Universidad Nacional del Nordeste

Facultad de Ciencias Exactas y Naturales y Agrimensura

Carrera: Licenciatura en Sistemas de Información

Cátedra: Sistemas Operativos

Año: 2021

Grupo Numero 12

Integrantes del Grupo:

Apellido y Nombre	**LU**	**DNI**
Alcaraz Ojeda Leandro Mauricio	55.696	43.346.121
Cáceres Braun, Juan Gabriel,	54.208	41.015.736
Gonzales Pasi, Farid José,	49.950	38.314.658
Haberles, Lucas Francisco,	55.577	41.811.578
Moraes Moreyra, Leandro Nelson,	54.391	42.743.277
Parodi, Lucas Ivan,	52.419	40.876.821

Tema 16: Planificación Multiprocesador y en Tiempo Real

Índice

Tablas

Figuras

INTRODUCCIÓN

En un sistema de un único procesador, sólo puede ejecutarse un proceso cada vez; cualquier otro tendrá que esperar hasta que la CPU quede libre y pueda volver a planificarse. El objetivo de la multiprogramación es tener continuamente varios procesos en ejecución, con el fin de maximizar el uso de la CPU.

En la siguiente monografía se investigó y desarrollo sobre la "**Planificación de multiprocesadores y de tiempo real",** estos cuentan con un esquema más complejo que los monoprocesadores en cuanto a la planificación de tareas. Veremos la clasificación de estos sistemas y los aspectos relacionados acerca de su planificación. Teniendo en cuenta los aspectos de diseño donde se considera varios puntos interrelacionados. Además, hablaremos sobre la planificación de procesos y de hilos en un sistema multiprocesador.

Seguidamente explicaremos sobre la planificación de tiempo real y sus características.

Para finalizar, describiremos el funcionamiento de la planificación de multiprocesadores y en tiempo real de los siguientes sistemas: LINUX, UNIX SVR4 y WINDOWS.

PLANIFICACIÓN MULTIPROCESADOR

Un multiprocesador es un sistema de cómputo en el que dos o más CPUs comparten todo el acceso a una RAM común en su mayor parte, los sistemas operativos multiprocesadores tienen características únicas. Estas son la sincronización de procesos, la administración de recursos y la programación de tarea. Cuando un sistema computador tiene más de un procesador, el diseño de la función de planificación plantea varias cuestiones nuevas. Los sistemas multiprocesadores los podemos clasificar en:

Tabla 1.0. Clasificación Multiprocesadores.

Débilmente acoplado o multiprocesador distribuido, o clúster	Colección de sistemas relativamente autónomos, donde cada procesador tiene su propia memoria principal y canales de E/S.
Procesadores de funcionalidad especializada	Un ejemplo es un Procesador de E/S donde hay un procesador de propósito general maestro y procesadores especializados que son controlados por el procesador maestro y que le proporcionan servicios.
Procesamiento fuertemente acoplado	Conjunto de procesadores que comparten la memoria principal y están bajo el control integrado de un único sistema operativo.

Granularidad.

Para caracterizar y situar a los multiprocesadores, se considera la frecuencia de sincronización entre los procesos del sistema. Se pueden distinguir cinco categorías de paralelismo que difieren en el grado de granularidad.

- **Paralelismo Independiente.** No hay sincronización explícita entre procesos. Cada uno representa un trabajo independiente y separada. Ejemplo. sistemas de tiempo compartido. Cada usuario desarrolla su propio trabajo con una aplicación particular, como tratamiento de textos o una hoja de cálculos. El multiprocesador proporciona el mismo servicio que un monoprocesador multiprogramado. Dado que hay más de un procesador disponible, el tiempo medio de los usuarios será menor.

Tabla 1.1. Granularidad de sincronización y procesos

Tamaño del Grano	Descripción	Intervalo de Sincronización (Instrucciones)
Fino	**Paralelismo inherente es un único flujo de instrucciones**	**<20**
Medio	**Procesamiento paralelo o multitarea dentro de una única aplicación**	**20-200**
Grueso	**Multiprocesamiento de procesos concurrentes en un entorno multiprogramado**	**200-2000**
Muy Grueso	**Procesamiento distribuido entre nodos de una red para conformar un único entorno de computación**	**2000-1M**
Independiente	**Múltiples procesos no relacionados**	**(N/D)**

- **Paralelismo de grano grueso y muy grueso.** Hay sincronización entre procesos, pero a un nivel muy burdo. Se trata como un conjunto de procesos concurrentes, ejecutando en un monoprocesador multiprogramado y puede proporcionarse en un multiprocesador con poco o ningún cambio en el software de usuario

  En [WOOD89] se da un ejemplo de una aplicación que puede explotar la existencia de un multiprocesador. Los autores han desarrollado un programa que, dada una especificación de archivos que necesitan ser recompilados para reconstruir un fragmento de software, determina cuál de estas compilaciones pueden ser ejecutadas simultáneamente. El programa lanza un proceso por cada compilación paralela. Los autores informan que el aumento de velocidad de un multiprocesador realmente excede lo que cabría esperar si simplemente sumamos el número de procesadores en uso. En general, cualquier colección de procesos concurrentes que necesitan comunicarse o sincronizarse pueden beneficiarse del uso de una arquitectura multiprocesador. En el caso de que la intersección entre procesos sea muy poco frecuente, un sistema distribuido puede proporcionar un buen soporte. Sin embargo, si la intersección es algo más frecuente, un sistema, entonces la sobrecarga entre comunicaciones a través de la red puede mermar la potencial mejora de velocidad, En este caso, la organización multiprocesador proporciona un soporte más eficaz.

- **Paralelismo de grano medio:** El paralelismo potencial de la aplicación debe ser especificado explícitamente por el programador. Deberá haber un grado bastante alto de coordinación e interacción entre los hilos de la aplicación, dando lugar a un nivel de sincronización de grano medio. Mientras que los paralelismos independientes, grueso y muy grueso pueden proporcionarse en un monoprocesador multiprogramado o en un multiprocesador con poco o ningún impacto en la función de planificación, cuando se trata de planificación de hilos es necesario reexaminar la planificación. Dado que varios hilos de una aplicación interactúan muy frecuentemente, las decisiones de planificación concernientes a un hilo pueden afectar a las prestaciones de la aplicación completa.
- **Paralelismo de grano fino.** Representa un uso mucho más complejo del paralelismo del que se encuentra en el uso de hilos. Aunque se ha realizado mucho más trabajo sobre aplicaciones altamente paralelas.

Aspectos de diseño

La planificación involucra tres aspectos interrelacionados.

- La asignación de procesos a procesadores.
- El uso de la multiprogramación en cada procesador individual.
- La activación del proceso, propiamente dicha.

la propuesta elegida, dependerá, generalmente del grado de granularidad de la aplicación y del número de procesadores que se encuentran disponibles.

Asignación de procesos a procesadores

Si se asume que la arquitectura del multiprocesador es uniforme, donde ningún procesador tiene una ventaja física particular con respecto al acceso en memoria principal o a dispositivos de E/S, se mantiene una cola a corto plazo dedicada por cada procesador. Una **ventaja** es que puede haber menos sobrecarga en la función de planificación, dado que la asignación a un procesador se realiza una vez y para siempre.

Una **desventaja** de la asignación estática, es que un procesador puede estar ocioso, con su cola vacía, mientras otro procesador tiene trabajo acumulado. Para evitar esta situación, puede utilizarse una **cola común**. Todos los procesos van a una cola global y son planificados sobre cualquier procesador disponible. En una arquitectura de memoria compartida fuertemente acoplada, la información de contexto de todos los procesadores estará disponible para todos los procesadores, y por lo tanto el coste de

planificación de un proceso será independiente de la identidad del procesador sobre cual planifica. Otra opción es el **balance dinámico de carga** que consiste en el que los hilos se mueven de una cola de un procesador a la cola de otro procesador (Linux, utiliza este enfoque).

se necesita alguna manera de asignar procesos a otros procesadores. Usando dos enfoques: maestro/esclavo y camaradas. Con las arquitecturas maestro/esclavo, ciertas funciones clave del núcleo del sistema operativo ejecutan siempre en un procesador concreto. Los otros procesadores sólo pueden ejecutar programas del usuario. El **maestro** es responsable de la planificación de trabajos. Una vez que el proceso está activo, si el **esclavo** necesita servicio (por ejemplo, una llamada E/S), debe enviar una solicitud al maestro y esperar a que el servicio se realice. La resolución de conflictos se simplifica, porque un procesador tiene todo el control de la memoria y de los recursos de E/S.

Hay dos desventajas en este enfoque.

1. Un fallo del maestro hace que falle el sistema completo.
2. El maestro puede llegar a ser un cuello de botella para el rendimiento del sistema.

En la arquitectura **camaradas,** el núcleo puede ser ejecutado en cualquier procesador, y cada procesador se auto planifica desde la colección de procesos disponibles. Este enfoque complica el sistema operativo. Porque el sistema operativo debe asegurar que dos procesadores no escojan el mismo proceso y que los procesos no se pierdan por ninguna razón. Para ello se emplea técnicas para resolver y sincronizar la competencia en la demanda de recursos.

Uno de ellas es tener un subconjunto de los procesadores dedicados a procesar el núcleo en vez de uno solo. Y otro enfoque es gestionar las diferentes necesidades entre procesos del núcleo y otros procesos sobre la base de la prioridad y la historia de ejecución.

El uso de la multiprogramación en procesadores individuales

Cuando cada proceso se asocia estáticamente a un procesador para todo su tiempo de vida, surge una nueva pregunta ¿Debería ese procesador ser multiprogramado?, parece particularmente ineficiente vincular a un procesador con un único proceso, cuando este proceso puede quedar frecuentemente bloqueado esperando por E/S o por otras consideraciones de concurrencia/sincronización.

En los multiprocesadores tradicionales, que tratan con sincronizaciones de grano grueso o independiente, está claro que cada procesador individual debe ser capaz de cambiar entre varios procesos para conseguir una alta utilización y por lo tanto un mejor rendimiento. No obstante, para las aplicaciones de grano medio ejecutando en un multiprocesador con muchos procesadores, la situación está menos clara. Cuando hay muchos procesadores disponibles, conseguir que cada procesador esté ocupado tanto tiempo como sea posible deja de ser lo más importante. En cambio, la preocupación es proporcionar el mejor rendimiento, medio, de las aplicaciones. Una aplicación que consista en varios hilos, puede ejecutar con dificultad a menos que todos sus hilos estén dispuestos a ejecutar simultáneamente.

Planificación de procesos

En los sistemas multiprocesador, los procesos no se vinculan a los procesadores. En cambio, hay una única cola para todos los procesadores o, si se utiliza algún tipo de esquema basado en prioridades, hay múltiples colas basadas en prioridad, alimentando a un único colectivo de procesadores. En cualquier caso, el sistema puede verse como una arquitectura de colas multiservidor.

Considérese el caso de un sistema biprocesador en el que cada procesador tiene la mitad de velocidad de procesamiento que un procesador en un sistema monoprocesador [SAUE81] informa sobre un análisis de colas que compara de las planificaciones FCFS, turno circular y tiempo restante más breve. El estudio se ocupa del tiempo de servicio de los procesos, que mide la cantidad de tiempo de procesador que necesita un proceso, bien para el trabajo completa o cada vez que el proceso está listo comparado con la sobrecarga de cambios de contextos y pequeño comparado con el tiempo medio de servicio. Los resultados dependen de la variabilidad que se observa en los tiempos de servicio.

Planificación De Hilos

Como hemos visto, con los hilos, el concepto de ejecución se separa del resto de la definición de un proceso. Una aplicación puede ser implementada como un conjunto de hilos, que cooperan y ejecutan de forma concurrente en el mismo espacio de direcciones.

En un monoprocesador, los hilos pueden usarse como una ayuda a la estructuración de programas y para solapar E/S con el procesamiento. Dada la mínima penalización por realizar un cambio de hilo comparado con un cambio de proceso, los beneficios se obtienen con poco coste.

Sin embargo, el poder completo de los hilos se vuelve evidente en un sistema multiprocesador. En este entorno, los hilos pueden explotar paralelismo real dentro de una aplicación. Si los hilos de una aplicación están ejecutando simultáneamente en procesadores separados, es posible una mejora drástica de sus prestaciones.

Sin embargo, puede demostrarse que para aplicaciones que necesitan una interacción significativa entre hilos (paralelismo de grano medio), pequeñas diferencias en la gestión y planificación de hilos pueden dar lugar a un impacto significativo en las prestaciones [ANDE89].

Enfoques Generales:

- **Compartición de carga:** Los procesos no se asignan a un procesador particular. Se mantiene una cola global de hilos listos, y cada procesador, cuando está ocioso, selecciona un hilo de la cola. El término compartición de carga se utiliza para distinguir esta estrategia de los esquemas de balanceo de carga en los que los trabajos se asignan de una manera más permanente.
- **Planificación en pandilla:** Un conjunto de hilos relacionados que se planifica para ejecutar sobre un conjunto de procesadores al mismo tiempo, en una relación uno-a-uno.
- **Asignación de procesador dedicado:** Esto es lo opuesto al enfoque de compartición de carga y proporciona una planificación implícita definida por la asignación de hilos a procesadores. Cada proceso ocupa un número de procesadores igual al número de hilos en el programa, durante toda la ejecución del programa.

Cuando el programa termina, los procesadores regresan al parque general para la posible asignación a otro programa.

- **Planificación dinámica:** El número de hilos de un proceso puede cambiar durante el curso de su ejecución.

Compartición De Carga:

La compartición de carga es posiblemente el enfoque más simple y que se surge más directamente de un entorno monoprocesador.

Algunas ventajas que podemos tener:

- La carga se distribuye uniformemente entre los procesadores, asegurando que un procesador no queda ocioso mientras haya trabajo pendiente.
- No se precisa un planificador centralizado; cuando un procesador queda disponible, la rutina de planificación del sistema operativo se ejecuta en dicho procesador para seleccionar el siguiente hilo.
- La cola global puede organizarse y ser accesible usando cualquiera de los esquemas expuestos en el Capítulo 9, incluyendo esquemas basados en prioridad y esquemas que consideran la historia de ejecución o anticipan demandas de procesamiento.
- **[LEUT90] Analiza tres versiones diferentes de compartición de carga:**

Primero en llegar, primero en ser servido (FCFS). Cuando llega un trabajo, cada uno de sus hilos se disponen consecutivamente al final de la cola compartida. Cuando un procesador pasa a estar ocioso, coge el siguiente hilo listo, que ejecuta hasta que se completa o se bloquea.

Menor número de hilos primero. La cola compartida de listos se organiza como una cola de prioridad, con la mayor prioridad para los hilos de los trabajos con el menor número de hilos no planificados. Los trabajos con igual prioridad se ordenan de acuerdo con qué trabajo llega primero. Al igual que con FCFS, el hilo planificado ejecuta hasta que se completa o se bloquea.

Menor número de hilos primero con expulsión. Se les da mayor prioridad a los trabajos con el menor número de hilos no planificados. Si llega un trabajo con menor

número de hilos que un trabajo en ejecución se expulsarán los hilos pertenecientes al trabajo planificado.

Algunas desventajas:

- La cola central ocupa una región de memoria a la que debe accederse de manera que se cumpla la exclusión mutua. De manera que puede convertirse en un cuello de botella si muchos procesadores buscan trabajo al mismo tiempo. Cuando hay sólo un pequeño número de procesadores, es difícil que esto se convierta en un problema apreciable. Sin embargo, cuando el multiprocesador consiste en docenas o quizás cientos de procesadores, la posibilidad de que esto sea un cuello de botella es real.
- Es poco probable que los hilos expulsados retomen su ejecución en el mismo procesador. Si cada procesador está equipado con una cache local, ésta se volverá menos eficaz.
- Si todos los hilos se tratan como un conjunto común de hilos, es poco probable que todos los hilos de un programa ganen acceso a procesadores a la vez. Si se necesita un alto grado de coordinación entre los hilos de un programa, los cambios de proceso necesarios pueden comprometer seriamente el rendimiento.

A pesar de las grandes desventajas, este es uno de los esquemas más utilizados en los multiprocesadores actuales

Planificación en pandilla:

El concepto de planificar un conjunto de procesos simultáneamente sobre un conjunto de procesadores es anterior al uso de hilos. [JONE80] se refiere al concepto de planificación en grupo y cita los siguientes beneficios:

- ➔ Si se ejecutan en paralelo procesos estrechamente relacionados, puede reducirse el bloqueo por sincronización, pueden necesitarse menos cambios de proceso y las prestaciones aumentarán.
- ➔ La sobrecarga de planificación puede reducirse dado que una decisión única afecta a varios procesadores y procesos a la vez.

El término planificación en pandilla se ha aplicado a la planificación simultánea de los hilos que componen un proceso individual [FEIT90b]. La planificación en pandilla es para aplicaciones paralelas de grano medio o fino cuyo rendimiento se ve degradado de

forma importante cuando cualquier parte de la aplicación no está ejecutando mientras otras partes están listas para ejecutar. También es beneficiosa para cualquier aplicación paralela incluso para aquellas de rendimiento no tan sensible. La necesidad de planificación en pandilla está ampliamente reconocida, y existen implementaciones en variedad de sistemas operativos multiprocesador.

La planificación en pandilla mejora las prestaciones de una aplicación individual son porque se minimizan los cambios de proceso. Suponga que un hilo de un proceso está ejecutando y alcanza un punto en el que debe sincronizarse con otro hilo del mismo proceso. Si este otro hilo no está ejecutando, pero está listo para ejecutar, el primer hilo quedará colgado hasta que suceda un cambio de proceso en otro procesador que tome el hilo necesario. En una aplicación con coordinación estrecha entre sus hilos, estos cambios reducirán dramáticamente el rendimiento. La planificación simultánea de hilos cooperantes puede también reducir el tiempo de ubicación de recursos. Por ejemplo, múltiples hilos planificados en pandilla pueden acceder a un fichero sin la sobrecarga adicional de bloquearse durante una operación de posicionamiento y lectura/escritura.

Asignación de procesador dedicado:

Una forma extrema de la planificación en pandilla, sugerida en [TUCK89], es dedicar un grupo de procesadores a una aplicación durante toda la duración de la aplicación. Esto es, cuando se planifica una aplicación dada, cada uno de sus hilos se asigna a un procesador que permanece dedicado al hilo hasta que la aplicación concluye.

Este enfoque puede parecer un desperdicio extremo del tiempo de procesador. Si un hilo de una aplicación se bloquea esperando por E/S o por sincronización con otro hilo, entonces el procesador de ese hilo se queda ocioso: no hay multiprogramación de los procesadores. Pueden realizarse dos observaciones en defensa de esta estrategia:

- ➔ En un sistema altamente paralelo, con decenas o cientos de procesadores, cada uno de los cuales representa una pequeña fracción del coste del sistema, la utilización del procesador deja de ser tan importante como medida de la eficacia o rendimiento.
- ➔ Evitar totalmente el cambio de proceso durante la vida de un programa debe llevar a una sustancial mejora de velocidad de ese programa.

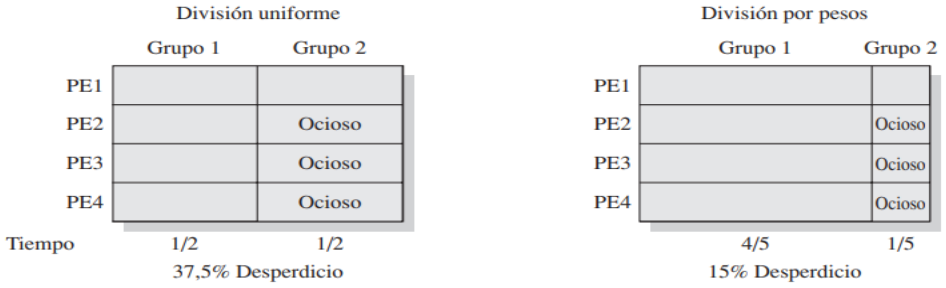

Figura 1.0. Planificación de grupos con cuatro y un hilo [FEIT90b]

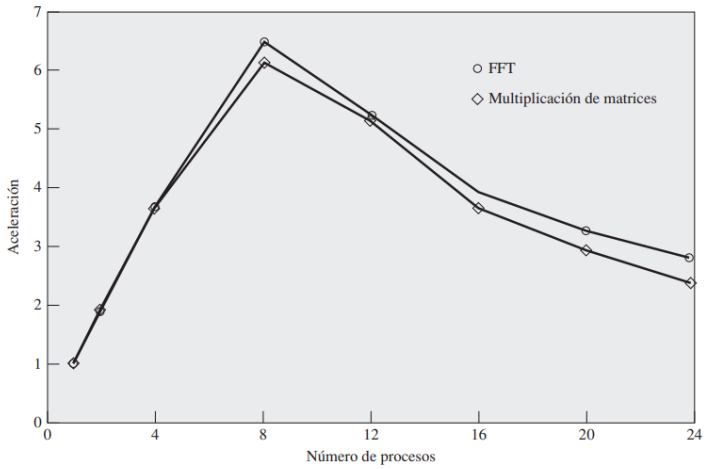

Figura 1.1. Aceleración de la aplicación en función del número de procesos [TUCK89].

Los autores concluyen que una estrategia eficaz es limitar el número de hilos activos al número de procesadores en el sistema. Si la mayoría de las aplicaciones son de un hilo único o pueden ser estructuradas como una cola de tareas, se conseguirá un uso eficaz y razonablemente eficiente de los recursos procesador.

Tanto la asignación de procesador dedicado como la planificación en pandilla atacan el problema de la planificación tratando el aspecto de la ubicación del procesador. cuántos procesadores asignar a un programa en un momento dado, que es análoga a cuántos marcos de página asignar a un proceso en dicho momento. [GEHR87] define el término conjunto residente de actividad, análogo al de conjunto residente de la memoria virtual, Planificación multiprocesador y de tiempo real 461 0 4 0 1 2 3 4 5 6 7 8 12 Número de procesos Aceleración FFT Multiplicación de matrices 16 20 24 Figura 1.1. Aceleración de la aplicación en función del número de procesos [TUCK89]. como el mínimo número de actividades (hilos) que deben ser planificados simultáneamente sobre procesadores para que la aplicación tenga un progreso aceptable.

Al igual que con los esquemas de la gestión de la memoria, no planificar todos los elementos de un conjunto residente de actividad puede provocar trasiego de procesador. Esto sucede cuando la planificación de hilos cuyos servicios se necesitan, provoca la expulsión de otros hilos cuyos servicios serán necesitados pronto. De igual modo, la fragmentación de procesador se refiere a una situación en la cual quedan

algunos procesadores sobrantes mientras otros están ubicados, y los procesadores sobrantes son o bien insuficientes en número, o bien no están organizados adecuadamente para dar soporte a los requisitos de las aplicaciones pendientes. La planificación en pandilla y la ubicación de procesador dedicado tienen como objetivo evitar estos problemas.

Planificación dinámica Para algunas aplicaciones, es posible proporcionar, en el lenguaje o en el sistema, herramientas que permitan que el número de hilos del proceso pueda ser alterado dinámicamente. Esto permitiría que el sistema operativo ajustase la carga para mejorar la utilización. [ZAHO90] propone un enfoque en el cual tanto el sistema operativo como la aplicación están involucrados en tomar las decisiones de planificación. El sistema operativo es el responsable de particionar los procesadores entre los trabajos. Cada trabajo utiliza los procesadores actualmente en su partición para ejecutar algún subconjunto de sus tareas ejecutables proyectando estas tareas sobre hilos. La decisión apropiada acerca del subconjunto a ejecutar, así como qué hilos suspender cuando el proceso sea expulsado, se les deja a las aplicaciones individuales. Este enfoque puede no ser adecuado para todas las aplicaciones. Sin embargo, algunas aplicaciones pueden ser consideradas como un único hilo mientras otras pueden ser programadas para sacar ventaja de esta particular característica del sistema operativo. En este enfoque, la responsabilidad de planificación del sistema operativo está limitada fundamentalmente a la ubicación del procesador y se realiza de acuerdo a la siguiente política. Cuando un trabajo solicita uno o más procesadores Si hay procesadores ociosos, utilizarlos para satisfacer la solicitud.

1) En otro caso, si el trabajo que realiza la solicitud acaba de llegar, ubicarlo en un único procesador quitándoselo a cualquier trabajo que actualmente tenga más de un procesador.

2) Si no puede satisfacerse cualquier parte de la solicitud, mantenerla pendiente hasta que un procesador pase a estar disponible, o el trabajo rescinda la solicitud (por ejemplo, si dejan de ser necesarios los procesadores extra). Cuando se libere uno o más procesadores (incluyendo la terminación de un trabajo)

3) Examinar la cola actual de solicitudes de procesador no satisfechas. Asignar un único procesador a cada trabajo en la lista que no tenga actualmente procesadores (por ejemplo, a todos los recién llegados en espera). Luego volver a examinar la lista, volviendo a asignar el resto de los procesadores siguiendo la estrategia FCFS.

PLANIFICACIÓN DE TIEMPO REAL

Un sistema operativo es responsable de administrar los recursos de hardware de una computadora y de alojar las aplicaciones que se ejecutan en ellas. Un Sistema Operativo de Tiempo Real realiza estas tareas, pero también está especialmente diseñado para ejecutar aplicaciones con una sincronización muy precisa y con un alto grado de confiabilidad.

