UNIDAD I TRADUCTORES DE BAJO NIVEL

La comunicación en lenguaje de máquina es particular de cada procesador que se usa, y programar en este lenguaje es muy difícil y tedioso, por lo que se empezó a buscar mejores medios de comunicación con ésta.

A principios de la década de 1950, y con el fin de facilitar la labor de los programadores, se desarrollaron códigos mnemotécnicos para las operaciones y direcciones simbólicas. Uno de los primeros pasos para mejorar el proceso de preparación de programas fue sustituir los códigos de operación numéricos del lenguaje de máquina por símbolos alfabéticos, que conforman un lenguaje mnemotécnico. Todas las computadoras actuales tienen códigos mnemotécnicos aunque, naturalmente, los símbolos que se usan varían en ¡as diferentes marcas y modelos. La computadora sigue utilizando el lenguaje de máquina para procesar los datos, pero los programas ensambladores traducen antes los símbolos de código de operación especificados a sus equivalentes en lenguaje de máquina. Los lenguajes ensambladores tienen ventajas sobre los lenguajes de máquina- Ahorran tiempo y requieren menos atención a detalles. Se incurren en menos errores y los que se cometen son más fáciles de localizar Además, los programas en lenguaje ensamblador son más fáciles de modificar que los programas en lenguaje de máquina, Pero existen limitaciones. La codificación en lenguaje ensamblador es todavía un proceso lento. Además, una desventaja importante de estos lenguajes es que tienen una orientación a la máquina. Es decir, están diseñados para la marca y modelo específico de procesador que se utiliza.

En el principio de la computación este era el lenguaje que tenía que "hablar" el ser humano con la computadora y consistía en insertar en un tablero miles de conexiones y alambres y encender y apagar interruptores. Aunque en la actualidad ya no se emplea, es importante reconocer que ya no es necesario que nos comuniquemos en este lenguaje de "unos" y "ceros", pero es el que internamente una computadora reconoce o "habla".

1.Introducción a los traductores de bajo nivel

INTRODUCCIÓN

Los traductores son programas que permiten pasar de un programa fuente a un programa objeto.

En los lenguajes de bajo nivel los programas que permiten pasar de un programa fuente a un programa objeto se llaman programas ensambladores, mientras en los lenguajes de alto nivel estos programas se denominan compiladores e intérpretes.

INTÉRPRETES

Un intérprete as un traductor que toma un programa fuente, lo traduce a un programa objeto instrucción por instrucción, al mismo tiempo que ejecuta el programa, COMPILADORES

Los Compiladores son programas que traducen los programas fuentes a *programas* objetos.

El compilador traduce sentencia a sentencia cada una de las instrucciones del programa fuente a código máquina y posteriormente ejecuta el programa.

El código ensamblador es una versión mnemotécnica del código de máquina donde se usan nombres en lugar de códigos binarios para operaciones, y también se usan nombres para las direcciones de memoria, una instrucción en ensamblador puede ser

MOV a.R1

ADD #2, R1

MOV R1, b

Este código pasa el contenido de la dirección a al registro 1: después le suma la constante *2,* tratando al contenido del registro 1 como un número de punto fijo, y por ultimo almacena el resultado en la posición de memoria que representa b,

2.-Definición de traductores de bajo nivel

Lenguajes de bajo nivel (ensamblador)

La programación en lenguaje máquina es difícil, por ello se necesitan lenguajes que permitan simplificar este proceso. Los lenguajes de bajo nivel han sido diseñados para este fin.

Estos lenguajes son generalmente dependientes de la máquina, es decir, dependen de un conjunto de instrucciones específicas del ordenador. *Un* ejemplo de este tipo de lenguajes es el ensamblador. En él, las instrucciones se escriben en códigos alfabéticos conocidos como mnemotécnicos (generalmente, *abreviaturas* de palabras inglesas).

Las palabras mnemotécnicas son mucho más fáciles de recordar que las secuencias de ceros y unos. Una instrucción típica de ensamblador *puede* ser: ADD *x,y,z*

Esta instrucción significaría *que* se deben *sumar* los números almacenados en las direcciones de memoria x e y, y almacenar el resultado en la dirección z. Pero aún así, a medida que los programas crezcan en tamaño y complejidad, el ensamblador sigue sin ser una buena solución.

La forma más simple de *un* ensamblador *hace* dos pasadas sobre la *entrada* en donde una pasada consiste en leer una vez un archivo de entrada. En la primera pasada, se encuentra todos los identificadores que denoten posiciones de memoria y se almacenan en una tabla de símbolos.

En la segunda pasada el ensamblador examina el archivo de entrada de nuevo traduce cada código de operación a la secuencia de bits que representa esa operación en lenguaje de maquina y traduce cada identificador que representa una posición de memoria ala dirección dada ese identificador en la tabla de símbolos. Es resultado de la segunda pasada normalmente es código de maquina relocalizable lo cual significa *que* puede cargarse empezando en cualquier posición L de la memoria

3.-Aplicaciones y usos principales de los traductores de bajo nivel

Todos los ensambladores realizan básicamente las mismas tareas.
El empleo de ensambladores cruzados (Cross-Assembler), permite aprovechar el soporte de medios físicos (discos, impresoras, pantallas, etc.). y de programación que ofrecen las máquinas potentes para desarrollar programas que luego los van a ejecutar sistemas muy especializados en determinados tipos de tareas.
Los Ensambladores Residentes permiten ejecutar inmediatamente el programa; con la desventaja de que deben mantenerse en la memoria principal tanto el ensamblador como el programa fuente y el programa objeto.
Motivos para *utilizarlo,*

- Rapidez: Corno el programador directamente selecciona las instrucciones que se ejecutan en el programa, el programa final queda mas optimizado que *un programa* generado por *un* compilador.
- Mayor control de la computadora: Un programa *puede* accesar directamente cualquier componente y periférico de la computadora,
- independencia del lenguaje: No depende de librerías o del lenguaje mismo para realizar una tarea especifica. Lenguajes como el Basic limitan al programador a lo *que* el lenguaje *puede* hacer.
- La mayoría de las computadoras pueden ensamblar: Los recursos necesarios para ensamblar un programa son mucho menores que los compiladores o interpretes. El ensamblador generalmente es más rápido ensamblando un programa *que un* compilador generando un archivo ejecutable.

Motivos para no utilizarlo.

Desafortunadamente, también existen motivos para no crear los programas con ensamblador.

- Dependencia del hardware: El código se hace en extremo dependiente del microprocesador, de los dispositivos, de los controladores, etc. Este punto será analizado con mas detenimiento en dependencias de hardware.
- Mayor tiempo de codificación: El numero de líneas de un programa hecho en ensamblador es mayor a uno hecho en un lenguaje de alto nivel (por ejemplo; Función en C *puede* realizar varias decenas o centenas de

instrucciones del microprocesador).

- Comprensión mas *profunda* de la computadora; Entender *un* lenguaje de alto nivel es generalmente más sencillo que el ensamblador. Comprender ensamblador requiere conocimientos más exactos sobre el funcionamiento interno de la computadora.

■ Errores mas frecuentes en el programa: El evitar un error o encontrar alguno que ya exista es difícil. Las herramientas para este caso (como el CodeView y el TurboDebbuger) ayudan en gran medida a ver !o que esta ocurriendo en la maquina, pero no localizan los errores.

4.- Ejemplos de traductores de bajo nivel

Ensamblador

Es *un* programa de bajo nivel que traduce el lenguaje de ensamble a lenguaje de máquina. Utiliza letras del alfabeto para representar los diferentes arreglos del código binario de la máquina. Los programadores de ensamble deben conocer profundamente ¡a arquitectura y el lenguaje de máquina de su computadora. El programa ensamblador traduce cada instrucción de ensamble escrita por el programador a *a* instrucción en lenguaje de máquina binario equivalente. En general, las instrucciones ("software") de *un* sistema se escriban en este lenguaje. Ejemplos: Sistema operativo y Sistemas de manejo de base de datos.

Bibliografía

1. AHO, Alfred, *Compiladores.. Principios técnicos y herramientas,* México, Addison-Wesley.
2. BECK, Leland, *Introducción a la programación de sistemas,* México, Addison- Wesley.
3. LEMONE, Karen, *Fundamentos de Compiladores,* México, C.E.C.SA
4. www.paginasclick.com.mx/cien_tec/ebc01
5. www.ítlp.edu.mx/pubí!ca/tutoriales/progsfs1/tema12.htm
6. www.educared.net/concurso/82/TEXT013.html
7. www.quimika.com/materias/programacion/lenguajes.htm

UNIDAD II. TRADUCTORES DE ALTO NIVEL

1.- Introducción a los traductores de alto nivel

Los primeros programas ensambladores producían sólo una instrucción en lenguaje de máquina por cada instrucción del programa fuente. Para agilizar la codificación, se desarrollaron programas ensambladores que podían producir una cantidad variable de instrucciones en lenguaje de máquina por cada instrucción del programa fuente. Dicho de otra manera, una sola macroinstrucción podía producir varias líneas de código en lenguaje de máquina. El desarrollo de las técnicas mnemotécnicas y las macroinstrucciones condujo, a su vez, al desarrollo de lenguajes de alto nivel que a menudo están orientados hacia una clase determinada de problemas de proceso.

A diferencia de los programas de ensamble, los programas en lenguaje de alto nivel se *pueden* utilizar con diferentes marcas de computadoras sin tener que hacer modificaciones considerables,

2.-Definición de traductores de alto nivel

Los lenguajes de programación de alto nivel son aquellos en los que las instrucciones o sentencias a la computadora son escritas con palabras similares a los lenguajes humanos, lo que facilita la escritura y la fácil comprensión por el programación

Los lenguajes de programación son -en general- transportables, esto significa que un programa escrito en un lenguaje de alto nivel se *puede* escribir con poca o ninguna modificación en diferentes tipos de computadora, otra propiedad es que son Independientes de *la* maquina, esto es, las *s*entencias del programa no *dependen* del diseño o hardware de una computadora especifica.

Los programas escritos en lenguaje de alto nivel no son entendibles directamente por la maquina, necesitan ser traducidos a instrucciones en lenguaje maquina.

Los programas *que realizan* esta traducción se llaman compiladores, y los programas escritos *en una lenguaje* de alto nivel se llaman programas fuente.

el proceso de traducción de un programa fuente se denomina compilación y tras la fase de enlace se obtiene *un* programa ejecutable directamente por la computadora.

3.-Aplicaciones y usos principales de los traductores de alto nivel

Los programas traductores son de dos tipos: intérpretes y compiladores. Con un Intérprete, los programas que repiten *un* ciclo para volver a ejecutar parte de sus

instrucciones, reinterpretan la misma instrucción cada vez que aparece. Por consiguiente, los programas interpretados se ejecutan con mucha mayor lentitud que los programas en lenguaje máquina, Por el contrario, los compiladores trad*ucen* un programa integro a lenguaje máquina antas de su ejecución, por lo cual se ejecutan con tanta rapidez como si hubiesen sido escritos directamente en lenguaje máquina.

Aunque existen centenares de lenguajes informáticos y de variantes, cabe destacar el PASCAL, el LOGO *para niños,* el C, *un* lenguaje de Bell Laboratories, o el LISP o el PROLOG que sirven para desarrollar el campo de inteligencia

4.-Ejemplos de traductores de alto nivel

Los lenguajes de programación se dividen en 4 principales paradigmas: imperativo, funcional, orientado a objetos y lógico.

Lenguajes imperativos

Son llamados así porque están casados en comandos que actualizan variables que están en almacenamiento,

- ADA
- C
- CLIPPER &XBASE
- ENSAMBLADOR
- BASIC
- EUPHORIA
- FORTRAN
- PASCAL

Lenguajes funcionales

Los lenguajes funcionales se basan en el concepto de función, por tanto, el objeto básico y fundamental que manejamos son las funciones, que se pueden considerar las principales estructuras de control en este tipo de lenguajes. Ejemplos de lenguajes funcionales:

- ML
- CAML
- Haskell
- Sheme
- LISP
- Lamda - Cálculo
- Iswim
- APL
- FP
- Hope
- Miranda

- Eden
- Gofer
- Erlang

Programación Orientada a Objetos

- SmallTalk

Programación Orientada a la lógica

- PROLOG

LENGUAJES ESTRUCTURADOS

Los lenguajes estructurados incorporan una serie de instrucciones que facilitan la construcción modular de los programas así como corrección de errores y soporte de sistemas.

- ALGOL
- ADA
- C
- COBOL
- PASCAL

LENGUAJES DE INTELIGENCIA ARTIFICIAL

Podemos distinguir tres grandes estilos o subfamilias de íos lenguajes de inteligencia artificial. Los tres estilos de programación son íos siguientes: programación funcional, programación relacional y programación por objetos. EL lenguaje más representativo del estilo funcional es el LISP, LOGO por su identificación con LISP, cae de lleno dentro de este estilo. El lenguaje más representativo del estilo relacional PROLOG, El lenguaje más representativo del estilo de programación por objetos es el SMALLTALK, pero existen varios dialectos de LISP que permite programar de esta forma.

Bibliografía

1. AHO, Alfred, *Compiladores, Principios técnicos y herramientas,* México, Addison-Wesley.

2. BECK, Leíand, *Introducción a la programación de sistemas,* México, Addison-

Wesley.

3. LEMONE, Karen, *Fundamentos de Compiladores,* México, C.E.C.SA
4. www.paginasc!ick.com.mx/cien_tec/ebc01
5. www.itlp.edu.mx/publica/tutoriales/progsis1/tema12.htm
6. www.educared.net/concurso/82/TEXT013.html
7. www.quimika.com/matenas/programacion/ieriguajes.htm

UNIDAD III: SISTEMAS OPERATIVOS

Sistema Operativo es en sí mismo un programa de computadora. Sin embargo, es un programa muy especial, quizá el más complejo e importante en una computadora. El SO despierta a la computadora y hace que reconozca a la CPU, la memoria, el teclado, el sistema de vídeo y las unidades de disco. Además, proporciona la facilidad para que los usuarios se comuniquen con la computadora y sirve de plataforma a partir de la cual se corran programas de aplicación.

El sistema operativo está formado por el software que permite acceder y realizar las operaciones básicas en un ordenador personal o sistema informático en general. Los sistemas operativos más conocidos son: AIX (de IBM), GNU/Unux, HP-UX (de HP), MacOS (Macintosh), Solaris (de SUN Microsystems), las distintas variantes del UNIX de BSD (FreeBSD, OpenBSD...), y Windows en sus distintas variantes (de la empresa Microsoft).

Cuando se enciende una computadora, lo primero que ésta hace es llevar a cabo un autodiagnóstico llamado auto prueba de encendido (Power On Self Test, POST). Durante la POST, la computadora identifica su memoria, sus discos, su teclado, su sistema de vídeo y cualquier otro dispositivo conectado a ella. Lo siguiente que la computadora hace es buscar un SO para arrancar (boot). El sistema operativo tiene tres grandes funciones: coordina y manipula el hardware de la computadora, como la memoria, las impresoras, las unidades de disco, el teclado o el mouse; organiza los archivos en diversos dispositivos de almacenamiento, como discos flexibles, discos duros, discos compactos o cintas magnéticas, y gestiona los errores de hardware y la pérdida de datos.

1.-Estructura de los Sistemas Operativos

Se deben observar dos tipos de requisitos cuando se construye un sistema operativo, los cuales son:

- Requisitos de usuario: Sistema fácil de usar y de aprender, seguro, rápido y adecuado al uso al que se le quiere destinar.
- Requisitos del software: Donde se engloban aspectos como el mantenimiento, forma de operación, restricciones de uso, eficiencia, tolerancia frente a los errores y flexibilidad.

Posteriormente se describirán las distintas estructuras que presentan los actuales sistemas operativos para satisfacer las necesidades que de ellos se quieren obtener.

Estructura monolítica.

Es la estructura de los primeros sistemas operativos constituidos fundamentalmente por un solo programa compuesto de un conjunto de rutinas entrelazadas de tal forma que cada una puede llamar a cualquier otra (Ver Fig. 2).

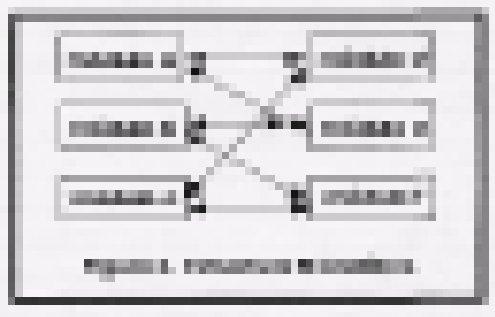
Las características fundamentales de este tipo de estructura son:

Construcción del programa final a base de módulos compilados separadamente que se unen a través del ligador.

Buena definición de parámetros de enlace entre las distintas rutinas existentes, que puede provocar mucho acoplamiento.

Carecen de protecciones y privilegios al entrar a rutinas que manejan diferentes aspectos de los recursos de la computadora, como memoria, disco, etc.

Generalmente están hechos a medida, por lo que son eficientes y rápidos en su ejecución y gestión, pero por lo mismo carecen de flexibilidad para soportar diferentes ambientes de trabajo o tipos de aplicaciones.



