Tema-1.pdf

cdl_99

Sistemas Operativos

2º Grado en Ingeniería Informática

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación
Universidad de Granada

Tema 1 Stalling

1. Objetivos y funciones de los SO

SO como interfaz usuario/computador

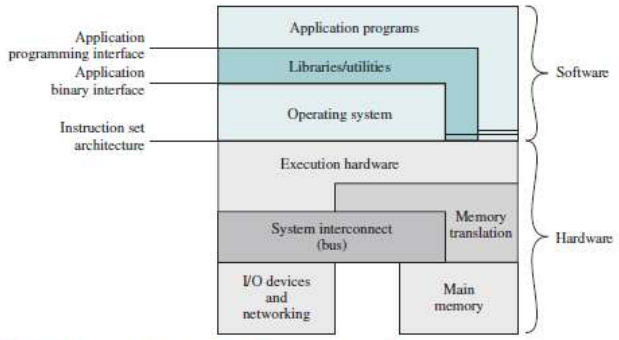


Figure 2.1 Computer Hardware and Software Structure

- **Desarrollo de programas:** el SO proporciona una variedad de utilidades y servicios, tales como editores y depuradores, para asistir al programador en la creación de los programas.
- **Ejecución de programas:** las instrucciones y datos deben de estar cargados previamente en la memoria principal. Los dispositivos de E/S y los ficheros se deben inicializaar, y otros recursos deben prepararse.
- **Acceso a dispositivos de E/S:** cada uno de estos dispositivos requiere su propio ocnjunto peculiar de intrucciones o señales de control para cada operación.
- **Acceso controlado a los ficheros:** el SO debe de reflejar un una comprensión detallada no solo de la naturaleza de los dispositivos de E/S sino tambien de la estructura de los datos contenidos en los ficheros del sistema de almacenamiento.
- **Acceso del sistema:** esta funcion debe proporcionar protección a los recursos y a los datos, evitando el uso no autorizado de los usuarios y resolviendo conflictos en el caso de conflicto de recursos.
- **Detección y respuesta de errores.**
- **Contabilidad:** consiste en recoger estadisticas de uso de los diferentes recursos y monitorizar parametros de rendimiento.
- **ISA (Instruction Set Architecture):** Es como el lenguaje que entiende la compu a nivel máquina. El ISA define las instrucciones que el hardware puede ejecutar. Los programas usan una parte de esas instrucciones (user

ISA), pero el sistema operativo tiene acceso a más instrucciones pa' manejar recursos (system ISA).

- **ABI (Application Binary Interface):** El ABI define cómo los programas interactúan con el sistema operativo, asegurando que el código sea compatible entre diferentes programas y sistemas que usen el mismo ISA.
- **API (Application Programming Interface):** El API permite que los programas accedan a los recursos del sistema usando librerías de alto nivel. Básicamente, facilita que las apps funcionen en diferentes sistemas siempre que usen el mismo

SO como gestor de recursos

Un computador es un conjunto de recursos utilizados para:

- Transportar datos
- Almacenar datos
- Procesar datos
- Controlar estas funciones

El sistema operativo (SO) se encarga de gestionar estos recursos de manera eficiente.

Características Peculiares del Control del SO

El SO actúa como un mecanismo de control inusual en dos aspectos principales:

Funciona como software:

- Es un programa o conjunto de programas
- Se ejecuta por el procesador como cualquier otro software

Control intermitente:

- Frecuentemente cede el control al procesador
- Depende del procesador para volver a retomar el control

El sistema operativo es un conjunto de programas. Como otros programas, proporciona instrucciones para el procesador. La principal diferencia radica en el objetivo del programa. El sistema operativo dirige al procesador en el uso de los otros recursos del sistema y en la temporización de la ejecución de otros programas. No obstante, para que el procesador pueda realizar esto, el sistema operativo debe dejar paso a la ejecución de otros programas. Por tanto, el sistema operativo deja el control para que el procesador pueda realizar trabajo «útil» y de nuevo retoma el control para permitir al procesador que realice la siguiente pieza de trabajo.

Recursos gestionados por el SO

Memoria Principal

Parte del SO reside en la memoria principal, incluyendo:

- El kernel o núcleo (funciones más utilizadas del SO)
- Otras porciones del SO actualmente en uso

El resto de la memoria contiene programas y datos de usuario

La asignación de memoria es controlada conjuntamente por:

- El sistema operativo
- El hardware de gestión de memoria del procesador

Dispositivos de Entrada/Salida (E/S)

El SO decide cuándo un programa en ejecución puede utilizar un dispositivo de E/S

Controla el acceso y uso de estos dispositivos

Archivos

Gestiona el acceso y uso de los archivos en el sistema

Procesador

Determina cuánto tiempo de procesador se asigna a la ejecución de cada programa de usuario

En sistemas multiprocesador, toma decisiones para todos los procesadores

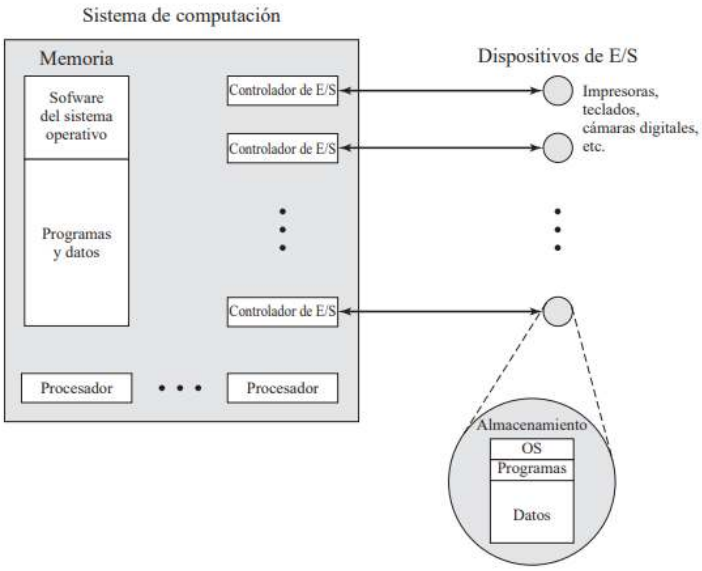


Figura 2.2. El sistema operativo como gestor de recursos.

Facilidad de evolución de un sistema operativo

- Actualizaciones de hardware más nuevos tipos de hardware.
- Nuevos servicios.
- Resolución de fallos.

