

Sistemas Distribuidos

1. Introducción

2. La Comunicación

☞ 3. Sistemas Operativos Distribuidos

1. Estructura de un Sistema Operativo

2. Gestión de Procesos

- Reparto de Carga

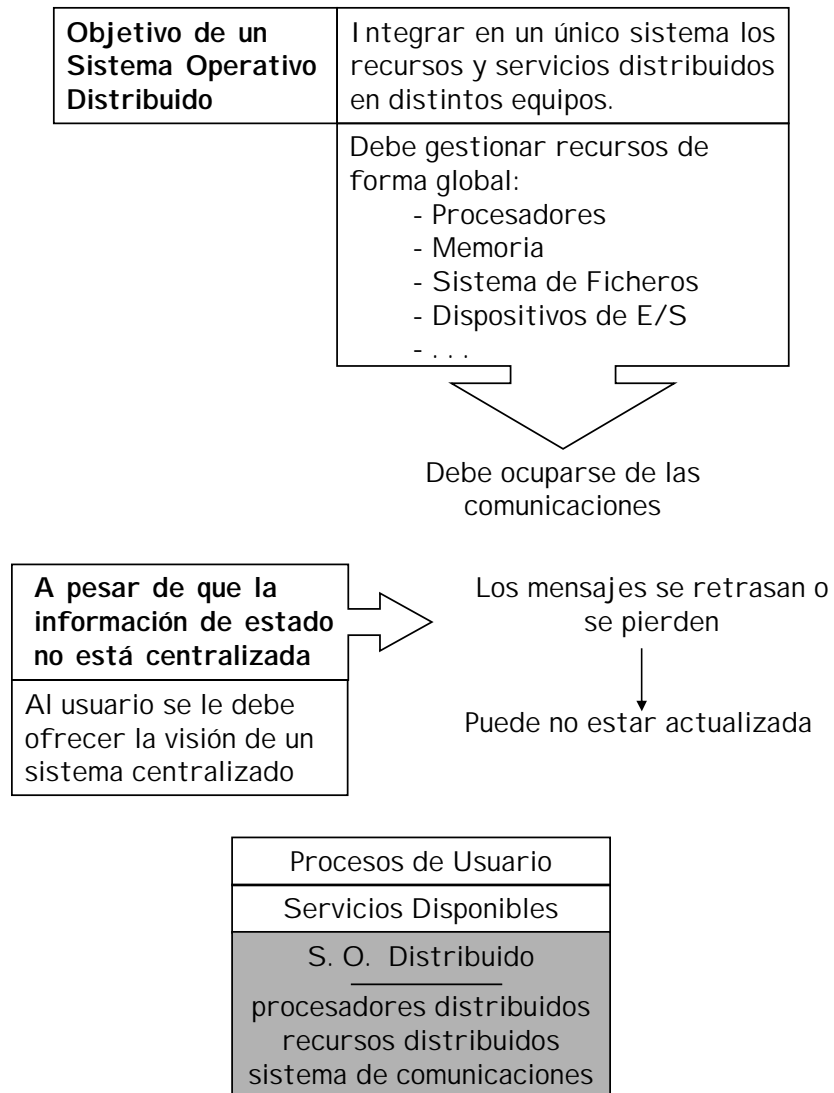
4. Sincronización y Coordinación

En este capítulo vamos a tratar dos cuestiones referentes a los sistemas operativos.

La primera, la estructura de los sistemas operativos, no es una cuestión particular o exclusiva de los sistemas distribuidos, pero dado que con la llegada de los sistemas distribuidos han tomado mucho auge los sistemas operativos basados en un micronúcleo (o *microkernel*), nos parece interesante comentar esta estructura.

A continuación trataremos la gestión de procesos desde el punto de vista de los sistemas distribuidos, esto es, teniendo en cuenta que se dispone de múltiples unidades de proceso. Por esto, será conveniente realizar una planificación de procesos que sepa repartir la carga convenientemente para obtener unos buenos tiempos de respuesta.

Sistemas Operativos Distribuidos



El objetivo principal de un sistema operativo distribuido es conectar los recursos y servicios disponibles, mediante una red de comunicaciones, e integrarlos en un único sistema. Para conseguir esto con las características vistas en los capítulos introductorios, se deben tomar algunas decisiones acerca de la estructura del sistema operativo que va a soportarlo.

A la hora de construir un sistema operativo distribuido, hay dos opciones. Una consiste simplemente en diseñarlo e implementarlo completamente, apoyándose directamente en la máquina desnuda o en *microkernels* (como veremos próximamente). La otra posibilidad es construir una capa que se apoya sobre un sistema operativo convencional, y que ofrece servicios distribuidos. Esto permite aprovechar el sistema operativo que se está utilizando y con su misma interfaz, y simplemente aumentando los servicios. Un ejemplo de este enfoque es el DCE (Distributed Computing Environment) de la OSF. Nosotros trataremos aquí cuestiones referentes a sistemas operativos completos.

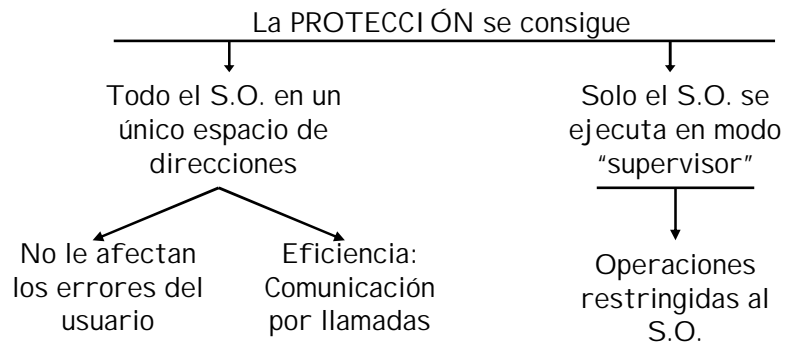
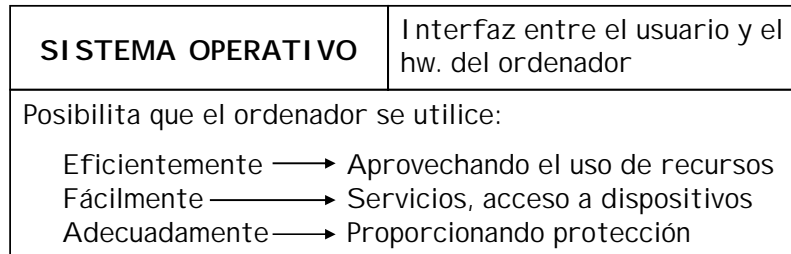
Aunque todos los recursos del sistema deben gestionarse de forma global, un sistema operativo distribuido también debe ocuparse, al igual que el sistema centralizado, de la gestión de los procesadores, de la asignación de memoria, de los dispositivos de entrada/salida, y de las comunicaciones entre los distintos equipos. En la gestión de estos elementos, la principal diferencia entre el sistema operativo centralizado y el distribuido es que en el primero se dispone de una información exacta y actualizada de todo el estado del equipo, mientras que en el caso distribuido no se dispone de un reloj común y, además, el paso de los mensajes (con la información de estado) tiene retrasos, e incluso se pueden perder. Sin embargo, a pesar de estos problemas, al usuario se le debe ofrecer la misma visión que en el sistema centralizado, es decir, el usuario solamente quiere ver que se le ofrecen servicios (en mayor o menor cantidad).

Según esto, una visión simple de la estructura de un sistema operativo distribuido tendrá el aspecto de la figura inferior, es decir, los procesos de usuario ven servicios disponibles proporcionados o apoyados sobre el sistema operativo distribuido. Para que el sistema operativo pueda ofrecer estos servicios, con las características añadidas ya conocidas (transparencia, seguridad, compartimiento, etc.), deberá ocuparse, al menos, de la gestión de los mismos componentes básicos que en el caso de un sistema operativo centralizado, esto es: gestión de procesos, gestión de memoria, gestión de entrada/salida o dispositivos, y de la gestión de ficheros.

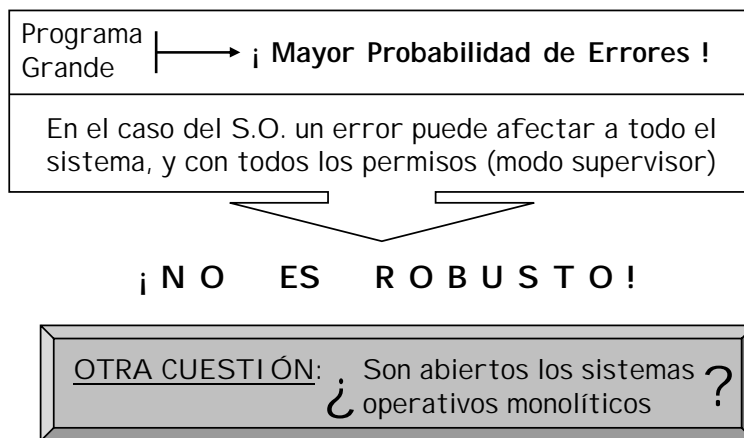
No obstante, ya que estos recursos pueden estar distribuidos, y se van a utilizar mecanismos globales para acceder a estos recursos, la gestión de estos componentes básicos va a requerir algunas características especiales o diferentes de la gestión realizada en los sistemas operativos centralizados.

Este capítulo vamos a dedicarlo a tratar las peculiaridades que se deben tener en cuenta en un sistema operativo distribuido a la hora de gestionar los recursos básicos (en especial, los procesadores) y de cómo estructurar los programas que van a ocuparse de esta gestión.

Comenzaremos, en el siguiente apartado, viendo dos enfoques diferentes que puede adoptar la estructura de un sistema operativo distribuido.



PERO ...



Una definición un tanto abstracta de sistema operativo podría ser esta: "es un programa o conjunto de programas que actúa de interfaz entre el usuario y el hardware del ordenador". Afinando un poco más, se debe tener en cuenta que el sistema operativo debe permitir que el ordenador se utilice:

- Eficientemente, aprovechando el uso de la CPU (gestión de recursos en un sistema multi-usuario).
- Fácilmente, proporcionando servicios (acceso a dispositivos).
- Adecuadamente, proporcionando protección.

Ya que los ordenadores eran caros y había que sacarles rendimiento, de lo primero que se preocuparon fue de conseguir la eficiencia, por lo que el sistema operativo se encargó de gestionar y repartir eficientemente los recursos del ordenador (procesador, memoria, dispositivos periféricos, etc.). Una vez conseguida la eficiencia, se pensó en proporcionar facilidad de uso, por lo que los sistemas operativos comenzaron a proporcionar servicios, empezando por los *drivers* de los dispositivos, de tal forma que mediante una capa de acceso al usuario se le abstraía a éste de la complejidad de manejo de los controladores de los dispositivos. Estos servicios incluyeron luego el servicio de ficheros y lo que en general se denomina como software de sistemas (intérpretes de comandos, compiladores, editores, etc.). Así pues, tenemos que los servicios del sistema operativo son estos:

- **Gestión de Servicios Básicos**
 - Gestión y protección de memoria
 - Gestión y comunicación de procesos, y planificación o reparto del procesador
 - Gestión de los dispositivos periféricos.
- **Servicios de Usuarios y Aplicaciones**
 - Gestión de ficheros
 - Control de fecha y hora

Con este sistema operativo, la protección se conseguía por dos vertientes; por una parte, los programas del sistema operativo se ejecutaban en su propio espacio de direcciones, con lo cual las aplicaciones del usuario no podían afectar ni al código ni a los datos de los programas del sistema operativo; por otra parte, estos últimos se ejecutaban en modo "supervisor", es decir, con permiso para acceder a todo el hardware subyacente (incluyendo todo el repertorio de instrucciones del procesador), mientras que las aplicaciones de usuario se ejecutaban en modo "usuario", y no disponen de permisos para un acceso indiscriminado al hardware, evitando así posibles utilizaciones indebidas de los recursos del ordenador (por ejemplo, formatear un disco duro).

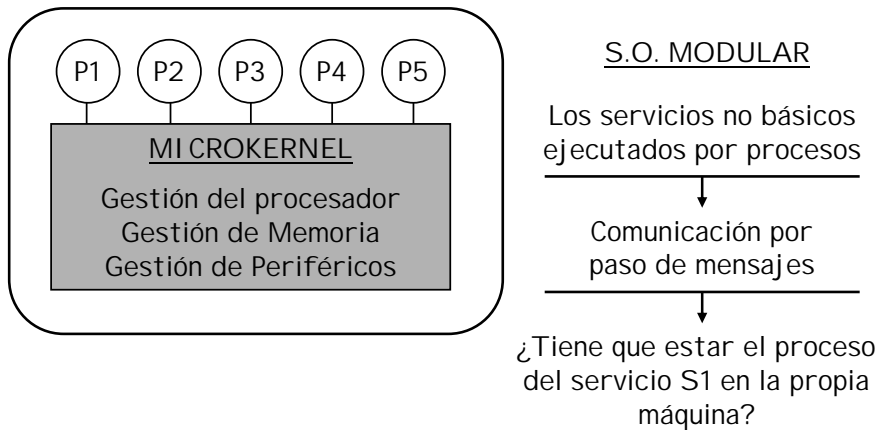
Teniendo encerrados dentro del sistema operativo todos los servicios que se le ofrecían al usuario, no solo se conseguía protección, además se conseguía mejorar la eficiencia, puesto que toda la comunicación entre los programas se realizaba mediante simples y rápidas llamadas a procedimientos.