Estructura jerárquica

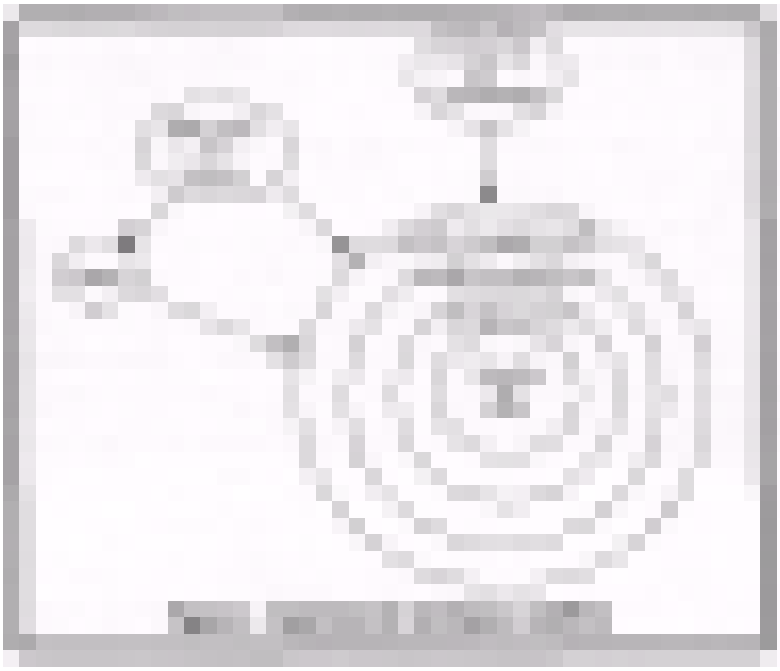
A medida que fueron creciendo las necesidades de los usuarios y se perfeccionaron los sistemas, se hizo necesaria una mayor organización del software, del sistema operativo, donde una parte del sistema contenía sub-partes y esto organizado en forma de niveles.

Se dividió el sistema operativo en pequeñas partes, de tal forma que cada una de ellas estuviera perfectamente definida y con un claro interface con el resto de elementos.

Se constituyó una estructura jerárquica o de niveles en los sistemas operativos, el primero de los cuales fue denominado THE (Technische Hogeschaol, Eindhoven), de Dijkstra, que se utilizó con fines didácticos (Ver Fig. 3). Se puede pensar también en estos sistemas como si fueran 'multicapa[1]. Multics y Unix caen en esa categoría. [Feld93].

-

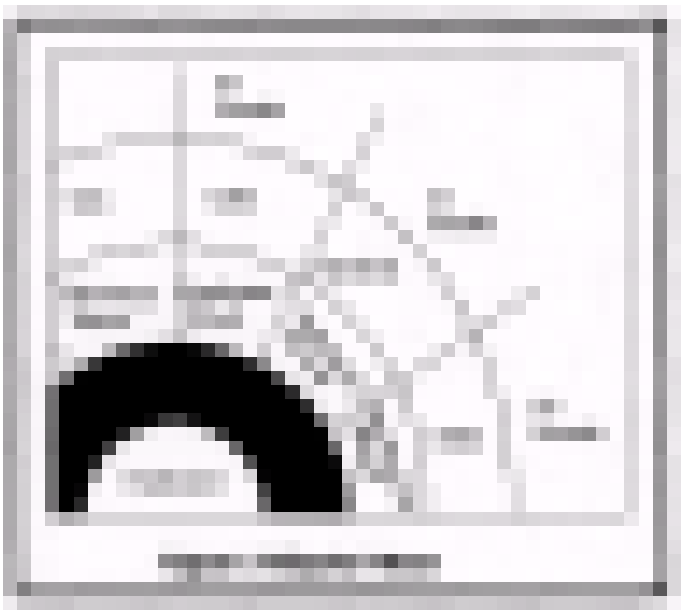
En la estructura anterior se basan prácticamente la mayoría de los sistemas operativos actuales. Otra forma de ver este tipo de sistema es la denominada de anillos concéntricos o "rings" (Ver Fig. 4).



En el sistema de anillos, cada uno tiene una apertura, conocida como puerta o trampa (trap), por donde pueden entrar las llamadas de las capas inferiores. De esta forma, las zonas más internas del sistema operativo o núcleo del sistema estarán más protegidas de accesos indeseados desde las capas más externas. Las capas más internas serán, por tanto, más privilegiadas que las externas.

Máquina Virtual,

Se trata de un tipo de sistemas operativos que presentan una interface a cada proceso, mostrando una máquina que parece idéntica a la máquina real subyacente. Estos sistemas operativos separan dos conceptos que suelen estar unidos en el resto de sistemas: la multiprogramación y la máquina extendida. El objetivo de los sistemas operativos de máquina virtual es el de integrar distintos sistemas operativos dando la sensación de ser varias máquinas diferentes.



El núcleo de estos sistemas operativos se denomina monitor virtual y tiene como misión llevar a cabo la multiprogramación, presentando a los niveles superiores tantas máquinas virtuales como se soliciten. Estas máquinas virtuales no son máquinas extendidas, sino una réplica de la máquina real, de manera que en cada una de ellas se pueda ejecutar un sistema operativo diferente, que será el que ofrezca ¡a máquina extendida al usuario (Ver Fig. 5).

Cliente-servidor (Microkernel)

El tipo más reciente de sistemas operativos es el denominado Cliente-servidor, que puede ser ejecutado en la mayoría de las computadoras, ya sean grandes o pequeñas.

Este sistema sirve para toda clase de aplicaciones por tanto, es de propósito general y cumple con las mismas actividades que los sistemas operativos convencionales.

El núcleo tiene como misión establecer la comunicación entre los clientes y los servidores. Los procesos pueden ser tanto servidores como clientes. Por ejemplo, un programa de aplicación normal es un cliente que llama al servidor correspondiente para acceder a un archivo o realizar una operación de entrada/salida sobre un dispositivo concreto. A su vez, un proceso cliente puede actuar como servidor para otro." Este paradigma ofrece gran flexibilidad en cuanto a los servicios posibles en el sistema final, ya que el núcleo provee solamente funciones muy básicas de memoria,

entrada/salida, archivos y procesos, dejando a los servidores proveer la mayoría que el usuario final o programador puede usar. Estos servidores deben tener mecanismos de seguridad y protección que, a su vez, serán filtrados por el núcleo que controla el hardware. Actualmente se está trabajando en una versión de UNIX que contempla en su diseño este paradigma.

2.-Control de Procesos

A continuación tenemos una metáfora que explica los diferentes estados de un proceso. Es la portada del libro "The Lógical Design of Operating Systems" de Lubomir Bic y Alan Shaw.

2.1.-Administración del procesador

La planificación del procesador se refiere a la manera o técnicas que se usan para decidir cuánto tiempo de ejecución y cuando se le asignan a cada proceso del sistema. Obviamente, si el sistema es monousuario y monotarea no hay mucho que decidir, pero en el resto de los sistemas esto es crucial para el buen funcionamiento del sistema.

2.2.-Algoritmos de planificación

Planificación a Plazo Fijo

Ciertos trabajos se planifican para ser terminados en un tiempo específico o plazo fijo. Es una planificación compleja debido a los siguientes factores:

- El usuario debe suministrar anticipadamente una lista precisa de recursos necesarios para el proceso, pero generalmente no se dispone de dicha información.

- La ejecución del trabajo de plazo fijo no debe producir una grave degradación del servido a otros usuarios.
- El sistema debe planificar cuidadosamente sus necesidades de recursos hasta el plazo fijo, lo que se puede complicar con las demandas de recursos de nuevos procesos que ingresen al sistema.
- La concurrencia de varios procesos de plazo fijo (activos a la vez) puede requerir métodos sofisticados de optimización.
- La administración intensiva de recursos puede generar una considerable sobrecarga adicional.

Planificación Garantizada

Se establecen compromisos de desempeño con el proceso del usuario, por ejemplo, si existen *"n"* procesos en el sistema, el proceso del usuario recibirá cerca del *"1 / n"* de la potencia de la cpu.

El sistema debe tener un registro del tiempo de cpu que cada proceso ha tenido desde su entrada al sistema y del tiempo transcurrido desde esa entrada.

Con los datos anteriores y el registro de procesos en curso de ejecución, el sistema calcula y determina qué procesos están más alejados por defecto de la relación *"1 /n"* prometida y prioriza los procesos que han recibido menos cpu de la prometida.

Planificación del Primero en Entrar Primero en Salir (FIFO)

Es muy simple, los procesos se despachan de acuerdo con su tiempo de llegada a la cola de listos.

Una vez que el proceso obtiene la cpu, se ejecuta hasta terminar, ya que es una disciplina "no apropiativa".

Puede ocasionar que procesos largos hagan esperar a procesos cortos y que procesos no importantes hagan esperar a procesos importantes.

Es más predecible que otros esquemas.

No puede garantizar buenos tiempos de respuesta interactivos.

Suele utilizarse integrado a otros esquemas, por ejemplo, de la siguiente manera:

- Los procesos se despachan con algún esquema de prioridad.
- Los procesos con igual prioridad se despachan **"FIFO".**

Planificación de Asignación en Rueda (RR: Round Robín)

Los procesos se despachan en **"FIFO"** y disponen de una cantidad limitada de tiempo de cpu, llamada "división de tiempo" o "cuanto".

Si un proceso no termina antes de expirar su tiempo de cpu ocurren las siguientes acciones:

1. La cpu es apropiada.
2. La cpu es otorgada al siguiente proceso en espera.

3. El proceso apropiado es situado al final de la lista de listos.

Es efectiva en ambientes de tiempo compartido.

La sobrecarga de la apropiación se mantiene baja mediante mecanismos eficientes de intercambio de contexto y con suficiente memoria principal para los procesos.

Tamaño del Cuanto o Quantum

La determinación del tamaño del cuanto es decisiva para la operación efectiva de un sistema computacional

Los interrogantes son: ¿cuanto pequeño o grande?, ¿cuanto fijo o variable? y ¿cuanto igual para todos los procesos de usuarios o determinado por separado para cada uno de ellos?.

Si el cuanto se hace muy grande, cada proceso recibe todo el tiempo necesario para llegar a su terminación, por lo cual la asignación en rueda ("RR") degenera en "FIFO".

Si el cuanto se hace muy pequeño, la sobrecarga del intercambio de contexto se convierte en un factor dominante y el rendimiento del sistema se degrada, puesto que la mayor parte del tiempo de cpu se invierte en el intercambio del procesador (cambio de contexto) y los procesos de usuario disponen de muy poco tiempo de cpu.

El cuanto debe ser lo suficientemente grande como para permitir que la gran mayoría de las peticiones interactivas requieran de menos tiempo que la duración del cuanto, es decir que el tiempo transcurrido desde el otorgamiento de la cpu a un proceso hasta que genera una petición de Entrada / Salida debe ser menor que el cuanto establecido, de esta forma, ocurrida la petición la cpu pasa a otro proceso y como el cuanto es mayor que el tiempo transcurrido hasta la petición de Entrada / Salida, los procesos trabajan al máximo de velocidad, se minimiza la sobrecarga de apropiación y se maximiza la utilización de la Entrada / Salida.

El cuanto óptimo varía de un sistema a otro y con la carga, siendo un valor de referencia 100 mseg (cien milisegundos).

Es una disciplina no apropiativa y por lo tanto no recomendable en ambientes de tiempo compartido.

El proceso en espera con el menor tiempo estimado de ejecución hasta su terminación es el siguiente en ejecutarse.

Los tiempos promedio de espera son menores que con "FIFO".

Los tiempos de espera son menos predecibles que en "FIFO".

Favorece a los procesos cortos en detrimento de los largos.

Tiende a reducir el número de procesos en espera y el número de procesos que esperan detrás de procesos largos.

Requiere un conocimiento preciso del tiempo de ejecución de un proceso, lo que generalmente se

desconoce.

Se pueden estimar los tiempos en base a series de valores anteriores.

Planificación del Tiempo Restante Más Corto (SRT)

Es la contraparte apropiativa del SJF,

Es útil en sistemas de tiempo compartido.

El proceso con el tiempo estimado de ejecución menor para analizar es el siguiente en ser ejecutado.

Un proceso en ejecución puede ser apropiado por un nuevo proceso con un tiempo estimado de ejecución menor.

Tiene mayor sobrecarga que la planificación SJF.

Debe mantener un registro del tiempo de servicio transcurrido del proceso en ejecución, lo que aumenta la sobrecarga.

Los trabajos largos tienen un promedio y una varianza de los tiempos de espera aún mayor que en SJF.

La apropiación de un proceso a punto de terminar por otro de menor duración recién llegado podría significar un mayor tiempo de cambio de contexto (administración del procesador) que el tiempo de finalización del primero.

Al diseñarse los Sistemas Operativos se debe considerar cuidadosamente la sobrecarga de los mecanismos de administración de recursos comparándola con los beneficios esperados.

Planificación el Siguiente con Relación de Respuesta Máxima (HRN)

Corrige algunas de las debilidades del SJF, tales como el exceso de perjuicio hacía los procesos (trabajos) largos y el exceso de favoritismo hacia los nuevos trabajos cortos.

Es una disciplina no apropiativa.

La prioridad de cada proceso está en función no sólo del tiempo de servicio del trabajo, sino que también influye la cantidad de tiempo que el trabajo ha estado esperando ser servido.

Cuando un proceso ha obtenido la cpu, corre hasta terminar.

Las prioridades, que son dinámicas, se calculan según la siguiente fórmula, donde p_r es la *"prioridad"*, t_e es el *"tiempo de espera" y* t_s es el *"tiempo de servicio":*

$$\Pr = \frac{(te + ts)}{ts}$$

Planificación por Prioridad

Considera factores externos al proceso. Las ideas centrales son que cada proceso tiene asociada una, prioridad y que el proceso ejecutable con máxima prioridad es el que tiene el permiso de ejecución. Los

procesos de alta prioridad podrían ejecutar indefinidamente, ya que el planificador del sistema puede disminuir la prioridad del proceso en ejecución en cada interrupción del reloj. Las prioridades también pueden ser asignadas dinámicamente por el sistema para lograr ciertas metas relacionadas con el procesador o la Entrada / Salida.

Los procesos limitados por la Entrada / Salida (requerimientos intensivos de Entrada / Salida) ocupan mucho de su tiempo en espera de operaciones de Entrada / Salida, por lo tanto:

- Deben tener prioridad para usar la cpu y efectuar la siguiente petición de Entrada / Salida, ya que se ejecutará (la operación de Entrada / Salida) en paralelo con otro proceso que utilice la cpu.
- Si deben esperar mucho tiempo a la cpu estarán ocupando memoria por un tiempo innecesario.

Un algoritmo sencillo consiste en establecer que la prioridad sea *"1 / f,* donde *T* es la fracción del último cuanto utilizado por el proceso.

Un proceso que utilice 2 mseg (dos milisegundos) de su cuanto de 100 mseg (cien milisegundos) tendrá prioridad 50 (cincuenta).

Un proceso que se ejecutó 50 mseg antes del bloqueo tendrá prioridad 2.

Un proceso que utilizó todo el cuanto tendrá prioridad 1.

Frecuentemente los procesos se agrupan en *"Clases de Prioridad",* en cuyo caso se utiliza la Planificación con Prioridades entre las clases y con Round Robin (RR) dentro de cada clase. Si las prioridades no se reajustan en algún momento, los procesos de las clases de prioridad mínima podrían demorarse indefinidamente.

Colas de Retroalimentación de Niveles Múltiples

Proporcionan una estructura para lograr !os siguientes objetivos:

- Favorecer trabajos cortos.
- Favorecer trabajos limitados por la Entrada / Salida para optimizar el uso de los dispositivos de Entrada / Salida.
- Determinar la naturaleza de un trabajo lo más rápido posible y planificar el trabajo (proceso) en consecuencia.

Un nuevo proceso entra en la red de línea de espera al final de la cola superior.

Se mueve por esta cola "FIFO" hasta obtener la cpu.

Si el trabajo termina o abandona la cpu para esperar por la terminación de una operación de Entrada / Salida o la terminación de algún otro suceso, el trabajo abandona la red de línea de espera.

Si su cuanto expira antes de abandonar la cpu voluntariamente, el proceso se coloca en la parte trasera de la cola del siguiente nivel inferior.

El trabajo recibe servicio al llegar a la cabeza de esta cola si la primera está vacía.

Mientras el proceso continúe consumiendo totalmente su cuanto en cada nivel, continuará moviéndose hacia el final de las colas inferiores.

Generalmente hay una cola en la parte más profunda a través de la cual el proceso circula en asignación de rueda hasta que termina.

Existen esquemas en los que el cuanto otorgado al proceso aumenta a medida que el proceso se mueve hacia las colas de los niveles inferiores, en tal caso, cuanto más tiempo haya estado el proceso en la red de línea de espera, mayor será su cuanto cada vez que obtiene la cpu y no podrá obtener la cpu muy a menudo debido a la mayor prioridad de los procesos de las colas superiores.

Un proceso situado en una cola dada no podrá ser ejecutado hasta que las colas de los niveles superiores estén vacías.

Un proceso en ejecución es apropiado por un proceso que llegue a una cola superior.

Es un mecanismo adaptable, es decir que se adapta a cargas variables.

Política Versus Mecanismo de Planificación

Puede ocurrir que haya procesos con muchos procesos hijos ejecutándose bajo su control, por ejemplo, un proceso en un DBMS con procesos hijos atendiendo funciones específicas, tales como, análisis de interrogantes, acceso a discos, etc.

es posible que el proceso principal (padre) pueda identificar la importancia (o criticidad) de sus procesos hijos, pero los planificadores analizados no aceptan datos de los procesos de usuario relativos a decisiones de planificación.

La solución es separar el **mecanismo de planificación** de la **política de planificación,** para ello se parametriza el algoritmo de planificación y los parámetros pueden ser determinados por medio de procesos del usuario; así el mecanismo está en el núcleo del Sistema Operativo pero la política queda establecida por un proceso del usuario

Planificación de Dos Niveles

Los esquemas analizados hasta ahora suponen que todos los procesos ejecutables están en la memoria principal.

