Resumen del libro Sistemas Operativos – Stallings, 4ta edición, español
Por Vera Caruso (verita@arda.com.ar)
Versión 0.1 al 09 de diciembre de 2003

Introducción / Conceptos Principales

Sistema operativo
- programa que controla la ejecución de los programas de aplicación y que actúa como interfaz entre las aplicaciones de usuario y el hardware del equipo.
- Objetivos: comodidad, eficiencia, capacidad de evolución.

Ofrece servicios de:
- Creación de programas, debugger
- Ejecución de programas
- Administrador de recursos: Acceso a e/s, Acceso controlado a archivos, Acceso al sistema (sistema compartido)
- Detección y respuesta a errores
- Contabilidad (accountabilty)

Facilidad de evolución, frente a:
- Actualizaciones de hardware
- Nuevos servicios de los usuarios
- Correcciones, patches

Historia / Evolución
- Proceso en serie
- Proceso por lotes
- Por lotes con multiprogramación
- Sistemas De tiempo compartido

Logros principales

Procesos

- Funciones principales del SO (kernel)
    - Creación y terminación de procesos
    - Planificación y expedicionm de procesos
    - Cambio de procesos
    - Sincronización y soporte a la comunicación entre proc
- Problemas principales (que surgían antes)
    - Sincronización incorrecta
    - Fallos de exlcusión mutua
    - Funcionamiento no determinista
    - Interbloqueo (deadlock)

Gestión de memoria

- Funciones principales del SO (kernel)
    - Asignación de espacio de direcciones a los procesos
    - Intercambio
    - Gestión de páginas y segmentos
- Responsabilidades principales
    - Aislamiento del proceso
    - Asignación y gestión automáticas
    - Soporte para la programación modular
    - Protección y control deacceso
    - Almacenamiento a largo plazo
- Memoria Virtual

Seguridad y protección de la información

- Control de acceso
- Control del flujo de información
- Certificación

Planificación y gestión de recursos

Funciones principales del SO (kernel) en la gestión de E/S

- Gestión de buffers
- Asignación de canales de e/s y dispositivos a los procesos

- Equidad
- Sensibilidades diferenciales
- Eficiencia, minimizar el tiempo de respuesta, maximizar productividad

Otras funciones de soporte:

- Tratamiento de interrupciones
- Contabilidad
- supervisión

Estructura del sistema

- Software modular, con interfaces entre sí
- Sistema operativo por niveles: 1..13 (1..4 hardware; 5..7 sisop; 8..13 trato con objetos externos, e/s, red)

Características de los sisop modernos

- Arquitectura microkernel (vs. gran kernel monolítico)
- Multihilos
- Multiproceso simétrico (SMP)
  - + rendimiento
  - disponibilidad
  - crecimiento incremental
  - escalabilidad
- Sisop distribuidos
- Diseño orientado a objetos

Procesos

Creación de un proceso

- Nuevo trabajo por lotes
- Conexión interactiva
- Creado por el so para dar un servicio
- Generado por un proceso existente

Estados de un proceso

Modelo	Descripción	Estados	Explicación
Modelo de 2 estados	Con una sola cola, puede seleccionarse uno que esté bloquedo esperando e/s	Ejecutando	
		No ejecutando	Al crearse. Cola de procesos en espera.

Modelo	Descripción	Estados	Explicación
Modelo de 5 estados	Se divide la *No ejecución* en 2 estados: Listo y Bloqueado	Nuevo	No está cargado en memoria, (admitir): Listo
		Listo	Para ejecutarse (expedir): Ejec. Terminar (muere su padre)
		Ejecución	Sale por quantum: Listo, sale por suceso: bloqueado, sale por terminado.
		Bloqueado	Ocurre suceso: listo Muere su padre: terminado
		Terminado	
Modelo de 7 estados	Usa memoria virtual, se agrega un estado Suspendido, que se combina con Listo y Bloqueado.	Nuevo	Se puede admitir como Listo o Listo,Susp
		Listo	A Listo,Susp (nohay mem) o Ejec
		Listo, Suspendido	En mem secund, pero disponible
		Bloqueado	Puede pasar a Bloq,Susp para liberar mem
		Bloqueado, Suspendido	Está en mem secund esperando un suceso Puede pasar a Listo,Susp
		Ejecución	Listo (quantum), Bloqueado, a Listo,Susp
		Terminado	

Observaciones:
Modelo de 5 estados: sin memoria virtual, todos los procesos de todas las colas tienen que estar en memoria principal, surge entonces el intercambio y un nuevo estado, Suspendido, que se combina con los demas.

Modelo 7 estados: Cuando se crea un nuevo proceso,puede añadirse a la cola de Listos o Listos,Susp

Razones para la suspensión:
- Intercambio: se necesita liberar memoria principal
- Solicitud de un usuario interactivo: se solicita la suspension
- Temporización
- Solicitud del proceso padre
- Otra razón: sospecha que es causante de un problema

Estructuras de control del so
Qué deben contener las tablas

Tablas de memoria
- Asignación de memoria principal a procesos
- Asignación de memoria secundaria a procesos
- Atributos de protección de bloques de mem principal o virtual

Tablas de Procesos
La imagen del proceso puede guardarse en mem secundaria, pero al menos una parte tiene que estar en mem principal

Imagen de un proceso
- Datos del usuario
- Programa del usuario
- Pila del sistema (LIFO) para almacenar parámetros y direcciones de retorno a llamadas a procedimientos
- PCB (Bloque de control del proceso)
    - PID del proceso

- Información del estado del procesador (registros)
- Información de control del proceso (estado, memoria, permisos, etc)

Modos de ejecución
- Modo usuario
- Modo de sistema, de control o modo kernel

Creación de un nuevo proceso
1) Asignar un pid
2) Asignar espacio para el proceso, imagen
3) Inicializar el PCB
4) Establecer los enlaces apropiados, colas de planificación

Cambios de proceso
- Interrupción de reloj, quantum
- Interrupción de E/S
- Page fault, fallo de memoria
- Cepo/Trap, error fatal o no, asociado con la ejecución del programa
- Llamada del supervisor, el proceso invoca una operación que se debe ejecutar en otro modo, hay un cambio de estado a Bloqueado

Cambios de modo y estado

Cambio	Descripción	Acciones
Cambio de modo	Cambio de modo usuario a modo nucleo para ejecutar alguna operación privilegiada	Salvar contexto de programa Asignar PC a direcc de comienzo de programa que maneja interrupcion Cambiar modo usuario-> nucleo
Cambio de estado	Para cambiar de un proceso a otro, un cambio de proceso (process switch) implica un cambio de estado.	Salvar contexto Actualizar el estado de la PCB del proceso Mover la pcb a la cola adecuada (listos, bloq,...) Seleccionar otro proceso Actualizar el estado del pcb del proceso elegido Actualizar las estruct de datos en memoria

Puede producirse un cambio de modo sin cambiar el estado del proceso que está actualmente en ejecución.
El cambio de proceso supone un cambio de estado, e implica un esfuerzo mayor que un cambio de modo.

Formas de ejecución del so
- Ejecutar el kernel fuera de cualquier proceso. El so tiene su propia región de memoria y su propia pila paracontrolar las llamadas y retornos de procedimiento. El concepto de proceso se aplica sólo a los programas del usuario.
- Ejecutar dentro de los procesos de usuario. Cuando hay interrupción se salva contexto, hay cambio de modo hacia una rutina del so, aunque la ejecución continúa dentro del proceso de usuario en curso, no hay un cambio de proceso.
- Sistema operativo basado en procesos. El sw que forma parte del kernel ejecuta en modo kernel, y las funciones importantes se organizan en procesos separados.

Hilos, SMP y microkernels

Concepto de proceso

- Programa ejecutable
- Datos asociados al programa
- Contexto de ejecución, estado.
- Unidad de propiedad de los recursos
- Unidad de expedición

De cada hilo se conoce
- El estado de ejecución
- El contexto del procesador
- Una pila de ejecución
- Almacenamiento estático para variables locales
- Acceso a la memoria y recursos del proceso, **compartidos** con todos los hilos del mismo

Entorno multihilo
- Capacidad del so para mantener varios hilos de ejecución dentro de un mismo proceso.
- Continua habiendo una sola PCB, y un espacio de direcciones del usuario asociados al proceso, pero ahora hay pilas separadas para cada hilo, así como distintos bloques de control para cada uno.

Beneficios de uso de hilos
Se tarda menos en crear un hilo que un proceso
Se tarda menos en terminar un hilo que un proceso
Se tarda menos en cambiar entre dos hilos de un mismo proceso
Se aumenta la eficiencia de la comunicación entre programas

Ejemplos de uso de hilos (sistemas monousuario y multiproceso)
trabajo interactivo y segundo plano
- Procesamiento asíncrono
- Aceleración de la ejecución
- Estructuración modular de los programas.

Estados de un hilo
- Ejecución, Listo y Bloqueado
- No tiene sentido hablar de suspensión, porque si un proceso está expulsado de la mem principal, todos sus hilos también lo están! (comparten el mismo espacio de direcciones del proceso)

Operaciones básicas de los hilos
- Creación
    - Tiene su propio conexto, pasa a la cola de listos
- Bloqueo
    - Cuando necesita esperar por un suceso, se bloqueda, ahora el procesador pasa a ejecutar otro hilo Listo
    - Solo para KLT: El bloqueo de un hilo NO supone el bloqueo de todo el proceso... sino se pierde la funcionalidad del uso de hilos, empieza a ejecutar algun otro de la cola de Listos.
- Desbloqueo
    - Cuando retorna el suceso por lo que el hilo se bloqueó, pasa a la cola de Listos.
- Terminación

Hilos a nivel usuario (ULT) y a nivel kernel (KLT)

Característica	ULT	KLT
Gestión de hilos	Lo realiza la aplicación (biblioteca de hilos), el kernel no es conciente de que existen hilos	Lo hace el núcleo.
Estados	Hay un unico estado (según el nucleo), el del proceso completo. Cuando se bloquea el proceso, alguno de los hilos puede pensar que sigue en estado Ejecutando! Cuando se reinicie el proceso, sigue en la ejecucion que estaba	Se mantienen los estados del proceso y de todos los hilos.
Intercambio de hilos	No necesita privilegios de modo kernel, se evita la sobrecarga de 2 cambios de modo (usuario-kernel y kernel-usuario)	El paso de un hilo a otro del mismo proceso necesita un cambio de modo (a kernel)
Planificación	Se puede hacer una planificación específica	Podrian ejecutar realmente en paralelo, se complica un poco
Portabilidad	Pueden ejecutar en cualquier so	Depende de la capacidad del SO de manejar KLT
Bloqueos	Cuando se bloquea un hilo, se bloquea todo el proceso. Solución: jacketing: hacer que la llamada bloqueadora no lo sea.	Se bloquea solo el hilo
Ejecución	Sólo puede ejecutar en un unico procesador	Pueden planificarse múltiples hilos en múltiples procesadores

