Tema 3: Procesos y Procesadores en Sistemas Distribuidos.

3.1. Hilos (THREADS).

- **3.1.1. Introducción a los hilos.**
- **3.1.2. Uso de las hilos.**
- **3.1.3. Aspectos del diseño sobre hilos.**
- **3.1.4. Implantación de un paquete de hilos.**
- **3.1.5. Hilos y RPC.**
- **3.1.6. Un ejemplo de paquete de hilos.**

3.2. Modelos de Sistemas.

- **3.2.1. El modelo de estación de trabajo.**
- **3.2.2. Uso de estaciones de trabajo inactivas.**
- **3.2.3. El modelo de la pila de procesadores.**
- **3.2.4. Un modelo híbrido.**

3.3. Asignación de Procesadores.

- **3.3.1. Modelos de asignación.**
- **3.3.2. Aspectos del diseño de algoritmos de asignación de procesadores.**
- **3.3.3. Aspectos de implantación de algoritmos de asignación de procesadores.**
- **3.3.4. Ejemplo de algoritmos de asignación de procesadores.**

3.4. Planificación en Sistemas Distribuidos.

Bibliografía:

- Andrew S. Tanenbaum: “Sistemas Operativos Distribuidos”; tema 4, Prentice-Hall, 1996.
- William Stallings: “Sistemas Operativos”; 4ª Edición, tema 4 y 14, Prentice-Hall, 2001.
- Coulouris, George: “Sistemas Distribuidos: conceptos y diseño”; tema 6, Addison-Wesley, 2001

Tema 3.- Procesos y Procesadores en Sistemas Operativos Distribuidos.

3.1. Hilos (THREADS).

- S. O. tradicionales: cada proceso tiene un espacio de direcciones y un único hilo de control.
- Podemos desear que varios procesos que comparten el mismo espacio de direcciones, se ejecuten de manera casi paralela (procesos independientes).

3.1.1. Introducción a los hilos.

- Proceso: es un entorno de ejecución formado por uno o más hilos.
  - **Hilo**: es un traza de ejecución
  - **Entorno de ejecución**: es una unidad de gestión de recursos que consiste en un espacio de direcciones, recursos de comunicación y sincronización de hilos y de alto nivel como ficheros abiertos y ventanas.
- Elementos por proceso (unidad de protección y de asignación de recursos):
  - Espacio de direcciones virtuales, que contiene la imagen del proceso;
  - acceso protegido a los procesadores, otros procesos, ficheros y recursos de E/S.
- Elementos por hilo (unidad de expedición):
  - Estado de ejecución (en ejecución, bloqueado, listo o terminado)
  - El contexto (CP y conjunto de registros del procesador).
  - Una pila de ejecución, almacenamiento estático para las variables locales, acceso a la memoria y a los recursos del proceso (compartidos con el resto de los hilos del proceso).
- Los hilos comparten la CPU (tiempo compartido).En un multiprocesador pueden ejecutarse en paralelo.

Tema 3.- Procesos y Procesadores en Sistemas Operativos Distribuidos.

