Índice

- Sistemas operativos y sistemas distribuidos
- Procesos e hilos en sistemas distribuidos
- Modelos de sistema
- Algoritmos de asignación de procesadores
- Tolerancia a fallos

Sistemas operativos y sistemas distribuidos

- **Sistemas operativos en red**: tienen **acceso a la red** y mediante esta a **recursos remotos**. Este acceso es **totalmente transparente** .
  - Tienen también **procesos que NO pueden ejecutarse en otros nodos**
  - El usuario debe elegir que procesos se **ejecutan en su nodo** y qué procesos en **remoto**
- **Sistemas operativos distribuidos**: existe **una ÚNICA imagen del sistema**. Se controlan todos los nodos
  - El usuario **NO debe preocuparse del nodo** en el que se ejecutará el proceso
  - **Se pierde la autonomía** en cada uno de los nodos
- Nos centraremos en sistemas operativos de red, ya que son los más extendidos y utilizados
- Aunque en cada nodo de los **sistemas operativos** en red existe un sistema operativo local (imagen local), todos los nodos **comparten el middleware**

Sistemas operativos y sistemas distribuidos

- Los sistemas operativos en red deben proporcionar como mínimo:
  - **Encapsulamiento**: interfaz sencilla para acceder a recursos. **Ocultando detalles** a los clientes
  - **Protección**: **gestión de los accesos** permitidos a recursos
  - **Procesamiento concurrente**: acceso a recursos concurrentemente desde el punto de vista el cliente. La gestión debe ser **transparente**
- Para cumplir con estas mínimas funcionalidades el sistema operativo requiere:
  - **Gestor de procesos**: **crea y gestiona** las operaciones de los **procesos**. Consideramos proceso la unidad de gestión de recursos que **incluye uno o más hilos**
    - **Gestor de hilos**: **creación**, **sincronización** y **planificación** de hilos
  - **Gestor de comunicaciones**: **comunicaciones entre hilos** de diferentes procesos en el mismo nodo
  - **Gestor de memoria**: compartición de **memoria virtual** y **física**
  - **Supervisor**: atiende **interrupciones** y otras **excepciones**

Ejemplos de Sistemas Operativos en red

Buscar en internet

Procesos e hilos en sistemas distribuidos

- Un **proceso** consiste en un **entorno de ejecución** formado por **varios hilos**

- El **entorno de ejecución** se encarga de gestionar los **diferentes recursos** a los que acceden los diferentes hilos. Los recursos son:
  - **Espacio de direcciones**
  - **Recursos e interfaces de comunicación** (sockets)
  - **Sincronización de hilos** (semáforos)
  - **Recursos de alto nivel**: archivos, interfaces gráficas, etc.

- Un **hilo** es la **abstracción** del sistema operativo para una **actividad**
  - Los hilos pueden **crearse y destruirse** de forma **dinámica**
  - Los hilos son **protegidos por el entorno de ejecución**

Ejercicio de razonar

¿Es posible depurar un programa multihilo?

Procesos e hilos en sistemas distribuidos

- La creación de procesos consta de dos pasos:
  - **Elección del procesador destino**
    - Dependiente de las políticas de **localización**
      - **Estáticas**: basadas en **análisis matemáticos** para la **optimización**
      - **Adaptativas**: reglas basadas en **factores no predecibles**, como la **carga** media del nodo (gestor de carga centralizado o jerárquico)
    - **Migración de procesos**: transferencia de un proceso de ejecución de un nodo a otro
  - **Creación del entorno de ejecución y del hilo asociado a él**
    - Se define el **tamaño de memoria** en el espacio de direcciones
    - Se definen **regiones** para permitir **crecimiento** y **evitar solape**
    - Se definen los **permisos** de cada región (lectura/escritura)

Procesos e hilos en sistemas distribuidos

- **Hilos**: En el servidor
  - **Servidor**: dispone de un conjunto de **uno o más hilos** que de forma repetitiva va atendiendo y ***procesando las solicitudes*** **recibidas**. Si tiene que **compartir memoria entre los diferentes hilos**, el aumento del **número de hilos NO será directamente proporcional al aumento de la eficiencia en el procesador**