Para ser considerado “de tiempo real”, un sistema operativo debe tener un tiempo máximo conocido para cada una de las operaciones que realiza (o al menos ser capaz de garantizar ese máximo la mayor parte del tiempo). Algunas de estas operaciones incluyes llamadas al sistema operativo y manejo de interrupciones.

Dicho sistema es aquel que debe procesar información y producir una respuesta dentro de un tiempo específico; de lo contrario, corre el riesgo de sufrir graves consecuencias, incluida la falla. Es decir, en un sistema con una restricción de tiempo real no es bueno tener la acción correcta o la respuesta correcta después de un tiempo límite determinado: si es después del tiempo límite, es inútil. El momento en el que se produce la salida es significativo.

En un sistema en tiempo real, la corrección del comportamiento del sistema depende no solo de los resultados lógicos de los cálculos, sino también del instante físico en el que se producen estos resultados.

Los sistemas de tiempo real se están generalizando. Los ejemplos típicos de sistemas en tiempo real incluyen sistemas de control de tráfico aéreo, sistemas multimedia en red y sistemas de control de comandos, etc.

Los sistemas en tiempo real se clasifican según factores externos al sistema informático y factores internos del sistema informático. Se hace especial hincapié en los sistemas en tiempo real **duros** y **blandos**. Un tiempo límite incumplido en los sistemas en tiempo real duros es catastrófica y en los sistemas flexibles en tiempo real puede provocar una pérdida significativa. Por lo tanto, la predictibilidad del comportamiento del sistema es la preocupación más importante en estos sistemas. La previsibilidad a menudo se logra mediante la programación estática o dinámica de tareas en tiempo real para cumplir con sus plazos. La programación estática toma decisiones de programación en tiempo de compilación. La programación dinámica utiliza una prueba de capacidad de programación para determinar si un conjunto de tareas puede cumplir con sus plazos.

Tiempo Real Duro y Tiempo Real Suave

Un sistema operativo que puede garantizar absolutamente un tiempo máximo para las operaciones que realiza se denomina de tiempo real duro. Por el contrario, un sistema operativo que normalmente puede realizar operaciones en un tiempo determinado se denomina de tiempo real suave.

Los sistemas en tiempo real duros están diseñados para garantizar **absolutamente** que una tarea se ejecutará dentro de un cierto período de tiempo en el peor de los casos. Por lo tanto, para proyectos que involucran seguridad o sistemas que podrían resultar en una gran inversión en caso de falla, el tiempo real duro es a menudo un requisito. Por otro lado, los sistemas blandos en tiempo real están diseñados para satisfacer sus requisitos de sincronización **la mayor parte del tiempo**, pero **sin una certeza absoluta**. Esto puede ser aceptable para operaciones como el procesamiento de video, donde una trama de datos perdida no es buena, pero puede que no sea necesariamente un problema crítico.

En conclusión, los sistemas en tiempo real duro garantizan (cuando se programan correctamente) que un tiempo límite se cumplirá consistentemente, mientras que los sistemas en tiempo real suave pueden exceder periódicamente al tiempo límite.

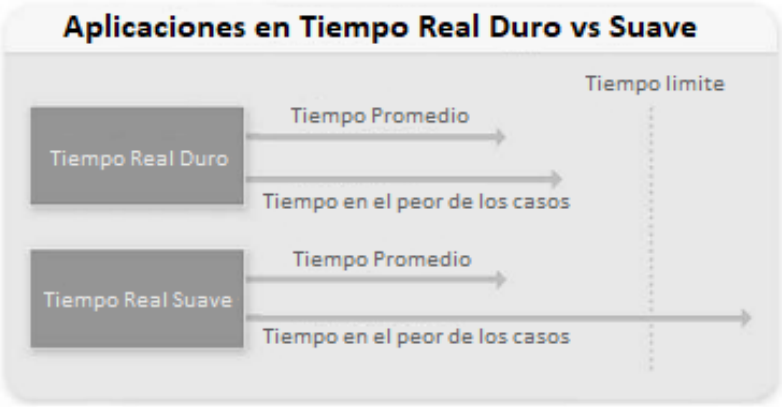

Figura.2.0. Aplicaciones en Tiempo Real Duro vs Suave.

Características de los Sistemas Operativos en Tiempo Real

Los sistemas operativos de tiempo real pueden ser caracterizados por tener requisitos únicos en cinco áreas:

Determinismo

Un sistema operativo es determinista si realiza operaciones en instantes de tiempos fijos predeterminados o dentro de intervalos de tiempo predeterminados. Cuando varios procesos compiten por los recursos y por el tiempo del procesador, depende, en primer lugar, de la velocidad con la que pueda responder a las interrupciones y, en segundo lugar, de si el sistema posee suficiente capacidad para gestionar todas las peticiones en el tiempo requerido.

Reactividad

Cuánto tiempo tarda el sistema operativo, después del reconocimiento, en servir la interrupción.

La misma incluye los siguientes aspectos:

1. Cantidad de tiempo para manejar inicialmente la interrupción y comenzar a ejecutar la rutina de servicio de la interrupción (RSI). Si la ejecución de la RSI necesita un cambio de proceso, entonces el retardo será mayor que si la RSI puede ser ejecutada dentro del contexto del proceso actual.

2. La cantidad de tiempo necesario para realizar la RSI. Esto depende de la plataforma hardware.

3. El efecto del anidamiento de interrupciones. Si una RSI puede ser interrumpida por la llegada de otra interrupción, entonces el servicio se retrasará.

El **determinismo** y la **reactividad** juntos conforman el tiempo de respuesta a eventos externos. Los requisitos de tiempo de respuesta son críticos para los sistemas de tiempo real.

Control de Usuario

El control del usuario es generalmente mucho mayor en un sistema operativo en tiempo real que en un sistema operativo ordinario. En un sistema operativo típico que no sea en tiempo real, el usuario no tiene control sobre la función de planificación del sistema operativo. En un sistema en tiempo real resulta esencial permitir al usuario un control preciso sobre la prioridad de las tareas. Un sistema en tiempo real también permitirá al

usuario especificar características como el uso de paginación en los procesos, que procesos deben residir en memoria principal, que algoritmos de transferencias a disco deben utilizarse, etc.

Fiabilidad

La fiabilidad es generalmente mucho más importante para los sistemas de tiempo real que para los que no son. Un fallo temporal en un sistema no de tiempo real puede resolverse simplemente re arrancando el sistema. El fallo de un procesador en un sistema de varios procesadores no de tiempo real puede dar lugar a un nivel de servicio degradado hasta que el procesador que falla sea reparado o sustituido. Sin embargo, un sistema de tiempo real ha de responder y controlar eventos en tiempo real. La pérdida o degradación de sus prestaciones puede tener consecuencias catastróficas: pérdidas económicas, daño en equipos importantes e incluso pérdida de vidas.

Operación de fallo suave

La operación de fallo suave es una característica que se refiere a la habilidad del sistema de fallar de tal manera que se preserve tanta capacidad y datos como sea posible.

Planificador

El corazón de un sistema en tiempo real es el planificador de tareas a corto plazo. Lo que es importante es que todas las tareas de tiempo real duro se completen (o comiencen) en su plazo y que tantas tareas de tiempo real suave como sea posible también se completen (o comiencen) en su plazo.

Las aplicaciones de tiempo real normalmente necesitan tiempos de respuesta deterministas en un rango de varios milisegundos, las aplicaciones al límite, como los simuladores de aviones militares, por Ejemplo, presentan a menudo restricciones en un rango de diez a cien microsegundos.

En las siguientes Figuras se muestran las distintas posibilidades.

Planificación con tablas estáticas: realizan un análisis estático de las planificaciones de expedición posibles. El resultado del análisis es una planificación que determina un tiempo de ejecución, cuando debe comenzar la ejecución de una tarea.

En un planificador preferente que emplea una planificación simple por turno rotatorio, una tarea de tiempo real se añadiría a la cola de Listas para esperar su siguiente fracción

de tiempo, como se ve en la figura 2.1. En este caso, el tiempo de planificación normalmente sería inaceptable para aplicaciones de tiempo real.

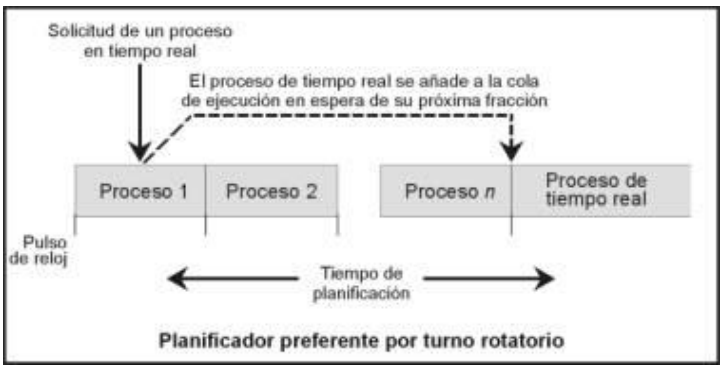

Figura 2.1. Planificador preferente por turno rotatorio.

Planificación preferente con prioridades estáticas: también se realiza un análisis estático, pero no se realiza ninguna planificación. En cambio, se usa dicho análisis para asignar prioridades a tareas de forma que se pueda emplear un planificador convencional preferente con prioridades

en un planificador no preferente, se puede emplear un mecanismo de planificación por prioridades, dando a las tareas de tiempo real la prioridad más alta. En este caso, una tarea de tiempo real que estuviera lista sería planificada tan pronto como el proceso actual se bloquee o termine su ejecución (Figura 2.2). Esto podría acarrear un retraso de varios segundos si una tarea lenta de baja prioridad se estaba ejecutando en un momento crítico. Tampoco, este método es aceptable.

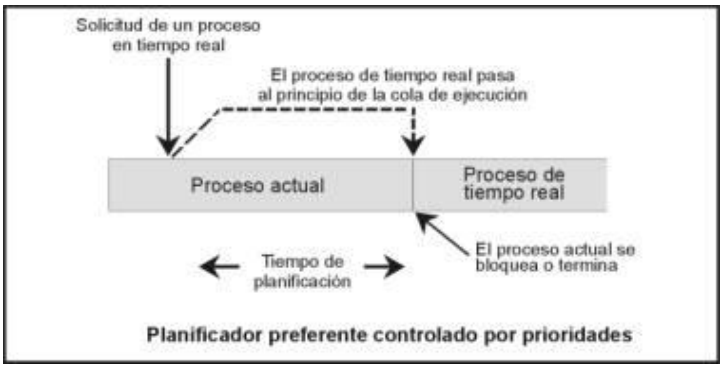

Figura 2.2. Planificador preferente controlado por prioridades

Planificación dinámica basada en un plan: se determina la viabilidad en tiempo de ejecución (dinámicamente) en vez de antes de empezar la ejecución (estáticamente). Se acepta una nueva tarea para ejecutar solo si es factible cumplir sus restricciones de

tiempo. Uno de los resultados del análisis de viabilidad es un plan o proyecto empleado para decidir cuándo se expide cada tarea.

Una idea más prometedora consiste en combinar las prioridades con interrupciones del reloj. Los instantes de apropiación se originan a intervalos regulares. Cuando llegue un instante de apropiación, se expulsará a la tarea que se esté ejecutando en ese momento si es que una tarea de prioridad superior está esperando, incluyendo la expulsión de tareas que son parte del núcleo del sistema operativo. Estos retardos pueden ser del orden de varios milisegundos Figura (2.3)

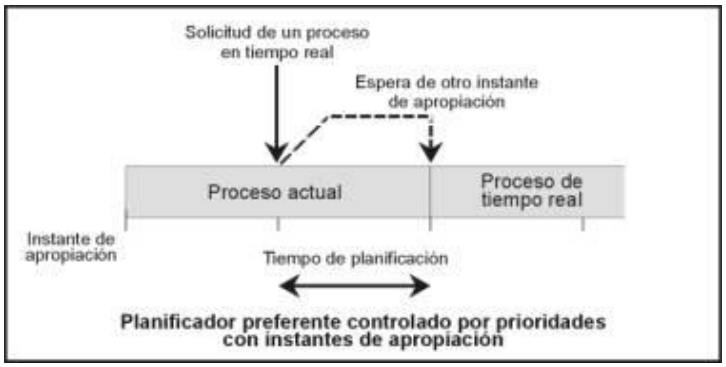

Figura 2.3. Planificador preferente controlado por prioridades con instantes de apropiación.

Planificación dinámica del mejor resultado: no se realiza ningún análisis de viabilidad. El sistema intenta cumplir todos los plazos y abandona cualquier proceso ya iniciado y cuyo plazo no se haya cumplido.

En estos últimos casos, se emplea la denominada apropiación inmediata. En la apropiación inmediata, el sistema operativo responde a las interrupciones casi inmediatamente, a menos que el sistema se encuentre ejecutando una sección de código crítico. Los retrasos de planificación de una tarea de tiempo real se pueden ver reducidos a 100 pso menos. Figura 2.4.

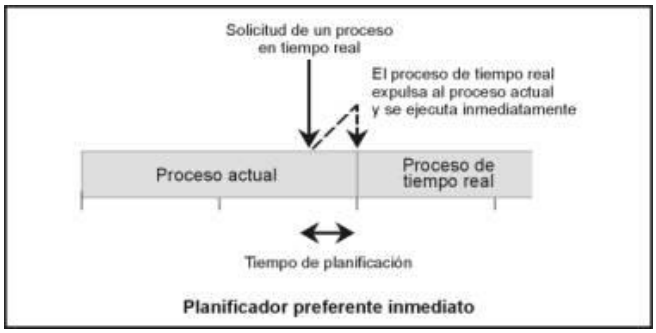

Figura 2.4. Planificador preferente inmediato.

PLANIFICACIÓN EN LINUX

En la versión de Linux 2.4 y también en anteriores, tiene la capacidad para la planificación de tiempo real, así como un planificador para procesos no de tiempo real que hace uso del algoritmo de planificación tradicional de UNIX. También en la versión de Linux 2.6 incluye esencialmente la misma capacidad de planificación de tiempo real que las ediciones previas y un planificador sustancialmente modificado para procesos no de tiempo real.

Existen tres clases de planificación en Linux:

- **SCHED_FIFO** hilos de tiempo real planificados FIFO
- **SCHED_RR** hilos de tiempo real planificados con turno circular.
- **SCHED_OTHER** otros hilos no de tiempo real

Dentro de cada clase, pueden utilizarse múltiples prioridades, siendo las prioridades de las clases de tiempo real mayores que las prioridades para la clase SCHED_OTHER. Los valores por omisión son los siguientes: las prioridades de las clases de tiempo real van del 0 al 99 inclusive, y para la clase SCHED_OTHER van del 100 al 139. Un número menor significa mayor prioridad.

Para los hilos FIFO, se aplican las reglas:

- El sistema no interrumpirá un hilo **FIFO** en ejecución excepto en los siguientes casos:
    1. Otro hilo FIFO de mayor prioridad pasa a estado listo.
    2. El hilo FIFO en ejecución pasa a estar bloqueado en espera por algún evento, como E/S.
    3. El hilo FIFO en ejecución cede voluntariamente el procesador realizando una llamada a la primitiva sched_yield
- Cuando un hilo FIFO en ejecución es interrumpido, se le sitúa en una cola asociada con su prioridad.
- Cuando un hilo FIFO pasa a estar listo y ese hilo es de mayor prioridad que el hilo actualmente en ejecución, entonces el hilo actualmente en ejecución es expulsado y el hilo FIFO listo de mayor prioridad es ejecutado. Si hay más de un hilo que tiene esa mayor prioridad, se escogerá el hilo que lleve más tiempo esperando.

La política SCHED_RR es similar a la política SCHED_FIFO, excepto por el añadido de una rodaja de tiempo asociada con cada hilo. Cuando un hilo SCHED_RR ha estado ejecutando durante toda su rodaja de tiempo, se le suspende y se selecciona para su ejecución un hilo de tiempo real de igual o mayor prioridad.

La Figura 3.0 es un ejemplo que ilustra la diferencia entre las planificaciones FIFO y RR. Asúmase un proceso con cuatro hilos con tres prioridades relativas asignadas como se muestra en la Figura 3. 0.a. Asúmase que todos los hilos en espera están listos para ejecutar cuando el hilo actual pasa a esperar o termina y que ningún hilo de mayor prioridad es despertado mientras el hilo está ejecutando.

La Figura 3.0.b muestra una secuencia en la que todos los hilos pertenecen a la clase SCHED_FIFO. El hilo D ejecuta hasta que espera o termina. Luego, aunque los hilos B y C tienen la misma prioridad, arranca el hilo B porque lleva esperando más tiempo que el hilo C. El hilo B ejecuta hasta que espera o termina; luego el hilo C ejecuta hasta que espera o termina. Finalmente, ejecuta el hilo A.

La Figura 3.0.c muestra un ejemplo de secuencia si todos los hilos perteneciesen a la clase SCHED_RR. El hilo D ejecuta hasta que espera o termina. Luego, los hilos B y C se entrelazan en el tiempo, porque ambos tienen la misma prioridad. Finalmente, ejecuta el hilo A. La última clase de planificación es SCHED_OTHER. Un hilo en esta clase sólo puede ejecutar si no hay hilos de tiempo real listos para ejecutar.

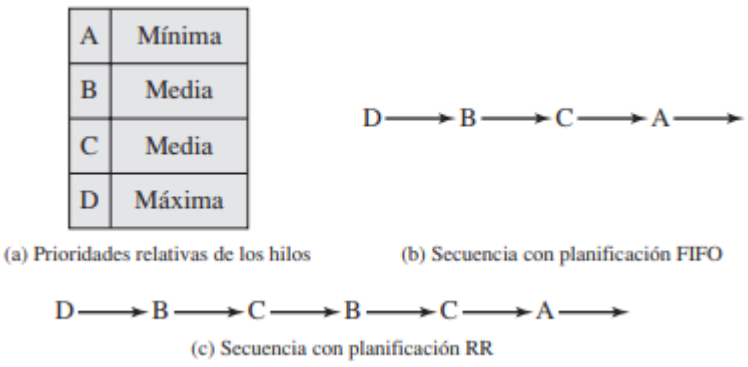

Figura 3.0. Ejemplo de la planificación de tiempo real de Linux

Planificación No De Tiempo Real

El planificador de Linux 2.4 para la clase SCHED_OTHER no escalaba bien con un número creciente de procesadores y un número creciente de procesos. Para atacar este problema, Linux 2.6 usa un planificador completamente nuevo conocido como el planificador O (1)4. El planificador ha sido diseñado para que el tiempo de seleccionar el proceso adecuado y asignarlo a un procesador sea constante, sin importar la carga del sistema y el número de procesadores. El núcleo mantiene dos estructuras de datos para la planificación por cada procesador del sistema, con la siguiente forma:

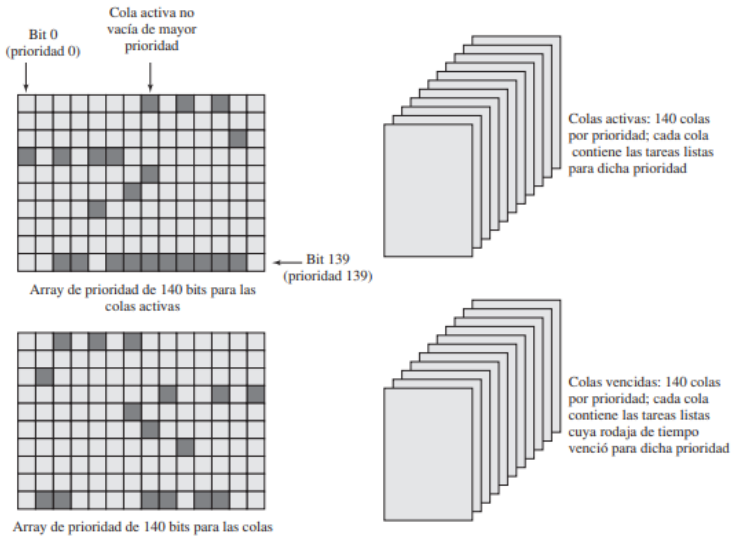

Figura.3.1. Estructuras de datos planificación en Linux por cada procesador.

La planificación es simple y eficiente. En un procesador dado, el planificador toma la cola no vacía de mayor prioridad. Si hay múltiples tareas en esa cola, las tareas se planifican de manera turno circular.

Linux también incluye un mecanismo para mover tareas de la lista de colas de un procesador a la de otro. Periódicamente, el planificador comprueba si hay un desbalanceo sustancial entre el número de tareas asignadas a cada procesador. Para balancear la carga, el planificador puede transferir algunas tareas. Las tareas activas de mayor prioridad se seleccionan para ser transferidas, dado que es más importante distribuir uniformemente las tareas de mayor prioridad.

Cálculo de prioridades y rodajas de tiempo A cada tarea no de tiempo real se le asigna una prioridad inicial en el rango de 100 a 139, por un valor por comisión de 120. Esta es

la prioridad estática de la tarea que es especificada por el usuario. A medida que la tarea ejecuta, se calcula una prioridad dinámica como función de la prioridad estática de la tarea y del comportamiento de su ejecución.

El planificador Linux está diseñado para favorecer las tareas limitadas por la E/S sobre las limitadas por el procesador. Esta preferencia tiende a proporcionar una buena respuesta interactiva. La técnica utilizada en el Linux para determinar la prioridad dinámica es guardar información sobre cuánto tiempo duermen los procesos (esperando por un evento) versus cuánto tiempo ejecutan. En esencia, a una tarea que pierde la mayor parte del tiempo durmiendo se le da la mayor prioridad. Las rodajas de tiempo se asignan en el rango de los 10 ms a los 200 ms. En general a las tareas de mayor prioridad se le asignan rodajas de tiempo más largas. Relación con las tareas de tiempo real Las tareas de tiempo-real se manipulan de manera distinta que las tareas no de tiempo real en las colas de prioridad. Se aplican las siguientes consideraciones:

- Todas las tareas de tiempo real tienen solamente prioridad estática; no se realizaron cálculos de prioridad dinámica.
- Las tareas SCHED_FIFO no tienen asignadas rodajas de tiempo. Estas tareas se planifican siguiendo la disciplina FIFO. Si una de estas tareas se bloquea, cuando se desbloquee volverá a la misma cola de prioridad en la lista de colas activas.
- Aunque las tareas SCHED_RR tienen asignadas rodajas de tiempo, tampoco se le traslada nunca a la lista de colas vencidas. Cuando una de estas tareas consume toda su rodaja de tiempo, se le devuelve a su cola de prioridad con el mismo valor de rodaja de tiempo. Los valores de rodaja de tiempo nunca se cambian.

El efecto de estas reglas es que el cambio entre la lista de colas activas y la lista de colas vencidas sólo sucede cuando no hay tareas de tiempo real listas esperando para ejecutar.

PLANIFICACION EN UNIX SVR4

El algoritmo de planificación utilizado en UNIX SVR4 es una revisión completa del algoritmo de planificación utilizado en los sistemas UNIX anteriores. El nuevo algoritmo está diseñado para dar la mayor preferencia a los procesos de tiempo real, la siguiente mayor preferencia a los procesos en modo núcleo, y la menor preferencia a otros procesos en modo-usuario, conocidos como procesos de tiempo compartido. Las dos principales modificaciones implementadas en SVR4 son las siguientes:

- El añadido de un planificador de prioridad estática expulsivo y la introducción de un conjunto de 160 niveles de prioridad divididos en tres clases de prioridad.
- La inserción de puntos de expulsión. Dado que el núcleo básico no es expulsivo, sólo puede ser dividido en pasos de proceso que deben ejecutar hasta su conclusión sin interrupción. Entre estos pasos de proceso, se han identificado ciertos puntos de expulsión donde el núcleo puede ser interrumpido de manera segura para planificar un nuevo proceso. La Figura 3.1 ilustra los 160 niveles de prioridad definidos en SVR4. Cada proceso se define como perteneciente a una de estas tres clases de prioridad y se le asigna un nivel de prioridad dentro de la clase. Las clases son los siguientes:
- Tiempo-real (159-100). El proceso en estos niveles de prioridad se garantiza que serán elegidos para ejecutar antes que cualquier proceso de núcleo o de tiempo compartido. Además, los procesos de tiempo real pueden hacer uso de los puntos de expulsión para expulsar procesos de núcleo y procesos de usuario.
- Núcleo (99-60). El proceso en estos niveles de prioridad se garantiza que serán elegidos para ejecutar antes que cualquier proceso de tiempo compartido, pero serán retrasados por los procesos de tiempo real.
- Tiempo-compartido (59-0). Son los procesos de más baja prioridad, pensados para las aplicaciones de usuario que no sean de tiempo real.

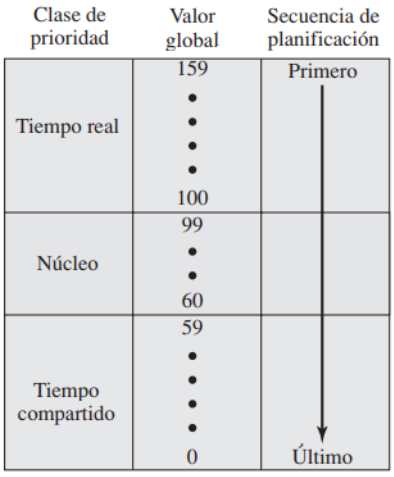

Figura.3.1. Clases de prioridad de SVR4.

