria, la cual puede verse como un arreglo, en donde se pueden colocar los caracteres y sus atributos para ser mostrados en la pantalla, esta zona comienza en la posición `0xB8000` de la memoria física, sabiendo esto, se codificaron las funciones que permitían el acceso a esa zona de memoria y su posterior manipulación.

IV.1.2.4 MÓDULO 3: INTERRUPCIONES

Aunque el manejo de la memoria escapa del alcance del proyecto, debido a requerimientos específicos de la plataforma, la cual obliga al uso de algunos de los esquemas de protección y direccionamiento de memoria, fue necesario

establecer un sistema básico para la manipulación de la misma, en este caso la segmentación. Para ello, se procedió a crear la clase necesaria para poder controlarla, dicha clase lleva por nombre `Tdg` y representa la estructura que posee la arquitectura IA-32 llamada Tabla de Descriptores Globales (comúnmente llamada GDT, por sus siglas en inglés, en lugar de TDG) la cual es la encargada de describir los segmentos de memoria utilizados indicando sus atributos de acceso y tipo se segmento (datos, ejecutable, sólo lectura, entre otros).

Para el desarrollo del proyecto se establecieron tres segmentos de memoria solapados, conformando lo que se conoce como "modelo de memoria plano". Este modelo implica que el inicio y fin de cada uno de los segmentos es el mismo y van desde la dirección cero hasta el fin de la memoria, a la que se puede acceder; en este caso 4Gb (por ser direcciones de 32 bits). De esta forma no existe protección real entre los segmentos, pero a cambio se obtiene una forma simple de manipular la memoria del sistema.

IV.1.2.5 MÓDULO 4: TECLADO

Para poder controlar el dispositivo de teclado era necesario contar con un sistema básico de manejos de interrupciones. Luego de haber identificado las interrupciones que el sistema controlaría, se codificaron las clases necesarias para reconocerlas y procesarlas, específicamente las interrupciones de hardware, las cuales incluyen las excepciones de los siguientes dispositivos: Teclado y Reloj.

Para lograr el manejo de las interrupciones, se procedió a codificar las clases y estructuras necesarias, las cuales fueron:

- La clase `Pic8259` con capacidad de iniciar el manejo de interrupciones. Esta clase contiene un método que permite configurar correctamente el PIC (por sus siglas en inglés, *Programable Interrup Controler*), para que el procesador pueda reconocer las interrupciones generadas por los distintos dispositivos. El método mencionado anteriormente utiliza una librería que representa una abstracción para el acceso a los puertos, el cual debe realizarse mediante el lenguaje de ensamblador. Dicha librería encapsula estas instrucciones. La siguiente figura muestra la forma en la que fue encapsulada cada una de las operaciones (entrada/salida) de los puertos.

```
#include "lib.h"

/**
 *Función leer de un puerto determinado
 * */
unsigned char inportb (unsigned short _port)
{
    unsigned char rv;
    __asm__ __volatile__ ("inb %1, %0" : "=a" (rv) : "dN" (_port));
    return rv;
}

/**
 *Función para escribir en un puerto determinado
 * */
void outportb (unsigned short _port, unsigned char _data)
{
    __asm__ __volatile__ ("outb %1, %0" : : "dN" (_port), "a" (_data));
}
```

Figura 9 – lib.cpp

Fuente: Elaboración propia.

- La clase `Tdi` representa una estructura llamada Tabla de Descriptores de Interrupciones, la cual contiene la referencia a las funciones manejadoras de cada una de las interrupciones. Dichas funciones fueron codificadas utilizando una combinación del lenguaje de ensamblador y C++, específicamente una función llamada `puenteCpp` para permitir el paso de las funciones en ASM a C++.

- La clase `Mint` posee los métodos necesarios para identificar e indicar el controlador correcto por tipo de interrupción manejada en el sistema. Debido a que en esta etapa se activarían las interrupciones, el dispositivo físico del reloj cada cierto tiempo produciría una interrupción que aún no poseería un manejador, por esto fue necesario colocar en el método `capturarIrq` una instrucción que permitió ignorar dicha interrupción.

- Una interfaz `IControlador`[5] que sería implementada por los controladores de dispositivos y de la cual se creó un arreglo perteneciente a la clase `Mint`, para permitir el polimorfismo. Esto quiere decir que para cada interrupción, se llamaría a la función `procesarInterrupcion` del controlador concreto y estos, a pesar de ser controladores de diferentes dispositivos, podrían manejarse como un solo tipo de dato, así que el código necesario para invocar al controlador sería el mismo sin importar la interrupción de hardware ocurrida. Luego, la lógica para manejarlo reside en cada controlador en

[5] Se decidió utilizar el estándar para el nombrado de interfaces de C#, por lo práctico que resulta.

particular.

- Para el desarrollo del módulo de teclado se siguió un esquema en el cual existe una representación del dispositivo físico, representado por la clase `Teclado`, y una clase que posee la lógica para manipular dicho dispositivo, la cual se encuentra en la clase `ControladorTeclado`, esta última es una implementación de la interfaz `IControlador`.

IV.1.3 PRUEBAS SOBRE LOS ENTORNOS DE DESARROLLO Y DE PRUEBAS.

Para realizar las pruebas unitarias se implementaron las clases que contendrían los métodos necesarios para probar las funcionalidades del módulo correspondiente. Cada una de estas clases posee los métodos `setUP` y `tearDown`, los cuales permiten inicializar y finalizar el ambiente de prueba respectivamente, realizando su ejecución sobre el simulador Bochs.

Además se definió una "región" que fue expandida de acuerdo a una instrucción específica que modificaría la forma de compilación, esta directiva se encuentra en el archivo `makefile` principal. De esta manera puede compilarse o no el núcleo con capacidad para ejecutar las pruebas unitarias (ver Figura 10).

```
#if TEST
    TestVideo video;
    video.setUp();
    video.testVideoImprimirText();
    video.testVideoImprimirColor();
    video.testVideoImprimirUnsignedLong();
    video.testVideoImprimirUnsignedInt();
    video.testVideoImprimirInt();
#endif
```

Figura 10 – Main.cpp. (Sección de las pruebas unitarias).

Fuente: Elaboración propia.