2. La evolución de los SO

Procesamiento serie

Estos sistemas iniciales (años 40 a finales de los 50) presentaban dos problemas principales:

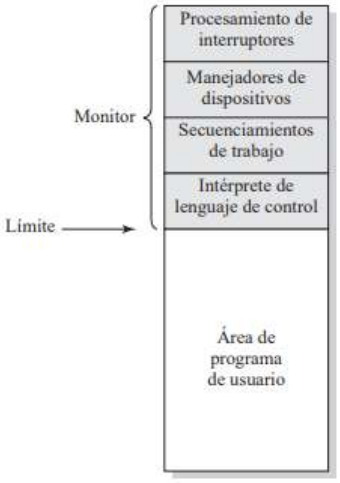
- **Planificación:** La mayoría de las instalaciones utilizaban una plantilla impresa para reservar tiempo de máquina. Típicamente, un usuario podía solicitar un bloque de tiempo en múltiplos de media hora aproximadamente. Un usuario podía obtener una hora y terminar en 45 minutos; esto implicaba malgastar tiempo de procesamiento del computador. Por otro lado, el usuario podía tener problemas, si no finalizaba en el tiempo asignado y era forzado a terminar antes de resolver el problema.
- **Tiempo de configuración:** Un único programa, denominado trabajo, podía implicar la carga en memoria del compilador y del programa en lenguaje de alto nivel (programa en código fuente) y a continuación la carga y el enlace del programa objeto y las funciones comunes.
  Cada uno de estos pasos podían suponer montar y desmontar cintas o configurar tarjetas. Si ocurría un error, el desgraciado usuario normalmente tenía que volver al comienzo de la secuencia de configuración. Por tanto, se utilizaba una cantidad considerable de tiempo en configurar el programa que se iba a ejecutar.

Sistema en lotes sencillos

La idea central bajo el sistema de procesamiento en lotes sencillo es el uso de una pieza de software denominada **monitor**. Con este tipo de SO, el usuario no tiene que acceder directamente a la máquina. En su luegar, el usuario envía un tranajo a través de una tarjeta o cinta al operador del computador, que crea un sistema por lotes con todos los trabajos enviados y coloca la secuencia de dtrabajos en el dispositivo de entrada, para que lo utilice el monitor.

Para entener este esquema hay que ver el punto de vista del monitor y el del procesador:

- **Punto de vista del monitor:** el monitor controla la secuencia de eventos. Para ello, una gran parte del monitor debe estar siempre en memoria principal y disponible para la ejecución. El monitor lee de uno en uno los trabajos desde el dispositivo de entrada. Una vez leido el dispositivo, el trabajo actual se coloca en el area de programa de usuario, y se le pasa el control. Cuando el trabajo se ha completado, devuelve el control al monitor, que inmediatamente lee el siguiente trabajo. Los resultados de cada trabajo se envían a un dispositivo de salida para entregarselo al usuario.



- **Punto de vista del procesador:** El procesador arranca ejecutando instrucciones del monitor (que está en la memoria principal). Estas instrucciones le dicen que lea el siguiente trabajo y lo meta en otra parte de la memoria. Cuando lo hace, el procesador recibe una instrucción que le dice que empiece a ejecutar el programa del usuario. Ejecuta ese programa hasta que termine o haya un error. En ese momento, vuelve a ejecutar instrucciones del monitor.

  Cuando se dice que "se le pasa el control a un trabajo", significa que el procesador está ejecutando el programa del usuario, y cuando "se devuelve el control al monitor", es que vuelve a ejecutar instrucciones del monitor.

  El monitor, o sistema operativo por lotes, es simplemente un programa de computadora. Depende de la capacidad del procesador para obtener instrucciones de varias partes de la memoria principal y alternar entre tomar y soltar el control. Ciertas otras características de hardware también son deseables:

  - Protección de memoria
  - Temporizador
  - Instrucciones privilegiadas
  - Interrupciones

  Las consideraciones sobre la protección de memoria y las instrucciones privilegiadas llevan al concepto de **modos de operación**. Un programa de usuario se ejecuta en **modo usuario**, donde ciertas áreas de la memoria están protegidas y no pueden ser usadas por el usuario, y donde ciertas instrucciones no se pueden ejecutar. El monitor se ejecuta en **modo sistema** o **modo kernel**, donde se pueden ejecutar instrucciones privilegiadas y acceder a áreas protegidas de la memoria.

Sistemas por lotes multiprogramados

Son aquellos en los que varios trabajos se mantienen en la memoria principal al mismo tiempo. Mientras un trabajo está esperando (por ejemplo, esperando que finalice una operación de entrada/salida), el procesador puede seguir

ejecutando otro trabajo. Esto aumenta la eficiencia del sistema al mantener el procesador ocupado la mayor parte del tiempo y permite que se ejecuten múltiples trabajos de manera simultánea, aprovechando mejor los recursos del sistema.

Sistemas de tiempo compartido

Son una evolución de la multiprogramación que permite la **interacción directa** entre el usuario y la computadora. A diferencia de los sistemas por lotes, donde los trabajos se ejecutan sin intervención del usuario, en los sistemas de tiempo compartido, múltiples usuarios pueden acceder al sistema simultáneamente a través de terminales. El procesador reparte su tiempo entre varios usuarios, ejecutando fragmentos cortos de cada programa en lo que se llama un "quantum" de tiempo.

3. Principales logros

El proceso

El concepto de proceso en sistemas operativos surgió de tres líneas principales de desarrollo en sistemas computacionales:

1. Operación batch multiprogramada:
    - Buscaba eficiencia máxima mediante el uso simultáneo de CPU y dispositivos de E/S.
    - Utilizaba interrupciones para alternar entre programas en memoria principal.
2. Tiempo compartido de propósito general:
    - Objetivo: responder a múltiples usuarios simultáneamente de manera eficiente.
    - Aprovechaba el tiempo de reacción relativamente lento de los usuarios.
3. Sistemas de procesamiento de transacciones en tiempo real:
    - Ejemplo: sistemas de reservas de aerolíneas.
    - Enfocados en aplicaciones específicas, a diferencia de los sistemas de tiempo compartido.
    - Prioridad en el tiempo de respuesta del sistema.

El desarrollo de estos sistemas presentó desafíos en sincronización y temporización. La herramienta principal para abordar estos problemas fue el uso de interrupciones, que permitían suspender y reanudar tareas.

Sin embargo, la complejidad de coordinar múltiples actividades simultáneas llevó a dificultades en el diseño del software del sistema:

- Imposibilidad de analizar todas las posibles combinaciones de secuencias de eventos.
- Uso de métodos ad hoc por parte de los programadores.

- Vulnerabilidad a errores sutiles de programación, difíciles de diagnosticar y reproducir.

Estos desafíos contribuyeron al desarrollo del concepto de proceso como una forma más sistemática de manejar y coordinar múltiples actividades en un sistema operativo.