  - Se requiere de un proceso **multi-hilo** que se adapta automáticamente a un **multiprocesador** de memoria compartida. Pueden surgir diferentes tipos de arquitecturas:
    - Arquitectura de **asociación de trabajadores**:
      - Se crea **un conjunto fijo de hilos trabajadores**
      - El **hilo receptor** y gestor de cola recibe las peticiones y gestiona la cola de **asignación de nodos trabajadores**

Procesos e hilos en sistemas distribuidos

- **Hilos**: En el servidor
  - Se requiere de un proceso multi-hilo que se adapta automáticamente a un multiprocesador de memoria compartida. **Tipos de arquitecturas**:
    - Arquitectura de **hilo por solicitud**: el hilo coordinador **genera un nuevo hilo** por **cada solicitud**. El **hilo se elimina a sí mismo** tras finalizar su labor. En esta arquitectura **NO EXISTE COLA** y se puede maximizar el rendimiento. Presenta **sobrecarga** al dedicar tanto procesado a la **generación** y **destrucción** de hilos
    - Arquitectura de **hilo por conexión**: el servido **crea un hilo** cada vez que un **nuevo cliente se conecta**. Si se **cierra la conexión desaparece el hilo**. Esta arquitectura requiere de una cola por conexión, pero reduce la sobre-carga del procesador a coste de mayores retrasos
    - Arquitectura de **hilo por objeto**: se genera un **hilo** por **cada objeto remoto**

Procesos e hilos en sistemas distribuidos

- **Hilos**: En el cliente
  - Se recomiendan **hilos separados** para procesos que pueden **bloquear el cliente**
  - El hilo que **no tiene procesos bloqueantes** **se encarga** de **gestionar las colas** de entrada y salida (únicamente **parará si se saturan los búferes**)

¿Por qué crear multi-hilos y no multi-procesos?

Porque los hilos son más **baratos** en:

- **Creación**
- **Conmutación** dentro de un mismo proceso (sólo es **escritura en un registro**)
- **Compartición de recursos**

Desventajas:

- Los **hilos NO están protegidos** dentro de un **mismo proceso**: **comparte variables**
- Un **hilo erróneo** puede **modificar** los **datos de otro hilo**

Procesos e hilos en sistemas distribuidos

¿Por qué crear multi-hilos y no multi-procesos?
Los procesos son del orden de **10 veces más lentos** que los hilos

Proceso	Hilo

Procesos e hilos en sistemas distribuidos

¿Por qué crear multi-hilos y no multi-procesos?
Los procesos son del orden de **10 veces más lentos** que los hilos

Proceso	Hilo
Espacio de direcciones	Registros en el procesador
Interfaces de comunicación	Estados bloqueado/preparado
Semáforos y sincronización	Interrupciones software
Lista de identificadores de hilos	-

Procesos e hilos en sistemas distribuidos

- Consideraciones en la programación basada en hilos:
  - **Sincronismo**: para **evitar condiciones de carrera**. Es posible **bloquear** (wait) o **despertar** hilos (notify o notifyAll)
  - **Planificación** de hilos: existen dos tipos
    - **Apropiativa**: un hilo **puede suspenderse en cualquier punto** para dejar paso a otro hilo
    - **No apropiativa**: un hilo se ejecuta hasta que **él mismo realiza una invocación** al sistema de gestión de hilos **para poder desalojarse** y dejar paso a otro hilo.
      Ventaja: convierte en **sección crítica todo el hilo**.
      Desventaja: **no** se puede utilizar en aplicaciones de **tiempo real**.
  - **Implementación**: existen **sistemas operativos** que sí que **dan soporte a multi-hilo**, pero otros no, con lo que el usuario **emula el multi-hilo** (aumenta complejidad, disminuye eficiencia)