Monohilo

- (1:1) UN proceso UN hilo [Unix clasicos]

Cada hilo de ejecución es un único proceso con sus propios recursos y espacio de direcciones

- (M:1) MUCHOS procesos, UN hilo cada uno [WinNT, Linux]

Un proceso define un espacio de direcciones y unos recursos dinámicos propios. Pueden crearse varios hilos que ejecuten en el proceso

Multihilo

- (1:M) UN proceso, MUCHOS hilos

Un hilo puede emigrar del entorno de un proceso a otro, esto permite que el hilo pueda migrar entre sistemas distintos

- (M:N) MUCHOS procesos, MUCHOS hilos

combina 1:m con m:1

Multiprocesamiento

Procesamiento en paralelo

Instrucción / Datos	Datos simple	Datos múltiples
Instrucción simple	SISD	SIMD
Instrucción múltiple	MISD (no implementada)	MIMD

Otra clasificación es como se asignan los procesos a procesadores

SIMD	

MIMD	Memoria compartida (fuertemente acoplados)	Maestro / esclavo El kernel siempre está en un procesador Un fallo puede hacer caer a todo el sistema
		Multiproceso simétrico (SMP) El kernel puede ejecutar en cualquier procesador El so es mas complejo pero mas estable
	Memoria distribuida (debilmente acoplados)	Agrupaciones (clusters) Diseño complejo del so

Diseño de un so multiprocesador

Un SO SMP gestiona el procesador y el resto de los recursos para que el usuario lo vea como un monoprocesador multiprogramado.
Puntos clave:

- Procesos o hilos concurrentes: se debe permitir que varios procesadores ejecuten el mismo código del nucleo al mismo tiempo. Se deben gestionar las tablas para evitar el interbloqueo y las operaciones no válidas.
- Planificación: se deben evitar los conflictos, porque la planificación se puede realizar en cualquier procesador
- Sincronización: mediante la sincro se impone la exclusión mutua y un orden de sucesos
- Gestión de memoria: explotar la capacidad de paralelismo de hardware, coordinar los mecanismos de paginación para garantizar consistencia.
- Fiabilidad y tolerancia a fallos.

Microkernels

Evolución:

- SO monolíticos
- SO por capas: las funciones se organizan jerárquicamente, sólo se producen interacciones entre capas adyacentes. La mayoría de las capas se ejecutan en modo kernel.
- Microkernel

Características

- Sólo las funciones absolutamente escenciales permanecen en el núcleo, las menos escenciales se construyen sobre el microkernel y se ejecutan en modo usuario.
- Tiende a reemplazar la arquitectura tradicional (en capas verticales) por una horizontal. Los componentes se implementan como procesos servidores e interactúan con los otros a través de mensajes distribuidos por el microkernel.

Ventajas

- Uniformidad de interfaces
- Extensibilidad
- Flexibilidad
- Portabilidad
- Fiabilidad
- Soporte a sistemas distribuidos
- Soporte para so OO.

Desventaja: rendimiento. Consume más tiempo construir y enviar un mensaje que efectuar una llamada al sistema. Aunque puede solucionarse agregando funcionalidades al microkernel, se le añadiría rigidez al diseño. Otro enfoque: reducir aún más el tamaño, para eliminar la penalización de rendimiento e incrementar la flex y fiabilidad.

Diseño del microkernel

Un microk debe incluir las funciones que dependen directamnete del hardware y cuya funcionalidad es necesaria para dar soporte a las aplicaciones y servidores que ejecutan en modo nucleo.

- Gestión de memoria de bajo nivel
    - Control del espacio de direcciones, traducción de páginas y direcc virtuales. La paginación y la protección podrían implementarse afuera
- Comunicación entre procesos
    - Paso de mensajes. Debe manejar puertos (cola de mensajes destinadas a un proceso en particular) y una lista de capacidades de acceso (para ver qué procesos tienen permiso para comiunicarse con él).
- Gestión de interrupciones y e/s
    - Podrían gestionarse las int de hw como mensajes, e incluir los puertos de e/s en espacios de direcciones. Debe poder reconocer las int pero no gestionarlas, sino que debe enviar un mensaje al proceso asociado

Concurrencia: exclusión mutua y sincronización

- Sistema monoprocesador multiprogramado: el intercalamiento da la apariencia de ejecución simultánea.
- Aunque no hay paralelismo real y existe cierta sobrecarga, hay beneficios en la ejecución intercalada (eficiencia y estructuración de los programas).

Formas de interacción entre procesos

Grado de conocimiento	Relación	Influencia entre sí	Posibles problemas
Independientes	Competencia	Los tiempos de ejecución se pueden ver afectados	Exclusión mutua Interbloqueo (renovables) Inanición
Indirecto	Cooperación por compartimiento	El resultado puede depender de la información obtenida de los otros Los tiempos de ejecución se pueden ver afectados	Exclusión mutua Interbloqueo (renovables) Inanición Coherencia de datos
Directo, hay comunicación	Cooperación por comunicación		Interbloqueo (consumibles) Inanición

Problemas de control a resolver
- Exclusión mutua
- Sincronización y coherencia de datos
- Interbloqueo (recursos renovables o no renovables)
- Inanición

Exclusión mutua

El cumplimiento de la exclusión mutua puede provocar: deadlock e inanición.

Requisitos para la exclusión mutua

- Sólo un programa puede acceder a su sección crítica en un momento dado
- Cuando el proceso accede a su sección crítica, no debe interferir con los otros procesos
- Un proceso no puede demorarse indefinidamente en su sección crítica (hay que evitar la inanición y el interbloqueo).
- Cuando no hay ningún proceso en su sección crítica, cualquier proceso que quiera entrar en la suya debe poder hacerlo sin demora.

Soluciones por software

Algoritmo de Dekker

Soluciones	Funcionamiento	Consecuencias
1er intento	1 variable global compartida turno = nro proceso accede cuando esta en su numero.	Se deben alternar en el uso, el ritmo de ejecución se da por el más lento. Busy waiting Si hay una falla, el otro se bloquea permanentemente.
2do intento	Se puede analizar el señal[i] del otro, pero no modificarlo Cuando la señal del otro está en falso, entra este, cambia su señal propia a true	Si el otro falla, no hay bloqueo Pero **puede no garantizar exclusión mutua**.
3er intento	Fija su estado sin conocer el del otro	Puede provocar un interbloqueo, cada proceso puede insistir en su derecho de entrar en la secc crítica, no hay vuelta atrás
4to intento	Activan su señal para indicar que quieren ingresar en la secc crítica, pero deben poder cederla a otro proceso	Puede provocar **livelock**, cortesía mutua, ninguno entra (salvo que cambie el ritmo de ejecución, y se libera)
Solución correcta	Var Turno indica quien tiene prioridad para exigir su ingreso en secc crítica Señal[i] para indicar intención de ingresar	Pongo señal[i]=1, miro señal[n] para ver si hay alguien mas con 1, si -> analizo turno, no, entro.

Algoritmo de Peterson

Al igual que el Alg de Dekker:
Var global señal[n] indica la posición de cada proceso frente a la exclusión mutua
Var turno resuelve los conflictos de simultaneidad.

```
Main()
        Señal[0]=0; señal[1]=0;

Proceso1()
        While(1)
        {
                señal[1]=1;
                turno = 0;
                while (señal[0] && turno == 0)
                        /* NO puedo entrar, tengo que esperar! */
        sección_critica()
        señal[1]=0;
        }
```

idem para el Proceso0 (pero el while analiza el del P1)

Soluciones por hardware

Inhabilitación de interrupciones

- Sólo para monoprocesadores.
- No hay superposición de los procesos concurrentes, sino sólo intercalación.
- Para garantizar exclusión mutua, alcanza con impedir que un proceso sea interrumpido, se pueden poner primitivas que inabiliten/habiliten las int y entre medio, poner el acceso a la sección crítica.

Instrucciones especiales de máquina
- Para configuraciones multiprocesador, con acceso común a la memoria.
- Los procesadores funcionan independiente, no es posible implementar mecanismos para trabajar sobre las int
- Es aplicable a sistemas con memoria compartida (monoprocesador y multiprocesador)
- Usa busy waiting, puede producir inanición e interbloqueo.

Instrucción Comparar y Fijar (Test & Set)
- Es una atómica.
- Var compartida cerrojo. Sólo puede entrar aquel proceso encuentre cerrojo=0.
- El resto, quedan en espera activa (busy waiting).
- Cuando un proceso abandona la sección crítica, vuelve a poner cerrojo=0.

Instrucción Intercambiar
- Cerrojo se inicia =0. Cada proceso tiene var local clave=1.
- Puede entrar el primero que encuentre cerrojo=0, y la cambia a cerrojo=1 (así excluye que los otros ingresen).
- Cuando un proceso abandona la sección crítica, vuelve a poner cerrojo=0.

Semáforos
Dos o más procesos cooperan mediante señales (semáforos). Pueden verse obligados a detener su ejecución hasta recibir una señal específica.
Primitivas atómicas para operar semáforos.

Funcionamiento
- **Inicialización**: semaforo=n; valor n>0, con la cantidad de recursos de cierto tipo
- **Wait**: semaforo--; si valor < 0, el proceso que ejecuta wait se bloquea.
- **Signal**: semaforo++; si valor<0, se debloquea un proceso bloqueado por un wait, si valor >0, se agrega un recurso a la lista de disponibles.

Mientras los procesos están bloqueados, esperando el acceso, se quedan en una cola.
Semáforos robustos: Si se especifica el orden (ej FIFO)
Semáforos débiles: no se indica el orden para extraer los elementos.

Características:
- Garantizan exclusión mutua.
- Si es robusto, garantiza que no se produce inanición (los débiles sí pueden producirla).
- Si se inicia con la cantidad de recursos disponibles, entonces se puede interpretar que:
    - Cantidad >=0, es el número de procesos que pueden ejecutar un wait sin bloquearse.
    - Cantidad < 0, en valor absoluto es la cantidad de procesos bloqueados en la cola.
- Desventaja: las primitivas (wait, signal) tienen que estar desperdigadas por todo el programa, haciendo difícil la detección de algún problema de sincronización.

Problema del productor – consumidor

Definición del problema:
- N productores generan datos y los guardan en un buffer

- 1 consumidor los retira de a uno
- no se deben superponer operaciones sobre el buffer.
- Bloqueos: del productor, si inserta en buffer lleno; del consumidor si quiere eliminar en el buffer vacío.

Problema de la barbería

Definición del problema:
Se quiere administrar la capacidad de la tienda(20), del sofa(¿), de las sillas(3), del pago y de coordinar las tareas de ser_barbero o ser_cajero.