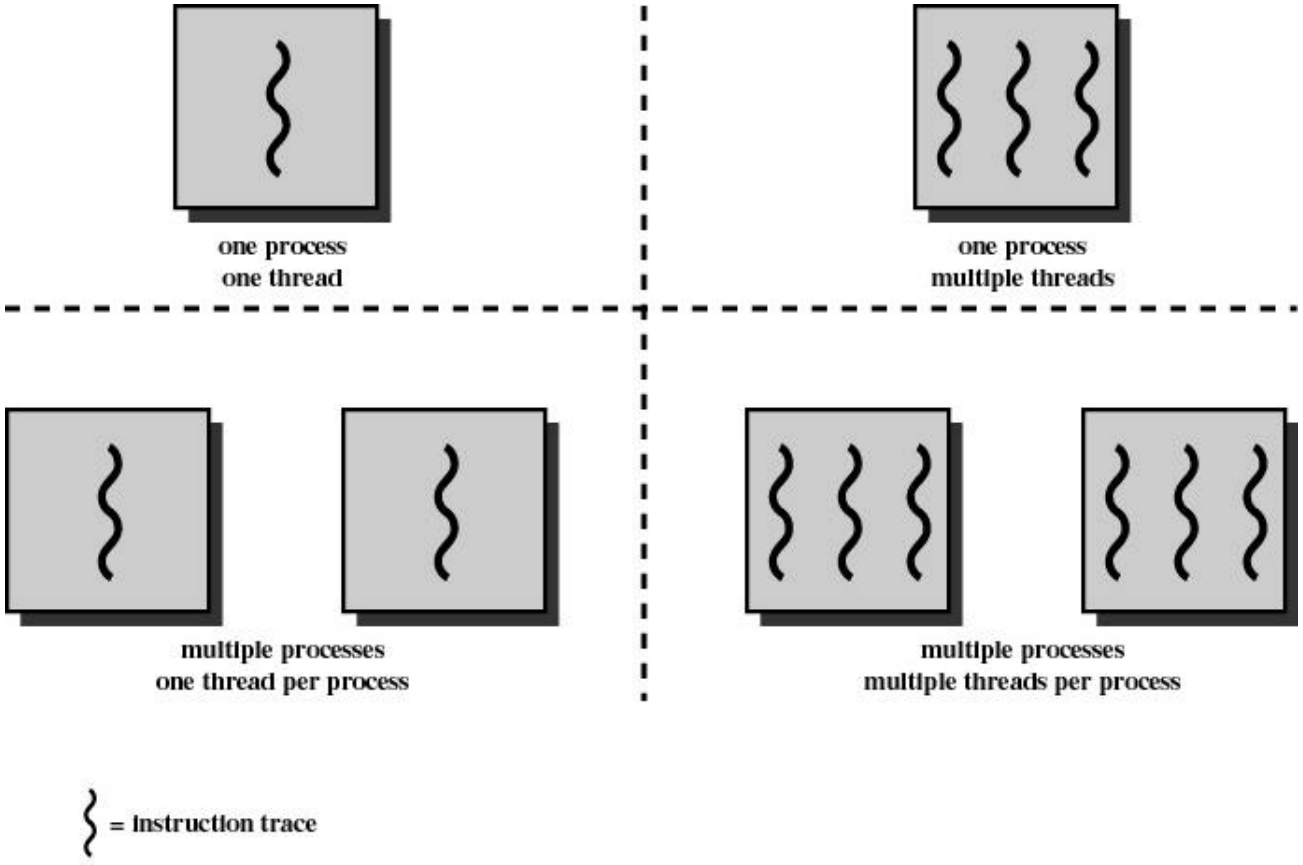


Figure 4.1 Threads and Processes [ANDE97]

Tema 3.- Procesos y Procesadores en Sistemas Operativos Distribuidos.

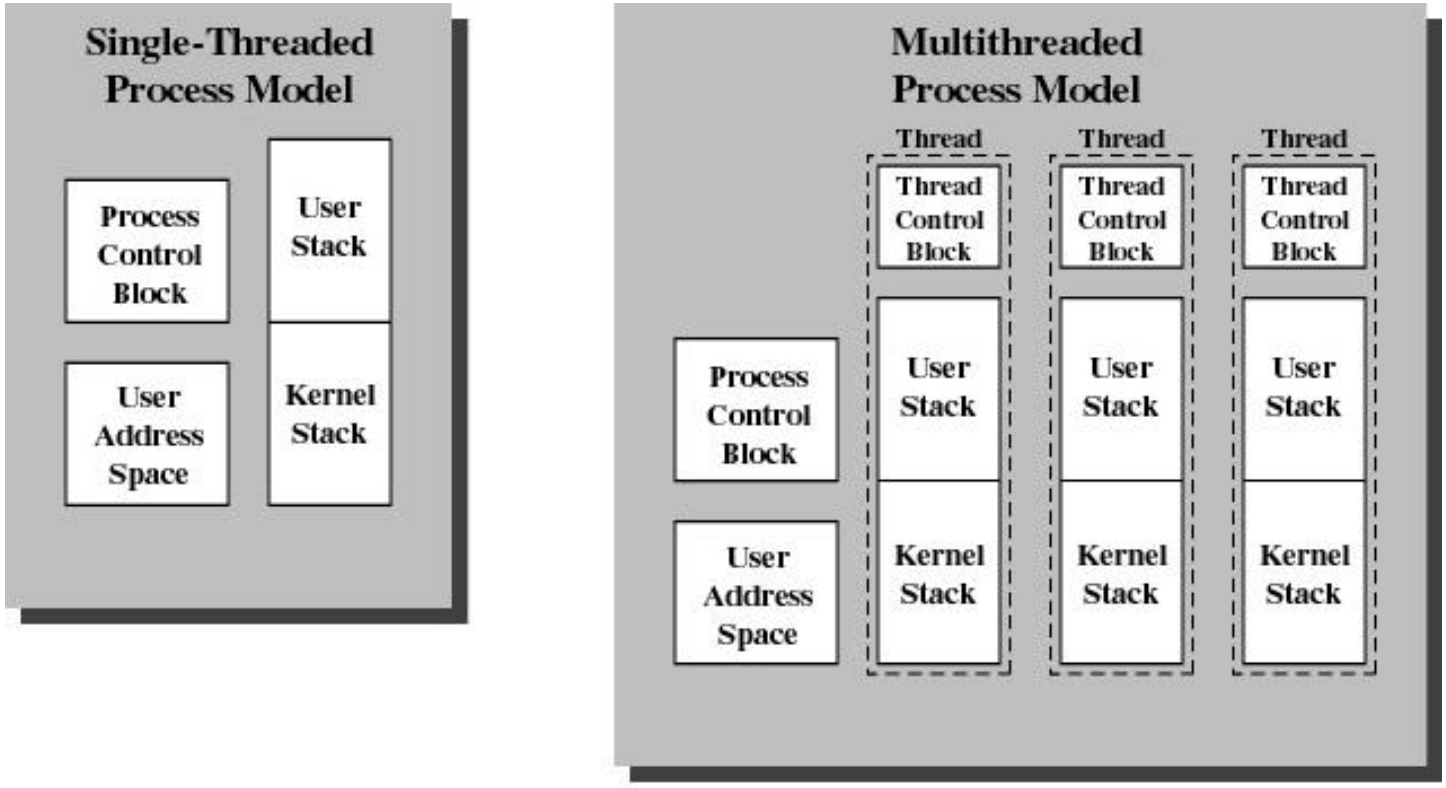


Figure 4.2 Single Threaded and Multithreaded Process Models

- Los hilos pueden crear hijos y se pueden bloquear. Si uno está bloqueado puede ejecutarse otro del mismo proceso.
- Los hilos comparten el mismo espacio y otros recursos como las mismas variables globales y los ficheros abiertos.
  - Cualquier modificación de un recurso desde un hilo afecta al entorno del resto de los hilos del mismo proceso.
  - Es necesario sincronizar las actividades de los distintos hilos para que no interfieran unos o otros o corrompan las estructuras de datos.
- No existe protección entre los hilos porque: 1) es imposible y 2) no deber ser necesaria. Los procesos de un hilo ? al mimo usuario.
- Hay acciones que afectan a todos los hilos de un proceso: la suspensión implica la descarga del espacio de direcciones de la memoria principal.