Si la memoria principal es insuficiente, ocurrirá lo siguiente

- Habrá procesos ejecutables que se mantengan en disco.
- Habrá importantes implicaciones para la planificación, tales como las siguientes:
    - o El tiempo de alternancia entre procesos para traer y procesar un proceso del disco es considerablemente mayor que el tiempo para un proceso que ya está en la memoria principal.
    - o Es más eficiente el intercambio de los procesos con un planificador de dos niveles.

El esquema operativo de un planificador de dos niveles es corno sipu-

1. Se carga en la memoria principal cierto subconjunto de los procesos ejecutables.

2. El planificador se restringe a ellos durante cierto tiempo.
3. Periódicamente se llama a un planificador de nivel superior para efectuar las siguientes tareas:
    1. Eliminar de la memoria los procesos que hayan permanecido en ella el tiempo suficiente.
    2. Cargar a memoria los procesos que hayan estado en disco demasiado tiempo.
4. El planificador de nivel inferior se restringe de nuevo a los procesos ejecutables que se encuentren en la memoria.
5. El planificador de nivel superior se encarga de desplazar !os procesos de memoria a disco y viceversa.

Los criterios que podría utilizar el planificador de nivel superior para tomar sus decisiones son los que se indican a continuación;

- ¿Cuánto tiempo ha transcurrido desde el último intercambio del proceso?.
- ¿Cuánto tiempo de cpu ha utilizado recientemente el proceso?.
- ¿Qué tan grande es el proceso? (generalmente los procesos pequeños no causan tantos problemas en este sentido).
- ¿Qué tan alta es la prioridad del proceso?.

El planificador de nivel superior podría utilizar cualquiera de los métodos de planificación analizados.

2.3.-Sistemas Operativos Multiusuarios

Los sistemas operativos multiusuarios son capaces de dar servicio a más de un usuario a la vez, ya sea por medio de varias terminales conectadas a la computadora o por medio de sesiones remotas en una red de comunicaciones. No importa el número de procesadores en la máquina ni el número de procesos que cada usuario puede ejecutar simultáneamente.

3.-Control de Recursos

Las computadoras modernas constan de procesadores, memorias, cronómetros, discos, terminales, unidades de cinta magnética, interfaces de red impresoras láser y una amplía gama de oíros dispositivos. El sistema operativo puede poner orden en el caos potencial almacenar en el disco toda salida destinada a la impresora, sostiene que su principal tarea es la de llevar un registro de la utilización de los recursos. Un aspecto económico también requiere que a menudo compartir la información entre aquellos usuarios que trabajan juntos. El control de recursos se refiere al manejo de los mismos. Un recurso es cualquier cosa que sólo puede ser utilizada por un único proceso en un instante dado.

3.1.- El problema del ínter bloqueo y la postergación indefinida

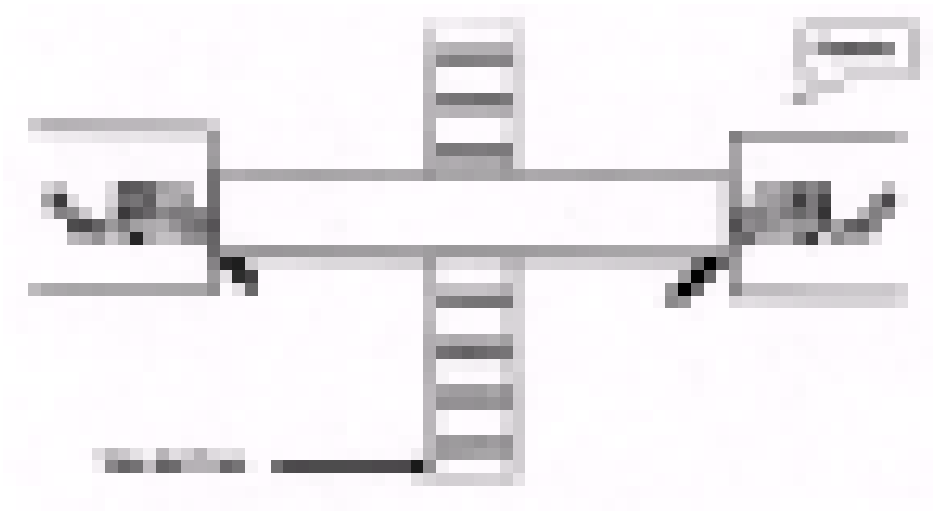
Los sistemas de cómputo tienen muchos recursos que sólo pueden ser utilizados por un proceso a la

vez. Si dos procesos utilizaran en forma simultánea la impresora, el resultado sería un desastre. En muchas aplicaciones, un proceso necesita el acceso exclusivo no sólo a un recurso, sino a varios. Un proceso que copie un archivo mayor que un disco desde una cinta magnética hacia la impresora necesita el acceso exclusivo a la unidad de cinta y a la impresora al mismo tiempo. En un sistema con un único proceso, éste puede tener el acceso a todos los recursos que necesite y realizar su trabajo. Sin embargo, en un sistema con multiprogramación, pueden surgir serios problemas. Supongamos, por ejemplo, que dos procesos desean imprimir cada uno un enorme archivo en cinta. El proceso A solicita el permiso para utilizar la impresora, el cual se le concede. Es entonces cuando el proceso B solicita permiso para utilizar la unidad cinta y se le otorga. El proceso A solícita entonces la unidad de cinta, pero la solicitud es denegada hasta que B la libere. Por desgracia, en este momento, en vez de liberar unidad de cinta, B solícita la impresora.

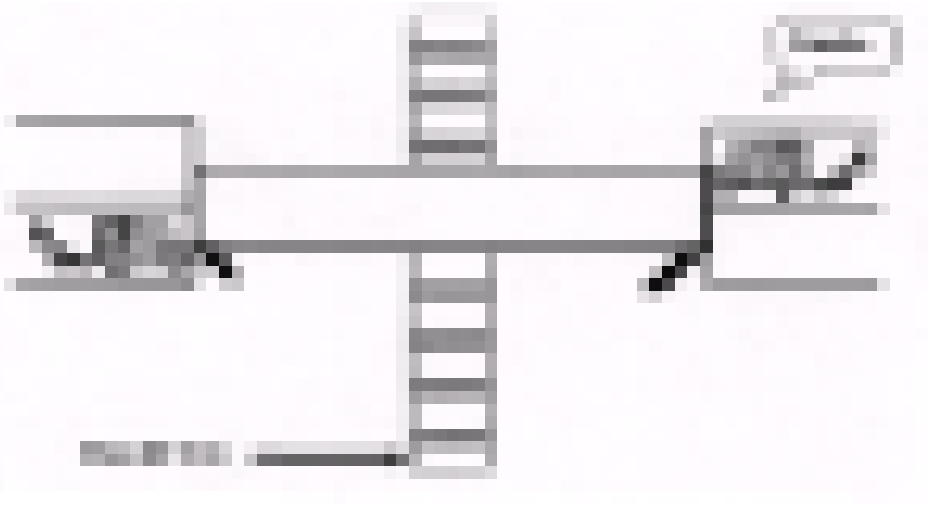
3.2.-Tratamiento de ínterbloqueo

En los sistemas operativos, se puede definir el problema del interbloqueo como la situación de un .conjunto de procesos que están en espera y que ninguno de ellos tiene las condiciones necesarias para continuar su ejecución. Veamos el siguiente ejemplo.

En una carretera de dos direcciones, donde en un determinado cruce con la vía del ferrocarril, se ha construido un puente que sólo deja pasar vehículos en un sentido. El bloqueo ocurre cuando dos carros intentan pasar por el puente al mismo tiempo.



Una manera de resolver el bloqueo es: el conductor situado, en uno de los extremos es \o suficientemente educado que deja pasar en primer lugar al del otro extremo y luego pasa é!.



El ejemplo anterior nos muestra como sucede el interbloqueo en nuestra vida diaria. Actualmente, la recuperación se suele realizar eliminando un proceso y quitándole sus recursos. El proceso eliminado se pierde, pero gracias a esto ahora es posible terminar. Algunas veces es necesario, eliminar varios procesos hasta que se hayan liberado los recursos necesarios para que terminen los procesos restantes.

Sus procesos pueden eliminarse de acuerdo con algún orden de prioridad, aunque es posible que no ' existan prioridades entre los procesos bloqueados, de modo que el operador necesita tomar una decisión arbitraria para decidir que procesos se eliminarán.

La detección y recuperación es la estrategia que a menudo se utiliza en grandes computadoras, especialmente sistemas por lote en los que la eliminación de un proceso y después su reiniciación suele aceptarse.

3.3.-Tratamiento de la postergación indefinida

Postergación indefinida (inanición o Lockout): cuando un proceso espera por un evento que puede ocurrir pero no se sabe cuando.

El problema de las postergaciones indefinidas no se circunscribe únicamente al mundo de la informática, sino que aparece en muchos otros ámbitos incluyendo el de la vida cotidiana. De hecho, algunos de los ejemplos utilizados por los investigadores en este tema están inspirados en situaciones

cotidianas.

La postergación indefinida surge debido a que se produce un conflicto entra las necesidades de los dos vehículos: el recurso que necesita cada vehículo lo posee el otro. Hay que resaltar que otros vehículos que intentarán cruzar el puente en ese momento en cualquiera de los dos sentidos se

quedarían detenidos detrás de ellos viéndose, por tanto, implicados también en la postergación indefinida. Las posibles estrategias para tratar el problema de las postergaciones indefinidas son:

■ **Detección y recuperación.**

Una vez detectada la situación de postergación indefinida uno de los vehículos debe liberar et recurso que posee para dejar que el otro lo utilice. Una posible recuperación de esta situación consistiría en seleccionar uno de los sentidos de circulación y hacer que el vehículo o vehículos detenidos en ese sentido dieran marcha atrás hasta el principio del puente, liberando así el paso en el otro sentido (se está suponiendo que un vehículo tiene capacidad para avanzar marcha atrás, 3ino fuera así, habría que tomar una acción más drástica como tirarlo al río que pasa por debajo del puente). Debería existir una política para determinar qué vehículo debe retroceder.

■ **Prevención o predicción.**

Un punto importante a resaltar en este ejemplo y, en general, sobre las postergaciones indefinidas es que, antes de producirse la postergación indefinida propiamente dicha (los vehículos detenidos frente a frente), existe un «punto de retorno» a partir del cual la postergación indefinida es inevitable.

4.-La gestión de Entrada/Salida

Distintas personas analizan de varias maneras el hardware de Entrada y Salida. Los ingenieros eléctricos lo hacen en términos de chips, cables, fuentes de poder, etc. Los programadores se fijan en la interfaz que se presenta al software (los comandos que acepta el hardware, las funciones que realiza y los errores que puede informar. Sin embargo es frecuente que la programación de muchos dispositivos de entrada y salida este íntimamente ligada con su operación interna. El código destinado a manejar la entrada y salida de los diferentes periféricos en un sistema operativo es de una extensión considerable y sumamente complejo. Resuelve la necesidades de sincronizar, atrapar interrupciones y ofrecer llamadas al sistema para los programadores.

4.1.-Principios de entrada/salida (hardware)

Los dispositivos de entrada salida se dividen, en general, en dos tipos: dispositivos orientados a bloques y dispositivos orientados a caracteres. Los dispositivos orientados a bloques tienen la propiedad de que se pueden direccionar, esto es, el programador puede escribir o leer cualquier bloque del dispositivo realizando primero una operación de posicionamiento sobre el dispositivo. Los dispositivos más comunes orientados a bloques son los discos duros, la memoria, discos compactos y, posiblemente, unidades de cinta. Por otro lado, los dispositivos orientados a caracteres son aquellos que trabajan con secuencias de byes sin importar su longitud ni ninguna agrupación en especial. No son dispositivos direccionables. Ejemplos de estos dispositivos son el teclado, la pantalla

o display y las impresoras.

La clasificación anterior no es perfecta, porque existen varios dispositivos que generan entrada o salida que no pueden englobarse en esas categorías. Por ejemplo, un reloj que genera pulsos. Sin embargo, aunque existan algunos periféricos que no se puedan categorizar, todos están administrados por el sistema operativo por medio de una parte electrónica - mecánica y una parte de software.

4.2.-Principios de entrada/salida (software)

Los principios de software en la entrada - salida se resumen en cuatro puntos: el software debe ofrecer manejadores de interrupciones, manejadores de dispositivos, software que sea independiente de los dispositivos y software para usuarios.

Manejadoras de interrupciones El primer objetivo referente a los manejadores de interrupciones consiste en que el programador o el usuario no debe darse cuenta de los manejos de bajo nivel para los casos en que el dispositivo está ocupado y se debe suspender el proceso o sincronizar algunas tareas. Desde el punto de vista del proceso o usuario, el sistema simplemente se tardó más o menos en responder a su petición.

Manejadores de dispositivos El sistema debe proveer los manejadores de dispositivos necesarios para los periféricos, así como ocultar las peculiaridades del manejo interno de cada uno de ellos, tales como el formato de la información, los medios mecánicos, los niveles de voltaje y otros. Por ejemplo, sí el sistema tiene varios tipos diferentes de discos duros, para el usuario o programador las diferencias técnicas entre ellos no le deben importar, y los manejadores le deben ofrecer el mismo conjunto de rutinas para leer y escribir datos.

Software independiente del dispositivo Este es un nivel superior de independencia que el ofrecido por los manejadores de dispositivos. Aquí el sistema operativo debe ser capaz, en lo más posible, de ofrecer un conjunto de utilerías para accesar periféricos o programarlos de una manera consistente. Por ejemplo, que para todos los dispositivos orientados a bloques se tenga una llamada para decidir si se desea usar 'buffers' o no, o para posicionarse en ellos.

Software para usuarios La mayoría de las rutinas de entrada - salida trabajan en modo privilegiado, o son llamadas al sistema que se ligan a los programas del usuario formando parte de sus aplicaciones y que no le dejan ninguna flexibilidad al usuario en cuanto a la apariencia de los datos. Existen otras librerías en donde el usuario si tiene poder de decisión (por ejemplo la llamada a "printf' en el lenguaje "C"). Otra facilidad ofrecida son las áreas de trabajos encolados (spooling áreas), tales como las de impresión y correo electrónico.

5.-Gestión de la memoria

La memoria se organiza en dos grupos:

Organización Contigua:

1. Sistemas (Monousuarios (dedicados a un solo usuario).
2. Sistemas de multiprogramación en memoria real:

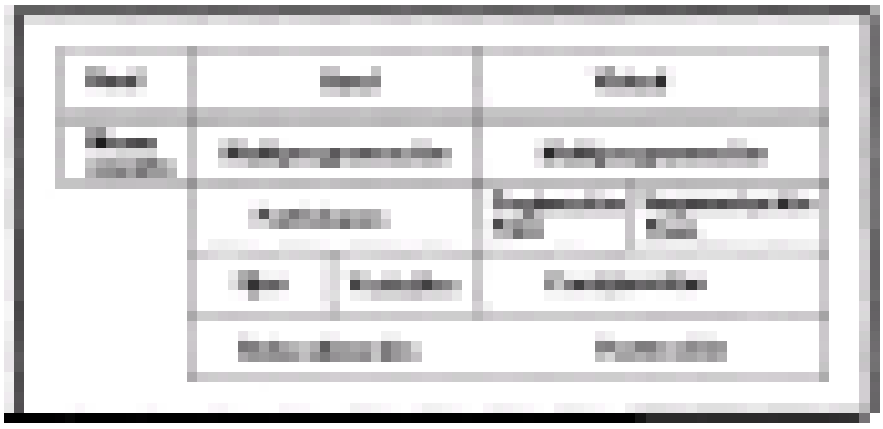
   - Multiprogramación en partición fija:
     - Absoluta.
     - Relocalizabíe (reubicable).
   - Multiprogramación en partición variable.

Organización <u>NO</u> Contigua:

3. Multiprogramación en almacenamiento virtual:
   - Paginación pura.
   - Segmentación pura
   - Combinación paginación / segmentación

En la "*asignación contigua*" cada programa ocupa un bloque contiguo y sencillo de localizaciones de almacenamiento.

En la *"asignación no contigua'* un programa se divide en varios bloques o *"segmentos"* que pueden almacenarse en direcciones que no tienen que ser necesariamente adyacentes, por lo que es más compleja pero más eficiente que la asignación continua.



6.-Gestión de la Administración

La gran categoría de llamadas al sistema se relaciona con el sistema de archivos una de funciones principales es ocultar las peculiaridades de los discos y demás dispositivos de E/S para presentar al programador un modelo agradable y nítido de archivos independientes. La necesidad de las llamadas al sistema operativo en la creación eliminación lectura y escritura de archivos antes de poder leer un archivo hay que abrirlo.

6.1.-Estructura de la Información

Para poder proporcionar un espacio donde almacenar los archivos la mayoría de los sistemas operativos soportan el concepto de directorio como una forma de agrupar los archivos. El proceso y jerarquías de archivos se organizan como árboles pero solo similares, las jerarquías de los procesos no son en general muy profundas mientras que las jerarquías de los archivos tiene por lo general 4 o 5 niveles. Las jerarquías de los procesos tiene una vida corta de unos cuantos minutos mientras que la jerarquía de los directorios puede durar años.

Cada uno de los archivos que se encuentra de la jerarquía del directorio puede determinarse mediante el nombre de la ruta de acceso desde la parte superior de dicha jerarquía el directorio raíz tales nombres absolutos de la ruta de acceso

6.2.-Acceso a disco

Discos duros. Para correr cualquier tipo y clase de SO es necesario contar con un disco duro, entre mas rápido y grande sea, mejor. Se puede llegar a una ecuación en donde, después de tomar en cuenta el espacio requerido por los programas de aplicación y el propio SO, se calcule la capacidad aproximada necesario para este disco o discos.