Sin embargo, esta acaparación de funciones por parte del sistema operativo en un único bloque (montones de subprogramas) que se ejecutaba en modo supervisor trajo consigo, curiosamente, una disminución progresiva de su robustez (resistencia a la caída del sistema).

El sistema operativo se convirtió en un programa muy grande (**sistema operativo monolítico**), y un programa muy grande es más propenso a tener errores. Tener un error aquí es muy peligroso, pues aunque el error se produzca en un programa no vital, sí puede afectar, por efectos laterales, a otras partes fundamentales del sistema operativo; y no hay protección contra esto, ya que el programa que originó el error se está ejecutando en el mismo espacio de direcciones y con los mismos permisos que el resto de los componentes del sistema operativo.

Surge otra cuestión con los sistemas operativos monolíticos: ¿Son abiertos los sistemas operativos monolíticos? Parece que un programa como el que hemos descrito no es muy abierto, puesto que no va a resultar fácil el intercambio de cualquiera de sus componentes, ni siquiera va a ser fácil realizar una pequeña modificación del software existente sin parar la máquina y, por lo tanto, sin que se entere el usuario (esto implica: modificación del fuente, compilación, montaje, carga y rearranque del sistema).

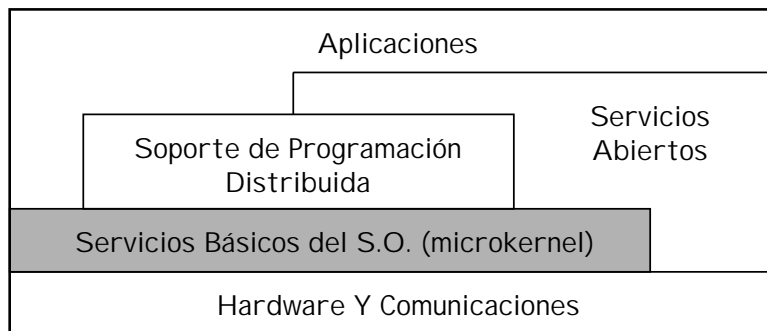
Veamos a continuación una alternativa a este enfoque monolítico de los sistemas operativos.



Para modificar o alterar los servicios
¡NO HAY QUE PARAR LA MÁQUINA!

HEMOS CONSEGUIDO

- Transparencia
- Tolerancia a Fallos
- Sistema más Abierto



Estructura de un S.D. Basado en un Sistema Operativo Modular (microkernel)

Ejemplos: Mach y Chorus. V-System y Amoeba

La alternativa que surgió para proporcionar sistemas operativos robustos fue la de minimizar un subconjunto del sistema operativo que sea el soporte básico de ejecución de la máquina, y haciendo que solamente este subconjunto se ejecute en modo supervisor. A este soporte básico de ejecución ya se le denominó núcleo (*kernel*) en los años 70, aunque con su resurgimiento en estos últimos años, como alternativa a la estructura monolítica de Unix, también se le ha rebautizado con el nombre de **microkernel**.

El resto de los servicios del sistema operativo deben ser programas distintos ejecutándose en distintos servidores, de tal forma que un problema o error en uno de ellos no afecte al resto de los servicios del sistema operativo, y mucho menos, a su núcleo.

Así pues, nos encontramos con que en un sistema operativo construido alrededor de un núcleo o microkernel, éste se encarga exclusivamente de la gestión de los servicios básicos (procesador, memoria, periféricos, ...), mientras que el resto de los servicios (gestión de ficheros, ...) están implementados por procesos, no por rutinas, que se encuentran fuera del microkernel. Esto quiere decir que ahora la comunicación entre los distintos servicios del sistema operativo se realiza mediante paso de mensajes.

Ya que cada servicio está asociado a algún proceso con el que nos comunicamos mediante paso de mensajes, ya no hay nada que nos impida tener ese proceso en la propia máquina o en una remota; funcionará siempre que dispongamos de una RPC que nos ofrezca ese servicio. Así, llamando a la RPC correspondiente, la ubicación del gestor del servicio que solicitamos es transparente.

Démonos cuenta, también, de que ahora los procesos que prestan un determinado servicio pueden estar replicados, con lo que ante el fallo de uno, si el sistema está bien implementado, otro proceso toma el control y, frente al usuario, el sistema sigue comportándose como si nada hubiera pasado. Igualmente, si se decide sustituir un servicio anticuado por otro equivalente con mejores prestaciones, la sustitución se puede realizar simplemente creando el nuevo proceso (con el nuevo gestor), actualizando la tabla de registros de servicios y, por último, parando y eliminando el proceso del servicio anticuado.

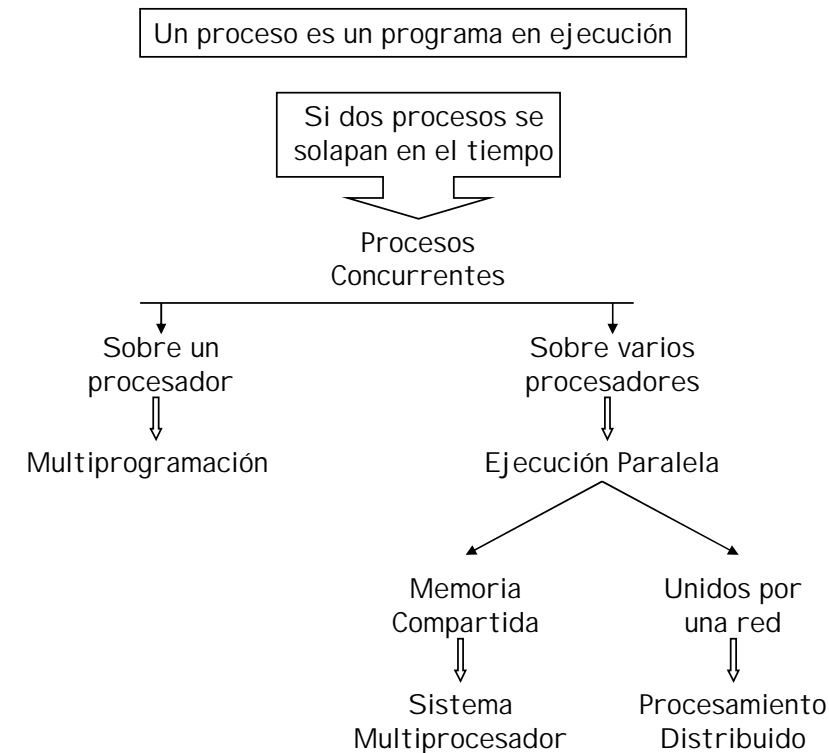
Disponiendo entonces de esta nueva estructura de sistema operativo, los nuevos niveles virtuales de una máquina (excluyendo el hardware y las aplicaciones) apropiados para los sistemas distribuidos son:

Servicios Básicos del Sistema Operativo (de *kernel*). Son los comentados servicios de gestión del procesador y de los procesos, memoria, periféricos y algunos más que se consideren imprescindibles para el arranque del ordenador. Este nivel requiere ser monolítico ya que debe estar bien protegido contra posibles modificaciones en tiempo de ejecución, es decir, debe ser una capa estable y fiable por encima de todo.

Servicios Abiertos. Componen los servicios para usuarios y aplicaciones. Se pueden añadir, quitar y modificar servicios a voluntad sin ni siquiera tener que apagar la máquina.

Soporte para Programación Distribuida. Está compuesto por los soportes de ejecución de los lenguajes para aplicaciones distribuidas, junto con los servicios de comunicación de difusión (*multicast*) y con las RPC's o interfaces de los servicios ofrecidos por máquinas remotas.

Mach y Chorus son dos ejemplos famosos de micronúcleos, mientras que Amoeba y V-System son dos sistemas operativos distribuidos basados en micronúcleos.



La Gestión de Procesos se ocupa de distribuir los recursos de proceso entre todos los procesos de toda la red.

Debe ofrecer mecanismos para realizar operaciones con procesos locales y remotos.

Se debe preocupar de minimizar los tiempos globales de respuesta

REPARTO DE CARGA

- Ejecución Remota
- Migración

Antes de entrar en profundidad en la gestión distribuida de procesos, demos un repaso rápido a los tipos de procesamiento que puede haber en un sistema. Ya sabemos que un **proceso** es un programa en ejecución, es decir, es la actividad que resulta de la ejecución de un algoritmo, con sus datos, sobre un procesador. En un ordenador los procesos no se suelen ejecutar de forma aislada, sino que acostumbran a ejecutarse con otros de forma solapada en el tiempo, ya que compartiendo adecuadamente los recursos físicos se puede obtener una aceleración global de la velocidad de cálculo del sistema. Así, en general, decimos que dos procesos son **concurrentes** si su ejecución se solapa en el tiempo. Si tales procesos comparten el procesador, decimos que se ejecutan en **multiprogramación**, mientras que si n procesos se ejecutan sobre n procesadores, entonces tenemos **ejecución paralela**. En este último caso, si los procesadores comparten una memoria central común, se tiene un **sistema multiprocesador**, mientras que si los procesadores están conectados mediante una red de comunicaciones, entonces tenemos **procesamiento distribuido**.

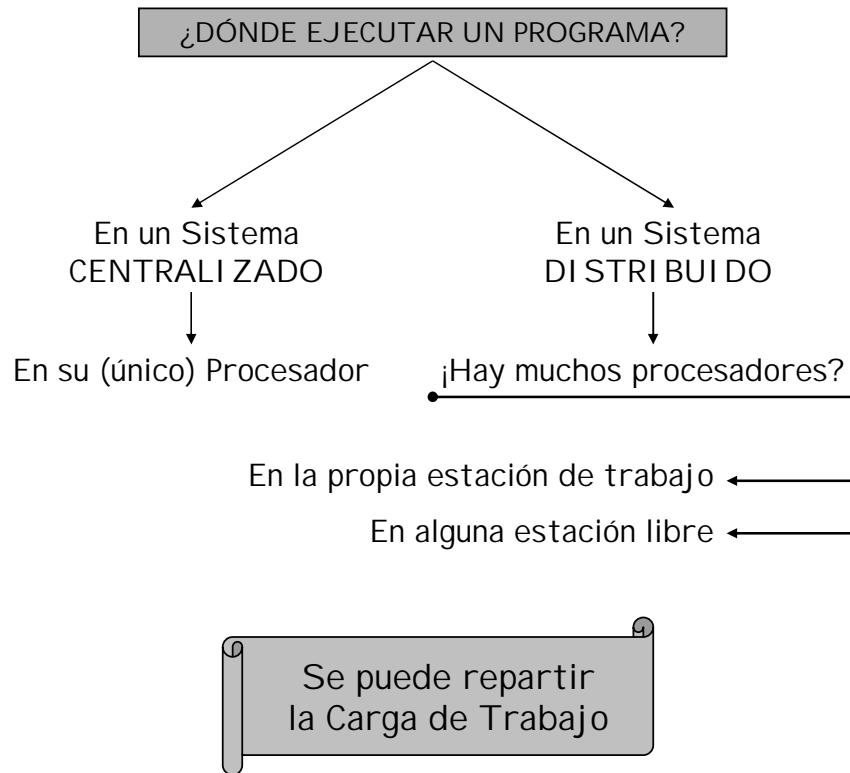
La gestión de procesos en un sistema operativo centralizado se ocupa de los mecanismos y políticas para compartir o repartir un procesador entre diversos procesos de usuario. El objetivo de la gestión de procesos en los sistemas operativos distribuidos es compartir todos los recursos de proceso (distribuidos por toda la red) entre todos los procesos de toda la red del sistema distribuido.

Para conseguir esto, es necesario proporcionar mecanismos y políticas para realizar operaciones con los procesos (crear, nombrar, borrar, ejecutar, ...), tanto locales como remotos, para gestionarlos, comunicarlos y sincronizarlos.

Estos mecanismos van a tener que ampliar los ya existentes en los sistemas centralizados, para poder tratar con la distribución de recursos y la distribución de la información del estado de los recursos por toda la red.

Para mejorar el tiempo de respuesta de los procesos, se va a necesitar la posibilidad de repartir la carga de trabajo de una estación entre otras que estén más descargadas, por lo que se deberá proporcionar la posibilidad de **ejecución remota** de procesos y de **migración** de procesos entre estaciones. Pasemos a ver, a continuación, en qué consisten estos dos mecanismos de reparto de carga.