Monitores
Son estructuras de un lenguaje de programación que ofrecen funcionalidad equivalente a la de los semáforos, y que son más faciles de controlar. (implementado en Java y como una biblioteca de programas)

Monitores con señales

Características:
- Encapsulamiento: Var de datos locales sólo accesible a los procedimientos del monitor
- Ingreso al monitor invocando a uno de sus procedimientos
- Sólo un proceso ejecuta por vez, cualquier otro que invoque se bloquea.
- Deben incluir alguna forma de sincronización (ej. Que deba esperar a una condición mientras esté ejecutando en el monitor, o sea que se bloquee hasta obtener la condición). Utiliza 2 variables de condicion (*diferentes* al uso de semaforos)
    - Cwait(c), suspende la ejecucion del proceso que llama, hasta que surja la condicion c. Ahora el monitor esta disponible para que otro proceso ingrese.
    - Csignal(c), reanuda la ejecucion de algun proceso suspendido esperando por c, si hay varios, elige uno, sino, no hace nada. Si el proceso que ejecuta un csignal no termino, hacen falta dos cambios de proces, uno para suspenderlo y otro para reanudarlo cuando el monitor vuelva a quedar disponible.
- La ventaja frente a los semáforos es que todas las funciones de sincronización están concentradas dentro del monitor, eso hace más facil el control.

Monitores con notificación y difusión

Características:
La primitiva csignal se reemplaza con cnotify. Cuando un proceso ejecuta un cnotify (x), se origina una notificación a la cola de la condición x, pero el proceso que da la señal, continúa ejecutándose. Cuando el monitor vuelve a estar disponible, se va a ejecutar el proceso que encabece la cola de x (ojo! Hay que volver a chequear la condicion, porque no se garantiza que no ingrese otro antes)
Podrían no reactivarse a la fuerza los procesos, utilizando cbroadcast(x), de forma tal que los procesos que experan por la condición, se pongan en estado de Listo.

Paso de mensajes
Se requiere la comunicación y la sincronización cuando dos procesos interactúan entre sí.
Este mecanismo tiene la ventaja de que puede ser implementado en sistemas distribuidos, en sistemas multiprocesador y en monoprocesador con memoria compartida.

Primitivas: send y receive.

Send bloqueante, Receive bloqueante (**Rendez Vouz**)	Los dos se bloquean hasta que se entrega el mensaje	Fuerte sincronización
Send NO bloqueante, Receive bloqueante	Se bloquea sólo el receptor	Ej. Proceso servidor Permite exclusión mutua
Ninguna bloqueante	Ninguno espera	

Direccionamiento
- Directo: el send indica el proceso destino. En el caso del receive, se podría indicar específicamente de quien se quiere recibir (ej. Procesos concurrentes y cooperantes) o no indicar (ej. Servidores, cuando no se sabe por adelantado quien puede pedir).
- Indirecto: los mensajes no se envían directamente del receptor al emisor sino que se usa una estructura intermedia de colas temporales (mailbox).
    - Permite la relación 1:1, M:1, 1:N, M:M. Una relación M:1 se llama puerto.
    - La asociación proceso-buzón puede ser dinámica o estática.

Formato de mensajes
- Mensajes de cortos de tamaño fijo
- Mensajes de tamaño variable

Cola: FIFO

Exclusión Mutua

En el caso de **send no bloqueante y receive bloqueante**, podría garantizarse la exclusión mutua. El mensaje en el buzón actúa de testigo (token) que se pasa de un proceso a otro. Garantiza la excl mutua en sistemas concurrentes.

- El buzón contiene inicialmente un mensaje, de contenido nulo.
- El proceso que quiere entrar en la sección crítica, intenta recibir el mensaje.
    - Si está vacío se bloquea hasta que llegue (incluso si más de un proceso lo intenta hacer, se bloquean)
    - Si está lleno, lo recibe e ingresa a la sección crítica (sólo uno podrá ingresar al buzón, aunque haya más de uno esperando)
- Cuando sale de su sección crítica, envía un mensaje al buzón.

Problema de los lectores – escritores

Definición del problema:
N lectores pueden acceder a leer simultáneamente. Los lectores sólo leen.
1 escritor puede escribir en cada instante. Los escritores sólo escriben.
Si un escritor está accediendo, ningún lector puede leerlo.

Soluciones:
- Con semáforos:
    - Prioridad a los lectores
    - prioridad a los escritores.
- Paso de mensajes, con prioridad a los escritores.

Concurrencia: deadlock y starvation

Interbloqueo (deadlock)

- Es un bloqueo permanente de un conjunto de procesos que compiten por los recursos del sistema o bien se comunican unos con otros. Se produce cuando cada proceso retiene un recurso y solicita otro.
- Que se produzca o no interbloqueo, depende tanto de la dinámica de la ejecución como de los detalles de la aplicación.
- *Puede haber deadlock tanto en recursos reutilizables como en los consumibles!*

Tipos de recursos

- Reutilizables: puede ser utilizado por el proceso sin que se agote, cuando se libera está disponible para ser usado por otro proceso.
- Consumibles: puede ser creado (producido) y destruido (consumido). Cuando un proceso "adquiere" un recurso, es destruido. Ejemplos: interrupciones, señales, mensajes, buffers de e/s, etc.

Condiciones de interbloqueo

1. Exclusión mutua
2. Retención y espera: un proceso puede retener unos recursos ya asignados mientras espera que se le asignen otros.
3. No apropiación: ningun proceso peude ser forzado a que abandone/libere los recursos asignados.
4. Círculo vicioso de espera: existe una cadena cerrada de procesos, cada uno de los cuales retiene al menos un recurso que necesita el siguiente.

Enfoques de resolución del problema

Principio	Esquema	Ventajas	Desventajas
Prevención (a través de eliminar alguna de las condiciones de deadlock)	Solicitar todos los recursos a la vez	No se necesita apropiar. Funciona bien con procesos que hacen 1 sola ráfaga de actividad	Ineficiente Retraso en inicio del proceso Conocimiento de las necesidades de recursos
	Apropiación	Conveniente con recursos cuyo estado es salvable facilmente	Expulsa más de lo necesario Sujeto a reinicios cíclicos
	Ordenación de recursos	Es viable aplicarlo con chequeos durante la compilación No hace falta procesamiento durante la ejecución	Poco uso de apropiación No permite solicitudes incrementales de recursos
Predicción	**Algoritmo del Banquero**: Búsqueda de un camino seguro	No es necesaria apropiación	Debe conocerse la demanda futura de recursos Los procesos pueden bloquearse por largos períodos
Detección	Comprobación periódica del deadlock	Nunca retrasa el inicio del proceso Facilita la gestión online	Pérdidas por apropiación

Prevención del deadlock

La estrategia de prevención consiste en excluir la posibilidad de interbloqueo. Hay dos métodos básicos: indirectos, buscan impedir la aparición de alguna de las tres condiciones necesarias. Los directos consisten en evitar el círculo vicioso de espera.

- Exclusión mutua: no puede anularse

- Retención y espera: podría prevenirse exigiendo que todos los procesos soliciten todos los recursos que necesitan a un mismo tiempo y bloqueando el proceso hasta que todos los recursos pueden concederse. Es ineficiente.
- No apropiación
    - Si a un proceso se le deniega una nueva solicitud de recursos, podría obligárselo a que devuelva sus recursos anteriores y que los solicite de nuevo a todos cuando lo necesite.
    - Si un proceso requiere de un recurso retenido por otro proceso, el SO podría expulsar al segundo proceso y exigirle que libere sus recursos. Esto funcionaría sólo si ambos procesos tienen diferente prioridad.
- Círculo vicioso de espera: puede prevenirse definiendo una ordenación lineal de los recursos. Si a un recurso se le asignaron recursos de tipo Ri solo podrá pedir los del tipo siguiente a Ri (por ej. Rj). Puede ser ineficiente, retardando procesos y denegando acceso a recursos.

Predicción del deadlock

Se pueden alcanzar las 3 condiciones necesarias para un deadlock, pero se realizan elecciones adecuadas para asegurase que no se llegue nunca a un interbloqueo.
Permite entonces, más concurrencia que la prevención.
Se decide dinámicamente si la petición actual de asignación podría generar un deadlock.

Negativa de iniciación de procesos
Consiste en NO iniciar un proceso si sus demandas pueden llevar a un deadlock.
Un proceso comenzará sólo si puede servirse con los recursos disponibles la demanda máxima de todos los recursos actuales más la del nuevo proceso.
Es poco óptima porque se asume el peor de los casos (que todos necesiten el maximo a la vez).

Algoritmo del Banquero: Negativa de asignación de recursos
- Estado: asignación actual de recursos a los procesos. Formado por vectores: Recursos y Disponible.
- Estado seguro: en el que existe al menos una secuencia que no lleva al interbloqueo (o sea que todos los procesos pueden ejecutarse hasta el final).
- Estado inseguro: aquel que no es seguro.
- Esta estrategia no predice el deadlock con certeza, sino que anticipa la posiblidad de deadlock y asegura que nunca exista tal posibilidad

Se necesitan:
- matriz de demanda,
- matriz de asignación,
- vector de recursos,
- vector de disponible

Requiere:
- que cada proceso presente la demanda máxima de recursos por anticipado;
- que los procesos sean independientes, que no tengan sincronización
- un número fijo de recursos y procesos
- los procesos no pueden finalizar mientras retengan recursos

Detección del deadlock

En lugar de imponer restricciones como las técnicas de prevención, la estrategia de detección no limita el acceso a los recursos ni restringen a los procesos, se concede todo cuanto sea posible, periodicamente el SO ejecuta un algoritmo que permite

detectar la condición de circulo vicioso de espera y decide cómo recuperar la situación.

Algoritmo de detección
Utiliza la matriz de Asignación y el vector Disponible, y una matriz Solicitud (las columnas representan los recursos de cada tipo, y las filas los procesos, cada Qij representa la cantidad de recursos del tipo j solicitados por el proceso i.

1. Se marcan los que no están interbloqueados. Al principio, todos están sin marcar.
2. Se marcan todos los que tienen asignación=0.
3. Se buscan los procesos que estén sin marcar, y que la asignación sea <=al disponible para ese recurso. Si no se encuentra, termina.
4. Si se encuentra algun proceso, se marca, y se incrementa el disponible con el valor de la asignación y se repite el paso 3.

Existe deadlock si y solo si hay procesos no marcados al terminar el algoritmo. Cada proceso no marcado está interbloqueado. La estrategia consiste en encontrar un proceso cuyas solicitudes de recursos pueden satisfacerse con los recursos disponibles, se supone que se conceden, que termina y libera todo.

Recuperación de un deadlock
- Una vez detectado, se peude seguir alguno de estos enfoques:
- Abortar todos los interbloqueados
- Retroceder cada proceso interbloqueado hasta algún punto de control definido previamiente y volver a ejecutar los procesos.
- Abortar sucesivamente procesos interbloqueados hasta que deje de haber deadlock
- Apropiarse de recursos sucesivamente hasta que deje de haber deadlock.