Si por ejemplo, se va a utilizar el SO para procesamiento de palabras, se puede definir que cada usuario ocupara un máximo de tres MB en documentos; si se cuenta con 6 usuarios, se necesitara unos 18 Mb únicamente para almacenar los datos de los usuarios. Esta aproximación siempre se hace en una forma sobrada, con objeto de no limitar al sistema por falta de espacio en el disco duro. Actualmente se tienen disco duros de 1,200 Mb, disponibles y a la venta en México, funcionando en Computadoras Personales, con lo que al instalar dos de estas unidades se obtiene 2,400 Mb (¡2.4 Gígabytes !) de almacenamiento en un espacio no mayor de unos 30 centímetros cúbicos. Sólo como comparición si se posee una máquina con 20 Mb imagínese 120 discos duros iguales, uno sobre otro, en donde se puede almacenar cuanta información pueda.

6.3.-Gestión del almacenamiento

El subsistema de archivos se debe encargar de localizar espacio libre en los medios de almacenamiento para guardar archivos y para después borrarlos, renombrarlos o agrandarlos. Para ello se vale de localidades especiales que contienen la lista de archivos creados y por cada archivo

una serie de direcciones que contienen los datos de los mismos. Esas localidades especiales se llaman directorios. Para asignarle espacio a los archivos existen tres criterios generales que se describen enseguida.

- Asignación contigua: Cada directorio contiene la los nombres de archivos y la dirección del bloque inicial de cada archivo, así como el tamaño total de los mismos. Por ejemplo, si un archivo comienza en el sector 17 y mide 10 bloques, cuando el archivo sea accesado, el brazo se moverá inicialmente al bloque 17 y de ahí hasta el 27. Si el archivo es borrado y luego creado otro más pequeño, quedarán huecos inútiles entre archivos útiles, lo cual se llama fragmentación externa.
- Asignación encadenada: Con este criterio los directorios contienen los nombres de archivos y por cada uno de ellos la dirección del bloque inicial que compone al archivo. Cuando un archivo es leído, el brazo va a esa dirección inicial y encuentra los datos iniciales junto con la dirección del siguiente bloque y así sucesivamente. Con este criterio no es necesario que los bloques estén contiguos y no existe la fragmentación externa, pero en cada "eslabón" de la cadena se desperdicia espacio con las direcciones mismas. En otras palabras, lo que se crea en el disco es una lista ligada.
- Asignación con índices (indexada): En este esquema se guarda en el directorio un bloque de índices para cada archivo, con apuntadores hacia iodos sus bloques constituyentes, de manera que el acceso directo se agiliza notablemente, a cambio de sacrificar varios bloques para almacenar dichos apuntadores. Cuando se quiere leer un archivo o cualquiera de sus partes, se hacen dos accesos: uno al bloque de índices y otro a la dirección deseada. Este es un esquema excelente para archivos grandes pero no para pequeños, porque la relación entre bloques destinados *para* índices respecto a los asignados para datos es incosteable.

Bibliografía

1. AHO, Alfred, *Compiladores, Principios técnicos y herramientas,* México, Addison-Wesley.
2. BECK, Leland, *Introducción a la programación de sistemas,* México, Addison-Wesley.
3. LEMONE, Karen, *Fundamentos de Compiladores,* México, C.E.C.S.A.
4. www.paginasclick.com.mx/cíen_íec/ebcO1
5. www.itlp.edu.mx/publica/tutoriales/progsis1/tema12.htm

6. www,educared,net/concurso/82/TEXTO13¡html

7. www.quimika.com/matenas/programacion/lenguajes.htm

UNIDAD IV: HERRAMIENTAS DE CONFIGURACIÓN, ARRANQUE Y OPERACIÓN DE LOS SISTEMAS DE CÓMPUTO.

Cuando hablamos de un sistema de cómputo nos referimos al conjunto de elementos que interactúan permitiendo a un usuario hacer uso de dicho computador con el fin o propósito de servirle de herramienta para cumplir con determinada tarea. Como todo sistema, cuando algún elemento falla, este repercute en los demás ocasionando así que la falla desencadene en la falla general del sistema. En un sistema de cómputo podríamos agrupar los elementos en 3 frentes o bloques: **Hardware** (Parte física y palpable, la cual es el conjunto de componentes electrónicos, chips, unidades, dispositivos etc.), **Software** (Parte lógica e intangible, conjunto de programas incluyendo el sistema operativo) y el **Firmware** (son los programas o instrucciones que se encuentran grabados o "embebidos" en dispositivos de Hardware, se asocia directamente con memorias ROM de solo lectura, el principal contenedor de Firmware en el computador es la BIOS).

El hardware se refiere a los componentes materiales de un sistema informático. La función de estos componentes suele dividirse en tres categorías principales: . entrada, salida y almacenamiento. Los componentes de esas categorías están conectados a través de un conjunto de cables o circuitos llamado bus con la unidad central de proceso (CPU) del computador, el microprocesador que controla la computadora y le proporciona capacidad de cálculo.

En cuanto al Firmware podemos decir que no solamente en nuestro computador existe la BIOS de nuestra tarjeta principal, existen ciertos dispositivos que tienen su propia memoria ROM que buscan ser reconocidos y entablar comunicación con la BIOS general, de esta forma los dispositivos plug and play pueden ser utilizados directamente por el sistema operativo. al Software en cambio se le puede clasificar de la siguiente forma:

- Software de sistema: Es el sistema operativo. Es el puente que existe entre el sistema de computo (Hardware, Software y Firmware) y el usuario, el cual permite entablar una comunicación entre ellos. Podría decirse que es el que traduce las instrucciones de las tareas que el usuario desea ejecutar y las manifestaciones que la máquina emite.
- Software aplicativo: Conjunto de aplicaciones que le permiten al usuario llevar a cabo las tareas que desea ejecutar. Existe una inmensa variedad de estas

aplicaciones algunas de ellas son desarrolladas por los programadores y desarrolladores de software para clientes específicos. Ejemplos de estas aplicaciones son los programas que vienen con el paquete de oficina de Microsoft Office (Word, Power Point, Excel Access etc.), programas para escuchar música, ver películas etc. Se podría hacer que hay software para lo que quiera, y lo que no esté se puede desarrollar, tarea de los ingenieros de sistemas.

- Software de control; También conocido como drivers. Son los controladores que permiten que el sistema operativo haga uso de los dispositivos. Cuando

instalamos una impresora debemos instalarle al sistema operativo de nuestra computadora el controlador para que este haga un uso apropiado de ella.

El sistema operativo tiene tres grandes funciones: coordina y manipula el hardware del computador, como la memoria, las impresoras, las unidades de disco, el teclado o el mouse; organiza los archivos en diversos dispositivos de almacenamiento, como discos flexibles, discos duros, discos compactos o cintas magnéticas, y gestiona los errores de hardware y la pérdida de datos.

1.-Herramientas de configuración de los sistemas de cómputo Las herramientas de configuración de los sistemas de cómputo no se refieren a hardware. Varían de acuerdo al sistema operativo que tenga instalado el sistema de cómputo (Windows, Linux, Unix, MacOS BeOS, Solaris, etcétera; en sus diferentes versiones y/o distribuciones). Indican la mejor forma de poder trabajar con el sistema de cómputo. Ejemplo:

Configuración de la seguridad del sistema: para configurar la seguridad de un sistema basado en Windows NT o Windows 2000, puede utilizar una de las opciones del conjunto de herramientas:

- **Extensión de configuración de seguridad del Editor de políticas de grupo:** se recomienda esta opción para la configuración si utiliza una infraestructura de Windows basada en Active Directory. También se puede utilizar localmente en equipos individuales con o sin Active Directory. En el caso local, se configura un objeto de política de grupo localmente en un equipo. Para utilizar esta opción, inicie el Editor de políticas de grupo y seleccione un objeto de política de grupo apropiado, que puede ser el que está almacenado en Active Directory o localmente en un equipo. Haga clic en **Configuración del equipo** y, luego, en **Configuración de seguridad.** El espacio de nombres de nodo que se muestra aquí es idéntico al que aparece en el Editor de configuración de segundad, en el que se modifica una configuración determinada. Puede copiar y pegar nodos específicos (cada uno en representación de un *área* de seguridad determinada) desde el Editor de configuración de seguridad al nodo correspondiente de la Política de grupo o puede importar una configuración completa a la Política de grupo. Esto hace que la configuración de seguridad se guarde en un objeto de política de grupo y se aplique para exigir las políticas de grupo. Los objetos de política de grupo se aplican a un

equipo, basados en el ámbito de Active Directory (dominio y unidades organizativas) en el que se encuentra el equipo. Esto puede hacer que se apliquen al equipo varias configuraciones de seguridad. Si contienen los mismos atributos, prevalece la que se escribe en último lugar, según el orden de aplicación de los objetos de política de grupo.

- **Administrador de configuración de seguridad:** se recomienda esta opción para la configuración sólo cuando no utilice una infraestructura basada en Active Directory y no necesite hacer que se aplique la configuración de seguridad periódicamente; es decir, prefiere controlar la configuración y el análisis manualmente. Para utilizarlo, inicie MMC y agregue el complemento Administrador de configuración de seguridad y sus complementos de extensión. De forma predeterminada, el complemento apunta a la base de datos de políticas de equipo local. Puede optar por cambiar a una base de datos distinta si hace clic con el botón secundario del *mouse* (ratón) en el nodo **Administrador de configuración de seguridad** y, después, hace clic en **Establecer base de** datos en el menú contextual. En el Administrador de configuración de seguridad, seleccione **Importar configuración** en el menú contextual. Así se abrirá el cuadro de diálogo **Abrir archivo,** que se utiliza para buscar una configuración almacenada y seleccionaría. Repita este proceso importando las configuraciones adicionales almacenadas como *opciones increméntales.* La base de datos combina diversas configuraciones para crear una configuración compuesta; los conflictos se resuelven mediante la regla según la cual prevalece lo último que se escribe. Una vez importadas las configuraciones a la base de datos seleccionada, haga clic en **Configurar ahora** en el menú contextual para aplicarlas al sistema. Un cuadro de diálogo de progreso mostrará cómo se está aplicando la configuración y, finalmente, mostrará el registro de errores si se producen errores en el proceso.
- **Herramienta de la línea de comandos Secedit:** se recomienda esta opción cuando no emplee una infraestructura basada en Active Directory y utilice varios equipos que se deben configurar con frecuencia, inicie una ventana de la consola y especifique Secedit.exe; después, seleccione las opciones apropiadas, por ejemplo, dónde se debe guardar la base de datos de seguridad, qué configuración o configuraciones se deben utilizar, etc. También puede crear archivos de comandos por lotes y, a continuación, programarlos para que se ejecuten en horas de

inactividad, mediante el programador de tareas. Puede utilizar Microsoft System Management Server para distribuir esta tarea entre varios equipos diferentes.

Nota El conjunto de herramientas admite la capacidad de aplicar múltiples configuraciones. Puede decidir aplicar unos pocos valores de configuración inicialmente y luego agregarle valores. La base de datos de seguridad almacena la combinación de configuraciones múltiples y las opciones de configuración más recientes sobrescriben cualquier valor anterior de la misma opción.

2.-Herramientas de arranque de los sistemas de cómputo **Arranque de la computadora**

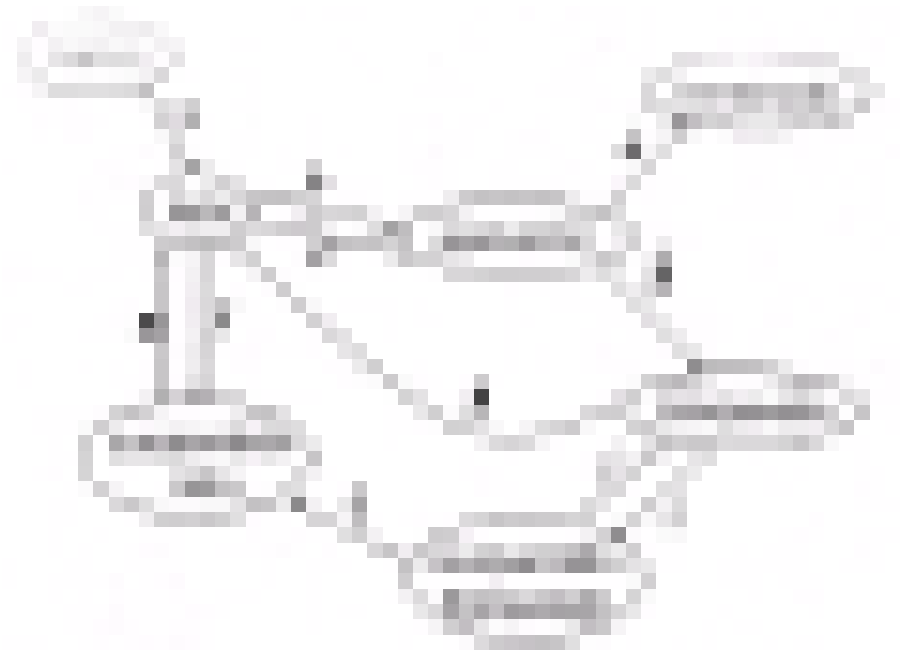
El arranque de una computadora actual tiene dos fases:

- Arranque hardware
- Arranque software

Que por el **arranque hardware** se entiende que es la parte dura es decir el inicio o encendido de todos los componentes de la PC

Ahora el **arranque software** es el inicio del sistema operativo en una computadora

Para el siguiente diagrama, se explican las transiciones más comunes, una vez iniciado el sistema de cómputo.



a) Cuando el sistema operativo acepta un nuevo proceso. Puede ser por nuevos trabajos baten, por conexiones interactivas (vía telnet), generado por el mismo sistema operativo para proporcionar un servicio o generado por otro proceso existente (padre/hijo). Al entrar a listo ya debe tener su PCB creado.

b) Se hace un *dispatch* para que el procesador ejecute a uno de los procesos listos. En sistemas de multiprocesamiento, pueden estarse ejecutando mas de un procesos a la vez (a lo más uno por CPU), pero esto no siempre ocurre así. Por ejemplo, uno de los CPUs puede no estar ejecutando ningún proceso si está ejecutando instrucciones en modo supervisor o protegido (efectuando intercambios de contexto, estar en espera de un *pin* o un *parent,* etc.)

c) Se puede regresar a un proceso a listo por expiración de quantum, por ingreso de un proceso de mayor prioridad, o por la ejecución de algunas interrupciones (restart, fallo de hardware...)

d) Se termina un proceso mediante la instrucción de terminación normal (END. en Pascal), tiempo límite excedido - cuando los procesos tienen un límite de CPU en cada ejecución (es una manera de detectar ciclos infinitos), por falta de memoria disponible, por accesar localidades de memoria restringidas (por ejemplo, un apuntador no aterrizado), por errores de cálculo (por ejemplo, SQRT(-1)), tiempo máximo de espera rebasado (es una manera de detectar deadlocks), fallos de I/O (por ejemplo, FILE NOT FOUND), instrucciones inválidas (por ejemplo, en códigos de archivos corruptos),

instrucciones privilegiadas (por ejemplo, tratar de cambiar la prioridad de un archivo público), error de datos (por ejemplo, char b=3.2), intervención del operador (por ejemplo, detección de un deadlock), terminación del padre (por ejemplo, al cerrar un proceso que estaba ejecutando una macro), solicitud del padre (por ejemplo, el padre termina a un hijo que contabilizaba el tiempo de ejecución de un subproceso).

e) Se activa un proceso que estaba en listo o que estaba en bloqueado pero ha terminado su operación de I/O.

f) Se **suspende** un proceso por: su proceso pare, el sistema operativo o el usuario mismo. Esto se puede deber a que el sistema está sobrecargado, falta memoria, *llega un* proceso con mucho mayor prioridad y se suspende a los demás, el operador suspende a *un* proceso que se sospecha está en deadlock o se utiliza el comando *nice* de UNIX.

g) Estando un proceso suspendido bloqueado, el dispositivo por el que estaba esperando el proceso genera una interrupción de terminación. NOTA: Al suspenderse un proceso no se le retiran todos los recursos, únicamente la memoria.

h) Estando en ejecución el proceso solicita un I/O (impresora, teclado, pantalla, mouse...) Este proceso rara vez completa todo su quantum.

i) Se activa antes de que la operación de 1/0 haya terminado de efectuarse.

j) Estando en espera de un dispositivo (terminación de un 1/0) el proceso es suspendido por: su proceso padre, el sistema operativo o el usuario mismo. Esto se puede deber a que el sistema está sobrecargado, falta **memoria, llega** un proceso con mucho mayor prioridad y se suspende a los demás, el operador suspende a un proceso que se sospecha está en deadlock o se utiliza el comando *nice de UNIX.*

k) Cuando un dispositivo interrumpe porque ha finalizado la operación de 1/0, el proceso pasa a listo.

Las siguientes transiciones no son comunes, pero pueden llegar a existir.

k) *Un* proceso se termina antes de ejecutarse. Por ejemplo**,** efectuar un download a un servidor de MP3s cuando se está como invitado y no se han subido archivos.

i) Un proceso padre termina al hijo.

m) Un proceso de muy alta prioridad termina una operación de 1/0 y quita alproceso que esté ejecutándose para poder continuar.

n) Un proceso padre termina al hijo.

3.-0peración de los sistemas de cómputo

La Computadora es una máquina destinada a procesar datos. Este procesamiento involucra dos flujos de información: el de datos y el de instrucciones. Se parte del flujo de datos que han de ser procesados. Este flujo de datos es tratado mediante un flujo de instrucciones de máquina, generado por la ejecución de un programa, y produce el flujo de datos resultado.

