

Sistemas Operativos

01/10/2020

Tema1: Introducción

- Sistemas concurrentes
- Breve historia de los SO
- Arquitecturas de los SO

Sistemas de Tiempo Compartido (diapositiva)

Sistema Concurrente es aquel sistema software en el cual existen diversas actividades separadas en progreso en el mismo instante.

Algunos sistemas concurrentes: sistemas operativos, entornos de ventanas, sistemas de control de tráfico aéreo, etc.

El SO consiste en un conjunto de rutinas que son necesarias y utilizadas por todos los programas en ejecuciones, esas rutinas tienen que estar en memoria principal.

Para que funcione un SO tiene que tener su parte hardware para realizar la entrada y salida además de los lenguajes de programación para crear los sistemas concurrentes que manejan las tareas que ocurran en el SO. El SO le da al programador una interfaz con la que puede hacer uso de él.

Pregunta:Cuál es la interfaz que proporciona el hardware al SO?

El SO es un programa que necesita para ejecutarse los recursos del hardware. Una de las formas de hacerlo es a través de interrupciones del sistema hardware de manera que se ponga en marcha el procesador.

Por ejemplo, cuando se pulsa una tecla, se pone en alza una parte del procesador que accede a una zona de memoria y activa un RST (Rutina de servicio de interrupción) que detecta esta interrupción y muestra su contenido por pantalla.

Otra de las formas que tiene el SO para interaccionar con el hardware se hace mediante el lenguaje máquina (instrucciones que puedo ejecutar en el procesador).

Otro sistema concurrente es el sistema de comunicación de datos y redes, donde una máquina está conectada a una red y necesita software. Además el sistema de tiempo real interacciona con el entorno físico y responden a los estímulos del entorno.

Todos los SO hoy en día son multiprogramados.

Pregunta: Diferencia entre un sistema multiprogramado y un sistema monoprogramado?

Un sistema multiprogramado puede ejecutar 2 o más procesos a la vez. Es decir, que el sistema operativo soporta tener en memoria principal varios programas que van avanzando en su ejecución. Cuando eran monoprogramados, el SO solo ejecutaba un solo programa en memoria pral.

Breve historia de los SO

En un principio, el SO era sólo un fragmento de código que se enlazaba con los programas, se cargaba todo en memoria, y se ejecutaba con el programa - como una biblioteca en tiempo de ejecución. Problema: uso ineficiente de recursos caros (baja utilización de la CPU) ya que el tiempo de preparación de una tarea era significativo

- Los sistemas por lotes:
 - ❖ Los sistemas batch fueron los primeros SO's reales: El monitor estaba almacenado en memoria. Cargaba un trabajo en memoria (tarjetas). Ejecutaba el trabajo. Cargaba el siguiente. Las tarjetas de control en el archivo de entrada indican qué hacer
El problema principal se debía a las largas esperas entre lotes de trabajos. La 2ª generación de ordenadores introduce: Hardware separado para gestionar las E/S Aparece del concepto interrupción un programa puede seguir ejecutándose mientras se está realizando una E/S. Nueva dificultad: ¿cómo gestionar la concurrencia? los SOs multiprogramados.
 - ❖ **Los sistemas multiprogramados optimizan la productividad (throughput) del sistema:** permiten varios programas en ejecución simultáneamente, todo ellos cargados en memoria pral y cuando la cpu se queda libre se asigna a otro programa. Esto ocurre con la segunda generación de ordenadores, donde se introdujo el concepto de interrupción.
Se benefician de que los dispositivos de E/S pueden operar asíncronamente. Mantienen varios trabajos ejecutables cargados en memoria pral. Solapan procesamiento de E/S de un trabajo con cómputo de otro. Necesitan interrupciones, o DMA.
Soporte para la multiprogramación: El SO debe suministrar las rutinas de E/S. Debe existir una gestión de memoria para poder asignar y controlar la memoria repartida entre varios trabajos. Debe existir una planificación de la CPU: el SO elige uno de los trabajos listos para ejecutarse que hay en memoria. El SO realiza la asignación de los dispositivos a los trabajos.

Pregunta: Interrupción y cuando ocurre?

Una interrupción es una llamada al sistema para que haga determinadas cosas. Otro tipo de llamada es cuando un programa necesita algún dato del disco, entonces se lanza una petición al disco y el sistema operativo para el programa ha estado bloqueado esperando que lleguen los datos y (en un sistema multiprogramado) pasa otro programa a ejecutarse. Ocurre una interrupción cuando el disco tiene los archivos y entonces el SO para el programa está preparado para volverse a ejecutar.

Asíncrona

La CPU está ejecutando un programa, este programa realiza una petición mientras que la CPU sigue activa pero ejecutando otro programa. Cuando llega la petición, la CPU hace la rutina del proceso de interrupción.

Síncrona

Como vemos en la imagen en la segunda parte la CPU está esperando hasta que el sistema realiza una petición y le llega, y comienza a ejecutar el programa. El programa por ejemplo se comporta de forma síncrona porque tiene que estar esperando. Para poder solucionar esto, los programadores crean programas concurrentes con hilos.



Soporte para la Multiprogramación

- El SO debe suministrar las rutinas de E/S
- Debe existir una gestión de memoria para poder asignar y controlar la memoria entre los procesos.
- Debe existir una planificación de CPU, es decir, como va repartiendo y a que le asigna la CPU.
- El SO realiza la asignación de los dispositivos.

Pregunta:Cuál es la misión de un SO?

Gestión de tareas con su planificación y compartir recursos. Además crea una abstracción de todo el sistema facilitando la vida al programador.

Pregunta: ¿Tiene que incorporar un sistema multiprogramado para además dar soporte a la interacción en tiempo compartido?

Tiene que implementar un quantum de tiempo. El SO va a diseñar la planificación de los programas en base a un tiempo en la que los programas comparten la CPU. El SO le da la CPU a otro programa cuando se produce una interrupción por espera de datos o cuando se termina ese quantum de tiempo.

Un **sistema operativo multiprogramado** deja de ejecutar un programa y pasa a otro cuando un programa se bloquea porque necesita unos datos

Un **sistema operativo de tiempo compartido** no necesita que un programa se bloquee, se le quita la cpu a un programa que ha cumplido su quantum de tiempo y se lo da a otro programa.

Sistemas operativos de tiempo real : ciertas tareas tienen más prioridad (cosa q el sistema no puede cambiar, en todo caso, el programador creador) que otras, aunque se hayan ejecutados más veces, las tareas van a sacar una salida en un tiempo planificado con una cota de tiempo conocida.

- suelen usarse en aplicaciones especializadas, por ejemplo sistemas de control
- garantiza la respuesta a sucesos físicos en intervalos de tiempo fijos
- problema: poder ejecutar las actividades del sistema con el fin de satisfacer todos los requisitos críticos de las mimas

El objetivo de estos sistemas no es que todos los procesos avancen por igual sino que todos hagan sus tareas en el tiempo estimado. Para poder hacer esto hay un proceso de planificación con herramientas que hace que, en el peor de los casos, las salidas siempre se hagan en el tiempo estimado, y no en cuestiones de que la cpu dedique el mismo tiempo a todos los procesos.

*linux es el origen de los sistemas operativos de tiempo real, porque es de código libre, por tanto muchos so de tiempo real son variantes de linux

Pregunta: cuál es la pral modificación que se tiene que incorporar a un so multiprogramado para que realmente se convierta en so de tiempo real?

Tiene que ver con cómo se gestionan las prioridades de los procesos. Uno de los requisitos típicos es que la prioridad de los procesos suele ser estática, no se puede cambiar dinámicamente.

Los sistemas de tiempo real lo importante es garantizar que si todos los programas necesitan entrar en funcionamiento, las respuestas de cada uno de ellos ocurren en los tiempos acotados que tienen cada uno.

No todos los so dan soporte a todas las características propias de los SO en tiempo real. Algunos so que tienen programas de root y programas de usuario, se dice que tienen características de “tiempo real”, es decir soporte de tiempo real blando, para diferenciarlos de los otros.

Sistemas Distribuidos

Es un conjunto de máquinas que se gestionan como un todo. Es decir, es un SO para varias máquinas, de manera que si una máquina está ocupada, lo puede pasar a otro para realizar una tarea.

Ventajas

- Aumento de fiabilidad del sistema, ya que si una máquina falla, otras pueden hacerse cargo.

Inconvenientes.

- SI había muchas comunicaciones al final esos sistemas caían y dejaban de funcionar
- Complejo de programar.

Pregunta: Sustituto actual de los sistemas distribuidos?

Servidores en la nube.

Sistemas paralelos

Es una máquina con varios procesadores que comparten una única memoria y un reloj. Estos sistemas se diseñan considerando que tiene varias CPUs, entonces cambia bastante el SO

El objetivo principal es aumentar la velocidad de tareas complejas. De esta manera podemos asignar permanentemente una tarea a una CPU.

Hay dos tipos de multiprocesamiento:

- **Simétrico:** es el más usado. Todos los procesadores son capaces de ejecutar cualquier tarea incluido el código del propio SO. Es el más potente ya que dinámicamente intenta aprovechar todos los recursos.
- **Asimétricos:** hay un proceso dedicado exclusivamente a una única cosa. Ej: hay un procesador dedicado exclusivamente a ejecutar núcleo del SO y los demás procesadores hacen otras cosas. Es más sencillo y estable, por lo tanto es menos propenso a errores.

Sistemas Operativos de internet

Suministran la funcionalidad mínima para arrancar una máquina y lanzar un navegador el cual cargará los applets que suministran la funcionalidad necesaria. El problema es que no son mucho más baratos que los SO completos así que no tiene mucho mercado.

****repaso:**

Un SO tiene también necesidades de llevar a cabo varias funcionalidades y esas funcionalidades se traducen en que el SO tiene que tener diferentes componentes para gestionar procesos, memoria, almacenamiento permanente... Entonces para tener ese componente dentro del SO hay diferentes alternativas de organización, de esta forma se definen las diferentes arquitecturas de un SO

El SO se ejecuta el 80% propias acciones del SO el resto del tiempo la CPU está ejecutando los programas de usuario, esto se traduce en que el SO está continuamente ejecutando servicios

Cómo conseguimos que un SO sea más eficiente??