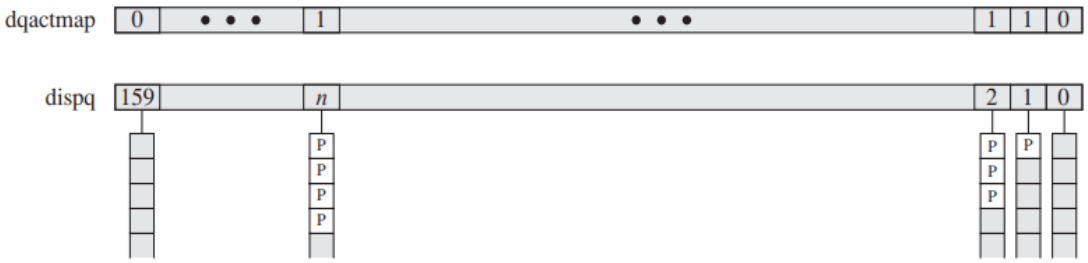

Figura.3.2 Colas de activación de SVR4.

La Figura 3.2 indica cómo está implementada la planificación en SVR4. Asociada con cada nivel de prioridad hay una cola de activación, y los procesos de un nivel de prioridad dado se ejecutan de manera turno circular. Un vector de bits, dqactmap, contiene un bit por cada nivel de prioridad; el bit se pone a uno por cada nivel de prioridad con cola no vacía. Cuando un proceso activo abandona el estado Ejecución, debido a un bloqueo, al vencimiento de su rodaja de tiempo, o por expulsión, el activador comprueba dqactmap y activa un proceso listo de la cola no vacía de mayor prioridad. Además, cuando se alcanza uno de los puntos de expulsión definidos, el núcleo comprueba una variable llamada kprunrun. Si no es 0, significa que al menos un proceso de tiempo real está en estado Listo, y el núcleo expulsa al proceso actual si es de menor prioridad que el proceso listo de tiempo real de mayor prioridad.

PLANIFICACIÓN EN WINDOWS

Prioridad De Procesos E Hilos

Para entender los algoritmos de planificación, primero se deben mostrar los niveles de prioridad que Windows utiliza. Windows utiliza 32 niveles de prioridad, con rango desde el 0 hasta el 31. La prioridad de Windows se organiza en dos bandas o clases: tiempo real y variable. Cada una de estas clases posee 16 niveles de prioridad. Los hilos que necesitan de una atención inmediata se ubican en tiempo real, el cual contiene a funciones tales como la de comunicación y las tareas que se están ejecutando en tiempo real.

- Dieciséis niveles de tiempo real (16 a 31)
- Dieciséis niveles variables (0 a 15), de los cuales el nivel 0 está reservado para la lectura de la página 0.

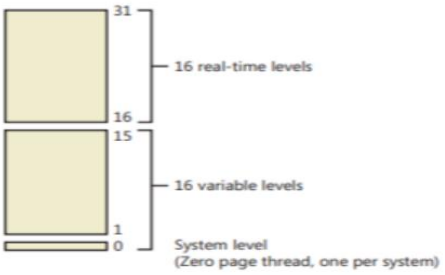

Figura.4.0 Prioridad que utiliza Windows.

Las prioridades se manejan de manera un poco diferente en las dos bandas (Figura 4.0). En la clase de prioridad de tiempo real, todos los hilos tienen una prioridad fija que nunca cambia. En la clase de prioridad variable, la prioridad del hilo comienza con algún valor asignado inicialmente y luego puede cambiar, arriba o abajo, durante la vida del hilo. Una cola FIFO en cada nivel de prioridad, pero un proceso puede migrar a una de las otras colas dentro de la clase de prioridad variable, pero un hilo en el nivel de prioridad 15 no puede pasar al nivel 16 ni a ningún otro nivel de la clase de tiempo real.

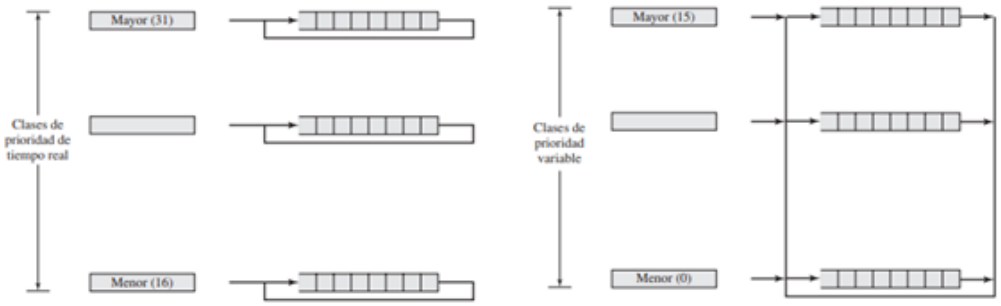

Figura.4.1. Prioridades de activación de hilos en Windows.

Al activarse un hilo de la clase de prioridad variable, que es conocida como prioridad dinámica, puede fluctuar dentro de unos límites dados. La prioridad dinámica nunca debe caer por debajo de rango inferior de la prioridad de base del hilo y no puede exceder 15 (Figura 4.2). Cada objeto hilo asociado con este objeto proceso posee una prioridad inicial entre 2 y 6, la prioridad dinámica de cada hilo se sitúa entre el rango de 2 a 15. Cuando un hilo es interrumpido porque ha utilizado totalmente su cuanto, de tiempo actual, Windows baja su prioridad. Si un hilo se interrumpe por el un evento de E/S, Windows sube su prioridad. Así, los hilos limitados por procesador tienden a prioridades menores y los hilos limitados por E/S tienden a prioridades mayores.

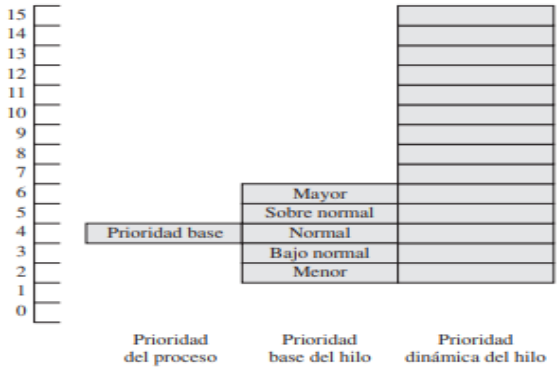

Figura.4.2. Ejemplo de relación entre las prioridades de Windows.

El hilo de mayor prioridad está siempre activo a menos que se encuentre esperando por un evento. Si hay más de un hilo que tiene la mayor prioridad, entonces se comparte el procesador, en turno circular, entre todos los hilos de ese nivel de prioridad. En un sistema multiprocesador con N procesadores, los (N - 1) hilos de mayor prioridad están siempre activos, ejecutando en exclusiva sobre los (N - 1) procesadores extra. El resto de los hilos, de menor prioridad, comparten el único procesador restante.

CONCLUSIÓN

Tal y como hemos podido comprobar en nuestra investigación, podemos concluir como grupo que se pudo observar la complejidad en cuanto a la planificación de tareas con multiprocesadores. Al momento de tratar el tema de los algoritmos de planificación vemos que son muy diferentes entre sí, pero cuando tratamos el tema de sistemas de multiprocesadores pasan desapercibido por los diversos factores por los que pueda estar componiendo al sistema. Asimismo, se pudo apreciar que la diferencia existente entre distintos algoritmos de planificación es menos evidente al usar un sistema multiprocesador. También se entendió que un proceso en tiempo real es aquel que se ejecuta en conexión con algún proceso, función o conjunto de sucesos externos al sistema y que debe cumplir uno o más plazos para intentar actuar de forma correcta y eficiente con el entorno externo. Pero debido a la gran exigencia de información requerida para efectuar dichas tareas es casi imposible trabajar directamente con plazos. Es por ello que los sistemas operativos abordan diferentes maneras de solucionar este problema, usando distintos algoritmos. Se concluye que la gran atención en este tipo de planificación se debe a las necesidades actuales y futuras.

REFERENCIAS BIBLIOGRÁFICAS

[1] W. Stalling, Sistemas Operativos: Aspectos internos y principios de diseño Quinta edición. Madrid: Universidad Politécnica, 2005.

[2] Tanenbaum Andrew - Sistemas Operativos Modernos-Prentice Hall-segunda edicion-2003

[3] Abraham Silberschatz, Greg Gagne, Peter Baer Galvin, Fundamentos de sistemas operativos McGraw-Hill Interamericana de España S.L., 27 feb. 2006 - 680 páginas

Universidad Nacional del Nordeste

Facultad de Ciencias Exactas y Naturales y Agrimensura

Licenciatura en Sistemas de Información

Cátedra: Sistemas Operativos

Año: 2021

Planificación multiprocesador y en tiempo real – Grupo Nro. 27

Alumno: Buzaglo, Julián Tomás

L.U. N°: 54480

D.N.I. N°:

Alumno: Mulas, Vittorio Augusto

L.U. N°: 49742

D.N.I. N°: 39633931

Alumno: Céspedes, Hernan

L.U. N°: 54383

D.N.I. N°: 42739764

Alumno: Pindo, Eduardo Gabriel

L.U. N°: 50737

D.N.I. N°: 37437186

Alumno: Colman, Alexis Nicolas

L.U. N°: 54341

D.N.I. N°: 42986206

Índice

Figuras

Introducción

Esta monografía se dedica a mostrar y explicar lo que constituye un sistema computacional con múltiples procesadores, junto con la planificación en tiempo real. Donde este se trata de los procesos en plazos de tiempo y la cantidad de tiempo en que se ejecuta. Como objetivo se tiene el poder comprender y capacitarnos sobre lo que es un sistema con multiprocesadores, las ventajas que otorga el utilizarlo y que lo hace diferente al monoprocesador.

Por último, se explica el funcionamiento de la planificación de múltiples procesadores y de tiempo en sistemas operativos como Windows y Linux.

Procesador

Conocemos al **procesador** como La Unidad Central de Proceso encargada de interpretar y procesar las instrucciones de un programa de computadora. Además, controla y coordina las operaciones que realiza el sistema, compuesto por una Unidad de Control, la Unidad Aritmética Lógica y la Unidad de Memoria.

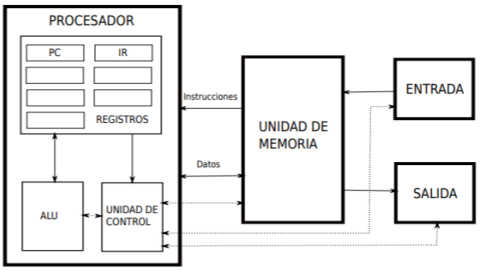

Figura 1.1 Composición de un procesador

Al hablar de procesadores se dan a conocer dos tipos: monoprocesador y multiprocesador. Como su nombre lo dice Monoprocesador es un procesador que solo podrá ejecutar un proceso a la vez, lo que implica que si se desearan ejecutar muchas tareas al mismo tiempo estas no van a ser posibles realizarse un corto tiempo.

Un Multiprocesador es aquel que puede ejecutar varios procesos diferentes, presentan un diseño diferente al de monoprocesadores ya que los programas pueden ejecutarse simultáneamente al poseer una memoria compartida (es decir, comparten el mismo espacio de direccionamiento).

Multiprocesadores

Cuando un sistema computador contiene más de un procesador, el diseño de la función de planificación plantea nuevas cuestiones. Un **multiprocesador** con memoria compartida (o sólo multiprocesador) es un sistema de cómputo en el que dos o más CPUs comparten todo el acceso a una RAM común. Un programa que se ejecuta en cualquiera de las CPUs ve un espacio normal de direcciones virtuales (por lo general paginadas). La única propiedad inusual que tiene este sistema es que la CPU puede escribir cierto valor en una palabra de memoria y después puede volver a leer esa palabra y obtener un valor distinto (tal vez porque otra CPU lo cambió). Si se organiza en forma correcta, esta propiedad forma la base de la comunicación entre procesadores: una CPU escribe ciertos datos en la memoria y otra lee esos datos.

En su mayor parte, los sistemas operativos multiprocesadores son sólo sistemas operativos regulares. Las tareas que realiza un multiprocesador son variadas, manejan las llamadas al sistema, administran la memoria, proveen un sistema de archivos y administran los dispositivos de E/S. Sin embargo, hay ciertas áreas en las que tienen características únicas, estas áreas son la sincronización de procesos, la administración de recursos y la programación de tareas.

Chips multinúcleo:

A medida que mejora la tecnología de fabricación de chips, los transistores se están haciendo cada vez más pequeños y es posible colocar cada vez más de ellos en un chip.

Los chips multinúcleo se les conoce algunas veces como CMPs (Multiprocesadores a nivel de chip). Desde la perspectiva del software, los CMPs no son en realidad tan distintos de los multiprocesadores basados en bus, o de los multiprocesadores que utilizan redes de conmutación. Aunque las CPUs pueden o no compartir cachés, siempre comparten la memoria principal, y ésta es consistente en el sentido de que siempre hay un valor único para cada palabra de memoria. Los circuitos de hardware especiales aseguran que, si hay una palabra presente en dos o más cachés, y una de las CPUs modifica la palabra, ésta se remueva en forma automática y atómica de todas las cachés para poder mantener la consistencia. A este proceso se le conoce como ***snooping***.

Los lenguajes de programación actuales no están preparados para escribir programas altamente paralelos, y los buenos compiladores y herramientas de depuración son escasos. Pocos programadores han tenido experiencia con la programación en paralelo y la mayoría saben poco acerca de cómo dividir el trabajo en varios paquetes que se puedan ejecutar en paralelo. La sincronización, la eliminación de las condiciones de carrera y la prevención del interbloqueo van a ser pesadillas, y el rendimiento sufrirá de manera considerable como resultado. Los semáforos no son la respuesta. Y más allá de estos problemas iniciales, está muy lejos de ser obvio qué tipo de aplicación realmente necesita cientos de núcleos. El reconocimiento de voz en lenguaje natural probablemente podría absorber mucho poder de cómputo, pero el problema aquí no es la falta de ciclos, sino de algoritmos que funcionen. En resumen, los diseñadores de hardware pueden estar ofreciendo un producto que los diseñadores de software no saben cómo usar, y que los usuarios no quieren[1].

Hardware de multiprocesador:

Los multiprocesadores suelen tener una propiedad de que cada CPU pueda direccionar toda la memoria, aunque existen algunos con características adicionales de que cada palabra de la memoria se podrá leer con la misma velocidad que cualquier otra palabra. Estas máquinas se las conocen como multiprocesadores[1]:

- ***UMA (Uniform Memory Access - Acceso Uniforme a la Memoria)***
- ***NUMA (Non-uniform Memory Access, Acceso NO Uniforme a la Memoria)***

Clasificación de sistemas multiprocesador:

Existen varias formas de clasificar a los multiprocesadores, en base a si es acceso uniforme a la memoria o acceso no uniforme a la memoria (UMA y NUMA) y también en base al acoplamiento de estas mismas[2]:

- **Débilmente acoplado o multiprocesador distribuido, o cluster**: consiste en una colección de sistemas relativamente autónomos: cada procesador tiene su propia memoria principal y canales de E/S.
- ***Procesadores de funcionalidad especializada****:* en este caso, hay un procesador de propósito general maestro y procesadores especializados que son controlados por el procesador maestro y que le proporcionan servicios.
- ***Procesamiento fuertemente acoplado****:* consiste en un conjunto de procesadores que comparten la memoria principal y están bajo el control integrado de un único sistema operativo.

Granualidad:

Una buena manera de caracterizar los multiprocesadores y de situarlos en contexto respecto de otras arquitecturas, es considerar la frecuencia de sincronización entre los procesos del sistema, granularidad de sincronización, o frecuencia, entre procesos en el sistema. Podemos distinguir cinco categorías de ***paralelismo*** que difieren en el grado de granularidad[2]:

- ***Paralelismo independiente:*** con el paralelismo independiente no existe una sincronización explícita entre los procesos. Un uso típico de este tipo de paralelismo se da en los sistemas de tiempo compartido. Como hay más de un procesador disponible, el tiempo medio de respuesta de los usuarios será menor.

- ***Paralelismo de grano grueso y muy grueso***: con el paralelismo de grano grueso y muy grueso existe una sincronización entre los procesos, pero a un nivel muy burdo. Este tipo de situación se trata sencillamente como un conjunto de procesos concurrentes ejecutando en un monoprocesador multiprogramado y puede proporcionarse en un multiprocesador con poco o ningún cambio en el software de usuario.

- ***Paralelismo de grano medio***: en este paralelismo el programador debe especificar explícitamente el posible paralelismo de la aplicación.

- ***Paralelismo de grano fino:*** el paralelismo de grano fino representa un uso mucho más complejo del paralelismo que se encuentra en el uso de hilos. Aunque se ha realizado mucho trabajo sobre aplicaciones altamente paralelas, esta es un área especializada y fragmentada con muchas propuestas diferentes.

Tamaño del Grano	Descripción	Intervalo de sincronización (Instrucciones)
Fino	Paralelismo inherente en un único flujo de instrucciones	<20
Medio	Procesamiento paralelo o multitarea dentro de una única aplicación	20-200
Grueso	Multiprocesamiento de procesos concurrentes en un entorno multiprogramado	200-2000
Muy grueso	Procesamiento distribuido entre nodos de una red para conformar un único entorno de computación	2000-1M
Independiente	Múltiples procesos no relacionados	(N/D)

Figura 1.2 Granularidad de procesos

Aspectos del diseño multiprocesador:

- ➔ **Asignación de procesos a procesadores**
- ➔ **El uso de la multiprogramación en cada procesador individual**
- ➔ **Activación del proceso**

Asignación de procesos a procesadores:

La arquitectura multiprocesador es uniforme, lo que quiere decir que ningún procesador tiene una ventaja física particular con respecto al acceso a memoria principal o a dispositivos E/S (entrada y salida), entonces el enfoque más simple de la planificación consiste en tratar cada proceso como un recurso colectivo y asignar procesos a procesadores por demanda. Surge la cuestión de si la asignación debería ser estática o dinámica[2].

Si se asigna un proceso a un procesador de forma permanente, desde su activación hasta su terminación, entonces debe mantenerse una cola a corto plazo dedicada para cada procesador. Una ventaja de este método es que puede existir una sobrecarga menor en la función de planificación por que la asignación al procesador se realiza una sola vez y para siempre[2].

Una *desventaja de la asignación estática* es que un procesador puede estar desocupado, con cola vacía, mientras que otro procesador tiene trabajos pendientes. Para prevenir esta situación, se puede usar una cola común[2].

La forma de asignar los procesos a los procesadores puede ser por medio de dos métodos:

-Arquitectura maestro

-E*sclavo y arquitecturas simétricas*

El maestro es el responsable de la planificación de trabajos. Una vez que un proceso está activo, si el esclavo necesita un servidor (por ejemplo, una llamada para realizar una operación de E/S), debe enviar una solicitud al maestro y esperar a que el servicio se lleve a cabo. Esta solución es bastante simple y exige pequeñas mejoras sobre el sistema operativo multiprogramados del monoprocesador. Las desventajas de esta solución son dos[2]:

1) Un fallo en el maestro hace caer todo el sistema.
2) El maestro puede llegar a ser un cuello de botella del rendimiento.

El uso de la multiprogramación en cada procesador individual*:*

En los multiprocesadores tradicionales, que tratan con sincronizaciones de grano grueso o independientes, está claro que cada procesador individual debe ser capaz de cambiar entre varios procesos para conseguir una alta utilización y por tanto un mejor rendimiento. No obstante, para aplicaciones de grano medio ejecutándose en un multiprocesador con muchos procesadores, la situación es menos clara[2].

Activación de procesos*:*

El último punto del diseño relacionado con la planificación de multiprocesadores es la selección real del proceso a ejecutar, se ha visto que, en un sistema monoprocesador multiprogramados, el uso de prioridades o algoritmos de planificación sofisticados basados en la utilización anterior pueden mejorar el rendimiento sobre la simple estrategia FIFO[2].

Planificación de Hilos:

Una aplicación puede ser implementada como un conjunto de hilos, que cooperan y ejecutan de forma concurrente en el mismo espacio de direcciones. Los **hilos** (encontrados dentro programas y siendo parte de él) son unas secuencias de tareas pequeñas encadenadas que pueden ser ejecutadas por un sistema operativo. Estos pueden ser ejecutados en forma independiente y realizando varias tareas[2].

En un monoprocesador, los hilos pueden usarse como una ayuda a la estructuración de programas para solapar E/S con el procesamiento. Dada la mínima penalización por realizar un cambio de hilo comparado con un cambio de proceso, los beneficios se obtienen con poco coste. Sin embargo, el ***poder completo de los hilos se vuelve evidente en un sistema multiprocesador***. En este entorno, los hilos pueden explotar el paralelismo real dentro de una aplicación. Si los hilos de una aplicación están ejecutándose simultáneamente en procesadores separados, es posible una mejora drástica de sus prestaciones. Sin embargo, para aplicaciones que necesitan una interacción significativa entre hilos (paralelismo de grano medio), pequeñas diferencias en la gestión y planificación de hilos pueden dar lugar a un impacto significativo en las prestaciones[2].

En la planificación multiprocesador de Hilo y la asignación a procesadores se encuentran cuatro enfoques generales:

- ***Compartición de carga***: el término *compartición de carga* se utiliza para distinguir esta estrategia de los esquemas de balanceo de carga en los que los trabajos se asignan de una manera más permanente. Los procesos no van a asignarse a un procesador particular. Se mantiene una cola global de hilos listos y cada procesador, cuando está ocioso, selecciona un hilo de la cola.

- ***Planificación en pandilla:*** un conjunto de hilos relacionados que se planifica para ejecutar, sobre un conjunto de procesadores al mismo tiempo, en una relación uno-a-uno.

- ***Asignación de procesador dedicado***: opuesto al enfoque de compartición de carga, dado que proporciona una planificación implícita definida por la asignación de hilos a procesadores. Cada proceso ocupa un número de procesadores igual al número de hilos en el programa, durante toda la ejecución del programa. Cuando el programa termina, los procesadores regresan al parque general para la posible asignación a otro programa.

- ***Planificación dinámica***: el número de hilos de un proceso puede cambiar durante el curso de su ejecución.

Compartición de carga:

La compartición de carga es el enfoque más simple y que se surge más directamente de un entorno monoprocesador. Posee algunas ventajas[2].:

- La carga se distribuye uniformemente entre los procesadores, asegurando que no haya procesadores ociosos.
- No se precisa una planificación centralizada: cuando un procesador queda disponible, la rutina de planificación del S.O. se ejecuta en dicho procesador para elegir el siguiente hilo.
- La cola global puede organizarse y ser accesible usando esquemas basados en prioridad, o esquemas que consideran la historia de ejecución o anticipan demandas de procesamiento.

Así mismo se pueden encontrar tres versiones diferentes de la compartición de carga:

- ***Primero en llegar, primero en ser servido (FCFS)***:al haber llegado un trabajo cada uno de sus hilos se disponen consecutivamente al final de la cola compartida. Cuando un procesador está ocioso, toma el siguiente hilo listo, que ejecuta hasta que completa o se bloquea.
- ***Menor número de hilos primero***: la cola compartida de listos se organiza como una cola de prioridad, con la mayor prioridad para los hilos de los trabajos con el menor número de hilos no planificados. Los trabajos con igual prioridad se ordenan de acuerdo con qué trabajo llega primero. Al igual que con FCFS, el hilo planificado se ejecuta hasta completar o bloquear.
- ***Menor números de hilos con expulsión***: es similar al anterior, pero si llega un trabajo con un menor número de hilos que el trabajo en ejecución se expulsarán los hilos pertenecientes al trabajo planificado.

A pesar de que la compartición de carga es denominado como el enfoque más simple sigue poseyendo desventajas:

- La cola central ocupa una región de memoria que debe accederse de manera que se cumpla la exclusión mutua, generando un cuello de botella si muchos procesadores buscan un trabajo al mismo tiempo.
- Es poco probable que los hilos expulsados retomen su ejecución en el mismo procesador. Si cada procesador tiene una caché local, ésta se volverá menos eficaz.

- ❖ Si todos los hilos se tratan como un conjunto común de hilos, es poco probable que todos los hilos de un programa ganen acceso a procesadores a la vez.

Planificación en pandilla:

El término de planificación en pandilla se aplica a la planificación simultánea de los hilos que van a componer un proceso individual. La planificación en pandilla es para aplicaciones paralelas de grano medio o fino cuyo rendimiento se ve degradado de forma importante cuando cualquier parte de la aplicación no está ejecutando mientras otras partes están listas para ejecutar. La planificación en pandilla posee los siguientes beneficios[2]:

- Si se ejecutan en paralelo procesos estrechamente relacionados, puede producirse el bloqueo por sincronización, necesitar menos cambios de procesos y aumento de prestaciones.
- La sobrecarga de planificación puede reducirse dado que una decisión única afecta a varios procesadores y procesos a la vez.

En la planificación en pandilla se minimizan los cambios de proceso. Supongamos que un hilo de un proceso está ejecutando y alcanza un punto en el que debe sincronizarse con otro hilo del mismo proceso. Si este otro hilo no está ejecutando, pero está listo para ejecutar, el primer hilo quedará colgado hasta que suceda un cambio de proceso en otro procesador que tome el hilo necesario.