Las cuatro causas principales de errores en sistemas operativos multiprogramados y de tiempo compartido:

1. Sincronización inadecuada:
   - Ocurre cuando una rutina debe esperar un evento en otra parte del sistema.
   - Problemas: pérdida de señales o recepción de señales duplicadas.
2. Fallo en la exclusión mutua:
   - Surge cuando múltiples usuarios o programas intentan usar un recurso compartido simultáneamente.
   - Requiere mecanismos de control de acceso.
   - Difícil de verificar su corrección en todas las posibles secuencias de eventos.
3. Operación no determinista del programa:
   - Los resultados de un programa deberían depender solo de sus entradas.
   - En sistemas compartidos, la interferencia entre programas puede causar resultados impredecibles.
   - El orden de programación de tareas puede afectar el resultado de un programa particular.
4. Deadlocks (interbloqueos):
   - Situación donde dos o más programas quedan esperando indefinidamente entre sí.
   - Ejemplo: dos programas requieren dos dispositivos de E/S, cada uno tiene control de un dispositivo y espera por el otro.
   - Pueden depender de la temporización aleatoria de la asignación y liberación de recursos.

Estos problemas surgieron de la complejidad de coordinar múltiples actividades en sistemas compartidos y contribuyeron a la necesidad de desarrollar conceptos más avanzados en sistemas operativos, como el de proceso, para manejar estas situaciones de manera más efectiva y sistemática.

Gestión de memoria

El sistema operativo tiene cinco responsabilidades clave para gestionar la memoria de manera eficiente:

1. **Aislamiento de procesos**: Debe prevenir que los procesos interfieran entre sí, tanto en los datos como en las instrucciones.

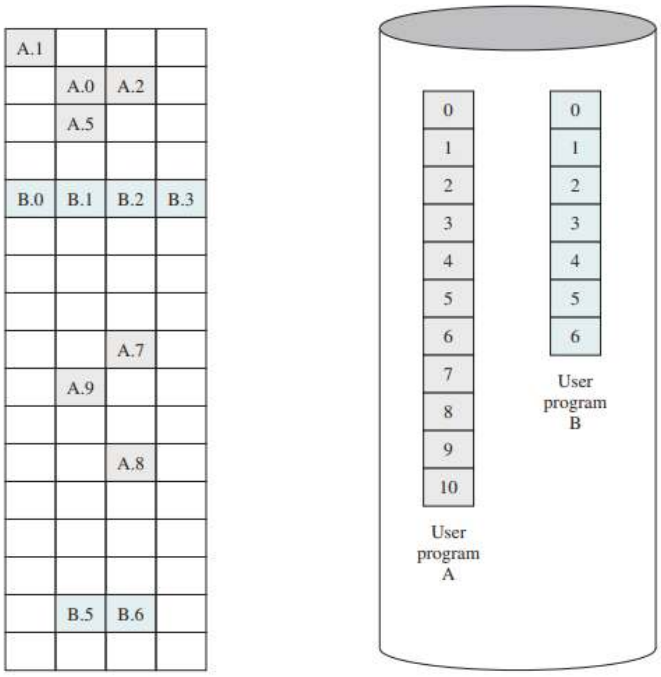
2. **Asignación y gestión automática**: La memoria debe asignarse dinámicamente según lo requiera el programa, de forma transparente para el programador, optimizando el uso de recursos.
3. **Soporte a la programación modular**: Permitir que los programadores definan, creen, destruyan y modifiquen módulos de manera dinámica dentro de los programas.
4. **Protección y control de acceso**: El sistema debe garantizar que se pueda compartir memoria entre programas cuando sea necesario, pero evitando accesos no autorizados que comprometan su integridad.
5. **Almacenamiento a largo plazo**: Proveer mecanismos para almacenar información de forma persistente, incluso cuando el sistema se apaga.

Los sistemas operativos generalmente cumplen con las necesidades de almacenamiento a largo plazo y gestión de memoria utilizando **memoria virtual** y **sistemas de archivos**. El sistema de archivos organiza los datos en objetos llamados **archivos**, que facilitan el acceso y control para el programador y el sistema operativo.

Memoria virtual permite que los programas trabajen con una visión lógica de la memoria, sin depender de la cantidad física disponible. Esto es clave para gestionar múltiples tareas al mismo tiempo, evitando pausas entre procesos cuando uno se escribe en el almacenamiento secundario y otro se carga.

Dado que los procesos varían en tamaño y es complicado organizarlos eficientemente en la memoria física, se introdujo el **sistema de paginación**. Este divide los procesos en bloques de tamaño fijo llamados **páginas**. Cada página tiene una dirección virtual (número de página y desplazamiento) que se traduce dinámicamente a una dirección física en la memoria principal, permitiendo que las páginas se ubiquen en cualquier parte de la memoria.

Con hardware capaz de hacer este mapeo dinámico, se eliminó la necesidad de que todas las páginas de un proceso estén en memoria al mismo tiempo. Ahora, las páginas se mantienen en disco y se cargan solo cuando el sistema operativo las necesita, lo que es la esencia de la **memoria virtual**.



Main memory consists of a number of fixed-length frames, each equal to the size of a page. For a program to execute, some or all of its pages must be in main memory.

Secondary memory (disk) can hold many fixed-length pages. A user program consists of some number of pages. Pages of all programs plus the OS are on disk, as are files.

Protección de información y seguridad

El crecimiento de los sistemas de **tiempo compartido** (time-sharing) y redes ha generado un aumento en la preocupación por la protección de la información. Dependiendo de la organización, las amenazas pueden variar, pero existen herramientas generales que se integran en los sistemas operativos para ofrecer mecanismos de protección y seguridad. La principal preocupación es controlar el acceso a los sistemas y a la información almacenada.

El trabajo en seguridad y protección dentro de los sistemas operativos se puede dividir en cuatro áreas clave:

1. **Disponibilidad**: Proteger el sistema contra interrupciones para asegurar su funcionamiento continuo.
2. **Confidencialidad**: Garantizar que los usuarios no puedan acceder a datos sin la autorización adecuada.
3. **Integridad de los datos**: Proteger los datos contra modificaciones no autorizadas.
4. **Autenticidad**: Verificar correctamente la identidad de los usuarios y la validez de los mensajes o datos intercambiados.

Programación y gestión de recursos

Una responsabilidad clave del sistema operativo es gestionar los recursos disponibles (memoria, dispositivos de E/S, procesadores) y programar su uso entre los procesos activos. Para una asignación y planificación efectiva, se deben considerar tres factores principales:

1. **Justicia**: Todos los procesos que compiten por un recurso deben tener acceso justo y equitativo, especialmente aquellos que pertenecen a la misma clase o tienen demandas similares.
2. **Respuesta diferencial**: El sistema operativo debe discriminar entre clases de trabajos con diferentes necesidades. Debe tomar decisiones de asignación y programación dinámicamente, como dar prioridad a procesos que esperan por un dispositivo de E/S, para liberar el recurso lo antes posible.
3. **Eficiencia**: El objetivo es maximizar el rendimiento, minimizar el tiempo de respuesta y, en entornos de tiempo compartido, atender a la mayor cantidad de usuarios posible.