- eficiencia, menor uso de la CPU, como? realizando sus funciones con menos instrucciones máquina (con menos código, porque está todo muy optimizado) de hecho hoy en día la gente que sabe de eso lo primero que miran es las líneas de código que tienen
- fiabilidad: se refiere a dos aspectos diferentes, robustez (SO debe responder de forma predecible a condiciones de error, incluidos fallos de hardware y el SO debe protegerse activamente a sí mismo y a los usuarios de acciones accidentales o malintencionadas).
- extensibilidad: el SO sea extensible a nuevos hardware y nuevas aplicaciones

Estructura Sistemas Operativos

El primer diseño que ocurre de un software es la arquitectura. Es importante ya que determina qué propiedades o capacidades de partida va a tener el sistema.

La estructuras normalmente se diseñan en base a componentes que tienen que:

- Ejecutar diversos programas
- Tiene que tener acceso a la información de forma permanente,
- Tiene que tener acceso a los dispositivos periféricos
- Utilizar todo de manera segura y fiable.
- Permita al usuario interacción con el sistema

El SO se encarga de crear procesos como una instancia de un programa en ejecución.

Pregunta: La CPU sabe cuántos procesos hay?

No, porque la CPU solo conoce instrucciones de memoria y no sabe si esta interrupción pertenece a un programa o a otro. Quien realmente conoce esto es el propio SO.

- **Memoria pral:** un proceso no puede acceder a la memoria de otro. Cuando un proceso termina, tiene que liberar la memoria para que pueda acceder a otro. La CPU nuevamente no lo sabe, entonces el propio núcleo tiene que guardar la memoria pral que está libre y la que está ocupada.
- **Gestión de archivos** donde un archivo es una colección de información con un nombre. Nuevamente este sistema de archivos lo gestiona el SO.
- **Gestión E/S** lo lleva le SO. Los SO ofrecen a los programas una interfaz estándar de los dispositivos, es decir, utilizan las misma funciones independientemente del dispositivo al que acceden. El manejador de dispositivo es un módulo que gestiona un tipo de dispositivo.
- **Sistemas de protección** donde el sistema operativo especifica si el uso de algo esta permitido o no, especificar que contr se debe imponer y los medios
- **Intérpretes de órdenes** es un programa o procesos que maneja la interpretación de la órdenes que da el usuario desde la terminal o desde un archivo para acceder a recursos.

Un sistema operativo tiene que ser:

- **Eficiente:** debe consumir pocos ciclos de reloj. Teniendo en cuenta que ya de por si la CPU consume muchos ciclos de reloj. La mayor parte del tiempo el sistema operativo se ejecuta dando servicio a los que necesita el programa.
- **Fiable:**
 - **Robusto:** debe responder de forma predecible a condiciones de error.
 - **Protección:** el SO tiene que protegerse activamente a si mismo y a los usuarios, de acciones accidentales.
- **Extensibilidad:** la aparición constante de nuevo hardware y de nuevos tipos de aplicaciones exigen al SO la adición de nueva funcionalidad. En vez de construir un nuevo SO lo que hacemos es construir un sistema cuya funcionalidad pueda variar o crecer de forma sencilla.

Pregunta: Cómo sabemos que un programa es eficiente?

El primer parámetro es haciendo menos uso de la CPU y se consigue utilizando menos instrucciones pero haciendo la misma funcionalidad estando todo optimizado.

Pregunta: Que un programa sea eficiente quiere decir que también sea seguro?

No tiene por qué, de hecho estas dos propiedades son conflictivas. Si quiere añadir seguridad a un sistema, es una funcionalidad más, ya que hay que grabar cierta información, hacer comprobaciones, encriptar datos, etc

Pregunta: Diferencia de ejecución en modo usuario y en modo kernel?

En modo kernel se pueden ejecutar cualquier instrucción máquina y no hay limitaciones tanto instrucciones privilegiadas como no privilegiadas. En modo usuario solo se pueden ejecutar las instrucciones no privilegiadas, si intenta ejecutar una instrucción privilegiada el propio sistema salta con una excepción.

Pregunta: Tiene que ver el modo de funcionamiento del procesador (en modo usuario y modo kernel) con que un usuario sea un super usuario (root) o usuario normal?

No, el shell sea quien sea el usuario se ejecuta en modo usuario, los dos piden al kernel pero un usuario normal sin permisos el sistema le niega la acción, en cambio si lo ha hecho un shell como root el SO le deja porque tiene permisos suficientes

Pregunta: El compilador que viene es el núcleo del SO?

El compilador no es núcleo de un SO, es una utilidad más en modo usuario del SO

El kernel es la parte del SO es la parte esencial del so que siempre debería estar residiendo en memoria física, porque proporciona la funcionalidad básica y fundamental para que todo el So esté funcionando, y esa parte del So es la que se ejecuta en modo kernel.

Pregunta: Qué es el SO?

Cuando mi programa puede invocar una función y saltar a otra función del SO, en que se traduce? ¿por qué puedo hacerlo? Porque las funciones kernel están dentro del mismo espacio de direcciones, el kernel está mapeado en el mismo espacio de direcciones que el programa

Cuando llamamos a open, llamamos al sistema operativo mediante el kernel para proteger el SO.

Suponemos una arquitectura de 32 bits, la palabra 32 bits, ¿cuántas posiciones de memoria se puede direccionar con 32 bits ? $2^{32} = 4\text{GB}$ y pico de posiciones diferentes de memoria. Estos sistemas hacen lo siguiente, si la primera pos corresponde a 0 y la última a 4 GB, la memoria está dividida en dos entre modo usuario y modo kernel.

El núcleo es parte de todos los programas que se ejecutan, está dentro del espacio de direcciones de todos los programas.

Estructura Monolítica

Los componentes del SO se ejecutan en modo kernel; la relación entre ellos es compleja.

El modelo de obtención de servicios es la llamada a procedimiento.

Es la primera arquitectura y se sigue usando hoy en día de alguna forma. Está escrita en C y es procedural (son un conjunto de instrucciones en funciones). Es básica y los componentes no están muy diferenciados.

El modelo es un programa de usuario que realiza peticiones al SO haciendo **llamadas a las funciones**. Esas funciones que ofrece el SO son las APIs del sistema operativo y se conocen como **llamadas al sistema**.

Para que nuestros programas de usuario puedan llamar a una función que ofrece el sistema operativo debe poder trabajar en modo dual, o sea modo usuario y modo supervisor (kernel). Además cuando se produce una llamada al sistema se produce una interrupción especial llamada **instrucción de interrupción o trap** a través del hardware que controla que se está invocando al kernel del SO.

El hardware controla que al kernel se entre por un único punto

¿Qué capacidad hardware hay actualmente para que soporten los SO de forma segura? el soporte para modo dual, modo usuario y modo supervisor (modo kernel)

Diferencia entre modo usuario y modo kernel? ¿qué se puede ejecutar en modo kernel que no se pueda ejecutar en modo usuario? en modo supervisor se puede ejecutar cualquier instrucción máquina (instrucciones privilegiadas); en modo usuario solo se pueden ejecutar instrucciones no privilegiadas. Si el modo usuario intenta ejecutar inst privi, el hardware reacciona.

Problemas de los sistemas monolíticos

- Son difíciles de comprender, ya que son un único programa, por tanto, difíciles de modificar (cuántos MB ocupa el núcleo actual de Linux, rondando los 90MB (en líneas de código máquina, pueden ser entre 10 o 20 millones de líneas de código); el de windows, 200 o 300 MB)
- Debido al problema anterior, estos son difíciles de modificar y mantener ya que hay cientos de MB.
- No son muy fiables, no hay una división clara, el núcleo es muy poco estable y propenso a errores.

Arquitectura Monolítica. Es un modelo en el cual el sistema operativo y todos los servicios fundamentales residen en un monitor monolítico que se accede a través de un mecanismo de llamada al núcleo.

Características

- Es un intento de paliar los problemas de la arquitectura plana.
- Su aportación estriba en que los procesos de usuario ejecutan en espacios de direccionamiento diferentes al del sistema operativo.
- Las implementaciones de UNIX han respondido tradicionalmente a este diseño.
- Se aíslan del sistema de los errores de los procesos de usuario, pero nuevos dispositivos aparecen en el mercado continuamente y es preciso escribir manejadores para soportarlos. De nuevo el sistema crece y la probabilidad del fallo aumenta.
- El programa de usuario lleva a cabo las llamadas al sistema mediante interrupciones software o traps.
- Sólo en modo supervisor se permite al procesador ejecutar instrucciones privilegiadas. Accesos a las posiciones de memoria asignadas a los adaptadores de dispositivo .
- Copia de datos entre espacios de direccionamiento diferentes.

Sistemas de micronúcleo. Posibilidad de separar funcionalmente los programas del sistema de los programas de aplicación y asegurar protección adicional con el hardware, origina otra arquitectura, la de Micronúcleo.

Características

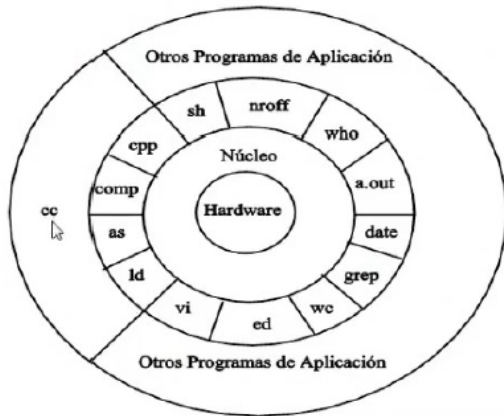
En esta, se trata de combinar el rendimiento y sencillez de la Arquitectura monolítica con la protección y organización de la arquitectura por capas. La idea fundamental es obtener un Núcleo lo más pequeño y rápido posible y tratar el resto de las funciones y componentes como procesos de aplicación.

En esta nueva concepción, es usual que el núcleo solo contenga lo necesario para la gestión de memoria y procesos. Todo el resto se ejecuta con el nivel de privilegios de las aplicaciones que no son del sistema.