Reparto de Carga

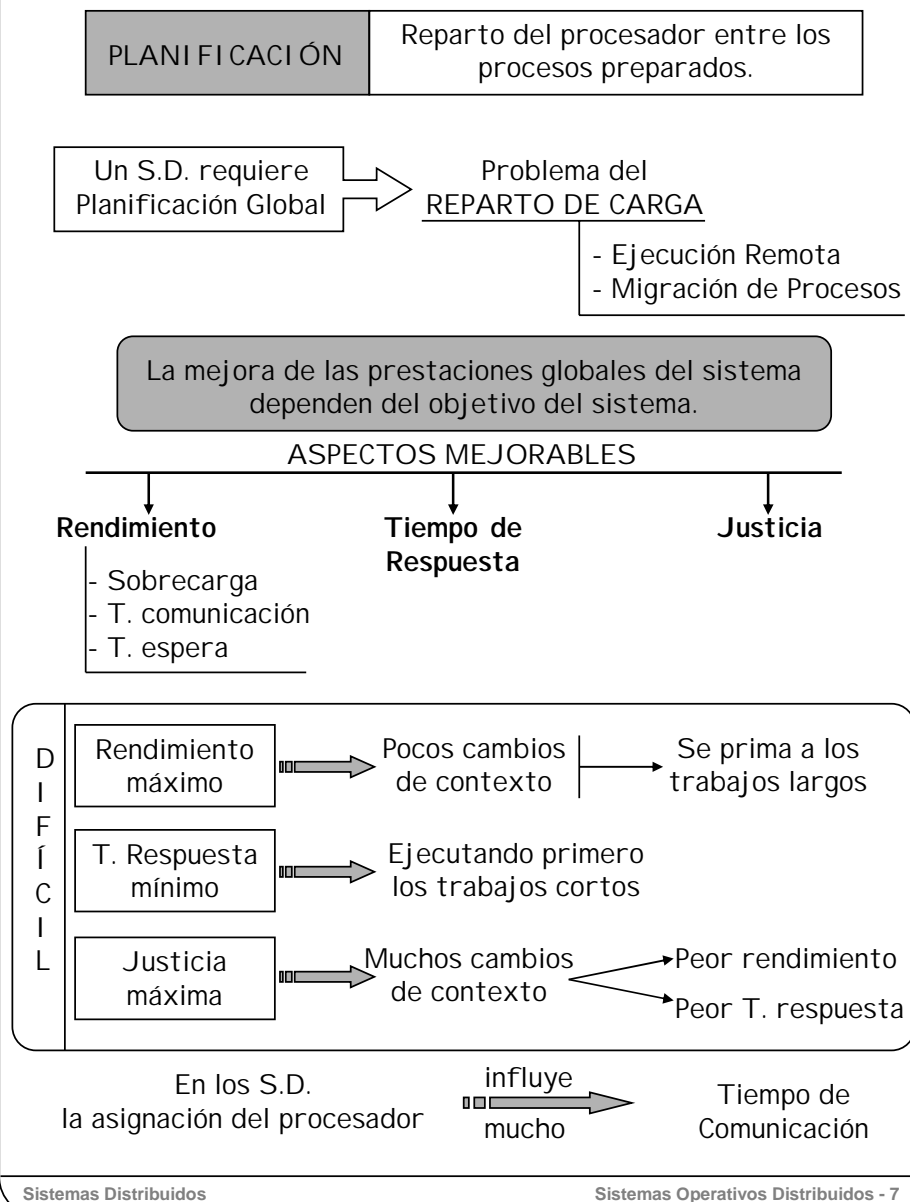


En un sistema centralizado, la asignación de los recursos disponibles está gestionada, como ya sabemos, por el sistema operativo. Para ejecutar un proceso, se le asigna memoria y se ejecuta sobre el (normalmente) único procesador del sistema. Sin embargo, en un sistema distribuido nos encontramos con que hay múltiples máquinas y procesadores, con lo que la decisión de dónde ejecutar un programa, es decir, sobre qué procesador, ya no es tan trivial.

En un sistema distribuido un proceso puede ejecutarse:

- Utilizando la propia estación de trabajo
- Utilizando otras estaciones libres

La elección del procesador sobre el que se debe ejecutar un programa en un sistema distribuido se realiza de acuerdo a una política de **Reparto de Carga de Trabajo**.



El planificador del sistema operativo se encarga del reparto o asignación del procesador entre los procesos preparados para ejecutarse. Esta asignación del procesador se realiza basándose en una política de planificación adecuada al propósito del sistema (tiempo compartido, tiempo real, etc.).

Aunque los sistemas distribuidos también se pueden dedicar al tiempo real, su principal objetivo inicial era el compartimiento de recursos, así que vamos a ocuparnos del reparto adecuado de procesos entre múltiples procesadores cuando lo que se persigue es la mejora de las prestaciones globales del sistema. Cuando la planificación global se realiza con esta política nos encontramos con el problema del **Reparto de Carga**.

El reparto de carga se puede realizar en dos momentos:

- Cuando se van a crear los procesos, eligiendo el procesador de ejecución, y necesitando para ello la posibilidad de la **ejecución remota de procesos**.
- Reasignando los procesadores a los procesos durante su ejecución, esto es, mediante la **migración de procesos**.

La mejora de las prestaciones globales del sistema dependen del objetivo del sistema. Se puede mejorar el rendimiento, el tiempo de respuesta, la justicia, o una combinación de las tres.

El **rendimiento** se define como la cantidad de trabajo desarrollado por unidad de tiempo. Trabajo útil significa programas ejecutados. Así, el rendimiento se maximiza minimizando la sobrecarga, el tiempo de comunicación y el tiempo de espera. **Sobrecarga** es el tiempo que se dedica el sistema operativo a la planificación y a los cambios de contexto. El **tiempo de comunicación** es el tiempo que pasan los procesos esperando a recibir datos o eventos, o esperando a que se vacíen buffers de salida de datos para poder continuar su ejecución. El **tiempo de espera** es el que pasan los procesos en la cola "preparados" esperando a que haya un procesador disponible.

El **tiempo de respuesta** es el tiempo que pasa desde la entrada de datos hasta que comienza la salida de resultados. Está claro que los usuarios interactivos buscan buenos tiempos de respuesta. Un planificador que minimice los tiempos de respuesta intentará minimizar la suma de los tiempos de respuesta de todos los procesos del sistema.

La **Justicia** es un concepto más esotérico. Si se mejora el rendimiento o los tiempos de respuesta, es muy posible que algún proceso nunca llegue a ejecutarse, sin embargo, un planificador justo dará a todos los procesos el mismo acceso a los procesadores.

Es fácil ver que maximizar el rendimiento, minimizar el tiempo de respuesta y asegurar justicia absoluta simultáneamente no es posible. El rendimiento es máximo cuando se minimizan los cambios de contexto, por lo que se prima a los trabajos largos sobre los cortos. El tiempo de respuesta, por contra, se minimiza ejecutando primero los más cortos. La justicia se obtiene mediante frecuentes cambios de contexto, lo cual no es bueno ni para el rendimiento ni para los tiempos de respuesta.

Los buenos planificadores de los sistemas distribuidos, además de intentar conseguir un equilibrio entre estos tres factores deben tener muy presente que la asignación del procesador influye mucho en el tiempo de comunicación, pues cuando dos procesos con mucha comunicación se asignan a equipos muy separados, los procesadores pueden quedarse ociosos esperando a que los mensajes viajen por la red, y además cuando un proceso que produce datos para otros consumidores no se ejecuta, los procesos consumidores quedan bloqueados en espera, lo cual afecta negativamente a los tiempos de comunicación.

En los apartados siguientes vamos a ocuparnos de los dos mecanismos de reparto de carga antes mencionados, es decir, de cómo arrancar la ejecución de un proceso en un nodo remoto, y de cómo un proceso que se está ejecutando en un nodo, puede trasladarse a otro para continuar allí su ejecución (migración).

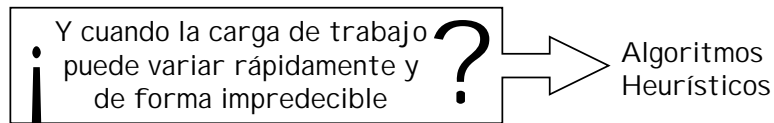
El primer paso para llevar a cabo la ejecución remota o la migración de procesos es la asignación de procesadores, por lo que comenzaremos por abordar las políticas de asignación de procesadores.

Consideraremos la planificación entre máquinas compatibles.

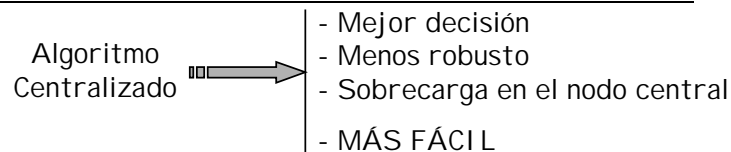
5 ASPECTOS SOBRE ALGORITMOS DE PLANIFICACIÓN

① ALGORITMOS DETERMINISTAS/HEURÍSTICOS

Los algoritmos deterministas son apropiados cuando se conocen todos los datos o se puede predecir.



② ALGORITMOS CENTRALIZADOS / DISTRIBUIDOS



③ ALGORITMOS ÓPTIMOS / CUASI-ÓPTIMOS

La solución óptima requiere mucho más trabajo

- | | |
|---|---|
| <ul style="list-style-type: none">- Mucho más tráfico- Mucho más proceso | que conformándose con una buena solución no óptima. |
|---|---|

Para simplificar el problema vamos a suponer que todas las máquinas son idénticas o, al menos, son compatibles en el código, y que difieren, como mucho, en la velocidad. En todo caso se puede pensar en disponer de listas de equipos, tal que todos los equipos de cada lista son homogéneos y compatibles. La única alternativa para poder ejecutar procesos sobre cualquier arquitectura (Intel, SPARC, MIPS, ...) es suponer que para cada programa se dispone de todos los ficheros ejecutables para las distintas arquitecturas, o que se dispone de los emuladores correspondientes.

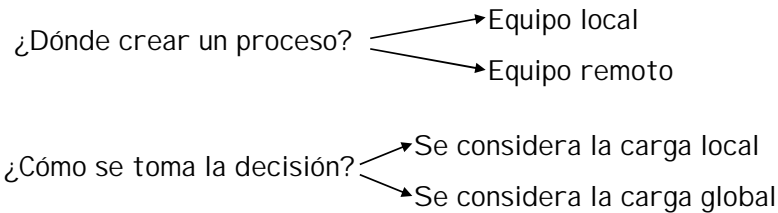
Hay muchos algoritmos propuestos para la asignación o reparto de procesadores. Aquí vamos a comentar cinco aspectos que se deben tener en cuenta a la hora de diseñar un algoritmo de asignación de procesador.

1) Algoritmos deterministas/heurísticos. Los algoritmos deterministas son apropiados para las situaciones en las que se conocen todos los datos (toda la lista de procesos y todos sus requisitos), o son fácilmente predecibles, por ejemplo, aplicaciones bancarias o reservas de líneas aéreas. Pero en otros casos, la carga de trabajo puede variar rápidamente y de forma impredecible, pues depende de quién está trabajando, y de lo que está haciendo. En estas situaciones la asignación del procesador no se puede realizar de una manera matemática y determinista, y es necesario utilizar técnicas heurísticas.

2) Algoritmos centralizados/distribuidos. La concentración de la información en un nodo central permite tomar una mejor decisión, pero es menos robusto y puede generar sobrecarga en el nodo central. Aunque suelen ser preferibles los algoritmos distribuidos, en algunos casos se utilizan algoritmos centralizados por no disponer de variantes distribuidas.

3) Algoritmos óptimos/casi-óptimos. Las soluciones óptimas pueden encontrarse tanto en los algoritmos centralizados como en los distribuidos, pero, obviamente, con mucho más trabajo (requiere mucho más tráfico de información y proceso) que conformándose con una buena solución que no sea la óptima. En la práctica, suelen utilizarse algoritmos heurísticos distribuidos y "casi-óptimos".

④ POLÍTICA DE TRANSFERENCIA



Los algoritmos globales dan una solución
ligeramente mejor
a un coste mucho mayor

⑤ POLÍTICA DE UBICACIÓN

En caso de ejecución remota

¿a qué procesador se envía el proceso?

La política de ubicación debe ser global
pues se requiere información sobre el
nivel de carga de los demás procesadores.