La directiva "`if TEST`" comprueba si al compilador se le a dado la instrucción de compilar en modo de prueba. `TEST` es una variable que se define al momento de ejecutar el compilador g++ mediante una orden de preprocesado, así, la región que se encuentra delimitada por "`#if TEST`" – "`#endif`", puede compilarse o no. La siguiente figura muestra la sección del archivo `makefile` que permite la compilación de las pruebas:

```
PruebasUnitarias: Main2.o Soporte.o Arranque.o Controladores.o
cd PruebasUnitarias/; make;
cd Binario/; $(LD)  -o Vianyx.bin $(OBJS) Pruebas.o -Ttext 0x100000; cp Vianyx.bin /media/floppy

Main2.o: Main.cpp
    $(G++) $(CPPFLAGS) -DTEST -c -o Binario/Main.o $<
```

Figura 11 – makefile del núcleo. (Sección de las pruebas unitarias).

Fuente: Elaboración propia.

Las palabras en rojo forman parte importante del sistema de pruebas:

Main2.o: indica una dependencia para lograr la compilación del sistema, a su vez `Main2.o` necesita el archivo `Main.cpp` (quien es el que contiene el código para ejecutar la prueba).

Pruebas.o: es el código compilado de la clase de prueba que es incluido en el proceso de enlace.

-DTEST: es la directiva que define la variable `TEST` y es por medio de esta que se incluye el código para la prueba.

La ejecución de estas pruebas se encuentra especificada en el Apéndice B – Casos de Pruebas.

IV.1.4 PRUEBAS EN ENTORNO DE PRODUCCIÓN.

Para la realización de estas pruebas se utilizó como entorno de producción un procesador Intel (Pentium IV) de 512Mb de RAM, se configuró la opción de arranque del computador de prueba, para que arrancara desde la unidad de disco extraíble. Se colocó el dispositivo extraíble y se reinició el computador, en este momento el sistema presentó una lista con distintas opciones, las cuales representaban cada una de las posibles versiones del núcleo que se podían ejecutar al ser seleccionadas, esto por medio del gestor de arranque GRUB.

IV.2 ITERACIÓN 2: DESARROLLO DE UN PLANIFICADOR BÁSICO DE PROCESOS, EL CUAL LLEVARÁ EL CONTROL DE LOS PROCESOS EXISTENTES.

IV.2.1 ANÁLISIS Y DISEÑO

En esta fase se realizaron los documentos de visión de los módulos de reloj, procesos, planificador y consola. Cada uno de estos documentos contiene la definición, los casos de uso, los requerimientos funcionales y no funcionales, y los diagramas de casos de uso para dichos módulos además contienen la descripción del elemento a desarrollar, y una breve definición sobre su funcionalidad. Además, se modelaron las clases para esta iteración. Estos documentos se encuentran en el Apéndice A - Documentos De Visión Y Diagrama De Clases.

IV.2.2 DESARROLLO.

El desarrollo y las pruebas para esta fase fueron realizados tomando como base la iteración anterior, es decir, se utilizó el mismo modelo de pruebas unitarias, pero aplicando la lógica correspondiente para probar cada módulo.

IV.2.2.1 MÓDULO 1: RELOJ

Apoyados en la iteración anterior se comenzó la codificación del módulo que sería el encargado del manejo del reloj, luego de crear las clases correspondientes al reloj y a su controlador, y contando con el manejo de las interrupciones, se

procedió a activar la entrada correspondiente en dicho manejador para poder controlar la interrupción que este dispositivo genera y que anteriormente era ignorada. Para ello fue necesario modificar el método `capturarIrq` de la clase `Mint`, eliminando la instrucción que permitía ignorar la interrupción de reloj (dicha instrucción se incluyó durante el desarrollo del módulo de teclado). A continuación se muestra el código:

```
/**
 *Esta función determina el número de irq y busca la función manejadora
 *en la tabla tablaControladores luego lo ejecuta
 */
void Mint::capturarIrq(Registros &registros)
{
    int posicionIrq=registros.numeroInt - 32;
    /*Hago uso de referencias para evitar la notacion ->*/
    IControlador &driver=*tablaControladores[posicionIrq];

        /* Si la interupcion es mayor a 40 enviamos
  * una señal de suprimir interrupcion al picEsclavo */
    if (registros.numeroInt >= 40)
    {
            outportb(0xA0, 0x20);
    }

        /* en este caso enviamos una señal de suprimir interrupcion al picMaestro */
        outportb(0x20, 0x20);

    //ignoramos la interrupcion de reloj por el momento
    //if(posicionIrq!=0)
    //{
        driver.procesarInterrupcion(registros);
    //}
}
```

Figura 12 – MnjInterrupciones.cpp

Fuente: Elaboración propia.

El reloj del sistema genera interrupciones en un intervalo regular de tiempo, esta característica fue usada más adelante para la administración de tiempo de

procesador por *Round Robin*, así que el `ControladorReloj` registra cada interrupción ocurrida como un aumento en una variable llamada `ticks` que posee la clase `Reloj`.

Además dicha clase posee una relación con el planificador, esta relación no es directa, en lugar de ello, el `Reloj` posee un método que recibe como tipo de dato la interfaz `IPlanificador`; de esta manera se mantiene un bajo acoplamiento entre las clases `Reloj` y `Planificador`. La siguiente figura muestra la relación indicada anteriormente:

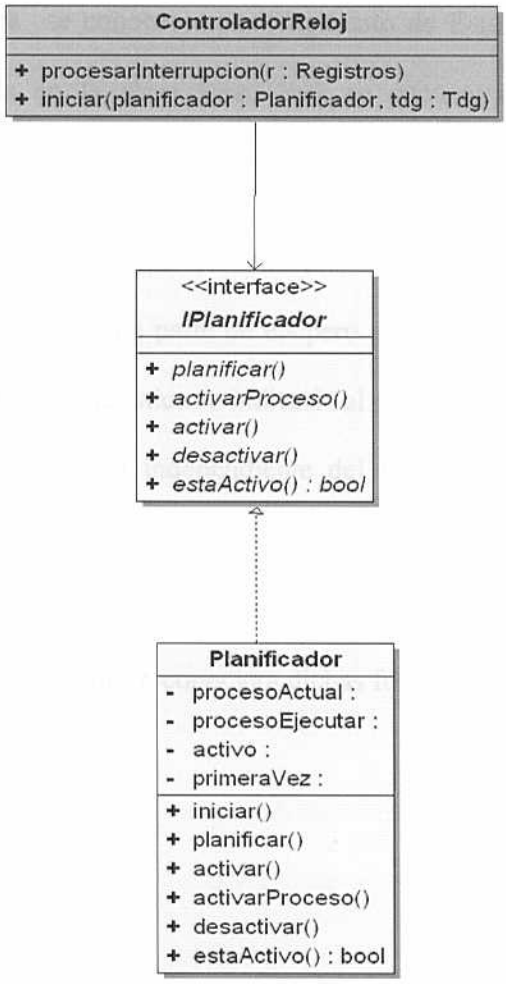

Figura 13 – Diagrama de clases Relación entre Reloj Planificador.