4. Desarrollos que han llevado a los sistemas modernos

Tradicionalmente, la mayoría de los sistemas operativos utilizaban un **núcleo monolítico**, donde todas las funciones del sistema operativo (gestión de archivos, redes, controladores de dispositivos, etc.) se integraban en un único proceso compartiendo el mismo espacio de direcciones.

Entre las principales tendencias y enfoques que se han probado, tanto en sistemas comerciales como experimentales, destacan:

- **Arquitectura de microkernel:** asigna solo las funciones esenciales al núcleo, como la gestión del espacio de direcciones, la comunicación entre procesos (IPC) y la planificación básica. Los demás servicios del sistema operativo son manejados por procesos independientes (servidores) que corren en modo usuario, como si fueran aplicaciones comunes. Esto separa el desarrollo del núcleo del de los servidores, permitiendo mayor flexibilidad y personalización. Además, facilita la construcción de sistemas distribuidos, ya que el microkernel interactúa de la misma manera con servidores locales y remotos.
- **Multihilo (multithreading):** es una técnica donde un proceso, que ejecuta una aplicación, se divide en varios hilos que pueden correr de manera concurrente. Aquí se puede hacer una distinción importante:
  - **Hilo (Thread)**: Es una unidad de trabajo despachable. Incluye un contexto de procesador (que contiene el contador de programa y el puntero de pila) y su propia área de datos para la pila (necesaria para gestionar subrutinas). Un hilo se ejecuta secuencialmente y puede ser interrumpido para que el procesador cambie a otro hilo.
  - **Proceso**: Es una colección de uno o más hilos y los recursos del sistema asociados, como la memoria que contiene tanto el código como los datos,

archivos abiertos y dispositivos. Se asemeja al concepto de un programa en ejecución.

Dividiendo una aplicación en múltiples hilos, el programador obtiene mayor control sobre la modularidad de la aplicación y la gestión del tiempo de los eventos relacionados con ella.

- **Procesamiento simétrico múltiple (SMP):** el sistema operativo programa procesos o hilos entre todos los procesadores disponibles. Este enfoque tiene varias ventajas sobre una arquitectura de un solo procesador:
  - **Mejor rendimiento**: Al ejecutar tareas en paralelo, varios procesadores aumentan el rendimiento en comparación con un solo procesador.
  - **Mayor disponibilidad**: Si un procesador falla, el sistema sigue funcionando, aunque con menor rendimiento.
  - **Crecimiento incremental**: Se puede mejorar el rendimiento añadiendo más procesadores.
  - **Escalabilidad**: Se pueden crear productos con diferentes niveles de rendimiento y precios, según el número de procesadores.
- **Sistemas operativos distribuidos:** crea la ilusión de una única memoria principal y secundaria, junto con acceso unificado a otros recursos, como un sistema de archivos distribuido. Aunque los **clusters** son cada vez más populares, los sistemas operativos distribuidos están menos desarrollados en comparación con los sistemas uniprocesador y SMP.
- **Diseño orientado a objetos:** permiten añadir módulos de manera ordenada a un núcleo pequeño, facilitando la personalización del sistema sin comprometer su integridad. Además, la orientación a objetos simplifica el desarrollo de herramientas y sistemas distribuidos.

5. Tolerancia a fallos

La **tolerancia a fallos** es la capacidad de un sistema o componente de seguir operando normalmente a pesar de fallos en hardware o software, generalmente mediante algún tipo de **redundancia**. Su objetivo es mejorar la **confiabilidad** del sistema. Sin embargo, implementar tolerancia a fallos suele tener un costo, ya sea en términos financieros, de rendimiento, o ambos. La adopción de medidas de tolerancia a fallos depende de la **importancia crítica** del recurso que se quiere proteger.

Fallos

Se pueden clasificar en las siguientes categorías:

- **Permanentes**: Son fallos que, una vez ocurren, permanecen hasta que se repara o reemplaza el componente defectuoso. Ejemplos incluyen fallos en el cabezal de un disco, errores de software o un componente de comunicación quemado.
- **Temporales**: No están presentes todo el tiempo ni bajo todas las condiciones. Estos se dividen en:

- **Transitorios**: Ocurren solo una vez, como errores de transmisión de bits por ruido, fluctuaciones en la fuente de energía o radiación que altera un bit de memoria.
- **Intermitentes**: Ocurren en momentos impredecibles y múltiples veces, como una conexión suelta que provoca fallos de manera esporádica.

La **tolerancia a fallos** generalmente se logra añadiendo **redundancia** al sistema. Existen varios métodos de redundancia:

- **Redundancia espacial (física)**: Consiste en usar múltiples componentes que realizan la misma función simultáneamente o tener un componente de respaldo en caso de que falle otro. Un ejemplo es el uso de circuitos paralelos o un servidor de nombres de respaldo en Internet.
- **Redundancia temporal**: Implica repetir una operación cuando se detecta un error. Es útil para fallos temporales, pero no para fallos permanentes. Un ejemplo es la retransmisión de datos cuando se detecta un error en protocolos de control de enlace.
- **Redundancia de información**: Consiste en replicar o codificar los datos para que los errores de bits puedan ser detectados y corregidos. Ejemplos incluyen la codificación de control de errores en sistemas de memoria y las técnicas de corrección de errores en discos RAID.

Mécanismos del SO

Los sistemas operativos pueden incorporar varias técnicas para soportar la **tolerancia a fallos**. Algunos ejemplos incluyen:

- **Aislamiento de procesos**: Los procesos están generalmente aislados en términos de memoria, acceso a archivos y ejecución, lo que protege a otros procesos de un fallo en uno de ellos.
- **Control de concurrencia**: Se utilizan técnicas para manejar la comunicación y cooperación entre procesos, evitando problemas como bloqueos (deadlocks) y recuperando el sistema en caso de fallos.
- **Máquinas virtuales**: Ofrecen mayor aislamiento entre aplicaciones, lo que ayuda a contener fallos. También pueden servir para redundancia, donde una máquina virtual puede actuar como respaldo de otra.
- **Puntos de control y retrocesos (checkpoints y rollbacks)**: Se guarda una copia del estado de la aplicación en un punto seguro. Si ocurre un fallo, el sistema vuelve a ese estado y reanuda la ejecución desde allí, ayudando a recuperarse de fallos transitorios o permanentes.

6. Consideraciones en el diseño del SO en multiprocesadores y multicore

Consideraciones sobre el SO multiprocesador simétrico

Un sistema operativo de **procesamiento simétrico múltiple (SMP)** gestiona los procesadores y otros recursos del ordenador para que el usuario pueda ver el sistema de manera similar a un sistema uniprocesador con multiprogramación.