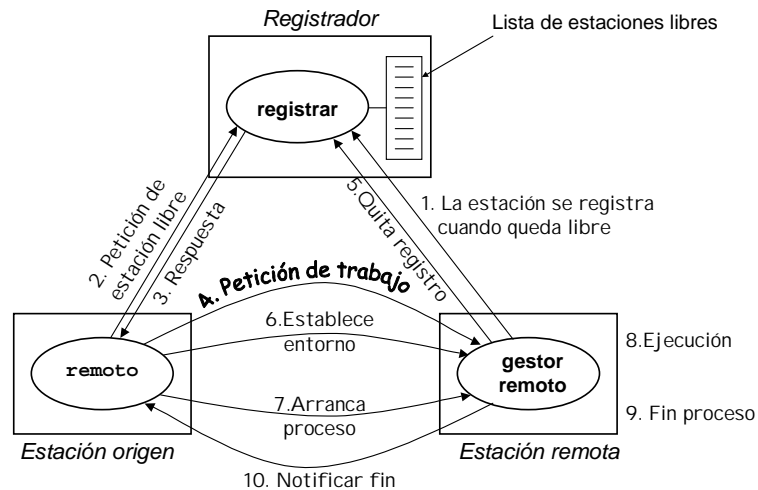
Los procesadores ociosos
difunden, a priori, su
condición de disponibles

El equipo emisor solicita
procesadores disponibles

4) Política de transferencia. Cuando se va a crear un proceso hay que tomar la decisión de si se ejecuta localmente o en un procesador remoto. Si la máquina está demasiado cargada, el nuevo proceso debe transferirse a un procesador remoto. La cuestión es si la decisión debe tomarse localmente o de una forma global (considerando el estado de carga de todos los procesadores). Unos abogan por la elección simple, de tal manera que si la carga de la máquina local pasa de cierto umbral, se envía a un equipo remoto. Otros piensan que esta decisión es demasiado simple, y debe consultarse el estado de los demás procesadores antes de tomar la decisión. Los algoritmos locales son más simples y más distantes de la solución óptima. Los algoritmos globales dan una solución solo ligeramente mejor, pero a un coste mucho mayor.

5) Política de ubicación. Una vez que la política de transferencia ha decidido que hay que deshacerse del proceso y enviarlo a otro procesador, la política de ubicación debe decidir a cuál. Claramente, esta política no puede ser local, pues se necesita información sobre la carga de los demás procesadores para tomar una elección. Esta información puede diseminarse por el sistema mediante dos técnicas. Una posibilidad es que el equipo emisor solicite ayuda o procesadores disponibles (algoritmos iniciados por el cliente). La otra opción es que los procesadores ociosos difundan a priori su condición de disponibles (algoritmos iniciados por el servidor).

Veamos a continuación un ejemplo de cada una de estas opciones.



El cliente solicita ayuda a un servidor de estaciones libres.

En primer lugar debemos establecer **¿qué es un ordenador libre?** Podríamos pensar que es un ordenador operativo en el que no está nadie conectado en su consola, ¡pero puede que tenga diversos procesos “demonios” en ejecución!, ¡o también puede ocurrir que el propietario se haya conectado por la mañana pero lleve horas tomando café! Aunque más adelante, en el apartado de “Nivel de Carga”, trataremos en profundidad cómo solventar este problema, una solución a primera vista podría consistir en que cada ordenador decida con su propia política cuándo está libre, y en ese caso se registre en un servidor de máquinas libres indicando que está disponible y cuáles son sus características (nombre, dirección, propiedades). Así, cuando un usuario desee una ejecución remota de un comando, lo único que tiene que teclear es:

```
remoto <comando>
```

El programa `remoto` se encarga de buscar en el servidor de estaciones libres una que se adapte a las necesidades de nuestro programa (procesador, s.o., etc.). Ahora que nuestro programa ya está en la máquina remota, ésta **debe tener el mismo entorno de trabajo** (la misma vista del sistema de ficheros, variables de entorno, etc.). Si el sistema de ficheros no es privado, no parece que esto presente problemas. Pero **¿qué pasa con las llamadas al sistema?**

¿QUÉ PASA CON LAS LLAMADAS AL SISTEMA?

- Accesos a discos locales - Lectura y escrituras en consola	→	Redirigidas a la estación de origen
- SBRK, NICE, PROFILE	→	Ejecutadas en la propia estación
- Fecha y hora	→	Problemas de consistencia

¿Qué pasa con los programas con acceso directo a dispositivos?

En cuanto a las llamadas al sistema, algunas se deberán redirigir al ordenador original (accesos a discos locales, lectura de teclado, escritura en pantalla), y otras se deben ejecutar en el ordenador remoto (SBRK, NICE, PROFILE, y en general, todas las preguntas sobre el estado de la máquina). Pero las llamadas al sistema relacionadas con la fecha y hora pueden ser un problema, pues es muy normal que todos los ordenadores de la red no tengan la hora sincronizada, y comandos como `make` pueden producir resultados incorrectos. Si estas llamadas se redirigen a la máquina original, se produce el correspondiente retardo, con lo que se continua sin disponer de una hora exacta y apropiada.

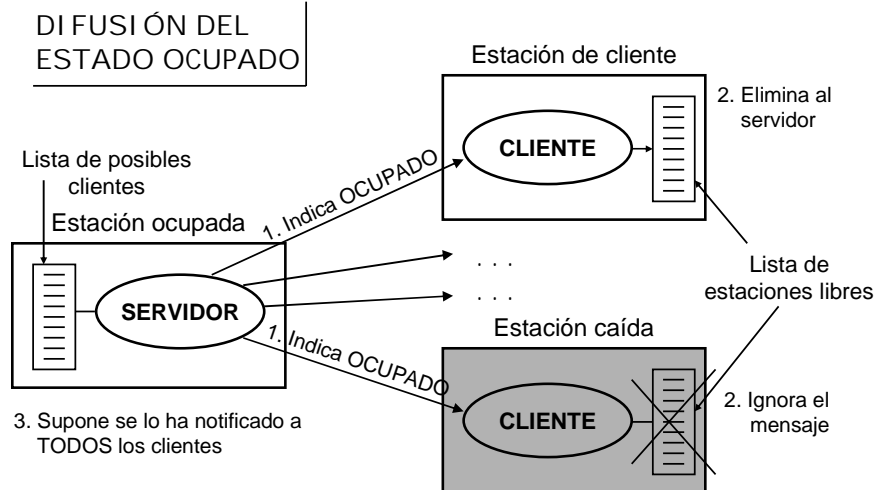
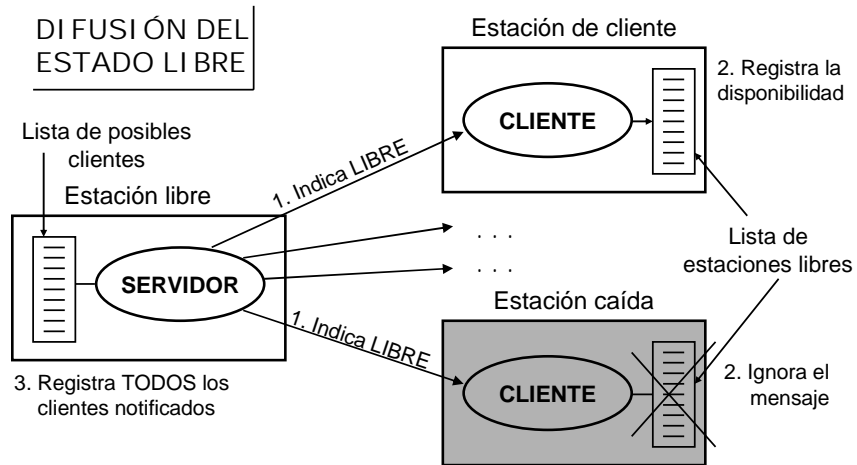
También es un problema la ejecución de programas con acceso directo a ciertos dispositivos hardware, como el ratón, la memoria mapeada de la pantalla, disquetes, etc.

SBRK establece un nuevo tamaño de un segmento de memoria.

NICE baja la prioridad del proceso que lo ejecuta.

PROFILE proporciona estadísticas de ejecución del proceso.

Asignación de Procesadores Una Política de Ubicación



Los procesadores libres difunden su condición.

La selección de un procesador para ejecutar un proceso puede realizarse basándose en la información almacenada en una base de datos en cada estación de trabajo. Esta base de datos contiene:

- Los servidores en los que esta estación está interesada (que le son útiles).
- Los clientes a los que le puede prestar sus recursos de proceso este equipo.

Solamente hay dos operaciones sobre esta base de datos: añadir al *pool* de servidores o quitar del *pool*.

Añadir al *pool* de servidores.

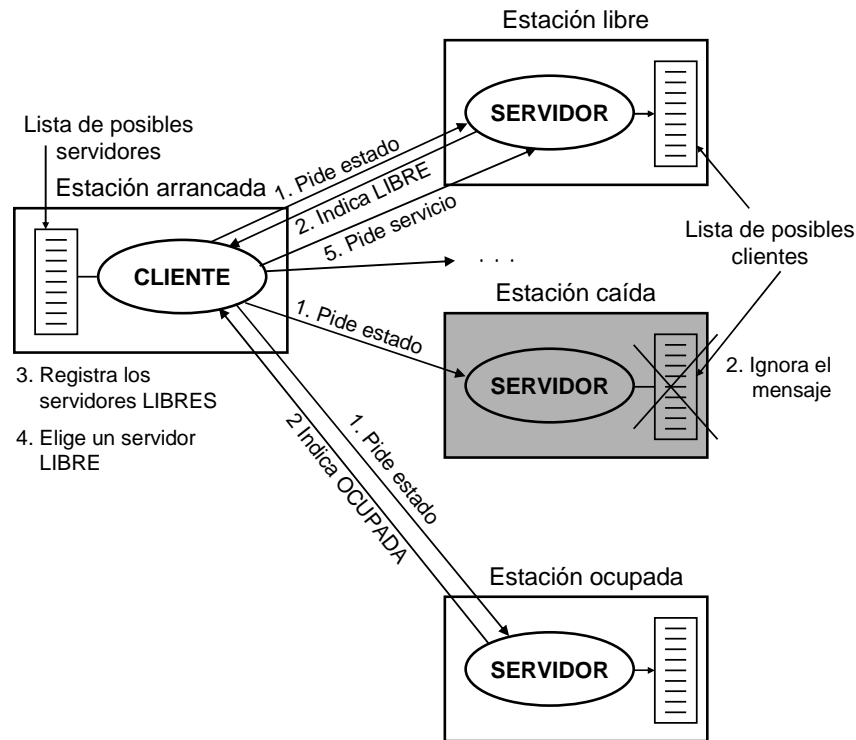
- Cuando un servidor decide ponerse a disposición del sistema difunde un mensaje a todos los clientes o a los que tiene en su base de datos como posibles clientes.
- Todos los clientes registran entonces la disponibilidad de este servidor.
- Si un cliente no recibe el mensaje de disponibilidad (si estaba apagado), el mensaje se ignora, y el servidor supone que lo ha recibido.

Sacar del *pool* de servidores.

- Cuando un servidor decide retirarse del *pool* de disponibles, difunde un mensaje a todos sus posibles clientes.
- Cuando un cliente recibe el mensaje, elimina a ese servidor de la lista de procesadores disponibles.
- Si un cliente no recibe el mensaje, se ignora, y el servidor supone que el cliente ha sido informado de que ya no está disponible para aceptar procesos.

Obsérvese que un servidor libre al que se le solicita la ejecución remota de un programa, no tiene por qué notificar inmediatamente a todos los posibles clientes que ya no está libre. Solamente debe hacerlo cuando estime que su nivel de carga está por encima de cierto umbral, es decir, que se considera "cargado". Más adelante veremos algunas aproximaciones de cómo un equipo puede estimar su nivel de carga.

¿...y cuando despierta un cliente caído?



Si un cliente va a ejecutar un proceso en un nodo remoto debe consultar su lista de servidores disponibles. Si el cliente ha estado activo y ha registrado todos los anuncios de disponibilidad de los servidores, puede realizar la selección basado en alguna política o estrategia de selección, por ejemplo FIFO o el servidor con menos carga. Pero si este equipo estaba apagado cuando se difundieron los ofrecimientos de los servidores, entonces debe recolectar ahora la información sobre los servidores disponibles. Esta recolección puede hacerse de la siguiente manera:

- 1) El cliente envía un mensaje a todos los servidores o a aquellos que le pueden ser útiles, preguntando por su estado de disponibilidad.
- 2) Los servidores apagados no escuchan este mensaje.
- 3) Un servidor que reciba el mensaje y no esté disponible responde negativamente.
- 4) Un servidor ocioso que reciba la petición de estado contesta indicando que está disponible.
- 5) Después de recibir las contestaciones, el cliente actualiza su base de datos de servidores disponibles y selecciona uno entre ellos para enviarle el proceso a ejecutar.

¿Cómo establece un equipo su nivel de carga?

SOLUCIÓN SIMPLE: Índice de Carga → Número de Procesos

¡Los "Demonios" Están Bloqueados!

e-mail
news
colas de impresión
gestores de ventanas
...

Suponen una
carga para el
sistema ?

SIGUIENTE PASO: Considerar Sólo los Procesos Activos

· Si el muestreo es poco frecuente
· Si coincide con un momento "despierto" → Medida Engañosa

Número de procesos: ¡DICE POCO !

MÁS PRÁCTICO: Calcular Porcentaje de CPU Ociosa

Hasta ahora hemos supuesto que todos los algoritmos conocen la carga de su propia máquina, por lo que pueden decidir si está sobrecargada o no y comunicárselo a los demás nodos de la red. Pero establecer el nivel de carga no es tan fácil como parece. Veamos algunas aproximaciones para establecer la carga de un equipo.

