T1-PSP.pdf

bloodyraintatii

Programación de servicios y procesos

1º Desarrollo de Aplicaciones Multiplataforma

Estudios España

Accede al documento original

Programación de Servicios y Procesos

Tema 1: Tipos de programación

• Conceptos básicos I: programa, proceso, servicio.
→ Programa → toda la info (código y datos) almacenada en disco de una aplicación y que resolverá un problema concreto.
→ Proceso → un programa en ejecución. Incluye el código y los datos pero también lo necesario para que se ejecute. Un proceso es una entidad independiente aunque se ejecute en un mismo programa.
→ Servicios → procesos no interactivos que se están ejecutando continuamente en segundo plano sin intermediación del usuario. Proporcionan un servicio básico para el resto de procesos. El SO tiene sus propios servicios que arrancan al iniciarse. Cada proceso puede tener uno o más servicios. También se les conoce como *demonios*.

• Conceptos básicos II: Hilos y ejecutable.
→ Hilos / threads / hebras → tarea que se puede ejecutar en paralelo a otras, lo que se traduce en mayor velocidad de ejecución de la tarea. Todos los hilos comparten recursos: espacio de memoria, archivos abiertos, puertos de comunicación en red, bbdd.
→ Todos los hilos de un proceso comparten la información por lo que si uno cambia algo, los demás ya tendrán la información cambiada.
→ El proceso termina cuando todos sus hilos terminan, y entonces se liberan los recursos.
→ Un hilo se va a ejecutar dentro del contexto de un proceso llamado proceso padre.

- Programas de flujo múltiple → los que se ejecutan mediante varios hilos.
- Programas de flujo único → los que solo tienen un hilo.

→ Ejecutable → fichero con la info necesaria para crear un proceso partiendo de los datos almacenados de un programa / fichero que permita poner el programa en ejecución como un proceso.

• Programación concurrente.
→ Programación concurrente → permite tener en ejecución simultáneamente varias tareas que pueden ser interactivas. Permite p.ej. escuchar música, ver la pantalla e imprimir a la vez. Proporciona mecanismos de comunicación y sincronización entre procesos.
→ Las tareas se pueden ejecutar en:

- Multiprogramación (ejecución en un único procesador) → parece que varios programas se ejecutan al mismo tiempo pero solo hay un proceso en ejecución en un determinado momento.
- Multitarea (varios núcleos en un mismo procesador) → varios núcleos en un procesador, p.ej. Dual Cores. Quad Cores… Cada uno puede ejecutar algo diferente al mismo tiempo.
- Programación paralela → permite que se ejecuten varias instrucciones a la vez paralelamente en diferentes núcleos, lo que mejora el rendimiento de un programa.

• Programación paralela

→ Programación paralela / multitarea multihilo / multihebra / multithreading → se diseñó para ejecutarse únicamente en un sistema multiprocesador. Permite la ejecución simultánea de una o varias tareas de un proceso.

→ Ventaja → mejora el rendimiento ya que las instrucciones de un mismo programa se pueden ejecutar en núcleos diferentes.

→ Desventaja → complejidad del diseño de los algoritmos, concretamente en la comunicación entre procesos o hilos. Se puede resolver con semáforos, interbloqueos, algoritmos de exclusión mutua, etc., pero sigue siendo complejo.

→ Arquitecturas que usan programación paralela:

- Sistemas multinúcleo → microprocesadores actuales.
- Microprocesadores específicos → procesadores gráficos, para videojuegos, embebidos, etc.

• Programación distribuida

→ Se da en sistemas distribuidos, que son ordenadores interconectados que comparten un mismo estado y recursos y funcionan como uno solo. P.ej. Internet.

→ Desarrolla software para sistemas distribuidos, abiertos, escalables, transparentes y tolerables a fallos. Usa la arquitectura cliente-servidor.