Esto permite a los usuarios desarrollar aplicaciones que utilicen múltiples procesos o múltiples hilos dentro de los procesos, sin preocuparse por si habrá un solo procesador o múltiples disponibles. Por lo tanto, un sistema operativo multiprocesador debe ofrecer todas las funcionalidades de un sistema de multiprogramación, además de características adicionales para manejar múltiples procesadores. Los principales problemas de diseño incluyen lo siguiente:

- **Procesos concurrentes**: Las rutinas del núcleo deben ser **reentrantes** para que varios procesadores ejecuten el mismo código simultáneamente, evitando la **corrupción de datos**.
- **Planificación**: Cualquier procesador puede realizar la planificación, lo que complica la implementación de políticas de programación y la protección de las estructuras de datos del planificador. Se puede programar múltiples hilos de un mismo proceso en diferentes procesadores.
- **Sincronización**: Es esencial implementar mecanismos que garanticen la **exclusión mutua** y el **orden de eventos** entre procesos activos que acceden a recursos compartidos, utilizando **bloqueos** como herramienta común.
- **Gestión de memoria**: Debe abordar problemas de sistemas uniprocesador y aprovechar el **paralelismo hardware**. Los mecanismos de **paginación** deben coordinarse para mantener la **consistencia** entre procesadores que comparten páginas.
- **Confiabilidad y tolerancia a fallos**: El sistema operativo debe permitir una **degradación gradual** ante fallos de procesadores, ajustando las estructuras de gestión según sea necesario.

Consideraciones sobre SO multinúcleo

Las consideraciones para sistemas operativos en **multicore**:

1. **Desafíos de diseño**: Al igual que en los sistemas SMP, pero con preocupaciones adicionales relacionadas con la escala del **paralelismo**. Los sistemas multicore actuales pueden tener diez o más núcleos en un solo chip, y la tendencia es hacia sistemas "many-core".
2. **Aprovechamiento eficiente de recursos**: El reto es gestionar eficazmente la potencia de procesamiento multicore y los recursos sustanciales en el chip, asegurando que el paralelismo inherente se alinee con los requisitos de rendimiento de las aplicaciones.
3. **Niveles de paralelismo**:
   - **Paralelismo de hardware**: Dentro de cada núcleo, conocido como **paralelismo a nivel de instrucción**, que puede o no ser aprovechado por programadores y compiladores.
   - **Ejecución de multiprogramación y multihilo**: Dentro de cada procesador.
   - **Ejecución concurrente de aplicaciones**: Utilizando múltiples núcleos.
4. **Soporte del sistema operativo**: Es crucial contar con un sólido soporte del sistema operativo para el paralelismo en la multiprogramación y multihilo

para garantizar un uso eficiente de los recursos hardware.

5. **Exploración de estrategias**: Desde la llegada de la tecnología multicore, los diseñadores de sistemas operativos buscan maneras de extraer el paralelismo de las cargas de trabajo computacionales. Se están investigando diversas estrategias para los sistemas operativos de próxima generación.

Las aplicaciones suelen dividirse en tareas que pueden ejecutarse en paralelo, pero el reto es que el desarrollador debe decidir cómo dividir el trabajo. Las herramientas, como el compilador y las características del lenguaje de programación, ayudan en este proceso. El sistema operativo (OS) también apoya al asignar recursos de manera eficiente entre las tareas paralelas.

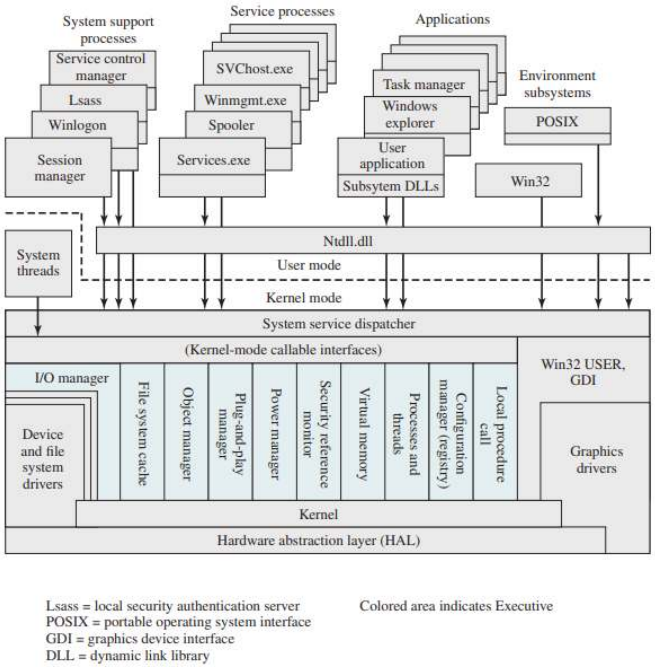
Otra estrategia es el enfoque de Máquina Virtual, donde en lugar de multiprogramar núcleos individuales, se dedican uno o más núcleos a un proceso específico, evitando la sobrecarga de cambios de tarea. El OS podría funcionar como un hipervisor, asignando núcleos a aplicaciones sin hacer mucha más gestión de recursos. Esta idea busca simplificar la gestión de recursos en sistemas con muchos núcleos, permitiendo que las aplicaciones manejen más eficientemente su propio uso de recursos.

> ⚠️ Hay dos tipos principales de hipervisores:
>
> 1. **Hypervisores Tipo 1 (Bare Metal)**: Se ejecutan directamente sobre el hardware físico del servidor. No requieren un sistema operativo host y gestionan directamente las máquinas virtuales. Ejemplos incluyen VMware ESXi, Microsoft Hyper-V, y KVM.
> 2. **Hypervisores Tipo 2 (Hosted)**: Se ejecutan como aplicaciones dentro de un sistema operativo host. Requieren un sistema operativo host para funcionar y gestionan las máquinas virtuales como procesos dentro de ese sistema operativo. Ejemplos incluyen VMware Workstation, VirtualBox, y Parallels Desktop.

7. Visión general de Windows

Arquitectura

Windows tiene una arquitectura modular donde cada función del sistema es gestionada por un solo componente del sistema operativo (SO). Las aplicaciones y el resto del SO acceden a estas funciones a través de interfaces estándar, y los datos clave del sistema solo se pueden acceder mediante las funciones apropiadas. Esto permite que cualquier módulo se pueda quitar, actualizar o reemplazar sin necesidad de reescribir todo el sistema o sus interfaces de programación de aplicaciones (APIs).



Los componentes en modo kernel de Windows son:

- **Executive**: Contiene los servicios centrales del SO, como gestión de memoria, procesos y hilos, seguridad, E/S y comunicación entre procesos.
- **Kernel**: Controla la ejecución de los procesadores, gestionando la planificación de hilos, el cambio de procesos, el manejo de excepciones e interrupciones, y la sincronización en multiprocesadores. A diferencia del Executive y el nivel de usuario, el código del Kernel no se ejecuta en hilos.
- **Hardware Abstraction Layer (HAL)**: Actúa como un intermediario entre comandos y respuestas de hardware genéricos y aquellos únicos de una plataforma específica, aislando al SO de las diferencias del hardware específico. El HAL estandariza componentes como el bus del sistema, el controlador DMA, controladores de interrupciones y temporizadores del sistema.
- **Controladores de dispositivos**: Bibliotecas dinámicas que amplían la funcionalidad del Executive, incluyendo controladores de hardware que traducen las llamadas de función de E/S del usuario en solicitudes específicas de hardware, así como componentes de software para implementar sistemas de archivos y protocolos de red que deben ejecutarse en modo kernel.
- **Sistema de ventanas y gráficos**: Implementa las funciones de la interfaz gráfica de usuario (GUI), gestionando ventanas, controles de interfaz de usuario y gráficos.