Modelos de sistema

- Modelo de **estación de trabajo**: computadoras dispersas que pueden ser **ACTIVAS O NO**
  - Modelo **sencillo**
  - **Cantidad fija** de poder de cómputo
  - Cada **nodo** puede incluir un **usuario autónomo** o procesarse de **manera distribuida**
  - Se pueden **aumentar los recursos de la estación de trabajo**
  - **No es eficiente** ya que algunos usuarios **reciben recursos que no necesitan**
- Si se monitoriza su evolución se puede **implementar una migración** de la computación, trasladando parte de los recursos, los procesos o los hilos de manera transparente. Existen las siguientes opciones:
  - **Asignación global**: cualquier nodo está disponible
  - **Particionado por rol de usuario**: en función del usuario el nodo está disponible
  - **Particionado por carga**: existen diferentes tipos de nodo en función del número de hilos que está ejecutando
  - **Particionado por tarea**: nodos disponibles dependiendo de la tarea a ejecutar

Modelos de sistema

- Modelo de la **pila de procesadores**: asignan de manera dinámica los nodos a los usuarios
  - Modelo **CENTRALIZADO**, con un **controlador que gestiona colas**
  - La visión del usuario no es la de múltiples nodos, sino la de un **único recurso** con la misma potencia de computación que múltiples nodos
  - **Se adapta a las necesidades existentes** en un momento determinado
  - **No** permite que se utilicen los **nodos de manera separada** o autónoma
  - Soporta **migración de procesos o hilos**

Algoritmos de asignación de procesadores

- Clasificación de **algoritmos de asignación** de procesadores:
  - **Migratorios**: se puede modificar la localización del hilo o proceso
    - **Aleatorios**: no existe criterio establecido para modificar la localización
    - **Optimizadores**: se busca mejorar un determinado parámetro
      - ***Latencia*** : tiempo de respuesta
      - ***Tasa*** : velocidad de respuesta
  - **No migratorios**: permanecen en el nodo en el que se crea el hilo o proceso

- **Requisitos mínimos** de los algoritmos: conocer estos detalles consume recursos extra
- Conocen la **carga** del nodo: no es trivial, por ejemplo número de procesos e hilos
- Conocen el **estado de los procesos e hilos** (activo, dormido, etc.)
- Conocen el **estado de la CPU global**

Algoritmos de asignación de procesadores

- Aspectos de **diseño de los procesadores**:
  - **Deterministas vs heurísticos**
    - **Deterministas**: **Conocimiento total del sistema distribuido** incluyendo hilos y procesos
    - **Heurísticos**: Sistemas **totalmente impredecibles** y no modelables matemática-mente
  - **Centralizados vs distribuidos**
  - **Óptimos vs subóptimos**
    - **Óptimos**: Requieren un mayor conocimiento del sistema y por tanto almacenar y **procesar más datos durante más tiempo**
    - **Subóptimos**: Establecen una **balanza** entre el **tiempo de procesado** y el grado de **optimización del sistema**
  - **Locales vs globales**: referente al **origen de la información** que utiliza para tomar decisiones

Algoritmos de asignación de procesadores

- Ejemplos de algoritmos de asignación de procesadores:
  - **Determinista** según la teoría de gráficas:
    - Pretende **minimizar el tráfico de red entre equipos** y puede representar mediante **arcos** el consumo de cada proceso o hilo del sistema
    - Requiere el **conocimiento de todo el sistema** (por eso es **POCO APLICABLE**)