La unidad de control de la computadora es la que establece el funcionamiento del

mismo.

Este funcionamiento está basado en una secuencia sencilla, que se repite a alta velocidad.

Esta secuencia consiste en tres pasos:

a) Lectura de memoria principal de la instrucción máquina apuntada por el contador del programa.

b) Incremento del contador de programa para que apunte a la siguiente instrucción máquina.

c) Ejecución de la instrucción.

La mayoría de las computadoras actuales presenta dos o más niveles de ejecución. En el nivel menos permisivo, generalmente llamado **nivel de usuario**

la computadora ejecuta solamente un subconjunto de las instrucciones máquina, quedando prohibidas las demás.

Además, el acceso a determinados registros, o a partes de esos registros, y a determinadas zonas del mapa de memoria *y* de E/S también queda prohibido. En el nivel más permisivo, denominado la computadora ejecuta todas

sus instrucciones sin ninguna restricción y permite el acceso a todos los registros y mapas de direcciones.

Bibliografía

1. AHO, Alfred, *Compiladores, Principios técnicos y herramientas,* México, Addison-Wesley,
2. BECK, Leland, *Introducción a la programación de sistemas,* México, Addison-Wesley.
3. LEMONE, Karen, *Fundamentos de Compiladores,* México, C.E.C.S.A.
4. www.paginasclick.com.mx/cien_tec/ebcQ1
5. www.itlp.edu.mx/publica/tutoriaies/progsts1/tema12.htm
6. www.educared.net/concurso/82/TEXTO13.htmi
7. www.quimika.com/materias/programacion/lenguajes.htm

COMPILADORES E INTÉRPRETES:
Definiciones y conceptos

Genéricamente hablando, en ciencias de la computación, los *procesadores de lenguajes* son aquellos programas destinados a trabajar sobre una entrada que, por la forma como ha sido elaborada, pertenece a un lenguaje particular reconocido o aceptado por el programa en cuestión. Los procesadores de Lenguajes se clasifican como **traductores** o **intérpretes.**

Un *traductores* un programa que recibe una entrada escrita en un lenguaje (el lenguaje fuente) a una salida perteneciente a otro lenguaje (el lenguaje objeto), conservando su significado. En términos computacionales esto significa que tanto la entrada como la salida sean capaces de producir los mismos resultados. Un intérprete, por otra parte, no lleva a cabo tal transformación; en su lugar obtiene los resultados conforme va analizando la entrada. Los traductores son clasificados en **compiladores, ensambladores y preprocesadotes.**

Un *compilador* es un programa que recibe como entrada un programa escrito en un lenguaje de nivel medio o superior (el programa fuente) y lo transforma a su equivalente en lenguaje ensamblador (el programa objeto), e inclusive hasta lenguaje máquina (el programa ejecutable) pero sin ejecutarlo. Un compilador es un traductor. La forma de como llevará a cabo tal traducción es el objetivo central en el diseño de un compilador. Para facilitar la portabilidad, se puede escribir un compilador en el propio lenguaje fuente que traduce, entonces se llama *autocompilador,* aunque, como es obvio, no es posible ejecutarlo la primera vez.

Un *ensamblador* es el programa encargado de llevar a cabo un proceso denominado de ensamble o ensamblado. Este proceso consiste en que, a partir de

un programa escrito en lenguaje ensamblador, se produzca el correspondiente programa en lenguaje máquina (sin ejecutarlo), realizando:

- La integración de los diversos módulos que conforman al programa.
- La resolución de las direcciones de memoria designadas en el área de datos para el almacenamiento de variables, constantes y estructuras complejas; así como la determinación del tamaño de éstas.
- La identificación de las direcciones de memoria en la sección de código correspondientes a los puntos de entrada en saltos condicionales e incondicionales junto con los puntos de arranque de las subrutinas.
- La resolución de los diversos llamados a los servicios o rutinas del sistema operativo, código dinámico y bibliotecas de tiempo de ejecución.
- La especificación de la cantidad de memoria destinadas para las áreas de datos, código, pila y montículo necesarias y otorgadas para su ejecución.
- La incorporación de datos y código necesarios para la carga del programa y su ejecución.

Un *precompilador,* también llamado *preprocesador,* es un programa que se ejecuta antes de invocar al compilador. Este programa es utilizado cuando el programa fuente, escrito en el lenguaje que el compilador es capaz de reconocer (de aquí en adelante denominado *lenguaje anfitrión* en inglés *host language),* incluye estructuras, instrucciones o declaraciones escritas en otro lenguaje (el *lenguaje empotrado* en inglés *embeded language).* El lenguaje empotrado es siempre un lenguaje de nivel superior o especializado (e.g. de consulta, de cuarta generación, simulación, cálculo numérico o estadístico, etcétera). Siendo que el único lenguaje que el compilador puede trabajar es aquel para el cual ha sido escrito, todas las instrucciones del lenguaje empotrado deben ser traducidas a instrucciones del lenguaje anfitrión para que puedan ser compiladas. Así pues un precompilador también es un traductor. Los precompiladores son una solución rápida y barata a la necesidad de llevar las instrucciones de nuevos paradigmas de programación (e.g. los lenguajes de cuarta generación), extensiones a

lenguajes ya existentes (como el caso de C y C++) y soluciones de nivel conceptual superior (por ejemplo paquetes de simulación o cálculo numérico) a código máquina utilizando la tecnología existente, probada, optimizada y confiable (lo que evita el desarrollo de nuevos compiladores). Facilitan la incorporación de las nuevas herramientas de desarrollo en sistemas ya elaborados (por ejemplo, la consulta a bases de datos relaciónales substituyendo las instrucciones de acceso a archivos por consultas en SQL).

Resulta común encontrar que el flujo de proceso en los lenguajes de cuarta generación o de propósito especial puede resultar demasiado inflexible para su implantación en los procesos de una empresa, flujos de negocio o interacción con otros elementos de software y hardware, de aquí que se recurra o prefiera la creación de sistemas híbridos soportados en programas elaborados en lenguajes de tercera generación con instrucciones empotradas de nivel superior o propósito especial.

Un *pseudo compilador* es un programa que actúa como un compilador, salvo que su producto no es ejecutable en ninguna máquina real sino en una *máquina virtual.* Un pseudo compilador toma de entrada un programa escrito en un lenguaje determinado y lo transforma a una codificación especial llamada *código de byte.* Este código no tendría nada de especial o diferente al código máquina de cualquier microprocesador salvo por el hecho de ser el código máquina de un microprocesador ficticio. Tal procesador no existe, en su lugar existe un programa que emula a dicho procesador, de aquí el nombre de máquina virtual.

La ventaja de los pseudocompiladores es que permite tener tantos emuladores como microprocesadores reales existan, pero sólo se requiere un compilador para producir código que se ejecutará en todos estos emuladores. Este método es una de las respuestas más aceptadas para el problema del tan ansiado lenguaje universal o código portable independiente de plataforma.

Un *intérprete* es un programa que ejecuta cada una de las instrucciones y declaraciones que encuentra conforme va analizando el programa que le ha sido dado de entrada (sin producir un programa objeto o ejecutable). La ejecución consiste en llamar a rutinas ya escritas en código máquina cuyos resultados u operaciones están asociados de manera unívoca al significado de las instrucciones o declaraciones identificadas.

Los intérpretes son útiles para el desarrollo de prototipos y pequeños programas para labores no previstas. Presentan la facilidad de probar el código casi de manera inmediata, sin tener que recurrir a la declaración previa de secciones de datos o código, y poder hallar errores de programación rápidamente. Resultan inadecuados para el desarrollo de complejos o grandes sistemas de información por ser más lentos en su ejecución.

COMPILADORES

De forma simple, un compilador es un programa que lee un programa escrito en un lenguaje (el lenguaje fuente) y lo traduce a un programa equivalente en otro lenguaje (el lenguaje destino). Como parte importante del proceso de traducción, el compilador informa de la presencia de errores en el programa fuente. Tradicionalmente, pensamos en un compilador como un programa que traduce un lenguaje fuente como Fortran en el código máquina de algún ordenador.

Los compiladores fueron imprescindibles tras la aparición de los lenguajes de programación. Los lenguajes de programación son el conjunto de reglas y elementos gramaticales de los que un programador dispone para escribir el conjunto de instrucciones necesarias para que una computadora realice aquello que él quiere. En un principio no se disponía de lenguajes de programación o algo que se le pareciera. La programación del computador se llevaba a cabo, literalmente, recableándolo. En un gran panel de contactos que asemejaba a un conmutador telefónico, el operador introducía el programa uniendo componentes

específicos al unirlos mediante un cable. Como tales, los primeros lenguajes fueron los lenguajes ensambladores. Estos requieren de un sólido conocimiento de la arquitectura del computador, por lo que la programación y depuración resultaba un proceso arduo y costoso. A mediados de los 50s surge **Fortran** (contracción del inglés FORmula TRANslation) como el primer lenguaje de alto nivel. Aunque originalmente incluía elementos de un lenguaje ensamblador permitía a los programadores escribir expresiones algebraicas en lugar de la codificación necesaria para el cálculo. Es a partir de la aparición de los lenguajes de alto nivel y los compiladores que la programación se vuelve mucho más cómoda para el programador. A primera vista, la variedad de compiladores puede parecer que nos supera. Hay miles de lenguajes fuente, desde los lenguajes tradicionales de programación como Fortran y Pascal hasta los lenguajes especializados que han aparecido virtualmente en cada área de aplicación de los ordenadores. Los lenguajes destino también son muy variados; un lenguaje destino puede ser otro lenguaje de programación, o el lenguaje máquina de cualquier ordenador entre un microprocesador y un superordenador. Los compiladores son a veces clasificados como de una sola pasada, de múltiples pasadas, "load-and-go", depuradores u optimizadores, dependiendo de cómo han sido construidos o que función se supone que tienen que realizar. A pesar de esta aparente complejidad, las tareas básicas que cualquier compilador debe realizar son esencialmente las mismas. Nuestro conocimiento sobre como organizar y escribir compiladores ha crecido enormemente desde que los primeros compiladores comenzaron a aparecer a principios de los 50. En estos años se consideraba a los compiladores como programas muy difíciles de escribir pero desde entonces se han descubierto técnicas para manejar muchas de las importantes tareas que tienen lugar durante la compilación. También se han desarrollado buenos lenguajes para la implementación, entornos de programación y herramientas software.

Los ordenadores son máquinas que sólo entienden un **lenguaje binario,** es decir, sólo entienden dos estados (encendido y apagado). Conceptualmente podemos

considerar que el ordenador está compuesto por millones de interruptores donde cada interruptor puede tomar los valores 1 ó 0, donde 1 significa que pasa corriente por ese interruptor y 0 significa que no pasa.

Sin embargo, los seres humanos usamos el **lenguaje natural** que cuenta con muchos más símbolos y reglas, que sin embargo no es entendido por el computador. Existen varios tipos de lenguajes de programación:

Lenguajes de bajo nivel: lenguaje máquina
lenguaje ensamblador

Los lenguajes de bajo nivel son aquellos que están más próximos al lenguaje que entiende el computador.

El lenguaje máquina:

- es un lenguaje de 1s y Os que es directamente entendible por el ordenador
- es muy engorroso
- es fácil cometer errores y difícil detectarlos
- depende del hardware

El **lenguaje ensamblador:**

- es un lenguaje de etiquetas, donde se sustituye cada instrucción máquina por una etiqueta más fácil de recordar que un conjunto de 1s y Os

- precisa que otro programa traduzca las etiquetas a lenguaje máquina
- menos engorroso que el lenguaje máquina
- depende del hardware

Lenguajes de alto nivel: Intérpretes
Compiladores

Los lenguajes de alto nivel están más próximos al lenguaje natural. Por tanto no son entendibles directamente por el computador. Es necesario hacer una traducción. Según la forma en que se haga esta traducción existen:

- Lenguajes interpretados (BASIC, PERL, JAVA):

o la traducción se realiza cada vez que se ejecuta el programa

o son más lentos

o para ejecutar el programa se precisa el intérprete

- Lenguajes compilados (PASCAL, C, C++): o la traducción se hace una única vez

o son más rápidos

o se obtiene un ejecutable en lenguaje máquina

o si se cambia el código fuente hay que volver a compilar

Ventajas de lenguajes de alto nivel:

- Más flexibles y fáciles de aprender
- Independencia de la máquina y del SO.
- Portabilidad a otras plataformas.

3.1 Determinación de la Factibilidad

Factibilidad se refiere a la disponibilidad de los recursos necesarios para llevar a cabo los objetivos o metas señalados, la factibilidad se apoya en 3 aspectos básicos:

- Operativo.
- Técnico.
- Económico.

El éxito de un proyecto esta determinado por el grado de factibilidad que se presente en cada una de los tres aspectos anteriores.

Estudio de Factibilidad.

Sirve para recopilar datos relevantes sobre el desarrollo de un proyecto y en base a ello tomar la mejor decisión, si procede su estudio, desarrollo o implementación.

Objetivo de un Estudio de Factibilidad.

1.- Auxiliar a una organización a lograr sus objetivos.

2.- Cubrir la metas con los recursos actuales en las siguientes áreas.

a). Factibilidad Técnica.

- Mejora del sistema actual.
- Disponibilidad de tecnología que satisfaga las necesidades.

b).- Factibilidad Económica.

- Tiempo del analista.
- Costo de estudio.
- Costo del tiempo del personal.
- Costo del tiempo.
- Costo del desarrollo / adquisición.

c).- Factibilidad Operativa.

- Operación garantizada.
- Uso garantizado.

DEFINICIÓN DE OBJETIVOS.

La investigación de factibilidad en un proyecto que consiste en descubrir cuales son los objetivos de la organización, luego determinar si el proyecto es útil para que la empresa logre sus objetivos. La búsqueda de estos objetivos debe contemplar los recursos disponibles o aquellos que la empresa puede proporcionar, nunca deben definirse con recursos que la empresa no es capaz de dar.

En las empresas se cuenta con una serie de objetivos que determinan la posibilidad de factibilidad de un proyecto sin ser limitativos. Estos objetivos son los siguientes:

- Reducción de errores y mayor precisión en los procesos.
- Reducción de costos mediante la optimización o eliminación de recursos no necesarios.
- Integración de todas la áreas y subsistemas de la empresa.
- Actualización y mejoramiento de los servicios a clientes o usuarios.
- Aceleración en la recopilación de datos.
- Reducción en el tiempo de procesamiento y ejecución de tareas.
- Automatización óptima de procedimientos manuales.

3.2 Recursos de los estudios de Factibilidad

La determinación de los recursos para un estudio de factibilidad sigue el mismo patrón considerado por los objetivos vistos anteriormente, el cual deberá revisarse y evaluarse si se llega a realizar un proyecto, estos recursos se analizan en función de tres aspectos:

- Operativos.
- Técnicos.
- Económicos.

Factibilidad Operativa.

Se refiere a todos aquellos recursos donde interviene algún tipo de actividad (Procesos), depende de los recursos humanos que participen durante la operación del proyecto. Durante esta etapa se identifican todas aquellas actividades que son necesarias para lograr el objetivo y se evalúa y determina todo lo necesario para llevarla a cabo.

Factibilidad Técnica.

Se refiere a los recursos necesarios como herramientas, conocimientos, habilidades, experiencia, etc., que son necesarios para efectuar las actividades o procesos que requiere el proyecto. Generalmente nos referimos a elementos tangibles (medibles). El proyecto debe considerar si los recursos técnicos actuales son suficientes o deben complementarse.

Factibilidad Económica.

Se refiere a los recursos económicos y financieros necesarios para desarrollar o llevar a cabo las actividades o procesos y/o para obtener los recursos básicos que deben considerarse son el costo del tiempo, el costo de la realización y el costo de adquirir nuevos recursos.

Generalmente la factibilidad económica es el elemento mas importante ya que a través de el se solventan las demás carencias de otros recursos, es lo mas difícil de conseguir y requiere de actividades adicionales cuando no se posee.

3.3 Presentación de un estudio de Factibilidad

Un estudio de factibilidad requiere ser presentado con todas la posibles ventajas para la empresa u organización, pero sin descuidar ninguno de los elementos necesarios para que el proyecto funcione. Para esto dentro de los estudios de factibilidad se complementan dos pasos en la presentación del estudio:

- Requisitos Óptimos.
- Requisitos Mínimos.

El primer paso se refiere a presentar un estudio con los **requisitos óptimos** que el proyecto requiera, estos elementos deberán ser los necesarios para que las actividades y resultados del proyecto sean obtenidos con la máxima eficacia.

El segundo paso consiste en un estudio de requisitos mínimos, el cual cubre los **requisitos mínimos** necesarios que el proyecto debe ocupar para obtener las metas y objetivos, este paso trata de hacer uso de los recursos disponibles de la empresa para minimizar cualquier gasto o adquisición adicional.