El Executive de Windows incluye módulos específicos para diversas funciones del sistema y proporciona una API para el software en modo usuario. Aquí te dejo una breve descripción de cada módulo:

```
Administrador de E/S
Administrador de Caché
Administrador de objetos
Administrador de Plug-and-play
Administrador de energía
Monitor de referencia de seguridad
Administrador de memoria virtual
Administrador de procesos/hilos
Administrador de configuración
Llamada a procedimiento local avanzado (ALPC)
```

Los tipos básicos de procesos en modo usuario que Windows soporta:

1. **Procesos especiales del sistema**: Servicios en modo usuario necesarios para gestionar el sistema, como el administrador de sesiones, subsistema de autenticación, administrador de servicios y proceso de inicio de sesión.
2. **Procesos de servicio**: Incluyen el spooler de impresoras, el registro de eventos, componentes en modo usuario que cooperan con controladores de dispositivos y varios servicios de red. Estos servicios son utilizados tanto por Microsoft como por desarrolladores externos para extender la funcionalidad del sistema y son la única forma de ejecutar actividades en segundo plano en modo usuario.
3. **Subsistemas de entorno**: Proveen diferentes "personalidades" del SO, como Win32 y POSIX. Cada subsistema tiene un proceso compartido entre aplicaciones que convierte las llamadas de las aplicaciones en llamadas nativas de Windows o de ALPC.
4. **Aplicaciones de usuario**: Archivos ejecutables (EXEs) y bibliotecas de enlace dinámico (DLLs) que ofrecen la funcionalidad que los usuarios ejecutan. Están dirigidos a un subsistema de entorno específico, aunque algunos programas del propio SO usan las interfaces nativas (NT API). También hay soporte para ejecutar programas de 32 bits en sistemas de 64 bits.

Modelo Cliente/Servidor

Las ventajas de una arquitectura cliente/servidor son las siguientes:

1. **Simplificación del Executive**: Permite construir varias APIs en servidores de modo usuario sin conflictos o duplicaciones en el Executive. Se pueden agregar nuevas APIs fácilmente.
2. **Mejora de la confiabilidad**: Cada servidor nuevo corre fuera del kernel y tiene su propia memoria protegida, lo que evita que el fallo de un servidor afecte al resto del SO.

3. **Comunicación uniforme**: Las aplicaciones se comunican con los servicios a través de llamadas a procedimientos remotos (RPC), ocultando el proceso de paso de mensajes mediante pequeños fragmentos de código llamados "stubs". Esto permite que las aplicaciones llamen APIs sin preocuparse por el manejo de mensajes.
4. **Base adecuada para computación distribuida**: La arquitectura cliente/servidor es ideal para computación distribuida, ya que permite que un servidor local pase mensajes a un servidor remoto para procesar solicitudes de aplicaciones locales, sin que el cliente necesite saber si el servicio se realiza local o remotamente.

Hilos y SMP

Las características clave de Windows relacionadas con hilos y multiprocesamiento simétrico (SMP):

1. **Rutinas del SO en cualquier procesador**: Las rutinas del SO pueden ejecutarse en cualquier procesador disponible, y diferentes rutinas pueden ejecutarse simultáneamente en distintos procesadores.
2. **Hilos múltiples en un proceso**: Windows permite el uso de múltiples hilos de ejecución dentro de un solo proceso, y estos hilos pueden ejecutarse en diferentes procesadores al mismo tiempo.
3. **Procesos de servidor con múltiples hilos**: Los procesos de servidor pueden utilizar múltiples hilos para manejar simultáneamente solicitudes de más de un cliente.
4. **Mecanismos para compartir datos y recursos**: Windows ofrece mecanismos para compartir datos y recursos entre procesos, así como capacidades flexibles de comunicación entre procesos.

Objetos de Windows

Aunque el núcleo de Windows está escrito en C, sigue principios de diseño orientado a objetos. Esto facilita compartir recursos y datos entre procesos, al mismo tiempo que protege los recursos de accesos no autorizados. Algunos de los conceptos clave de diseño orientado a objetos utilizados por Windows incluyen:

1. **Encapsulación**: Los datos de un objeto solo se acceden a través de sus servicios, protegiéndolos de usos no autorizados o incorrectos.
2. **Clase e instancia de objeto**: Una clase define atributos y servicios, y el SO crea instancias específicas según sea necesario.
3. **Herencia**: El Executive extiende clases de objetos añadiendo características nuevas basadas en clases base que manejan la creación, seguridad y eliminación de objetos.
4. **Polimorfismo**: Windows usa funciones API comunes para manipular diferentes tipos de objetos, aunque también tiene APIs específicas para algunos tipos.

Los **objetos del Executive** (o del kernel) en Windows son bloques de memoria accesibles solo por componentes en modo kernel. Estos objetos tienen atributos comunes (como nombre, parámetros de seguridad, y uso) y específicos (como la prioridad de un hilo). Las aplicaciones no acceden directamente a estos objetos, sino que lo hacen a través de funciones que manipulan objetos mediante **handles**. Un handle es un índice en una tabla que apunta al objeto referenciado, permitiendo que los hilos dentro del proceso lo usen.

Los objetos pueden tener un **Descriptor de Seguridad (SD)**, que limita el acceso según el usuario, y pueden ser nombrados o no. Los objetos **sin nombre** se refieren solo mediante handles, mientras que los **objetos con nombre** permiten a otros procesos obtener handles para acceder a ellos.

Para la sincronización, Windows utiliza dos tipos de objetos:

- **Objetos despachadores**: Controlan la sincronización de operaciones basadas en hilos.
- **Objetos de control**: Gestionan el uso del procesador fuera de la planificación normal de hilos.

Aunque Windows no es completamente orientado a objetos, aplica muchos conceptos de esta tecnología en su diseño.