Estrategia integrada de deadlock
- Agrupar los recursos por clases
- Usar la estrategia de ordenación lineal para prevenir circulos viciosos de espera e impedir interbloqueos entre clases de recursos

Deadlock de un solo proceso! (pág 471 Stallings)

Un proceso quiere hacer una lectura de un bloque de datos de un disco/cinta.
Podría hacerse espera activa (se queda colgado hasta que la operación de e/s termine) o bien se lo puede suspender.
El problema es que las operaciones de e/s interfieren en las decisiones de intercambio del SO.
Riesgo de deadlock: si a un proceso que emite una orden de e/s, queda suspendido esperando el resultado, si se lo expulsa antes de comenzar la operación, sale de memoria bloqueado, esperando a que la operación termine. Mientras tanto, la operación de e/s queda bloqueada hasta que el proceso vuelva a memoria.
Cómo se evita: la memoria del usuario implicada en la operación de e/s debe quedar permanentemente en memoria principal, inmediatamente después de emitir la orden de e/s, aunque ésta orden se quede en cola y no se ejecute durante un tiempo!

Problema de la cena de los filósofos

Definición del problema:
Hay 5 filósofos que piensan(), comen().
Hay 5 tenedores, y 5 platos en la mesa.

Cada filósofo usa para comer 2 tenedores, el de la derecha y la izquierda.

Se debe satisfacer la exclusión mutua, el interbloqueo y la inanición.

Solución adecuada: usar semáforos. establecer un máximo de 4 filósofos simultáneos sentados, al menos uno se asegurará de tener los dos tenedores. Esto garantiza que está libre de interbloqueos e inanición.

Gestión de Memoria

Debe satisfacer los siguientes requisitos:
- Reubicación: cuando vuelve a cargarse en memoria un proceso descargado a disco, hay que reapuntar todas las direcciones.
- Protección: contra "interferencia" de otros procesos. Las referencias que hace un proceso deben verificarse para evitar cruces.
- Compartición: debe admitir la flexibilidad para compartir en memoria principal cierta zona entre varios procesos.
- Organización lógica
- Organización física

Partición de la memoria
Técnicas de partición de la memoria

Técnica	Descripción	Ventajas	Desventajas
Partición estática	Particiones estáticas al momento de generaración del sistema	Fácil de implementar, poca sobrecarga al SO	Fragmentación interna
Partición dinámica	Los procesos se cargan en una partición del tamaño exacto	No hay fragmentación interna Uso eficiente de mem principal	Fragmentación externa Se requiere compactación (uso ineficiente del procesador)
Paginación simple	Memoria dividida en frames de igual tamaño. Cada proceso se divide en páginas, la última tiene fragmentación interna. Un proceso se carga situando las páginas en frames libres, pueden ser no contiguos.	No hay fragmentación externa	Hay pequeña fragmentación interna
Segmentación simple	Proceso dividido en segmentos. Se cargan en particiones dinámicas que pueden no ser contiguas	No hay fragmentación interna	Mejora utilización memoria Reduce la sobrecarga comparada con part dinámica
Memoria virtual paginada	Idem, paginación simple, no hay que cargar todo en memoria principal	No hay fragmentación externa Alto grado de multiprogramación	Sobrecarga por gestión compleja de memoria
Memoria virtual segmentada	Idem segmetnación simple, no es necesario cargar todo en memoria principal	No hay fragmentación interna Alto grado de multiprogramación Soporte para protecc y compartición	Sobrecarga por gestión compleja de memoria

- Fragmentación interna: se malgasta el espacio interno de una partición, cuando el bloque dedatos cargado es más pequeño que la partición.

- Fragmentación externa: hay un gran numero de “huecos” pequeños entre particiones; la memoria externa a todas las particiones se fragmenta cada vez más.
- Compactación: de vez en cuando el SO desplaza los procesos para que estén contiguos, de forma que toda la memoria libre quede junta en un bloque.

Algoritmos de ubicación
- First Fit: recorre la memoria y asigna el primero disponible que sea suficientemente grande. Es el más sencillo y suele ser uno de los mejores.
- Best Fit: elige el bloque de tamaño más parecido al solicitado. Es uno de los peores: porque al buscar el bloque más parecido, deja los fragmentos más pequeños posibles, aumentando la fragmentación externa.
- Next Fit: recorre la memoria desde el lugar de la última ubicación y elige el siguiente bloque disponible que sea suficientemente grande. Lleva a asignaciones frecuentes al final de la memoria, requiere compactación más frecuente.
- Worst Fit: elige el más grande, para que quede el mayor espacio disponible

Buddy System (sistema de colegas)

Los bloques se asignan de tamaños variables. Son potencias de 2
Al principio, esta toda la memoria en un bloque. Cuando viene una solicitud, se divide la memoria en 2, y a su vez uno de esos bloques puede volver a dividirse, hasta que el bloque más chico sea menor o igual al tamaño que se busca.
El sistema mantiene una lista de los huecos. Cuando un par está libre, se los junta y se los elimina de la lista y se une en un solo bloque.

Paginación (simple)
- Dirección lógica: referencia a una posición de memoria independiente de la asignación actual de los datos a la memoria, hay que traducirla a física antes del acceso.
- Dirección relativa: caso particular de la dirección lógica, la dirección se expresa como una posición relativa a un punto conocido (en gral al inicio del programa).
- Dirección física (o absoluta): posición real en la memoria principal.

La memoria se divide en particiones de tamaño fijo (frames, marcos).
Los procesos se dividen en porciones (páginas).
Se asigna cada página a un frame libre. Un programa puede ocupar más de una página, y pueden no estar contiguas. *El mecanismo es similar a la partición estática*.
Esto es transparente para el programador.

Las direcciones lógicas del programa tendrán: un número de página y un desplazamiento dentro de la página. Hay que hacer la traducción de direcciones lógicas a físicas.

Ejemplo de cálculos de direcciones y cantidad de páginas Datos: direcciones de 16 bits. Tamaño de página 1Kb. Cálculo: 1Kb = 1024 bytes = 2^{10}. Se necesitan 10 bits para el desplazamiento dentro de la página; en consecuencia (16-10)=6 bits para el número de página. Entonces, el sistema puede tener como máximo, 2^6 = 64 páginas de 1 Kb cada una.

El SO mantiene:
- una lista de marcos libres.
- Una tabla de páginas para cada proceso

Segmentación (simple)
Es otra forma de dividir a un programa: tanto el programa como los datos asociados se dividen en segmentos. No es necesario que tengan todos la misma longitud, aunque tiene una long máxima.

Direcciones: las direcc lógicas tienen dos partes, un número de segmento y un desplazamiento. Por tener segmentos de tamaños desiguales, no hay correspondencia simple entre las direcc lógicas y físicas.

El mecanismo es similar a una partición dinámica. La diferencia es que puede ocupar más de una partición y no es necesario que todas estén en memoria ni contiguas. Es visible para el programador, y se ofrece como ventaja.

El SO mantiene:
- una lista de bloques libres en la memoria
- Una tabla de segmentos para cada proceso

Ejemplo de cálculos de direcciones y segmentos. Datos: direcciones de n+m bits. N bits -> número de segmento M desplazamiento Tamaño máximo del segmento: 2^m.

Memoria Virtual

Las referencias dentro de un proceso son con direcciones lógicas, que se traducen, con lo cual no hay problema con la carga / descarga del proceso.
No es necesario que las páginas o segmentos se encuentren contiguas durante la ejecución.
No es necesario que todas las páginas/segmentos estén cargados en memoria simultáneamente en tiempo de ejecución.

Si el procesador encuentra una dirección lógica que no está en memoria, genera una interrupción que indica un fallo de acceso a memoria (page fault). Hay que traer a la memoria principal el fragmento de proceso que contiene la dirección lógica que provocó el fallo. Después de emitir la solicitud de e/s, el SO puede expedir otro proceso para que se ejecute mientras hace la e/s. Una vez que retorna la e/s de la página, entonces el proceso queda en Listo.

Principio de cercanía: las referencias a los datos y al programa dentro de un proceso tienden a agruparse, por lo tanto es válida la suposición de que durante períodos cortos de tiempose necesitarán sólo algunas páginas en memoria.

Estructura de las tablas de páginas
En gral hay una tabla de páginas por proceso, pero cada proceso puede ocupar una cantidad enorme de memoria virtual.

Pero ... la cantidad de memoria dedicada sólo a la tabla puede ser inaceptable, en consecuencia en general los SO almacenan las tablas de páginas en memoria virtual, en lugar de en memoria real..... (o sea: puede generar un Page Fault la búsqueda en la tabla!!!)
Cuando un proceso se ejecuta al menos una parte de su tabla de páginas debe estar en memoria

Otra forma sería tener un esquema de 2 niveles para organizar grandes tablas de páginas. Hay un directorio de páginas en el que cada entrada apunta a una tabla de páginas. Si el directorio de páginas es X y el tam de página Y, entonces un proceso puede estar formado por hasta X x Y.

Otro enfoque alternativo es el uso de una tabla de páginas invertida (Power PC, AS400). Con este método la parte del número de página de una dirección virtual se traduce a una tabla de hashing. La tabla de hash tiene un punteroa la tabla de páginas invertida, que contiene a su vez las entradas de la tabla de páginas.
Hay una tabla de páginas invertida por cada página de memoria real en lugar de una por cada página virtual. Entonces, se neceista una parte fija de memoria real para las tablas, sin importar cuantos procesos o páginas virtuales se soporten.

Buffer de traducción adelantada (TLB)
Con el modelo explicado antes (la tabla de páginas en memoria virtual), cada rereferencia a una dirección, puede generar dos accesos a la memoria, uno para obtener la entrada a la tabla de páginas correspondiente, y otro para encontrar el dato deseado.
La mayoria de los esquemas resuelven estos dos accesos haciendo uso de una caché especial para las entradas de páginas, llamada TLB (translation lookaside buffer), funciona igual que las otras caché (usando correspondencia asociativa), guardando las entradas de las tabla de página usada más recientemente.

Funcionamiento: hay una búsqueda de una dirección lógica, entonces el sistema:

1. Primero examina la TLB (si hay acierto, tlb hit) se obtiene el número de frame y se completa la dirección real.
2. Si no está en la TLB (fallo, tlb miss), el procesador emplea el número de página como índice para buscar en la tabla de páginas delproceso y examinar la entrada que corresp de la tabla de páginas.
3. Si el bit está activo, entonces está enmemoria, y se puede obtener el número de frame. El procesador actualiza la TLB para incluir esta nueva entrada.
4. Si el bit está inactivo (la página no está en memoria), se produce un Page Fault.

Decisiones acerca del tamaño de página:

- Páginas chicas, implica menos fragmentación interna, mayor es lacantidad de páginas que se necesitan por proceso, un número mayor depáginas por proceso significa que las tablas de páginas serán mayores.
- Tablas más grandes, implican que probablemente no estén en memoria real, en consecuencia, una única referencia puede provocar dos fallos de página (uno para traer la tabla, otro para la página).
- Si son páginas más chicas, va a haber más en memoria, y por el principio de cercanía va a haber menos fallos de página.