En la práctica es un poco difícil conseguir esto sin una pérdida apreciable de rendimiento y en alguna medida deben incluirse en el núcleo otras funciones como el manejo de hardware y algunos Drivers

Estructura Sistema UNIX

En esta estructura, el núcleo rodea al hardware y es por esto que cualquier programa que forme parte del sistema puede acceder al hardware, porque el acceso lo gestiona el núcleo del SO. El SO interacciona directamente con el hardware, proporcionando servicios comunes a los programas.



La capa de control hardware se suele nombrar como capa hal (capa de abstracción sobre hardware, en español) última capa del nivel del núcleo pegando al nivel hardware

En Unix el paso a los controladores de dispositivos se hace a través del Sistema de Archivos

Arquitectura UNIX

Los programas que se encuentran por encima del “Nivel de usuario” se ejecutan en modo usuario y lo que está por debajo en modo kernel.

Cuando un programa de usuario llama al SO para pedirle un servicio ocurre el mecanismo de llamada al sistema que produce una interrupción que captura el procesador y se lo pasa al kernel.

La interfaz es el primer punto de entrada al SO y lo primero que mira es a quien están llamando y si los parámetros son correctos. Ahora, como podemos ver en el esquema, desde la interfaz puede acceder perfectamente a las dos primeras capas, pero si quiere acceder a la capa de control de dispositivos, tiene que pasar por el Subsistema de Archivos, porque en Unix todo es archivo. Esto deriva a uno de estos 3 componentes:

- Subsistema de archivos: gestión de archivos de forma permanente.
- Subsistema de control de procesos: se encarga de gestionar programas en ejecución.
 - Gestión de memoria.
- Controlador de dispositivos: gestión de entrada..

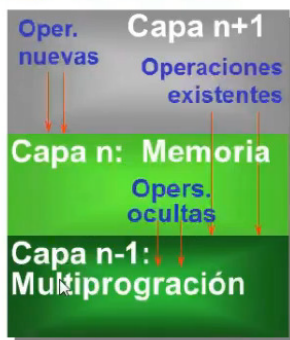
Después tenemos una capa que se ejecuta en modo kernel, es la capa “Control de Hardware” (Capa Hal→ capa de abstracción sobre el hardware) que es la capa que se llama por el resto de los módulos para acceder al hardware. Esta capa se encuentra para dar abstracción al sistema. De manera que la capa hardware es una capa de alto nivel y va accediendo a la capa de bajo nivel en “Hardware”

En Linux se crea esta capa para dar fiabilidad de manera que si cojo este kernel y me lo llevo a otra arquitectura, lo único que tengo que cambiar es el código de esa capa. Y por eso Unix, y en concreto, Linux, tuvo tanta fama.

Un ejemplo de la llama la sistema, cuando yo quiero formatear el sistema tengo que pasarle la instrucción y después el archivo sda. Por eso cuando usamos los controladores de dispositivos debemos pasar por el subsistema de archivos.

Arquitectura de Capas

Capa en desarrollo



Esta arquitectura consiste en programar el SO ordenado por capas muy bien implementadas. Cada capa debería ser desarrollada, codificada y compilada independientemente del resto de capas.

En esta arquitectura existe una sobrecarga de comunicaciones entre las distintas capas, ya que una función que está en una capa puede llamar a otra de una capa inferior, y sucesivamente.

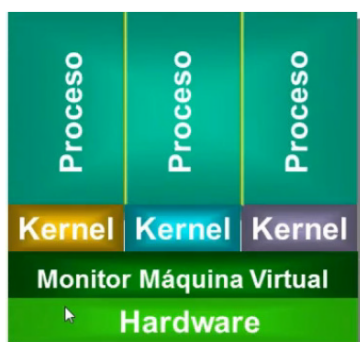
Por modularidad, las capas se seleccionan para que cada capa utilice sólo las funciones de las capas inferiores, de manera que haya mayor abstracción arriba.

La arquitectura que hemos visto anteriormente de UNIX es una arquitectura de capas, lo que pasa es que no se cumple totalmente ya que una capa no está generada ni esperada de otra, ya que como vemos en la imagen, las flechas no van solo de arriba abajo.

El mayor problema que tiene esta arquitectura:

- Cuando hay una abstracción tan fuerte, al final las capas no se acaban respetando y hay que fusionarlas y de esta manera se tiene un único componente. Claro el problema es que al final quedan cosas muy grandes y esto no funciona.

Arquitectura de máquina virtual



Una máquina virtual es una réplica del hardware que crea una ilusión del sistema (copia del sistema) que puede ejecutar cualquier SO. El SO gestiona a todos los sistemas operativos y estos sistemas se gestionan en su módulo.

El “Monitor maquina virtual” (hipervisor) tiene que replicar el hardware para cada máquina y además tiene que gestionarlas porque solo hay una CPU que se tiene que repartir entre esos diferentes kernel. Y esos kernel reparten lo que le dejan entrar sus procesos.

Esta arquitectura intrínsecamente implementa una estructura por capas como se puede ver en la imagen.

Características de MV

- Cuanto **mayor aislamiento entre máquinas mejor**, asegura la protección de los recursos del sistema.
- Sirve para investigar/desarrollar SOs: no interrumpe el funcionamiento del sistema en su ejecución.
- Permite la ejecución de aplicaciones desarrolladas por otros SOs.

Características NO deseable:

- Como las máquinas están aisladas, la compartición no es fácil.
- Son difíciles de implementar perfectamente debido a la complejidad de crear un duplicado exacto del hardware. Ya que hay veces que un proceso en un SO puede poner al SO en un modo distinto, entonces no puede crear un réplica exacta ya que no se puede dar los recursos hardware que pide el kernel.

clase 15/10/20

Arquitectura Microkernel



Esta arquitectura se denomina microkernel ya que muchas de las funciones importantes del SO se van a intentar sacar del kernel, como si fueran programas de usuario con lo cual estaríamos reduciendo el kernel todo lo posible.

Este kernel tiene que tener:

- Control de Hardware (HAL).
- Rutina de gestión de interrupciones.
- Cuestiones básicas de planificación de procesos y comunicación.

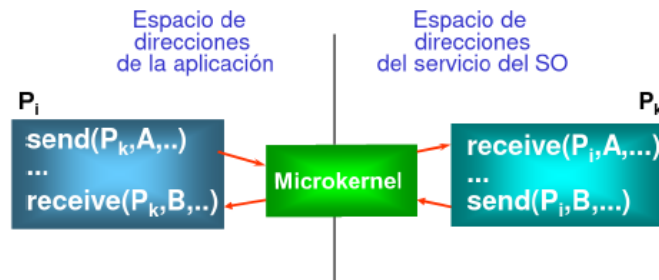
Importante: El microkernel prácticamente es un canal de comunicación, básicamente un programa llama a la memoria pral, mi petición (cliente) la tiene que trasladar al servicio del SO (servidor) y su respuesta la tiene que pasar por el canal de comunicación. Toda esta comunicación la sincroniza el microkernel porque todos los recursos están abstraídos para que no se corrompa.

El microkernel sirve de canal de comunicación.

Ventajas

- **Más fiable**, ya que el kernel se reduce por tanto es más probable que no haya errores escondidos entre el código (se reduce su tamaño y su código).
- **Más extensible y personalizable**, si queremos añadir más funcionalidad simplemente creamos nuevos procesos donde estarán las funcionalidades.
- Estas **funcionalidades se ejecutarán en modo usuario** con lo cual no sobrecargan a la máquina porque no están en modo kernel.

El **modelo de comunicación** que utiliza es el **paso de mensajes**:



1. La aplicación envía el mensaje de solicitud, **send(P,A)**, y espera una respuesta,
2. El kernel verifica el mensaje y lo entrega al servidor.
3. El servidor que esta en espera, realiza el servicio y devuelve el resultado con **receive(P,b)**.

Las **desventajas** que tienen son las siguientes:

- Tiene peor rendimiento ya que tiene una arquitectura monolítica de manera que para obtener un servicio se realizan muchos cambios de modo y espacio de direcciones.
- La estructura ukernel es mas flexible que la monolítica



Pregunta:Cuál es el único costo adicional para saltar a una función que esté dentro del kernel dentro de la arquitectura monolítica?

El coste adicional consiste en que se producen cambios de modo, que lo produce es una instrucción de máquina llamada trap que cuando el procesador pasa a ejecutarla, realiza el cambio de modo, Es es una interrupción que que salta a ejecutar una rutina del núcleo pero en vez de ser una interrupción del hardware, es el software.

Cuando termina la instrucción que se ha pedido, se regresa a modo usuario por lo que habría otro cambio.

En total han ocurrido dos cambios de modo y eso se ve como un coste adicional.

¿Cuántos cambios de modo de procesador son necesarios? 2?

Supongamos que usuario invoca a `open(fichero)`, implementado como otro proceso

¿Cuántos cambios de modo se hacen? 4 : programa usuario a kernel, kernel deriva los mensajes a los servicios que tiene que dar respuestas, cuando se ejecuta , devuelve una respuesta al kernel, y microkernel traslada el mensaje de vuelta al usuario.

+ 2 cambios de contexto

ipc: inter process communication

Pregunta: ¿Cuántos cambios de modo son necesarios en un SO microkernel?

Ahí ocurren **4 cambios** ya que el programa usuario envía el mensaje al kernel y deriva el mensaje al servicio del SO que realiza la petición, la respuesta se la pasa nuevamente al kernel y este al programa de usuario.

Además hay una carga añadida ya que en la carga microkernel, el programa de usuario y los servicios del SO no están en el mismo espacio de direcciones y por tanto cuando el programa de usuario pide algo, mientras le llega, se detiene y el kernel activa el servicio del sistema y por tanto ahí se produce un cambio de proceso.

Este cambio de proceso tiene un coste mucho mayor que un cambio de procesador. Por tanto, además de tener 2 cambios más que el sistema monolítico, tiene el cambio de proceso.