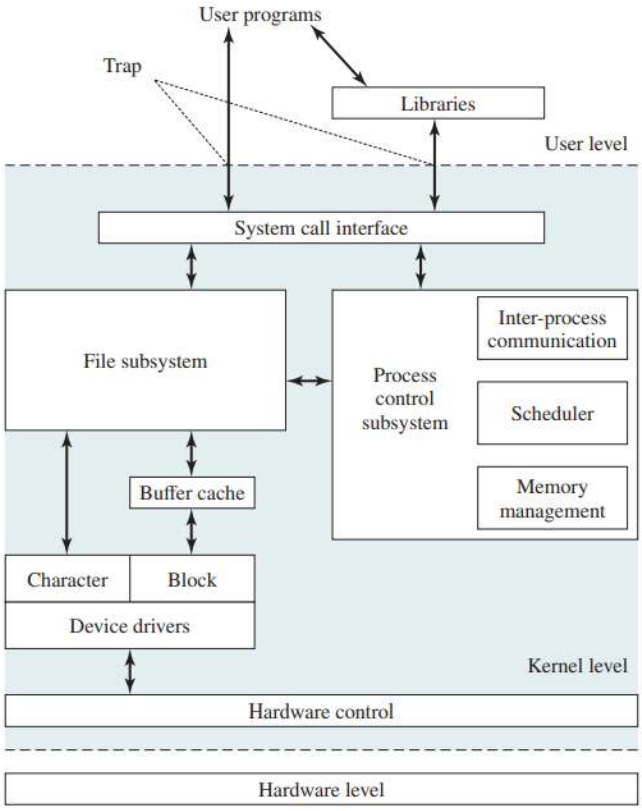
8. Sistemas UNIX tradicionales

Arquitectura

La arquitectura clásica de UNIX se divide en tres niveles: hardware, kernel y usuario. El **kernel** es el núcleo del sistema operativo y se aísla de los usuarios y aplicaciones, interactuando directamente con el hardware. Los servicios y las interfaces para el usuario incluyen el shell, las llamadas al sistema y herramientas como el compilador C.

El kernel se organiza en dos partes principales: **control de procesos** y **gestión de archivos y E/S**. El control de procesos maneja la memoria, la planificación de procesos y la comunicación entre ellos. La gestión de archivos se encarga del intercambio de datos entre la memoria y los dispositivos externos mediante controladores de dispositivos, usando un caché de disco para optimizar las transferencias de bloques de datos.

Este diseño es típico de los sistemas UNIX tradicionales como **System V Release 3** y **4.3BSD**, que están pensados para funcionar en un solo procesador. No tienen la capacidad de proteger sus estructuras de datos ante accesos concurrentes y su kernel no es muy modular, lo que provocó que al agregar nuevas funciones el sistema se volviera más complejo y difícil de extender.



9. LINUX

Estructura modular

La mayoría de los kernels de UNIX son **monolíticos**, lo que significa que incluyen casi toda la funcionalidad del sistema operativo en un solo bloque de código que corre como un único proceso. Esto implica que cualquier cambio, como añadir un nuevo controlador de dispositivo, requiere relinkear todos los módulos y reiniciar el sistema, lo que dificulta las modificaciones, especialmente en Linux.

A pesar de no usar un enfoque de **microkernel**, Linux logra algunas ventajas de este modelo a través de su **arquitectura modular**. Está compuesto por **módulos** que se pueden cargar y descargar automáticamente según la demanda. Un módulo es un archivo objeto que se puede enlazar al kernel en tiempo de ejecución y generalmente implementa funciones específicas como controladores de dispositivos o sistemas de archivos, ejecutándose en modo kernel en nombre del proceso actual.

Aunque Linux se considera monolítico, su **estructura modular** ayuda a sortear algunas dificultades en el desarrollo y evolución del kernel. Los **módulos cargables de Linux** tienen dos características importantes:

- **Vinculación dinámica**: Un módulo del kernel puede cargarse y vincularse al kernel mientras ya está en memoria y en ejecución. También se puede desvincular y eliminar de la memoria en cualquier momento.
- **Módulos apilables**: Los módulos están organizados en una jerarquía. Los módulos individuales funcionan como bibliotecas cuando son referenciados por módulos clientes que están más arriba en la jerarquía, y como clientes cuando referencian módulos que están más abajo.

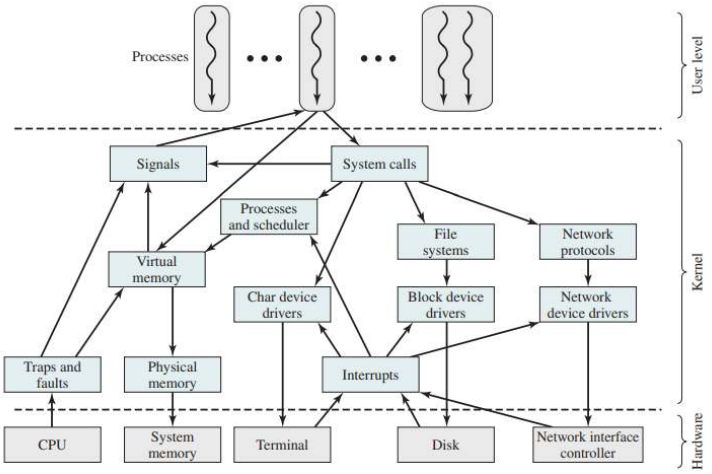
La **vinculación dinámica** facilita la configuración y ahorra memoria en el kernel. En Linux, un programa de usuario o el usuario pueden cargar y descargar explícitamente módulos del kernel usando los comandos **insmod** o **modprobe** para cargar, y **rmmod** para descargar. El kernel también monitorea la necesidad de funciones particulares y puede cargar o descargar módulos según sea necesario. Con los módulos apilables, se pueden definir dependencias entre módulos, lo que tiene dos beneficios:

1. **Reducción de replicación**: El código común a un conjunto de módulos similares (por ejemplo, controladores para hardware similar) puede trasladarse a un solo módulo.
2. **Gestión de dependencias**: El kernel se asegura de que los módulos necesarios estén presentes, evitando descargar un módulo del que dependen otros módulos en ejecución, y cargando cualquier módulo adicional requerido cuando se carga un nuevo módulo.

Componentes del núcleo

Los principales componentes del kernel son los siguientes:

- **Señales**: El kernel utiliza señales para llamar a un proceso. Por ejemplo, se usan para notificar a un proceso sobre ciertos errores, como la división por cero.
- **Llamadas al sistema**: Las llamadas al sistema son el medio por el cual un proceso solicita un servicio específico del kernel. Hay varios cientos de llamadas al sistema, que se pueden agrupar en seis categorías: sistema de archivos, proceso, planificación, comunicación entre procesos, sockets (redes) y misceláneos.
- **Procesos y programador**: Se encarga de crear, gestionar y programar procesos.
- **Memoria virtual**: Asigna y gestiona la memoria virtual para los procesos



- **Sistemas de archivos**: Proporcionan un espacio de nombres jerárquico y global para archivos, directorios y otros objetos relacionados, además de funciones para gestionar el sistema de archivos.
- **Protocolos de red**: Soportan la interfaz de Sockets para los usuarios en la suite de protocolos TCP/IP.
- **Controladores de dispositivos de carácter**: Gestionan dispositivos que requieren que el kernel envíe o reciba datos byte a byte, como terminales, módems e impresoras.
- **Controladores de dispositivos de bloques**: Gestionan dispositivos que leen y escriben datos en bloques, como discos magnéticos y CD-ROMs.
- **Controladores de dispositivos de red**: Gestionan tarjetas de interfaz de red y puertos de comunicación conectados a dispositivos de red, como puentes y enrutadores.
- **Trampas y fallos**: Manejan trampas y fallos generados por el procesador, como fallos de memoria.
- **Memoria física**: Administra el conjunto de marcos de página en la memoria real y asigna páginas para la memoria virtual.
- **Interrupciones**: Manejan interrupciones provenientes de dispositivos periféricos.