Tablas de segmentos en memoria virtual
- Cada proceso tiene su propia tabla de segmentos
- Cada entrada de la tabla de segmentos contiene la dirección de comienzo del segmento correspondiente de la memoria principal y su longitud.
- Se necesita un bit en cada entrada de la tabla de segmentos para indicar si el segmento se encuentra en memoria principal, si el bit está activo,entonces también indica la dirección de comienzo del segmento y la longitud.
- Tiene un bit de control para analizar si el contenido fue modificado desde que se cargó.

Paginación y segmentación combinadas.

- Cuando se combinan ambos métodos, el espacio de direcciones de un usuario se divide en varios segmentos según el criterio del programador.
- Cada segmento se vuelve a dividir en varias páginas de tamaño fijo, que tienen el mismo tamaño que un frame de memoria principal.
- Para el programador una dirección es un segmento + desplazamiento, para el sistema, es un número de página dentro de un segmento y un desplazamiento dentro de una página

Tablas:
Por cada proceso hay una tabla de segmentos y varias tablas de páginas, una por cada tabla de segmento del proceso (como si fueran dos niveles).
Cuando un proceso se ejecuta, un registro contiene la direcciónde l comienzo de la tabla de segmentos para ese proceso.
Búsqueda de la dirección: dada una direcc virtual, el procesador emplea parte del número de segmento como índice en la tabla de segmentos del procecso para encontrar la tabla de páginas de dicho segmento. Y así la parte del número de la direcc virtual se usará como índice en la tabla de páginas para localizar el número de frame que corresponde.

Protección: como cada tabla de segmentos incluye la longitud (además de la direcc base) un programa no puede acceder más allá de los límites del segmento.
Compartimiento: para conseguir compartición, es posible que un segmento se referencie desde las tablas de segmentos de más de un proceso.

Políticas de un SO al respecto de memoria virtual

Lectura:

- Por demanda
- Paginación previa (ojo, no es intercambio!, cuando un proceso se suspende, todas las páginas se vuelan de la memoria principal).

Ubicación: (idem memoria gral)

- Best Fit
- First Fit
- Next Fit
- Sin embargo .... en paginación pura o paginación-segmentación la ubicación carece de importancia.

Políticas de reemplazo de páginas
Qué página reemplazar en la memoria cuando hay que cargar una nueva página. EL objetivo es reemplazar una página que tenga la menor posibilidad de ser referenciada en un futuro cercano.

Asignación / Alcance	Reemplazo Local	Reemplazo Global
Fija	Nro fijo de frames a un proceso La página a reemplazar se elige entre los frames asignados al proceso	IMPOSIBLE!

Variable	El nro de frames asignados al proceso puede cambiar durante la ejecución La página se elige entre los frames asignados en ese momento	La página se elige entre todos los frames disponibles

Alcance
- Reemplazo local: elige entre las páginas residentes del proceso que originó el Page Fault.
- Reemplazo global: todas las páginas en memoria son candidatas a ser reemplazadas.

Con asignación variable y alcance local, cómo hacer para cambiar la asignación (que sea variable!)
Algoritmo PFF, frecuencia de fallos de página. (Page Fault Frequency)
Se asocia un bit de uso a la página. Se pone en 1 cuando se accede a la página. Cuando hay Page Fault, el SO anota el tiempo transcurrido desde el último PageFault para ese proceso. Se compara contra un umbral F, si es menor, entonces la página se añade al conjunto residente del proceso, sino, se descartan las páginas con bit en uso 0 (y entonces se reduce el conjunto residente!)

Algoritmos básicos

Algoritmo	Descripción	Características
Optimo	Se reemplaza la página que tiene que esperar el mayor tiempo para ser referenciada.	Es **inimplementable**! Sirve para comparar los otros algoritmos
LRU	Reemplaza aquella página que no ha sido referenciada en más tiempo	Por principio de cercanía, debería ser la de menos posibilidad de ser referenciada. Difícil de implemnetar
FIFO	Buffer circular (round robin), reemplaza la página que estuvo más tiempo en memoria	Fácil de implementar, pero probablemente poco efectivo
Clock	Asocia un bit adicional a cada frame (bit de uso). Al cargarse inicialmente, bit=0. Cuando se referencia despues, bit=1. Es un buffer circular, pero elige el primer 0 que encuentra. Si da una vuelta, cambia todos los 1 por 0.	

Anomalía de Belady: con más frames, tengo más page faults!

Políticas de vaciado
- Vaciado por demanda: una página se escribe a memoria secundaria sólo cuando ha sido elegida para reemplazarse.
- Vaciado previo: escribe las páginas modificadas antes de que se necesiten sus frames, para que las páginas puedan ser escritas por lotes.

Control de carga (grado de multiprogramación)
Es crítico para la eficiencia del sistema.
- Si hay pocos procesos: probablemtne haya momentos donde todos están bloqueados,entonces se tarda mucho tiempo en intercambios.
- Si hay demasiados, el tamaño medio del conjunto residente no es adecuado, se producen muchos fallos de página -> hiperpaginación (trashing).

Suspensión: cómo elegir los procesos para bajar cuando hay que reducir la multiprogamación... sacar a:
- Los procesos con menos prioridad
- Procesos con fallos de página
- Último proceso activado
- Proceso con el conjunto residente más pequeño
- El proceso mayor.
- Procesos con la mayor ventana de ejecución restante.

Planificador de monoprocesadores

Tipos de planificación
- De largo plazo: añadir procesos al conjunto de procesos a ejecutar. Es decir, qué programas son admitidos por el sistema. Determina el nivel de multiprogramación.
- De mediano plazo: agregar procesos al conjunto de de los que se encuentran parcialmente en memoria. Es parte del proceso de intercambio.
- De corto plazo: decisiones acerca de qué proceso disponible será ejecutado por el procesador.
- De entrada/salida: qué solicitud de e/s pendiente procesada.

Políticas de planificación

Algoritmo	Apropiación	Selección	Características
FCFS	No	FIFO: El primero que llega	Mínima sobrecarga Penaliza cortos Favorece a los de cpu y no a los de e/s No inanición
Round Robin	Preemptive, por quantum	Cola circular	Equitativo Rendimiento pobre de los procesos de e/s, puede solucionarse con VRR (Virtual Round Robin, que combina colas FCFS con prioridades) No inanición
SPN	No	Primero los más cortos	Penaliza largos Hay que conocer de antemano el tiempo de ejecución. Posible inanición
SRT	Preemptive, en la llegada de un proceso a la cola de Listos.	Al que le falta menos para terminar	Penaliza largos Posible inanición
HRRN Highest Response Ratio Next	Preemptive	Elige al que tiene máx tiempo esperado de servicio (tiempo consumido + tiempo servicio / tiempo servicio)	Equitativo, evita la inanición incluyendo el tiempo esperando No inanición
Feedback	Preemptive, por quantum	Penaliza a que estuvieron ejecutando más tiempo. Múltiples colas con prioridades. El proceso arranca en RQ0, a medida que ejcuta va bajando.	Puede favorecer a los que tienen carga de e/s Posible inanición
FSS Fair Share Scheduling (para grupos, conjuntos de procesos)		Asignar menos recursos a los usuarios que consumieron más de lo que les corresponde y más a los que usaron menos.	Planificación por prioridades, se tiene en cuenta el uso que hizo el grupo de la CPU.
SJN; Shortest Job Next	Preemptive		

Apropiación

- No Preemptive / Preferente: el proceso que pasa al estado Ejecución, continúa ejecutando hasta que a) termina o b) se bloquea (en espera de una e/s). Implican menor “costo” pero son menos equitativas.
- Preemptive / Preferente: el proceso que está ejecutando puede ser interrumpido y pasado a Listo por el SO. La apropiación puede producirse por que un proceso se bloquee o periódicamente (reloj). Implican mayor esfuerzo del procesador cambiando procesos, pero son más justas.

Planificación de multiprocesadores

Relación de los sistemas multiprocesador:
- Agrupación o multiprocesador débilmente acoplado o distribuido
- Procesadores especializados funcionalmente
- Multiprocesador fuertemente acoplado: es un conjunto de procesadores que comparten una memoria principal común

Grado de sincronización
- Fino: paralelismo inherente a un único flujo de instrucciones
- Medio: procesamiento paralelo o multitarea dentro de una aplicación individual. Más complejo que el paralelismo con hilos.
- Grueso: multiprocesamiento de procesos concurrentes en un entorno multiprogramado. Poca sincronización.
- Muy grueso: proceso distribuido. Poca sincronización.
- Independiente: procesos no relacionados. No hay sincronización explícita de los procesos.

Consideraciones de diseño
- Asignación permanente / estática.
    - Poca eficiencia si no hay una cola común (si hay uno con pendientes y otro desocupado)
    - Dos métodos de asignar:
        - Maestro/esclavo
        - Arquitecturas simétricas
- Asignación dinámica
- En general no se asignan en forma dedicada,sino que hay una cola única para todos los procecsadores o se usa un esquema de prioridades (varias colas por prioridad).
- Uso de multiprogramación en procesadores individuales (sobre todo cuando la asignación es estática,vale la pena?)
- Expedición: selección real del proceso a ejecutar. La planificación no tiene sentido que sea tan compleja como en un monoprocesador.
- Manejo de hilos: es mucho mas potente que en un sistema monoprocesador! (porque hay paralelismo real).
    - Reparto de carga: los procesos no se asignan a un procesador particular, se mantiene una cola global de hilos listos y cadaprocesador selecciona uno de los hilos cuando esté disponible.
        - La carga se distribuye uniformemente entre los procesadores
        - No es necesario un planificador centralizado.
        - La cola central (en memoria compartida) debe tener exclusión mutua.
        - Si se reaunda la ejecución de cierto hilo, es improbable que vuelva almismo procesador, no se utilizaría la caché local.
        - Si se requiriera alto grado de coordinación entre los hilos, los intercambios pueden afectar al rendimiento.

        - 3 versiones diferente:
            - FCFS
            - Primero el de menor numero de hilos
            - Primero el de menor número de hilos, preemtpive.
    - Planificación por grupos: se planifica un grupo de hilos para su ejecución en un conjunto de procesadores
        - Si procesos próximos se ejecutan en paralelo, podrian reducirse los bloqueos por sincronización
        - Minimiza los intercambios de procesos, es efectiva cuando hay gran sincronización entre los hilos de un proceso.
    - Asignación dedicada de procesadores: opuesto a reparto de carga, se le asigna un procesador a cada hilo. Es una forma extrema de la planificación por grupos.
        - Si un hilo se bloquea, el procesador queda desocupado, no hay multiprogramación.
        - Como no hay intercambios, se acelera la ejecución del programa (salvo en elmomento de e/s, tiene elprocesador dedicado!)
    - Planificación dinámica: el número de hilos de un programa puede cambiar durante la ejecución. Cuando un proceso pide uno o más procesadores:
        - Si hay procesadores desocupados, se le asignan
        - O al recién llegado se le asigna un procesador individual, que se le quita a otro que tenga más de uno asignado.
        - Si no puede satisfacerse el pedido, se lo deja en cola hasta que alguno anule la petición o pase a estar disponible.