Un estudio de factibilidad debe representar gráficamente los gastos y los beneficios que acarreará la puesta en marcha del sistema, para tal efecto se hace uso de la **curva costo-beneficio.**

3.6. ANÁLISIS COSTO-BENEFICIO

El análisis Costo-Beneficio, permitir definir la factibilidad de las alternativas planteadas o del proyecto a ser desarrollado.

a) Objetivo:

La técnica de Análisis de Costo - Beneficio, tiene como objetivo fundamental proporcionar una medida de los costos en que se incurren en la realización de un proyecto informático, y a su vez comparar dichos costos previstos con los beneficios esperados de la realizaci0n de dicho proyecto.

b) Utilidad:

La utilidad de la presente técnica es la siguiente:

* Para valorar la necesidad y oportunidad de acometer la realización del proyecto.
* Para seleccionar la alternativa más beneficiosa para la realización del proyecto.
* Para estimar adecuadamente los recursos económicos necesarios en el plazo de realizaci0n del proyecto.

c) Descripción :

Si queremos realizar un Análisis de Costo - Beneficio fiable, debemos de seguir los siguientes pasos:

3.6.3.1 Producir estimaciones de costos-beneficios.

3.6.3.2 Determinar la viabilidad del proyecto y su aceptación.

3.6.3.1 Producir estimaciones de costos - beneficios.

Lo primero que debemos de realizar es elaborar dos tipos de listas, la primera con lo

requerido para implantar el sistema y la segunda

con los beneficios que traer consigo el nuevo sistema.

Antes de redactar la lista es necesario tener presente que los costos son tangibles, es decir se pueden medir en alguna unidad económica, mientras que los beneficios pueden ser tangibles y no tangibles, es decir pueden darse en forma objetiva o subjetiva.

La primera lista (requerimiento para implantar el sistema) deber estar integrada por requerimientos necesarios para ejecutar el proyecto, el valor que tiene cada uno y sus posibles variaciones de acuerdo a la inflación, de esta forma, la Dirección obtendrá información detallada de como se distribuyen sus recursos.

Para elaborar la lista se necesita contar con experiencia en la participación de proyectos similares, as¡ como datos hist0r¡cos que le permitan estimar adecuadamente los requerimientos necesarios para ejecutar el proyecto.

Para mayor explicación proporcionaremos ejemplos de algunos gastos necesarios para ejecutar un proyecto en informática:

* Costos de equipo, donde se detallar el tipo de equipo requerido para el proyecto.

* Costos de infraestructura, donde se determinar el ambiente adecuado para el equipo, as¡ como el mobiliario requerido para cada uno de ellos.

* Costo de personal, se determinar el número de personal requerido tanto técnico como administrativo, sus características y el tipo
de capacitación que se le deber de proporcionar a cada empleado.

* Costo de materiales, se determinar n todos los materiales necesa- rios para el desarrollo del proyecto.

* Costo de consultoría, se determinar el tipo de garantía a proporcionar a la Dirección luego de desarrollado el sistema.

Esta valoración ser realizada en las reas correspondientes. La segunda lista, beneficios que traer consigo el proyecto, ser elaborado en forma subjetiva y deber n estar acorde a los requerimientos de información de los usuarios.

Por ejemplo, los beneficios proporcionados por un proyecto de informática pueden ser:

* El aumento de las cuentas debido al mayor servicio de los clientes.

* La mejora en la toma de decisiones debido a un mejor soporte informático.

* La optimización de los procedimientos administrativos.

3.6.3.2 Determinar la viabilidad del proyecto y su aceptación.

Para determinar si un proyecto es conveniente o no realizarlo es necesario realizar un estudio de viabilidad, donde se determinar si el proyecto es factible o no; para lo cual nos basaremos en uno de los métodos siguientes:

* Retorno de la inversión.

Este método consiste en calcular el costo y beneficio anual, sabiendo el costo total al iniciar el proyecto ®C0 .

Este método nos permitir saber en que a»o se recuperar el costo total inicialmente estimado en el proyecto, donde el año de recuperación de la inversión es cuando la sumatoria de los beneficios netos es igual al costo total del inicio del proyecto (ZBen.Neto =C0).

Año	Costo	Beneficio	Beneficio Neto
0	C0	0	
1	C1	B1	B1-C1
2	C2	B2	B2-C2
n	Cn	Bn	Bn-Cn

* Valor actual.

Este método nos permitir tener en cuenta que un gasto invertido durante un cierto tiempo produce un beneficio.

Con este método podremos determinar la cantidad de dinero que es viable invertir inicialmente para que se recupere la inversión en un periodo de tiempo determinado por la Dirección.

Se deber calcular en primer lugar, el beneficio neto que se obtendrá cada ano. Dicho beneficio no es real, ya que se debe estimar el valor real de dicha cantidad en el a»o n.

Para ello se aplicar la siguiente formula: Valor Actual =
Beneficio Neto / (1 + r/100)n Donde:

n = año ,1,2,3.... n

r = interés utilizado en la evaluación.

Para determinar la viabilidad del proyecto, se deber estudiar en

cuantos anos se recuperar la inversión realizada inicialmente y si esta inversi0n es retornada en un periodo de año fijado previamente por la Dirección.

Si la inversión es el C0, se determinar la viabilidad del proyecto consultando la siguiente tabla.

Año	Costo	Beneficio	Valor Actual
0	C0		
1	C1	B1	VA1=(B1-C1)/(1+r/100)
2	C2	B2	VA2=(B2-C2)/(1+r/100)2
			VA3=(B3-C3)/(1+r/100)3
N	Cn	Bn	Van=(Bn-Cn)/(1+r/100)n

El proyecto ser viable si la sumatoria del Valor Actual es mayor al Costo Inicial (ZVAi > C0)a lo largo del proyecto.

3.6. ANÁLISIS COSTO-BENEFICIO

El análisis Costo-Beneficio, permitir definir la factibilidad de las alternativas planteadas o del proyecto a ser desarrollado.

a) Objetivo:

La técnica de Análisis de Costo - Beneficio, tiene como objetivo fundamental proporcionar una medida de los costos en que se incurren en la realización de un proyecto informático, y a su vez comparar dichos costos previstos con los beneficios esperados de la realización de dicho proyecto.

b) Utilidad:

La utilidad de la presente técnica es la siguiente:

* Para valorar la necesidad y oportunidad de acometer la realización del proyecto.
* Para seleccionar la alternativa m s beneficiosa para la realización del proyecto.
* Para estimar adecuadamente los recursos económicos necesarios en el plazo de realizaci0n del proyecto.

c) Descripción :

Si queremos realizar un Análisis de Costo - Beneficio fiable, debemos de seguir los siguientes pasos:

3.6.3.1 Producir estimaciones de costos-beneficios.

3.6.3.2 Determinar la viabilidad del proyecto y su aceptación.

3.6.3.1 Producir estimaciones de costos - beneficios.

Lo primero que debemos de realizar es elaborar dos tipos de listas, la primera con lo requerido para implantar el sistema y la segunda con los beneficios que traer consigo el nuevo sistema.

Antes de redactar la lista es necesario tener presente que los costos son tangibles, es decir se pueden medir en alguna unidad económica, mientras que los beneficios pueden ser tangibles y no tangibles, es decir pueden darse en forma objetiva o

subjetiva.

La primera lista (requerimiento para implantar el sistema) deber estar integrada por requerimientos necesarios para ejecutar el proyecto, el valor que tiene cada uno y sus posibles variaciones de acuerdo a la inflación, de esta forma, la Dirección obtendrá información detallada de como se distribuyen sus recursos.

Para elaborar la lista se necesita contar con experiencia en la participación de proyectos similares, as¡ como datos histéricos que le permitan estimar adecuadamente los requerimientos necesarios para ejecutar el proyecto.

Para mayor explicación proporcionaremos ejemplos de algunos gastos necesarios para ejecutar un proyecto en informática:

* Costos de equipo, donde se detallar el tipo de equipo requerido para el proyecto.

* Costos de infraestructura, donde se determinar el ambiente adecuado para el equipo, as¡ como el mobiliario requerido para cada uno de ellos.

* Costo de personal, se determinar el número de personal requerido tanto técnico como administrativo, sus características y el tipo

  de capacitación que se le deber de proporcionar a cada empleado.

* Costo de materiales, se determinar n todos los materiales necesarios para el desarrollo del proyecto.

* Costo de consultoria, se determinar el tipo de garantía a proporcionar a la Dirección luego de desarrollado el sistema.

Esta valoración ser realizada en las reas correspondientes. La segunda lista, beneficios que traer consigo el proyecto, ser elaborado en forma subjetiva y deber n estar acorde a los requerimientos de información de los usuarios.

Por ejemplo, los beneficios proporcionados por un proyecto de informática pueden ser:

* El aumento de las cuentas debido al mayor servicio de los clientes.

* La mejora en la toma de decisiones debido a un mejor soporte informático.

* La optimización de los procedimientos administrativos.

3.6.3.2 Determinar la viabilidad del proyecto y su aceptación.

Para determinar si un proyecto es conveniente o no realizarlo es necesario realizar un estudio de viabilidad, donde se determinar si el proyecto es factible o no; para lo cual nos basaremos en uno de los métodos siguientes:

* Retorno de la inversión.

Este método consiste en calcular el costo y beneficio anual, sabiendo el costo total al iniciar el proyecto ®C0~.

Este método nos permitir saber en que año se recuperar el costo total inicialmente estimado en el proyecto, donde el año de recuperación de la inversión es cuando la sumatoria de los beneficios netos es igual al costo total del inicio del proyecto (ZBen.Neto =C0).

Año	Costo	Beneficio	Beneficio Neto
0	C0	0	
1	C1	B1	B1-C1
2	C2	B2	B2-C2
n	Cn	Bn	Bn-Cn

* Valor actual.

Este método nos permitir tener en cuenta que un gasto invertido durante un cierto tiempo produce un beneficio.

Con este método podremos determinar la cantidad de dinero que es viable invertir inicialmente para que se recupere la inversión en un periodo de tiempo determinado por la Dirección.

Se deber calcular en primer lugar, el beneficio neto que se obtendrá cada año. Dicho beneficio no es real, ya que se debe estimar el valor real de dicha cantidad en el año n.

Para ello se aplicar la siguiente formula:

Valor Actual = Beneficio Neto / (1 + r/100)n

Donde :

n = año ,1,2,3,...,n

r = interés utilizado en la evaluación

Para determinar la viabilidad del proyecto, se deber estudiar en cuantos anos se recuperar la inversión realizada inicialmente y si esta inversi0n es retornada en un periodo de año fijado previamente por la Dirección.

Si la inversi0n es el C0, se determinar la viabilidad del proyecto consultando la siguiente tabla.

Año	Costo	Beneficio	Valor Actual
0	C0		
1	C1	B1	VA1=(B1-C1)/(1+r/100)
2	C2	B2	VA2=(B2-C2)/(1+r/100)2
			VA3=(B3-C3)/(1+r/100)3
n	Cn	Bn	Van=(Bn-Cn)/(1+r/100)n

El proyecto ser viable si la sumatoria del Valor Actual es mayor al Costo Inicial (ZVAi > C0)a lo largo del proyecto.

SISTEMAS OPERATIVOS

CAPITULO 1: INTRODUCCIÓN

Un Sistema Operativo es un programa que actúa como interface entre un usuario de computadora y el hardware de la misma. El propósito de un SO es proveer de un ambiente de trabajo en el cual el usuario puede ejecutar programas. Por ende, el primer objetivo de un SO es hacer que un equipo pueda ser usado convenientemente. Un segundo objetivo es usar el hardware del equipo en forma eficiente.

Para entender qué son los sistemas operativos, es necesario estudiar como fueron desarrollados, por lo que veremos rápidamente la evolución de los mismos, desde los SO primitivos hasta los actuales de tiempo-compartido y multiprogramación.

Hay muchas razones importantes para estudiar los SO:

a) para el uso de propósito especial, uno puede desarrollar uno propio o modificar alguno existente (esto es particularmente cierto con el SO UNIX, donde una de las razones de su éxito en universidades de EE.UU. es que el 90% del mismo esta escrito en lenguaje de alto nivel -C-, por lo que sus modificaciones y agregados se hacen sumamente sencillos);

b) La selección del SO y sus opciones es una decisión importante en la mayoría de las instalaciones de computación;

c) el usuario debe interactuar con el SO para lograr su tarea, ya que su primer contacto con el computador;

d) muchos conceptos y técnicas que se encuentran en los SO se pueden aplicar en forma general.

1.1.- QUE ES UN SO ?

Un SO es una parte importante de prácticamente cualquier computador.

Un sistema electrónico de procesamiento de datos puede dividirse en las siguientes partes:

1) El hardware (cpu, memoria, canales, dispositivos de E/S, etc.)

2) El SO

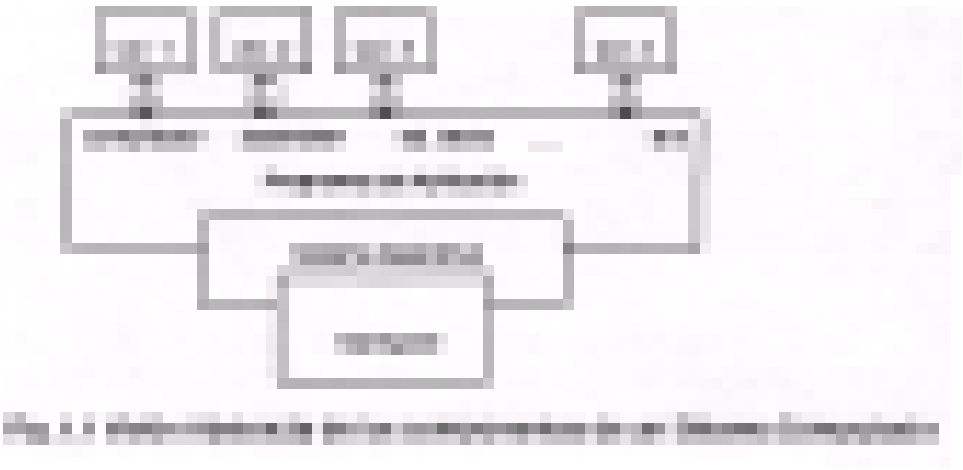
3) Resto del software de base: compiladores, ensambladores, cargadores, editores de

texto, utilitarios de base de datos, utilitarios de depuración (debuggers), etc.

4) Programas de aplicación y librerías.

5) Usuarios (personas, maquinas u otras computadoras)

El hardware provee los recursos básicos del sistema. El software de base define las maneras en que esos recursos serán usados para resolver los problemas computacionales de los usuarios. Pueden haber muchos usuarios tratando de resolver distintos problemas. El SO controla y coordina el uso del hardware entre los distintos requerimientos.



Para nuestro estudio de SO, dividiremos a este en 4 administradores bien diferenciados:

1) Administrador del procesador;
2) Administrador de memoria;
3)Administrador de dispositivos y
4) Administrador de información.

Un SO es similar a un gobierno. Los recursos básicos de un computador lo dan su hardware, software y datos. El SO provee el uso apropiado de estos recursos. Igual que un gobierno el SO no ejerce ninguna tarea útil por si mismo: simplemente brinda un ambiente de trabajo para que otros programas puedan realizar alguna tarea útil.

Podemos ver a un SO como un asignatario de recursos. De todos los recursos de un sistema de computación, muchos pueden ser requeridos para resolver un problema: tiempo de cpu, espacio en memoria, espacio en disco, dispositivos de E/S, etc.
El SO actúa como el administrador de estos recursos y los asigna a programas específicos y a usuarios para que puedan realizar sus trabajos. Ya que los pedidos pueden ser muchos, pudiendo existir conflictos, el SO decide a quien asigna recursos, en forma tal que el sistema funcione bien y eficientemente.

Otro punto de vista distinto de enfocar a un SO es el de la necesidad de controlar diversos dispositivos de E/S y programas de usuarios. Un SO es un programa de control (varios SO han incorporado este punto de vista en sus nombres: CP/M significa Control Program for Microcomputers, por ejemplo.)

En general, sin embargo, no existe una definición completamente adecuada para un SO. Estos existen pues constituyen una manera razonable de resolver el problema de crear un sistema de computación que pueda ser usado. El objetivo fundamental de los sistemas de computación es ejecutar programas de usuarios y resolver sus problemas. Precisamente, para cumplir este objetivo, se construye el hardware del sistema.

Como esto por si solo no es muy fácil de usar, se ha desarrollado el software de base. Las funciones comunes de controlar y asignar recursos se ha desarrollado en una sola pieza de software: el SO.

Los 2 objetivos de los SO, conveniencia y eficiencia, a veces resultan en soluciones contradictorias. En el pasado, pesaba más la eficiencia que la conveniencia. Así, mucha de la teoría de SO se concentra en el uso óptimo de los recursos del equipo.
La arquitectura del equipo y su SO, ejercen una influencia mutua. Para facilitar el uso del hardware precisamente se desarrollaron los SO. A medida que los SO fueron usados, aparecía en forma obvia que cambios en el diseño del hardware podían simplificar el SO.

1.2. SISTEMAS PRIMITIVOS

Inicialmente, solo existía el hardware. Las primeras computadoras fueron (físicamente) máquinas muy grandes que se controlaban desde una consola. El programador podía escribir un programa y luego controlar su ejecución desde la consola del operador. Primeramente, el programa se cargaba manualmente a memoria desde un panel con llaves, cinta de papel perforada o tarjetas. Luego, se debían presionar los botones apropiados para cargar la dirección inicial de memoria y comenzar la ejecución del programa. El programador / operador podía monitorear la ejecución mediante las luces del panel frontal. Si existían errores, se podía parar la ejecución del programa, examinar los contenidos de memoria y registros y depurar el programa directamente desde la consola. La salida era impresa o de lo contrario se perforaba una cinta de papel o tarjetas para una impresión posterior.

En estos sistemas el programador era el operador. Existían planillas de reserva de uso de máquina, donde cada uno fijaba día y hora para correr sus programas. Los problemas de esta técnica eran que si por un inconveniente uno se retrasaba inevitablemente debía parar, recolectar lo que podía y volver más tarde u otro día para continuar el trabajo. Si por el contrario, las cosas iban muy bien y el trabajo era terminado antes del tiempo previsto, era muy probable que el computador estuviera sin uso todo el tiempo ganado.