Asignación de procesador dedicado:

Cada uno de sus hilos se asigna a un procesador que permanece dedicado al hilo hasta que la aplicación concluye. Este enfoque puede parecer un desperdicio de tiempo de procesador. Si un hilo de una aplicación se bloquea esperando por E/S o por sincronización con otro hilo, entonces el procesador de ese hilo se queda ocioso: no hay multiprogramación de los procesadores[2].

Pueden realizarse dos observaciones para esta estrategia:

- ❖ En un sistema altamente paralelo, con decenas o cientos de procesadores, cada uno de los cuales representa una pequeña fracción del coste del sistema, la utilización del procesador deja de ser tan importante como medida de la eficacia o rendimiento.
- ❖ Evitar totalmente el cambio de proceso durante la vida de un programa debe llevar a una sustancial mejora de velocidad de ese programa.

Planificación dinámica:

Tanto el sistema operativo como la aplicación van a estar involucrados en tomar las decisiones de planificación. El sistema operativo es el responsable de particionar los procesadores entre trabajos. Cada trabajo utiliza los procesadores actualmente en su partición para ejecutar algún subconjunto de sus tareas ejecutables proyectando estas tareas sobre hilos[2].

Con este método la responsabilidad de planificación del SO se limita sobre todo a la asignación del procesador y actúa de la siguiente manera:

- Si hay procesadores ociosos, utilizarlos para satisfacer la solicitud.
- En otro caso, si el trabajo que realiza la solicitud acaba de llegar, ubicarlo en un único procesador quitándoselo a cualquier trabajo que actualmente tenga más de un procesador.
- Si no se puede satisfacer cualquier parte de la solicitud, mantenerla pendiente hasta que un procesador pase a estar disponible, o el trabajo rescinda la solicitud.
- Examinar la cola actual de solicitudes de procesadores no satisfechas. Asignar un único procesador a cada trabajo en la lista que actualmente no tenga procesadores.
- Luego volver a examinar la lista, asignando el resto de los procesadores de manera FCFS.

Multihilos:

Multihilo se refiere a la capacidad de un sistema operativo de dar soporte a múltiples hilos de ejecución en un solo proceso. El enfoque tradicional de un solo hilo de ejecución por proceso, en el que no se identifica con el concepto de hilo, se conoce como estrategia monohilo. Otros sistemas operativos, como algunas variedades de UNIX, soportan múltiples procesos de usuario, pero solo un hilo por proceso. El entorno de ejecución de Java es un ejemplo de sistema con único proceso y múltiples hilos[1].

En un entorno multihilo, un proceso se define como la unidad de asignación de recursos y una unidad de protección. Se asocian con los siguientes procesos:

- Un espacio de direcciones virtuales que soporta la imagen del proceso.
- Acceso protegido a procesadores, otros procesos (para comunicación entre procesos), archivos y recursos de E/S

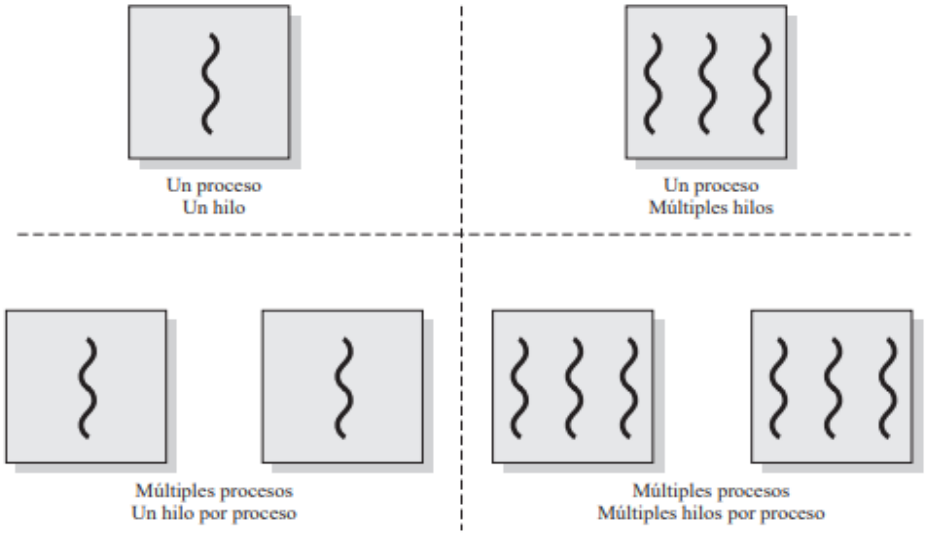

Figura 1.3 Ejecución de hilos

Buses:

Se denomina **bus**, en informática, al conjunto de conexiones físicas (cables, placa de circuito impreso, etc.) que pueden compartirse con múltiples componentes de hardware para que se comuniquen entre sí. El propósito de los buses es reducir el número de rutas necesarias para la comunicación entre los distintos componentes, al realizar las comunicaciones a través de un solo canal de datos. Esta es la razón por la que, a veces, se utiliza la metáfora "autopista de datos".

Buses Múltiples:

Cada procesador está conectado a todos los buses y cada bus a todos los módulos de memoria, así un procesador puede acceder a un módulo de memoria vía un bus en tiempo compartido, si el módulo direccionado está libre y un bus disponible. El sistema de interconexión no puede ser una unidad totalmente pasiva, cada bus necesitará de un dispositivo que controle cada uno de los puntos de conexión de las unidades al bus. Este circuito debe ser modular y rápido al ejecutar el algoritmo de asignación de buses.

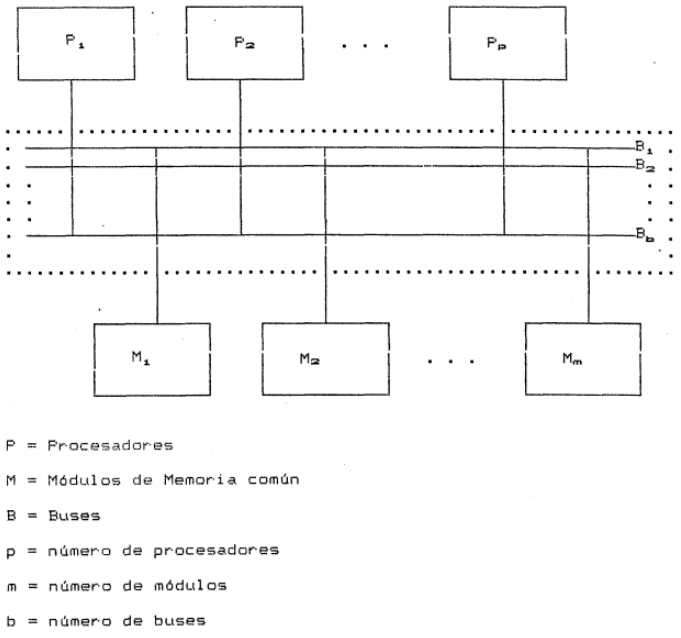

Figura 1.4 Buses múltiples

Planificación de tiempo real

La **planificación en tiempo real** es una disciplina cuyo objetivo es cumplir plazos, donde la corrección del sistema no solo dependerá del resultado sino también de cuando se obtiene. Se asocia un plazo de inicio y fin a cada tarea[2].

Sistema en tiempo real son aquellos que deben producir respuestas correctas dentro de un intervalo de tiempo definido. Si el tiempo excediera ese límite se produciría un funcionamiento erróneo[2].

Clasificación (según requisitos temporales):

– Tiempo real estricto (hard real time): Cuando es absolutamente necesario que la respuesta se produzca dentro del límite especificado, de lo contrario se produciría un daño inaceptable o un control fatal en el sistema. Ej.: control de vuelo.

– Tiempo real no estricto (soft real time): Cuando se permite la pérdida ocasional de especificaciones temporales, aunque debe cumplirse normalmente. Ej.: sistema de adquisición de datos

– Tiempo real firme (firm real time): Cuando se permite la pérdida ocasional de especificaciones temporales, pero dicha pérdida no implica beneficios ya que la respuesta retrasada es descartada. Ej.: sistema multimedia.

Clasificación de tareas de tiempo real:

- ***Tareas periódicas:*** se ejecutan en forma periódica, es decir una vez por periodo[3].
- ***Tareas aperiódicas:*** tiene un plazo de inicio o de fin, es decir podrá tener una restricción ya sea al comienzo o en su finalización[3].

Requisitos de los sistemas operativos de tiempo real:

- ***Determinismo:*** Realiza operaciones en instantes de tiempo fijos o dentro de intervalos de tiempo predeterminados. El determinismo se preocupa de cuánto tiempo tarda el sistema operativo antes del reconocimiento de una interrupción. Cuando múltiples procesos compiten por los recursos y tiempos de procesador, ningún sistema podrá ser totalmente determinista[2].

- ***Reactividad:*** Se preocupa de cuánto tiempo tarda el sistema operativo después del reconocimiento, en servir la interrupción. *Tiempo de respuesta = determinismo + reactividad.* La reactividad incluye a tres aspectos[2]:
    - Cantidad de tiempo necesario para manejar inicialmente la interrupción y comenzar a ejecutar la rutina de servicio de la interrupción (RSI).
    - La cantidad de tiempo necesario para realizar la RSI, esto dependerá de la plataforma hardware.
    - El efecto del anidamiento de interrupciones. Si una RSI es interrumpida por la llegada de otra interrupción, entonces el servicio va a retrasarse.

- ***Control de usuario*:** En un sistema de tiempo real se le permite al usuario un control de grano fino sobre la prioridad de la tarea. Los usuarios deben de ser capaces de distinguir cuando una tarea es dura o blanda, y poder especificar cuáles son las prioridades relativas dentro de cada clase[2].

- ***Fiabilidad:*** En un sistema de tiempo real se responde y controla los eventos en tiempo real[2].
    - ***Fallo suave:*** Relacionado con la fiabilidad. Una característica que se refiere a la habilidad del sistema de fallar de tal manera que se preserve tanta capacidad y datos como sea posible, es decir en caso de fallo que este sea lo menos perjudicial posible. Ejemplos: no abortar procesos con fallos, incumplir los plazos de las tareas menos críticas.
- ***Estabilidad:*** Un sistema de tiempo real es estable sí, en los casos en los que sea imposible cumplir los plazos de todas las tareas, el sistema cumplirá los plazos de sus tareas más críticas, de más alta prioridad, aunque los plazos de algunas tareas menos críticas no se satisfagan[2].

Los algoritmos de planificación de tiempo real caracterizan en función de varios factores[3]:

- ¿Cuándo se realiza el análisis de planificabilidad?
- ¿Cómo se realiza el análisis, estática o dinámicamente?
- ¿El resultado del análisis es un plan o no?

Tipos de algoritmos:

- **Estáticos:**
    - ➢ ***Dirigidos por tablas:*** análisis estático de planificabilidad que da lugar a un plan de tiempos de inicio de ejecución.
    - ➢ ***Dirigidos por prioridad:*** análisis estáticos de planificabilidad que permite asignar prioridades a las tareas y que debe utilizarse junto a un algoritmo de planificación expulsivo basado en prioridades.
- **Dinámicos:**
    - ➢ ***Basados en un plan:*** una nueva tarea es aceptada solo si es posible crear un plan de ejecución que satisfaga todas las restricciones temporales.
    - ➢ ***De mejor esfuerzo:*** intenta cumplir todos los plazos y en caso de no poder intenta provocar un "fallo suave".

La **planificación estática dirigida por tabla** se aplica a las tareas periódicas. Los datos de entrada para el análisis van a ser el tiempo periódico de llegada, plazo periódico de finalización y la prioridad relativa de cada tarea. El planifica busca encontrar el plan que le pueda permitir cumplir con todos los requisitos de todas las tareas periódicas[2].

La **planificación estática con expulsión dirigida por prioridad** hace uso del mecanismo de planificación expulsivo dirigido por prioridades común a la mayoría de los sistemas multiprogramados que no son de tiempo real[2].

Con la **planificación dinámica basada en un plan** al llegar una nueva tarea (antes de que comience su ejecución) se intenta crear un plan que pueda contener a las tareas previamente planificadas al igual que la nueva. Si la tarea recién llegada puede ser planificada sin que ninguna otra tarea pueda perder su plazo entonces la nueva tarea va a ser aceptada y se va a agregar en el nuevo plan de planificación[2].

Con la **planificación dinámica de mejor esfuerzo** cuando llega una nueva tarea el sistema le asigna una prioridad basada en las características de la misma. Se utiliza algún tipo de planificación basada en plazos, como la planificación del plazo más cercano[2].

Planificación por plazos:

Esta planificación por plazos se basa en poseer información adicional acerca de cada tarea. En su forma más general puede utilizarse la siguiente información de cada tarea[2]:

- ***Tiempo de activación:*** instancia en el cual la tarea pasa a estar lista para ser ejecutada. Cuando la tarea es repetitiva o periódica se conoce de antemano la secuencia de tiempo. Caso contrario cuando la tarea es aperiódica el tiempo será conocido previamente o el sistema operativo es consciente de la tarea cuando esta esté lista para ser ejecutada.

- ***Plazo de comienzo:*** instancia en la cual la tarea debe comenzar.

- ***Plaza de conclusión***: instancia en la cual la tarea debe de haber completado. Las aplicaciones de tiempo real van a tener plazos de comienzo o conclusión, pero no ambos.

- ***Tiempo de proceso:*** tiempo necesario para que se ejecute la tarea hasta su conclusión.

- ***Recursos requeridos:*** conjunto de recursos (distintos del procesador) que la tarea va a necesitar durante su ejecución.

- ***Prioridad:*** mide la importancia relativa que va a tener la tarea. Una tarea de tiempo real dura tiene una prioridad absoluta, fallando el sistema si un plazo no se cumple.

- ***Estructura de subtareas***: las tareas pueden ser descompuestas en una subtarea obligatoria (poseyendo un plazo duro) y una subtarea opcional.

Planificación en Windows

Un **sistema operativo para tiempo real** es un sistema operativo capaz de garantizar los requisitos temporales de los procesos que controla. Los sistemas operativos convencionales no son apropiados para la realización de sistemas de tiempo real, debido a que: no tienen un comportamiento determinista y no permiten garantizar los tiempos de respuesta[4].

Un sistema operativo para tiempo real debe ofrecer las siguientes facilidades[4]:

– ***Concurrencia***: procesos ligeros (threads) con memoria compartida.
– ***Temporización***: medida de tiempos y ejecución periódica.
– ***Planificación***: prioridades fijas con desalojo, acceso a recursos con protocolos de herencia de prioridad.
– ***Dispositivos de E/S***: acceso a recursos hardware e interrupciones.

El diseño de un **procesador Windows** está limitado por la necesidad de proporcionar soporte a diversos entornos de sistemas operativos. Los procesos soportados por diferentes entornos de sistemas operativos se diferencian en varias cosas, como por ejemplo[2]:

1-Denominación de los procesos.

2-Si se proporcionan hilos con los procesos.

3-Cómo se protege a los recursos de los procesos.

4-Qué mecanismos se utilizan para la comunicación y sincronización entre procesos.

5-Cómo se relacionan los procesos entre sí.

Como consecuencia, las estructuras de los procesos y servicios proporcionados por el núcleo de Windows son relativamente sencillos y de propósito general, permitiendo a cada subsistema del sistema operativo que emule una estructura y funcionalidad particular del proceso. Algunas características importantes de los procesos Windows son las siguientes:

-Los procesos están implementados como objetos.

-Un proceso ejecutable puede contener uno o más hilos.

-Tanto el objeto proceso como el objeto hilo, tienen funciones de sincronización preconstruidas.

La **estructura de Windows**, orientada a objetos facilita el desarrollo de procesos. Hace uso de dos tipos de objetos relacionados: los procesos e hilos[2].

- *Procesos:* corresponde a la representación de un trabajo de usuario o aplicación que posee recursos (memoria y archivos abiertos).

- *Hilos:* son entidades de trabajo activables, que trabajan secuencialmente y son interrumpibles de forma que el procesador puede cambiar entre hilos.

Cada vez que Windows trabaja con un proceso, este se representa por un objeto. A continuación, se observa cómo es la estructura de este mismo:

Un ***proceso*** se define por atributos y se encapsula una serie de acciones que puede realizar. El proceso realizará el servicio cuando reciba el mensaje que se espera. La forma de acceder o invocar este servicio es a través de mensajes a un objeto "proceso" que proporciona este servicio.

Cuando se crea un nuevo proceso, utiliza una plantilla, o clase para luego instanciarla y generar un nuevo objeto "proceso". En el momento de la creación se asignan valores a sus atributos. Siempre en Windows se debe contener al menos un hilo para ejecutar.

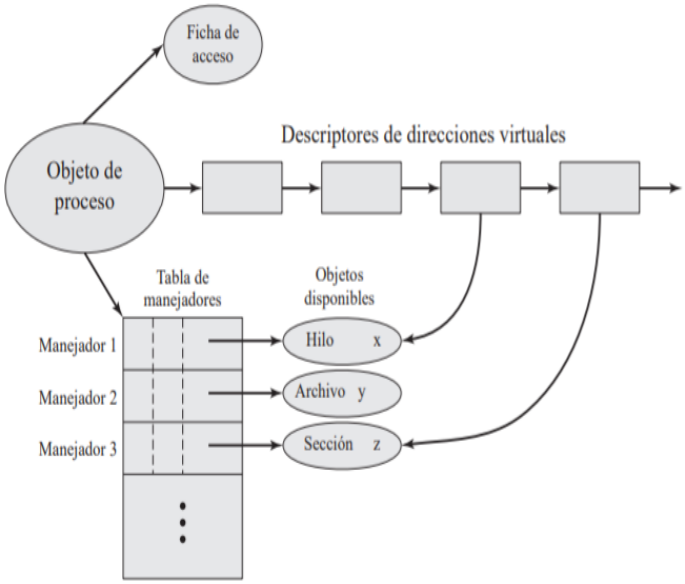

Figura 2.1 Estructura de un proceso

Este hilo puede a su vez, crear más hilos. Con esto se comienza a trabajar con procesadores multi hilos, o multiprocesadores. En un sistema con esta característica, múltiples hilos de un mismo proceso, pueden trabajar en paralelo, y es importante destacar que algunos de los atributos de los hilos se asemejan a los de los procesos, y en estos casos, el valor del atributo del hilo se deriva al valor del proceso.

Windows hace uso de un planificador expulsivo basado en las prioridades, los hilos con prioridades de tiempo real tienen preferencias sobre otros hilos. En un monoprocesador cuando un hilo cuya prioridad es mayor que la del hilo actualmente en ejecución pasa a estar listo, el hilo menos prioritario es expulsado y se le entrega el procesador al hilo de mayor prioridad. Una vez que el hilo de la clase con prioridad variable fuera activado, su prioridad real conocida como prioridad dinámica del hilo, podrá moverse dentro de los límites dados. Esta prioridad dinámica nunca puede caer por debajo del rango inferior de la prioridad base del hilo y tampoco excederlo.

En un sistema multiprocesador con N procesadores, los (N-1) hilos de mayor prioridad están siempre activos, ejecutándose de manera exclusiva en los (N-1) procesadores extra. El resto de los hilos de menor prioridad comparten el único procesador restante.

Planificación en Linux

El sistema operativo ***Linux posee la capacidad para la planificación en tiempo real***, así como un planificador para procesos no de tiempo real que hacen uso del algoritmo de planificación tradicional. Así también incluye un planificador para procesos NO de tiempo real. Dentro de Linux se pueden encontrar tres clases de planificación: SCHED_FIFO, SCHED_RR, SCHED_OTHER[2].

Clases de planificación en Linux:

- SCHED_FIFO: Hilos de tiempo real planificados FIFO
- SCHED_RR: Hilos de tiempo real planificados con turno circular
- SCHED_OTHER: otros hilos no de tiempo real

Dentro de las clases se pueden utilizar múltiples prioridades, siendo las prioridades de las clases de tiempo mayores que las prioridades para una clase SCHED_OTHER.

Los hilos de FIFO aplican tres reglas:

1. El sistema no interrumpirá un hilo FIFO en ejecución excepto en estos casos:
    a. Otro hilo FIFO de mayor prioridad pasa a estado listo
    b. Un hilo FIFO pasa a estar bloqueado en espera por algún evento (como E/S).
    c. Un hilo FIFO en ejecución cede voluntariamente el procesador realizando una llamada primitiva.

2. Cuando un hilo FIFO en ejecución es interrumpido, se le sitúa en una cola asociada con su prioridad.

3. Cuando un hilo FIFO pasa a estar listo y ese hilo es de mayor prioridad que el hilo actualmente en ejecución, entonces el hilo actualmente en ejecución será expulsado y el nuevo hilo FIFO de mayor prioridad pasará a ser ejecutado.

El planificador de ***Linux utiliza una estructura de datos clave***, conocida como cola de ejecución (runqueue). Una cola de ejecución está asociada con cada CPU en el sistema, y mantiene (entre otra información) dos arreglos: activas y expiradas[1].

Cada uno de estos campos es un apuntador a un arreglo de 140 cabezas de listas, cada una de las cuales corresponde a una prioridad distinta. La cabeza de lista apunta a una lista doblemente enlazada de procesos con una prioridad específica[1].

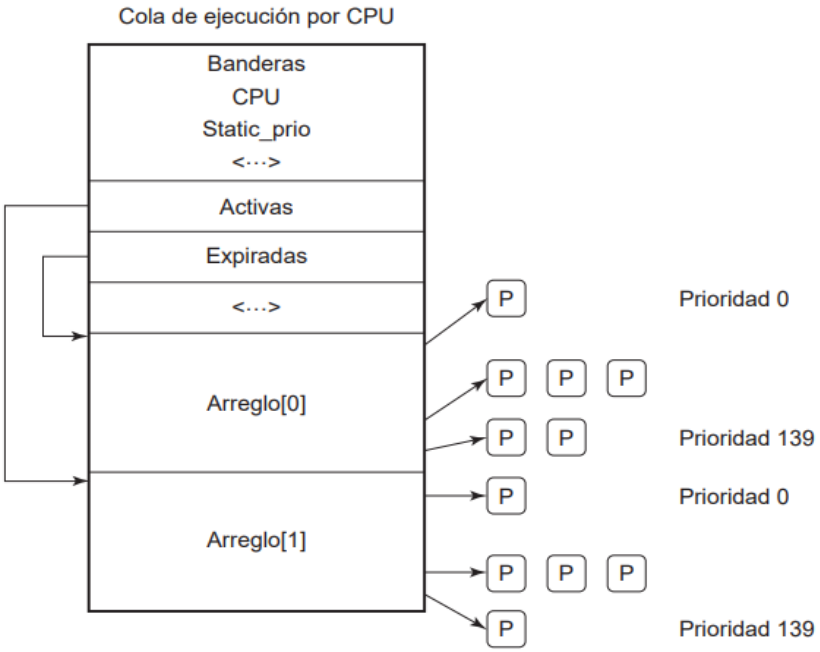

Figura 2.2 Estructura de ejecución Linux

El *planificador selecciona una tarea del arreglo activo de mayor prioridad*. Si expira la ranura de tiempo (quantum) de esa tarea, se pasa a una lista de tareas expiradas (que puede estar en un nivel de prioridad distinto). Si la tarea se bloquea (por ejemplo, para esperar un evento de E/S) antes de que expire su ranura de tiempo, una vez que ocurra el evento y se pueda reanudar su ejecución, se coloca de vuelta en el arreglo de tareas activas y su ranura de tiempo se disminuye para reflejar el tiempo de la CPU que ya consumió.

Relación con las tareas de tiempo real: Las tareas de tiempo-real se manipulan de manera distinta que las tareas no de tiempo real en las colas de prioridad. Se aplican las siguientes consideraciones[2]:

1. Todas las tareas de tiempo real tienen solamente prioridad estática; no se realizaron cálculos de prioridad dinámica.

2. Las tareas SCHED_FIFO no tienen asignadas rodajas de tiempo. Estas tareas se planifican siguiendo la disciplina FIFO. Si una de estas tareas se bloquea, cuando se desbloquee volverá a la misma cola de prioridad en la lista de colas activas.

3. Aunque las tareas SCHED_RR tienen asignadas rodajas de tiempo, tampoco se le traslada nunca a la lista de colas vencidas. Cuando una de estas tareas consume toda su rodaja de tiempo, se le devuelve a su cola de prioridad con el mismo valor de rodaja de tiempo. Los valores de la rodaja de tiempo nunca cambian.

Administración de la memoria en Linux:

El modelo de memoria de Linux es simple, para que los programas sean portátiles y se pueda implementar Linux en máquinas con unidades de administración de memoria que tengan amplias diferencias, desde casi nada (por ejemplo, la IBM PC original) hasta el hardware de paginación sofisticado. Ésta es un área del diseño que ha cambiado muy poco en décadas. Ha funcionado bien, por lo que no ha necesitado mucha revisión. Ahora examinaremos el modelo y la forma en que se implementa[1].

Conclusión

A partir de las investigaciones realizadas, consideramos efectivas las ventajas y eficiencias que posee un multiprocesador (viéndose claro a la hora de ejecutar algún proceso) sin antes poder dar a conocer lo que es procesador. Además de haber conocido y fundamentado el funcionamiento dentro de los sistemas operativos de Windows y Linux (otros temas tratados en la monografía).