- Redes → varios microprocesadores se conectan a través de una red de conexión de alta velocidad y forman un clúster (varios pcs funcionan como 1).
- Supercomputadores → sistemas computacionales muy potentes usados para tareas que necesitan enorme capacidad de cálculo.
- Grid computing → se usan ordenadores muy potentes conectados en red entre sí.
- Cloud Computing / cómputo en la red → sistemas donde se pueden tener varios recursos (como el espacio en disco). Las máquinas que ofrecen este servicio están en otro lugar interconectadas.

Tema 2: Procesos

• Introducción a los procesos

→ Proceso → actividad caracterizada por la ejecución de una secuencia de instrucciones, que tiene un estado actual y un conjunto de recursos que toma del sistema.

→ Todos los SO son multitarea así que pueden ejecutar varios procesos a la vez o compartir el núcleo del procesador. Así se realizan las tareas más rápidamente.

→ Procesos según el modo de ejecución:

- Procesos por lotes → lo importante es el resultado final, no la ejecución. P.ej. pasar el antivirus al pc.
- Procesos interactivos → interacción del usuario y del proceso. P.ej. procesador de textos.
- Procesos en tiempo real → el tiempo de respuesta del sistema es crucial. P.ej. brazo mecánico en una operación o sistema de conducción automática.

→ Procesos según el origen de la ejecución:

- Procesos en modo Kernel → los ejecuta el núcleo del SO y el usuario no los puede controlar. P.ej. gestionan la memoria y el tráfico de red.
- Procesos en modo usuario → los lanza el usuario. Son los más usuales.

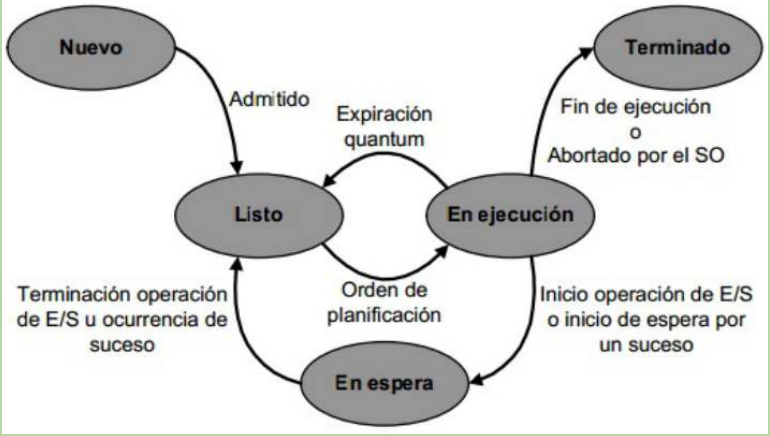
• Estado de un proceso

→ Planificador → encargado de crear y poner en ejecución los procesos del SO.

→ Ciclo de vida de un proceso → serie de estados por los que pasa un proceso a lo largo de su ejecución, teniendo que cumplir ciertos requisitos para pasar de un estado a otro.

→ Estados de los procesos:

- Nuevo → se crea el proceso a través de un fichero ejecutable aunque no haya sido admitido por el SO en el grupo de procesos ejecutables.
- Listo → proceso creado pero no justo para ejecutarse porque el SO no lo ha seleccionado para entrar a ejecución.
- En ejecución → el SO ha elegido al proceso para que se ejecute. Se ejecutará hasta que se cumpla el máximo tiempo de ejecución o hasta que el planificador lo saque de este estado, pasando de “nuevo” a “listo”.
- Bloqueado / en espera → el proceso ha pedido algún recurso mediante una interrupción y está esperando a que le sea concedido. Cuando pase, volverá al estado “listo”.
- Terminado → ejecución finalizada. Se liberan todos los recursos acaparados.



• Gestión de procesos

→ El SO es el encargado principal de toda la gestión de los procesos. Casi siempre sigue las órdenes del usuario. Cuando el procesador pasa a ejecutar un nuevo proceso, el SO guarda todo el contexto del proceso actual y restaura el contexto anterior. El SO también controla la comunicación y sincronización entre los procesos y su eliminación y terminación.