Planificación en tiempo real

Es el tipo de procesamiento donde la exactitud del sistema no depende sólo del resultado lógico del cálculo sino también del instante en que se obtenga.
No preocupa la velocidad absoluta, sino completar (o iniciar) las tareas en el momento más adecuado, ni antes ni después.

Características de un SO en tiempo real

- Determinismo: cuando realiza las operaciones en instantes fijos o intervalos de tiempo predeterminados. Se analiza cuál es el retardo máximo que tiene el SO para responden a las peticiones.
- Sensibilidad: cuánto tiempo tarda el SO en reconocer una interrupción. Es la cantidad de tiempo necesario para iniciar y ejecutar la gestión de interrupción (ISR, Interrupt Service Routine)
- Control del usuario: el usuario debe dar control preciso sobre las prioridades de las tareas.
- Fiabilidad: un fallo transitorio puede tener consecuencias muy graves.
- Tolerancia a fallos: capacidad del sistema de conservar la máxima capacidad y datos en caso de fallo. Es estable cuando puede cumplir los plazos de las tareas críticas y de mayor prioridad.

Determinismo + sensiblidad = tiempo de resupuesta a sucesos externos

El corazón de un sistema de tiempo real es el planificador de tareas corto plazo.
En el diseño es importante la equidad y la reducción del tiempo medio de respuesta.

Se combinan la prioridades con interrupciones de reloj. Los instantes de apropiación (preemptive!) se original a intervalos regulares, cuando llega el mometno, se expulsa a la tarea que está ejecutnado si existe otra esperando que tenga mayor prioridad.
Si hubiera algunas aún más criticas, podría usarse apropiación inmediata.

Algoritmos de planificación en tiempo real
- Métodos con tablas estáticas o periódicas: aplicable a tareas periódicas, es predecible pero inflexible.
- Preferentes con prioridades estáticas: hace análisis estático pero no planifica, se usa el análisis para determinar las prioridades para un sistema preferente con prioridades.
    - algoritmo monónoto de frecuencia. (RMS, Rate Monotonic Scheduling). Asigna prioidades en función a sus períodos, período=tiempo que transcurre entre que llega la tarea y la siguiente llegada de la misma tarea. La tarea con mayor prioridad es la que tiene período más corto, y asi.
- De planificación dinámica; se determina la viabilidad en tiempo de ejecución. Se acepta una nueva tarea sólo si es factible de ejecutarse.
- Dinámicos del mejor resultado: el más usado. La prioridad se asigna a la llegada, como son tareas a-periódicas,no hay forma de saber de antemano (esta'ticamente) su prioridad. No se sabe si puede cumplirse la restriccion de tiempo. Fácil de implementar.

De cada tarea, más allá de la prioridad, se debería saber (para poder planificarla adecuadamente).
- Instante en que está lista
- Plazo de comienzo y finalización
- Tiempo de proceso
- Exigencias de recursos
- Prioridad
- Subtareas?

Gestión de Entrada – Salida

Organización de las funciones de E/S
- E/S programada: el proceso espera a que termine la operación para seguir
- E/S dirigida por interrupciones: continúa la ejecución de las instrucc siguientes (del mismo proceso o el proceso se suspende y se sigue con otro) y el módulo de e/s lo interrumpe cuando completa su trabajo.
- DMA: Acceso directorio a la memoria, un módulo DMA controla el intercambio de datos entre la memoria principal y el módulo de e/s. Se puede mover un bloque entero de datos sin que intervenga el procesador, excepto al inicio o al final. El módulo DMA toma control del bus cuando el procesador no lo necesita, o bien lo obliga a suspender momentáneamente su operación (robo de ciclo).

Estructura lógica de las funciones de E/S
- E/S lógica: el módulo de e/s logica trata al dispositivo como un recurso lógico y no se preocupa de los detalles de control real del dispositivo. Este modulo se ocupa de las funcioens generales (Abrir, Cerrar, Leer, Escribir).
    En los dispositivos de almacenamiento secundario, se incluyen 3 niveles que "abren" más el nivel de e/s lógica
    - Gestión de directorios: traducción de nombres simbólicos a identificadores que apuntan al archivo o a una tabla de indices (inodos?).

    - Sistema de archivos: se encarga de la estructura lógica de los archivos y las operaciones (abrir, cerrar, ...)
    - Organización física: las referencias lógicas a archivos se traducen a direcciones físicas. También gestiona el buffer (almacenamiento intermedio)
- E/S con dispositivos: las operaciones se conviertenen secuencias adecuadas de instrucciones de e/s y órdenes para el canal y el controlador.
- Planificación y control: la planificación y puesta en cola de las operaciones y el control se hacen en este nivel

Almacenamiento intermedio de la E/S
Cuidado,... puede haber deadlock de un solo proceso, debido al uso del almacenamiento intermedio.

Tipos de dispositivos:
Orientados a bloque: almacenan los datos en bloques, normalmente de tamaño fijo, hacen las transferencias de un bloque cada vez. (discos, cintas)
Orientados a flujo: transfieren datos como una serie de bytes, no tienen estructura de bloques. (terminales, impresoras, puertos de comunicación)

Esquemas de almacenamiento intermedio

Esquema	Características	Descripción
Memoria intermedia sencilla	Lectura por adelantado (anticipada) Complica al SO porque debe guardar constancia de las asignaciones de la mem intermedia del sistema a procesos del usuario	Cuando se completa la transferencia, se pide otro bloque inmediatamente, esperando que el bloque se necesite más adelante. El SO puede expulsar al proceso porque la op de entrada no tiene lugar en el espacio del proceso
Memoria intermedia doble	Un proceso puede llenar/vaciar una memoria intermedia mientras que el so llena/vacía el otro	El doble almacenam asegura que el proceso no tiene queesperar en la e/s Es mas rapido que el sencillo, pero mas complejo Puede ser inapropiado si el proceso lleva a cabo rapidas ráfagas de e/s (hay que ir al ritmo del proceso)
Memoria intermedia circular	Es un modelo de productor/consumidor con memoria intermedia ilimitada.	Se puede solucionar el tema del ritmo, agregando más memorias intermedias.

Planificación de discos

Funcionamiento del disco

Velocidad de giro, constante
Una pista tiene muchos sectores.
Si un disco tiene muchos platos, todas las cabezas se mueven juntas.
La combinación de todas las pistas en la misma posición relativa sobre el plato se llama cilindro.

1. Colocar la cabeza en la pista deseada. Para leer o escribir, la cabeza se debe colocar en la pista deseada, al comienzo del sector. Hay que mover la cabeza para elegir la pista. Tiempo para posicionar la cabeza en la pista es el **tiempo de búsqueda**.
    a. Es el tiempo de arranque inicial + tiempo en recorrer las pistas.

2. Esperar a que el sector se aline con la cabeza, es el **retardo de giro** o **latencia de giro**.

Cálculo del retardo de giro (ejemplo) Velocidad de rotación = 10.000 rpm (revoluciones por minuto). Por regla de tres, 10.000 rev = 60 seg => 1 rev = 0,006 seg = 6 ms

Tiempo de acceso = tiempo que se tarda en llegar a la posición de lectura/escritura. Es tiempo de búsqueda + retardo de giro.

Cálculos sector lógico Sector lógico -> (CIL, CAB, SECT). N = (A, B, C) A = parte entera [sect_lógico / (sectores . cabezas)] B = parte entera [resto A / sectores] C = resto B Ejemplo: sect lógico 1301. Datos: 3 platos -> 6 cabezas, 100 sectores cada cilindro A = 1301 / (100.6) = 2, resto 101. B = 101/ 100 = 1, resto 1. C = 1 N = (2, 1, 1)

Algoritmos de planificación de disco

Algoritmo	Descripción	Características
RSS, aleatorio	El peor!	Se usa para comparar con los otros.
FIFO	Primero en entrar, primero en salir	El más justo de todos
PRI	De acuerdo a la prioridad del proceso	El control es fuera de la gestiónde la cola del disco Poco favorable para BD
LIFO	Ultimo en llegar, primero en salir	Maximiza uso recursos y cercanía Implica pocos o nulos movimientos del brazo en arch secuenciales Posibilidad de inanición
SSTF	Primero el más corto (cercano). En ambos sentidos. Busca minimizar el movimiento	Aprovechamiento, colas pequeñas
SCAN	Recorre el disco de un lado a otro (hasta el extremo del disco, despues, después todo para abajo)	Buena distribución Favorece a los trabajos con pistas cercanas a los cilindros mas interiores y exteriores
LOOK	Idem SCAN pero no va hasta el extremo sino hasta el ultimo pedido.	
C-SCAN	Recorre el disco de un lado a otro, **en un solo sentido**	Poca variabilidad de servicio, equitativo
SCAN N pasos	SCAN pero de N registros a la vez	Garantia de servicio Se procesan de a paquetes para evitar “pegajosidad” y que la cabeza esté en el mismo lugar todo el tiempo Usa subcolas de long n
FSCAN	Idem SCAN N pero con N=longitud de la cola al comienzo del scan	Sensible a la carga Usa dos subcolas, las nuevas solicitudes quedan en la 2 mientras se procesa la 1.

RAID

Nivel	Categoría	Descripción y uso	Tasa pedidos (lect/escr)	Tasa transferenc (lect/escr)
0	Bandas	No reduncia Alto rendimiento, app no críticas Lasbandas se distribuyen por disco, si los datos están en discos diferentes, acceso en paralelo	Dist pequeñas: Excelente	Dist grandes: Excelente
1	Espejo	Copia espejo Duplica todos los datos. Archivos críticos. Cualquier disco puede servir el pedido. La escritura es doble, en los 2 discos.	Bueno/justo	Justo / Justo
2	Acceso paralelo	Redundancia por Hamming	Pobre	Excelente
3		N/A El nro de discos redundantes es log del nro de discos de datos.		
		Paridad por intercalación de bits Sólo un disco redundante, no importa cuantos discos de datos. Cualquier transferencia involucra a más de un disco, acceso paralelo.	Pobre	Excelente
4	Acceso independiente	Paridad por intercalación de bloques Un solo disco de paridad. Cada disco opera independiente, se puede servir pedidos en paralelo. En cada escritura hay que actualziar los datos y los bits de paridad.	Excelente / justo	Justo / Pobre
5		N/A		
6		Paridad por intercalación distribuida de bloques Distribuye las bandas de paridad en todos los discos.	Excelente / justo	Justo / Pobre
		Paridad por intercalación doblemente distribuida de bloques Dos calculos distintos de paridad, con n discos de datos, se necesitan n + 2 discos. Alta disponibilidad de datos, con gran penalización de escritura (cada escritura afecta 2 bloques de paridad)	Excelente / pobre	Justo / Pobre

Código de Hamming: detectar 2 bits erróneos, corregir hasta 1 bit erróneo.