Comprendimos y obtuvimos mas respuestas sobre el funcionamiento del sistema sobre los diferentes casos de tareas que puedan llegar a surgir. A esto logramos relacionarlo con la planificación en tiempo real y sus tipos de planificaciones.

Por último, queremos destacar el uso de términos ya aprendidos anteriormente en materias cursadas tales casos como: procesador, multiprocesador, hilos y clases.

Referencias

[1]T. Andrew.Sistemas Operativos Modernos.Vrije Universiteit Amsterdam, Holanda.2008

[2] W. Stalling, Sistemas Operativos: Aspectos internos y principios de diseño Quinta edición. Madrid: Universidad Politécnica, 2005.

[3] G.R López. Planificación multiprocesador y de tiempo real, March 2017.[Online].Available: https://docplayer.es/53324024-Planificacion-multiprocesador-y-de-tiempo-real.html

[4] Facultad de Informática.Universidad de Murcia.[Online].Available: http://www.isa.uniovi.es/docencia/TiempoReal/Recursos/rtos_linux.pdf

Universidad Nacional del Nordeste

Facultad de Ciencias Exactas y Naturales y Agrimensura

Licenciatura en Sistemas

Cátedra: Sistemas Operativos

Año: 2024

Grupo N°3 - Integrantes:

Alumno	DNI:
➢ Benites, Matías Maximiliano -	42098665
➢ Gómez, Sebastián Exequiel -	42603108
➢ Más González, Fernando -	37157720
➢ Orban, Tobías Naim -	46385637
➢ Ristovich, Mauro Ezequiel Alejandro –	41380447

Monografía

Tema N°19 - Procesamiento Distribuido, Cliente/Servidor y Clúster

INDICE GENERAL

Contenido

ÍNDICE DE FIGURAS

1 - INTRODUCCION

Los sistemas distribuidos están formados por un conjunto de computadoras interconectadas. Aunque cada una de estas máquinas se encuentra en ubicaciones físicas distintas, y tiene su propio hardware y software, todas comparten una red de comunicación común que las enlaza. De esta manera, el programador puede verlas como un único sistema, pero con múltiples componentes, experimentando una distribución transparente que opera de manera coordinada y sincronizada.

Desde una perspectiva informática, un sistema distribuido es aquel cuyos elementos, ya sean de hardware o software, están alojados en diferentes nodos de una red. Estos elementos se comunican y se sincronizan mediante el intercambio de mensajes.

El objetivo principal de este trabajo es exponer los aspectos clave que forman parte de los sistemas distribuidos, analizando algunos de los conceptos esenciales del software distribuido, como la arquitectura cliente/servidor, la transmisión de mensajes y las llamadas a procedimientos remotos. También se revisa la importancia de la arquitectura en clústeres.

2 – DESARROLLO

2.1 – COMPUTACION CLIENTE/SERVIDOR

2.1.1 – ¿Qué es la Computación Cliente Servidor?

Como el propio término sugiere, un entorno **"Cliente / Servidor"** está formado por clientes y servidores:

- Las máquinas cliente son normalmente simples “PCs” o **"Estaciones de Trabajo"** que proporcionan una interfaz de fácil manejo al usuario final.
- Los servidores son un entorno **"Cliente / Servidor"** que brinda un conjunto de servicios compartidos a los clientes.

La computación **"Cliente / Servidor"** normalmente es una computación distribuida, cuyos usuarios, aplicaciones y recursos se distribuyen en respuesta a los requisitos de trabajo, y se unen a través de una **"LAN"**, **"WLAN"** o **"Internet"**.

Algunos términos característicos de productos **"Cliente / Servidor"** son:

- **Interfaz de programación de aplicaciones (API)**: Un conjunto de funciones y programas que permiten a los clientes y servidores intercomunicarse.
- **Cliente**: Un elemento de la red que solicita información, normalmente un “PC” o **"Estación de trabajo"**. Puede interrogar a una base de datos o solicitar información de un servidor.
- **Middleware**: Un conjunto de controladores, “API”, y software adicional que mejoran la conectividad entre una aplicación cliente y un servidor.
- **Base de datos relacional**: Una base de datos en la que el acceso a la información está restringido por la selección de filas que satisfacen todos los criterios de búsqueda.
- **Servidor**: Un computador, normalmente una estación de trabajo de gran potencia, un minicomputador, o un mainframe, que almacena la información para los clientes de la red.
- **Lenguaje estructurado de Consultas (SQL)**: Lenguaje desarrollado por IBM y estandarizado por ANSI que permite acceder, crear, actualizar e interrogar bases de datos relacionales.

Figura 01. Esquema del Modelo Cliente Servidor.

2.1.2 - ¿QUÉ LO DIFERENCIA DE OTRAS SOLUCIONES DE PROCESAMIENTO DISTRIBUIDO?

- Es fundamental que los usuarios cuenten con aplicaciones intuitivas en sus equipos, lo que les otorga mayor control sobre el uso de sus recursos y permite a los administradores responder eficientemente a las demandas locales.
- La clave está en centralizar tanto las bases de datos corporativas como las herramientas de gestión de red, aunque las aplicaciones se encuentren distribuidas. Este enfoque facilita a los administradores un control más efectivo y mejora la interoperabilidad entre sistemas, aliviando a los departamentos de TI de gran parte del trabajo de mantenimiento.
- Se apuesta por mantener un sistema abierto y modular, ofreciendo a los usuarios la flexibilidad de elegir y combinar productos de diferentes proveedores según sus necesidades.
- La administración y la seguridad de la red son prioritarias en la estructura y operación de los sistemas de información, garantizando un entorno fiable y bien gestionado.

2.1.3 – APLICACIONES CLIENTE/SERVIDOR

La característica fundamental de una arquitectura “Cliente / Servidor” es la distribución de las tareas de la aplicación entre el cliente y el servidor.

La plataforma y el sistema operativo del cliente y del servidor pueden ser diferentes. De hecho, pueden existir diferentes plataformas de clientes y sistemas operativos y protocolo de comunicación y soportar las mismas aplicaciones.

Quien permite que interactúen el cliente y el servidor es el software de comunicaciones. El principal ejemplo de este software es **"TCP/IP".**

El diseño de la interfaz de usuario es sumamente decisivo en la máquina cliente, para que esta sea un éxito en el entorno **"Cliente / Servidor",** ya que es como el usuario interactúa con el sistema.

2.1.4 – APLICACIONES DE BASES DE DATOS

Para explicar el concepto utilizaremos las bases de datos relacionales como ejemplo de una de las aplicaciones de tipo cliente/servidor. En este entorno, el servidor es básicamente un servidor de base de datos. Mediante transacciones es que interactúan el cliente con el servidor, en las que el cliente hace una apetición a la base de datos y recibe una respuesta.

Como el servidor es el responsable del mantenimiento de la base de datos se requiere un complejo sistema gestor de base de datos y lo que une al cliente y al servidor es el software que hace posible que el cliente realice peticiones para acceder al servidor de la base de datos, como lo es, por ejemplo, el lenguaje estructurado de consultas (SQL) **[1].**

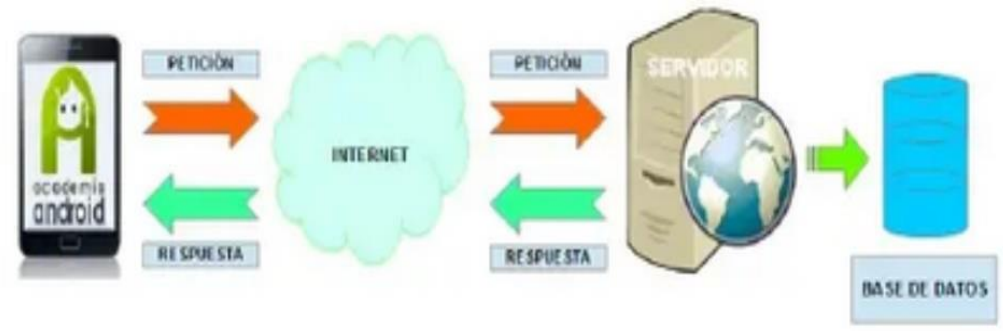

Figura 02. Arquitectura cliente/servidor para aplicaciones de bases de datos.

2.1.5 – CLASES DE APLICACIONES CLIENTE/SERVIDOR

Hay una serie de implementaciones que dividen el trabajo entre el cliente y el servidor de diferentes maneras que podríamos analizar a continuación:

Procesamiento basado en el HOST: Se refiere a los entornos “mainframe” tradicionales en los que virtualmente todo el procesamiento se realiza en el host central. No es una verdadera computación cliente/servidor como tal. menudo, la interfaz de usuario se realiza a través de un interfaz tonto. Incluso si el usuario está empleando una computadora, la estación del usuario se limita al papel de emulador de terminal **[2].**

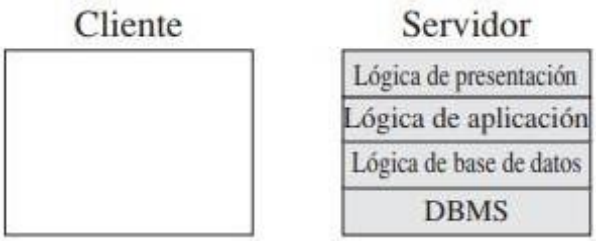

Figura 03. Procesamiento basado en el host.

- Procesamiento basado en el SERVIDOR: En esta configuración el cliente es el responsable de proporcionar la interfaz gráfica de usuario, mientras todo el procesamiento se realiza en el servidor. Es una configuración típica en sistemas a nivel de departamento. Sin embargo, este tipo de configuraciones no suele generar grandes ventajas en productividad, ni cambios fundamentales en las funciones de negocios soportadas por el sistema.

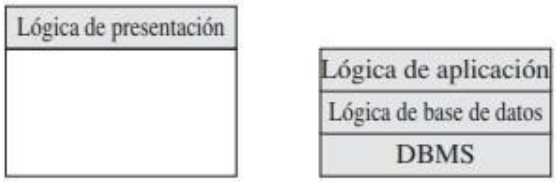

Figura 04. Procesamiento basado en el servidor.

- Procesamiento basado en el CLIENTE: Contrariamente al caso anterior, todo el procesamiento se puede realizar en el cliente, con excepción de las rutinas de validación de datos y otras funciones de la lógica de la base de datos que se pueden realizar mejor en el servidor. Esta arquitectura permite a los usuarios el uso de las aplicaciones adaptadas a las necesidades locales **[2].**

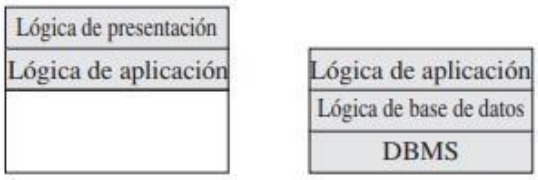

Figura 05. Procesamiento basado en el cliente.

- Procesamiento cooperativo: En este tipo de configuraciones, el procesamiento de la aplicación se realiza de forma óptima, beneficiándose de las maquinas cliente y servidora y de la distribución de los datos. Es más compleja de configurar y mantener, pero proporciona mayor productividad a los usuarios y mayor eficiencia de red que otras configuraciones cliente/servidor **[2].**

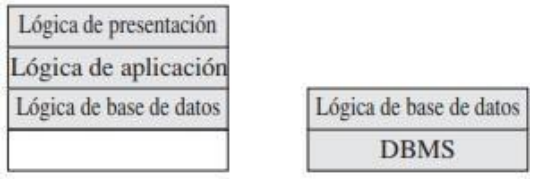

Figura 06. Procesamiento cooperativo.

El procesamiento basado en el cliente y el procesamiento cooperativo se corresponden con las configuraciones en que gran parte de la carga está en la parte cliente. **Las figuras 5 y 6** representan este modelo denominado cliente pesado (fat client). La principal ventaja de este modelo es que se beneficia de la potencia de los escritorios, descargando a los servidores y haciéndolos más eficientes y menos propensos a ser el cuello de botella. Existen, sin embargo, varias desventajas en la estrategia de los clientes pesados. Añadir nuevas funcionalidades suele sobrecargar la capacidad de los ordenadores de escritorio,forzando a las compañías a actualizarse.
Por último, es difícil mantener, actualizar o reemplazar aplicaciones distribuidas entre decenas o centenas de ordenadores. El procesamiento basado en servidor es

representativo de un enfoque de cliente ligero. Este enfoque imita al enfoque tradicional centralizado del host y es, a menudo, la ruta de migración para pasar las aplicaciones corporativas de los mainframes a un entorno distribuido.

2.2 – PASO DE MENSAJES DISTRIBUIDO [3]

En los sistemas de procesamiento distribuido, las computadoras generalmente no comparten la memoria principal, ya que cada una opera como un sistema independiente. Por lo tanto, las técnicas de comunicación entre procesos que dependen de la memoria compartida, como los semáforos, no son aplicables. En su lugar, se emplean métodos basados en el intercambio de mensajes. En esta y la próxima sección, exploraremos las dos técnicas más comunes. La primera es una implementación directa del uso de mensajes dentro de un único sistema. La segunda se basa también en el envío de mensajes, pero a través de llamadas a procedimientos remotos.

- La primera es una aplicación directa de los mensajes tal y como se utilizan en un único sistema.

▪ Un **“Cliente”** necesita algún servicio envía un mensaje con la petición de servicio a un proceso **“Servidor”**. El “Servidor” realiza la petición y envía un mensaje con la respuesta. En su forma más básica, sólo se necesitan dos funciones:

→ **“Send”** (mandar): La función Send especifica un destinatario e incluye el contenido del mensaje.

→ **“Receive”** (recibir). La función Receive indica de quién se desea recibir un mensaje y proporciona un buffer donde se almacenará el mensaje entrante.

- La segunda es una técnica que se basa en el paso de mensajes: las llamadas a procedimiento remoto.

▪ Los procesos hacen uso de los servicios de un módulo de paso de mensajes. Las peticiones de servicio se pueden expresar en términos de primitivas y parámetros.

→ Una primitiva especifica la función que se desea realizar, el formato real de las primitivas depende del software de paso de mensajes que se utilice, esta puede ser una llamada a procedimiento o puede ser un mensaje en sí mismo a un proceso que sea parte del sistema operativo.

→ Los parámetros se utilizan para pasar los datos y la información de control.

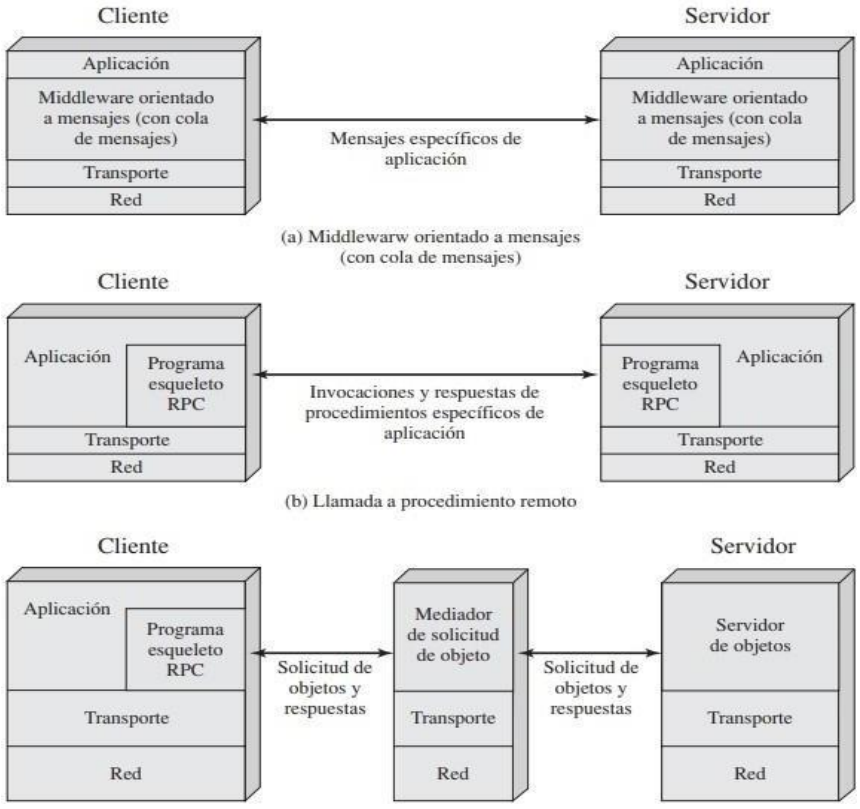

Figura 07. Mecanismo middleware

Figura 08. Primitivas básicas de paso de mensajes

2.2.1 – Fiable vs. No fiable

Un servicio fiable de paso de mensajes es el que garantiza la entrega si es posible. Estos servicios hacen uso de un protocolo de transporte fiable o de una lógica similar, y realizan comprobación de errores, acuse de recibo, retransmisión y reordenamiento de mensajes desordenados. Ya que se garantiza el envío, no es necesario informar al proceso emisor de que el mensaje ha sido enviado. En cualquier caso, si falla el envío del mensaje (por ejemplo, fallo persistente de la red, fallo del sistema de destino), se informa del fracaso al proceso emisor.

En el otro extremo, el servicio de paso de mensajes no fiable simplemente envía el mensaje por la red, pero no se le indica ni el éxito ni el fracaso. Esta alternativa reduce enormemente la complejidad y la sobrecarga de procesamiento y comunicación del servicio de paso de mensajes. **[9]**

2.2.2 - BLOQUEANTE VS. NO BLOQUEANTE

Con las primitivas no bloqueantes o asíncronas, no se suspende a un proceso como resultado de realizar un Send o Receive. De esta forma, cuando un proceso realiza una primitiva Send, el sistema operativo devuelve el control al proceso tan pronto como el mensaje ha sido puesto en la cola para su transmisión o se ha realizado una copia. Si no se realiza copia, cualquier cambio que el proceso emisor haga al mensaje, antes o durante su transmisión, se realizan bajo la propia responsabilidad del proceso. Cuando

el mensaje ha sido transmitido o copiado a un lugar seguro para su transmisión, se interrumpe al proceso emisor para informarle de que el buffer puede ser reutilizado. De forma similar, un Receive no bloqueante permite que el proceso continúe ejecutando. Cuando el mensaje llega, se informa al proceso con una interrupción, o bien el propio proceso puede verificar el estado periódicamente.

La alternativa es utilizar primitivas bloqueantes o asíncronas. Un Send bloqueante no devuelve el control al proceso emisor hasta que el mensaje ha sido transmitido (servicio no fiable) o hasta que el mensaje ha sido enviado y se ha recibido el acuse de recibo (servicio fiable). Un Receive bloqueante no devuelve el control hasta que el mensaje ha sido situado en su correspondiente buffer.

2.3 – LLAMADAS A PROCEDIMIENTO REMOTO

Las llamadas a procedimiento remoto son una variante del modelo básico de paso de mensajes. Hoy en día este método es muy común y está ampliamente aceptado para encapsular la comunicación en un sistema distribuido. Lo esencial en esta técnica es permitir a los programas en diferentes máquinas interactuar a través del uso de llamadas a procedimiento, tal y como lo harían dos programas que están en la misma máquina. Es decir, se utilizan las llamadas a procedimiento para acceder a los servicios remotos. El mecanismo de llamadas a procedimiento remoto se puede ver como un refinamiento del paso de mensajes fiable y bloqueante. La Figura 7(b) muestra la arquitectura general, y la Figura 9 proporciona una vista más detallada. El programa llamante realiza una llamada a procedimiento normal con parámetros de su máquina. Por ejemplo:

CALL P (X, Y) donde:

P = nombre del procedimiento

X = argumentos pasados

Y = valores devueltos

La intención de invocar a un procedimiento remoto en otra máquina puede ser transparente o no al usuario. En el espacio de direcciones del llamante se debe incluir el esqueleto (stub) de un procedimiento P o se debe enlazar dinámicamente en tiempo de llamada. Este procedimiento crea un mensaje que identifica al

procedimiento que se está llamando e incluye sus parámetros. De esta forma se envía el mensaje al sistema remoto y se espera una respuesta. Cuando se recibe una respuesta, el procedimiento esqueleto vuelve al programa llamante, proporcionando los valores devueltos.

En la máquina remota, hay otro programa esqueleto asociado con el procedimiento llamado. Cuando llega un mensaje, se examina y se genera un CALL P (X, Y) local. Este procedimiento remoto se llama localmente, de forma que las suposiciones normales de dónde encontrar los parámetros, el estado de la pila y otros, son idénticos al caso de una llamada local.

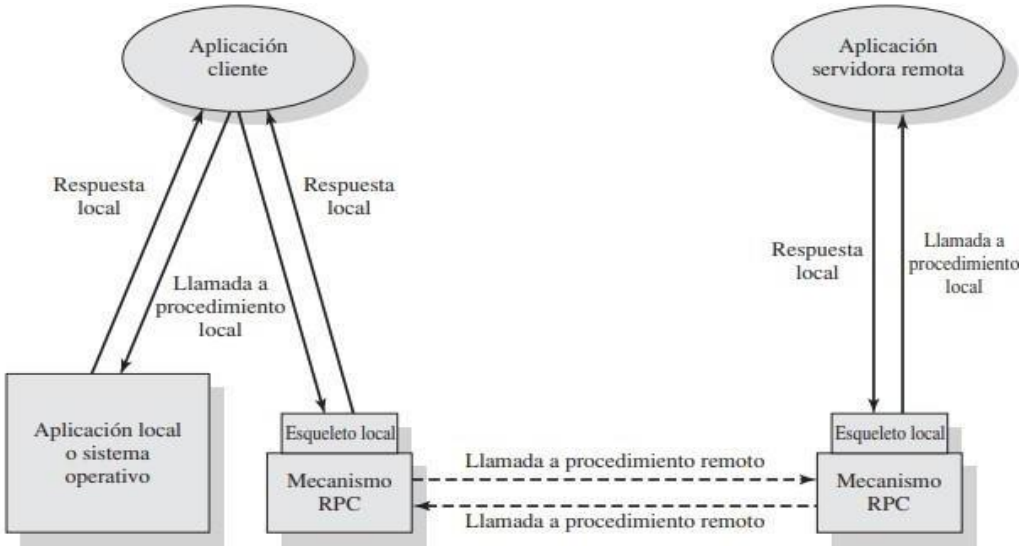

Figura 09. Mecanismo de llamadas a procedimiento remoto

2.3.1 - PASO DE PARÁMETROS

La mayor parte de los lenguajes de programación permiten que los parámetros se pasen como valores (denominado por valor) o como punteros que contienen los valores (llamado por referencia). Las llamadas por valor son fáciles de implementar en las llamadas a procedimiento remoto: los parámetros se copian en el mensaje y se envían al sistema remoto. Es más difícil implementar las llamadas por referencia. Para cada objeto se necesita un puntero único y válido para todo el sistema. No suele merecer la pena el esfuerzo por la sobrecarga que se genera **[4].**

2.3.2 – ENLACE CLIENTE/SERVIDOR

El enlace describe cómo se establecerá la relación entre un procedimiento remoto y el programa que lo invoca. Un enlace se crea cuando dos aplicaciones han establecido una conexión lógica y están listas para intercambiar datos y comandos.

Un enlace temporal implica que la conexión lógica entre los dos procesos se crea en el momento de la llamada al procedimiento remoto y se cierra tan pronto como se devuelven los resultados. Esta conexión requiere mantener información de estado en ambos extremos, lo que consume recursos, por lo que el estilo no permanente se utiliza para ahorrar estos recursos. Sin embargo, debido a la sobrecarga de establecer conexiones repetidamente, este tipo de enlace no es adecuado para procedimientos remotos que se llaman con frecuencia.

En cambio, los enlaces persistentes mantienen la conexión después de la ejecución de la llamada al procedimiento remoto y pueden ser reutilizados para futuras llamadas. Si no hay actividad en la conexión durante un periodo determinado, esta se cierra. Para aplicaciones que requieren invocaciones frecuentes de procedimientos remotos, el enlace permanente conserva la conexión lógica, permitiendo que varias llamadas consecutivas utilicen la misma conexión.

2.3.3 - SÍNCRONOS VS. ASÍNCRONOS

Las llamadas a procedimiento remoto (RPC) tradicionales son de tipo síncrono, lo que significa que el proceso que las invoca debe esperar a que el proceso al que se llama le devuelva un valor. En este sentido, el RPC síncrono se asemeja a una llamada a una subrutina.

El RPC síncrono es sencillo de comprender y programar debido a su comportamiento predecible. Sin embargo, no aprovecha el paralelismo inherente en los sistemas

distribuidos, lo que limita las interacciones de las aplicaciones distribuidas y puede resultar en un rendimiento más bajo.

Para aumentar la flexibilidad, se han desarrollado servicios de RPC asíncronos que permiten un mayor grado de paralelismo sin perder la simplicidad del RPC. En este modelo, las llamadas no bloquean al proceso que las invoca, lo que significa que las respuestas pueden recibirse cuando sea necesario, permitiendo que el cliente y el servidor trabajen en paralelo.
Un caso común de uso del RPC asíncrono es cuando un cliente realiza múltiples invocaciones a un servidor, manteniendo varias respuestas en proceso simultáneamente, cada una con su propio conjunto de datos, optimizando así la ejecución.

