Sistemas Distribuidos. Temario.

1. Introducción
2. Comunicación
3. Procesos
4. Nombrado
5. Sincronización
6. Consistencia y replicación
7. Tolerancia a fallos

Tema 3.- Procesos

1. Tareas
2. Procesos clientes
3. Procesos servidores
4. Migración de código
5. Agentes

Bibliografía: Capítulo 3 de Tanenbaum

1.- Tareas

1. Introducción
2. Clientes multitarea
3. Servidores multitarea
4. Programación con tareas

1.1.- Introducción

Conceptos generales:

- **Proceso:** programa en ejecución (espacio de memoria, recursos de alto nivel, contexto hardware)
  - **Contexto de un proceso:** Contexto del procesador (valores de los registros del la CPU, CP, SP) registros de la MMU.
  - **Cambio de contexto:** caro. Puede incluir intercambio entre memoria y disco.
  - En general sencillo de utilizar: el SO proporciona transparencia de concurrencia.
  - Mecanismos de comunicación entre procesos caros.
  - Tiempo de creación elevado.
- **Tarea**: contexto de la CPU y pila dentro de un proceso.
  - **Contexto de una tarea**: no incluye la memoria, compartida dentro de un proceso
  - **Cambio de contexto**: barato.
  - En general más complejo de utilizar: el SO no proporciona transparencia de concurrencia.
  - Mecanismos de comunicación entre tareas baratos
  - Tiempo de creación de tareas (~10 veces menor que procesos)

1.1.- Introducción

Implementación de tareas:

- En general se presentan como funciones de biblioteca.
- Operaciones para crear y destruir, así como operaciones sobre variables de sincronización (cerrojos, semáforos, etc)
- ¿Dónde se implementan?
  - En el sistema operativo: Linux, WinNT, Solaris, etc.
  - En espacio de usuario: pthreads
    - (-) Llamadas al sistema bloquean todo el proceso
    - (-) Planificador no expulsivo
    - (-) No se beneficia de multiprocesadores
    - (+) baratos (sin llamadas al sistema)
    - (+) planificador modificable
  - Dentro de un lenguaje de programación: Java, Ada.
    - Pueden estar dentro del sistema o fuera
  - Aproximaciones híbridas: cada tarea del sistema operativo puede contener varias tareas en espacio de usuario (fibras en WinNT, Light Weight Processes (LWP) de Solaris).

1.- Tareas

1. Introducción
2. Clientes multitarea
3. Servidores multitarea
4. Programación con tareas

1.2.- Clientes multitarea

- El objetivo es reducir el retardo de las transmisiones de red creando una tarea por cada “conversación”.
- Los procesos clientes inician la comunicación y proceden inmediatamente con otro trabajo.
- Ejemplos
  - Navegador Web: se descarga una página que contiene múltiples documentos empotrados. Una tarea para recibir cada documento y una tarea para mostrar la página al usuario.
  - RPC: un cliente puede iniciar diferentes invocaciones RPC desde diferentes tareas, logrando que no se quede todo el proceso bloqueado.

1.- Tareas

1. Introducción
2. Clientes multitarea
3. Servidores multitarea
4. Programación con tareas

1.3.- Servidores multitarea

- El objetivo es servir el máximo de peticiones simultáneas. Se crea una tarea por cada petición o por cada cliente.
- Es frecuente la organización siguiendo el patrón de diseño de **repartidor / trabajador**.
  - El repartidor espera peticiones y ante cada petición, crea un trabajador (o lo elige de un conjunto de trabajadores en caliente).
  - Cada trabajador conversa con un cliente.