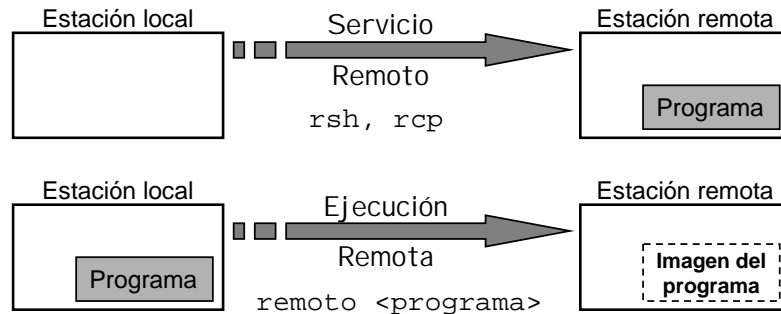
Una solución simple puede consistir simplemente en contar el número de procesos de cada máquina, y utilizar ese número como el índice de carga del sistema. Sin embargo, incluso en una estación ociosa puede haber muchos procesos arrancados, tales como los "demonios" del correo electrónico y de las *news*, de las colas de impresión o los gestores de las ventanas. Es decir, procesos que, aunque arrancados, pueden estar bloqueados o en espera de algún evento externo, por lo que no suponen ninguna carga de trabajo.

El siguiente paso es considerar solamente los procesos que están en ejecución o preparados. Sin embargo, ya que hay "demonios" que se despiertan periódicamente y se vuelven a dormir, puede suceder que se haga el muestreo en uno de estos momentos, con lo que la medida sería engañosa.

En cualquiera de estas dos primeras aproximaciones, este índice no dice mucho, pues lo que para una estación de trabajo puede suponer mucha carga, para otra estación con procesadores mucho más potentes puede suponer el estar ocioso la mayor parte del tiempo.

Una medida más directa para realizar la medición de la carga, aunque más costosa, es calcular el porcentaje de tiempo que la CPU está ociosa. Está claro que una máquina con un 20% de utilización de CPU está más cargada que otra con solo un 10%, independientemente del número de programas en ejecución. Una forma de medir esta utilización es programar un temporizador (*timer*) para que interrumpa periódicamente. En cada interrupción se comprueba si el proceso que está en ejecución es el proceso ocioso o no. De esta manera se puede calcular la proporción de tiempo que la CPU está ociosa.

Ejecución Remota



¿CUÁNDO ES CONVENIENTE?

Reparto General → Menor tiempo global de ejecución

Mejorar tiempo de ejecución en aplicaciones paralelas

Cuando se requieren servicios especiales.

Volumen de datos remotos de entrada > Tamaño del proceso local

REQUISITOS

- ✓ Debe haber un mecanismo de difusión de estado de carga
- ✓ El proceso remoto puede verse obligado a abandonar la ejecución
- ✓ La ejecución remota debe realizarse sin penalización

Depende de las facilidades disponibles:

Nivel de programación: RPC

Nivel de intérprete de comandos: `remoto <programa>`

Entornos
Heterogéneos

- Sintaxis y semántica de los intérpretes de cmds.
- Emuladores o traducción binaria (código y datos)

Se deben diferenciar estos dos conceptos:

Servicio Remoto: Ejecutar en un equipo remoto un programa residente en el sistema remoto (por ejemplo, `rsh`).

Ejecución Remota: Ejecutar en un equipo remoto un programa residente en el equipo local.

¿Cuándo es conveniente la ejecución remota de procesos? Hay diversas situaciones en las que un reparto de carga mediante ejecución remota mejora el rendimiento global de un sistema distribuido:

- Reparto general de la carga entre los procesadores disponibles para conseguir un menor tiempo global de ejecución.
- Mejorar el tiempo de ejecución de aplicaciones paralelas. Distribuyendo un proceso en sub tareas se puede conseguir una disminución del tiempo total de ejecución.
- Ejecución remota de procesos que requieren servicios especiales en las máquinas que mejor proporcionan tales servicios. Por ejemplo, acudiendo a un equipo con procesadores específicos para cálculo numérico.
- El volumen de los datos de entrada es mayor que el espacio de memoria ocupada por el proceso. Por ejemplo, cuando se realizan análisis estadísticos con una gran cantidad de datos residentes en otro equipo, requiere menos tiempo la comunicación para mover el proceso al otro equipo y ejecutarlo allí, que la comunicación para acceder a todos los datos de entrada.

Requisitos para la ejecución remota. Hay tres puntos que se deben tener en cuenta en el diseño de los mecanismos de ejecución remota:

- Debe haber un mecanismo o protocolo para difundir el estado de carga de una estación. Esto está directamente asociado con los mecanismos de gestión general de recursos de todo el sistema.
- Cuando se selecciona una estación de trabajo para una ejecución remota, el tiempo de respuesta de esa estación para la ejecución de cualquier programa se degrada. Si además el usuario de ese ordenador remoto empieza a trabajar, puede notar unas bajas prestaciones. En esta situación, el proceso cuya ejecución se ha asumido puede verse obligado a abandonar la ejecución para no degradar más las prestaciones del sistema remoto.
- La ejecución remota deberá tener lugar tan fácilmente como si se realizara localmente, sufriendo las menos penalizaciones posibles. Esto quiere decir que la ejecución de un proceso deberá ser independiente de la ubicación.

Este último punto no es fácil de cumplir al cien por cien, y depende de las facilidades que se le ofrezcan a los programas remotos:

- A nivel de programación: Llamadas remotas a procedimientos (RPC).
- A nivel del intérprete de comandos. Por ejemplo, algo parecido a las utilidades ofrecidas por Unix de Berkeley, como la copia remota de ficheros (`rcp`) o la ejecución remota (`rsh`). (¡Ojo! `rcp` y `rsh` no son ejemplo de ejecución remota, sino de servicio remoto). LOCUS ofrece estos mismos comandos de forma transparente.

Entornos homogéneos/heterogéneos. Una de las principales cuestiones en la ejecución remota es la facilidad con la que los recursos remotos pueden ofrecerse a los clientes. Claramente va a resultar más fácil la ejecución remota en un entorno semejante al propietario del proceso (arquitectura, procesador, ...) que en entornos heterogéneos. Cuando ambos entornos son homogéneos lo único que hay que hacer es indicar las características del entorno de ejecución necesario (memoria, ficheros, etc.).

El problema principal de la ejecución en un sistema remoto distinto del propietario del proceso es la descripción del servicio a ejecutar, ya que hay diferencias a dos niveles:

- Sintaxis y semántica de los intérpretes de comandos de los sistemas operativos.
- Cuando las arquitecturas son distintas, se requiere o bien traducción binaria (para código y datos), o bien la ejecución interpretada mediante emuladores.

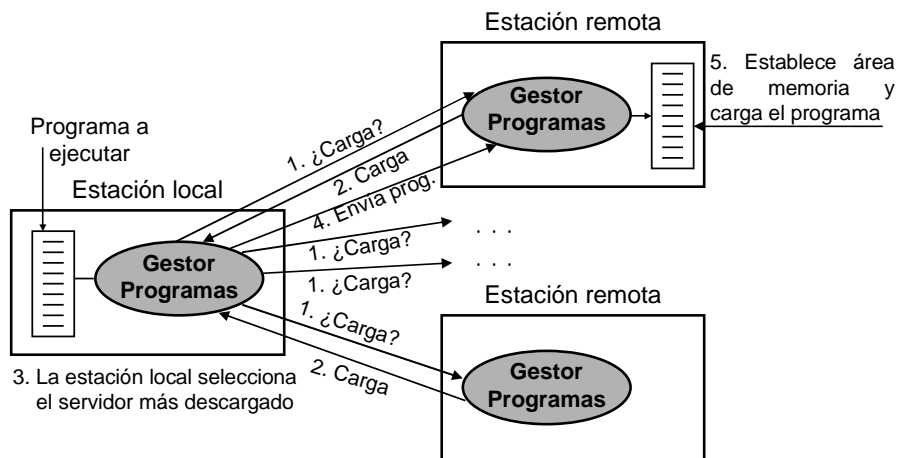
Ejecución Remota en el V-System

Intérprete de Comandos

Llamadas al Sistema

<programa><argumentos>@<máquina>

<programa><argumentos>@*



El programa se ejecuta con transparencia de red debido a:

1. Espacio de direcciones igual que en equipo origen
2. Las ref. fuera del espacio de dir. → equipo origen
3. Los servicios del G. de programas y del kernel son iguales que los ofrecidos a los procesos locales.

REQUISITO: No acceder directamente al hardware.

La ejecución remota de programas en V-System se arranca escribiendo esto:

<programa><argumentos>@<máquina>

para ejecutarlo en un equipo concreto. O bien se puede escribir lo siguiente para que se elija un nodo al azar:

<programa><argumentos>@*

Se ofrece una biblioteca o librería para proporcionar la misma funcionalidad a los programas.

La ejecución remota de un proceso en una máquina al azar requiere dos etapas: inicialización y ejecución. Para **inicializar** un proceso primero hay que enviar un mensaje de petición a todos los gestores remotos de programas solicitándoles su estado de carga o de recursos disponibles. Se recibirá una o más respuestas. El equipo cliente seleccionará al gestor de programas que responda en primer lugar, por suponer que será el menos ocupado. A continuación se le envía una petición al gestor remoto de programas para que establezca un área de memoria principal y cargue en ella la imagen del programa a ejecutar.

La **ejecución** de programas en V-System se realiza en un entorno de ejecución con transparencia de red. Esta facilidad y la condición de que los programas no accedan directamente al hardware de la máquina hace que la ejecución remota de programas parezca como si la ejecución fuera local.

Esta transparencia en la ejecución se consigue por lo siguiente:

- 1) El espacio de direcciones del programa se inicializa en la máquina remota de igual manera a como se haría en la máquina propietaria del proceso.
- 2) Todas las referencias que realice el programa fuera de su espacio de direcciones se redirigen a la máquina original mediante primitivas transparentes a la red.
- 3) Los servicios proporcionados por el gestor de programas y por el *kernel* a los procesos remotos son exactamente los mismos que los ofrecidos a los procesos locales.

Los Procesos Son Muy Dinámicos

- ¿Qué procesos se van a crear?
- ¿Qué procesos se van a destruir?
- ¿Cuándo?
- ¿Carga del proceso?

Difícil
planificación estática
a largo plazo

Planificación Dinámica
o
MIGRACIÓN

Se Desarrolla en Dos Fases

①

Planificación

- ☞ ¿Cuál?
- ☞ ¿Dónde?

②

Transferencia

(Estado del Proceso
Espacio de Direcciones)

- ☞ ¿Cuánto?
- ☞ ¿Cuándo?

Como ya sabemos los procesos tienen una naturaleza muy dinámica, de tal manera que en un sistema los procesos se crean y destruyen continuamente sin poder tener un conocimiento adelantado de qué procesos y cuándo se van a crear o destruir. La ejecución remota de programas mejora esta situación, pero lo que nunca se sabe a priori es la carga que realmente va a generar cada proceso. Por esto, la asignación o planificación estática de procesos no garantiza un aprovechamiento óptimo del sistema, y se hace necesaria la planificación dinámica que proporciona la migración de procesos.

La idea básica de la migración es la planificación dinámica de procesos, es decir, la posibilidad de mover un proceso, durante su ejecución, de un procesador a otro, de tal forma que continúe la ejecución en otro procesador distinto con acceso completo a todos los recursos que estaba utilizando en el procesador anterior. Esta operación puede arrancarse sin que lo sepa ni el propio proceso ni ninguno de los procesos con los que está interactuando. En cualquier caso, la migración de procesos debe mantener todos los elementos de cálculo y de comunicación que esté utilizando.

La migración de procesos puede realizarse entre sistemas homogéneos o heterogéneos. Aquí vamos a considerar solamente la migración entre sistemas homogéneos, dada la complejidad adicional que supone migrar procesos a otras arquitecturas distintas.

La migración de procesos se desarrolla en dos fases:

1º Planificación (¿cuál?, ¿a dónde?)

2º Transferencia del proceso (¿cuánto? y ¿cuándo?)

La planificación consiste en tomar una decisión para responder a estas dos preguntas: ¿qué proceso va a ser transferido? y ¿a qué procesador va a ser transferido?

Como veremos enseguida, para poder contestar a estas preguntas, lo primero que se debe hacer en la fase de planificación es obtener datos para poder tomar decisiones.