Caché de disco

Es una memoria intermedia (buffer) que reduce el tiempo medio de acceso a la memoria aprovechándose del principio de cercanía.

Cuestiones de diseño

- Cómo pasarle los datos cuando hay un acierto
    - Se transfiere el bloque de datos de la memoria principal al proceso
    - Se le pasa unpuntero a la entrada apropiada de la caché de disco
- Estrategia de reemplazo
    - LRU (Least Recently Used), se reemplaza el bloque que permanece en la caché con menor cantidad de referencias. Organización: pila, cuando se referencia se pone arriba, se elimina el del fondo, abajo.
    - LFU, Least Frequently Used, se elimina el bloque que sufrio la menor cantidad de referencias. Se necesita un contador.
    - Reemplazo en función a la frecuencia: pila como LRU, pero con dos secciones. Si hay acierto, se pone en la arriba. Si el bloque estaba en la secc nueva, su contador no se incrementa, sino, se incrementa. Si se

produce una falla, se elimina el bloque conel menor contador y que no esté en la secc nueva. Podría hacerse también con 3 secciones.

- Rendimiento: afectado por la tasa de fallas, depende del comportamiento de la cercanía delas referencias a disco, el algoritmo de reemplazo, etc.

Gestión de archivos

Campos, registros, archivo, base de datos.

Arquitectura de un sistema de archivos

(capas)

- Visto desde el programa de usuarios: (Pila / Secuencial / Secuencial indexado / indexado / hashing): método de acceso.
- E/s lógica: permite a los usuarios y programas acceder a los registros
- Supervisor básico de e/s: responsable de iniciar y terminar todas las e/s
- Sistema de archivos básico / nivel de e/s física: interfaz primaria con entorno exterior del sistema
- Gestor del dispositivo (disco, cinta): device drivers.

Organización de archivos (estructura lógica de los registros)

Tipo	Características	Registros
Pila	Se recogen en el orden que llegan. Acceso: por búsqueda exhaustiva	Sin estructura
Secuencial	Es la unica guardable en cinta/disco. Acceso: búsqueda secuencial que corresponda con la clave	Formato fijo, un campo es la clave Para agregar, se ponen separados, despues se mergea.
Secuencial indexado	Idem, se agrega: Indice para acceso aleatorio y un archivo de desbordamiento (overflow)	Lo que se agrega se pone en el file de desbordamiento, apuntado desde su registro predecesor. Indice: campo clave + ptr a archivo principal. De vez en cuando se combina y rearma el file
Indexado	Múltiples indices, uno por cada campo usado para la busqueda.	No hay concepto de secuencialidad y clave unica, sólo se accede a través de sus índices.
Directo o hashing	Se requiere un campo clave en cada registro. No hay ordenación secuencial.	

Agrupación de registros en bloques.

Métodos de agrupación	**Registros**	**Desperdicio**	**Utilización**
Fijos	Registros de longitud fija	Fragmentación interna (en el último bloque)	Secuenciales
Longitud variable con tramos	Registros de long variable Si ocupa dos bloques, se usa un puntero al siguiente	Sin fragmentación.	Difícil de implementar, cuando ocupa dos bloques hay 2 lecturas.
Longitud variable sin tramos	Registros de long variable No se dividen en tramos	Se desperdicia mucho espacio si el registro siguiente es mayor que el tamaño restante	

Asignación de archivos:

- Asignación previa de todo el espacio
- Asignación dinámica

Tamaño de la sección

- Secciones contiguas variables y grandes: mejor rendimiento, tablas más pequeñas, espacio difícil de reutilizar. Es importante la fragmentación; algoritmos posibles de asignación First Fit, Best Fit, Next Fit.

- Bloques: más flexibilidad, pero puede necesitar tablas más grandes

Métodos de asignación de archivos

	Contigua	**Encadenada**	**Indexada**	
Descripción	Uno conjunto contiguo de bloques.	Asignación con bloques individuales, cada uno tiene un puntero al siguiente bloque de la cadena.	La tabla de asignación de archivos contiene un índice separado de un nivel para cada archivo, el indice tiene una entrada para cada seccion asignada al archivo	
Recomendado para	Secuencial	Secuencial	Secuencial Directo (hashing)	
Asignación previa	Necesaria	Posible	Posible	
Secc tamaño fijo o variable	Variable	Bloques fijos	Bloques fijos	Variable
Tamaño sección	Grande	Pequeño	Pequeño	Medio
Frecuencia asignación	Una vez	Media - baja	Alta	Baja
Tiempo para asignar	Medio	Largo	Corto	Medio
FAT	Una entrada	Una entrada	grande	medio
Problemas	Fragmentación externa. Hay que compactar		Elimina fragmentación externa	Mejora la cercanía

Gestión del espacio libre
Tabla de asignación de disco
Tabla de asignación de archivos

Técnicas:
- Tablas de bit: usa un vector que contiene un bit por cada bloque de disco. 0=libre, 1=usado. Es fácil de encontrar un bloque contiguo de datos de bloques disponibles. Como puede ocupar mucho (tam disco en bytes / 8 x tam bloque), la alternativa es ponerla en disco, pero seguiría pensando mucho, entonces se mantienen estructuras que resumen el contenido de subrangos de la tabla de bits.
- Secciones libres encadenadas: las secc libres se encadenan como una lista mediante un ptr y un valor de long de espacio libre de la secc. Genera fragmentación.
- Indexación: trata al espacio libre como si fuera un archivo, y usa una tabla índice. Sólo puede trabajr con secc de tamaño variable porefeiciencia. Hay una entrada en la tabla para cada sección libre del disco.
- Lista de bloques libres: cada bloque tiene asignado un nro secuencia y la lista de nros de todos los bloques libres se mantiene en una sección reservada del disco. Si se usa un numero de bloque de 32 bits, se penaliza entonces 4 bytes por cada bloque de 512 bytes. El espacio es muy chico, y para almacenar la lista en mem principal se podría usar la lista como una pila con los primeros miles de elementos cargados en mem, o como una cola FIFO, con un par de miles de entradas entre el principio y el final.

I-nodos
- Todos los archivos se contemplan por Unix como un flujo de bytes.
- Todos son adminsitrados por el SO por medio de i-nodos.
- Un i-nodo (nodo indice) es una estructura de control que contiene la info clave del archivo que necesita el SO.
- Varios nombres pueden asociarse a 1 inodo, pero un inodo solo puede asociarse con 1 archivo. Cada archivo solo puede ser controlado por un inodo.

Por lo general se necesitran dos lecturas a disco para acceder a un determinado archivo: una al i-nodo del archivo y otra al bloque del archivo.

En UNIX se usa asignación en bloques, dinámica y a medida que se necesita, no se usa asignación previa. No se requiere que los bloques estén contiguos.
Se usa indexación, con parte de la info almacenada en el inodo.
Entre otros datos, el inodo tiene 39 bytes de info de dirección, organizada como 13 direcciones (punteros) de 3 bytes cada uno (3 bytes x 13 = 39 bytes)

Directo	Primeras 10 direcciones	Apuntan a los primeros 10 bloques de datos del archivo
Indirecto simple	11	Apunta a un bloque de disco que tiene la siguiente parte del indice. Cada bloque ind simple tiene los ptrs a los siguientes bloques del file
Indirecto doble	12	Apunta a un bloque que tiene la lista de bloques indirectos simples adicionales
Indirecto triple	13	Apunta a unbloque ind triple, conun 3er nivel de indexación, que a su vez apuntan a ind dobles.

Calculo del tamaño máximo de un archivo:
Datos: Ptr directos 12, ind simple 1, doble 1, triple 1
tamaño de bloque = tamaño del sector, 8 Kb (8192 bytes).
Puntero a disco = 32 bits: 8 bits identifican disco, 24 bits identifican bloque físico
Tamaño puntero 32 bits = 4 bytes

Cant en Bytes = cant_ptr . (Tbloque / TamPtr)^n . Tbloque

Directos = 12. (8192 / 4)^0 . 8192 = 98304 = 96 Kb
Indirectos simples = 1. (8192 / 4)^1 . 8192 = 16.777.216= 16384 Kb = 16 Mb
Indirectos dobles = 1. (8192 / 4)^2 . 8192 = 32 Gb
Indirectos triples = 1. (8192 / 4)^3 . 8192 = 64 Tb
Bytes = Sumatoria total!

Cuántos accesos al disco hacen falta para localizar el byte enla posición x: tantos como accesos a los inodos se necesiten (busco la suma de bytes hasta posic x).
Máx tamaño File System = 2^{Tam_ptro} . Tbloque
tamaño File System = 2^{24} . 8192 = …

Procesamiento distribuido

Tendencia hacia el proceso de datos distrbuido
- Soporte en la arquitectura de comunicaciones
- Sistemas operativos de red
- Sistemas operativos distribuidos

Procesamiento cliente / servidor
- Elementos
- Cliente
- Servidor
- Middleware: son interfaces y protocolos estándar.
  conjunto de controladores, API u otro sw que mejora la conectividad entre las aplicaciones entre cliente y servidor. Para resolver problemas de interoperación entre diversos vendors y protocolos.
- Red
- Aplicaciones se distribuyen
- Bases de datos se centralizan

Clases de aplicaciones cliente/servidor

- Proceso basado en máquina central (host): no es realmente c/s, sino entorno tradional, mainframe.
- Proceso basado en servidor
- Proceso basado en cliente: se procesa todo ahí salvo las validaciones de datos y funciones de la bd.
- Proceso cooperativo: aplicación optimizada, aprovechando la potencia de los equipos y la distribución de los datos.

Arquitectura
En dos capas

- Capa cliente
- Capa servidor

En tres capas

- Capa cliente
- Capa intermedia (servidor de aplicaciones)
- Capa servidor (backend, de datos)

Consistencia de la cache de archivos

El rendimiento de la e/s se puede degradar por el retardo introducido por la red
Los equipos individuales pueden usar cachés de archivos, pero hay que administrar cuidadosamente la consistencia.
Al momento de una petición, hay que chequear la cache del puesto (trafico de archivo), luego al disco (trafico de disco), o al servidor (tráfico del servidor). Una vez en el servidor, examinar primero su cache y si hay fallo, acceder a disco.
Si las caches tienen copias exactas, hay consistencia de caches.
Para evitar la inconsistencia (se cambian datos remotos y no se desechan las copias locales), se pueden usar técnicas de bloqueo (garantiza consistencia a costa de rendimiento y flexibilidad), sino que todos habran como lectura, y cuando se abre como escritura, se notifica que deben reescribirse los bloques alterados inmediatamente.

Paso distribuido de mensajes

En proceso distribuido se suele dar que los equipos NO compartan memoria principal, son equipos aislados, entonces para comunicar procesadores NO se puede usar semáforos ni usar un área de memoria común, entonces se usa una técnica de paso de mensajes.