1.- Tareas

1. Introducción
2. Clientes multitarea
3. Servidores multitarea
4. Programación con tareas

1.4.- Programación con tareas

Conceptos generales

- **Exclusión mutua**: Evitar que una tarea lea o modifique una variable que está siendo modificada por otra y evitar que una tarea modifique una variable que está siendo leída por otra.
- **Sección crítica**: el código que accede al recurso compartido y que debe ejecutarse en exclusión mutua.
- Solución: serializar los accesos a cada recurso (variable) mediante un "protocolo". Este protocolo exigirá el uso de algún mecanismo de sincronización: cerrojos, semáforos, etc.
- Problemas:
  - **Interbloqueos**
  - **Inanición**
- **Condición de carrera**: ocurre cuando más de una tarea acceden al mismo recurso y al menos una de ellas no emplea en protocolo de acceso a la sección crítica.
- **Código reentrante**: código que puede ser ejecutado concurrentemente por varias tareas sin que exista riesgo de que aparezcan condiciones de carrera.

1.4.- Programación con tareas: Java

- Para crear una tarea se crea una nueva instancia de la clase `Thread`.
- Como argumento se le debe proporcionar un objeto que cumpla la interfaz `Runnable`.
- Las tareas se crean en estado suspendido, y para que empiecen a funcionar habrá que invocar la operación `start`() de la tarea, que a su vez invocará al método `Run`() del objeto que cumple con `Runnable`
- La forma de sincronizar las tareas se realiza mediante monitores.
  - Todo objeto java es un monitor (una variación) que sincroniza el acceso a sus operaciones si éstas se declaran con la palabra reservada `synchronized`.
  - Si dos tareas intentan acceder simultáneamente a operaciones `synchronized` (la misma o distintas operaciones) de **mismo** objeto, sólo la que llegue antes podrá ejecutarse. La otra esperará a que la tarea que entró abandone el monitor.
  - Una tarea abandona el monitor cuando finaliza el código de la operación `synchronized` o si se suspende invocando a `wait`()
  - Invocando `notify() o notifyall()` una tarea despuerta a una o a todas las tareas que estuvieran bloqueadas en un `wait`() sobre ese mismo objeto.
  - En todo caso, se garantiza que sólo una tarea se ejecutará en el monitor.
- De forma adicional, una tarea puede esperar a que termine otra, invocando a `Thread.join`()

Tema 3.- Procesos

1. Tareas
2. Procesos clientes
3. Procesos servidores
4. Migración de código
5. Agentes

Bibliografía: Capítulo 3 de Tanenbaum

2.- Procesos clientes

1. Interfaces gráficas
2. Transparencia en el cliente

2.1.- Interfaces gráficas

- La misión principal de los procesos cliente suele ser representar gráficamente la información que proporcionan los servidores.
- Hay multitud de sistemas gráficos y textuales para crear interfaces cliente: teléfonos móviles, PDA's, sistemas de ventanas.
- Tiene interés especial el sistema X-Window, que permite crear interfaces gráficas cliente/servidor:
  - El servidor es la máquina que tiene el entorno de ventanas.
  - Se pueden lanzar aplicaciones cliente que utilicen como servidor un entorno de ventanas ubicado en otra máquina.
  - Componentes principales de X-Window:
    - X-Kernel: manejo a bajo nivel del dispositivo por medio de drivers.
    - XLib: biblioteca para acceder a los servicios
    - Window managers: aplicaciones sobre Xlib que manejan todo el terminal y establecen restricciones.
    - Aplicaciones gráficas: aplicaciones sobre Xlib sometidas a las restricciones del window manager (si existe uno)
    - Nótese que tanto los window managers como las aplicaciones pueden ejecutarse en máquinas diferentes a la máquina donde se ejecuta el X-Kernel (el servidor de X Window)

2.2.- Transparencia en el cliente

- Para lograr transparencia en sistemas distribuidos, los procesos cliente deben colaborar ocultando detalles al programador.
- Habitualmente mediante “proxies” para el acceso a servicios.
- Los **proxies** pueden ocultar:
  - La ubicación
  - El grado de replicación
  - Los fallos