Beneficios de los hilos:

- Se tarda menos en crear un hilo en un proceso existente que un nuevo proceso (11mseg/proceso frente a 1 mseg/ hilo).
- Se tarda menos en terminar un hilo.
- Es más rápido cambiar entre dos hilos de un mismo proceso.
- Los hilos de un mismo proceso pueden comunicarse entre sí sin invocar al núcleo debido a que comparten memoria y ficheros.

3.1.2. Uso de los hilos.

- Trabajo interactivo y en segundo plano: en una hoja de cálculo, un hilo visualiza los menús y lee la entrada, otro ejecuta órdenes y la actualiza.
- Procesamiento asíncrono: crear un hilo para que haga copias en un procesador de textos.
- Aceleración de la ejecución: computar un lote de datos mientras se lee el siguiente.
- Estructuración modular de programas.

Tema 3.- Procesos y Procesadores en Sistemas Operativos Distribuidos.

3.1.3. Aspectos del diseño sobre hilos.

- *paquete de hilos*: un conjunto de primitivas relacionadas con los hilos.
- ¿Cuál es la arquitectura y funcionalidad de estos paquetes de hilos?

¿Hilos estáticos/ hilos dinámico?

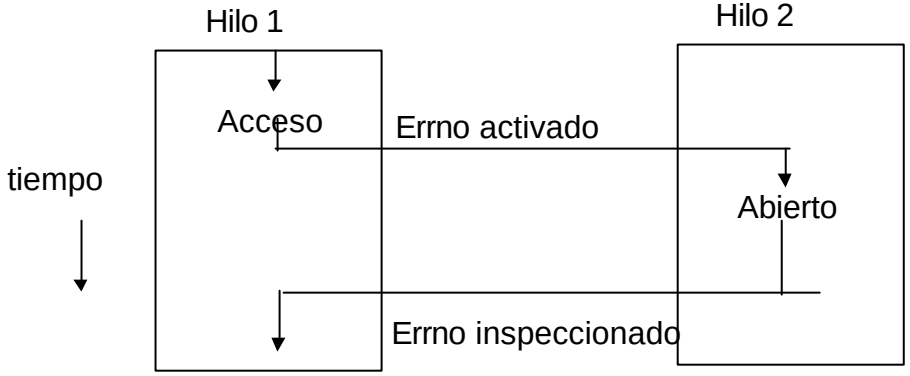
- Estáticos: Se fija el número de hilos al escribir el programa (asociada una pila fija). Simple pero inflexible.
- Dinámicos: permitir la creación y destrucción de los hilos durante la ejecución.

planificación: los hilos se pueden planificar mediante ? algoritmos (prioridad, round robin, al + corto ...).

El acceso a los datos compartidos se programa mediante secciones críticas (semáforos, monitores, mensajes).

Problemas:

- El código de un hilo consta de varios procedimientos. Estos tienen variables locales, globales y parámetros de procedimiento.
- Variables globales de un hilo que no son en todo el programa!!!
- Variable *errno* UNIX. Cuando un proceso falla, se coloca el código de error en *errno*.



Soluciones:

- Prohibir las variables globales. Conflicto con el software existente!!
- Asignar a cada hilo sus variables globales. Los lenguajes de programación no incorporan estas formas intermedias!!!. Se puede transferir el bloque de memoria de variables globales como parámetro.
- Introducir procedimientos de biblioteca para crear, dar valores y leer estas variables globales.

3.1.4. Implantación de un paquete de hilos.

¿Implantar el paquete de hilos en el espacio del usuario o espacio del núcleo?

- Espacio de usuario: gestión de los hilos la realiza la aplicación y el núcleo no es consciente de la existencia de los hilos. Se puede implantar en un SO que no soporta estos hilos.
- Una biblioteca (colección de procedimientos) que se ejecuta en modo usuario contiene código para crear y destruir hilos, intercambiar mensajes entre los hilos, planificarlos y salvar y restaurar el contexto de los hilos.
- Cuando un hilo ejecuta una llamada al sistema, se bloquea mediante una operación, por ej., de semáforo. Almacena los registros del hilo en una tabla, busca un hilo no bloqueado, restablece su entorno y lo ejecuta.
- Todas las operaciones se realizan en el espacio de usuario y dentro del mismo proceso.
- El núcleo continúa planificando el proceso como una unidad y asignándole un estado (listo, en ejecución o bloqueado).
- Los hilos de cada proceso puede tener su propio algoritmo de planificación.

Ventajas

1. El intercambio de hilos no necesita de los privilegios del modo núcleo. Se evita sobrecargar los cambios de modo.
2. Planificación específica para cada proceso.
3. Se puede ejecutar en cualquier Sistema Operativo.

Tema 3.- Procesos y Procesadores en Sistemas Operativos Distribuidos.