La transferencia del proceso se refiere a los mecanismos utilizados para realizar la transferencia del proceso a otro procesador. La transferencia de un proceso distribuido se realiza en estas dos fases:

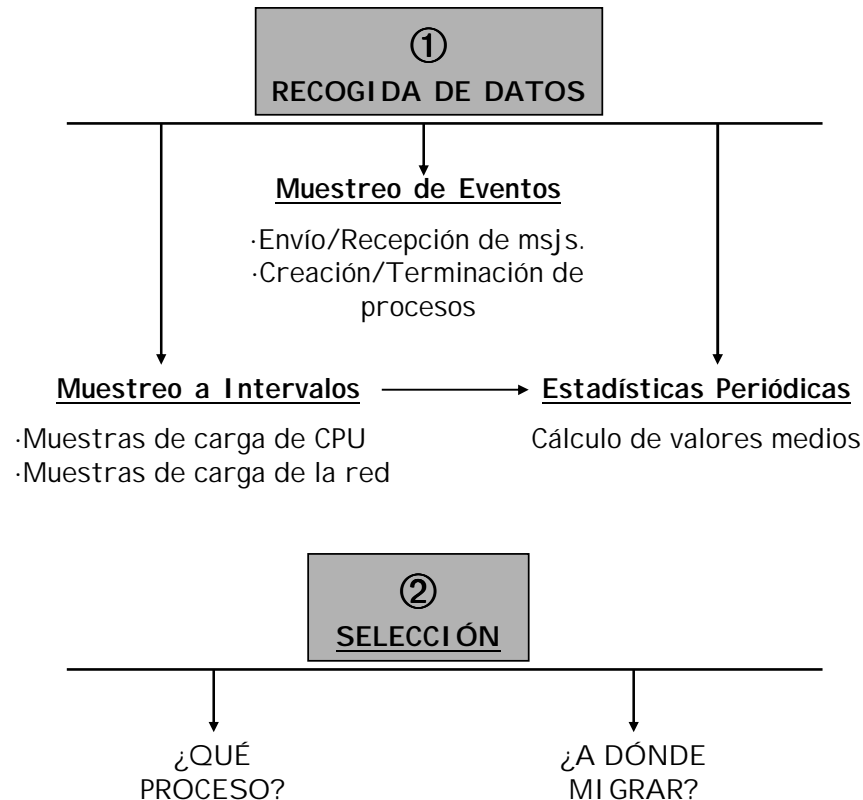
1. Transferencia del estado del proceso.

2. Transferencia de su espacio de direcciones (la memoria que ocupa).

Las preguntas que deben hacerse en esta fase es ¿qué cantidad del proceso va a transferirse? y ¿cuándo se va a reanudar la ejecución del proceso?

En los siguientes apartados vamos a abordar algunos aspectos de estas dos fases de la migración, ayudándonos, en algunos casos, con los ejemplos de la implementación de algunos sistemas concretos.

La Planificación se Realiza en Dos Pasos



En la migración de procesos, la planificación se realiza en dos pasos:

1º. Recogida de datos

En general, los datos estadísticos pueden recogerse mediante estos tres modos:

- **Muestreo de eventos.** Operaciones tales como el envío/recepción de mensajes y de creación/terminación de procesos se encargan de registrar tales operaciones para obtener una estadística de comunicaciones.
- **Muestreo a intervalos.** A intervalos fijos de tiempo se toman muestras de la carga de la CPU (expresada por el número de procesos preparados) y la carga de la red (indicada por el número de mensajes en espera de ser transmitidos).
- **Estadísticas periódicas.** Cada cierto número de muestras tomadas como se indica en punto anterior, se calculan valores medios.

2º. Selección

Cuando ya se tiene información sobre el estado de todo el sistema distribuido, se deben tomar las decisiones: qué proceso migrar y cuál es el procesador de destino. Para ello se debe tener en cuenta tanto el estado de carga de cada estación, o de cada procesador, como el estado de congestión de la red.

Planificación en el Sistema Charlotte

Información Recogida

✓CARGA DE CADA ORDENADOR

- N° total de procesos
- Enlaces de comunicaciones
- Carga media de CPU

✓RECURSOS POR PROCESO

- Antigüedad
- Carga media de CPU

✓DATOS DE COMUNICACIONES (por proceso)

- N° paquetes enviados y recibidos por los enlaces de comunicación más activos

Intervalo de Toma de Datos

50 - 80 ms.

Datos Estadísticos

Cada 100 tomas

Sobrecarga \approx 1% del total de CPU

A modo de ejemplo, comentaremos, muy por encima, unas notas sobre la información que utiliza el *kernel* o núcleo del sistema Charlotte para tomar las decisiones de la migración. La información que recoge, y en la que basa la selección de cuál, dónde y cuándo, es la siguiente:

- Carga de cada ordenador de la red: número de procesos (activos o no), enlaces de comunicaciones y la carga media de la CPU de cada nodo (porcentaje de tiempo que no se ejecuta el proceso ocioso).
- Recursos de proceso utilizados: Antigüedad y carga media y total de la CPU por cada proceso.
- Datos de comunicaciones: El número de paquetes enviados y recibidos por los enlaces de comunicación más activos de cada proceso.

Estos datos se recogen periódicamente cada 50 a 80 milisegundos, ofreciendo datos globales estadísticos cada 100 intervalos o tomas de datos. La sobrecarga que genera en el sistema la recogida de datos es menor del 1 por ciento del tiempo total de CPU.

Hay que
Transferir

ESTADO DEL PROCESO

- | | |
|--------------------------------|--------------------|
| • Estado Interno (contexto) | Buzones |
| • Estado de las Comunicaciones | Colas de envío |
| | Colas de recepción |
| | Mensajes en curso |

ESPACIO DE DIRECCIONES

- | | |
|----------|------------|
| • Código | → ¿Cuánto? |
| • Datos | |
| • Pila | |
| | → ¿Cuándo? |

Problemas

- ✓ Determinar el Estado del Proceso
- ✓ Determinar el Espacio de Direcciones a Transferir
- ✓ ¿Cuándo Reanudar la Ejecución?

La transferencia de un proceso hacia el ordenador elegido implica la transferencia del estado del proceso y de su espacio de memoria. Esto va a suscitar los siguientes problemas o situaciones relevantes que iremos tratando:

- Determinación y transferencia del contexto o entorno de ejecución
- ¿Qué parte o porcentaje de su espacio de direcciones se transfiere al nuevo procesador?
- ¿Cuándo reanudar la ejecución del proceso en el procesador de destino?

Una de las primeras cosas que hay que hacer para comenzar la transferencia de un proceso es **determinar claramente su estado o contexto de ejecución**, separar el proceso de su entorno actual, transferirlo con toda su información de contexto y restablecerla en su nuevo entorno.

La determinación del **estado de un proceso** no es nada simple, pues éste no está definido solamente por su **estado interno** (o lo que conocemos por contexto), es decir, el contenido de los registros del procesador y el contenido de su espacio de direcciones (segmento de código, segmentos de datos y pila), sino también por el **estado de sus comunicaciones** con otros procesos, esto es, buzones o colas de envío y recepción de mensajes, e incluso los mensajes que se estén transmitiendo en el momento de la migración.

Aunque la determinación del estado interno del proceso y las colas de mensajes con procesos locales puede no resultar muy complicado, pues es información local al proceso, lo que sí resulta difícil es establecer el estado de las comunicaciones con los procesos remotos. En concreto: ¿qué pasa con los mensajes que le están enviando al proceso que está a punto de migrar?

Veámoslo a continuación.

Entrega de Mensajes

¿Qué Hacer con los Mensajes Dirigidos al Proceso Migrado?

Enlaces en el
Equipo Original
(Redirección)

Actualizar
Direcciones en
Equipos Comunicados

Costoso si no se
dispone de
traducción dinámica

- Anfitriones Modificados
- Dependencia Residual

S O L U C I O N E S

REDIRECCIÓN
DE MENSAJES

PREVENCIÓN DE
PÉRDIDA DE MENSAJES

RECUPERACIÓN DE
MENSAJES PERDIDOS

No solamente es difícil determinar y guardar el estado de comunicaciones de un proceso que va a migrar (colas de mensajes recibidos y a enviar), también hay que preocuparse de los mensajes futuros que van a enviarle al proceso migrado cuando haya cambiado de ubicación. Es decir, **¿qué hacer con los mensajes que van dirigidos al proceso migrado?**

Hay dos enfoques para solucionarlo. Una posibilidad es establecer unos enlaces en el equipo original del proceso, de tal forma que los mensajes que le llegan los redirigen hacia el nuevo anfitrión. Esto tiene dos inconvenientes:

- Deja modificado el entorno de los equipos por los que va pasando.
- Dependencia residual: El equipo migrado depende de todos los equipos por los que ha pasado, de tal forma que se ve afectado por la caída de cualquiera de ellos.

Para evitar esto tenemos otra posibilidad, aunque bastante más costosa, pues supone la modificación y actualización de su nueva dirección en todos los equipos con los que el proceso migrado mantiene alguna comunicación. Si el proceso migrado es un servidor de algún servicio, también será necesario realizar el apunte necesario en los servidores de nombres afectados.

Esto se podría realizar fácilmente si el sistema dispusiera de traducción dinámica de direcciones, lo cual no es habitual. Obsérvese que los nombres estructurados (nombres de objetos) suelen incluir un nombre de máquina, lo cual dificulta notablemente el proceso de traducción, pues todas las referencias al proceso migrado deben actualizarse.

Hay tres métodos para solucionar este problema. Basado en el primer enfoque está la *redirección de mensajes*, mientras que considerando el segundo enfoque tenemos la *prevención de pérdida de mensajes* y la *recuperación de mensajes perdidos*. Veamos a continuación estas soluciones con un poco de detalle.

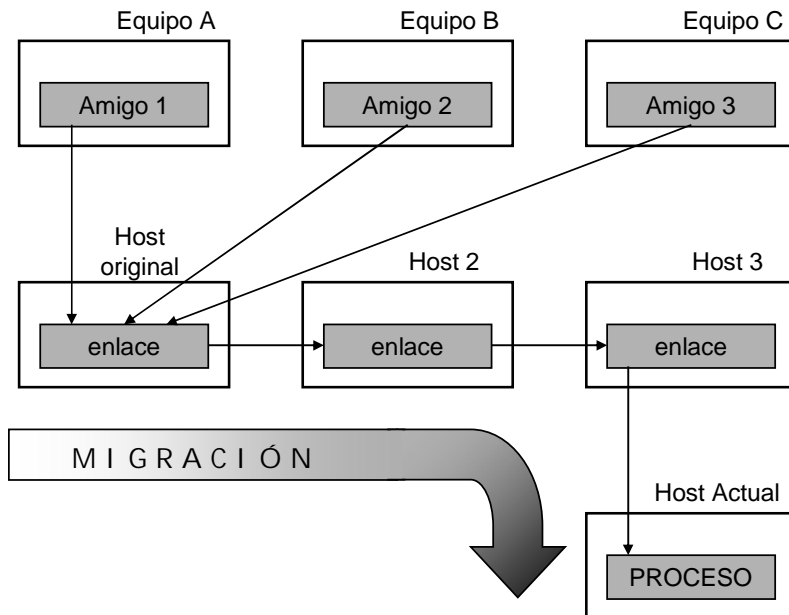
① REDIRECCIÓN DE MENSAJES

Deja en nodo origen un enlace de redirección al nodo destino

Los nodos comunicados
siguen enviando a la
antigua dirección

- ✓ Nodo Origen Responsable de Reenviar los Mensajes (**Dependencia Residual**)
- ✓ Deja modificados los equipos anfitriones y sobrecargados
- ✓ Incremento de Comunicaciones

SÓLO FACTIBLE si los anfitriones son FIABLES

**1. Redirección de Mensajes.**

Según este esquema, cuando se suspende un proceso en su ordenador origen para comenzar la migración, se apunta en el nodo origen la dirección del nodo destino del proceso, y todos los mensajes que le lleguen se guardan en un buffer. Una vez se ha transferido el proceso a su destino, el nodo origen le reenvía los mensajes del buffer, y el proceso continúa sin informar de su nueva ubicación a los procesos con los que mantiene alguna comunicación. Esto significa que van a seguir enviando los mensajes a la antigua dirección del proceso, por lo que la estación original del proceso va a ser responsable de reenviar estos mensajes a la nueva dirección del proceso migrado.

Esta solución es factible solamente si la máquina origen va a seguir siendo fiable después de la migración. Obsérvese que si un proceso migra varias veces, debe confiar en todos los ordenadores por los que ha ido pasando (dependencia residual). Esto supone, además de que los entornos de las máquinas por las que pasa quedan modificados, un incremento en el tráfico de comunicaciones y una carga adicional para los nodos por los que se va pasando.

② PREVENCIÓN DE PÉRDIDA DE MENSAJES

1. El Sistema de Migración avisa a los procesos en comunicación que el proceso va a migrar.
2. El proceso comienza la migración.
3. Si durante la migración hay mensajes, el equipo origen los guarda en un buffer.
4. Terminada la migración, el proceso comunica su nueva dirección.
5. El nodo origen reenvía los mensajes recién recibidos.
6. Los nuevos mensajes llegarán a la nueva dirección.