Tema 3.- Procesos

1. Tareas
2. Procesos clientes
3. Procesos servidores
4. Migración de código
5. Agentes

Bibliografía: Capítulo 3 de Tanenbaum

3.- Procesos servidores

- Se implementan para proporcionar servicios a un conjunto de clientes.
- Hay varios aspectos de diseño que deben considerarse al implementar servidores: Concurrencia, localización de los servicios, control sobre los servidores, estado de los servidores.
- Tipos de servidores respecto a su **concurrencia**:
  - Servidores **iterativos**: un solo hilo espera peticiones y las procesa.
  - Servidores **concurrentes**: un hilo (o proceso) espera peticiones y ante cada petición, le pasa la petición a otro hilo (o proceso).
- Tipos de servidores respecto a su **localización** (punto de entrada)
  - **Punto de entrada fijo y conocido**: los clientes conocen de antemano la ubicación (p.ej: ftp, http, etc)
  - **Punto de entrada desconocido**: resoluble mediante un servicio de nombres (debe conocerse la ubicación del servicio de nombres)

3.- Procesos servidores

- Tipos de servidores respecto al **control** que se puede ejercer sobre ellos:
  - **Sin control**: los clientes que quieren desconectar, simplemente abortan la conexión, por ejemplo terminando la propia aplicación cliente
  - **Con control**: se establece un canal de comunicación adicional entre cliente y servidor, de forma que el cliente puede actuar sobre el servidor: para detenerlo, pararlo, reanudarlo, etc.
- Tipos de servidores respecto a su **estado interno**:
  - **Sin estado**: los servidores no guardan información respecto al estado de los clientes ni respecto a su conversación (p.ej: un servidor http)
  - **Con estado**: el servidor guarda información de cada cliente. Ante caídas y recuperaciones del servidor, se debe recuperar la información de cada cliente.

Tema 3.- Procesos

1. Tareas
2. Procesos clientes
3. Procesos servidores
4. Migración de código
5. Agentes

Bibliografía: Capítulo 3 de Tanenbaum

4.- Migración de código

A veces resulta conveniente enviar un programa de un lugar a otro incluso mientras el código se encuentra en ejecución: migración de código

1. Motivos para migrar código
2. Modelos de migración de código
3. El problema de los recursos

4.1.- Motivos para migrar código

- **Aumentar la eficiencia**:
  - **Repartir carga computacional**: si una máquina tiene mucha carga, se observará una mejora global del rendimiento del sistema distribuido, si se migra su código a otra máquina con menos carga.
  - **Disminuir carga de la red de comunicaciones**: Si se observa mucho tráfico entre dos máquinas debida a la interacción entre dos "procesos", puede ser conveniente migrar uno de ellos a la máquina del otro.
  - **Aumentar la velocidad de procesamiento**: distribuyendo el procesamiento entre máquinas para aumentar el grado de paralelismo.
- **Permitir la carga dinámica de código**:
  - **Uso de partes del código no conocidas a priori**: Por ejemplo podemos construir clientes que hagan uso de protocolos genéricos de comunicación con los servidores, pero que carguen del servidor el protocolo específico que se utiliza en cada momento.
  - **Mejorar la distribución/instalación del código en sistemas grandes**: muchas veces no es posible instalar una aplicación en todo un sistema, de forma que la mejor opción consiste en que los clientes descarguen el código que deben ejecutar conforme lo usen.

4.2.- Modelos de migración de código

- Consideramos que los programas a migrar constan de:
  - Segmento de código: contiene únicamente instrucciones
  - Segmento de recursos: contiene referencias a recursos externos, tales como archivos, impresoras, mecanismos de sincronización, etc.
  - Segmento de ejecución: contiene el estado de la ejecución del proceso: variables, pila y contador de programa.