→ Se usa una ejecución concurrente de procesos en la que los procesos irán intercambiándose con los que están en ejecución para que todos usen el procesador de igual forma.

→ Colas en las que el SO organiza los procesos:

- Cola de procesos con todos los que hay en el sistema.
- Cola de procesos preparados con los que están "listo".
- Varias colas con procesos "bloqueado" en espera de alguna operación de entrada/salida. Habrá una cola por cada dispositivo de entrada/salida.

→ Eventos que provocan la creación de un proceso:

1. Arranque del sistema.
2. Ejecución desde un proceso de una llamada al sistema para la creación de otro proceso.
3. Petición de usuario para crear un proceso.
4. Inicio de un fichero por lotes.

• Planificación de procesos I: Tipos

→ Planificador de procesos / scheduler → gestiona las colas de procesos imponiendo un orden de ejecución entre los procesos de las colas.

→ Tipos de planificación:

- Planificación de procesos a corto plazo → seleccionan qué proceso de la cola de preparados va a pasar a ejecución. Son muy frecuentes y rápidos. Tienen algoritmos muy sencillos:
  - Planificación sin desalojo o cooperativa → solo se cambia el proceso en ejecución si se ha bloqueado o terminado.
  - Planificación apropiativa → los anteriores + cambia el proceso en ejecución si se puede ejecutar otro con mayor prioridad.
  - Tiempo compartido → quita el proceso en ejecución para que se ejecute otro cada cierto tiempo (quantum). Todas las prioridades son iguales.
- Planificación de procesos a largo plazo → seleccionan los procesos nuevos que deben pasar a la cola de preparados. Son muy lentos y poco frecuentes.

• Cambios de contexto

→ El contexto es todos los datos de la situación del proceso según se ejecute en un estado u otro. Este cambia cuando el proceso cambia de estado. El SO guarda el contexto y lo restaura cuando es necesario.

→ Por cada cambio de contexto, el SO guarda: estado inicial, estado del procesador, info de gestión de memoria, contador de programa (dónde va ejecutándose el proceso), y puntero de pila.

→ El cambio de contexto es "tiempo perdido" porque el procesador no hace un "trabajo útil" en ese tiempo.

Tema 4: Programación Paralela o multihilo

• Hilos de ejecución en un proceso

→ Programación paralela o multihilo → programación concurrente capaz de ejecutar varias tareas o hilos al mismo tiempo.

→ Hilo / hebra → trozo de código ejecutado dentro de un proceso, con la ventaja de que puede ser ejecutado en paralelo con otros hilos.

→ El proceso es el que crea y lanza sus hilos. Estos hilos solo pueden usar los recursos que tenga el proceso.

→ Puede suceder una inconsistencia en el programa si más de un hilo necesita acceder a los mismos recursos.

→ Los procesos activos seguirán en ejecución hasta que todos los hilos terminen su ejecución. Entonces, el SO destruirá el proceso y liberará los recursos.

→ Cuando se ejecuta un programa, se crea un proceso y un hilo primario.

→ Los hilos no pueden existir independientemente a un proceso no pueden ejecutarse por sí solos.

→ Podemos tener todos los hilos necesarios dentro de un proceso.

• Ventajas y desventajas de uso de hilos

→ Ventajas.

- Compartir recursos → los hilos dentro de un proceso comparten los recursos del proceso → operaciones más rápidas.
- Uso más eficiente y ahorro de memoria → crear nuevos hilos no supone usar más memoria.
- Capacidad de respuesta → el proceso podrá atender las peticiones del usuario antes.
- Paralelismo real → en sistemas con procesadores multinúcleo, cada hilo se podrá ejecutar en un núcleo diferente, por lo que el proceso usará el procesador de forma paralela, ejecutando varias instrucciones al mismo tiempo.