2.3.4 - MECANISMOS ORIENTADOS A OBJETOS

A medida que la tecnología orientada a objetos se vuelve más común en el diseño de los sistemas operativos, los diseñadores de la tecnología cliente/servidor han empezado a adoptar este enfoque. En este enfoque, los clientes y los servidores mandan mensajes entre objetos. La comunicación entre objetos se puede basar en una estructura de mensajes o RPC subyacente o puede estar directamente desarrollada sobre los servicios de orientación a objetos del sistema operativo.

Un cliente que necesita un servicio, manda una petición a un mediador de solicitud de objeto (object request broker), que actúa como un directorio de todos los servicios remotos disponibles en la red (Figura 14.10c). El mediador llama al objeto apropiado y le transfiere todos los datos relevantes. A continuación, el objeto remoto atiende la petición y responde al mediador, que devuelve la información al cliente.

2.4 - CLUSTERS

Los *Clusters* son una alternativa al Multiprocesamiento Simétrico, son sistemas que proporcionan un alto rendimiento y una alta disponibilidad y que son particularmente atractivos para aplicaciones de servidor **[7]**

Se enumera cuatro beneficios que se pueden lograr con los *cluster*. Estos beneficios también se pueden ver como objetivos o requisitos de diseño: **[5].**

- Escalabilidad absoluta: Es posible crear un gran *cluster* que supere la potencia de incluso la mayor de las máquinas. Un *cluster* puede tener decenas o incluso centenas de máquinas, cada una de ellas un multiprocesador.
- Escalabilidad incremental: Un *cluster* se configura de tal manera que sea posible añadir nuevos sistemas al *cluster* en pequeños incrementos. De esta forma, un usuario puede comenzar con un sistema pequeño y expandirlo según sus necesidades, sin tener que hacer grandes actualizaciones en las que un pequeño sistema debe ser reemplazado por uno mayor.
- Alta disponibilidad: Ya que cada nodo del *cluster* es una computadora en si´ mismo, el fallo de uno de los nodos no significa perdida del servicio. En muchos productos, el software maneja automáticamente la tolerancia a fallos.
- Relación precio/prestaciones: A través del uso de bloques de construcción es posible hacer un *cluster* con igual o mayor poder computacional que una única máquina mayor, con mucho menor coste.

2.4.1 - CONFIGURACIONES DE LOS CLUSTERS

Uno de los principales desafíos al construir un clúster es identificar y eliminar los puntos de fallo únicos (single points of failure).

Por ejemplo, en un clúster de supercomputación que depende de un servidor central para distribuir las tareas, si ese servidor falla, todo el clúster quedará inoperativo. De manera similar, en un clúster diseñado para balanceo de carga o alta disponibilidad, es crucial asegurar que los servidores continúen funcionando. Sin embargo, si estos servidores están conectados a una red corporativa o a Internet a través de una única interfaz, un fallo en esa interfaz podría aislar todo el sistema.

La redundancia es fundamental para prevenir que la falla de un solo componente de hardware —considerando que un clúster está compuesto por numerosos elementos, aumentando la probabilidad de fallos— afecte la funcionalidad del sistema en su totalidad.

Además, es esencial elegir la tecnología adecuada según nuestras necesidades. Por ejemplo, mantener un clúster en una red Ethernet de 10 Mb puede ser una buena opción

si hay pocos nodos, pero al agregar más, la red puede convertirse en un cuello de botella, haciendo que los servidores queden inactivos mientras esperan los datos durante períodos prolongados.

Los clústeres se pueden clasificar de diversas maneras, siendo la más simple aquella que se basa en si las computadoras del clúster comparten acceso a los discos.

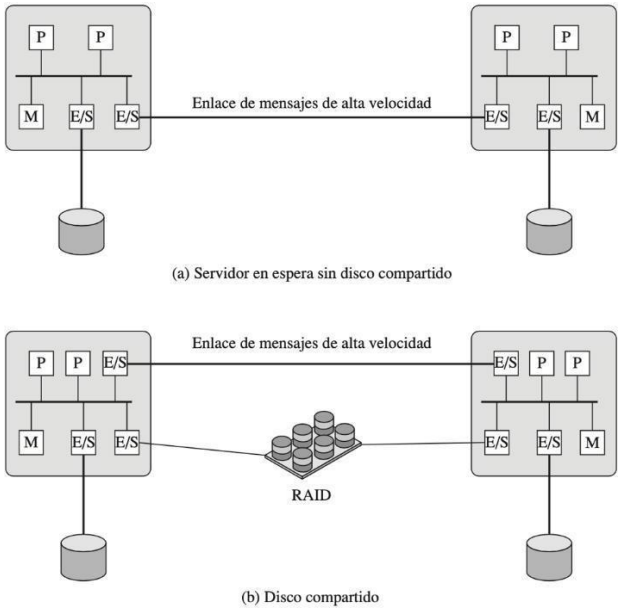

Figura 10. Configuraciones de cluster.

2.4.2 - ASPECTOS DE DISEÑO DE SISTEMAS OPERATIVOS

Para obtener todas las ventajas de una configuración hardware de un *cluster* se necesitan algunas mejoras en los sistemas operativos.

Gestión de Fallos. En general, para el tratamiento de los fallos se pueden seguir dos enfoques: *clusters* con alta disponibilidad y *clusters* con tolerancia a fallos.

Un *cluster* de alta disponibilidad ofrece alta posibilidad de que todos los recursos estén en servicio. Si sucede algún fallo, tal como la caída de un nodo o que se pierda un volumen, se pierden las peticiones en progreso. Cualquier petición perdida que se reintente, será atendida por una computadora diferente del *cluster*. Sin embargo, el

sistema operativo del *cluster* no garantiza el estado de las transacciones parcialmente realizadas. Esto se necesita gestionar a nivel de aplicación.

Un *cluster* tolerante a fallos asegura que todos los recursos están siempre disponibles. Esto se logra a través del uso de discos redundantes compartidos y mecanismos para deshacer transacciones incompletas.

La función de intercambiar una aplicación y los datos de un sistema fallido por un sistema alter-nativo del *cluster* se denomina recuperación de fallos (*failover*). La restauración de aplicaciones y de datos al sistema original una vez que se ha reparado, se denomina restauración de fallos (*failback*). La restauración puede ser automática, pero esto sólo es deseable si el problema está realmente solucionado y es poco probable que vuelva a suceder. De otra forma, la restauración automática puede provocar fallos sucesivos en el sistema inicial, generando problemas de rendimiento y de recuperación.

Equilibrado de carga. Un *cluster* necesita tener la capacidad de equilibrar la carga entre todas las computadoras disponibles. Esto incluye el requisito de que un *cluster* debe ser escalable. Cuando se añade una nueva computadora al *cluster*, el servicio de equilibrado de carga debe incluir automáticamente la nueva computadora en la planificación de las aplicaciones. Los mecanismos del *middleware* deben saber que pueden aparecer nuevos servicios en diferentes miembros del *cluster*, pudiendo migrar de un miembro a otro.

Computación paralela. En algunos casos, el uso eficiente de los *cluster* necesita ejecutar una única aplicación en paralelo.

Compilación paralela**.** Un compilador paralelo determina, en tiempo de compilación, qué partes de la aplicación se pueden ejecutar en paralelo. Estas partes se pueden asignar a computadoras diferentes del *cluster*. El rendimiento depende de la naturaleza del problema y de lo bueno que sea el diseño del compilador.

- Aplicaciones paralelas**.** En este enfoque, el programador escribe la aplicación para que ejecute en un *cluster* y utiliza paso de mensajes para mover datos, según se requiera, entre nodos del *cluster*. Esto supone una gran carga para el programador, pero probablemente es la mejor forma de explotar los *cluster* para algunas aplicaciones.

• Computación paramétrica**.** Este enfoque se puede utilizar si la esencia de la aplicación es un algoritmo que se debe ejecutar un gran número de veces, cada vez con un conjunto diferente de condiciones o parámetros iniciales. Un buen ejemplo es un modelo de simulación, que ejecutará un gran número de diferentes escenarios y luego calculará estadísticas de los resultados. Para que este enfoque sea efectivo, se necesitan herramientas de procesamiento paramétricas para organizar, ejecutar y gestionar los trabajos de forma ordenada.

2.4.3 – ARQUITECTURA DE UN CLÚSTER

Cada computadora es capaz de operar independientemente, además, en cada computadora está instalada una capa de software middleware que permite la operación del *cluster*. El middleware del *cluster* proporciona una imagen única al usuario, conocida como **imagen única del sistema** (*single-system image*). El middleware también podría´ ser responsable de proporcionar alta disponibilidad, a través de balanceado de carga y de la respuesta a los fallos de los componentes. Enumera los siguientes servicios y funciones deseables en un *cluster*: **[8]**

- **Un único punto de entrada.** Un usuario se autentifica en el *cluster* y no en una determinada computadora.
- **Una única jerarquía de ficheros.** Los usuarios ven una sola jerarquía de directorios bajo el mismo directorio raíz.
- **Un único punto de control.** Hay un nodo por defecto encargado de gestionar y controlar el *cluster*.
- **Una única red virtual.** Cualquier nodo puede acceder a cualquier otro punto del *cluster*, incluso si la configuración del *cluster* tienen múltiples redes interconectadas. Se opera sobre una única red virtual.
- **Un único espacio de memoria.** La memoria compartida distribuida permite a los programas compartir variables.
- **Un único sistema de control de trabajos.** Con un planificador de trabajos en el *cluster*, un usuario puede enviar su trabajo sin especificar la computadora que lo ejecutará.

- **Una única interfaz de usuario.** Todos los usuarios tienen un interfaz grafico común, independientemente de la estación de trabajo que utilicen.
- **Un único espacio de E/S.** Cualquier nodo puede acceder remotamente a cualquier periférico de E/S o disco, sin conocer su localización física.
- **Un único espacio de procesos.** Se utiliza un esquema uniforme de identificación de procesos. Un proceso en cualquier nodo puede crear o se puede comunicar con cualquier otro proceso en un nodo remoto.
- **Puntos de control.** Esta función salva periódicamente el estado del proceso y los resultados de computación intermedios, para permitir recuperarse después de un fallo.
- **Migración de procesos.** Esta función permite el balanceo de carga.

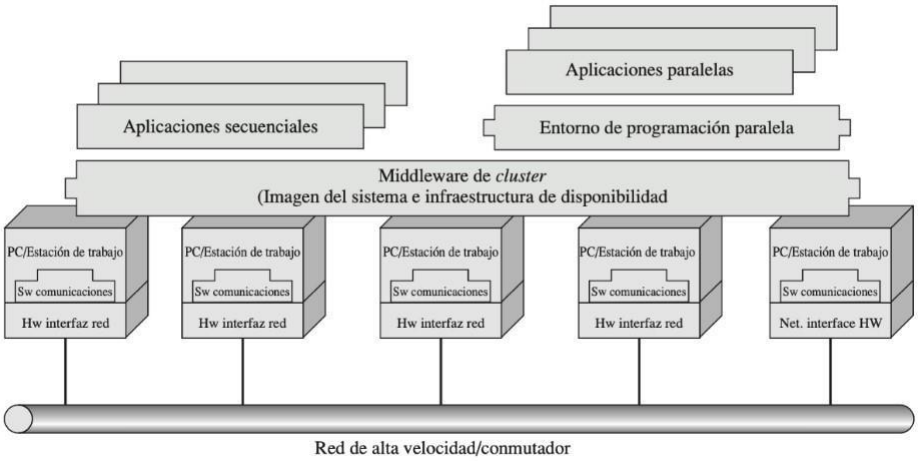

Figura 11. Arquitectura de computación cluster (BUY99a)

2.4.4 - *CLUSTER* FRENTE A SMP

Tanto los *clusters* como el multiprocesamiento simétrico proporcionan una configuración con múltiples procesadores para dar soporte a aplicaciones con alta demanda. Ambas soluciones están disponibles en el mercado, aunque SMP ha estado presente durante más tiempo.

La principal fuerza del enfoque SMP es que es más fácil de gestionar y configurar que un *cluster*. El SMP está mucho más cercano al modelo original de un solo procesador para los que están escritas prácticamente todas las aplicaciones. El principal cambio requerido para pasar de un uniprocesador a un multiprocesador es la función de planificación. Otro beneficio del SMP es que normalmente ocupa menos espacio físico y gasta menos energía que un´ *cluster* comparable. Por último, un beneficio importante es que los productos SMP están bien establecidos y son muy estables.

A largo plazo, sin embargo, las ventajas del *cluster* probablemente le llevarán a dominar el mercado de servidores de alto rendimiento. Los *clusters* son mucho más superiores que SMP en relación a la escalabilidad incremental y absoluta. Los *clusters* son también superiores en términos de disponibilidad, porque todos los componentes del sistema pueden ser altamente redundantes.

2.5 – SERVIDOR CLUSTER DE WINDOWS

El Servidor *Cluster* de Windows es un *cluster* no compartido, en que cada volumen de disco y otros recursos son propiedad de un único sistema a la vez.

El diseño del Servidor hace uso de los siguientes conceptos **[6].**

- Servicio *Cluster***.** Colección de software de cada nodo que gestiona actividades específicas del *cluster*.
- Recurso**.** Elemento gestionado por el servicio *cluster*. Son objetos que representan recursos reales en el sistema, incluyendo dispositivos hardware tales como discos o tarjetas de red y elementos lógicos tales como volúmenes lógicos de disco, direcciones TCP/IP etc.
- En línea (*online*). Un recurso está en línea en un nodo cuando está proporcionando servicio en ese nodo específico.

- Grupo**.** Colección de recursos gestionada como una unidad. Contiene todos los elementos necesarios para ejecutar una aplicación y para que los sistemas cliente se conecten al servicio proporcionado por esta aplicación.

Un grupo combina recursos en unidades mayores, tanto para la recuperación de fallos como para el balanceado de carga. Las operaciones realizadas en un grupo, tales como transferir el grupo a otro nodo, afectan a todos los recursos de ese grupo. Se implementan como bibliotecas dinámicas (DLL) y se gestionan por un monitor de recursos. El monitor de recursos interactúa con el servicio *cluster* a través de llamadas a procedimiento remoto y responde a los comandos del servicio *cluster* para configurar y mover grupos de recursos.

El **gestor de nodo** es responsable de mantener la pertenencia de este nodo al *cluster*. Periódicamente, manda mensajes a los gestores de nodo del resto. Cuando que un gestor de nodo detecte la pérdida de mensajes, difunde un mensaje a todo el *cluster*, haciendo que todos los miembros intercambien mensajes para verificar su estado. Si uno no responde, se le quita del *cluster* y sus grupos activos se transfieren a uno o más nodos activos del *cluster*.

El **gestor de la base de datos de configuración** guarda la base de datos de configuración del *cluster*. Contiene información de recursos, grupos, y pertenencia de grupos a nodos. Los gestores de base de datos de cada uno de los nodos del *cluster* cooperan para mantener la información de configuración. Para asegurar que los cambios, se utiliza software de transacciones tolerante a fallos.

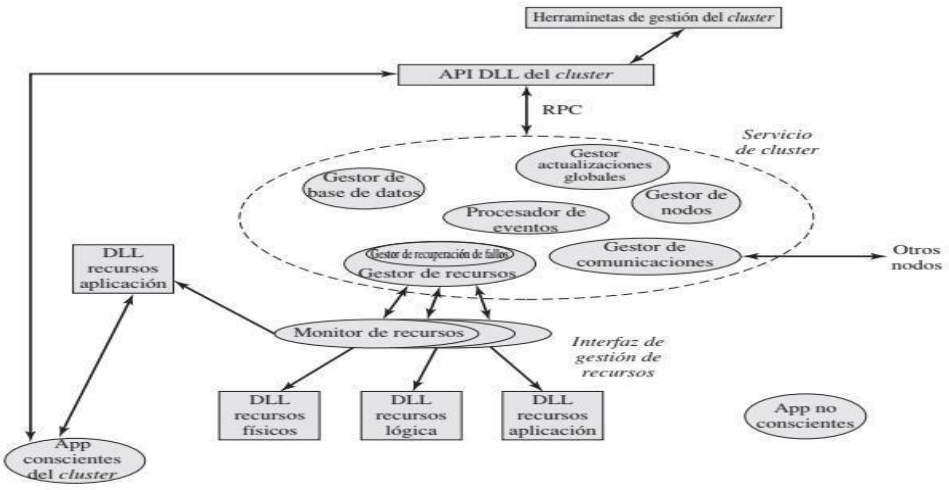

Figura 12. Diagrama de bloques del Windows Cluster Server (SHO97).

El gestor de recursos/gestor de recuperación de fallos toma las decisiones a los grupos de recursos e inicia acciones apropiadas tales como inicializar, reinicializar y recuperar fallos.

Cuando se requiere recuperar un fallo, el gestor de recuperación coopera para negociar una distribución de los grupos de recursos que ha fallado en el resto de los sistemas activos. Cuando un sistema se reinicia, el gestor de recuperación de fallos puede decidir hacer regresar algunos grupos a este sistema. Se puede configurar cualquier grupo con un propietario. Si ese propietario falla y luego se reinicia, el grupo se vuelve a llevar al nodo.

El procesador de eventos conecta todos los componentes del servicio, maneja operaciones comunes y controla la inicialización. El gestor de comunicaciones gestiona el intercambio de mensajes con el resto de los nodos. El gestor global de actualizaciones proporciona un servicio utilizado por otros componentes del cluster.

2.6 - SUN CLUSTER

Sun Cluster es un sistema operativo distribuido, construido como un conjunto de extensiones del sistema UNIX Solaris. Proporciona que, los usuarios y las aplicaciones ven al cluster como una única computadora ejecutando el sistema operativo Solaris.

Los principales componentes son:

- Soporte de objetos y comunicaciones.
- Gestión de procesos.
- Redes.
- Sistema de ficheros distribuido global.

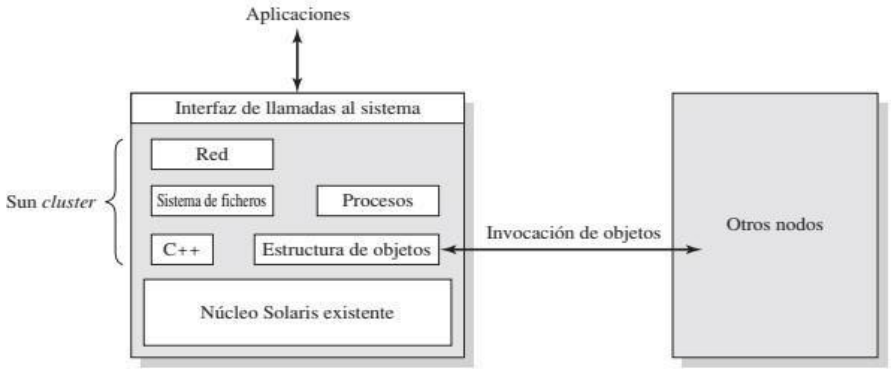

Figura 13. Estructura de Cluster

2.6.1 - SOPORTE DE OBJETOS Y COMUNICACIONES

La implementación está orientada a objetos. Se utiliza el modelo de objetos de CORBA para definir los objetos y las llamadas a procedimiento remoto (RPC). Se utiliza el Lenguaje de Definición de Interfaces (IDL) de CORBA para especificar los interfaces entre los componentes MC de los diferentes nodos. Los objetos de MC se implementan en C++. El uso de un modelo de objetos y de IDL, proporciona una comunicación entre los procesos, tanto dentro de un nodo como entre ellos.

2.6.2 - GESTIÓN DE PROCESOS

Extiende las operaciones de los procesos de forma que la localización de un proceso es transparente al usuario. Sun Cluster mantiene una visión global de los procesos de forma que en el cluster hay un identificador único por proceso y que cada nodo puede saber la localización y estado de cada proceso. Un proceso se puede mover de un nodo a otro, con el objetivo de lograr un balanceado de carga y de recuperar fallos. Sin embargo, todos los hilos de un proceso deben estar en el mismo nodo.

2.6.3 – REDES

Los diseñadores de Sun Cluster consideraron tres enfoques para el manejo del tráfico de red:

1. Realizar todo el procesamiento del protocolo de red en un único nodo. Para una aplicación en TCP/IP, el tráfico entrante podría ir a través de un nodo. Para el tráfico entrante el nodo analizaría las cabeceras TCP/IP y mandaría los datos encapsulados al nodo apropiado. Para el tráfico saliente, el nodo encapsularía los datos de otros nodos con cabeceras TCP/IP.
2. Asignar una dirección única y ejecutar los protocolos de red en cada nodo. Un problema es que la configuración del cluster deja de ser transparente. Otra es la dificultad de la recuperación de fallos cuando una aplicación en ejecución se mueve a otro nodo con una dirección de red diferente.
3. Utilizar un filtro de paquetes para enviar los paquetes al nodo apropiado y realizar el procesamiento del protocolo en dicho nodo. Las conexiones entrantes (peticiones de clientes), se balancean entre todos los nodos disponibles del cluster.

El subsistema de red de Sun Cluster tiene tres elementos principales:

i. Los paquetes entrantes se reciben en el nodo receptor que filtra el paquete y lo envía al nodo objetivo correcto usando la interconexión del cluster.
ii. Todos los paquetes salientes se envían a través de la interconexión del cluster al que tiene la conexión física de red externa. Todo el procesamiento del protocolo de los paquetes salientes se realiza en el nodo origen.
iii. Se mantiene una base de datos de configuración de red para anotar el tráfico de red de cada nodo.

2.6.4 - SISTEMA DE FICHEROS GLOBAL

El elemento más importante de Sun Cluster es el sistema de ficheros global, que compara la gestión de ficheros de MC con el esquema básico de Solaris. Ambos se basan en el uso de los conceptos nodo-v y sistema de ficheros virtual.

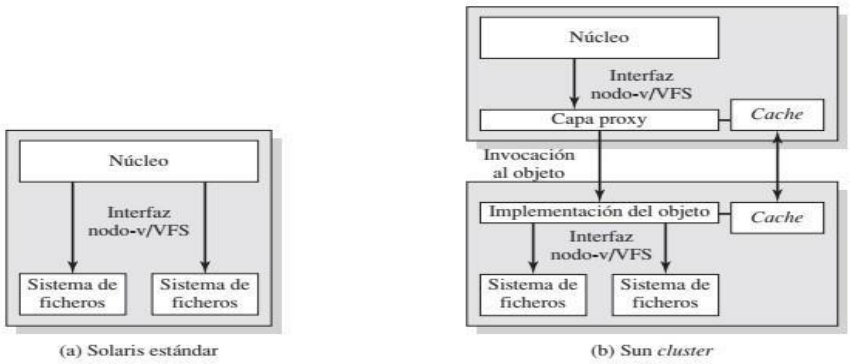

Figura 14. Extensiones del sistema de fichero de Sun Cluster.

En Solaris, la estructura de nodo virtual (nodo-v) se usa para ofrecer una interfaz común para todos los tipos de sistemas de archivos. El nodo-v asocia páginas de memoria con el espacio de direcciones de un proceso y autoriza el acceso a un sistema de archivos. A diferencia de los nodos-i, que vinculan procesos con archivos UNIX, los nodo-v pueden conectar un proceso con un objeto de cualquier sistema de archivos. Así, una llamada al sistema solo necesita saber cómo utilizar la interfaz del nodo-v, sin importar el objeto que maneje.

Esta interfaz acepta comandos como leer o escribir, y los convierte en acciones específicas del sistema de archivos. Los nodo-v describen objetos individuales, mientras que las estructuras de sistema de archivos virtual describen el sistema completo.

En Sun Cluster, el sistema de archivos global proporciona una interfaz unificada para archivos distribuidos en el cluster. Un proceso puede acceder a un archivo en cualquier parte del cluster usando la misma ruta. Para esto, se emplea un sistema de archivos proxy sobre el existente en Solaris usando la interfaz nodo-v, permitiendo que los objetos invocados residan en cualquier nodo. Este entorno no requiere modificar el núcleo ni el sistema de archivos base. Además, se utiliza una caché para reducir invocaciones remotas, almacenando contenido de archivos, información de directorios y atributos.

2.7 - CLUSTERS BEOWULF Y LINUX

En 1994 se inició el proyecto Beowulf con el patrocinio del proyecto de la NASA High Performance Computing and Communications (HPCC)

2.7.1 - CARACTERÍSTICAS DE BEOWULF

Las principales características de Beowulf incluyen las siguientes [RIDG97]:

- Componentes genéricos disponibles en el mercado.
- Procesadores dedicados (mejor que ciclos disponibles de estaciones de trabajo ociosas).
- Una red privada y dedicada (LAN o WAN o una combinación de redes).
- Ningún componente propio.
- Fácilmente replicable para múltiples vendedores.
- E/S escalable.
- Basado en software gratuito disponible.
- Utiliza herramientas de computación gratuitas con mínimos cambios.
- Retorno del diseño y de las mejoras a la comunidad.