Problemas hilos a nivel de usuario:

1. La mayoría de las llamadas al sistema son bloqueantes pero se bloquea no sólo el hilo sino todos los hilos del proceso.
   - ? Se pueden modificar las llamadas al sistema para que no utilicen el bloqueo (transformarlo en un fallo). Esto supone hacer cambios en el SO ? No se puede ejecutar en los SO existentes.
   - ? **Solución**: Comprobar si la llamada puede provocar bloqueo. Si es así, se bloquea el hilo y no se hace la llamada. Esto supone reescribir gran parte de las bibliotecas de sistema El código que se coloca junto a la llamada al sistema para hacer la verificación se denomina *jacketing (*técnica de recubrimiento*)*.
2. Dentro de un único proceso, no existen interrupciones de reloj.
   - ? Solución: el sistema de ejecución solicite una señal al reloj por cada seg. para obtener el control. Programación problemática!!!.
3. Los programadores desean los hilos en aplicaciones donde los hilos se bloquean a menudo (servidor de ficheros). Si el sistema de ejecución resuelve los bloqueos el núcleo se limita a la conmutación de hilos. Constante verificación de la seguridad de las llamadas al sistema.
4. No se aprovecha la ventaja de tener un multiprocesador.

Tema 3.- Procesos y Procesadores en Sistemas Operativos Distribuidos.

- Espacio del núcleo (W2K, Linux, OS/2): El núcleo maneja los hilos. El núcleo mantiene información del entorno del proceso como un todo y la de cada hilo del proceso.
- Cualquier aplicación puede ser programada como multihilo. Las propias funciones del núcleo pueden ser multihilo.
- Para cada proceso, el núcleo mantiene una tabla con una entrada por cada hilo con los registros, estado, prioridades e información relativa al proceso.
- Todas las llamadas que pueden bloquear un hilo se implementan como llamadas al sistema (costo mayor que una llamada a procedimiento).
- Cuando un hilo se bloquea, el núcleo puede ejecutar otro hilo del proceso o un hilo de otro proceso.

Problemas de ambos esquemas:

- Muchos procedimientos de biblioteca no son reentrantes. Ej) el envío de un mensaje a través de la red que utilice un buffer fijo.
- Los procedimientos para asignación de memoria (*malloc*) no se preocupan de establecer y utilizar secciones críticas protegidas. Habría que escribir toda la biblioteca!!!.
- Solución: proporcionar a cada biblioteca un *jacket* que cierre un semáforo global al iniciar el procedimiento. Sólo un hilo puede estar activo en la biblioteca en un instante dado.
- Las señales no son directamente manejables en un ambiente con varios hilos. Están asociadas a un proceso!!!

Tema 3.- Procesos y Procesadores en Sistemas Operativos Distribuidos.

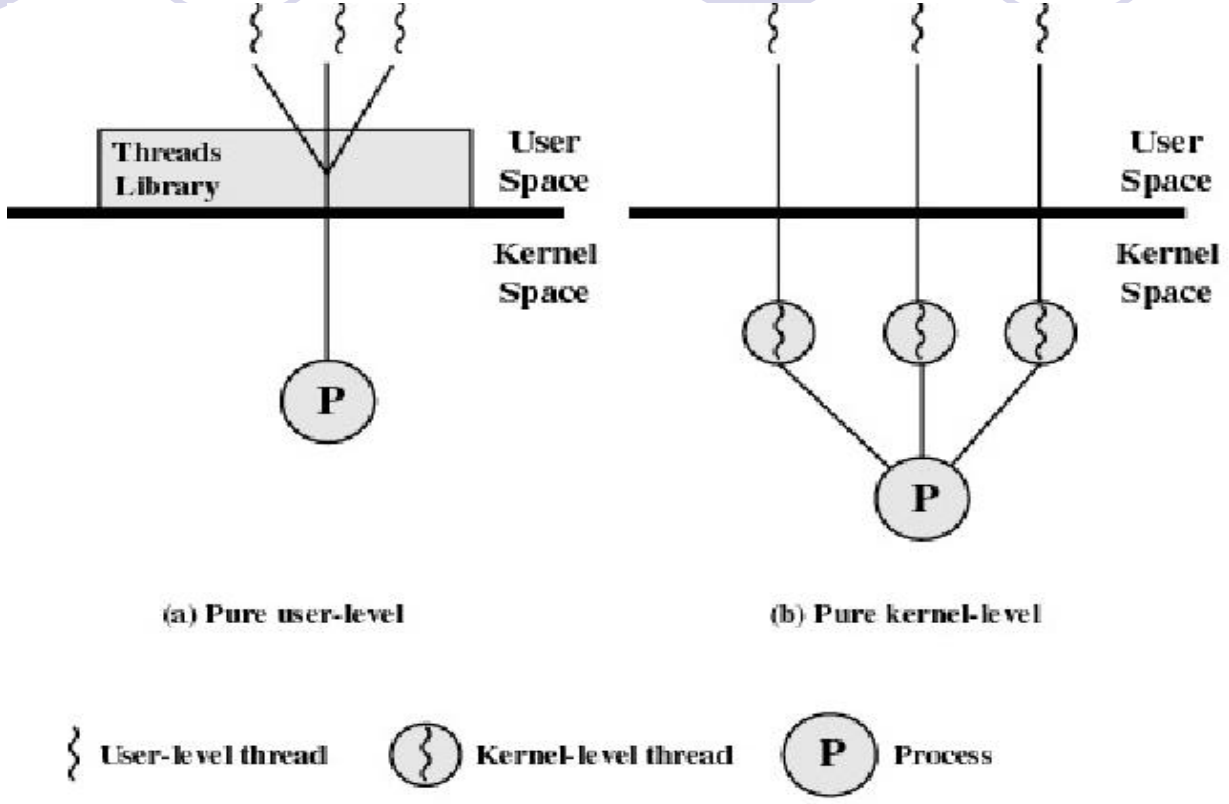


Figure 4.6 User-Level and Kernel-Level Threads

Tema 3.- Procesos y Procesadores en Sistemas Operativos Distribuidos.