→ Desventajas.

- No todos los lenguajes de programación soportan la programación multihilo.
- El programador debe controlar todos los problemas relacionados con la comunicación y sincronización de los hilos. Estos problemas incluyen: inanición, bloqueo activo, acceso a los recursos críticos, zonas de exclusión mutua, condiciones de carrera o errores de inconsistencia en la memoria compartida.

msi

BLACK FRIDAY

Portátiles desde
499€

ENCUENTRA
EL TUYO

VER OFERTAS

• Recursos compartidos por los hilos

→ Los hilos tienen:

- Un identificador único.
- Un contador de programa para que pueda ejecutar su código independientemente.
- Un conjunto de registros asociados para que pueda hacer operaciones aritmético-lógicas independientemente.
- Una pila propia para ejecutar las llamadas a las funciones necesarias independientemente.

→ Los hilos comparten:

- El código a ejecutar.
- Las variables globales de la zona crítica.
- Los recursos del SO (ficheros, sockets, bbdd, etc.)

→ Para que los hilos no se bloqueen entre ellos se usan esquemas de bloqueo y sincronización entre varios hilos. No tienen una implementación sencilla y complican los programas pero resolverán el problema.

• Estados de un hilo

→ Los estados de un hilo son los mismos para los hilos creados por el usuario como para los creados por el sistema (hilos demonio o de sistema).

- Nuevo → hilo creado y listo para ejecutarse pero no ha sido elegido para ejecutarse.
- Listo o ejecutable → entrará en este estado cuando indiquemos.
- En ejecución → hilo listo para ejecutarse y el SO lo ha elegido para ejecutarse.
- Bloqueado → necesita recursos de entrada/salida del usuario o sistema. No se le asignará tiempo de CPU aunque estará listo para volver a ser usado.
- Finalizado → ejecución terminada y a la espera de que el SP lo destruya y libere sus recursos.

• Clases para hilos en Java

- Clase Thread → creamos hilos funcionales y les asignamos el código que queramos. Las clases hilo deberán heredar de esta. Métodos:
  - new() → crear un hilo.
  - start() → inicia la ejecución del hilo, de su método run().
  - run() → método del hilo que se ejecutará.
  - sleep() → hace que el hilo se bloquee durante un tiempo en milisegundos.
  - wait() → deja el hilo en espera.
  - getState() → devolverá el estado actual del hilo.
- Interfaz Runnable → la implementamos para añadir la funcionalidad de hilo a cualquier clase.
- Clase ThreadDeath → podemos manejar y notificar errores en el uso de las hebras. Hereda de la clase Error.
- Clase ThreadGroup → manejamos un grupo de hilos conjuntamente, haciendo que se ejecuten más eficientemente.

Tema 7: Introducción a la comunicación entre aplicaciones

• El modelo OSI

→ Conjunto de reglas creadas por ISO que dictan cómo deben funcionar las redes y sus comunicaciones, haciendo la interconexión más fácil.

→ El modelo OSI está formado por siete capas que dividen todo el funcionamiento y estados por los que deben pasar los datos para viajar de una red a otra.

→ Las capas son:

- Capa física → hardware, como topología de red y conexiones del ordenador.
- Capa de enlace de datos → direccionamiento físico de los datos que se envíen. Detecta errores, controla el flujo y ordena la llegada de los paquetes.
- Capa de red → enruta las redes y se encarga de que los datos lleguen a su destino.
- Capa de transporte → transporta los datos.
- Capa de sesión → mantiene la conexión entre dos equipos, reanudándola si se interrumpe.
- Capa de presentación → presenta la información. Trata semántica y sintaxis.
- Capa de aplicación → accede a los servicios de las demás capas. Se definen los protocolos que se usarán.

• El modelo TCP/IP

→ La comunicación entre capas/layers está controlada por protocolos que indican cómo tienen que ser, los datos que deben tener, cómo se deben enviar estos datos, cómo se deben recibir, etc.