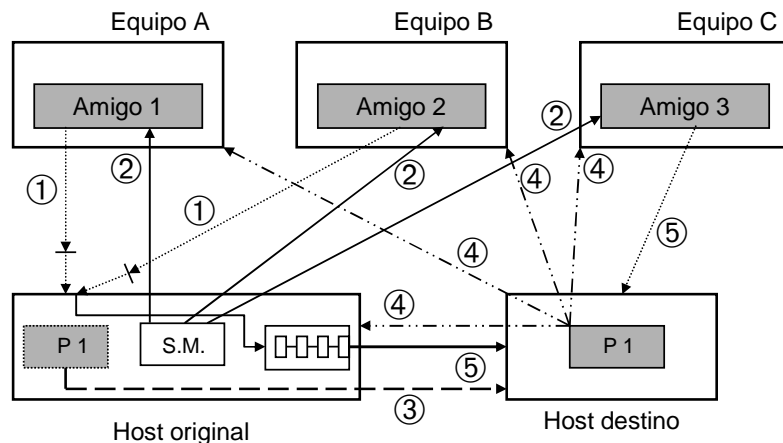


- ✓ Menos carga de comunicaciones
- ✓ No hay dependencia residual
- ✓ Deja inalterados a los equipos intermedios



Todos los nodos colaboran en la migración

SOLUCIÓN MÁS COMPLICADA



2. Prevención de Pérdida de Mensajes.

En este caso, el sistema de migración del nodo origen se encarga de avisar a todos los procesos con los que tiene comunicación el proceso a migrar, informándoles de la intención de migrar al proceso. Durante la migración, si todavía llega algún mensaje a la antigua dirección, se guarda en un buffer. Una vez terminada la transferencia del proceso, éste se lo comunica al ordenador origen y a los demás nodos relacionados, informándoles de su nueva dirección, por lo que los nuevos mensajes se le enviarán directamente a su nuevo nodo.

Ya que según este enfoque deben cooperar los mecanismos de migración tanto de los nodos origen y destino del proceso, como de los nodos con los que mantiene comunicación, esta solución es mucho más complicada que la redirección de mensajes. Por otra parte, el tráfico de mensajes es mucho menor que en la solución anterior, y además el proceso migrado no depende tampoco del nodo origen.

Descripción del Escenario Inferior de la Transparencia.

El proceso P1 va a migrar del *host* original al *host* destino.

1. Los procesos Amigo 1 y Amigo 2, de los equipos A y B, le envían sendos mensajes al proceso P1, pero antes de que lleguen al destino, en el nodo original se toma la decisión de migrar al proceso P1.
2. El Sistema de Migración comunica a los procesos relacionados con P1 la intención de migrar al proceso, lo cuales suspenden el envío de mensajes a P1.
3. El proceso P1 comienza la transferencia. En el equipo original se reciben los mensajes que habían enviado los procesos Amigo 1 y Amigo 2, los cuales se guardan en un buffer.
4. Cuando P1 reanuda su ejecución en el nodo destino, se lo comunica al nodo original, y a los procesos amigos les envía su nueva dirección. Se reanuda el envío de mensajes a P1.
5. El nodo original envía a P1 los mensajes recibidos durante la transferencia. El proceso Amigo 3 envía un mensaje a P1, ahora ya directamente a su nueva dirección.
6. El nodo original puede deshacerse del buffer de mensajes para P1. Ya no queda rastro de P1 en su anterior anfitrión.

③ RECUPERACIÓN DE MENSAJES PERDIDOS

Similar al enfoque anterior, pero ...

No se informa que se va a migrar

El nodo origen no guarda mensajes

Cuando se reanuda la ejecución, se comunica la nueva dirección



✓ No requiere colaboración de los demás nodos

✓ En simple

✓ Es rápido → Apropiado para momentos críticos



Hay que recuperar los mensajes perdidos



Requiere
Protocolo Robusto

3. Recuperación de Mensajes Perdidos.

Esta solución es similar a la anterior, pero en este caso, el nodo origen no guarda la dirección del nuevo ordenador. Es más, ni siquiera se informa a los nodos con los que mantiene comunicación de la intención de migrar, por lo que se pierden todos los mensajes que se le envían durante la transferencia. No obstante, antes de la transferencia se guardan las direcciones de todos los nodos con los que se comunica el proceso a migrar, de tal manera que una vez que ha llegado al nodo destino y reanuda su ejecución, les informa de su nueva dirección. Lo único que queda es recuperar los mensajes perdidos, lo cual resulta fácil si se dispone de un protocolo robusto de comunicaciones.

Este método no necesita colaboración con los demás nodos durante la transferencia, solamente después. Es simple y rápido, por lo que puede utilizarse en sistemas en los que el tiempo disponible para la migración es crítico; por ejemplo, cuando el ordenador origen detecta un fallo y no va a poder continuar la ejecución (por ej. fallo de tensión). Como ya hemos dicho, solamente se requiere un protocolo robusto de comunicaciones.

Transvase de Memoria

La migración consume tiempo
NO OLVIDAR

Beneficio de la Migración > Coste de la Migración

¿CUÁNTAS PÁGINAS SE DEBEN TRANSMITIR?

CARNEGIE MELLON / ACCENT SOBRE MACH

Se transfiere el estado del proceso + conjunto de trabajo
Al reanudar la ejecución, se accede al resto bajo demanda

- ⊗ Ejecución ligeramente más lenta
- ⊗ El proceso tiene su estado repartido (dependencia residual)
- ⊗ Ocupa memoria en los anfitriones anteriores
- ☺ No se transmiten páginas que no se van a referenciar

STANFORD / V-SYSTEM

Se van enviando las páginas "sucias"
Las páginas enviadas se marcan como "limpias"
Si se modifican, se marcan como sucias

La ejecución se suspende para enviar estado + cjto. trabajo.

- ⊗ Transmite páginas que no va a utilizar
- ☺ No genera dependencia residual
- ☺ Libera la memoria en los anfitriones anteriores.

A la hora de realizar una planificación para el reparto de carga, se debe tener muy en cuenta el coste de las operaciones. Por ejemplo, migrar un proceso que ocupa 10 Mbytes de memoria a través de una red de ethernet de 10 Mbps, supone 8 segundos solamente para la transferencia del contenido (por copia directa o *direct copy*). Obviamente, el beneficio de la migración deberá superar el coste de la operación, es decir, que al proceso migrado le deberían quedar al menos 8 segundos de tiempo total en su equipo original.

Esto indica que los tiempos de migración deberán ser lo más pequeños posible o, más concretamente, que el proceso migrado deberá permanecer suspendido por la migración el menor tiempo posible.

Veamos dos soluciones para realizar la transferencia del espacio de direcciones.

En la universidad de Carnegie Mellon, Zayas construyó en 1987 un mecanismo de migración en Accent (sobre Mach) en el que en la migración se transmitía el estado del proceso junto con las páginas de memoria que formaban el conjunto de trabajo del proceso. Cuando éste reanudaba su ejecución en el nuevo destino, el resto de las páginas de memoria se suministraban bajo demanda a través de la red (*copy-on-reference*). Esto hace que los tiempos de ejecución se puedan incrementar ligeramente.

En la universidad de Stanford se tomó otro enfoque. En el System-V, cuando se va a proceder a la migración, el proceso continua la ejecución en el nodo original mientras sus páginas de memoria se van copiando al nuevo ordenador. A medida que se van copiando páginas, se van marcando como "limpias", pero si el proceso accede posteriormente a alguna de ellas y la modifica, se marca como "sucias". Cuando la primera transferencia de páginas ha terminado, se hace otra ronda de copias para pasar de nuevo las páginas marcadas como "sucias". Y así sucesivamente hasta que solo queda por transmitir la serie de páginas que conforman el conjunto de trabajo. El efecto de este algoritmo es que el proceso a migrar se suspende, como máximo, el tiempo requerido para copiar las páginas del conjunto de trabajo, a las que está accediendo continuamente para modificarlas.

Como puede verse, aunque estos dos enfoques son muy diferentes, el efecto es muy similar. La ventaja del método de Zayas es que no se pierde tiempo en transmitir por la red las páginas que ya no se van necesitar, aunque, por contra, tiene el inconveniente de que un proceso que migre varias veces tiene su estado repartido entre varias máquinas de la red después de haber completado las migraciones, lo que le hace más sensible a las caídas de otras máquinas de la red. El método utilizado por el System-V tiene el inconveniente de que algunas páginas hay que transmitir las varias veces por la red, pero no deja la memoria distribuida entre varios equipos y, además, libera la memoria en el ordenador origen más rápidamente, lo que supone un beneficio para el equipo original.

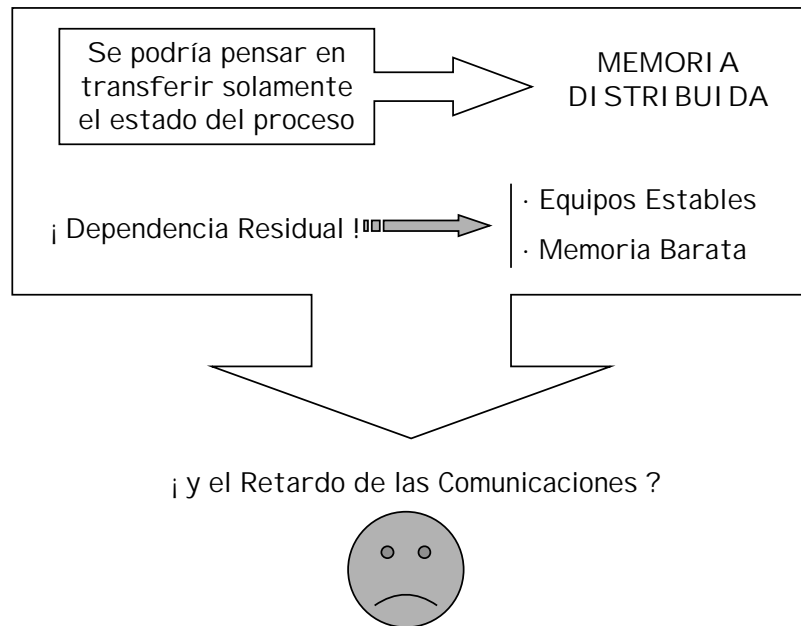
Transferencia ¿Cuándo Reanudar el Proceso?

Se debe minimizar el “tiempo de congelación”

Se pierde tiempo en transferir el espacio de direcciones

¿ Mayor espacio de direcciones \Rightarrow Mayor tiempo de congelación ?
lineal

Depende, más bien, del conjunto de trabajo



Otra cuestión asociada con la migración de procesos es **¿cuándo puede reanudar su ejecución un proceso migrado?**

El tiempo que permanece suspendido un proceso durante la migración se denomina *tiempo de congelación*. Obviamente, interesa reducir este tiempo lo más posible.

Este problema lo genera el crecimiento lineal del tiempo que supone el coste de la migración a medida que aumenta el tamaño en memoria del proceso. Esto es así cuando el espacio de memoria a transmitir es significativamente mayor que el ancho de banda disponible en la red y porque, en la mayoría de los sistemas que proporcionan migración de procesos, el proceso migrado no comienza su ejecución en el nuevo destino hasta que se ha transferido una copia completa de su estado y de su todo su espacio de direcciones (copia directa) al nuevo nodo.

Excepciones a esto son los mecanismos que acabamos de ver en el apartado anterior, que permiten al proceso migrado comenzar su ejecución en el destino casi inmediatamente después de suspender su ejecución en su equipo origen.

En el caso de transmitir solamente las páginas que constituyen el conjunto de trabajo, hemos visto que generan dependencia residual, lo cual no supone ninguna ventaja. No obstante, algunos ven que esta distribución de la memoria de un proceso entre varios equipos se adapta a la imagen de la memoria distribuida. Si suponemos que los equipos son estables y fiables, y si se tiene en cuenta que la memoria está cada vez más barata, podría pensarse en transferir solamente el estado del proceso, y no su espacio de direcciones, lo cual podría realizarse con extraordinaria rapidez, o lo que es lo mismo, la migración de procesos apenas supondría sobrecarga. En cualquier caso, no hay que olvidar los retrasos debidos a la red de comunicaciones, sin duda más lenta que el acceso a una memoria local.

Un Ejemplo: Charlotte

CARACTERÍSTICAS

- ✓ Separa las decisiones de los mecanismos
 - Planificación → Espacio de usuario
 - Mecanismos de migración → Núcleo
- ✓ Migración transparente a los procesos implicados
- ✓ Puede abortarse en cualquier momento

TAREAS DEL KERNEL

Atender Llamadas al Sistema

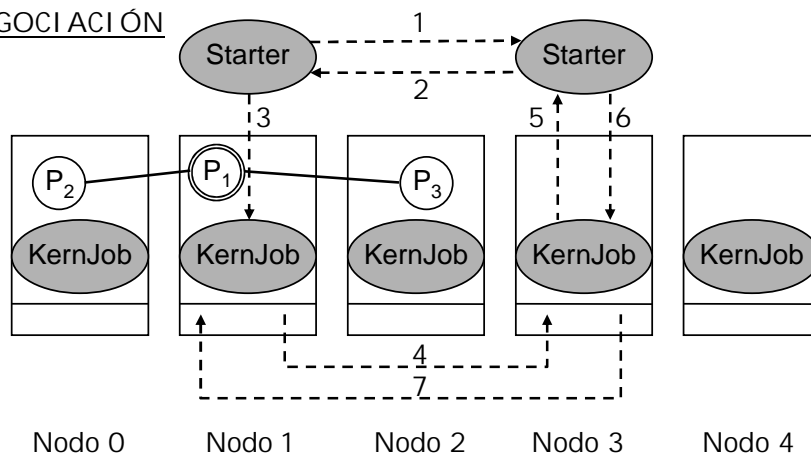
HandShake
MigrateIn
MigrateOut
CancelMigrate

Ejecutar la Migración

- ① Negociación
- ② Transferencia
- ③ Limpieza

Recolectar Estadísticas

NEGOCIACIÓN



Migración de Procesos en el Sistema Charlotte

Este sistema desarrollado en la Universidad de Wisconsin alrededor de 1983 estaba compuesto de 20 ordenadores VAX-11/780 bajo Unix BSD 4.2, unidos por una red token-ring a 80 Mbps.