4.2.- Modelos de migración de código

- Modelos de migración respecto a los segmentos que migran:
  - **Movilidad débil**: solo migra el segmento de código (p.ej: los Applets de Java)
  - **Movilidad fuerte**: migran el segmento de código y el segmento de ejecución.
- Modelos de migración respecto al iniciador:
  - **Migración iniciada por el emisor**: un programa decide enviar su propio código a máquinas remotas. P..ej.: Un buscador Web que envía subprogramas buscadores a otras máquinas.
  - **Migración iniciada por el receptor**: un programa decide pedir código para continuar con su ejecución. P. ej.: los Applets de Java

4.3.- El problema de los recursos

- Migrar el segmento de recursos muchas veces puede ser problemático. Por ejemplo migrar la referencia a un socket abierto TCP de una máquina a otra.
- La clave para entender las posibilidades de migración del segmento de recursos radica en el enlace que exista entre el **recurso y el proceso** y el **recurso y la máquina**
- **Tipos de enlace recurso-proceso**:
  - **Enlace por identificador**: un proceso accede a los recursos por un identificador. P. Ej.: URL, dirección TCP, etc. En este caso el recurso es único y debe mantenerse la asociación.
  - **Enlace por valor**: lo que necesita el proceso es el valor de un recurso, no su unicidad. P. Ej.: cierta biblioteca de código.
  - **Enlace por tipo**: El proceso necesita únicamente cierto tipo de recurso. P..ej.: terminal, disco, etc.

4.3.- El problema de los recursos

- Otro aspecto clave que afecta a la migración del segmento de recursos, es el enlace entre recurso y máquina (a veces a través de un nivel de indirección adicional en el proceso)
- **Tipos de enlace recurso-máquina**
  - **Recursos no enlazados**: son fáciles de migrar pues no existe esta asociación. P. ej.: ficheros de datos únicamente utilizados por los procesos que migran.
  - **Recursos no enlazados, pero interrelacionados** con otros. P. ej.: bases de datos.
  - **Recursos fijos**: recursos que sólo existen en una máquina y no se pueden mover. P. ej.: socket TCP

4.3.- El problema de los recursos

- Una vez clasificados los tipos de recursos atendiendo a su enlace con los procesos y máquinas donde residen, podemos detallar las acciones a realizar para migrar los segmentos de recursos:
  1. **Mover** el recurso a la máquina destino. (MV)
  2. Establecer una **referencia global** al recurso, dejando el recurso fijo, el proceso migrado lo continuará referenciando. (RG)
  3. **Copiar** el valor del recurso a la máquina destino. (CP)
  4. **Re-enlazar** con un recurso similar en la máquina destino. (RE)

4.3.- El problema de los recursos

- Las acciones a realizar sobre los recursos de un segmento de recursos dependiendo de su enlace son las siguientes:

	No enlazados	**Interrelacionados**	**Fijos**
Por identificador	MV (o RG)	RG (o MV)	RG
Por valor	CP (o MV,RG)	RG (o CP)	RG
Por tipo	RE (o MV,CP)	RE (o RG,CP)	RE (o RG)

Tema 3.- Procesos

1. Tareas
2. Procesos clientes
3. Procesos servidores
4. Migración de código
5. Agentes

Bibliografía: Capítulo 3 de Tanenbaum

5.- Agentes

- Se trata de un concepto en general poco definido y ambiguo, pero muy utilizado.

Definiremos Agente, como un proceso autónomo, capaz de reaccionar ante cambios de su entorno y de iniciar cambios en su entorno, posiblemente en colaboración con usuarios y otros agentes.

5.- Agentes

Tipos de agentes

- Agentes **colaborativos**: se construyen como procesos independientes enfocados a ofrecer cierta funcionalidad de forma común. P. ej.: agendas colaborativas
- Agentes **móviles**: agentes diseñados para migrar de una máquina a otra. P. ej.: wwww (world wide web worm)
- Agentes **de interfaz**: diseñados para adaptarse al uso que realice el usuario de cierta funcionalidad. P. ej.: interfaces que acercan compradores y vendedores en subastas.
- Agentes **de información**: dedicados a recopilar información de diferentes medios y operar sobre la información antes de proporcionarla. P. ej.: agentes de correo electrónico para filtrado.