Aunque los elementos software de Beowulf se han implementado en diversas plataformas, la elección más obvia se basa en Linux, y la mayor parte de las implementaciones Beowulf utiliza un cluster de estaciones de trabajo Linux sobre PCs. El cluster tiene una serie de estaciones de trabajo, posiblemente con diferentes plataformas hardware, ejecutando el sistema operativo Linux. El almacenamiento secundario de cada estación de trabajo puede estar disponible para acceso distribuido (para compartición de ficheros distribuida, memoria virtual distribuida y otros usos). Los nodos del cluster (los sistemas Linux) se interconectan con una red, normalmente Ethernet. La Ethernet puede estar basada en un solo conmutador (switch) o en un conjunto de conmutadores interconectados. Se utilizan productos Ethernet con velocidades estándar (10 Mbps, 100Mbps y 1 Gbps).

2.7.2 - BEOWULF SOFTWARE

El software Beowulf es una extensión de distribuciones Linux gratuitas. Cada nodo del cluster ejecuta su propio núcleo de Linux y puede funcionar de forma autónoma, pero se realizan modificaciones para permitir a los nodos compartir espacios de nombres globales.

Ejemplos:

- BPROC: Expande el espacio de procesos a múltiples nodos y permite iniciar procesos en nodos remotos, dando una visión unificada del cluster.
- Unión de canales Ethernet: Combina múltiples redes Ethernet en una sola, mejorando el ancho de banda y balanceando la carga.
- Pvmsync: Proporciona sincronización y datos compartidos para procesos en el cluster.
- EnFuzion: Ejecuta trabajos paramétricos con diferentes parámetros en el cluster.

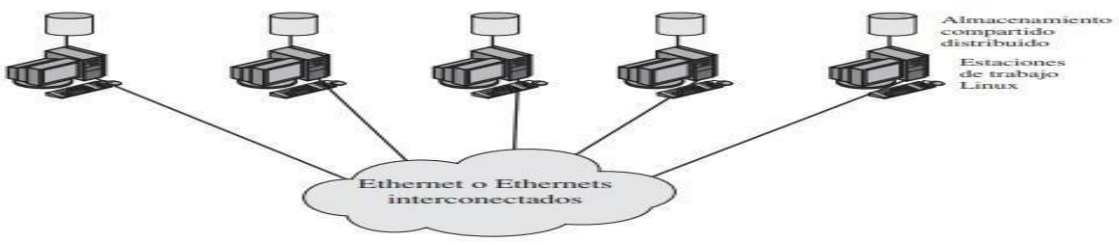

Figura 15. Configuración genérica de Beowulf.

3 – CONCLUSION

En resumen, la arquitectura cliente/servidor, junto con la computación distribuida y el uso de clústers, se ha convertido en un pilar fundamental para el desarrollo de sistemas de información eficientes y escalables.

La computación cliente/servidor es esencial para aprovechar al máximo los recursos de una red y mejorar la productividad organizacional, ya que distribuye las aplicaciones entre los clientes y los servidores, facilitando el acceso eficiente a recursos compartidos. Los clientes interactúan a través de interfaces gráficas sencillas, mientras que los servidores gestionan tareas clave, como bases de datos, dividiendo las responsabilidades para optimizar el rendimiento. En los sistemas distribuidos, la comunicación entre procesos es clave, utilizando técnicas como el paso de mensajes y las llamadas a procedimientos remotos para permitir la interacción entre programas en distintas máquinas como si estuvieran en la misma. Además, los clusters de computadoras unifican múltiples sistemas en una sola entidad, proporcionando un entorno de procesamiento más potente y flexible, combinando la descentralización con la gestión centralizada de los recursos.

4 – REFERENCIA BIBLIOGRAFICA

- **[1].** Oracle, "Overview of SQL Statements," *Oracle Database SQL Language Reference 11g Release 1 (11.1)*, https://docs.oracle.com/cd/B28359_01/server.111/b28286/intro002.htm#SQLRF50928.
- **[2]** Sistemas Operativos: Aspectos internos y principios de diseño Quinta EdiciónWILLIAM STALLING - Capítulo 14 páginas 625, 626.
- **[3]** Sistemas Operativos: Aspectos internos y principios de diseño Quinta EdiciónWILLIAM STALLING - Capítulo 14 página 630
- **[4]** Gibbons, P. «A Stub Generator for Multilanguage RPC in Heterogeneous Environments.» IEEE Transactions on Software Engineering. Enero 1987.
- **[5]** Brewer, E. «Clustering: Multiply and Conquer.» Data Communications, Julio1997.
- **[6]** Short, R.; Gamache, R.; Vert, J. y Massa, M. «Windows NT Clusters for Availability and Scalability.»
- **[7]** Sistemas Operativos: Aspectos internos y principios de diseño Quinta EdiciónWILLIAM STALLING - Capítulo 14 página 636.

- **[8]** Hwang, K, et al. «Designing SSI Clusters with Hierarchical Checkpointing andSingle I/O Space.» IEEE Concurrency, enero-marzo 1999.
- **[9]** Sistemas Operativos: Aspectos internos y principios de diseño Quinta EdiciónWILLIAM STALLING - Capítulo 14 página 632.

Universidad Nacional del Nordeste

Facultad de Ciencias Exactas y Naturales y Agrimensura

Sistemas Operativos

Grupo N.º 5
Alumnos:

Gomez Victor Agustín	DNI: 40047803	LU: 55262
Peñaloza Raúl Facundo	DNI: 42251232	LU: 54448
Torreani Cáceres Jimena Soraya	DNI: 37809444	LU: 48353
Riquelme Rodolfo Iván	DNI: 39634606	LU: 51776
Romero María Cecilia	DNI: 36469060	LU: 48454
Ruiz Diaz Emmanuel	DNI: 39318061	LU: 50383

Tema: Seguridad

Índice de contenidos

Contenido

Índice de Figuras

Figuras

Índice de Tablas

Tablas

Introducción

Cada día, más y más personas mal intencionadas intenta tener acceso a datos de nuestros ordenadores, el acceso no autorizado a una red informática o a los equipos que en ella se encuentran pueden ocasionar en la gran mayoría de los casos graves problemas.

Con la constante evolución de las computadoras y los ataques en sí, es fundamental saber que recursos necesitar para obtener seguridad en los sistemas.

En el presente informe abordaremos los tipos de amenazas más comunes, manera de proteger nuestro sistema operativo, los distintos tipos de técnicas que utilizan los intrusos para acceder a nuestra información, los distintos tipos de software maliciosos más comunes y a que se conoce como sistema confiable.

1.- ¿Que es seguridad?

Se entenderá por seguridad, a los problemas generales relativos a la garantía de que los archivos no sean leídos, o modificados por personal no autorizado; esto incluye aspectos técnicos de administración, legales y políticos. [1]

Se considerarán mecanismos de protección a los mecanismos específicos del sistema operativo utilizados para resguardar la información de la computadora, la frontera entre seguridad y mecanismos de protección no está bien definida. [1]

La seguridad tiene múltiples usos. En términos generales, se puede decir que este concepto proviene del latín “securitas” se centra en la propiedad de seguro, es decir, mejorar la propiedad de algo donde hay peligro, daño o la comprobación de riesgos. Una cosa es segura es algo fuerte, cierta e indudable. La seguridad, por lo tanto, puede ser considerada como una certeza. [2]

2.-Amenazas

Requisitos que necesitamos para la seguridad: [3]

-Confidencialidad: La información debe estar disponible para personas autorizadas solamente

-Integridad: Solo podrán modificar los contenidos de un sistema las personas autorizadas.

-Disponibilidad: Los componentes deberán estar disponibles para las personas autorizadas.

-Autenticación: Debe poder verificar la identidad de los usuarios.

2.1-Tipos de peligros

Existen 4 grandes categorías de tipos de ataques: [4]

-Interrupción: Se destruye un componente del sistema o se encuentra no disponible o utilizable. Se centra en la disponibilidad.

-Intercepción: Una parte no autorizada consiga acceso a un componente. Se centra en la confidencialidad

-Modificación: Un elemento no autorizado no solo tiene acceso a un elemento, sino que también puede modificarlo. Se centra en la integridad.

-Fabricación: Un elemento no autorizado inserta objetos extraños en el sistema. Se centra en la autenticación.

2.2-Componentes del Sistema

Se clasifican en varios tipos: [4]

-Hardware: Es el componente más vulnerable y el menos accesible a una manipulación remota, sobre todo en el área de la disponibilidad.

-Software: Lo que hace del hardware del sistema algo útil para negocios e individuos son el sistema operativo, las utilidades y los programas de aplicación. Existen diferentes tipos de peligros a tener en cuenta.

Un peligro importante en relación al software es el referente a la disponibilidad. Es fácil de borrar, alterar o dañar. Esto podría solventarse con el uso de copias de respaldo y actualización del software. Un cambio en el comportamiento del software implica un peligro a la integridad y autenticación. El ultimo problema es el relativo a la privacidad. [6]

Tabla 1 Peligros de seguridad y componentes

	Disponibilidad	Privacidad	Integridad/Autenticación
Hardware	Equipamiento robado o deshabitado, por lo tanto denegación de servicio.		
Software	Borrado de programas, denegación de acceso a los usuarios.	Copia no autorizada de software.	Modificación de un programa, bien para hacer que falle durante la ejecución o para que realice una tarea diferente.
Datos	Borrar ficheros, denegación de acceso a los usuarios.	Lectura no autorizada de datos. Un análisis estadístico de los datos que revele la información subyacente.	Modificación de los ficheros existentes o creación de nuevos ficheros.
Líneas de comunicación	Borrado o destrucción de mensajes. Las líneas de comunicación o redes no se encuentran disponibles.	Lectura de mensajes. Observación de los patrones de tráfico de mensajes.	Modificación, borrado, reordenación o duplicación de mensajes. Fabricación de mensajes falsos.

-Datos: Un problema amplio es el relativo a la seguridad de los datos, que incluye ficheros o cualquier tipo de datos controlados por individuos, grupos u organizaciones. Incluyéndola disponibilidad, privacidad e integridad. En el caso de la disponibilidad, se centra en la destrucción de ficheros de datos, de forma accidental o maliciosa.

Un aspecto evidente en el concerniente a la privacidad es la lectura no autorizada de datos o bases de datos,

La integridad de datos es un aspecto clave en muchas instalaciones. La modificación de los ficheros de datos puede llevar a una serie de consecuencias que van desde problemas menores hasta desastrosos. [6]

-Líneas de comunicaciones y redes:

En este caso un ataque pasivo intenta aprender o hacer uso de la información del sistema, por ejemplo, el espionaje o la monitorización de transmisiones. El objetivo es obtener información que se está transmitiendo. Como la lectura de los contenidos de los mensajes y el análisis de tráfico. [7]

Los ataques activos implican algunas modificaciones en el flujo de datos o la creación de datos falsos, se pueden dividir en, enmascaramiento, reenvío, modificación de mensajes y denegación del servicio. [7]

El enmascaramiento ocurre cuando el elemente intenta hacerse de pasar por otro diferente, El reenvío implica la captura pasiva de una unidad de datos y su posterior retransmisión para producir un efecto no autorizado. La modificación de mensajes significa sencillamente que una parte de un mensaje valido se ha alterado o que los mensajes se han borrado y reordenada, para producir un efecto no autorizado. La

denegación del servicio previene o imposibilita el uso normal o la gestión de instalaciones de comuniones. [7]

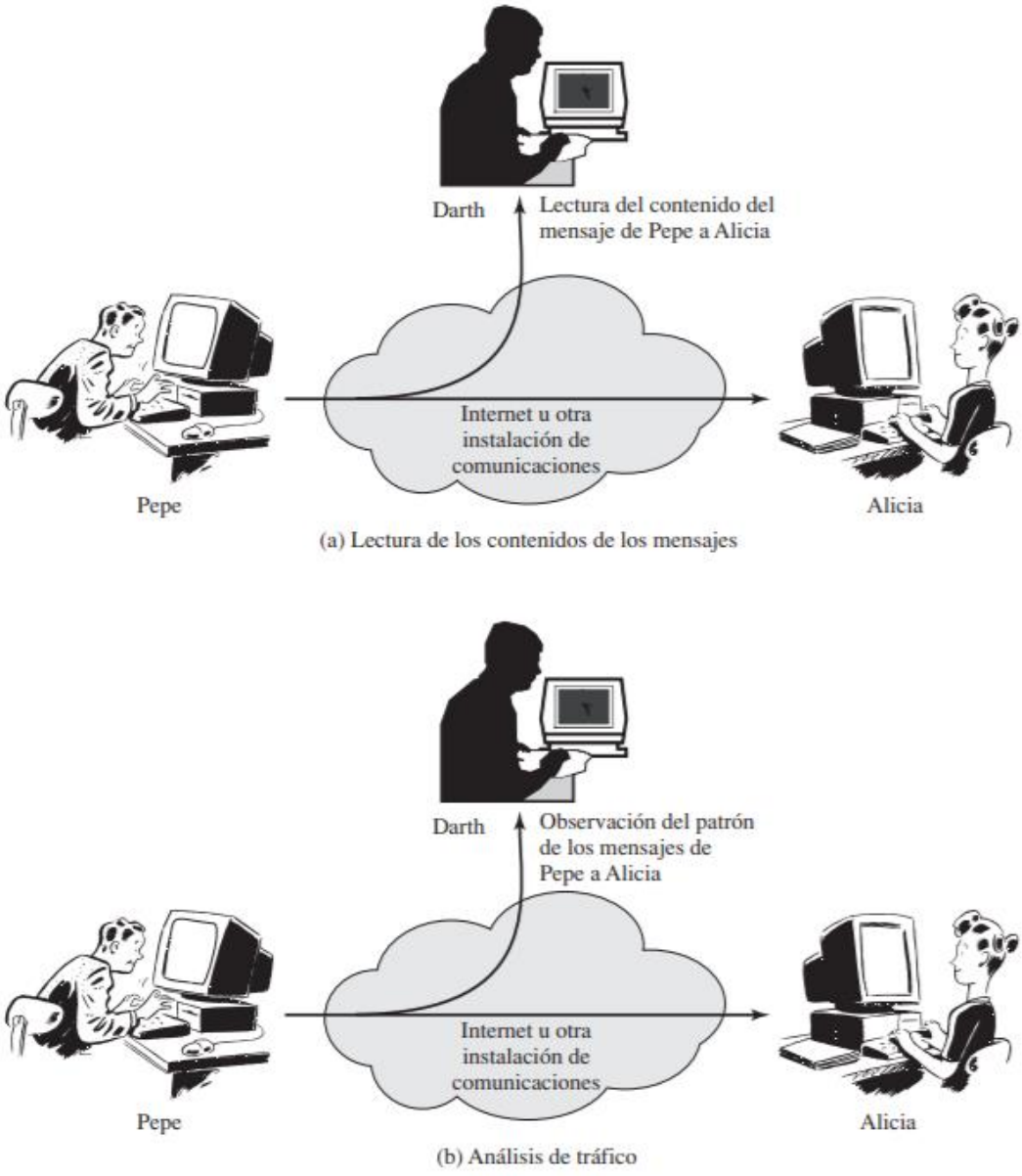

Figura 1. Ataques pasivos

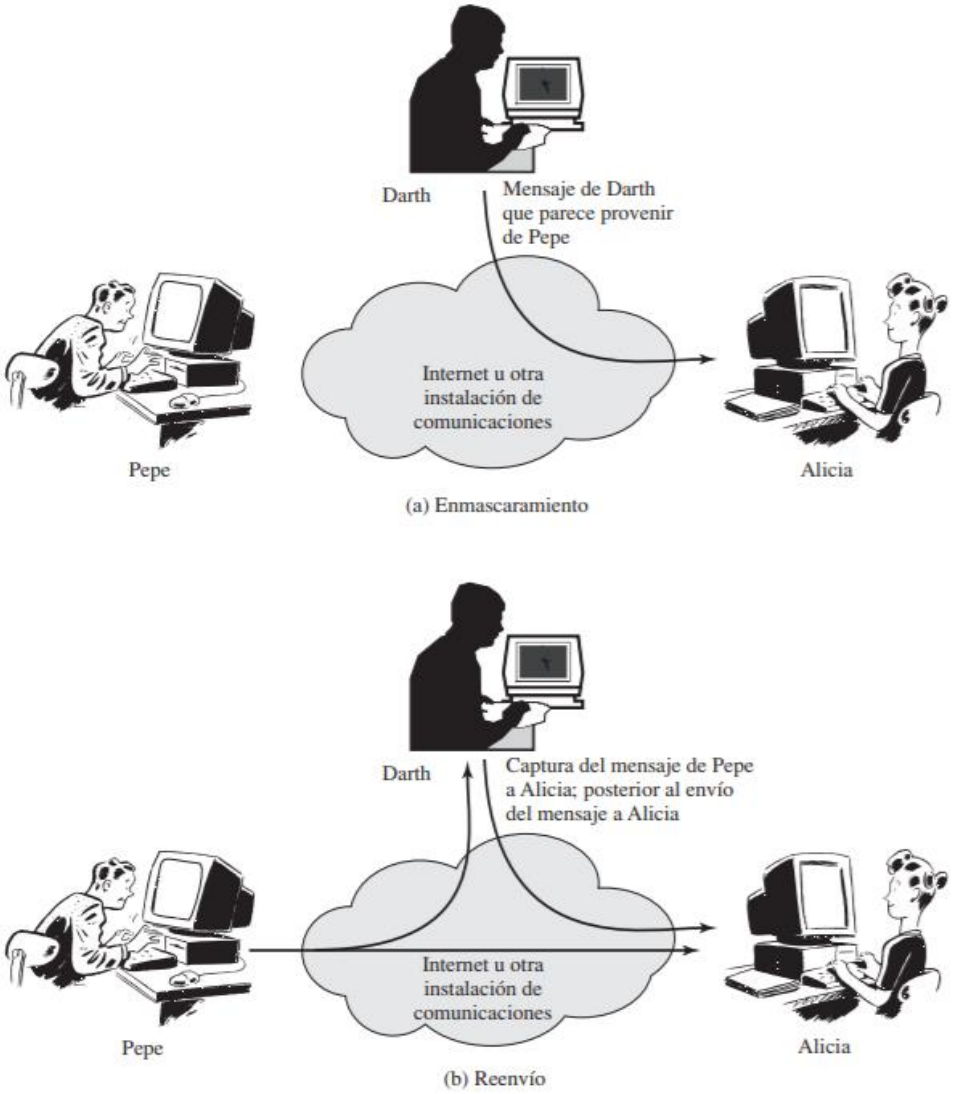

Figura 2. Ataques activos

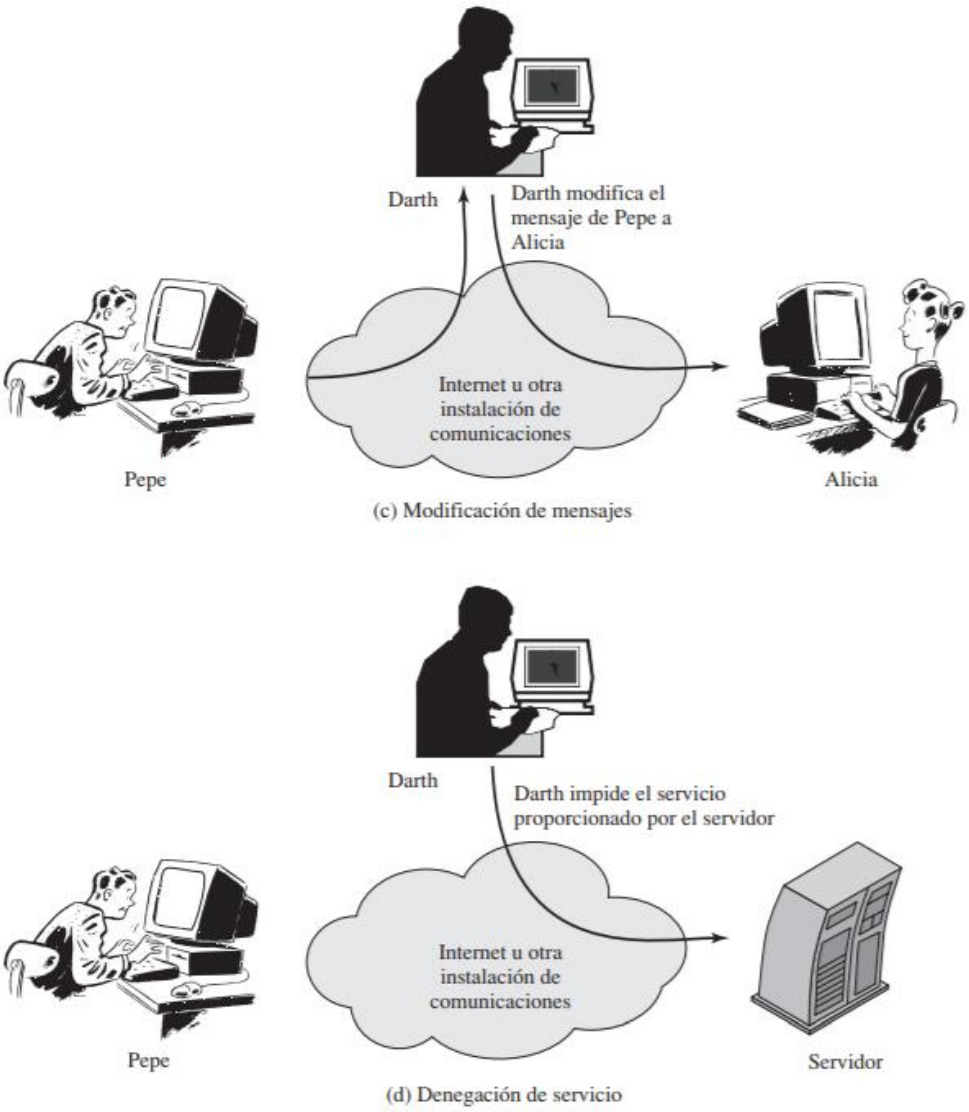

Figura 3. Continuación

3.-Proteccion

La posibilidad de compartir recursos, respecto de la multiprogramación, introduce la necesidad de protección, El sistema operativo deberá ofrecer niveles de protección a lo largo del siguiente rango: [8]

-Sin protección alguna: Apropiado por los procedimientos que son sensibles de ejecutar en instantes diferentes.

-Aislamiento: Cada proceso opera de forma separada con otros procesos, sin compartición ni comunicación.

-Compartición completa o sin compartición: El propietario del objeto declara si va a ser público o privado.

-Compartición vía limitaciones de acceso: El sistema operativo verifica la permisibilidad de cada acceso por parte de cada usuario sobre cada objeto.

-Acceso vía capacidades dinámicas: Permite la creación dinámica de derechos de acceso a los objetos.
-Uso limitado de un objeto: Esta forma de protección limita no solo el acceso a un objeto sino también el uso que se puede realizar a dicho objeto

3.1-Proteccion de la memoria

La separación de los espacios de memoria de los diferentes procesos es un requisito fundamental de los esquemas de memoria virtual. Ya sea en base a la segmentación o a la paginación o a la combinación de ambas, es necesario proporcionar unos mecanismos efectivos para gestionar la memoria principal. Si lo que se busca es el aislamiento completo, entonces el sistema operativo únicamente debe asegurar que a cada segmento o página se accede sólo por parte del proceso al que está asignada. [9]

Éste es un requisito fácil de alcanzar, asegurándose únicamente de que no hay entradas duplicadas en las tablas de páginas y/o segmentos. Si se desea permitir la compartición, entonces el mismo segmento o página puede aparecer en más de una tabla. Ese tipo de compartición es más fácil de alcanzar en un sistema que soporte segmentación o la combinación de segmentación y paginación. En este caso, la estructura de segmentos es visible para la aplicación, y es la propia aplicación la que puede declarar que determinados segmentos se encuentren compartidos o no. En un entorno de paginación, por tanto, resulta más difícil diferenciar entre estos dos tipos de memoria, debido a que la estructura de memoria es completamente transparente a la aplicación. [9]

3.2-Control de acceso orientado a usuarios

La técnica más común para el control de acceso por usuario en un sistema compartido o en un servidor es el registro o conexión por usuario, se requiere un ID de usuario y contraseña. Este sistema es un método notablemente poco fiable para proporcionar control de acceso a los usuarios. Los usuarios pueden olvidar sus contraseñas y accidentalmente o de forma intencionada revelarlas. Los hackers han demostrado ser especialmente habilidosos en adivinar identificadores y contraseñas de usuarios especiales. [10]

El control de acceso de usuarios en un entorno distribuido puede ser centralizado o descentralizado. De una forma centralizada, la red proporciona un servicio de conexión, que determina a quien esta permitido el uso de la red y con quien le esta permitido conectarse. De una forma descentralizada, trata a la red como u enlace de comunicación transparente, y el mecanismo de acceso habitual se realiza por parte del ordenador destino. O también en muchas redes se puede utilizar el control de dos niveles. Los ordenadores de forma particular pueden proporcionar un servicio de conexión y adicionalmente la red en su conjunto proporciona una protección de acceso restringido únicamente a usuarios autorizados. [10]

3.3-Control de acceso orientado a datos

Una vez el usuario se conecta a la red de datos, asociado a cada usuario, puede existir un perfil que especifica las operaciones permitidas en los accesos a ficheros. El sistema operativo puede aplicar reglas que se basen en los perfiles de usuario. Esta decisión o dependerá únicamente de la identidad del usuario sino también de una parte especifica de los datos a los cuales se va a acceder o incluso de la información ya divulgada al usuario. [10]

Un modelo general para el control de acceso aplicado por un sistema de gestión de baso datos o un sistema de ficheros es el denominado matriz de acceso. Los elementos básicos del modelo son los siguientes: [11]

-Sujeto: Un elemento capaza de acceder a objetos.