→ Capas:

- Capa de aplicación → aplicaciones de red, que usan los niveles inferiores para transferir mensajes entre ellas. Protocolos que trabajan en esta capa:
  - HTTP → protocolo de comunicación entre servidores y navegadores web.
  - SMTP → dicta la manera de gestionar el correo electrónico.
  - DNS → traduce los nombres de dispositivos a direcciones IP.
  - FTP → posibilita las transferencias de ficheros.
  - NFS → habilitará la compartición de ficheros en diferentes pcs de la red.
  - TELNET → posibilita la conexión remota de terminales.
- Capa de transporte → softwares que crean el canal de comunicación, descomponen el mensaje enviado en diferentes paquetes y gestionan la transmisión entre el emisor y el receptor. Aquí actúan los protocolos TCP y UDP.
- Capa de Internet → softwares que dirigen los paquetes por la red y que se aseguran de que los paquetes lleguen al destino.
- Capa de red → hardwares de comunicaciones (tarjetas de red, cables, etc.) que transmiten los paquetes de información; ya que los protocolos deben conocer los detalles físicos de la red para enviar correctamente los paquetes.

• Protocolos de comunicaciones: protocolo TCP

→ Los modelos TCP/IP y OSI tienen la misión de proporcionar un transporte de información confiable entre el emisor y el receptor, independientemente de la capa física usada.

→ Protocolo TCP (Transmission Control Protocol) → protocolo orientado la conexión que garantiza que la información llegue sin errores. Parte el mensaje en paquetes y los envía por el canal de comunicación. Les asigna un número para poder reconstruirlos al llegar.

→ También controla el flujo del canal de comunicación. Controla si hay más o menos tráfico, evitando que se sature la red, o que un emisor rápido sature a un receptor lento.

→ Es fiable, ya que garantiza la llegada de los paquetes al receptor.

→ No es sencillo de implementar ya que debe cubrir muchos aspectos del transporte de información.

→ Ejemplos → HTTP, FTP, Telnet, etc.

• UDP

→ User Datagram Protocol / UDP → protocolo que no está orientado a la conexión, por lo que no tiene sincronización para el envío de mensajes entre emisor y receptor.

→ Las aplicaciones no necesitan asignación de control de secuencia ni de control de flujo.

→ Parte la info en datagramas y los envía sin control, pudiendo llegar o no al destinatario. No enumera los datagramas.

→ Se usa cuando es más importante la velocidad de entrega de paquetes → streaming o transmisiones de voz.

→ No es fiable ya que no garantiza la llegada de todos los datagramas ni controla el flujo en las comunicaciones.

→ Ejemplos → DHCP, BOOTP, DNS, etc.

• Los puertos

→ Puertos → direcciones de transporte constantemente a la escucha por si llegan paquetes o datagramas. Son el receptor de los paquetes o datagramas.

→ La mayoría se asignan aleatoriamente y sin orden, pero hay algunas aplicaciones que ya tienen un puerto asignado y no pueden ser asignados aleatoriamente o por nosotros.

→ Autoridad de Asignación de números de Internet / IANA → asigna los puestos predefinidos a las aplicaciones que lo necesiten. Tienen definidos los rangos de puertos:

- Puertos conocidos → reservados para aplicaciones estándar → del 0 al 1013. Ejemplos: FTP (21), HTTP(80).
- Puertos que están registrados → asignados a servicios o aplicaciones específicas → del 1024 al 49151. Será el rango que usaremos para nuestras aplicaciones.
- Puertos dinámicos → no están registrados para ningún servicio sino que atienden a conexiones temporales entre aplicaciones → del 49151 al 65535.

Tema 8: Modelos de comunicaciones

• Arquitectura cliente/servidor

→ Modelo de comunicación entre equipos que ofrece flexibilidad, interoperabilidad y estabilidad para acceder a recursos centralizados en un ordenador.