En SO microkernel, el kernel está dentro del espacio de direcciones del servicio del SO que se ejecuta en modo usuario, el microkernel sirve de intersección entre todos los programas que se ejecutan, pero el espacio de direcciones virtual no es el mismo el del micro y el SO. Esto implica que el programa usuario cuando se está ejecutando y tiene que pedir algo y tiene que esperar, se detendrá y para que se realice el programa SO(que también incluye microkernel) tomará el control. Esto, este cambio de proceso, lleva un coste que es bastante grande.

En el monolítico no hay cambio de contexto

Cuando se produce un salto al SO porque se le ha pedido un servicio, en principio la mayoría de autores entienden que ahí no ocurre un cambio de contexto excepto en el tema de que el contexto hardware sí que cambia (contador de programas de la CPU), pero eso es solo una parte del proceso.

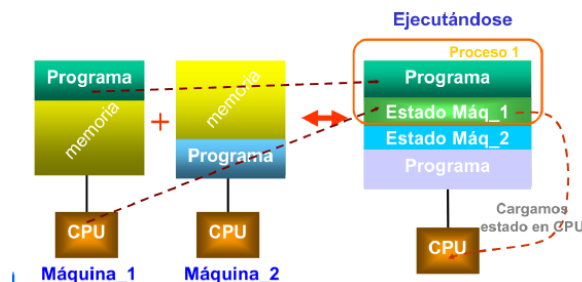
X La estructura microkernel tiene peor rendimiento que la monolítica; para obtener un servicio se realizan más cambios de modo y espacio de direcciones

V La estructura microkernel es más flexible que la monolítica

Tema 2: Proceso y hebras

1. Diseño

Idea de proceso



Si queremos ejecutar en una misma máquina mas de un proceso, entonces tenemos que ir dando paso a los programas. Osea consiste en multiplexar el uso de la CPU dentro de una misma máquina.

Ambos procesos tienen que residir **en la memoria** pero en principio la CPU es única a no ser que sea multiprocesador o un procesador con varios núcleos. Para poder cambiar de un proceso a otro, tiene que haber un contexto hardware con los datos del programa, si en un momento dado se detiene, y le da la CPU a otro programa.

Parte de un proceso.

Partes de un programa ejecutable:

- Conjunto de **cabeceras**: describen los contenidos de ese archivo ejecutable
- Texto de programa
- Datos inicializados y no inicializados (bss)
- Otras secciones (ej tabla de símbolos)

Pregunta: Porque hay secciones para datos inicializados y no inicializados?

Porque si por ejemplo tenemos 10 millones de enteros en el archivo ejecutable no debería estar reservado el espacio sino que se irá creando a medida que sea necesario, ya que es una tontería reservar un espacio que no sirve para nada. Se llama bss porque se guarda una especificación de esa estructura pero no guarda esas posiciones.

El núcleo divide el proceso en regiones (segmentos). Esas regiones se conoce como la técnica de segmentación:

- región de **texto**: solo lectura
- región de **datos**: se pueden leer y escribir datos
- región de **pila**: no aparece en el ejecutable pues solo tiene sentido en ejecución cuando se carga el ejecutable, se reserva un espacio para la gestión de pila.

Pila de usuario

var. locales	
dir. marco 2	
dir. retorno	
parámetros	marco 3
var. locales	f2()
dir. marco 1	
dir. retorno	
parámetros	marco 2
var. locales	f1()
dir. marco 0	
dir. retorno	
parámetros	marco 1
var. entorno	main()
línea órdenes	marco 0
	comienzo

La pila es una estructura de datos LIFO, guarda marcos de pila.
La pila de usuario se crea automáticamente y su tam se ajusta dinámicamente.

Consta de marcos de pila lógicos que se insertan cuando se llama a una función y se elimina cuando finaliza.

En el **espacio de usuario** de la imagen anterior tenemos:

- Código
- Datos en modo usuario
- Pila
- Marco de pila

En el otro espacio (**texto y datos del núcleo**) tenemos:

- Código del núcleo
- Cambios que suceden en el núcleo
- Estados maquina de los procesos: mantiene la estructura para que no se corrompan los datos

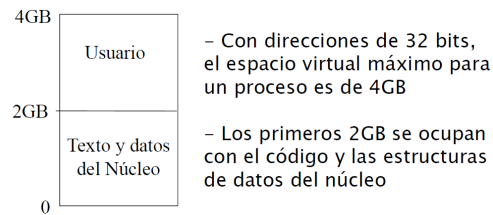
Como un proceso puede ejecutarse en modo supervisor y modo usuario, existe una pila para cada modo.

El retorno de una función conlleva la eliminación de un marco de pila, recuperas la dirección de retorno para continuar por el punto en el que se produjo el salto. Marco 0 para la propia línea de órdenes, que ha dado lugar a que se cargue el programa en sí, y las variables del SO. El marco1, por ejemplo en un programa c corresponden a la función main, donde empieza a ejecutarse, dentro ya habrá var locales, parámetros etc etc

¿Por qué hay secciones para datos inicializados y no inicializados? No tiene sentido reservar memoria en disco que no vas a usar por eso esa sección bss guarda una especificación de esas estructuras pero no guarda los bytes necesarios para albergarlos.

Diseño de Memoria

La estructura del espacio de memoria virtual de un proceso se divide en dos partes:



En usuario está nuestro código, datos en modo usuario, la pila

En texto y datos del núcleo tenemos:

- código del núcleo
- contexto hardware de los procesos

El núcleo necesita pila? Si, fundamentalmente porque el lenguaje de programación de programación para q se usa en los núcleos es c (lenguaje procedural, llamada a funciones) entonces la mayor parte del código fuente es, programado con funciones y por tanto necesita una pila. En las últimas versiones de SO, tienen partes de c++.

Si el núcleo estuviera implementado completamente en ensamblador no necesita pila.

Qué más cosas hay en un proceso?

Para representar un proceso debemos recoger toda la información que nos de el estado de ejecución de un programa:

- El códigos y datos del programa
- una pila de ejecución(tam, dond esta)
- el pc indicando la próxima instrucción a ejecutar
- los valores actuales del conjunto de registros de la CPU
- un conjunto de recursos de sistemas y su auditoría (memoria, archivos abiertos, E/S, etc..)

Clase 22/10/20

2.Diagrama de estados

Todos los **programas en ejecución son procesos** y todos los procesos, aparte el archivo ejecutable en disco, tienen otra información adicional para poder seguir ejecutando el proceso cuando esté disponible.

Varios procesos pueden ejecutar incluso el mismo programa pero cada uno tiene su propia representación.

Cada proceso está en un estado de ejecución que caracteriza lo que hace y determina las acciones que se pueden hacer sobre él.



Diagrama de estados



Los estados por lo que va pasando durante su ejecución **son preparados, bloqueados y ejecutándose.**

Después hay dos **pseudosestados** que son **nuevos y finalizados**, ya que estos estados son temporales. Son estados transitorios y están casi totalmente gestionados por el SO

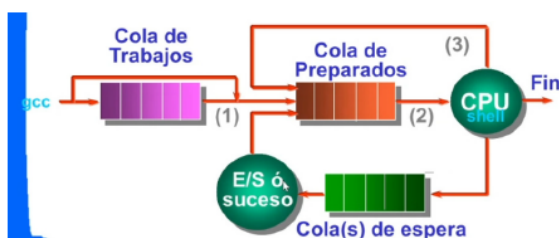
El estado de un proceso es parte del contexto de un proceso, en función del momento en el que se encuentra el proceso.

Ej: los procesos que están esperando una interrupción del disco, los ponen en la misma cola. Las colas se montan en la lista de procesos que mantiene el sistema. Una vez el proceso se bloquea, cuando el sistema le da paso, el proceso pasa de estar bloqueado a preparado y de ahí ya a ejecutándose.

Pregunta: ¿Porque al sistema le interesa esta clasificación en cuanto a bloqueado en diferentes colas?

Por motivos de eficiencia. Cuando ocurre un evento, el sistema no tiene que estar recorriendo todas las entradas para ver si ese evento afecta a algún proceso que se encuentra esperando. De esta manera, el sistema coge de la cola el proceso que le toca al evento que se ha disparado. De hecho, puede ser que quizás 10 ficheros esten esperando los mismos datos, de esta manera no tiene que estar buscando si no que simplemente desbloquea los procesos que quieren usar esos datos.

Cuando pasa de ejecutar un proceso a otro proceso se le llama switch content. El modelo de un sistema donde hay varias colas de trabajo que se dividen en cola de preparados a cola de espera que finalmente van a la CPU.



Como implementar esto el sistema?

PCBs y colas de estados

El so mantiene una cola de pcbs por estado, cada una de las cuales contiene los procesos que estan en ese estado

Modelos del sistema

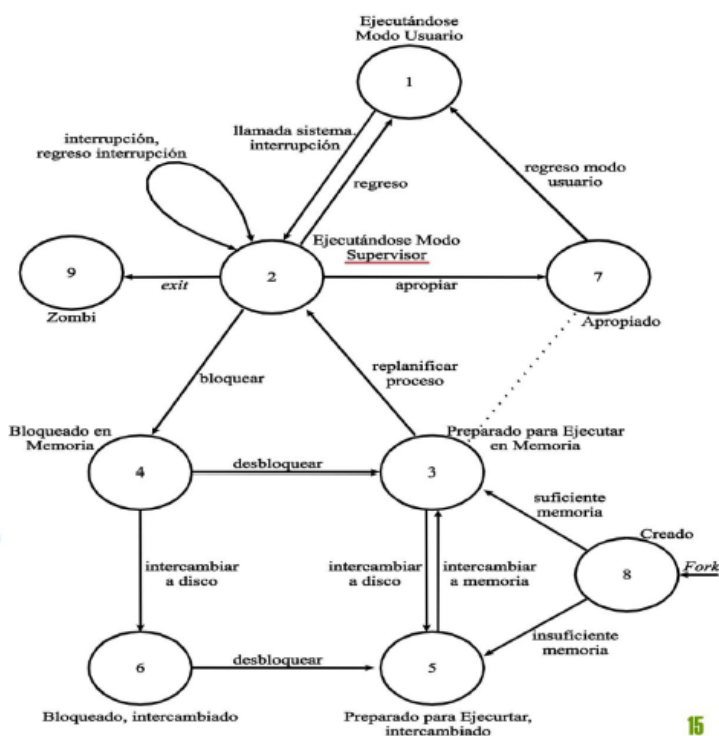
Podemos modelar el sistema como un conjunto de procesadores y de colas

gcc---> colas de trabajo---> cola de preparados---> cpu (shell)---> fin

----> colo en espera (e/s o un suceso)

3. Estados de UNIX

Estado /transiciones en Unix(copiar diapositiva 15, esquema)



La unión del estado **1 y 2**, es el estado ejecutándose que se divide en dos.