Aproximaciones del nucleo para maquinas virtuales

Namespaces

Namespaces ofrecen una forma ligera de virtualización permitiendo ver las propiedades globales de un sistema en ejecución desde diferentes perspectivas. Este mecanismo es similar a las zonas en Solaris o al jail en FreeBSD. Tras un repaso general del concepto, discutiré la infraestructura que ofrece el framework de namespaces.

Concepto

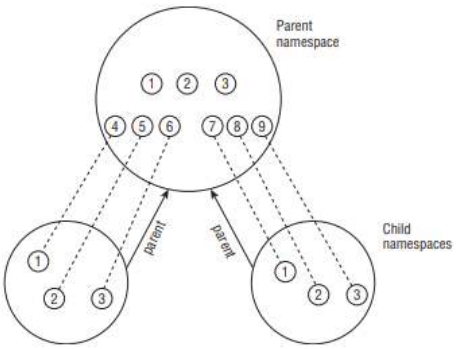
En Linux y otros sistemas Unix, muchos recursos se manejan globalmente, como los PIDs, los nombres de usuario y la información del sistema. Todos los procesos tienen un PID único y global, y la información del sistema obtenida con **uname** es la misma para todos los usuarios. Lo mismo aplica a los UIDs, que identifican a cada usuario de forma global.

Los identificadores globales permiten al kernel gestionar privilegios, como que solo el usuario root (UID 0) pueda hacer cualquier cosa, mientras que otros usuarios tienen limitaciones. Aunque los usuarios no pueden interferir con procesos ajenos, sí pueden ver que otros están activos en el sistema, lo cual generalmente no es un problema. Sin embargo, en algunos casos, como en un servidor web que ofrece acceso completo a clientes, esto puede ser indeseado.

Antes, esto implicaba una máquina por cliente, lo cual es costoso. La virtualización con KVM o VMWare es una opción, pero no optimiza bien los recursos ya que requiere un kernel por cliente. Los namespaces ofrecen una solución más eficiente, permitiendo que un único kernel maneje múltiples entornos virtuales, separando recursos previamente globales. Los procesos se agrupan en contenedores, que pueden estar completamente aislados o compartir algunos recursos, como partes del sistema de archivos.

En un ejemplo, tres namespaces están presentes: uno principal y dos secundarios. Cada contenedor tiene su propio proceso de inicio (PID 0) y otros procesos con PIDs que pueden repetirse, pero no son únicos globalmente. El namespace principal ve todos los procesos de los hijos, mapeando sus PIDs a un rango distinto. Aunque hay 9 procesos en total, se requieren 15 PIDs para representarlos, ya que un proceso puede estar asociado a múltiples PIDs según el contexto.

Namespaces también pueden ser no jerárquicos, como el namespace UTS, que no tiene conexión entre parent y child namespaces.



Nuevos namespaces se pueden crear de dos formas:

1. Al crear un nuevo proceso con las llamadas al sistema **fork** o **clone**, donde se puede especificar si los namespaces serán compartidos con el proceso padre o si se crearán nuevos.
2. La llamada al sistema **unshare** separa partes de un proceso del padre, incluyendo namespaces. Para más detalles, consulta la página de manual

unshare(2).

Una vez que un proceso se ha desconectado del namespace del padre usando alguno de los dos mecanismos anteriores, cualquier cambio en una propiedad global, desde su punto de vista, no se propagará al namespace del padre, ni los cambios en el padre afectarán al hijo, al menos en términos simples. La situación es más compleja con los sistemas de archivos, donde los mecanismos de compartición permiten muchas posibilidades.

Los namespaces siguen siendo experimentales en el kernel estándar, y el desarrollo para hacer que todas las partes del kernel sean completamente conscientes de namespaces sigue en proceso.

UTS Namespace

Un **UTS namespace** (Unix Timesharing System) es un tipo de namespace en Linux que aísla dos propiedades del sistema: el **hostname** (nombre del sistema) y el **NIS domain name**. Básicamente, permite que diferentes contenedores o procesos dentro de un sistema tengan su propio hostname y nombre de dominio, aunque todos estén ejecutándose en el mismo kernel.

Esto es útil en entornos donde se quiere que cada contenedor parezca ser una máquina independiente, con su propio nombre de sistema, sin que los cambios en un contenedor afecten a los demás.

User Namespace

El **user namespace** en Linux permite la separación de identificadores de usuarios (UIDs) y grupos (GIDs) entre diferentes namespaces. Con un user namespace, un proceso dentro del namespace puede tener privilegios elevados (como ser root) sin tener esos privilegios en el sistema anfitrión. Es decir, puedes tener un proceso que tenga UID 0 (root) dentro del user namespace, pero fuera de ese namespace, sigue siendo un usuario sin permisos especiales.

Esto permite una mayor seguridad y flexibilidad, ya que puedes ejecutar contenedores o procesos con permisos de superusuario dentro de un entorno aislado, pero sin otorgarles permisos en el sistema anfitrión. Los **UIDs** y **GIDs** dentro del user namespace pueden ser mapeados a diferentes valores en el sistema anfitrión, creando una separación efectiva de privilegios.

Control Groups

Scheduling Domains y **Control Groups** permiten organizar la asignación de recursos de manera más eficiente en el sistema Linux. En lugar de que el **scheduler** maneje directamente los procesos, trabaja con **entidades programables**, lo que facilita la planificación en grupos.

Con **group scheduling**, los procesos se agrupan, y el scheduler asigna el tiempo de CPU de forma justa entre los grupos primero y luego entre los procesos de cada grupo. Por ejemplo, se puede garantizar que cada usuario reciba una porción equitativa de tiempo de CPU. Si un usuario tiene más procesos, esos procesos compartirán su tiempo asignado, pero el total de CPU que recibe ese usuario no cambia por la cantidad de procesos que ejecuta.

Además de agrupar por usuarios, el kernel permite usar **control groups (cgroups)**, un sistema que crea colecciones arbitrarias de tareas que pueden

organizarse en múltiples jerarquías. Los **cgroups** permiten gestionar los recursos de manera personalizada para diferentes tareas o procesos, independientemente de los usuarios a los que pertenezcan.