3.1.5. Hilos y RPC.

- Los sistemas distribuidos utilizan RPC se pueden favorecen con los hilos.
- Al iniciar un hilo servidor S, éste exporta su interfaz para informar al núcleo. La interfaz define los procedimientos que se pueden invocar, sus parámetros.
- Al iniciar un hilo cliente C, importa la interfaz del núcleo y se le proporciona un identificador especial que utilizará en la llamada.
- El núcleo sabe que C llamará a S y crea estructura de datos para preparar la llamada. Crea una pila compartida por C y S.
- C coloca sus parámetros en la pila y señala al núcleo. El núcleo modifica el mapa de memoria del cliente y lo coloca en el espacio del cliente. No se necesita ni copiado ni ordenamiento. RPC local es más rápida.
- Un hilo servidor se desvanece cuando termina su acción. El hilo se crea en el momento de servir la solicitud.
- Los hilos no tienen que bloquearse y no se tiene que realizar cambio de contexto.
- La creación de un hilo es económica porque no implica restauración del contexto.
- Se reduce el tiempo al no tener que copiar los mensajes.

3.1.6. Un ejemplo de paquete de hilos.

- Equivale a la programación concurrente. Java da soporte directo para hilos, proporciona métodos para la creación, destrucción y sincronización de hilos.
- **Estados**: Cada hilo se crea mediante *Thread*() en estado BLOQUEADO. Cuando pase a PREPARADO con el método *start*(), ejecuta el método *run*(). Un hilo finaliza cuando vuelve del método *run*() o cuando invoca *destroy*(). Los hilos pueden tener una prioridad *setPriority*() y *getPriority*().
- **Sincronización**: Java proporciona la palabra clave *synchronized* para designar un monitor. Sólo un hilo puede ejecutarse dentro del monitor en un instante.
- Se puede bloquear y desbloquear hilos mediante condiciones.
- Para bloquearse se invoca el método *wait*(). Otro hilo llamará a *notify*() para desbloquear un hilo o *notifyAll*() para desbloquearlos todos. El método *join*() bloquea al invocador hasta la terminación del hilo destino. *Interrupt*() desbloquea a un hilo en espera.
- El monitor java tiene la limitación de que únicamente puede usar una variable condición.
- **Planificación**: se utiliza el método *yield*() para permitir que otro hilos puedan planificarse y progresar en su trabajo.

Tema 3.- Procesos y Procesadores en Sistemas Operativos Distribuidos.

3.2. Modelos de Sistemas.

- Sistema Distribuido con varios procesadores ? ¿cómo organizarlo?
- Modelo de estación de trabajo / M. de pila de procesadores / M. híbrido.

3.2.1. El modelo de estación de trabajo.

- M. estación de trabajo: el sistema consta de estaciones de trabajo dispersas y conectadas por medio de un LAN de alta velocidad.
- Las estaciones pueden tener un único usuario, varios e incluso estar inactivas.
- Las estaciones pueden tener discos locales o carecer de disco y acudir a uno o varios servidores de ficheros en la red.
- Si la estación tiene sus propios discos, se pueden utilizar:

1. Paginación y ficheros temporales. Únicamente se utilizan los discos para el intercambio y los ficheros temporales que son eliminados al final de la sesión.
2. Paginación, ficheros temporales y binarios del sistema. El disco también contiene los ejecutables (reduce la carga de la red). Estos programas no suelen modificarse y se pueden instalar en todos los discos locales.
3. Paginación, ficheros temporales, ejecutables y ocultamiento de ficheros. Utilizarlos también para cachés. Los usuarios pueden cargar ficheros desde el servidor, trabajar de manera local y luego devolverlos al servidor. Problema de consistencia!!!.
4. Un sistema local de ficheros completo. Cada máquina tiene su sistema de ficheros con la posibilidad de montar sistemas. de ficheros de otras máquinas. Proporciona un tiempo de respuesta uniforme y reduce la carga de la red. El sistema es más un SO de red que un sistema distribuido y transparente.

Tema 3.- Procesos y Procesadores en Sistemas Operativos Distribuidos.

Ventajas Modelo estación de trabajo:

- Fácil de comprender.
- Los usuarios disponen de capacidad de cómputo y por tanto, su tiempo de respuesta está garantizado. Los programas gráficos pueden ser rápidos
- Los usuarios disponen de cierta autonomía y hacen posible que el trabajo pueda continuar si el servidor falla.

Problemas Modelo estación de trabajo:

- Los chips de procesadores cada vez son más baratos. Será normal que una máquina cuente con 10 CPUs, deberíamos aprovechar su capacidad de cómputo.
- La mayor parte del tiempo, los usuarios están inactivos, mientras se podría utilizar esa capacidad en otro trabajo que se encuentra en una máquina saturada.

3.2.2. Uso de estaciones de trabajo inactivas.

- Primer intento: *rsh machine command*
- Problemas: El usuario debe conocer qué máquinas están inactivas. El ambiente de la máquina local difiere de la local. Si alguien entra en la máquina "inactiva" (ejecuta el proceso remoto) el usuario debe aceptar un desempeño menor.

Debemos responder a las preguntas:

- ¿Cómo encontrar un estación de trabajo inactiva?
- ¿Cómo lograr que un proceso remoto se ejecute de forma transparente?
- ¿Qué ocurre si vuelve el poseedor de la máquina?

Tema 3.- Procesos y Procesadores en Sistemas Operativos Distribuidos.

¿Qué es una estación de trabajo inactiva?!!!