Para el sistema no es lo mismo que este ejecutándose en modo usuario que en modo supervisor.

El **3,5,7** son estados equivalentes al preparado, pero el sistema distingue entre ellos para separar preparado para ejecutándose, porque se le acaba el quantum, o preparado para ejecutarse pero que para ejecutarse necesita algo que no está en la memoria principal.

4: bloqueado en memoria

6: bloqueado, intercambio

9 estado zombie: proceso sin capacidad de ejecución (finalizado)

un proceso se crea, pasa por el pseudoestado de creándose (**8**), si hay suficiente recurso de memoria para ir a preparado(**3**), y si no la hay, pasa a dispositivos de intercambio (**5**).

Si ha pasado a preparado en memoria, se ejecuta cuando se le asigne la cpu, el proceso se ejecuta en modo supervisor porque el sistema al hacer la llamada fork para crear un proceso hijo que va a ejecutar otro programa, hace una copia del proceso padre, eso quiere decir que el contador del programa del hijo es el mismo que en el de padre.

Si para el sistema, el proceso padre estaba llamando a fork, como el proceso hijo es una copia del proceso padre y el contador del programa es igual al del padre, es como si el proceso hijo hubiese llamado al fork, cuando esto no es verdad porque no lo ha ejecutado

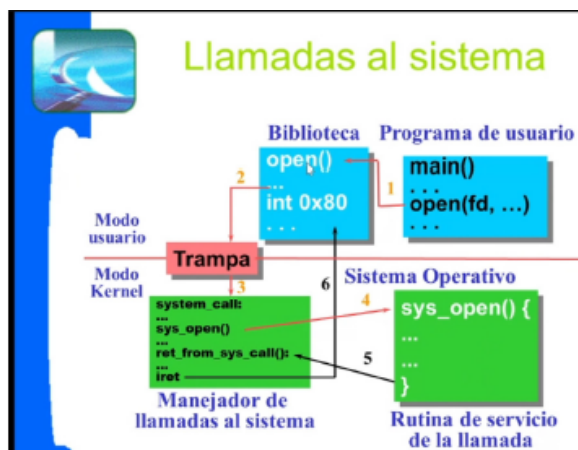
el. Y por eso el proceso hijo pasa a preparado y cuando pase a ejecución tiene que seguir ejecutando código de núcleo para terminar de ejecutar la llamada al sistema, que realmente no ha hecho él, y después ejecutar en modo usuario porque además el contador de programa tiene la misma instrucción que el proceso padre.

En definitiva, hacen todo igual menos que el proceso padre cuando todo ha ido bien, el fork le devolverá el PID del proceso hijo, pero este último, fork le devolverá un 0.

Mas cosas:

- Un proceso termina como consecuencia de una llamada al sistema (exit). El compilador justo al final del main añade ahí un exit, aunque no esté explícito en nuestro código.
- Los procesos bloqueados en memoria, son los primeros candidatos, al menos parte de ellos, de ir al dispositivo de intercambio.
- el estado 2 (ejecutándose en modo supervisor), el sistema distingue si un proceso se ejecuta en modo usuario o supervisor. porq la conexión en modo supervisor es alta, el que toma el control es el núcleo, ha cambiado el modo del procesador
- El estado 7 (apropiado) es similar al 3, eso quiere decir que el sistema tiene conocimiento sobre una proceso que ha pasado a preparado porque pidió un recurso y se bloqueó, después de ejecutarse en modo usuario ocurre la interrupción de reloj. Los apropiados solo les falta regresar a modo usuario para ejecutarse, esta para retorno de superior a usuario, pero ahí tomó el control la rutina de interrupción de reloj por eso solo les falta volver a modo usuario.

4.Llamadas al sistema



¿Cómo se programan las llamadas al sistema?

tenemos nuestro programa de usuario y dentro del main hay una llamada a una función, open (argm) por ejemplo, open si todo va bien, devuelve un entero y después habrá otras peticiones, hacer un read, copy etc etc

Al realizar un open, enlaza con la biblioteca del sistema, donde suele estar escrito en ensamblador el código que realmente implementa lo que es necesario para que

ocurran las llamadas al sistema. Para procesadores intel, se llama int 0x80 (puerta de entrada 80, hexadecimal) y esta instrucción es la que produce una interrupción y reacciona el procesador. El sistema se comporta como si fuera una interrupción hardware, la diferencia es que la interrupción viene producida por una interrupción software (otros nombres, instrucción trampa, de interrupción mediante software). Entonces el procesador cambia de modo usuario a modo kernel, tomando el control el núcleo, y se guardan los

valores de ciertas var en los registros donde corresponda; y pasa a ejecutar la primera instrucción que está especificada en el manejador de llamadas a sistema, entra en funcionamiento una rutina que hace que todas las llamadas del sistemas pasen por ahí. Desde el manejador se hace la llamada a la función correspondiente en el núcleo del SO

El procesador devuelve los resultados tras ejecutar la llamada y se retorna a la siguiente instrucción despues de open

Por qué existe la función implementada en el main que salta a la biblioteca y después se pasa previamente por un manejador sin especificarlo?

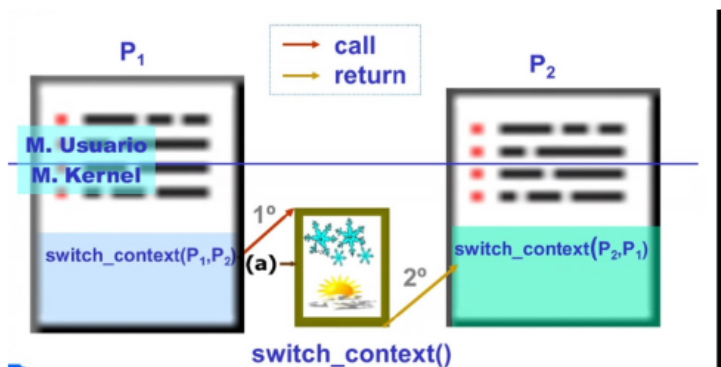
Por la fiabilidad, de forma que el programador no tenga que conocer los detalles del hardware subyacente para poder llamar a la función. Además es una forma de implementar abstracción en el SO y para facilitar la portabilidad del programa.

¿Por qué existe este manejador del sistema?

Para hacer las comprobaciones necesarias para que las llamadas al sistema cumplan con las cuestiones comunes básicas, y reducir el código del núcleo. De esta forma el código es más modular.

pagina 15 de los apuntes de blanca*****

5.Cambio de contexto



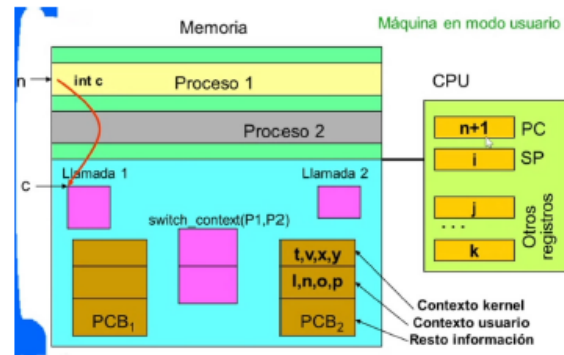
Supongamos que tenemos dos programas en ejecución, por tanto dos procesos, p1 y p2. El proceso p1 se está ejecutando en modo usuario y en algún momento salta a modo kernel por una llamada al sistema o una interrupción por reloj, y el sistema que ya está en modo kernel bloquea a p1 mediante la función de cambio de contexto, **switch_context()**, que pasa a ejecutar otro proceso.

La segunda parte de la función se encarga de activar el otro proceso que estaba esperando a que se reanude, en este caso a P2.

Es la única función del núcleo que se comporta así.

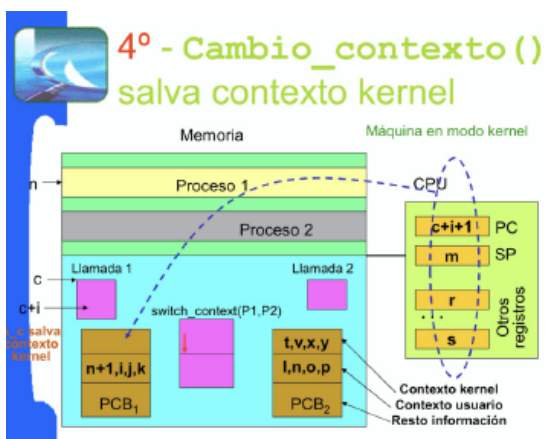
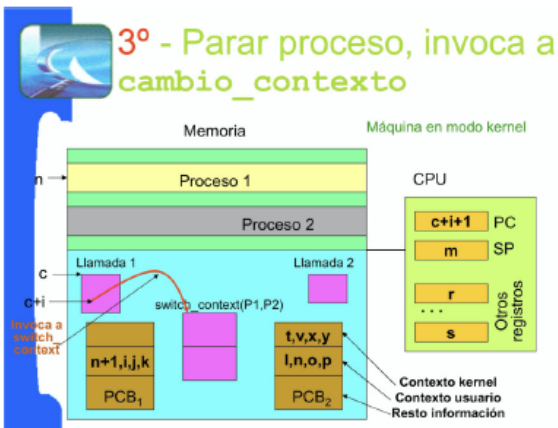
Dato importante: hay sistemas de switch la invocan como consecuencia de haber invocado al planificador de procesos, y hay sistemas que invocan primero al switch y este invoca al planificador.