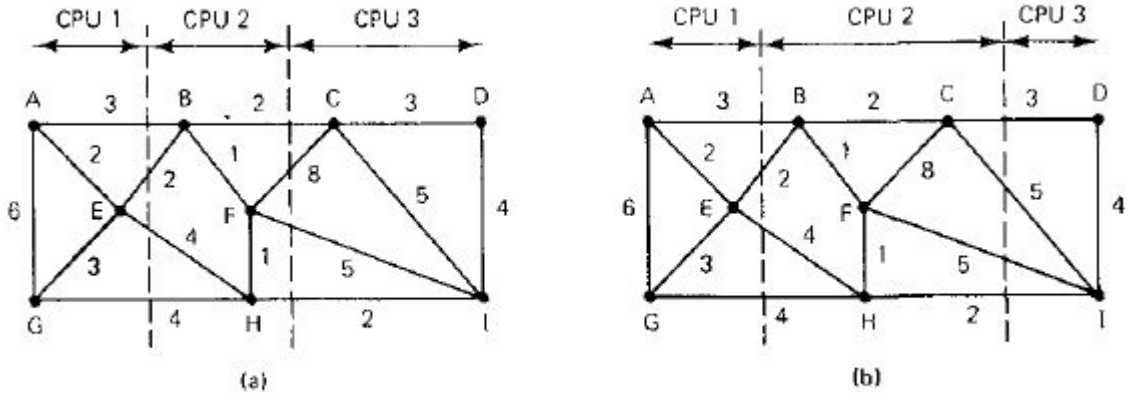

Figura 4-17. Dos formas de asignar 9 procesos a 3 procesadores.

Algoritmos de asignación de procesadores

- Ejemplos de algoritmos de asignación de procesadores:
  - **Centralizado**:
    - Con un **sistema de penalizaciones y bonificaciones** distingue gestionado por tablas distingue los procesos **pesados** de los **ligeros**
      - Da **prioridad a los ligeros**
    - No se adapta bien a los sistemas de gran tamaño ya que el **nodo central** que asigna las bonificaciones se convierte rápidamente en el **cuello de botella**
  - **Remates**: Sigue un modelo económico
    - Cada **procesador anuncia su precio aproximado** en un archivo que todos pueden leer. Se indican los **servicios prestados** y el **tiempo de respuesta**
    - Cuando se inicia un proceso se verifica si alguien ofrece un el servicio de manera más económica : **relación velocidad precio**

Algoritmos de asignación de procesadores

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.

Ejercicio.

Imaginemos que estamos intentando pensar en un algoritmo de asignación de procesadores en base al nivel de carga de los distintos procesadores.

Si tuvieras que definir en una fórmula el índice de carga, ¿a qué lo igualarías?

Intenta iterar sobre la fórmula para ver si eres capaz de mejorarla

Algoritmos de asignación de procesadores

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.

Algoritmos de asignación de procesadores

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”.

Algoritmos de asignación de procesadores

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.

Algoritmos de asignación de procesadores

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).

Tolerancia a fallos

Recordamos. Sistema tolerante a fallos

Sistema que posee la capacidad interna para asegurar la ejecución correcta y continuada de un sistema a pesar de la presencia de fallos HW o SW

Tolerancia a fallos

Explosión del Ariane 5 en 1996

- Enviado por la ESA en junio de 1996 (fue su primer viaje)
- Coste del desarrollo: 10 años y 7.000 millones de dólares.
- Explotó 40 seg. después del despegue a 3.700 metros de altura.
- El fallo se debió a la pérdida total de la información de altitud.

- Causa: error del diseño software.

- El SW del sistema de referencia inercial realizó la conversión de un valor real en coma flotante de 64 bits a un valor entero de 16 bits. El número a almacenar era mayor de 32.767 (el mayor entero con signo de 16 bits) y se produjo un fallo de conversión y una excepción.

Tolerancia a fallos

Fallo de los misiles Patriot

- Misiles utilizados en la guerra del golfo en 1991 para interceptar los misiles iraquíes Scud

- Fallo en la interceptación debido a errores en el cálculo del tiempo.
- El reloj interno del sistema proporciona décimas de segundo que se expresan como un entero
- Este entero se convierte a un real de 24 bits con la pérdida de precisión correspondiente.
- Esta pérdida de precisión es la que provoca un fallo en la interceptación

Tolerancia a fallos

Fallo en la sonda Viking enviada a Venus

En lugar de escribir en Fortran:

DO 20 I = 1,100

que es un bucle de 100 iteraciones sobre la etiqueta 20, se escribió:

DO 20 I = 1.100

y como los blancos no se tienen en cuenta el compilador
lo interpretó como:

DO20I = 1.100

es decir, la declaración de una variable (O20I) con valor 1.100.
D indica un identificador real

Fuente: http://www.cs.tau.ac.il/~nachumd/verify/horror.html