→ Ejemplos: WhatsApp, Telegram, navegadores web, etc.

→ Elementos:

- Cliente → interactúa con el usuario, procesa las peticiones para comprobar que sean válidas, realiza las peticiones al servidor y muestra los resultados al usuario en un formato correcto.
- Servidor → provee de servicios a los clientes. Acepta las peticiones, las procesa en orden, da formato a los datos enviados, valida los datos, asegura que la información sea consistente y mantiene seguro el sistema.

• Características básicas del modelo cliente/servidor

→ Características de las aplicaciones que usan el modelo cliente/servidor:

- Hay una combinación cliente-usuarios y otra servidor-recursos. El cliente proporciona una interfaz gráfica al usuario y el servidor permite el acceso remoto a los recursos.
- El servidor hace todo el procesamiento de tratamiento de información necesario.
- La relación cliente-servidor se puede ejecutar en uno o varios pcs distribuidos en red.
- Un servidor puede dar servicio a uno o muchos clientes en orden y sin dejar sin atender a ninguno.
- Los servidores tienen un rol pasivo de espera a que los clientes, con rol activo, envíen las peticiones.
- Las comunicaciones cliente-servidor se hacen mediante el envío de mensajes.
- Los clientes pueden usar cualquier plataforma para conectarse a los servidores.

• Ventajas e inconvenientes del modelo cliente/servidor

→ Ventajas:

- Los clientes son ligeros = no necesitan hardware muy potente.
- Facilidad de integración entre sistemas → interfaces gráficas sencillas de usar para el usuario.
- Predominan interfaces interactivas, con las que transmitiremos solo los datos → menor congestión en la red.
- Sencillez de desarrollo y mantenimiento de este tipo de aplicaciones.
- Popularidad → permite la integración de nuevos componentes y tecnologías → crecimiento escalable.
- Acceso centralizado de los recursos del servidor.
- Los clientes pueden acceder simultáneamente al servidor, compartiendo los datos entre ellos.

→ Inconvenientes:

- Complejidad de mantenimiento de este tipo de sistemas.
- Se tienen que controlar todos los errores posibles del sistema.
- Se tiene que garantizar una buena integridad en el sistema.
- Se debe garantizar la consistencia de la información transmitida.

• Modelos cliente/servidor

→ Clasificación según las capas/triers que tenga:

- Sistema de 1 capa → el cliente y el servidor están en el mismo ordenador y realmente no hay conexión de red para la comunicación.
- Sistema de 2 capas → hay un cliente y un servidor. Este modelo tiene bajísima escalabilidad y se puede sobrecargar si hay muchos clientes o muchas peticiones.
- Sistema de 3 capas → añade una nueva capa que recibe las peticiones del servidor de aplicación con lo que requieren los clientes para poder resolverlas.
- Sistema de n capas → se añaden tantas capas de servidores como sean necesarias, pudiendo separar funcionalidades y mejorar el rendimiento. Ejemplos: P2P, BitTorrent, Skype, Bitcoin.

• Comunicación en el modelo cliente/servidor

→ El modelo cliente/servidor tiene un sistema de comunicación íntegro a través de intercambio de mensajes, que contendrán la información necesaria para ser identificados y enviados a donde se les solicite.

→ La pérdida de mensajes resulta en la corrupción de ficheros cuando se quieren reconstruir. Para evitarlo, por cada mensaje recibido, se envía uno de confirmación al emisor (ACK/acknowledgement). Si esto no le llega al emisor, enviará el paquete de nuevo.

→ Método de envíos ACK es efectivo y útil pero lento. Esto se soluciona enviando varios ACK a la vez al recibir varios paquetes a la vez. Esto necesita llevar un control de errores para paquetes perdidos o desordenados; errores que pueden ser gestionados usando un vector de ACK compuesto por pares (número de mensaje enviado + su ACK).