Con el tiempo, nuevo hardware y software aparecieron. Lectores de tarjetas, impresoras de línea y cintas magnéticas se hicieron comunes. Se diseñaron ensambladores, cargadores, link editores para facilitar la programación. Se crearon librerías con funciones comunes para no tener que re-escribirlas constantemente.

Las rutinas que ejecutaban E/S fueron especialmente importantes. Cada nuevo dispositivo de E/S tenía sus propias características, lo que requería una programación muy cuidadosa. Para ello se escribió una subrutina especial para manejo de cada dispositivo de E/S. Estas reciben el nombre de drivers. Un driver de un dispositivo conoce todos los buffers, banderas, registros, bits de control y de status que deben usarse para ese dispositivo en particular. Una tarea simple como sería leer un carácter de una cinta perforada requiere una secuencia compleja de operaciones específicas. En vez de escribir el código necesario cada vez que se necesite, el driver de ese dispositivo era usado simplemente de la librería.

Más tarde aparecen lenguajes como el FORTRAN, COBOL, etc. que volvieron a la programación más simple, pero la operación más compleja. Por ejemplo, para preparar un programa en FORTRAN para su ejecución el programador primeramente debía cargar el compilador en la computadora. El compilador normalmente estaba grabado en cinta, por lo que tenía que montarse el carrete en la unidad.
El programa sería leído probablemente con una lectora de tarjetas. La salida era en assembler por lo que tenía que montarse la cinta con el ensamblador. La salida debía linkearse para soportar las rutinas almacenadas en la librería. Finalmente, el programa objeto estaba listo para su ejecución. Debía cargarse a memoria y dejarse igual que antes, desde la consola.
Observemos que existía un tiempo considerable de preparación para la ejecución de un trabajo. Cada trabajo tenía muchos pasos: carga de la cinta con el compilador FORTRAN, ejecución del mismo, desmontaje de la cinta, montaje de la cinta con el ensamblador, carga del programa objeto y ejecución. Si ocurría algún error en cualquier paso, se debía recomenzar totalmente.

1.3. MONITOR SIMPLE

El tiempo de preparación para la ejecución de programas era realmente un problema, ya que durante ese tiempo la cpu estaba inactiva. Como los costos de los equipos eran muy elevados (millones de dólares), una alta utilización de cpu era necesario. La solución en ese momento fue tomar operadores profesionales. Así los programadores dejaron de operar los equipos. Una vez que finalizaba un trabajo el operador iniciaba el

próximo. La planilla de reserva de equipo ya no fue necesaria. Pero obviamente, los operadores no podían depurar programas. Por lo tanto, si ocurría un error, se copiaba el contenido de memoria completo y de registros y el programador se encargaba luego del problema. Por supuesto, ahora los programadores tenían mucha mayor dificultad para solucionarlos.

Otra forma de reducir el tiempo de preparación fue la de elegir correr en paquete los trabajos similares. (Batch). Por ejemplo si tenía que ejecutar un trabajo en FORTRAN, otro en COBOL y otro en FORTRAN, se corría el primero y el último y luego el segundo.

El primer SO rudimentario apareció con la creación de un secuenciador de trabajos automático. El mismo era un procedimiento de transferencia automática del control de un trabajo a otro. Un programa muy chico, llamado monitor residente, estaba siempre en memoria. Inicialmente al prender el equipo, el control de la máquina estaba en el monitor residente, quien transfiere el control del sistema a un programa. Cuando el programa termina su ejecución, el control vuelve al monitor residente, quien cederá el control al próximo trabajo. Así el monitor va cediendo en forma secuencial el control a los distintos programas a ejecutarse.

Pero, ¿cómo sabe el monitor residente qué programa ejecutar? Previamente, el operador debía dar una corta descripción de qué programas correr y con qué datos. Se inventaron las tarjetas de control, que daban esa información al monitor. La idea es muy simple. Adicionalmente al programa y datos para un trabajo, se incluyen tarjetas especiales que instruyen al monitor residente el programa a correr. Por ejemplo, un usuario puede requerir la ejecución de 3 programas: el compilador FORTRAN (FTN), el ensamblador (ASM) y el programa objeto de él. Tendremos las siguientes tarjetas de

control:

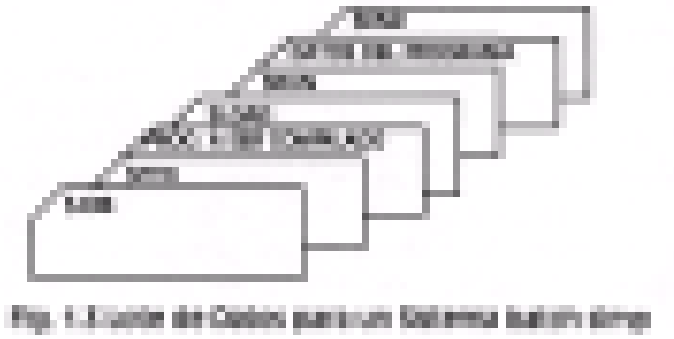
* $FTN: Ejecutar el compilador FORTRAN.
* $ASM: Ejecutar el ensamblador.
* $RUN: Ejecutar el programa de usuario.

Estas le dicen al monitor qué programas ejecutar.

Usamos 2 tarjetas de control adicionales para marcar los límites de cada trabajo

* $TRB: Primer tarjeta del trabajo. * $FIN: Ultima tarjeta del trabajo

Muchos sistemas usan el signo $ en la primer columna para identificar a una tarjeta de control. Por ejemplo, IBM usa // en las 2 primeras columnas para su JCL (Job Control Languaje).



Un monitor residente posee varias partes perfectamente identificables: el intérprete de tarjetas de control, que periódicamente necesitará un cargador para cargar los programas del sistema y del usuario. Por lo tanto, también el cargador forma parte del monitor residente. Estos 2 módulos necesitaran realizar operaciones de E/S y por ende el monitor contendrá un conjunto de drivers para cada dispositivo del sistema.

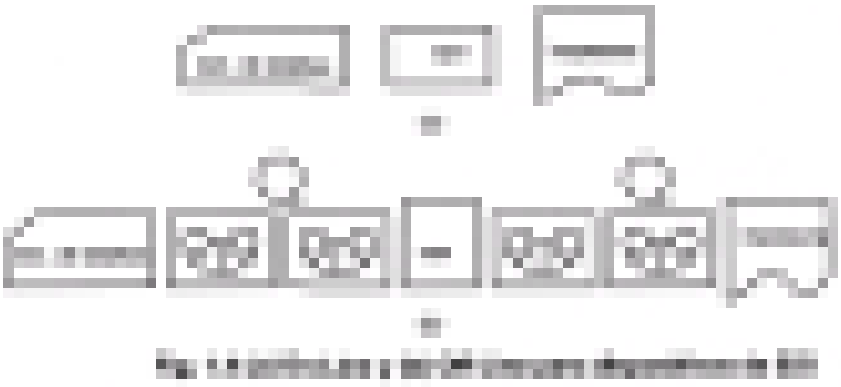
1.4. PERFORMANCE

Aún con el secuenciamiento de trabajos, la cpu está a menudo ociosa. Esto sucede por la baja velocidad intrínseca de los dispositivos electromecánicos de E/S con respecto a los electrónicos. Una cpu muy lenta está por las centenas de nanosegundos o el microsegundo (millones de instrucciones por segundo), en cambio una lectora de tarjetas rápida lee 1000 tarjetas por minuto. Así las diferencias de magnitudes en el peor de los casos es de 1000 a 1. Un compilador podía procesar unas 300 o más

tarjetas por segundo mientras que una lectora unas 2 por segundo. Eso significa que la compilación de 1200 tarjetas requiere 4 segundos de tiempo de cpu y 60 segundos de lectura. La cpu así está ociosa 56 de los 60 segundos o sea casi el 94% del tiempo total.

1.4.1. OPERACIÓN FUERA DE LÍNEA

Con el transcurso del tiempo, la mejora en la tecnología resultó en dispositivos de E/S más veloces; pero las velocidades de cpu aumentaron mucho más, por lo que el problema no pudo resolverse. Una solución muy común fue reemplazar lectoras de tarjetas (dispositivos de entrada) e impresoras de línea (dispositivos de salida) por unidades de cinta magnética. Los sistemas de fines de la década del 50 y comienzos del 60 procesaban en batch y leyendo de lectoras de tarjetas e imprimiendo las salidas. Lo que se hizo fue previamente copiar a cinta lo leído por las lectoras.



Cuando la cinta estaba casi llena, se la montaba en la unidad del computador. Igualmente, toda la salida se copiaba a cinta para imprimirse más tarde. Así, tanto las lectoras como las impresoras se operaban fuera de línea. Para eso se desarrollaron dispositivos de propósitos especiales que tenían el hardware necesario para la realización de estas tareas. Otra solución fue dedicar a estos trabajos equipos computadores muy pequeños que eran satélites del principal. Así, la velocidad de cpu quedaba limitada ahora a la velocidad de las unidades de cinta, que era muy superior a la de las impresoras y lectores. La real ventaja del procesamiento fuera de línea está en la posibilidad de usar múltiples instalaciones satélite por una cpu. La desventaja radica en que ahora el tiempo para ejecutar un trabajo en particular es mucho mayor.

(a)Operación en línea (b) Operación de dispositivos de E/S fuera de línea

1.4.2. BUFFERING

Otra solución para la lentitud de los dispositivos de E/S es el uso de lo que se conoce como buffering, que intenta mantener tanto la cpu como el dispositivo de E/S ocupados todo el tiempo. Luego de leerse los datos y cuando la cpu va a comenzar a procesarlos, se instruye al dispositivo de entrada que comience a leer la próxima entrada inmediatamente. Con suerte, al finalizar la cpu su trabajo ya tendremos disponibles nuevos datos. Igual método puede usarse para las salidas.

En la práctica sin embargo, siempre la cpu tiende a esperar. Pero hay una mejora con respecto a no usar buffers. Esta técnica sin embargo tiene sus problemas. Por ejemplo, debe detectarse el momento en que el dispositivo de E/S ha terminado su operación lo antes posible. La próxima operación de E/S debe empezar recién cuando la anterior ha finalizado.

Las interrupciones resuelven este problema: ni bien el dispositivo de E/S termina una operación manda una señal de interrupción a la cpu. Esta para su trabajo y transfiere el control a una dirección de memoria fija. Las instrucciones localizadas en esas direcciones son básicamente una rutina de servicio de interrupciones que chequea si el buffer no está lleno (dispositivo de entrada) o vacío (dispositivo de salida). Si ésta condición se cumple se inicia la próxima operación de E/S. La cpu sigue con su trabajo hasta terminar. Así, tanto los dispositivos de E/S como la cpu pueden trabajar a máxima velocidad. Las interrupciones son una parte importante en la arquitectura de un equipo. Cada computadora tiene su propio mecanismo de interrupciones, pero muchas de las funciones son comunes. La interrupción transfiere control a una rutina de servicio de interrupciones. Generalmente se reserva un conjunto de direcciones bajas para ubicarlas (100 o menos). Se usa un arreglo o vector de direcciones que están indexadas de acuerdo al número de dispositivo que pidió la interrupción.

La rutina de interrupciones también debe guardar la dirección de la instrucción que ha interrumpido. Arquitecturas más sofisticadas incluso prevén manejar la llegada de una interrupción en el momento que otra está siendo procesada, basándose en un sistema de prioridades. Consideremos, por ejemplo un driver de terminal de video. Cuando va a ser mostrada una línea en la pantalla se envía el primer carácter. Luego de mostrado,

la terminal interrumpe a la cpu. Cuando llega el pedido la cpu seguramente estará por ejecutar alguna instrucción (si la cpu está ejecutando una instrucción en ese momento, la interrupción se mantiene pendiente hasta su finalización). Se salva entonces la dirección de la instrucción y el control es transferido a la rutina de servicio de interrupción (rsi) de terminal. La *rsi* salva los contenidos de cualquier registro que necesite usar; chequea cualquier error que pueda haber ocurrido en la última operación de salida; chequea el buffer donde están almacenados los caracteres a mostrarse; si hay caracteres extrae uno del buffer ajustando punteros y contadores; el carácter es enviado a la pantalla y los bits de control son puestos para permitir una nueva interrupción luego que el carácter sea mostrado. Luego la *rsi* restaura los contenidos de los registros salvados y transfiere el control a la instrucción interrumpida.

Si uso una terminal conectada a 1200 baudios, esta puede aceptar y mostrar 1 carácter cada 8 milisegundos aproximadamente (8000 microsegundos). A una buena *rsi* le toma solo 20 microsegundos transferir del buffer a pantalla un carácter. Tenemos así 7980 microsegundos de cpu disponible cada 8000 microsegundos.

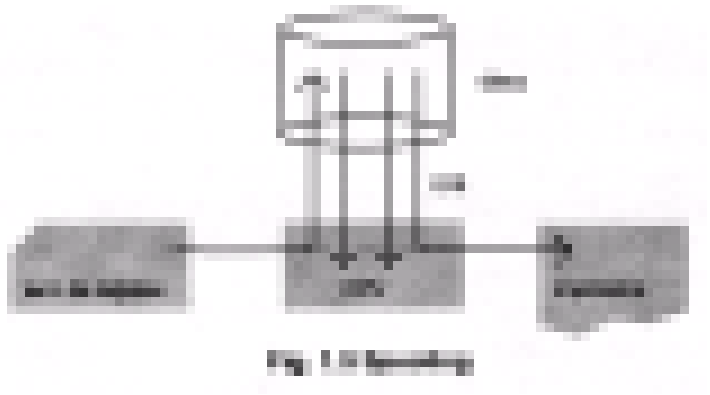
Otros dispositivos como cintas son mucho más veloces y pueden transmitir información a velocidades cercanas a las de memoria. En estos casos, por ejemplo, la cpu necesitaría 20 microsegundos para cada interrupción, pero éstas llegando cada 4 microsegundos. Para resolver este problema se usa el Acceso Directo a Memoria (DMA) para dispositivos de alta velocidad. Luego de actualizar buffers, punteros y contadores para el dispositivo de E/S, se transfiere un bloque completo de datos a o de memoria directamente sin intervención de cpu. Se genera solo una interrupción por bloque en vez de una por byte o palabra. (Las longitudes típicas de bloques varían entre 128 bytes y 32 Kbytes)

¿Cómo afecta el buffering a la performance?

Si las velocidades promedio de cpu y dispositivos son aproximadamente iguales, esta técnica permite que la cpu se encuentre un poco más adelantada o retrasada que el dispositivo, pero ambos trabajaran a máxima velocidad. Sin embargo, si en promedio la cpu es mucho más veloz que los dispositivos, el uso de buffers es de poca ayuda, ya que la cpu siempre tendrá que esperar al encontrar buffers vacíos o llenos.

1.4.3. SPOOLING

La preparación de trabajos fuera de línea fue rápidamente reemplazada en la mayoría de los sistemas al disponerse en forma generalizada de discos. Así se eliminó el problema que presentaban las cintas de no poder ser leídas hasta que hubiesen sido escritas completamente y rebobinadas. Con los discos, las tarjetas leídas desde las lectoras se almacenan a imagen en los mismos. Para impresión, en forma similar: primeramente la salida va a un buffer, el que es copiado a disco para ser impreso más tarde. Esta técnica se llama spooling (Simultaneous Peripheral Operations On-Line).



Esencialmente se usa el disco como un gran buffer, para ser leído lo mas rápidamente posible (dispositivos de entrada) o para aceptar operaciones de salida hasta que los dispositivos acepten el pedido. Acá, a diferencia del buffering, puedo solapar operaciones de E/S con operaciones de cpu pero de otros trabajos. El spooling beneficia en forma directa la performance de un sistema, al solo costo de espacio en disco y el uso de algunas pocas tablas. Adicionalmente el spooling provee una muy importante estructura de datos: un conjunto de trabajos.

Tendremos generalmente una serie de trabajos escritos en disco esperando ejecutarse. Esto permite al SO seleccionar cual trabajo ejecutar primero. Cuando los trabajos vienen en tarjetas o en cinta deben ejecutarse necesariamente en forma Secuencial (PEPS), en cambio en disco, un dispositivo de acceso directo, el concepto del planificador de trabajos es posible.

1.5. MULTIPROGRAMACIÓN

El aspecto más importante de un planificador de trabajos es su capacidad de

multiprogramación. La operación fuera de línea, buffering y spooling tienen sus limitaciones. Un solo usuario, en general, no puede mantener la cpu o todos los dispositivos de E/S ocupados todo el tiempo. La multiprogramación intenta aumentar la utilización de cpu.

La idea es la siguiente: el SO toma uno de los trabajos de la cola de trabajos y comienza a ejecutarlo. Eventualmente, este trabajo deberá esperar por algo, como por ejemplo montar una cinta, un comando a ejecutarse desde el teclado o la terminación de una operación de E/S. En un sistema monousuario la cpu quedaría ociosa esperando. En un sistema con multiprogramación el SO operativo toma otro trabajo y lo ejecuta. Cuando este trabajo necesita esperar, la cpu toma otro trabajo y así sucesivamente. Cuando el primer trabajo termino su espera, tendrá nuevamente disponible el uso de cpu. En tanto haya algún trabajo para ejecutar la cpu nunca estará ociosa.

Los sistemas con capacidad de multiprogramación son bastante sofisticados. Para que existan varios trabajos listos para correr, deben estar todos en memoria.

Mantener varios programas en memoria simultáneamente requiere alguna forma de administración. Además si varios están en condiciones de ser ejecutados, algún tipo de decisión se debe tomar para elegir uno: debe existir un planificador de cpu. También es necesario una planificación de dispositivos, manejo de abrazo mortal, control de concurrencia y protección.