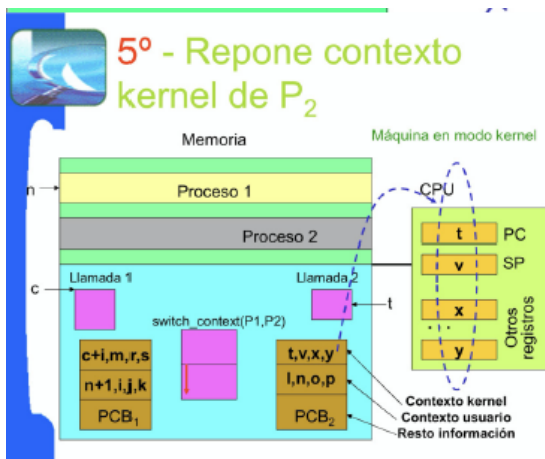
- Suponemos dos procesos:
 - P1 esta ejecutando la instrucción n que es una llamada al sistema.
 - P2 se ejecutó anteriormente y ahora esta en el estado preparado esperando su turno.
- Convenio:
 - Código del SO
 - Estructura de datos
 - Flujo de control
 - - - Salvar estructuras de datos
 - Instrucción i-ésima a ejecutar



Tenemos P1 y P2 en memoria pral, y además está el kernel. Los PCB son los bloques de control de los procesos donde se va aguardando si el proceso está bloqueado, ejecutando

Primero se produce un salto a una llamada al sistema, el kernel (CPU) salta de modo usuario a supervisor, el hardware guarda los valore sy se guardan en PCB y después llama a switch_content() y se queda bloqueado y se despierta al proceso P2.





Ahora se reanuda P2 y recupera la siguiente instrucción que esta en PCB2. Vuelve al proceso P2, lo cargamos en la CPU y cambiamos el modo de la CPU.

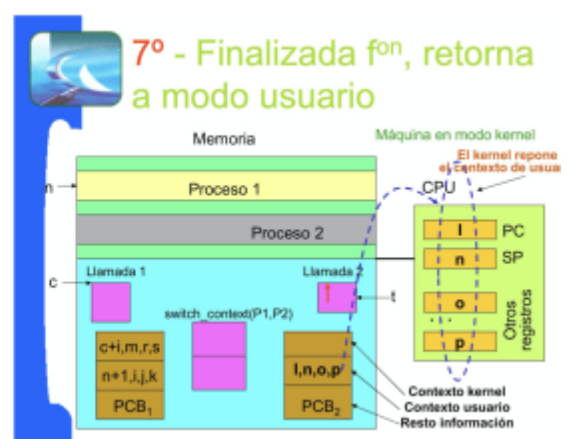
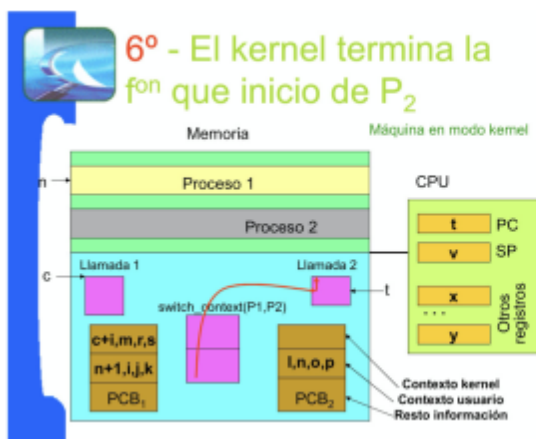
Toda la información del proceso está en la PCB (Bloque de control de proceso) donde hay una serie de registros (**n+1, i,j,k**). Estos registros, en contexto hardware, es el contexto que se suele asociar a los valores que mantiene el proceso durante la ejecución en los registros del procesador.

Estos valores se guardan dos veces ya que realmente el sistema tiene dos modos de funcionamiento (usuario y kernel) y entre el funcionamiento de uno y otro hay un salto. Por tanto, hay que guardar tanto el contexto en modo usuario como el contexto en modo kernel.

Cuando se recupera el nuevo proceso en la segunda parte del switch context(), se recupera el contexto de donde se ejecutó el proceso que se detuvo anteriormente (lo recuperamos del contexto hardware). Esto se carga en la CPU, con lo cual continúa el proceso ejecutándose pero saliendo del modo kernel, más o menos.

Luego hay que recuperar, en el modo usuario del proceso 2, el contexto hardware que se asocia con el proceso en modo usuario.

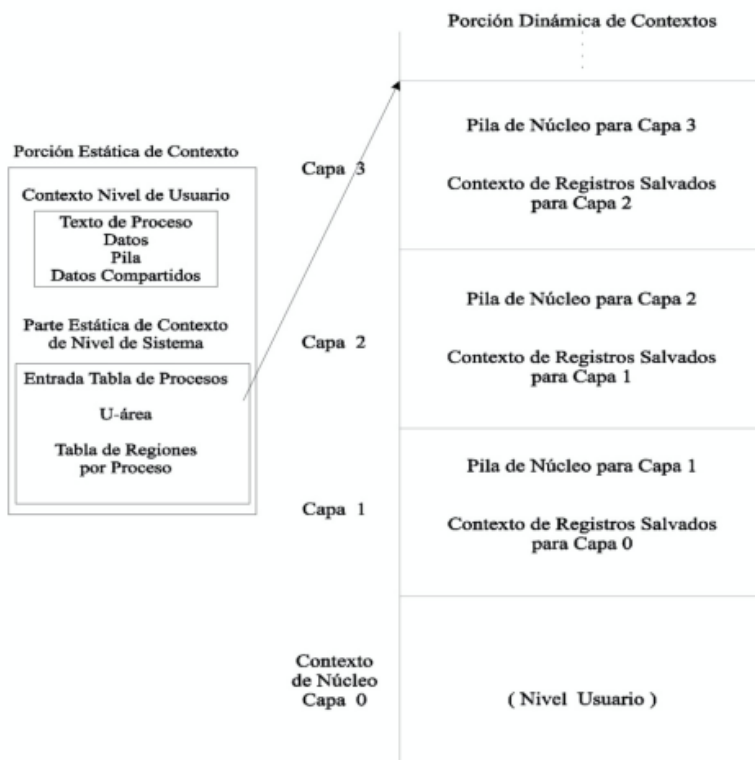
Los valores que tiene la CPU son los valores que se cargan para continuar con el modo usuario justo en el punto siguiente donde se produjo por ejemplo una llamada.





El sistema tiene la visión de que cambiar de proceso es realmente conmutar entre contexto de procesos. Se habla de conmutar porque se va pasando/recuperando entre procesos que ya existen.

- Contexto a nivel de usuario: Tiene condio, la pila en modo usuario
- Contexto a nivel de registros / hardware: PC (Contador de programa) Tiene los dos niveles de registro (usuario y kernel)
- Contexto a nivel de sistema: parte fija y parte dinámica que se maneja como una pila.



En la porción estática de contacto hay dos partes: el contexto a nivel de usuario que también es estática y después la parte estática de contexto de nivel de sistema, la cual tiene un identificador, un enlace a los procesos hijos que tenga, etc.. Esta parte tiene un puntero que siempre va a localizar el top de esa pila que corresponde a la parte dinámica de un proceso.

La pila es dinámica porque crece y decrece.

Pregunta: si mientras se está ejecutando la rutina de servicio de interrupción de teclado, ocurre una interrupción de disco, ¿qué va a hacer el sistema?

Se detiene la rutina de servicio de interrupción de teclado en ese punto y se salta a ejecutar la rutina de servicio de interrupción del disco. Cada vez que ocurre cada una de esas interrupciones, se añade una capa a la pila. Porque cuando acabe esta rutina, hay que retomar la rutina anterior en el estado en el que estaba y concluir.

Las interrupciones se atiende entre ejecuciones de instrucción máquinas. Cuando concluye la instrucción máquina que está actualmente ejecutando el procesador, antes de pasar a la ejecución de la siguiente instrucción máquina, el sistema compruebe si el hardware ha sufrido interrupciones antes de ejecutar la siguiente instrucción máquina reacciona a esa interrupción.

Pregunta: Esa pila de capas de contexto puede crecer indefinidamente?

No, tiene un límite ya que nunca se atenderán interrupciones que tengan igual o menos prioridad. Las interrupciones están jerarquizadas:

- Llamadas al sistema.
- Dispositivos de E/S.
- Comunicaciones a través de la red
- Dispositivos muy rápidos: suelen trabajar con bloques de datos. Ej: disco duro
- Interrupciones de reloj.
- Excepciones. Ej: errores.

Donde 1 es lo que menos prioridad tiene y 6 lo que más prioridad tiene. Cuando se está atendiendo una interrupción de nivel igual o inferior, no se atiende. Por tanto se podrían producir hasta 6 interrupciones como máximo.

Pregunta: Si durante una llamada al sistema ocurre una interrupción de llamada al sistema no se atiende?

No puede ocurrir a no ser que sea el propio núcleo el que llame también al sistema. Se produce pero no se atiende. La interrupción está latente, pero no se guarda en la pila.


El núcleo entra varias veces en funcionamiento sin haber acabado en funciones anteriores, podemos decir así que es reentrante en su código.

Pregunta: Por qué la rutina de servicio de interrupción o la interrupción de reloj está por encima de todas las otras interrupciones que hay (excepto por las excepciones)?

El kernel tiene las tareas que son limitadas en el tiempo y es importante llevarlas a cabo para que el sistema responda dentro de plazos razonables. Por ejemplo, implementando el tiempo compartido; Si tiene varios procesos en los que se está implementando multiprogramación con tiempo compartido, es decir, los procesos se detienen o si no se han

detenido en un quantum de tiempo, el sistema los va a detener y les va a quitar la CPU a ese proceso para dárselo a otro.

Si la interrupción de reloj no está por encima de todo el resto de interrupciones no se cumplirían esos tiempos.



Respuesta

- La llamada al sistema `CrearProceso()` esta diseñada para crear un proceso cuyo PCB tiene la estructura anterior.
- ¿Qué valores tiene el contexto de este PCB?
 - El SO ajusta los valores del contexto de usuario para que el proceso recién creado se ejecute desde su primera instrucción.
 - Se crea un contexto kernel para que parezca que el proceso retorna de una llamada al sistema.