Fuente: Elaboración propia.

IV.2.2.2 MÓDULO 2: PROCESOS

Para crear éste módulo se procedió a codificar la clase que contendría la información necesaria para el control de los procesos que el sistema manejaría. La clase `Proceso`, representa lo que comúnmente se conoce como "Bloque de Control de Proceso".

Además, fue necesario crear una clase más que representara una porción de memoria en donde se almacenaría la información necesaria para el cambio de proceso, esta estructura se conoce como "Segmento de Estado de Tarea" y se encuentra representada en la clase `Tss`. Ésta se encuentra contenida en la descripción del proceso, es decir, forma parte de la clase `Proceso`.

El código ejecutable del proceso se encontraría en el mismo espacio de direcciones que el núcleo - forma parte de él- pero gracias a la clase `Proceso`, fue posible aislar ciertas características e indicarle al procesador que debería tratar a ese código como una unidad independiente del núcleo. Así se crearon dos funciones con el código que representan la parte ejecutable de los procesos y luego el `ControladorProcesos` se encargaría de inicializar los campos necesarios para que el procesador reconociera dichas funciones como procesos.

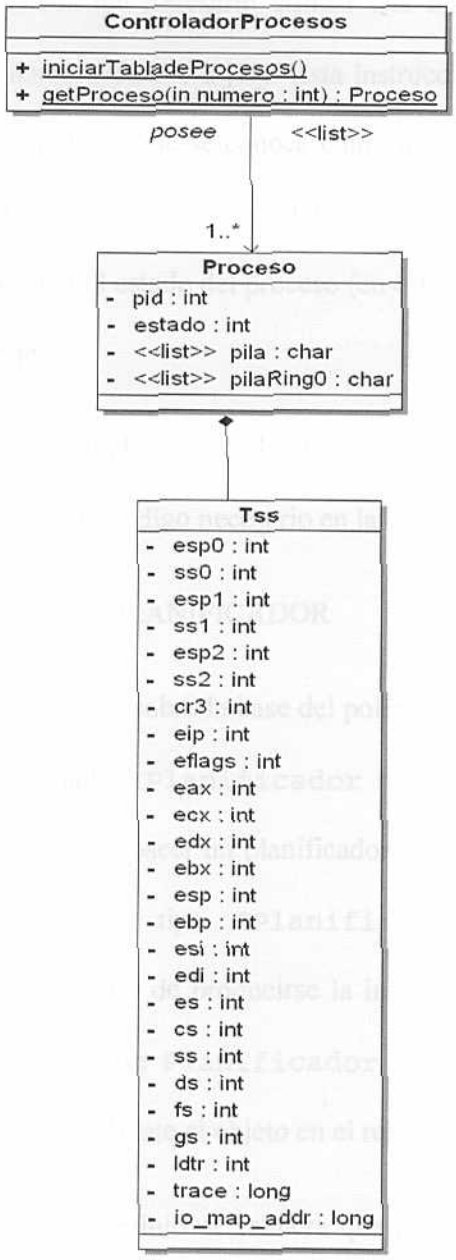

Figura 14 – Diagrama de clases Procesos.

Fuente: Elaboración propia.

Para el cambio de tareas fue necesario utilizar una instrucción en lenguaje de ensamblador, dicha instrucción fue `ljmp`. Esta instrucción nos permitió cambiar de una tarea a otra usando lo que se conoce como multitarea por hardware, este tipo de multitarea se logra con asistencia del equipo físico, el cual automáticamente preserva el estado del proceso (en este caso en la estructura `Tss` indicada anteriormente).

En esta etapa aún no se contaba con el planificador, así que para lograr el cambio entre procesos, se incluyó el código necesario en la clase `ControladorReloj`.

IV.2.2.3 MÓDULO 3: PLANIFICADOR

El planificador fue construido sobre la base del polimorfismo. Para ello se definió una clase abstracta llamada `IPlanificador` la cual define algunos de los métodos generales que debe poseer un planificador en Vianyx. Luego, al reloj se le agregó un atributo de tipo `IPlanificador` para desacoplar la implementación y al momento de producirse la inicialización de los objetos, se instancia un objeto de la clase `Planificador` específica y se establece por medio del método correspondiente el objeto en el reloj.

La interfaz mencionada se definió de manera que poseyera un método llamado `planificar`, es allí donde se colocó la lógica necesaria para realizar la planificación. Gracias a este modelo de programación orientada a objetos, las posibles modificaciones pueden ser más sencillas de implementar. Por ejemplo, es posible definir un nuevo planificador que reutilice características de alguno ya existente.