- una estación sin ninguna persona puede ejecutar docenas procesos (demonios).
- una estación con un usuario que no hace nada.
- "inactiva": cuando nadie toca el teclado o pulsa el ratón durante minutos y no se ejecuta ningún proceso iniciado por el usuario.

¿cómo localizar una estación inactiva?

- controlada por el servidor:

1. cuando una estación está inactiva anuncia su disponibilidad y se proporciona su dirección en una BD. Cuando un usuario quiera ejecutar una orden en una máquina inactiva ? *remote command* y el programa busca la estación.
2. la estación inactiva anuncia el hecho enviando en mensaje a todas las máquinas de la red. Tenemos un registro en cada máquina.

- existe el peligro de condiciones de competencia, se debe eliminar del registro cuando se asigne.

- controlada por el cliente: al ejecutar *remote*, transmite el programa que desea ejecutar, la cantidad de memoria necesaria, etc.. *remote* consulta a las máquinas que envían su respuesta con un retraso proporcional a su carga. De este modo se elige la de menor carga.

Tema 3.- Procesos y Procesadores en Sistemas Operativos Distribuidos.

¿Cómo lograr que un proceso remoto se ejecute de forma transparente?

- Desplazar el código es fácil. Debemos también proporcionar el entorno de ejecución local:
  - sistemas de ficheros, directorio de trabajo, variables de entorno.

Problema: llamadas al sistema!!

- Hay llamadas que deben hacerse en la máquina de origen: READ si la máquina tiene disco local, lectura del teclado, escritura en pantalla
- Las llamadas relacionadas con el tiempo, los relojes no están sincronizados ¿tiempo local o remoto?
- Los programas que escriben en dispositivos hardware (buffer de pantalla, disco flexible, cinta) no se pueden ejecutar modo remoto.

¿Qué ocurre si vuelve el poseedor de la máquina?

- No hacer nada: Perdemos la idea de máquina personal.
- Eliminar al intruso: Se pierde el trabajo. Avisar y si no termina el trabajo, eliminarlo.
- Hacer que el proceso emigre a otra máquina. Se debe dejar la máquina en el mismo estado en que se encontró: eliminar todos los hijos, buzones, conexiones de red, estructuras de datos, ignorar los RPC y mensajes que lleguen después de ser eliminado, eliminar los ficheros temporales y restaurar los ficheros de caché.

Tema 3.- Procesos y Procesadores en Sistemas Operativos Distribuidos.

3.2.3. El modelo de la pila de procesadores.

- Construir una *pila de procesadores* que se pueden asignar dinámicamente a los usuarios según la demanda.
  - Los usuarios tiene terminales X en vez de estaciones de trabajo.

Justificación:

- Si el sistema de ficheros se concentra en un reducido nº de servidores para economizar, concentrar las CPU supondrá una mayor capacidad de cómputo a menor precio.
- Facilita el crecimiento por incrementos (sólo ? procesadores).
- No existe el concepto de propiedad: todos los procesadores pertenecen a todos. Los usuarios obtienen tantas CPUs como necesiten durante periodos cortos de tiempo, después regresan a la pila.
- Una pila de procesadores se puede modelar analíticamente por tratarse de un sistema de colas.
- ?: tasa de entradas solicitudes; ?: tasa de procesamiento solicitudes
- Operación estable: ? > ?; en otro caso, la cola crecerá sin límite!!!

  (se pueden permitir intervalos de tiempo en los que ? < ? pero $?_{media} > ?_{media}$ y debe existir el espacio suficiente en los buffers)
- T: tiempo promedio entre solicitud y respuesta 1/(? - ?)

  cuando ?? 0 (no carga); T no tiende a 0; el tiempo de respuesta siempre incluye al tiempo de procesamiento.
- Si tenemos *n* multiprocesadores con cierto número de CPUs, cada uno con su sistema de colas; T =1/(? - ?). Si reunimos todos los CPUs en una pila, tenemos un sistema de colas ejecutándose en paralelo T_P= 1/(n? - n?)= T/n. Reducimos el tiempo promedio *n* veces.

- La mayor parte del tiempo se perdía porque la mayoría de los servidores estaban inactivos. Este tiempo se elimina con el modelo de pila de procesadores. Máxima concentración de cómputo!!!

¿Cómputo distribuido?:

- relación precio/desempeño mejor.
- confiabilidad y tolerancia a fallos.
- las estaciones de trabajo tienen una respuesta uniforme independiente de la carga.
- hemos supuesto que una pila de n procesadores es igual a un único procesador *n* veces más rápido. Esto sólo es justificable si todas las solicitudes se pueden ejecutar en todos los procesadores en paralelo.
- En cualquier caso, la pila de procesadores es una forma más limpia de obtener cómputo adicional que la búsqueda de estaciones inactivas.
- Si el tipo de trabajo es interactivo de bajo cómputo es suficiente con estaciones de trabajo; si los usuarios necesitan de grandes cálculos, la pila de procesadores es sencilla y atractiva.

3.2.4. Un modelo híbrido.

- Se puede proporcionar a cada usuario un estación de trabajo y tener una pila de procesadores.
- Solución más cara pero combina las ventajas de ambos.
- El trabajo interactivo se realiza en las estación de trabajo (respuesta garantizada) y el cómputo pesado se ejecuta en la pila de procesadores.

Tema 3.- Procesos y Procesadores en Sistemas Operativos Distribuidos.