■ Nueva pregunta ¿por qué hacer esto así? ...

Esta es la forma que tiene el contexto de cada proceso del sistema. Información básica fija como es la PCB y aunque los otros dos contextos también son estáticos, el contexto modo kernel está compuesto por parte estático y parte dinámica.

La parte del PCB está más dedicada a campos como el identificador, la prioridad que tiene el proceso, los hijos que tiene..

Las otras dos partes de los contextos aunque se suelen asumir que también son PCB pero más bien referentes a la ejecución.

Pregunta: ¿Qué pasa cuando un proceso que entra en el sistema se acaba de crear?

El kernel hace todo lo posible para que cuando se cree un proceso (que esto ocurre cada 5-10 s) la llama la sistema hace a la vez todo el proceso de inicialización de los valores para que cuando llegue a retomar ese proceso sea como si se hubiera ejecutado ya previamente.

La función de llamada al sistema `Fork` o `CrearProceso()` lo que hace es ajustar todo para que realmente parezca que ese proceso ya existía previamente en el sistema y ya había sido ejecutado. Cuando un proceso crea otro proceso, el propio proceso hijo es una copia del contexto del proceso padre. Es una copia tan exacta que el proceso hijo es como si él mismo hubiera ejecutado el `fork`.

Para diferenciar el proceso padre del hijo, `fork` devuelve un valor. Este valor que devuelve, para el proceso padre es el PID del hijo y a este le devuelve 0.

El propio código del sistema operativo detiene un proceso y reanuda otro proceso. Esto está directamente relacionado con la programación concurrente. Para el diseño de programas concurrentes hay que usar mecanismos que permitan la comunicación y sincronización de las diferentes ejecuciones de un mismo programa. Hay muchos mecanismos de entre los que destacamos los semáforos, monitores, paso de mensajes... que permiten mantener la consistencia de los datos del núcleo. Fundamentalmente el problema que siempre hay que solucionar es la exclusión mutua.

Pregunta: Como un sistema UNIX implementa esta sincronización necesaria?

Utiliza diferentes mecanismos:

- El sistema de prioridad jerárquico anteriormente mencionado evita que se ejecuten 2 rutinas del mismo tipo.
- Cuando los núcleos no son apropiativos: los procesos que se ejecutan en modo supervisor no pueden ser apropiados, es decir, el kernel no les quita la CPU. El kernel quita la CPU a un proceso cuando se le acaba el quantum de tiempo y se lo da a otro.

Pregunta:Cuál es la solución para implementar un núcleo apropiativo que no tenga solo derecho de apropiación por quantums de tiempo? ¿Qué conlleva eso en el código del núcleo?

Tener un atributo que indique prioridad y de forma apropiativa. Para implementar esto en el código del núcleo, hay que estudiar donde hay secciones críticas y protegerlas con el equivalente a unos cerrojos/semáforos binarios (SpinLocks) y que no permitan que entre otro proceso.

La estructura de datos interna del núcleo son recursos que pueden estar en uso o libres. Cuando los procesos están en algún archivo abierto, esa entrada que representa el acceso a ese archivo es un recurso para el sistema. Si algún proceso está tocando campos de ese recurso, el proceso le ha tenido que poner un cerrojo indicando que este recurso está en uso.

Las rutinas sleep y wake up son de los pocos momentos en los cuales el núcleo para garantizar que se evitan las condiciones de carrera, deshabilita todas las interrupciones, incluso las de reloj. Entonces esto puede inducir a pensar que no se está cumpliendo el tiempo compartido, ya que se han deshabilitado las interrupciones; esto es cierto en parte ya que algún proceso puede sobrepasar el quantum de tiempo. De todas formas, el sistema tiene controlado el peor de los casos en el que pueda exceder mucho tiempo dicho proceso.

Los núcleos actuales son todos apropiativos: sino totalmente, al menos, parcialmente. Es decir, en cualquier lugar del núcleo, cuando un proceso se está ejecutando en modo núcleo se le puede quitar la CPU y pasar a ejecutar otro proceso.

Contexto proceso

Ese estado (sinónimo de contexto aquí) del proceso, la unión de:

1. Contexto a nivel de usuario(texto, pilas, datos)
2. Contexto a nivel de registros: (esta también tiene nivel usuario y kernel)
 - a. pc, dirección de la siguiente instrucción
 - b. Registro de estado del procesador (estado hardware)
 - c. sp, puntero de pila
 - d. registros de propósito general
3. Contexto a nivel de sistema
 - a. parte estática: estrada de un proceso a una lista de procesos
 - b. parte dinámica, variable, se conoce como pila de capas de contexto

Contexto proceso II

Se introduce una capa:

- ocurre una interrupción
- llamada al sistema
- cambio de contexto

Se saca una capa:

- regreso de una interrupción
- regreso de modo supervisor a modo usuario
- cambio de contexto

Se dice siempre que un proceso se ejecuta dentro de su capa de contexto actual

Ej. situación:

Hay una capa 0, nivel de usuario, ahora supongamos que realiza una llamada al sistema open para abrir un fichero, entonces salta a modo supervisor para que se empiece a ejecutar la rutina, sys open. Supongamos que se está ejecutando y el usuario le da a una tecla, cuando se está ejecutando el sys_open se produce una interrupción por teclado.

qué ocurre ?

el sistema va a reaccionar, y que va a hacer? las interrupciones se atienden entre ejecuciones de instrucciones máquinas, el hardware antes de pasar a ejecutar la siguiente ejecución máquina mira a ver si hay interrupciones. Así que cuando hay una interrupción el sistema la atiende y lanza la rutina de servicio de interrupción de teclado y esta rutina tendrá un conjunto de instrucciones a ejecutar

Ahora supongamos que estando el sistema ejecutando esta rutina de servicio de interrupción de teclado, y ocurre una interrupción de disco (el disco duro produce una interrupción), **qué ocurre, qué haría el kernel?** se detiene la rutina del teclado y se salta a ejecutar la rutina de interrupción de disco. Cada vez que ocurre esa interrupción y se van disparando las rutinas correspondientes hay que estar añadiendo una capa en la pila de contextos.

ahora, esa pila de capas de contexto puede crecer indefinidamente??

es decir, ocurre una interrupción, se atiende, ahora ocurre otra interrupción de la tarjeta de red, mientras se ejecuta esta rutina ocurre otra interrupción de disco etc etc

Las interrupciones están jerarquizadas, las interrupciones que puedan ocurrir están jerarquizadas, normalmente el nivel más bajo pertenece a las llamadas al sistema (son las que menos prioridad tienen), el siguiente E/S y periféricos, el tercer nivel reservados a las interrupciones de comunicación a través de la red. después las interrupciones que tiene que ver con dispositivos muy rápidos, los que trabajan con bloques de datos, disco duro memoria flash, después las interrupciones de reloj (hardware en el nivel superior) y por último las excepciones del sistema

Si se está ejecutando una capa superior y se produce una interrupción inferior, no se detiene, se añade a la cola; si ocurre una al mismo nivel, tampoco se detiene, esta tendrá más prioridad pero no se detiene la actual, se añade a la cola.

La pila puede acumular como máximo 6 capas a la vez.

Pues todo esto lo tiene que mantener el núcleo con su estructura de datos para darle cavidad a toda la información. El núcleo entra varias veces a funcionamiento sin haber terminado otras ejecuciones anteriores.

¿Por qué la rutina de interrupción de reloj está por encima de todas?

Si la interrupción de reloj no está por encima no se cumplirían todos los tiempos, no estaría acotado el tiempo y no se respetan los quantum, las funciones de kernel están acotadas todos por tiempos, de forma que hay tiempo compartido entre los procesos.

Observaciones

El sistema está diseñado para hacer continuamente la función `switch_context(P1,P2)` (entra de media en torno de 100 veces en un segundo) es una función que detiene la ejecución de un proceso salvando toda su información y ejecutando otro proceso, recuperando los valores que tenía antes. Ahora, ¿qué pasa cuando un proceso se acaba de crear? hemos dicho que se conmuta entre un proceso P1, P2, **qué pasa si P2 acaba de entrar en el sistema y se ejecuta por primera vez?** cómo gestiona el switch, cómo resuelve eso el kernel?

Hay una capa de inicialización de procesos, donde está esa capa?

en la propia rutina, si P2 es un nuevo proceso, hay que inicializar sus valores, hacer algunas comprobaciones..., poniendo varios if para comprobar si es un proceso nuevo.

esto anterior creo que no es ... a ver por donde sale el profesor**

El kernel no puede añadir más instrucciones y hacer la rutina más pesada, entonces los if no los pone en la rutina, entonces todo esto lo lleva el kernel a las llamadas al sistema que crean nuevos procesos, al fork.

Respuesta

La llamada al sistema `CrearProceso()` está diseñada para crear un proceso cuyo PCB tiene la estructura anterior.

el fork cuando un proceso crea a otro, la forma más rápida que tiene para hacer de forma eficiente, el propio hijo es una copia del contexto del proceso padre, de forma que parece que el proceso hijo parece que ha ejecutado el fork.

¿Cómo se diferencia el padre y el hijo? el padre devuelve el PID del hijo, y el hijo devuelve un 0.

¿Por qué hace el sistema todo esto?

Esto lo hace así por temas de rendimiento y eficiencia.

Consistencia estructuras

Es un problema de exclusión Mutua

En UNIX tradicional, utiliza di

- enmascaramiento de interrupciones
- núcleo no apropiado: los procesos que se ejecutan en modo supervisor no pueden ser apropiados. Se dice q no son no apropi, significa q el kernel no les quita la cpu
- bloquear a los procesos cuando una estructura del núcleo está en uso por otro proceso (sleep/ wakeup)

Los núcleo actual son apropiativos (total como Solaris, o parcialmente como SVR4)

Hay algún momento en el que el proceso quita la cpu? Cuando hay tiempo compartido, y cuando se acaba el quantum a un proceso, el kernel está diseñado para quitarle la cpu, llamando al switch_context.

Aunque sea un núcleo no apropiativo, el modo supervisor si le quita la cpu a un proceso y se la da a otros, aun así se entiende q son no apropiativos, porque son momentos en los q si hay que quitársel.