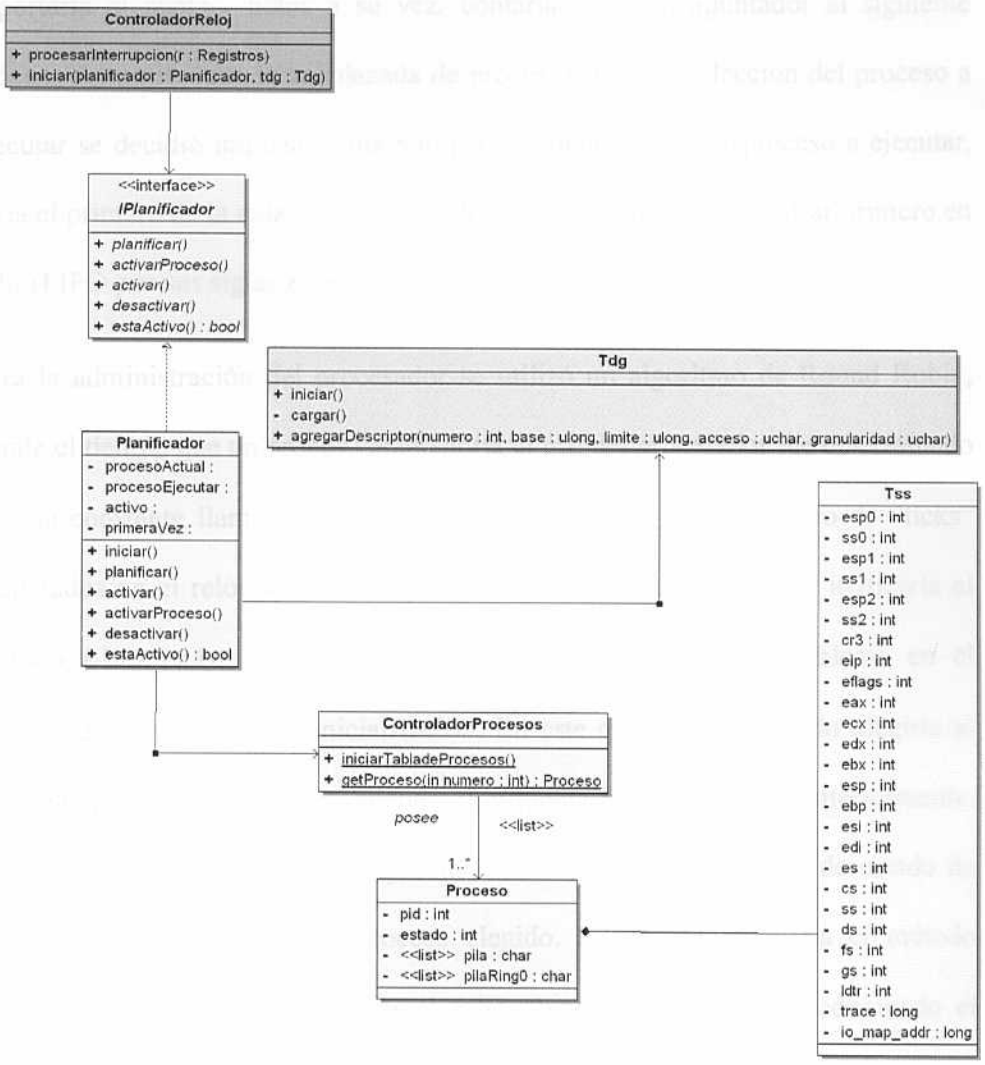

Figura 15 - Diagrama de clases del Planificador.

Fuente: Elaboración propia.

La figura anterior muestra las relaciones entre las distintas clases que conforman la parte fundamental del esquema de planificación. En ella se puede observar la clase `Planificador`, implementación de la interfaz `IPlanificador`, sus respectivos métodos y las relaciones que esta posee.

Se creó una estructura estática (arreglo de procesos) contenida en la clase `ControladorProcesos`, la cual contendría el máximo de procesos que

soportaría el núcleo. Estos a su vez, contarían con un apuntador al siguiente proceso formando una lista enlazada de procesos. Para la selección del proceso a ejecutar se decidió implantar una política en donde el primer proceso a ejecutar, sería el primero de la cola; en lo que se conoce como primero en entrar primero en salir (FIFO por sus siglas en ingles).

Para la administración del procesador se utilizó un algoritmo de Round Robin, donde el tiempo que un proceso mantendría el uso del procesador fue determinado en una constante llamada `QUANTUM`, la cual de ser igual al número de "ticks" registrados en el reloj activaría al planificador, en ese momento, se invocaría al método `planificar` del objeto `Planificador` que se estableció en el `reloj` durante la fase de inicialización. En este caso, dicho método elegiría al siguiente proceso de la cola siguiendo el esquema FIFO explicado anteriormente. Luego de conocer el proceso a ejecutar, se cambiaría el segmento de estado de tarea indicando el valor del proceso elegido. Luego, se invocaría el método `activarProceso` para realizar el salto al segmento elegido produciendo el cambio de contexto.

Las dos estrategias mencionadas anteriormente forman la base de la planificación en el sistema.

IV.2.2.4 MÓDULO 4: CONSOLA

Para el desarrollo de la interfaz entre el usuario y el sistema, se eligió un esquema sencillo, el sistema presenta un menú con un conjunto de opciones enumeradas, donde el usuario puede seleccionar la opción de su preferencia. La capacidad de determinar la opción a ejecutar fue añadida al módulo de teclado, por lo que fue

necesario agregar dicha funcionalidad en el método encargado de interpretar la tecla pulsada. En la siguiente figura se puede observar el fragmento de código necesario para lograr dicha capacidad.

```
/**
 * Esta función es utilizada para procesar la interrupción producida por el teclado
 * al momento de pulsar una tecla.
 *\param r clase que contiene los valores de los registros del sistema
 *\return void
 */
void ControladorTeclado::procesarInterrupcion(Registros &r)
{
    char c;
    int i=0;
    // Lee del buffer de teclado
    c=teclado.convertirAscci(teclado.getScanCode());
    bufferTeclado[i%255]=c;
    video.escribir(c);

    if(c=='1') {
        planificador->activar();
    }
    if(c=='2') {
        planificador->desactivar();
    }
    if(c=='3') {
        ControladorProcesos::mostrarTablaprocesos();
    }
}
```

Figura 16 – Teclado.cpp (Módulo de consola)

Fuente: Elaboración propia.

En la figura mostrada arriba se puede observar parte de la codificación de la clase `Teclado`, donde se pueden observar las sentencias que se encargan de evaluar la respectiva opción seleccionada por el usuario.

IV.2.3 PRUEBAS SOBRE LOS ENTORNOS DE DESARROLLO Y DE PRUEBAS.

Para esta iteración, se utilizaron pruebas unitarias siguiendo el esquema de la iteración anterior, exceptuando el módulo de procesos, y de consola. El resto de los módulos (reloj y planificador) poseen sus respectivas pruebas, descritas en el Apéndice B – Casos de Pruebas.

IV.2.4 PRUEBAS EN ENTORNO DE PRODUCCIÓN.

Para la realización de estas pruebas se utilizó como entorno de producción un procesador Intel (Pentium IV) de 512Mb de RAM, se configuró la opción de arranque de el computador de prueba, para que arrancara desde la unidad de disco extraíble. Se colocó el dispositivo extraíble y se reinició el computador, en este momento el sistema presento una lista con distintas opciones, las cuales representaban cada una de las posibles versiones del núcleo que se podían ejecutar al ser seleccionadas, esto por medio del gestor de arranque GRUB.

Para esta fase, se utilizaron los mismos mecanismos de pruebas de la iteración anterior.

IV.3 ITERACIÓN 3: ESTABLECIMIENTO DE LA PLATAFORMA EDUCATIVA.

Para establecer un ambiente de trabajo que sirviera de apoyo a la enseñanza de los principios básicos de los Sistemas Operativos, se eligió una plataforma que brindará las características que se necesitaban para facilitar la enseñanza y la

distribución de la información.

Dicha plataforma debía ser sencilla y brindar aplicaciones simples tanto para administradores como para los visitantes.

IV.3.1 ANÁLISIS Y DISEÑO

Para el entorno Web se tomaron en cuenta dos plataformas, Moodle[6] y Dokeos[14], para las cuales se evaluó la sencillez de manejo y facilidad para el aprendizaje. Moodle es una herramienta muy completa que tiene muchas actividades y lugares de acceso, pero, no es tan sencilla de manejar como Dokeos. También se presentó el hecho de que Dokeos es una herramienta conocida por el equipo, ya que anteriormente se había trabajado con la herramienta y esto facilitó su aprendizaje y comprensión.