Send / Receive
Especificar el destino e incluir el contenido del mensaje
Las solicitudes de servicio peuden ser primitivas (funcion a realizar) y parámetros
La recepción tiene un área de almacenamiento intermedio (un buffer) donde se alojan losmensajes que llegan.

Mensajes fiables / no fiables: so se garantiza la entrega, se debe usar un protocolo fiable.

Bloqueante / no bloqueante

- Primitivas NO bloqueantes o asíncronas:
    - hay empleo eficiente y flexible del paso de mensajes
    - programas dificiles de probar y depurar, con secuencias irreproducibles dependientes del tiempo y la ejecución
- Primitivas bloqueantes o síncronas:

- No devuelve control hasta que el mensaje se transmite (servicio no fiable) o hasta que se hace obtenido un acuse de la recepción (servicio fiable).

RPC: llamadas a procedimiento remoto
Permite que programas de máquinas diferentes interactúen mediante una semántica de llamados/retornos a simples procedimientos como si estuvieran ambos en la misma máquina.

Es un refinamiento de un sistema fiable y bloqueante.
Formato estándar de mensajes, el llamado se hace indicando el nombre del procedimiento, los argumentos pasados y devueltos (paso de parámetros por valor o como puntero por referencia, más dificiles de implementar)

Enlaces:
- Persistentes: la conexión lógica se establece entre dos procesos al momento de la llamada y se pierde tan pronto como se devuelvan los valores
- No persistentes: se establece y se mantiene después de que el procedimiento termina.

- Sincronas (bloq): se comporta similar a una subrutina. Es fácil de programar por ser predecible, pero no explica por completo el paralelismo de las app distribuidas.
- Asíncronas: se consigue más paralelismo. No bloquean al llamador. Para sincronizar se puede iniciar y terminar la conexión con otra app o bien que el cliente emita una cadena de rpc asigncronca seguida de una última síncrona.

Agrupaciones
Son una alternativa al SMP para dar alto rendimiento y alta disponibilidad.
Cada equipo es un nodo

Ventajas
- Escalabilidad total: pueden superar facilmente las máquinas autónomas más grandes
- Escalabilidad intremental: se pueden añadir nodos fácilmente.
- Alta disponibilidad
- Mejor relación rendimiento – precio

Métodos de agrupación

Método	Descripción	Ventajas	Limitaciones
Espera pasiva	El secundario arranca si falla el primario	Facil de implementar + disponibilidad no mejora rendimiento	Alto costo
Secundaria activa	El secundario también procesa	Costo menor	Mas complejidad
Servidores separados	Cada uno tiene sus propios discos y los datos se copian continuamente.	Alta disponibilidad	Sobrecarga en servidor y red por copiado
Servidores conectados a discos	Cada uno tiene los suyos, pero están conectados, si uno falla, otro servidor pasa a estar a cargo	Reduce sobrecarga y uso de red por no espejar discos	Necesita RAID
Servidores compartiendo discos	Multiples equipos comparten acceso a disco	Baja sobrecarga Reducción de riesgo de caída por falla disco	Soft para gestionar bloqueos RAID

Gestión de fallos: depende del método de agrupación utilizado
- Resistencia a fallos (failover): función de intercambiar una aplicación y los datos de un sistema fallido por uno alternativo.
- Restauración de fallos (failback): Restaurar las aplicaciones y datos al sistema original cuando se reparó

Equilibrio de carga:
- Compilador paralelo: se determinan en tiempos de compilación qué partes de la aplicación se pueden ejeuctar en paralelo.
- Aplicaciones paralelas: se usa el paso de msg para mover datos
- Computación paramétrica: algoritmo que debe ejecutarse muchas veces con datos diferentes

Proceso paralelo

Arquitectura
- Único punto de entrada para el usuario
- Única jerarquía de archivos
- Único punto de control
- Única gestión de red virutal
- Único espacio de memoria
- Único sistema de gestión de trabajos
- Única interfaz de usuario
- Único espacio de e/s y de procesos
- Puntos de comprobación
- Migración de procesos.

Gestión distribuida de procesos

Motivos de la migración de procesos
- Compartimiento de carga
- Rendimiento de las comunicaciones
- Disponibilidad
- Utilización de capacidades especiales
- Quién inicia el proceso de migración? Una unica entidad? El destino puede “opinar” acerca de recibir unproceso migrado?
- Desalojo: un sistema destino podría rechazar una migración.
- Transferencias preferentes / no preferentes: transferir un proceso parcialmente ejecutado o con la creación completada, o bien (la no preferente) involucraar procesos que no comenzaron la ejecución (con lo cual no es necesario transferir el estado delproceso)

Cuando se migran procesos que ya ejecutaron: Se migra la imagen del proceso, se destruye en el sistema de origen y se crea en el destino. Se mueve y no se duplica. El problema está en el espacio de direcciones y los archivos abiertos.

Estrategias:
- Transferencia (completa): se transfiere todo en el momento de la migración. Elegante, pero puede ser costoso sin necesidad.
- Copia anticipada: se ejecuta en origen mientras se copia a destino. Loque se modifico se copia de nuevo
- Transferencia (modificado): se transfiere solo lo modificado, y el resto bajo demanda
- Copia por referencia: idem, pero se desplazan las páginas cuando se las referencia

- Volcado (flusing): se eliminan las páginas del proceso de la memoria de origen, volcando a disco las modificadas. Se accede a cada página según se necesite y desde el disco y no de memoria.

Problemas de concurrencia (determinar el estado global)
Aparecen los mismos problemas de concurrencia que antes (exclusión mutua, interbloqueo, inanición), las estrategias acá varían por el hecho de no tener un "estado global" del sistema en un instante dado. O sea, un sistema o proceso no puede conocer el estado de los otros.
La complicación surge (entre otras cosas) por el retardo temporal de las comunicaciones.

Algoritmo de instantáneas distribuidas

- Los mensajes se entregan en el mismo orden que se envían, y no se pierden mensajes.
- Los mensajes se propagan
- Cualquier proceso inicia el algoritmo mandando un mnesaje marcador por un canal que comunica con otros (n) procesos. Cada uno registra su propio estado y el de todos los canales entrantes.
- Se obtiene un estado global consistente cuando cada uno envía por todos los canales salientes los datos del estado que registró y retransmite los que recibió.

Exclusión mutua distribuida

Los algoritmos dependen del paso de mensajes (y no del acceso a memoria porque no tienen).

Algoritmo centralizado: hay un nodo desginado como control del acceso a todos los objetos compartidos. Cuando se quiere acceder a un recurso crítico, se emite un Pedido, obtiene una Respuesta y luego cuando termina de usarlo, envia un mensaje de Liberación.

Algoritmo distribuido para exclusión mutua
- Todos los nodos pueden decidir
- Cada uno tiene info parcial
- Como no hay reloj comun, para sincronizarse se usa una ordenación de sucesos (suceso=envio). Se usan "marcas de tiempo". Cada sistema mantiene un contador local, cada vez que envíe un msg incrementa un reloj. El tiempo asociado con el suceso de cada mensaje es lamarca de tiempo que acompaña al msg. NO importa qué suceso ocurre primero en realidad, solo importa que se puedan ordenar de alguna forma.

Cola distribuida
- Primera versión: n nodos, cada uno tiene procesos que peuden hacer pedidos de acceso mutuamente exlcusivo a un recrso de otros procesos. Cada uno hace de arbitro de sus propios recursos.
  Este algoritmo hace cumplir la exclusión mutua, es equitativo, evita el interbloqueo y la inanición.
  Un proceso puede entrar a una secc crítica cuando:
    - haya recibido por parte de todos los demas nodos (n-1) un msg que garatniza que no hay otro msg en camino (un Pedido anterior al del pedido) que quiera accder a la cola.

    - Todos los mensajes del vector de pedidos se ordenan (y son todos posteriores al otro)
      3 tipos de mensajes: Petición, Respuesta, Liberación.
- Segunda versión: pretende optimizar la primera versión, eliminado los mensajes de Liberación, con propagación y respuestas.

Método de paso del testigo (token)
Testigo: entidad que un proceso retiene en un instante dado.
El proceso que tiene el testigo, es el que puede entrar en su sección crítica sin pedir permiso, cuando la abandona, pasa el testigo a otro proceso.

- La asignación inicial del token es arbitraria
- Puede acceder si tiene el token, o sino difunde un mensaje de peticion[k] con una marca de tiempo y espera hasta recibir el testigo.
- Cuando otro abandona la secc crítica y debe pasar el testigo, elige el proceso que tien petición[k] > testigo[k]

Interbloqueo distribuido

Dos tipos de deadlock: por asignación de recursos o comunicación de mensajes.

Puede aparecer el fenómeno de “interbloqueo fantasma”: falsas detecciones, no un deadlock real, originado por la falta de un estado global.

Prevención del deadlock
Necesitan determinar por adelantado las necesidades de recursos:
- El **círculo vicioso de espera** puede prevenirse definiendo una ordenación lineal de los tipos de recursos. (desventaja, podrian retenerse los recursos por mas tiempo)
- La condición de **retención y espera** puede prevenirse pidiendo a los procesos que todos los pedidos se concedan simultaneamente. (no es eficiente porque se podrian conceder los recursos por más tiempo que el necesario)

Método esperar-morir (wait, die): dados 2pedidos, cada uno con su marca de tiempo. Se previene el circulo vicioso. Si T1 tiene el recurso R, y T2 tiene marca más vieja, entonces se bloquea o se mata. La más antigua tiene prioridad, una transacción matada revive con su tiempo original y tiene mas prioridad (por envejecimiento).
Método herir-esperar (wound, wait): concede de inmediato los pedidos más antiguos, matando las transacciones más jovenes que estén usando el recurso compartido.

Predicción
Es poco práctica! Porque habría que guardar el estado global, inspeccionar un estado global seguro requiriría exlcusión mutua, implica un proceso extra consdierable.

Detección del deadlock
Control centralizado
    Simples y faciles de implemntar, un nodo centrar tiene toda la info
    Sobrecarga de comunicaciones al nodo central
    Vulnerable a fallas del nodo central
Control jerárquico
    Tipo arbol. No vulnerable afallas en un solo punto
    Limitado si los deadlocks se concentran (localizados)
    Alguna sobrecarga, dficil de configurar el sistema
Control distribuido
    No vulnerable a fallos en un solo punto
    Ningun nodo sobrecargado

Resolución complicada, algoritmo dificil de diseñar

Deadlock en la comunicación de mensajes
- Espera mutua
- No disponibilidad de buffers de mensajes

Seguridad

Amenazas
- Confidencialidad (secreto) -> interceptación
- Integridad -> modificación
- Disponibilidad -> interrupción
- Autenticidad -> invención

Controles de acceso:
- Orientados al usuario
- Orientados a los datos

Protección:
- Cifrado unidireccional
- Control de acceso

Software maligno
Backdoors, Bomba lógica, Caballos de troya, Virus (parásitos, residentes en memoria, del boot sector, clandestinos, polimorfos), gusanos