En cuanto a la migración de procesos, tiene 3 características muy importantes:

1. Separa las decisiones de los mecanismos. Así, aunque los mecanismos de la migración están soportados por el núcleo del sistema operativo, la planificación se realiza mediante procesos fuera del kernel.
2. La migración es transparente tanto al proceso en cuestión, como a los procesos con los que mantiene comunicación.
3. La migración puede abortarse en cualquier momento. Es más, el proceso a migrar puede rescatarse, bajo ciertas circunstancias, si se produce la caída del equipo origen o destino durante la migración.

Para llevar a cabo la migración, el kernel cuenta con 3 *threads* que se reparten estas tareas:

1. Atender las nuevas llamadas al sistema: **HandShake** (arranca o para la recolección de datos estadísticos), **MigrateOut** (arranca la migración en la máquina origen), **MigrateIn** (deniega o permite y arranca la aceptación de un proceso migrante en la máquina destino), **CancelMigrate** (cancela, si es posible, una primitiva MigrateIn o MigrateOut).
2. Ejecutar las fases de la migración.
3. Recolectar estadísticas. Tal como se comentó en el ejemplo de planificación en la Transparencia 18.

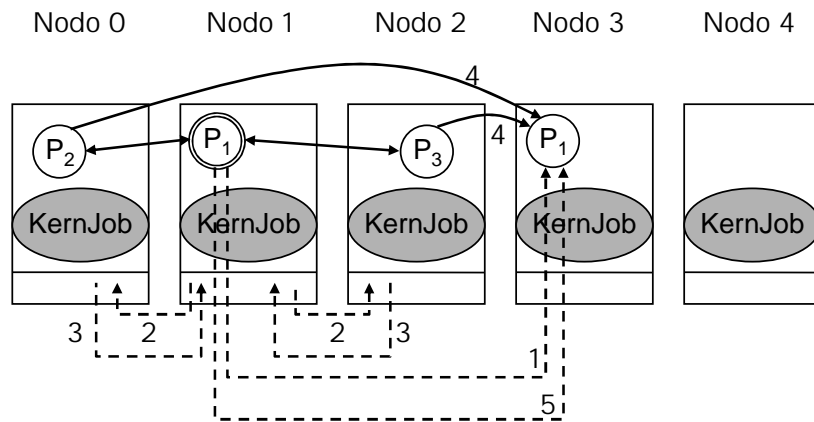
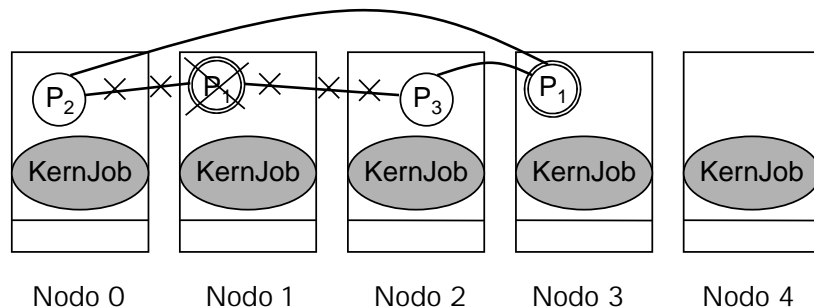
Una vez que se ha seleccionado un proceso para migrar y un nodo de destino, el procedimiento se realiza en tres fases: Negociación, Transferencia y Limpieza.

Negociación. El **Starter** (controla la política de migración) del ordenador origen envía una oferta de migración al nodo destino (mensaje 1). Si el **Starter** del nodo destino accede, contesta aceptando la migración (mensaje 2). El **Starter** origen le pasa la orden de migrar al kernel (mensaje 3 o **MigrateOut**). El *thread* encargado de llevar a cabo la migración envía al nodo destino un mensaje (mensaje 4) especificando el tamaño de la memoria necesaria para albergar el proceso, su tabla de enlaces de comunicaciones e información sobre el estado de la CPU del proceso y de las comunicaciones. Si el kernel destino dispone de los recursos necesarios para el nuevo proceso, envía una copia del mensaje 4 (mensaje 5) al **Starter** destino. El **Starter** destino ejecuta una **MigrateIn** (mensaje 6) para reservar todos los recursos necesarios. El kernel crea una tarea para aceptar la imagen del proceso que llegará por la red. Por último, este kernel destino responde con un mensaje de aceptación (mensaje 7), lo cual origina el comienzo de la transferencia del proceso (este es el "Punto de Compromiso").

Hasta el Punto de Compromiso, podría haberse abortado la migración en cualquier momento (incluso el 7 podría ser un mensaje de rechazo), y el nodo origen tendría que haber vuelto a negociar la migración con otro equipo.

TRANSFERENCIA

- Transferencia de la Imagen
- Actualización de los enlaces de Comunicaciones
- Transferencia del Estado del Proceso

LIMPIEZA

Transferencia. En esta fase, en la que el proceso está congelado, hay que enviar al nodo destino el estado del proceso y su imagen en memoria. Consta de tres pasos:

a) Transferencia de Imagen. El kernel origen transfiere la imagen del proceso (1), la cual será almacenada por el nodo destino en el espacio reservado en la fase de negociación.

b) Actualización de los Enlaces de Comunicaciones. Al kernel de cada proceso con el que mantiene comunicación el proceso migrante se le comunica la nueva dirección del proceso (2). Cada mensaje de actualización de dirección debe contestarse con un ACK (3). Una vez recibidas todas las confirmaciones puede iniciarse el siguiente paso (transferencia del estado del proceso) puesto que ya no se recibirán más mensajes en su equipo origen.

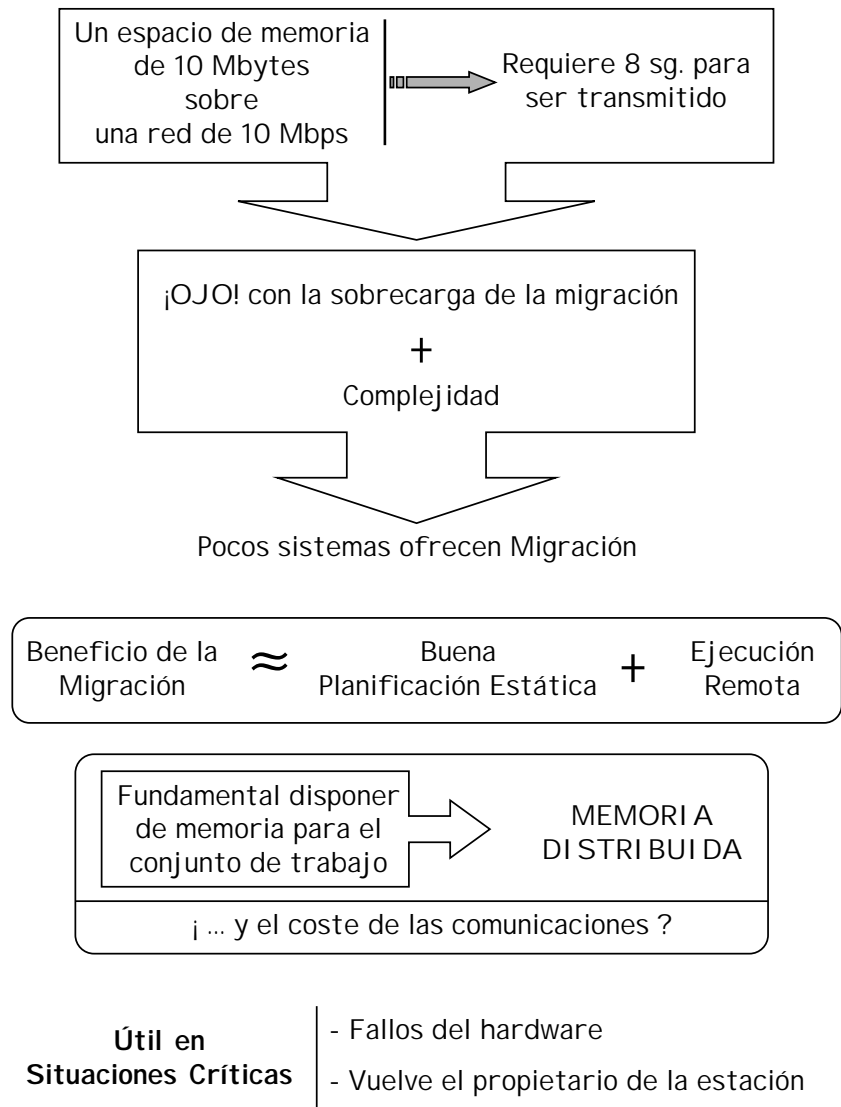
Hasta este momento, todos los mensajes recibidos para el proceso migrante en el nodo origen se han guardado en un buffer, y se transferirán posteriormente con el estado del proceso (en el paso c).

A partir de este momento el nodo destino ya puede recibir mensajes para el proceso migrante (4), incluso antes de haberse transferido el estado del proceso. Esto implica que los mensajes recibidos en el nodo destino a partir de este momento deben guardarse en un buffer hasta que el proceso migrante reanude su ejecución y esté dispuesto para que le entreguen mensajes.

Obsérvese que durante esta fase hay momentos de inconsistencia en los enlaces de comunicaciones, de tal manera que algunos mensajes pueden enviarse al nodo origen y otros al nodo destino.

c) Transferencia del Estado del Proceso. Se transfiere toda la información relativa al proceso (5): su descriptor, enlaces de comunicaciones, mensajes recibidos, etc. Esto implica la serialización de estructuras y punteros.

Limpieza. Después de enviar toda la información, el núcleo origen destruye todas las estructuras de datos relativas al proceso migrado. El núcleo destino, por su parte, una vez recibida toda la información añade el proceso a la lista de procesos preparados o en espera (dependiendo de su estado en el momento de comenzar la migración) y entrega todos los mensajes guardados en los buffers, empezando por los que se habían recibido en el equipo origen. La migración ha finalizado.



Lo que sigue a continuación es una recapitulación de distintas opiniones obtenidas de distintos autores de textos sobre sistemas distribuidos. Aunque en algunos casos estas opiniones pueden estar encontradas o no estar completamente de acuerdo, me ha parecido oportuno incluirlas aquí para ayudar al lector a que elabore sus propias conclusiones.

La necesidad de la migración de procesos se detectó a finales de los años 70 y ha recibido mucha atención desde entonces, pero hasta ahora solo unas pocas implementaciones han tenido éxito, pues se ha comprobado que su complejidad y la sobrecarga que conlleva es bastante mayor de lo que se esperaba.

Antes de realizar una migración (que no sea forzosa), debe evaluarse cuidadosamente su coste. Como ya dijimos anteriormente, el coste de migrar un proceso que ocupa un espacio de memoria de 10 Mbytes a través de una red de 10 Mbps, requiere 8 segundos. Debe tenerse cuidado para que el coste que implica la migración no sea mayor que el beneficio que se obtiene de ella.

Excepto en casos forzados, la migración de un proceso solamente está justificado si el tiempo restante de ejecución (en su equipo actual) es un orden de magnitud mayor que el coste de la propia migración.

La migración de procesos es una operación muy costosa, tan costosa que no resulta fácil encontrar sistemas distribuidos que la utilicen para realizar un reparto de carga. Como alternativa a la migración de procesos, puede conseguirse un buen reparto de carga sin más que realizar una buena asignación inicial del procesador, es decir, simplemente con una buena política de planificación estática y la disposición de mecanismos para la ejecución remota de procesos.

La experiencia práctica también muestra que la cantidad de memoria disponible para un proceso suele ser un factor decisivo para el rendimiento general del sistema. Si las páginas de memoria que componen los conjuntos de trabajo de los procesos arrancados en un equipo ocupan más que la memoria central disponible, el rendimiento puede degradarse radicalmente debido al trasiego de la paginación. El reparto de memoria disponible por todo el sistema suele ser una mejor estrategia que la del reparto dinámico de la carga.

La migración de procesos puede resultar útil en situaciones en las que los procesos tienen que abandonar forzosamente la máquina en la que se están ejecutando, bien porque ésta falla (fallo detectado por el hardware del equipo) o porque se debe expulsar al proceso debido a que el propietario de la estación de trabajo, que antes estaba libre, vuelve a utilizarla otra vez.