Dokeos permite llevar un seguimiento de visitas por parte de los participantes, así como también permite subir archivos, modificarlos y eliminarlos, colocar vínculos en los archivos, colocar evaluaciones, todo esto de forma muy sencilla, lo cual ayuda con las prácticas, posee medios de conversación en linea (*Chats)* y lugares de interacción entre el administrador y los miembros de la materia. En la siguiente tabla se muestran los criterios que se tomaron en cuenta para la elección de la herramienta.

[6] Es un sistema de gestión de cursos de libre distribución que ayuda a los educadores a crear comunidades de aprendizaje en línea.

Criterio	Dokeos	Moodle
Software Libre	Sí.	Sí.
Conocimiento Previo	Sí.	No.
Interfaz	Sencilla.	Compleja.
Aplicaciones	Predeterminadas.	Pueden añadirse aplicaciones nuevas.

Tabla 1 - Criterios de elección de herramienta Web.

Fuente: Creación propia

En la tabla anterior se puede observar que se evaluaron criterios básicos, si era un Software Libre (gratuito), el conocimiento sobre la herramienta fue uno de los puntos más significativos porque permitía tener más confianza sobre su uso, sobre la interfaz se evaluó si era fácil de utilizar o fácil de navegar, de las aplicaciones se evaluó si tenía las necesarias para poder satisfacer el aprendizaje en línea (la idea era utilizar las ya existentes para disminuir el trabajo, no es necesario crear aplicaciones).

Moodle basa su modelo pedagógico en el constructivismo social, esto es, en el establecimiento de comunidades alrededor de un tema que realizan actividades, reflexión crítica, etc. Esto marca profundamente su organización e interfaz, construida alrededor de 3 modelos de interacción en línea:

- *Weekly*, en la que toda la interfaz gira alrededor de la asignación de actividades semanales.
- *Topics*, en la que queda organizada en base a los temas propuestos en el curso.
- *Social*, en la que el eje central del curso pasa a ser un foro de discusión.

Por otra parte, Dokeos posee un modelo distinto. La interfaz es mucho más amigable debido a sus iconos y espacios, se organiza en base al concepto de curso como agrupación de distintos tipos de recursos: contenido, foro, auto-evaluaciones, descargas, etc.

Se decidió implementar Dokeos debido a la sencillez para dividir el curso, lo cual facilita su entendimiento y su uso, además, Dokeos permite crear un curso con una modalidad de auto-estudio, con elementos de colaboración y comunicación para el apoyo al estudiante.

IV.3.2 DESARROLLO.

Luego de elegir la plataforma a utilizar, se procedió al establecimiento de la misma, se creó un nuevo curso que tiene por nombre "Sistemas Operativos - Vianyx", dicho curso puede encontrarse en la siguiente dirección Web: http://campus.dokeos.com/index.php?language=spanish. Para poder revisar el contenido del curso debe realizarse un registro en el curso, los privilegios por defecto son los de "Estudiante", lo cual permite ver la descripción del curso, revisar los archivos en el área de documentos, descargarlos, utilizar las conversaciones en línea, foros, y realizar evaluaciones publicadas por el profesor o administrador del curso.

El área de documentos se encuentra dividida de la siguiente manera:

- **Código Fuente de Vianyx:** contiene el código fuente del sistema, éste se encuentra separado por carpetas, cada una con el nombre de los módulos del sistema.

- **Documentación del Código:** contiene los archivos generados en Doxygen[7], se encuentra dividido el código por clases, con los nombres de cada una de las funciones y los comentarios referentes a las mismas. También posee los diagramas de herencia de cada una de las clases y objetos del código.
- **Prácticas:** contiene las prácticas propuestas para la enseñanza de los tópicos principales de la materia de Sistemas Operativos. Dichas prácticas fueron realizadas en función al código desarrollado para el sistema.
- **Solución de las Prácticas:** contiene una solución posible para las prácticas propuestas.
- **Tutoriales:** contiene un tutorial de los primeros pasos a seguir para desarrollar un sistema operativo básico.

IV.3.3 PRUEBAS SOBRE LOS ENTORNOS DE DESARROLLO Y DE PRUEBAS.

Para esta iteración no se realizaron pruebas en entorno de desarrollo ni en el entorno de pruebas, debido a que se utilizó un sistema existente, el cual no necesitaba este tipo de pruebas.

IV.3.4 PRUEBAS EN ENTORNO DE PRODUCCIÓN.

En esta etapa se crearon dos cuentas, una que tenía todos los privilegios con una vista de administrador y otra limitada con una vista de estudiante. Un integrante

[7] Es un Generador de documentación para C++, C, Java, Objective-C, Python, IDL (versiones Corba y Microsoft) y en cierta medida para PHP, C# y D. Dado que es fácilmente adaptable, funciona en la mayoría de sistemas Unix así como en Windows y Mac OS X.

del equipo se encargó de colocar la distribución del curso en la plataforma, subir archivos de pruebas, colocar descripciones y documentos necesarios para el despliegue del curso. El otro integrante, con la cuenta de estudiante, se encargó de probar la plataforma ya estructurada, bajar archivos, revisar los documentos, bajar el código fuente del sistema Vianyx, verificar que no se dañaran los archivos y que no pudiera modificar los archivos si no tenía los privilegios para ello.

También se colocaron evaluaciones y encuestas para verificar la interacción de los participantes del curso, esto para apoyar a los profesores o preparadores en llevar un orden de evaluaciones y prácticas, y una guía para las evaluaciones de las prácticas propuestas.

CAPÍTULO V. RESULTADOS

El sistema fue desarrollado siguiendo la metodología de trabajo que la empresa Softclear de Venezuela C.A propuso, lo cual permitió tener un desarrollo ordenado y exitoso.

En la etapa de análisis de cada uno de las iteraciones se obtuvieron los siguientes artefactos:

V.1 Iteración 1. Desarrollo de un núcleo con características básicas.

V.1.1 Módulo 1: Arranque

En el módulo de arranque se obtuvo, un entorno configurable, capaz de cargar al núcleo indicado en la memoria del computador y otorgarle el control para que éste realizara sus operaciones. Para comprobar el correcto funcionamiento de este módulo se realizó un caso de prueba descrito en el Apéndice B – Casos de Pruebas.