-Objeto: Todo elemento sobre el que se accede de forma controlada.

-Derecho de acceso: La forma en la cual un objeto es accedido por un sujeto.

Una de las dimensiones de la matriz consiste en los sujetos identificados que pueden intentar acceder a los datos. Habitualmente, esta lista consistirá en los usuarios o grupos de usuarios, aunque el control de acceso también se puede determinar por terminales, ordenadores y aplicaciones en lugar de o en conjunción con los usuarios. La otra dimensión indica los objetos sobre los cuales se realizan los accesos. Cada entrada en la matriz indica los derechos de acceso que un sujeto tiene sobre dicho objeto. [12]

La matriz se puede descomponer por columnas, definiendo listas de control de acceso. De esta forma por cada objeto, hay una lista de control de acceso que muestra los usuarios y sus derechos de acceso. [12]

La descomposición por filas lleva a la definición de los tickets de capacidades.

Un ticket de capacidad especifica objetos y operaciones autorizadas para un determinado usuario. El usuario tiene un numero de tickets y puede encontrarse autorizado para cederlos o entregarlos a otros. Debido a que los tickets pueden encontrarse dispersos representa un problema de seguridad mayor que las listas de control de acceso. Un ticket no debe ser falsificable. Una forma de llevarlo a cabe es hacer que el sistema se encargue de mantener los tickets de todos los usuarios. Dichos tickets pueden almacenarse en una región de memoria no accesible. Las consideraciones de red para el control de acceso orientado a datos son equiparables a aquellas para control de acceso orientado a usuarios. [12]

4.-Intrusos

Uno de los dos peligros de seguridad más conocidos son los intrusos (el otro son los virus), denominados hackers o crackers, pueden identificarse 3 clases: [12]

-Enmascarado: Un individuo que no está autorizado a utilizar el ordenador y que penetra los controles de acceso del sistema para aprovechar de una cuenta de usuario legítimo.

-Trasgresor: Un usuario legítimo que accede a datos, programas o recursos para los cuales dicho acceso no está autorizado, o estando autorizado para dicho acceso utilizar sus privilegios de forma maliciosa.

Usuario clandestino: Un usuario que sobrepasa el control de supervisión del sistema y usa dicho control para evadir la auditoria y el control de acceso o para suprimir la recogida de registros de acceso.

El enmascarado se trata habitualmente de un usuario externo, el transgresor interno y el clandestino puede ser cualquiera de los dos. [12]

Los ataques de los intrusos pueden ser benignos, que se encuentran personas que simplemente desean explorar redes y saber que hay ahí afuera y del lado serio se encuentras los que intentan leer datos privilegiados, realizar modificaciones no autorizadas a dichos datos o destruir un sistema. [13]

4.1-Tecnicas de intrusión

El objetivo de un intruso es ganar acceso a un sistema o incrementar el rango de sus privilegios de acceso. Generalmente esto requiere que el intruso consiga información que debiera estar protegida. El fichero de contraseñas se puede proteger de diferentes formas: [13]

-Cifrado unidireccional: El sistema almacena únicamente una forma cifrada de la contraseña de usuario. Cuando el usuario presenta una contraseña, el sistema cifra compara con el valor almacenado.

-Control acceso: El acceso al fichero que contiene las contraseñas se encuentra limitado a una o muy pocas cuentas.

Si se aplican una o ambas de estas contramedidas, se requiere que el intruso invierta un esfuerzo extra para conocer las contraseñas. Para conocer estas, estos intrusos usan técnicas como: [13]

1. Intentar las contraseñas por defectos utilizadas para las cuentas estándar que vienen creadas con el sistema.
2. Ensayar de forma exhaustiva todas las contraseñas de pequeño tamaño.
3. Probar palabras en el diccionario del sistema o una lista de contraseñas habituales
4. Recolectar información de los usuarios.
5. Intentar el número de teléfono del usuario, el numero de la seguridad social y la dirección del domicilio.
6. Utilización de troyanos para sobrepasar las restricciones de acceso.
7. Pinchar la línea entre un usuario remoto y el sistema destino.

Los primeros 6 métodos son diferentes mecanismos para adivinar una contraseña. Si el intruso tiene que verificar todas estas hipótesis intentando acceder a sistema, resulta tedioso y fácilmente evitable. [14]

Los ataques por medio de la adivinación de contraseñas son habituales, e incluso altamente efectivos, sobre todo cuando se pueden evaluar un gran número de hipótesis de forma automática y verificar su validez o no, sin que le proceso de prueba pueda detectarse. [14]

El séptimo método es difícil de contrarrestar, ya que es un error de usuario, de por ejemplo descargar software de dudosa procedencia. [14]

El octavo ataque es una cuestión de seguridad física, puede evitarse cifrando los enlaces. [14]

A continuación, vamos a repasar las dos principales contramedidas: prevención y detección. [14]

4.2-Proteccion de contraseñas

La primera línea de defensa contra los intrusos es el sistema de contraseñas. El identificador proporciona seguridad de la siguiente manera: [14]

-El identificador determina si el usuario esta autorizado a conseguir el acceso al sistema.

-El identificador determina los privilegios asociados al usuario.

-El identificador también se utiliza para un control de acceso discrecional.

4.3-Vulnerabilidad de las contraseñas.

Por lo general mucha gente, cuando se les permite elegir su propia contraseña, seleccionan una palabra que es fácilmente adivinable, tal como su propio nombre, nombre de una calle, una palabra común del diccionario y similares. Esto hace que el trabajo del password cracker sea as sencillo. [15]
Por esto mismo se deben prever intentos de penetración consistentes en prueba de combinaciones de nombres y contraseñas. [15]

Silas contraseñas fueran de 7 caracteres elegidos al azar de los 95 caracteres ASCII se pueden imprimir: [15]

-El espacio de búsqueda seria de 95ª la 7, alrededor de 7 x 10 a la 13.

-A 1000 ciframientos por segundos tomaría 2000 años construir la lista a verificar contra el archivo de contraseñas.

Una mejora al esquema de contraseñas consiste en:

-Asociar un número aleatoria de “n” bits a cada contraseña.

-El numero aleatorio se modifica al cambiar la contraseña.

-El número se guarda e el archivo de contraseñas en forma no cifrada.

-Se concatenan la contraseña y el numero aleatorio y se cifran juntos.

-El resultado cifrado se almacena en el archivo de contraseñas.

-Se aumenta por 2 a la n el espectro de búsqueda: a esto se llama salar el archivo de contraseñas.

Una protección adicional consiste en hacer ilegible el archivo de contraseñas encriptadas, otra protección adicional consiste en que el sistema sugiera a los usuarios contraseñas generadas según estos criterios. [15]

También es conveniente que el sistema obligue al usuario a cambiar sus contraseñas con regularidad. [15]

Una variante de la idea de contraseña es solicitar al usuario respuestas sobre información de contexto que debe conocer. Y otra variante es la de reto-respuesta: [15]

-Se acuerdan con el usuario algoritmos que se utilizaran según el día y/o la hora.

-Cuando el usuario se conecta: El sistema suministra un argumento y luego el usuario debe responder con el resultado correspondiente al algoritmo vigente ese día a esa hora.

5.-Software Malicioso

Los tipos más sofisticados de amenazas para un sistema informático se encuentran presentes en los programas que explotan las vulnerabilidades de dicho sistema. El termino genérico de estas amenazas es software malicioso o malware. Software diseñado para causar daño o utilizar recursos del ordenador, suele encontrarse escondido dentro de un programa. [17]

5.1-Programas maliciosos

Estas amenazas se pueden dividir en dos categorías, aquellas que necesitan un programa anfitrión y otras que son independientes.

También podemos realizar diferencia entre los casos que no se pueden replicar y aquellos que sí. [18]

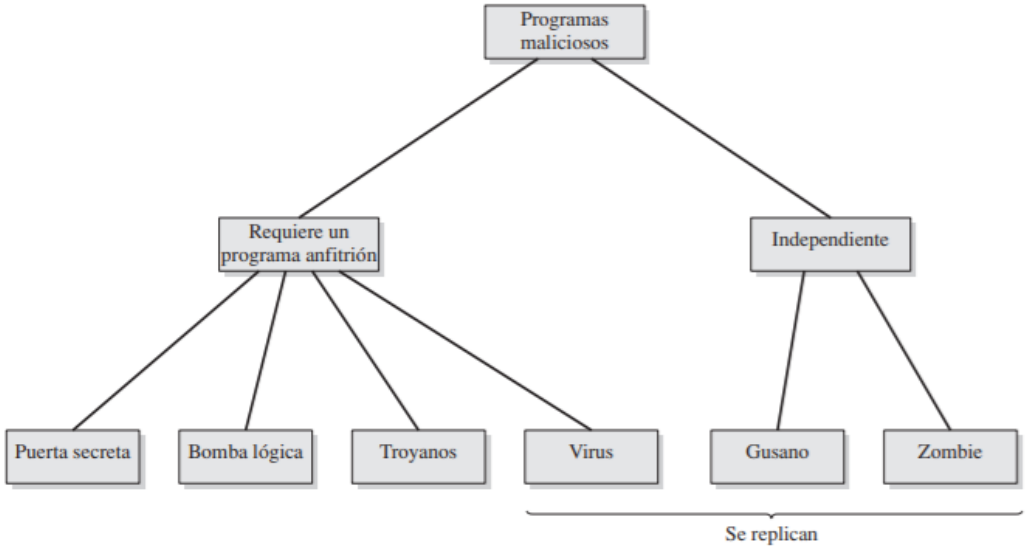

Figura 4. Taxonomía de programas maliciosos

5.2-Puerta secreta

Pueden convertirse en amenazas cuando son utilizados por programadores sin escrúpulos para ganar accesos no autorizados. Fue la idea básica de vulnerabilidad retrasada en la película Juegos de Guerra. Una técnica empleada era enviar una actualización falsa del sistema operativo a un ordenador y dicha actualización contuviera un troyano, que se podría activar por medio de una puerta secreta que permitía al tiger team conseguir acceso al sistema. [18]

La medida de seguridad se debe centrar en el desarrollo de programas y las actividades relativas a la actualización de software. [18]

5.3-Bomba lógica

Uno de los más viejos peligros, es un código insertado dentro de un programa legítimo que explotara bajo ciertas condiciones. [19]

5.4-Troyano

Es un programa útil o aparentemente útil, que contiene código oculto que, al invocarse, realiza una función no deseada o dañina. [19]

Los programas troyanos se utilizan para realizar tareas de forma indirecta que el usuario no autorizado no podría realizar directamente. [19]

5.5-Virus

Un virus es un programa que puede infectar otros programas modificándolos, y estas incluyen la copia del programa virus, que pude a continuación infectar a otros programas. [19]

Como su análogo biológico, un virus informático contiene código de instrucciones que se encarga de realizar copias de si mismo, infectando un ordenador por completo [19]

5.6-Gusanos

Los gusanos utilizan las conexiones de red para expandirse de un sistema a otro. Una vez que se encuentran activos en un sistema, se puede comportar como un virus informático, puede implantar troyanos o realizar cualquier otro tipo de acciones destructivas. [20]

Utilizan distintos tipos de vehículos de comunicación: [20]

-Herramientas de correo electrónico.

-Capacidad de ejecución remota.

-Capacidad de conexión remota.

5.7-Zombie

Un programa zombie toma el control de otro ordenador conectado a internet y posteriormente utiliza el mismo para lanzar ataques que son difíciles de trazar ocmo provenientes del creador del zombie. Se utilizan para denegar servicio, como por ejemplo sitios webs. [20]

5.8-Estrategias de los antivirus

La solución ideal para los virus es la prevención, no permitiendo que el virus entre, pero esto es muy difícil de lograr. La siguiente opción es conseguir realizar los siguiente: [21]

-Detección: Una vez que ha corriendo una infección, determinar que ha sucedido y localizar el virus.

-Identificación: Una vez que se detecta, identificar que virus especifico ha infectado el programa

-Eliminación: Una vez identificado el virus, se eliminan todos los restos del virus del programa infectado.

Si se detecta, pero no se puede identificar o eliminar, hay que recurrir a cargar una versión limpia del SO desde la copia de seguridad.

Existen 2 estrategias importantes de los antivirus actualmente. [22]

El **descifrado genérico.** Permite a los programas antivirus detectar fácilmente los virus, manteniendo una velocidad de exploración alta. Al querer desencriptarse el virus a si mismo. El escáner GD detecta estas estructuras. [22]

El **sistema de inmunidad digital**. El objetivo de este sistema es proporcionar un tiempo de respuesta rápido de forma que los virus puedan ser expulsados casi tan rápido como se introducen. Cuando un nuevo virus entra en una organización, el sistema de inmunidad automáticamente lo captura, lo analiza, añade mecanismos de detección y defensa contra él, lo elimina y pasa información sobre dicho virus a los sistemas que se ejecutan en IBM Anti Virus. [23]

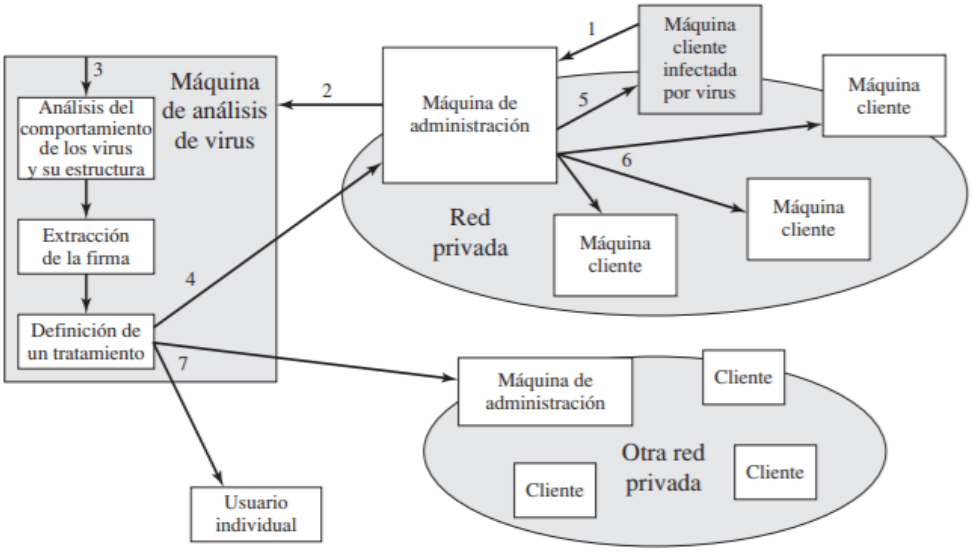

Figura 5. Sistema de inmunidad digital

5.9-Virus por correo

Este tipo de virus cuando se abre, se envía a si mismo a todo el mundo en el listín de direcciones del software de correo del usuario y además realiza daño localmente. [24]

6.-Sistemas confiables

Un requisito ampliamente aplicable es la protección de datos o de recursos por medio de niveles de seguridad. Donde la información se puede organizar en grandes categorías y en las que a los usuarios se les pueden otorgar credenciales de acceso a ciertas categorías de datos. [24]

A esto se le llama seguridad multinivel. La directriz general para los requisitos de seguridad multinivel es que el sujeto a nivel alto no puede compartir información un con sujeto de nivel inferior o no comparable a menos que el flujo de información refleje la voluntad de un usuario autorizado. Por ello el sistema debe proporcional lo siguiente: [25]

-No leer hacia arriba: Un sujeto solo puede leer un objeto de un nivel de seguridad igual o inferior.

-No escribir hacia abajo: Un sujeto solo puede escribir en un objeto de un nivel de seguridad igual o superior.

Otra estrategia es la del monitor de referencias, que es un elemento de control en el hardware con el sistema operativo de un ordenador que regula el acceso de sujetos a objetos en base a unos parámetros de seguridad de dichos participantes. El monitor tiene acceso a un fichero, conocido como base de datos del núcleo de seguridad, que muestra una lista de los privilegios de acceso de cada sujeto y os atributos de protección de cada objeto. El monitor de referencia aplica las reglas de seguridad y tiene las siguientes propiedades: [24]

-Mediación completa: Las reglas de seguridad se aplican a cada acceso.

-Aislamiento: El monitor de referencia y base de datos están protegidos de cualquier modificación no autorizada.

-Verificabilidad: La corrección del monitor de referencias debe estar probada.

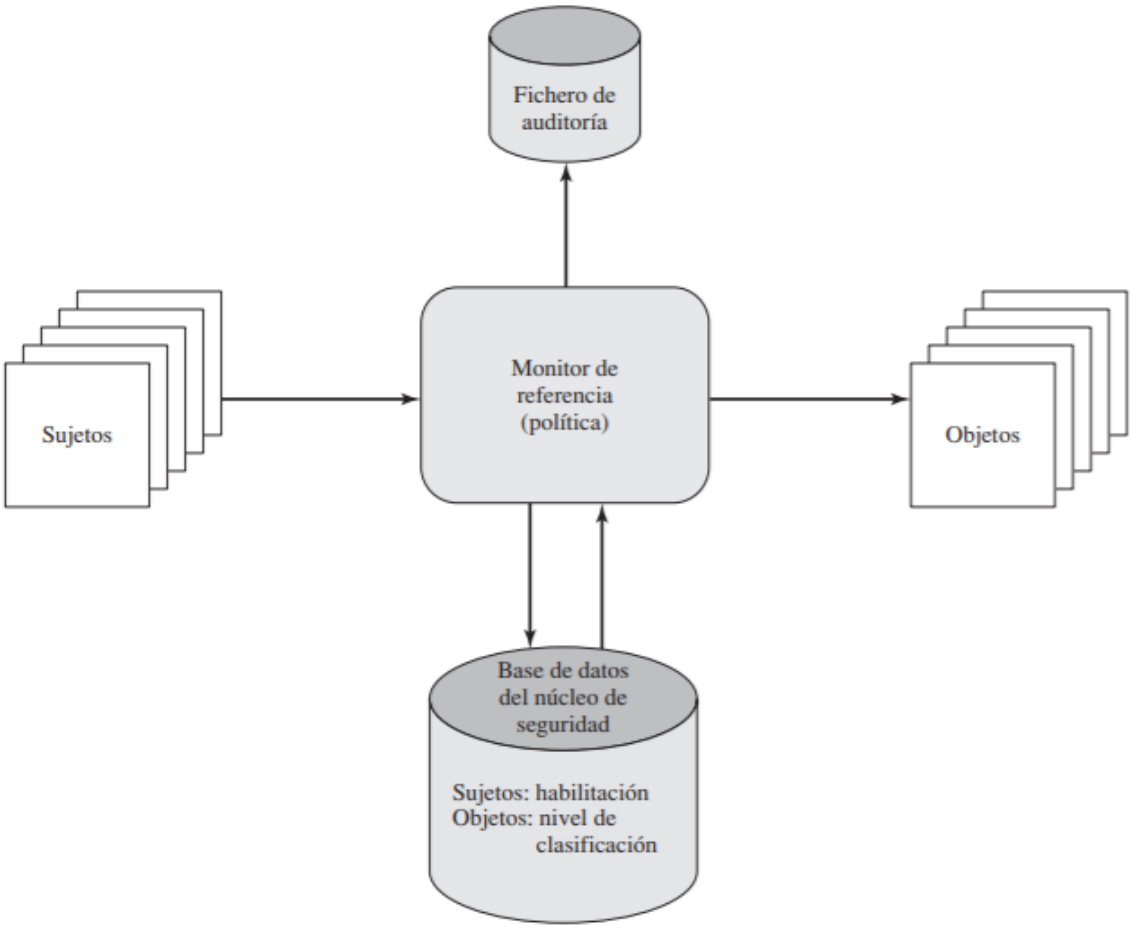

Figura 6. Concepto de monitor de referencia.

El último elemento mostrado en la figura 6. Es el fichero de auditoria, que almacena eventos de seguridad importantes, como por ejemplo la detección de violaciones de seguridad y los cambios autorizados en la base de datos del núcleo de seguridad. [25]

Conclusión

Todos los días aparecen nuevos y complejos tipos y maneras de penetrar nuestro sistema, constantemente se van a seguir encontrando fallas en la seguridad. Por lo tanto, es necesario mantener un estado de alerta y actualización permanente, ya que la seguridad es un proceso continuo que exige aprender sobre la propia experiencia.

Debido a estas constantes amenazas es necesario que los usuarios y empresas enfoquen atención en el grado de vulnerabilidad y herramientas de seguridad con las que cuentan para hacerle frente a posibles ataques, que luego se pueden traducir en importantes pérdidas.

El eslabón más débil en este caso es el usuario común, por eso hay que hacer hincapié en que no basta con tener el mejor antivirus o el más actualizado a la fecha, lo importante es que uno mismo se transforme en su propio antivirus.

Referencias

[1] Sistemas Operativos. MAGISTER DAVID LUIS LA RED MARTINEZ.

Cap. 4 - Pág. 144

[2] Fundamentos de seguridad informática. Carlos Arturo Avenia. Fundación Universitaria del Área Andina. 2017.Fondo editorial Areandino

Cap. 1 - Pág. 10

[3] W. Stallings. Sistemas Operativos. 5ta Edición. Madrid. Pearson, Prentice Hall.

2005. Cap. 6 - Pág. 690

[4] W. Stallings. Sistemas Operativos. 5ta Edición. Madrid. Pearson, Prentice Hall.

2005. Cap. 6 - Pág. 691

[5] W. Stallings. Sistemas Operativos. 5ta Edición. Madrid. Pearson, Prentice Hall.

2005. Cap. 6 - Pág. 692

[6] W. Stallings. Sistemas Operativos. 5ta Edición. Madrid. Pearson, Prentice Hall.

2005. Cap. 6 - Pág. 693

[7] W. Stallings. Sistemas Operativos. 5ta Edición. Madrid. Pearson, Prentice Hall.

2005. Cap. 6 - Pág. 695

[8] W. Stallings. Sistemas Operativos. 5ta Edición. Madrid. Pearson, Prentice Hall.

2005. Cap. 6 - Pág. 697

[9] W. Stallings. Sistemas Operativos. 5ta Edición. Madrid. Pearson, Prentice Hall.

2005. Cap. 6 - Pág. 698

[10] W. Stallings. Sistemas Operativos. 5ta Edición. Madrid. Pearson, Prentice Hall.

2005. Cap. 6 - Pág. 699

[11] W. Stallings. Sistemas Operativos. 5ta Edición. Madrid. Pearson, Prentice Hall.

2005. Cap. 6 - Pág. 700

[12] W. Stallings. Sistemas Operativos. 5ta Edición. Madrid. Pearson, Prentice Hall.

2005. Cap. 6 - Pág. 701

[13] W. Stallings. Sistemas Operativos. 5ta Edición. Madrid. Pearson, Prentice Hall.

2005. Cap. 6 - Pág. 702

[14] W. Stallings. Sistemas Operativos. 5ta Edición. Madrid. Pearson, Prentice Hall.

2005. Cap. 6 - Pág. 703

[15] W. Stallings. Sistemas Operativos. 5ta Edición. Madrid. Pearson, Prentice Hall.

2005. Cap. 6 - Pág. 706

[16] Sistemas Operativos. MAGISTER DAVID LUIS LA RED MARTINEZ.
Cap. 4 - Pág. 148

[17] W. Stallings. Sistemas Operativos. 5ta Edición. Madrid. Pearson, Prentice Hall. 2005. Cap. 6 - Pág. 713

[18] W. Stallings. Sistemas Operativos. 5ta Edición. Madrid. Pearson, Prentice Hall. 2005. Cap. 6 - Pág. 714

[19] W. Stallings. Sistemas Operativos. 5ta Edición. Madrid. Pearson, Prentice Hall. 2005. Cap. 6 - Pág. 715

[20] W. Stallings. Sistemas Operativos. 5ta Edición. Madrid. Pearson, Prentice Hall. 2005. Cap. 6 - Pág. 716

[21] W. Stallings. Sistemas Operativos. 5ta Edición. Madrid. Pearson, Prentice Hall. 2005. Cap. 6 - Pág. 719

[22] W. Stallings. Sistemas Operativos. 5ta Edición. Madrid. Pearson, Prentice Hall. 2005. Cap. 6 - Pág. 720

[23] W. Stallings. Sistemas Operativos. 5ta Edición. Madrid. Pearson, Prentice Hall. 2005. Cap. 6 - Pág. 721

[24] W. Stallings. Sistemas Operativos. 5ta Edición. Madrid. Pearson, Prentice Hall. 2005. Cap. 6 - Pág. 722

[25] W. Stallings. Sistemas Operativos. 5ta Edición. Madrid. Pearson, Prentice Hall. 2005. Cap. 6 - Pág. 723

[26] W. Stallings. Sistemas Operativos. 5ta Edición. Madrid. Pearson, Prentice Hall. 2005. Cap. 6 - Pág. 724

Bibliografías

[1] W. Stallings. Sistemas Operativos. 5ta Edición. Madrid. Pearson, Prentice Hall. 2005.

[2] Fundamentos de seguridad informática. Carlos Arturo Avenia. Fundación Universitaria del Área Andina. 2017.Fondo editorial Areandino.

[3] Sistemas Operativos. MAGISTER DAVID LUIS LA RED MARTINEZ.