3.3. Asignación de Procesadores.

- Un sistema distribuido consta de varios procesadores.
- Necesitamos de un algoritmo que nos diga qué proceso hay que ejecutar y en qué máquina.
- Modelo estación de trabajo: ¿cuándo ejecutar el proceso de manera local y cuándo buscar una estación inactiva?.
- Modelo de pila de procesadores: decisión para cada nuevo proceso.

3.3.1. Modelos de asignación.

- Supone que todas las máquinas son idénticas o compatibles en el código y sólo pueden diferenciarse en la velocidad.
- Supone que todas las máquinas están interconectadas. Se puede establecer conexión de transporte entre cualquier pareja de máquinas.

- <u>Estrategias de asignación de procesadores:</u>
- No migratoria: al crearse el proceso, se toma la decisión sobre donde ejecutarlo y permanece allí hasta que termina.
- Migratoria: un proceso puede trasladarse a otra máquina aunque haya iniciado su ejecución. Permiten balancear la carga pero los algoritmos son más complejos.
- Un algoritmo que asigne procesos a procesadores busca el optimar el sistema, en otro caso, la asignación sería aleatoria.

Tema 3.- Procesos y Procesadores en Sistemas Operativos Distribuidos.

Pero, ¿qué optimar???.

- Maximizar el uso de la CPU. Evitar la inactividad de la CPU.
- Minimización del tiempo promedio de respuesta.

Ejemplo) 2 procesos y 2 procesadores.

procesador 1: 10MIPS;

procesador 2: 100MIPS y una lista (5seg).

proceso A: 100 millones de instrucciones.

tiempos respuesta: 10seg (procesador 1)
6seg (procesador 2)

proceso B: 300 millones de instrucciones.

tiempos respuesta: 30seg (procesador 1)
8seg (procesador 2)

Asignación: A a 1 y B a 2 ? (10+8)/2= 9seg.
A a 2 y B a 1 ? (6+30)/2= 18 seg.

- Variación ? tasa de respuesta: cantidad de tiempo necesario para ejecutar un proceso en cierta máquina dividido entre el tiempo que tardaría en ejecutarse en procesador de referencia no cargado.
- Esta métrica es mejor porque tiene en cuenta que los procesos de gran tamaño tardan más que los pequeños.

ejemplo)

trabajo 1 seg que tarda 5 seg

trabajo 1 min que tarda 70 seg ? 5/1 >> 70/60.

Tema 3.- Procesos y Procesadores en Sistemas Operativos Distribuidos.

3.3.2. Aspectos del diseño de algoritmos de asignación de procesadores.

- Decisiones de diseño:

Algoritmos deterministas vs. heurísticos.

- A. deterministas son adecuados cuando se sabe las necesidades y comportamiento de los procesos.
- En ningún sistema se sabe!! Pero se pueden dar aproximaciones estadísticas para determinados sistemas (bancos, líneas aéreas).
- A. distribuidos, la carga es impredecible. La asignación se hace mediante heurísticas.

Algoritmos centralizados vs. distribuidos.

- Centralizar la información permite tomar mejores decisiones pero menos robustas y carga el peso en una sola máquina. Son preferibles los algoritmos centralizados; no hay soluciones distribuidas adecuadas!!!.

Algoritmos óptimos vs. subóptimos.

- los algoritmos óptimos son más caros; la mayoría de los S.D. buscan soluciones subóptimas, heurísticas y distribuidas.

Algoritmos locales vs. globales (política de transferencia).

- cuando se crea un proceso hay que decidir si se ejecuta o no en la máquina que lo genera.
- se puede decidir con la información local o recolectando información sobre la carga de otras máquinas. Los A. locales son baratos pero menos óptimos.

Algoritmos iniciados por el emisor vs. iniciados por el receptor.

- ¿dónde enviar un proceso desechado?
- Esta búsqueda puede ser iniciada por el emisor (lanza un mensaje de auxilio para que alguna máquinas se haga cargo) o por el receptor (las máquinas inactivas anuncian su disposición para trabajar).

3.3.3. Aspectos de implantación de algoritmos de asignación de procesadores.

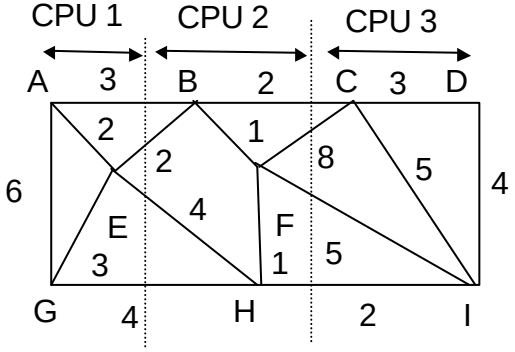
- Casi todos los algoritmos suponen que las máquinas conocen su propia carga. Medir la carga no es sencillo!!!.
- Podemos medir el nº de procesos, ¿qué procesos contar??.
- Podemos contar los procesos listos o en ejecución pero muchos procesos se ejecutan de forma periódico por breves momentos.
- Medir la fracción de tiempo que la CPU está ocupada. Podemos utilizar un cronómetro que observe de forma periódica el estado de la CPU pero si se ejecuta el núcleo no admite interrupciones!!!
- Recolectar medidas y trasladar procesos supone un costo excesivo. Tal vez el beneficio no supere el gasto del traslado.
- Los algoritmos suelen ser demasiado complejos y por tanto, lentos. *Eager* demostró que es preferible un algoritmo sencillo si proporciona casi la misma ganancia que uno más caro.
- En situaciones de inestabilidad (con falta de información) se puede llegar a interminables traslados de un proceso entre máquinas con parecida carga.