V.1.2 Módulo 2: Vídeo

En este módulo se obtuvo la clase `Video` encargada de manejar la pantalla del computador, esta clase permite mostrar mensajes en la pantalla con soporte para distintos tipos de datos, permite además, limpiar la pantalla y decidir el color del fondo y/o texto en el que se desea mostrar la información. El correcto funcionamiento de este módulo fue comprobado mediante el caso de prueba descrito en el Apéndice B – Casos de Pruebas.

V.1.3 Módulo 3: Interrupciones

Se obtuvieron las clases para el manejo de excepciones, interrupciones, y protección de la memoria. Dichas clases en conjunto proveen al sistema una forma básica de manejo de interrupciones (su reconocimiento) y control de memoria. Para probar este módulo, se codificaron pruebas que generarían diversas excepciones y se observó la respuesta del sistema. Dicha prueba se encuentra descrita en el Apéndice B – Casos de Pruebas.

V.1.4 Módulo 4: Teclado

Se obtuvo la clase que representa al dispositivo físico de teclado y una clase que se encarga de procesar la interrupción llamada `ControladorTeclado`. Estas clases en conjunto, reconocen la pulsación de una tecla en el dispositivo y procesan la interrupción ocurrida con dicha pulsación; permitiendo la interacción entre el sistema y el usuario. Al igual que los módulos anteriores para comprobar el funcionamiento de este módulo, se utilizó un caso de prueba descrito en el Apéndice B – Casos de Pruebas.

V.2 Iteración 2. Desarrollo de un planificador básico de procesos, que llevará el control de los procesos existentes.

V.2.1 Módulo 1: Reloj

Se obtuvo la clase que representa al dispositivo físico de reloj, y una clase que se

encargaría de procesar la interrupción llamada `ControladorReloj`, estas clases en conjunto reconocen la activación periódica de la interrupción del reloj del sistema Este módulo sirvió de base para crear el esquema de planificación que el sistema tendría. La prueba para este módulo se encuentra descrita en el Apéndice B – Casos de Pruebas.

V.2.2 Módulo 2: Procesos

En este módulo se obtuvieron las clases que representarían a los procesos del sistema, se encuentra en las clases `TSS` y `Proceso`. Además se obtuvo una clase que se encargaría de la manipulación de los procesos, es decir, de su creación y su cambio de estado, dicha clase tiene por nombre `ControladorProcesos`.

V.2.3 Módulo 3: Planificador

En esta etapa se obtuvo una serie de clases relacionadas entre si, que representan al sistema de planificación en general, las cuales son `IPlanificador Procesos`, `ControladorProcesos` y `PlanificadorVianyx`.

La relación que existe entre dichas clases, permite crear otras implementaciones de un planificador (totalmente nuevo o basado en la herencia) y ser cambiado fácilmente dentro del núcleo. Al igual que los módulos anteriores, para comprobar el funcionamiento de este módulo se utilizó un caso de prueba

V.2.4 Módulo 3: Consola

Este módulo representa la parte final del sistema a nivel funcional. El resultado obtenido fue un sistema capaz de responder a determinadas órdenes del usuario,

dichas órdenes serían dadas por medio del dispositivo de teclado.

Al igual que los módulos anteriores, para comprobar el funcionamiento de este módulo se utilizó un caso de prueba descrito en el Apéndice B – Casos de Pruebas.

V.3 Iteración 3. Establecimiento de la plataforma educacional

Se generaron diagramas de clases y la documentación del sistema que contiene la descripción de las clases y las funcionalidades existentes en el sistema. Esto provee una ventaja ante otros sistemas de este tipo, que no poseen esta documentación.

Se estableció un entorno Web, el cual permite obtener la información básica para facilitar el aprendizaje en el estudio de los Sistemas Operativos, en dicho entorno, pueden conseguirse la documentación del sistema, el código fuente separado por módulos, la guía de cómo desarrollar un núcleo básico (abarca hasta el planificador de procesos), las prácticas propuestas para dicho sistema y la solución de las mismas. Dicho entorno sirve también como plataforma iterativa entre el profesor y sus alumnos, permitiendo que los alumnos puedan consultar fácilmente, aclaren sus dudas sobre el tema y que el profesor pueda ver los progresos, dificultades y posibles mejoras en el sistema de enseñanza de los Sistemas Operativos.

CONCLUSIONES

Durante la realización del presente Trabajo Especial de Grado se emplearon conocimientos adquiridos durante la formación académica, con la finalidad de apoyar al reforzamiento práctico del curso de Sistemas Operativos. Con este fin, se creó un gestor de tareas sencillo en cuanto a su estructura se refiere, fácil de manipular, modificar, y principalmente, de entender.

El gestor de arranque GRUB demostró ser una herramienta configurable, simple y fácil de adaptar al sistema, con capacidad para reconocer múltiples formatos ejecutables y sistemas de archivos; su documentación es amplia y fácil de encontrar. Puede explorarse más en su configuración y capacidades, ya que sólo se utilizaron las características básicas para este desarrollo.

Una de las pocas complicaciones que se encontraron al momento de utilizar esta herramienta, fue la forma de creación de una imagen de GRUB para ser utilizada en el floppy, ya que ésta varía según la distribución de Linux que se utilice. Debe tomarse en cuenta que la versión del cargador podría descontinuarse con el tiempo.

Durante el desarrollo del núcleo básico, fue necesario aplicar conocimientos de lenguaje de bajo nivel y funcionamiento de la arquitectura IA32, así como la compresión de los dispositivos de hardware, específicamente, el teclado y el reloj, junto con el reconocimiento y procesamiento de las interrupciones.

Este núcleo posee la virtud de ser sencillo, debido a que se limitó a incluir lo necesario para que el planificador pudiera ejecutarse. Es por ello, que podrían

ampliarse las capacidades agregando ciertas características más avanzadas, tales como: mejor manejo de memoria, manejo de archivos y capacidad para mostrar gráficos

Por su parte, el planificador del sistema, al presentar un módulo programado usando la orientación a objetos, posee facilidad para la extensión de las funcionalidades y la reutilización del código. Incluso, es posible cambiar fácilmente la implementación del mismo, debido a que todo planificador, que sea desarrollado para ejecutarse en el núcleo de Vianyx, debe implementar la interfaz IPlanificador. Estas características, permiten que dicho planificador pueda usarse para realizar prácticas relacionadas con las estrategias de planificación estudiadas en la teoría, de forma más cómoda que con otros sistemas con el mismo fin, sumado a la ventaja que provee realizar dichas prácticas sobre un sistema bien documentado.

Sin embargo, el uso de conceptos de bajo nivel para el manejo de interrupciones y la falta de librerías que faciliten la extensión del planificador, podrían complicar ligeramente su extensión.