1.6. TIEMPO COMPARTIDO

Actualmente un sistema batch está definido por características distintas a la de sus comienzos. Lo que definitivamente define a un procesamiento batch es la imposibilidad de una interacción entre el usuario y el trabajo, durante su ejecución. El tiempo transcurrido entre el envío del trabajo para su ejecución y la terminación del mismo, llamado a menudo tiempo de retorno, puede depender de la cantidad de cpu necesaria o por atrasos producidos antes que el SO comience a ejecutarlo. Es muy difícil depurar un programa (realizado en forma "estática") con tiempos de retorno grandes.

Un sistema interactivo provee comunicación entre el sistema y el usuario, recibiendo respuesta inmediata. Los primeros sistemas fueron básicamente interactivos: todo el equipo estaba a disposición del operador / programador. La técnica de tiempo compartido surge como necesidad de proveer de capacidad interactiva a un equipo a costos razonables.

Un SO de tiempo compartido usa un planificador de cpu y multiprogramación para brindar a cada usuario con una porción de uso del equipo. Cada usuario tiene un programa por separado en memoria. Cuando se ejecuta, el tiempo es muy corto.
Las operaciones de E/S son interactivas: la salida es la pantalla de vídeo y la entrada el teclado. Esta interactividad se produce a una velocidad fijada por el usuario: segundos. La entrada, por ejemplo será de 5 caracteres por segundo, que para el ser humano es muy veloz, pero para el sistema es una velocidad muy lenta. En lugar de que la cpu espere, la misma irá a ejecutar otro trabajo.

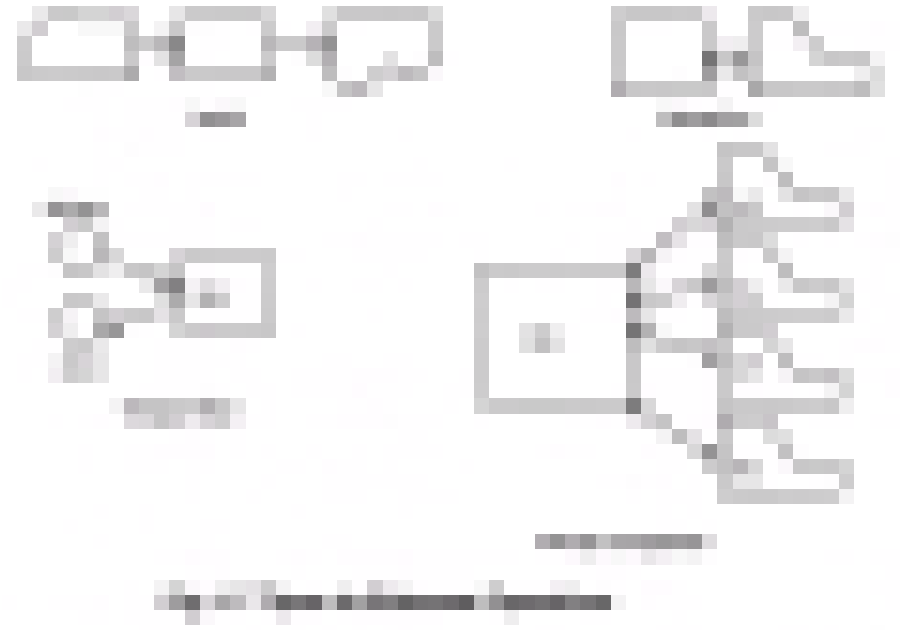
Un SO de tiempo compartido permite que varios usuarios compartan el computador. La idea de un sistema de tiempo compartido surgió al principio de la década del 60, pero por su complejidad y costo, recién en los 70 se hicieron populares. Desde entonces, muchos han intentado mezclar en un solo SO las capacidades de tiempo compartido y batch. Incluso, muchos SO que fueron diseñados para procesamiento batch, fueron modificados para crear un subsistema de tiempo compartido.

1.7. SISTEMAS DE TIEMPO REAL

Los SO de tiempo real son usados a menudo como dispositivos de control en aplicaciones dedicadas. Normalmente sensores proveen de datos a la computadora. El equipo debe analizar estos datos recibidos y posiblemente tomar alguna acción que modificará controles de salida. Sistemas computarizados para medicina, controles industriales y algunos sistemas de visualización son sistemas de tiempo real. El procesamiento debe realizarse dentro de ciertos márgenes definidos de tiempo o el sistema fallará.

Estos requerimientos de tiempo podemos contrastarlos con un sistema de tiempo

compartido donde la respuesta veloz es deseable pero no obligatoria, o con los sistemas batch donde no existe ningún tipo de tiempos máximos de respuesta preestablecidos.



1.8. PROTECCIÓN

En los sistemas primitivos el control del equipo lo tenía por completo el programador / operador. A medida que los SO se fueron desarrollando, las tareas de control se fueron transfiriendo al SO. En el monitor residente el SO comenzó a desarrollar funciones que previamente las hacía el operador, especialmente E/S.

Con técnicas más avanzadas, la posibilidad de que un programa (por error o en forma intencional) provoque errores en la ejecución de otros programas o del propio SO, aumenta. Por lo tanto deben existir protecciones que prevengan estas situaciones. Muchos errores de programación son detectados por el hardware. Estos errores son generalmente manejados por el monitor residente. Si el programa de un usuario falla en alguna forma, ya sea una instrucción ilegal o un direccionamiento indebido, el hardware provocará un trap al monitor residente. **Un trap es una interrupción generada internamente por el sistema, usualmente como resultado de un error de programa.** El trap que actúa como una interrupción, salvará el contenido del contador de programa de la instrucción ilegal y transferirá el control a una rutina de servicio del monitor residente. Ante una situación así, el monitor automáticamente hace un vuelco

de memoria y registros antes de atender próximos trabajos. Esta solución funciona mientras el error sea detectado por el hardware. Debemos asegurarnos, sin embargo, que cualquier tipo de error sea detectado: se debe proteger el SO y todos los programas de usuarios con sus datos del mal funcionamiento de cualquier programa.
Se necesita protección de todo recurso compartido. Por lo tanto, por lo menos necesitamos 3 tipos de protección: *de E/S, de memoria y de cpu.*

1.8.1. PROTECCIÓN DE E/S

Consideremos un programa que intente leer datos de entrada una vez que los mismos terminaron. ¿Como podemos parar el programa? Una solución es usar las subrutinas comunes de E/S, los drivers. Así, un driver de lectora de tarjetas puede ser modificado para que detecte el intento de leer una tarjeta de control. Cada tarjeta leída debe controlarse para detectar si es una de control (signo $ en columna 1). Si no lo es, el control debe retornar al programa de usuario. Si se intenta leer una tarjeta de control, el driver transferirá el control directamente al monitor residente. Este trata la situación como error, hace un vuelco de memoria y ejecuta el próximo trabajo. Esto funciona si el usuario usa el driver. Pero como esto es voluntario, puede suceder que el programador decida no usar drivers y escriba su propio programa.
Por ejemplo, supongamos que el programador esté preocupado por la velocidad y piensa que puede lograr un mejor tratamiento de buffers o simplemente piensa que sería interesante programar su propio driver.

Se debe evitar que cualquier usuario pueda leer tarjetas de control; solo el monitor podrá hacerlo. De esta forma, queremos que el equipo trate diferente al SO del resto de programas de usuarios. Se desean 2 modos de operación: ***modo usuario y modo monitor*** (también llamado modo supervisor o del sistema). Para ello agregamos un bit al hardware que indique el modo de trabajo: monitor (0) o usuario (1).
Luego definimos todas las instrucciones de E/S como privilegiadas: el hardware permitirá la ejecución de las instrucciones privilegiadas solo si estamos en modo monitor. Si se intenta ejecutar una instrucción privilegiada en modo usuario, el hardware no la ejecuta, la trata como una instrucción ilegal, emite un trap y transfiere el control al monitor residente. Cuando se produce un trap o una interrupción,

automáticamente el hardware cambia el modo usuario a monitor. Contrariamente, el monitor siempre cambia el estado de la máquina a modo usuario antes de pasar el control a un programa de usuario.

Este modo dual de trabajo nos asegura que solo por medio de drivers podamos leer tarjetas, (el driver es parte del monitor, por lo tanto son ejecutados en modo monitor). Un programa de usuario nunca puede, por lo tanto, leer una tarjeta de control como un dato. La operación del computador es ahora muy simple: arranque en modo monitor en el monitor residente; este lee la primer tarjeta de control y el programa deseado es cargado a memoria; cambio de modo monitor a usuario (instrucción privilegiada) y el monitor transfiere el control al programa recién cargado; cuando el programa termina el control retorna al monitor que continúa con la siguiente tarjeta de control. El programa de usuario no puede leer una tarjeta de control como si fuera un dato, pues el programa se ejecuta en modo usuario y no puede en forma directa realizar una operación de E/S.

El usuario debe pedir al SO que realice una operación de E/S a su servicio.

La mayoría de los sistemas tienen una instrucción especial conocida como ***llamada al monitor o llamada al sistema.*** Cuando se la ejecuta, el hardware la interpreta como una interrupción por software, el control pasa a través de un vector de interrupciones a una rutina de servicio del monitor residente. Se pasa a modo monitor. Este examina qué tipo de llamada ha ocurrido, la ejecuta y pasa el control al usuario.

Este modo dual de trabajo posibilita al SO tener completo control del sistema todo el tiempo. Adicionalmente, ya que el SO es llamado para cada operación de E/S, puede manejar el spooling, buffering y bloqueo para mejorar la performance. También puede definir dispositivos lógicos y asignarlos a dispositivos físicos, posibilitando independencia de dispositivos. Este modo dual de trabajo permite protección de E/S y además mejorar la performance del sistema.

1.8.2. PROTECCIÓN DE MEMORIA

Para que la protección de E/S sea completa debemos asegurarnos que nunca un

programa de usuario pueda ganar control del equipo en modo monitor.

Si estamos en modo usuario, pasaremos a modo monitor cuando ocurra una interrupción o trap, saltando a la dirección especificada en el vector de interrupción. Supongamos que un programa de usuario, como parte de su ejecución, almacene una nueva dirección en ese vector de interrupción. Esta nueva dirección podría re-escribirse sobre la dirección previa de la rutina de servicio con una dirección en la zona de programa de usuario. Entonces cuando ocurra una interrupción o trap, el hardware pasa a modo monitor y transfiere el control a través de esa dirección modificada al programa del usuario!!! Así, el usuario toma el control del equipo en modo monitor.

Por lo tanto debemos evitar que el vector de interrupciones pueda ser modificado por un programa de usuario. Por supuesto, también debemos proteger las rutinas de servicio de interrupciones. Aún si el usuario no toma control del equipo, la modificación no autorizada de estas rutinas, llevará seguramente a mal funcionamiento del sistema, del spooling y buffering.

Esta protección será provista por el hardware del equipo, agregando simplemente un registro valla (o barrera) que divida la memoria en 2 partes: del usuario y del monitor. La zona de memoria del monitor no podrá ser accedida o modificada por el usuario. Lo logramos comparando cada dirección generada en modo usuario con el registro valla. Cualquier intento de acceder a esta zona en modo usuario producirá un trap al monitor, que interpretará este intento como un error fatal.
El registro valla será cargado por el monitor con una instrucción privilegiada especial. Esto también posibilita que el SO cambie el valor del registro todas las veces que el monitor cambie de tamaño. Generalmente, el SO operando en modo monitor tiene acceso irrestricto a cualquier zona de memoria: monitor y usuario. Esto permite que el SO cargue programas de usuarios en la zona correspondiente, haga un vuelco de memoria cuando haya errores, acceda y modifique parámetros de llamadas al sistema, etc.

1.8.3. PROTECCIÓN DE CPU

Un cambio final en la arquitectura de un equipo es el agregado de un timer o

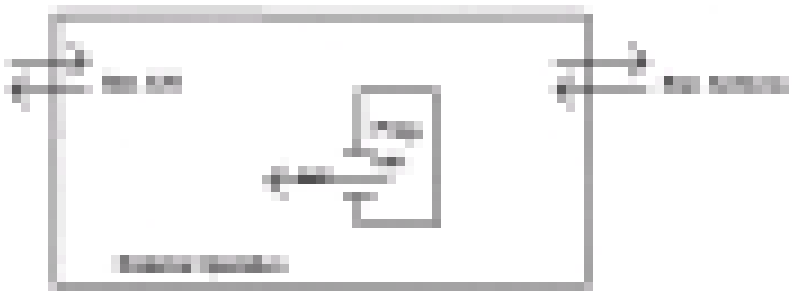
temporizador. Esto evita que un programa de usuario entre en un loop infinito que impida devolver el control al monitor. Puede establecerse, mediante el timer, un tiempo máximo fijo o variable, luego del cual se genere una interrupción. El timer se implementa mediante un reloj y un descontador. El SO carga el valor deseado y luego cada impulso del reloj decrementa el contador. Cuando llega a cero se genera una interrupción, luego el monitor decide si dar más tiempo al trabajo o tratar como un error. Por supuesto, las instrucciones de carga del contador son claramente privilegiadas.

1.8.4. ARQUITECTURA RESULTANTE

El deseo de reducir el tiempo de preparación y optimizar la utilización de cpu llevó al procesamiento batch de trabajos, buffering, spooling, multiprogramación y tiempo compartido. Esto produjo que los recursos sean compartidos por muchos trabajos y usuarios. El procesamiento en tiempo compartido provocó cambios en la arquitectura de los equipos, especialmente en lo concerniente a permitir que el control del equipo y el manejo de operaciones de E/S lo retenga el monitor residente. Este control es necesario si pretendemos un funcionamiento del sistema en forma continua, consistente y correcta.

Surge así el concepto de instrucciones privilegiadas. Existen muchas dentro de esta categoría, por ejemplo la instrucción **HALT** (parar). Una instrucción de usuario nunca puede llegar a parar todo un sistema.

Se debe mantener una protección de memoria del monitor, vector de interrupción y programas y datos de usuarios. El programa de usuario está así confinado por el SO.



El diseñador de sistema debe asegurarse que nunca, bajo ninguna circunstancia, un programa de usuario pueda interrumpir el funcionamiento del sistema (a menos que

exista un error en el SO, por supuesto).

1.9. DIFERENTES CLASES DE COMPUTADORAS

Casi todo nuestro análisis se ha centrado fundamentalmente en "mainframes", grandes equipos que se encuentran en centros de procesamiento de datos. Gran parte de la teoría de SO fue desarrollada para estos sistemas. Los SO para mainframes han ido desarrollándose en los últimos 30 años.

Las minicomputadoras aparecieron en la mitad de la década del 60 y son considerablemente más pequeñas en tamaño y de menor precio que los mainframes. A su vez, los microcomputadores aparecidos en la mitad de la década del 70 son aún más chicos y baratos. Los SO para estos equipos, de alguna forma se han beneficiado del desarrollo de SO para mainframes. Las minis y micros inmediatamente adoptaron la tecnología desarrollada para SO de grandes equipos, evitando desde su nacimiento los errores cometidos previamente. Es así que la mayoría de este equipamiento ha evitado el procesamiento batch, yendo directamente a SO interactivos y de tiempo compartido.
Por otra parte, al ser el hardware tan barato para estos equipos, muchas veces se ha evitado la necesidad de compartir recursos entre varios usuarios, con el fin de aumentar la utilización de cpu. Por ende, algunas decisiones de diseño apropiadas para mainframes no se tuvieron en cuenta para los nuevos equipos.

En general, sin embargo, examinando los SO de mainframes, minis y micros, vemos que funciones que en una época eran disponibles solo en los primeros, ahora han sido adoptadas por minis. Y aquellas propias de las minis, ahora se encuentran en las micros, concluyendo que los mismos conceptos y técnicas se aplican en todo el rango de equipos.
Un buen ejemplo de este "movimiento" puede verse considerando el SO Multics, que fue desarrollado entre 1965 y 1970 en el MIT como un utilitario. Este corría en un mainframe muy grande y complejo. Muchas de las ideas desarrolladas para el Multics fueron luego usadas por Laboratorios Bell (uno de los socios originales del desarrollo del Multics) para el diseño del SO Unix en 1969/70.
En esta década, el Unix fue la base para los sistemas compatibles (Xenix) para microcomputadores. Una de las teorías existentes es que con microprocesadores y

chips de memoria baratos, los SO serán en breve obsoletos. Creemos, que el abaratamiento del hardware hará que conceptos sofisticados de SO (tiempo compartido y memoria virtual) sean desarrollados aún para una mayor cantidad de equipos, lo que traerá como consecuencia, una mayor necesidad del entendimiento de los conceptos de SO.

1.10. SISTEMAS MULTIPROCESADORES

Existen intentos de crear sistemas multiprocesadores. Un sistema standard posee un solo procesador. Un sistema multiprocesador real tiene más de una cpu que comparten memoria y periféricos. Las ventajas obvias serían mayor poder computacional y mayor confiabilidad.

Para ello una de las siguientes arquitecturas se emplean.

La más común es asignar a cada procesador una tarea especifica. Un procesador maestro controla el sistema; los demás o esperan instrucciones del maestro o ya tienen tareas específicas. Este esquema define una relación amo/esclavo. En particular, pequeños procesadores localizados a cierta distancia del principal pueden usarse para controlar periféricos, relevando de estas tareas al principal.

La otra solución es una red de computadores. En una red, varios equipos pueden comunicarse intercambiando archivos e información entre ellos. Sin embargo, cada

equipo tiene su propio SO y opera independientemente. Las redes de equipos abre un nuevo campo de posibilidades en el procesamiento distribuido.