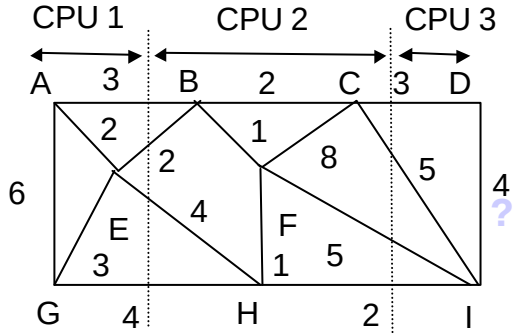
Tema 3.- Procesos y Procesadores en Sistemas Operativos Distribuidos.

3.3.4. Algoritmos de asignación de procesadores.

Un algoritmo determinista según la teoría de gráficas

- Sistemas con procesos con requerimientos conocidos de CPU y memoria, además de una matriz con el tráfico promedio entre cada dos procesos.
- Se pretende hacer una asignación que minimice el tráfico en la red de forma que se cumplan las restricciones de CPU y memoria de los procesos. Ej) a) 30 unidades y b) 28 unidades.
- Se busca asignaciones fuertemente acoplados pero que interactúen poco.





Un algoritmo centralizado

- Algoritmo que emplea heurística. El coordinador mantiene una tabla de uso con una entrada por cada estación de trabajo con un valor inicial a 0. Cuando ocurren eventos se envían mensajes al coordinador para actualizar la tabla.
- Las decisiones de asignación se basan en esta tabla. Intenta proporcionar al poseedor de la máquina una capacidad de cómputo justa.
- Cuando se crea un proceso y la máquina que lo generó considera que está cargada, le pide al coordinador que le asigne un procesador.
- Si existe uno disponible y nadie lo desea se le otorga el permiso. En otro caso se toma nota de la petición.
- Cuando una máquina ejecuta procesos en otras máquinas acumula puntos de penalización en la tabla de usos. Cuando acumula peticiones no satisfechas se le restan los puntos de penalización.
- Con esta puntuación mantenemos información sobre las máquinas. Cuando un procesador se libera se le proporciona a la solicitud con puntuación más baja.

Tema 3.- Procesos y Procesadores en Sistemas Operativos Distribuidos.

Un algoritmo jerárquico

- Los algoritmos centralizados se convierten en un cuello de botella y en un único punto de fallo.
- Podemos organizar una colección de procesadores como una jerarquía lógica independiente de la estructura física de la red.
- Unas máquinas son trabajadores y otras administradoras. Para cada k trabajadoras, una administradora mantiene un registro de las máquinas ocupadas e inactivas. Si el sistema es grande, estas administradoras dependen de otras máquinas de rango superior.
- Cada procesador sólo necesita comunicarse con su superior y unos cuantos subordinados; el flujo de información es controlable.
- ¿Qué ocurre si un “jefazo” muere? Podemos sustituirlo por otro elegido por los subordinados o por el jefe del difunto.
- Para evitar un único “super-jefe” se trunca el árbol y se forma un comité.
- Los administradores se preocupan de suministrar los procesadores necesarios para un determinada solicitud. Si no cuenta con suficientes subordinados recurre a su jefe. Y la solicitud sigue propagándose...
- Las solicitudes de procesadores se puede generar en cualquier parte del sistema, de forma que tendremos varias solicitudes en diversas etapas en el algoritmo de asignación. El peligro de bloqueo es bastante elevado!!!

Tema 3.- Procesos y Procesadores en Sistemas Operativos Distribuidos.

Un algoritmo distribuido heurístico

- Al crearse un proceso, la máquina original mediante envío de mensajes de manera aleatoria investiga la carga de otra máquina. Si es menor que cierto valor de referencia envía allí su proceso.
- Si no continúa la búsqueda pero este intento lo repetirá sólo N veces.
- Resulta estable y adecuado para distintos rangos de cargas.

Un algoritmo de remates

- Cada procesador anuncia su precio a través de un fichero que todos pueden leer (este precio es el que pagó el último cliente).
- Cuando un proceso desea iniciar su proceso hijo, verifica si alguien ofrece el servicio que necesita. Del conjunto que lo puede hacer, se queda con el mejor candidato.
- Genera una oferta y la envía a este procesador. La oferta puede ser menor o mayor que el precio anunciado.
- Los procesadores reúnen las ofertas y eligen la mayor de todas. El proceso se ejecuta en esta máquina y el precio se ajusta a la nueva tasa.
- Ferguson deja muchas preguntas sin responder respecto a este algoritmo: ¿de dónde obtienen los procesos el dinero para hacer sus ofertas?, ¿ tienen el mismo salario?, si entran más usuarios, ¿se elevan los precios? ....

Tema 3.- Procesos y Procesadores en Sistemas Operativos Distribuidos.

3.4. Planificación en Sistemas Distribuidos.

- Cada procesador hace su planificación local.
- Pero si existen dependencias entre procesos que se ejecutan en distintas máquinas esta forma de actuar no es la más conveniente.
- Deberíamos sincronizar la ejecución de procesos que se comunican. Todos los miembros de un grupo se ejecuten al mismo tiempo.
- Se puede utilizar una matriz conceptual en la que los distintos procesadores sitúan en la misma posición los procesos que forman parte de un grupo.

Procesador

Espacio de tiempo	0	1	2	3	4	5	6	7
0	X				X			
1			X			X		
2		X			X		X	
3	X					X		
4		X		X				X
5			X		X			

- Cada columna es la tabla de procesos de un procesador.
- Cada procesador planifique mediante turno rotatorio, los procesos que deben comunicarse durante el mismo espacio de tiempo.