Por otro lado, la decisión de implementar el algoritmo de planificación tipo FIFO se basó en el hecho de que éste algoritmo es simple de implementar. No obstante, es su simplicidad la que produce que, el solo uso de esta estrategia, no proporcione lo suficiente para profundizar en el tema de planificación de procesos, por lo que se sugiere combinarse con otras estrategias, tales como colas por prioridad, o en este caso, Round Robin.

De hecho, el algoritmo de administración de tiempo de procesador Round Robin,

es ampliamente utilizado por muchos sistemas operativos y es la base para la multitarea en sistemas mono-procesador, esto motivó a que dicho algoritmo se incluyera para que, junto al algoritmo FIFO, formaran parte de la estrategia de planificación del sistema Vianyx.

Las prácticas propuestas poseen claridad en los objetivos, y vienen acompañadas de soluciones propuestas, esto último, resulta de gran utilidad para el desarrollo y evaluación de las mismas.

Ahora bien, establecer un ambiente de aprendizaje en línea sobre la herramienta, permitió centralizar la información y dar una mayor facilidad de acceso. Dokeos ofrece una organización de documentos, herramientas para la gestión del sitio Web, posibilidad de realizar respaldo del curso y una capacidad de almacenamiento razonable, además de características más avanzadas que pueden ser exploradas en el futuro.

Finalmente, el sistema ofrece sencillez, documentación, modularidad y una metodología clara para que, de esta forma, los interesados en la materia puedan estudiar con detalle su funcionamiento. Entre otras ventajas que provee, es que se encuentra en español, tiene la posibilidad de ampliarse y se adapta al desarrollo y entendimiento de los conceptos básicos.

No obstante, cabe destacar, que el estudio del funcionamiento de los Sistemas Operativos no es una tarea fácil, y el nivel de conocimientos necesarios, tanto teóricos como de programación, es alto.

RECOMENDACIONES

- Para una mejor comprensión y utilización del sistema, se recomienda leer la guía de desarrollo que se encuentra en el sistema Web. Ésta proporcionará información sobre los temas importantes del desarrollo, a la vez que sirve de guía teórica para la creación de este tipo de sistemas.

- El proyecto sólo cubre una serie de conceptos básicos, a saber: manejo de interrupciones, control básico de dispositivos y planificación de procesos. Luego, se recomienda la inclusión de las siguientes funcionalidades y características al Sistema, en aras de apoyar de mejor forma a los usuarios del mismo:

    - Ampliar el manejo de los dispositivos, como por ejemplo, la capacidad del dispositivo teclado, esto proporcionaría una mayor interactividad entre el usuario y el sistema. El sistema actual cubre lo necesario para reconocer las entradas de usuario.
    - Incluir el manejo de memoria, el cual es un tema bastante estudiado en los Sistemas Operativos. Se recomienda incluir paginación y funciones de localización y liberación de memoria.
    - Incluir el manejo de sistemas de archivos. Uno de los temas más complejos, tanto en sus fundamentos teóricos como prácticos, es precisamente el manejo de archivos; este tópico es muy importante para los Sistemas Operativos.

- Incluir el uso de mecanismos de sincronización, exclusión mutua y manejo de interbloqueos.
- Ampliar las capacidades de la Interfaz Usuario-Computador. Se recomienda incluir nuevas capacidades de comunicación y respuesta entre el usuario y el computador.
- La inclusión de nuevas prácticas y actividades sobre el sistema. Se sugiere seguir la investigación y proponer nuevas actividades, a medida que el sistema vaya evolucionando, para mantener el espíritu creativo e investigativo de aquellos que usen el mismo como guía para el aprendizaje.

- Para la plataforma educativa se sugiere:
  - Desarrollar más tutoriales acerca del desarrollo de los Sistemas Operativos, los cuales deberán colocarse dentro del apartado de "Tutoriales" del curso en línea.
  - Se espera que en futuros desarrollos puedan habilitarse otras de las aplicaciones que ofrece el entorno Web, ya que en este trabajo sólo se abarcaron las descritas en el capitulo de desarrollo.

BIBLIOGRAFÍA

LIBROS

[1] Liberty, J. y Horvath, D. B. (2001). *Aprendiendo C++ para Linux* (1ª. ed.). México: Prentice hall.

[2] Pressman, R. (2006). *Ingenieria del Software, Un enfóque práctico* (6a. ed.). México: McGraw-Hill.

[3] Stallings, W. (2001). *Sistemas Operativos.* (4ª. ed.). Madrid: Prentice Hall.

[4] Tanenbaum, A. (1998). *Sistemas Operativos Diseño e Implementación.* (2ª. ed.). México: Prentice Hall.

[5] Carretero, J., Anasagasti, P., Carballeira, F. y Costoya, F. (2001). *Sistemas Operativos Una Visión Aplicada* (1a. ed.). Madrid: McGraw-Hill.

[6] Silberschatz, A., Galvin, P. y Gagne, G. (2006). *Fundamentos de Sistemas Operativos* (7a. ed.). Madrid: McGraw-Hill.

REFERENCIAS ELECTRÓNICAS

[7] Wikipedia (2007, Noviembre). *IA-32* [en línea]. Disponible en: http://es.wikipedia.org/wiki/IA-32 [2007, Noviembre].

[8] GCC, the GNU Compiler Collection (2007, Diciembre). *GCC* [en línea]. Disponible en: http://gcc.gnu.org/ [2007, Diciembre].

[9] Bochs (2007, Septiembre). *Welcome to the Bochs IA-32 Emulator Project* [en línea]. Disponible en:
http://bochs.sourceforge.net/ [2007, Diciembre].

[10] Sourceforge (2007, Noviembre). *The Netwide Assembler* [en línea]. Disponible en:
http://sourceforge.net/projects/nasm [2007, Diciembre].

[11] GNU Project (2008, Julio). *GNU GRUB* [en línea].
Disponible en: http://www.gnu.org/software/grub/ [2008, Julio].

[12] Administración de Linux: Una guía básica (2008, Enero) *Arranque del sistema* [en línea]. Disponible en:
http://www.bdat.net/documentos/arranque/x32.html
[2008, Enero].

[13] Soriano, F. (2007, Febrero). *Desarrollo de Software Iterativo e Incremental* [en línea]. Disponible en:
http://fernandosoriano.com.ar/?p=14 [2007, Noviembre].

[14] Wikipedia (2008, Enero). *Dokeos* [en línea]. Disponible en:
http://es.wikipedia.org/wiki/Dokeos [2008, Enero].

[15] Real Academia Española (2008, Febrero). *Mono* [en línea].
Disponible en:
http://buscon.rae.es/draeI/SrvltConsulta?TIPO_BUS=3&LEMA=mono [2008, Febrero].

[16] Kioskea (2008, Enero). *El Procesador* [en línea].
Disponible en: http://es.kioskea.net/pc/processeur.php3 [2008, Enero].

[17] Trevinca (2008, Febrero) *Espera infinita o inanición de procesos* [en línea]. Disponible en:
http://trevinca.ei.uvigo.es/~formella/doc/cd05/node61.html
[2008, Febrero].

[18] Redhat (2008, Enero). *Ld Linker* [en línea]. Disponible en:
http://www.redhat.com/docs/manuals/enterprise/RHEL-3-Manual/devtools/s1-ld-gnu.html [2008, Enero].

PÁGINAS WEB

- BonaFide, OS Development. Disponible en: http://www.osdever.net/ (Enero, 2008)
- Bran's Kernel Development, A tutorial on writing kernels
  Version 1.0. Disponible en: http://www.osdever.net/bkerndev/Docs/intro.htm (Enero, 2008)
- James Molloy, JamesM's kernel development tutorials. Disponible en: www.jamesmolloy.co.uk (Enero, 2008)
- Kelpi, Share your Source Code. Disponible en: http://kelpi.com/ (Febrero, 2008)
- Nondot.org, Operating System. Disponible en:
  http://www.nondot.org/sabre/os/articles (Enero, 2008)
- Scribd, Writing a Kernel in C++. Disponible en:
  http://www.scribd.com/doc/2187324/Writing-a-Kernel-in-C (Febrero, 2008)
- Sigops, OS Development Resources. Disponible en:
  http://www.acm.uiuc.edu/sigops/rsrc/ (Enero, 2008)
- Skelix, Skelix OS Tutorial. Disponible en:
  http://en.skelix.org/skelixos/tutorial04.php (Enero, 2008)
- supportWeb, School of Computer Science. Disponible en:
  http://supportweb.cs.bham.ac.uk/ (Marzo, 2008)
- Wikilearning, comunidades de wikis libres para aprender. Disponible en:
  http://www.wikilearning.com (Enero, 2008)

- Zator Systems, Tecnología de la información para el conocimiento. Disponible en: http://www.zator.es/ (Marzo, 2008)

GLOSARIO

ASCII: Acrónimo inglés de *American Standard Code for Information Interchange* - (Código Estadounidense Estándar para el Intercambio de Información), pronunciado generalmente [áski], es un código de caracteres basado en el alfabeto latino tal como se usa en inglés moderno y en otras lenguas occidentales.

Cambio de contexto: Es el conjunto de las siguientes operaciones: se salva el estado del proceso en el correspondiente BCP, se pasa a ejecutar la rutina de tratamiento de interrupción del sistema operativo.

Cambio de Modo: Operación de hardware que hace que el procesador ejecute en un modo diferente (núcleo o usuario). Cuando el modo cambia de usuario a núcleo, se guardan el contador de programa, palabra de estado del procesador y otros registros. Cuando el modo cambia de núcleo a usuario, se restauran.

GCC: GNU *Compiler Collection* -Colección de Compiladores GNU- es un conjunto de compiladores creados por el proyecto GNU. GCC es software libre se distribuye bajo la licencia GPL. Estos compiladores se consideran estándar para los sistemas operativos derivados de UNIX, de código abierto o también de propietarios, como Mac OS X. GCC requiere el conjunto de aplicaciones conocido como binutils para realizar tareas como identificar archivos objeto u obtener su tamaño para copiarlos, traducirlos o crear listas, enlazarlos, o quitarles símbolos innecesarios.

Inanición: Condición en la que un proceso se retarda indefinidamente porque

otros procesos siempre tienen la preferencia.

Mono Procesador: Un sistema mono-procesador es aquel que tiene solo un procesador.

NASM: El *Netwide Assembler* o NASM, es un ensamblador libre para la plataforma Intel x86. Puede ser usado para escribir programas tanto de 16 bits como de 32 bits (IA-32). En el NASM, si se usan las bibliotecas correctas, los programas de 32 bits se pueden escribir de una manera tal para que sean portables entre cualquier sistema operativo x86 de 32 bits.

Periférico: Se denominan periféricos tanto a las unidades o dispositivos a través de los cuales la computadora se comunica con el mundo exterior, como a los sistemas que almacenan o archivan la información, sirviendo de memoria auxiliar de la memoria principal.

PIC (*Programmable Interrupt Controller*): Es el controlador programable de interrupciones usado especialmente en los IBM PCs.

Es un dispositivo que permite asignar niveles de prioridad a sus interrupciones de salida. Cuando el dispositivo tiene múltiples interrupciones de salida para hacer, las realiza en un orden dependiendo de una relativa prioridad. Se llama programable porque si recibe múltiples y simultáneas interrupciones, puede ordenarlas y ejecutarlas dependiendo de una prioridad programada por el sistema operativo, y así dárselas al CPU una a una.

Uno de los más conocidos PICs es el 8259A, que fue incluido en las PCs x86. En las actuales PCs x86, no es incluido como un chip separado sino que su

funcionalidad la realiza una parte del puente sur de la placa madre.

RPM (*Red hat Package Manager*): RPM es el gestor de paquetes de Red Hat. Aunque aparece Red Hat en su nombre, la intención es que sea un sistema de empaquetado abierto y disponible para el uso de cualquiera. Permite a los usuarios tomar el código fuente (*source code*) y empaquetarlo en forma de fuentes y binaria de forma que los ficheros binarios sean fácilmente instalables y rastreables, y las fuentes puedan ser reconstruidas con facilidad.

RTTI (*RunTime Type Information*): Estas siglas se refieren al mecanismo C++ que permite determinar en tiempo de ejecución el tipo de un objeto. Es una parte importante del mecanismo de comprobación de tipos del lenguaje.

Segmentación: División de un programa o aplicación en segmentos como parte del esquema de memoria virtual.

Software Libre: Es el software que una vez obtenido por el usuario puede ser usado, copiado, estudiado, modificado y distribuido libremente.

SUSE: SUSE Linux es una de las más conocidas distribuciones Linux existentes a nivel mundial. Entre las principales virtudes de esta distribución se encuentra el que sea una de las más sencillas de instalar y administrar, ya que cuenta con varios asistentes gráficos para completar diversas tareas.

Vianyx: Sistema Operativo con enfoque educacional para el estudio de los conceptos básicos acerca de los Sistemas Operativos. Escrito principalmente en C++ y compuesto de un núcleo básico y un planificador de procesos (en su primera versión).