Los apropiativos son los que le puede quitar la cpu a un proceso, en cualquier momento/lugar o por cualquier circunstancia. **Cual es la solución que tiene que dar el kernel, para implementar un núcleo apropiativo? ¿Qué conlleva eso en la implementación del código del núcleo?** Hay que proteger las partes críticas con semáforos, en el código del núcleo hay que identificar esas secciones críticas y asegurar que en esa partes del código sólo entre un proceso, si otro proceso intenta entrar el núcleo lo bloquea. El código del núcleo aumenta , porque hay que incluir estas primitivas. Hoy en día los núcleos implementan cerrojos, de forma que hay una espera ocupada, los procesos no se bloquean, están “dando vueltas” esperando a que se abran los cerrojos, están basado en espera ocupada.

Tipos de procesos



Tipos de procesos

- Procesos de **usuario**
- Procesos **demonios** (*daemons*)
 - No están asociados a ningún terminal
 - Realizan funciones del sistema (spooler de impresión, administración y control de redes, ...)
 - Pueden crearlos el proceso **init** o los procesos de usuario
 - Se ejecutan en modo usuario
- Procesos del **sistema**
 - Se ejecutan en modo supervisor
 - Los crea el proceso 0
 - Proporcionan servicios generales del sistema
 - No son tan flexibles como los demonios (recompilación)

El núcleo es parte de todos los procesos

- Procesos de usuario
- Procesos demonios (daemons): se arrancan con el sistema, están constantemente activos. Tienen código que se ejecuta en modo usuario, pero son procesos que habitualmente no tienen ningún terminal asociado y que se arrancan cuando se arranca el sistema. Cuando necesitan algo del sistema, hacen una llamada al sistema directamente.
 - No están asociados a ningún terminal
 - Realizan funciones del sistema (spooler de impresión, administración y control de redes,...)
 - Pueden crearlos el proceso **init** o los procesos de usuario
 - Se ejecutan en modo usuario
- Procesos del sistema: no tienen código en modo usuario, el código que ejecutan están en el kernel. Por ej, os crea el proceso 0, siempre se ejecutan en modo supervisor y proporciona un servicios de sistema muy importante, intercambio de páginas de memoria a disco
 - Se ejecutan en modo supervisor
 - Los crea el proceso 0
 - Proporcionan servicios generales del sistema
 - No son tan flexibles como los demonios (recompilación)

El núcleo identifica a los procesos por su PID(identificadores de procesos) .

Todos los procesos en unix se van a crear con las llamadas al sistema **fork**, todo excepto el proceso 0, **porque?** Porque el **fork** funciona haciendo una copia casi completa de otro proceso y por tanto el cero no puede crearse copiando a otro, porque no hay otro.

El propio núcleo de Linux actualmente está programado con hebras. Está diseñado como un programa concurrente pero con sus propias hebras.

El proceso 1 (**init**) es el antecesor de todos los procesos que hay en un sistema. A partir de ahí todos los procesos se ejecutan con **fork()**:

- Procesos especiales:

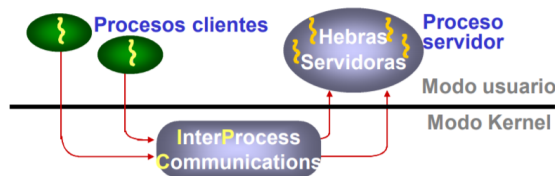
- Proceso 0: creado “a mano” cuando arranca el sistema. Crea al proceso 1 y se convierte en el proceso intercambiados.
- El proceso 1(proceso init): padre del resto de procesos de un sistema. Todos de alguna forma acaban apuntados al proceso 1.

Limitaciones de procesos

Los procesos tienen limitaciones.

Por ej: cuando un servidor tiene muchas peticiones de clientes, el sistema no va a crear un proceso para cada petición. Para esto aparecen las hebras. Las hebras resultan de separar dos conceptos de un mismo proceso. ***Buscar definición de servidores idempotentes

1. Por naturaleza paralela de aplicaciones necesitan un espacio común de direcciones, p.e. servidores idempotentes.



2. Un proceso sólo puede usar un procesador a la vez.

Hebras

Aprovechar una parte común de un mismo código, obteniendo diferentes flujos de ejecución de un mismo código.

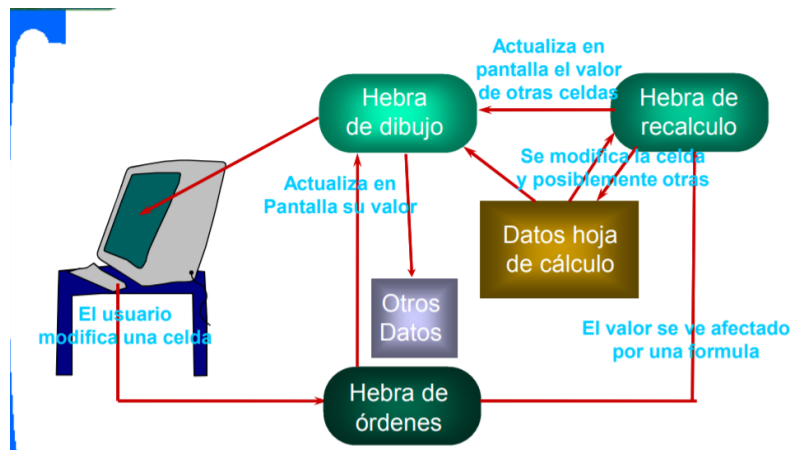
En un proceso confluyen dos ideas que podemos separar:

- flujo de control: secuencia de instrucciones a ejecutar determinadas por PC, la pila y los registros.
- espacio de direcciones: direcciones de memoria y recursos asignados(Archivos,...)

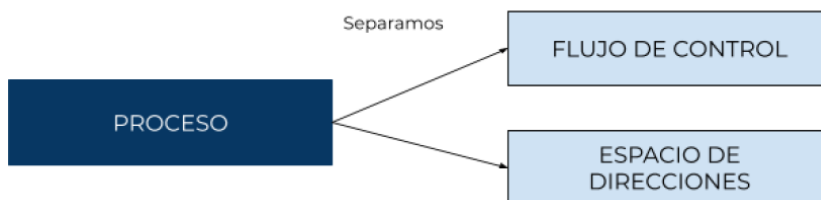
Permitir más de un flujo de control dentro del mismo espacio de direcciones

Ej: Hoja de cálculo

La idea es que las hebras resultan de separar dos conceptos diferentes que hay en los procesos: espacios de direcciones y flujo de control. Ambos conceptos se pueden englobar en un mismo código pero se pueden desacoplar diferentes flujos de ejecución.



Las hojas de cálculo se implementan como programas con multitud de hebras,. Hay actividades paralelas q tiene q sincronizarse. Los programas multihebras permiten que varias hebras trabajen conjuntamente. Todo eso se presta a que todo ocurra dentro de un mismo programa con los recursos que ya están reservados y no se dupliquen.



Clase 05/11/20

6.E/S Asincronas

El sistema hardware esta diseñado para trabajar de manera asíncrono pero en cambio la interfaz del sistema operativo en los programas (abrir ficheros, leer datos,...) es síncrona, es decir si nosotros pedimos datos y esos datos no se encuentra en memoria pral, pasaría a estar bloqueado hasta que se encuentre preparado y entonces pasa a ejecutarse.

Aqui es cuando surgen **las hebras** porque nuestro programa podría estar ejecutando otro programas mientras se estan pidiendo unos datos, siempre que no entre en conflicto.

Proceso asíncronas

Proceso multihebrado

E/S asíncrona para la hebra principal

E/S síncrona para la hebra que gestiona la E/S

Si tenemos un proceso tradicional y en su ruta de necesita unos datos pero no queremos que el flujo de ejecución espere, en ese punto en el que son necesarios los datos se crea una hebra(un nuevo flujo de control de ejecución) del mismo programa que se encarga de realizar esa petición de los datos, y por tanto el flujo de programa principal continua a no ser que para seguir necesitará si o si los datos, entonces esperaría. .

Cuando se necesitan programas multihebras?

- Cuando sean tareas independientes: depurador que necesite E/S asincronas.
- Programas únicos que tienen muchas operaciones concurrentes: servidores de archivos y web, Kernels de SOs: ejecución simultánea de varias peticiones de usuario...
- Uso del hardware multiprocesador: o un procesador con varios núcleos, cualquier programa que pueda hacer dos tareas en paralelo se puede sacar más partido. Los kernels de los SO ya el propio programa es multihebrado.

Cuando no:

- Cuando una máquina no tiene varias CPUs
- Si hay varias CPU pero todas las operaciones son intensivas (los datos son dependientes de lo anterior), porque no tendría sentido.
- Las hebras tienen algún coste aunque menor que los procesos(ej memoria): una hebra con pocas líneas de código no es muy útil.

Crear una hebra es menos costoso(hoy en día 5, 10 veces) que crear un proceso. Pero tiene un coste en líneas de código.

Cuestión importante asociado a las hebras:

7.Código reentrante

El código reentrante es aquel que funciona correctamente si 2 o más hebras lo ejecutan simultáneamente. Ej: una parte del SO es reentrante, el propio kernel lo es; o las funciones recursivas.

El código reentrante consiste también en que esa función no debería tener datos locales en un módulo (datos globales a ese módulo o static) porque puede pasar que en otro parte se este tocando y cree un conflicto.

Ej. el kernel 2.4 linux no es 100% reentrante, el 2.6 si. MS-DOS y la BIOS no son reentrantes.

Otro ejemplo típico, las funciones recursivas

Otra parte interesante es que el código reentrante no solo consiste en crear otra instancia sino que también consiste en que esa función no acceda a los mismos datos globales ya que no estarán especificados como secciones críticas con los mecanismos correspondientes de seguridad.

8.Diseño de hebras

Una hebra es como un CPU virtual. Necesita que se le asigne una CPU física. Cada hebra tiene un flujo de ejecución que determina su contador de programa para saber por dónde va ejecutando su código. Cada hebra tiene su pila porque es parte del flujo de control de ejecución, por ejemplo una hebra sala a ejecutar una parte del código y crea un marco de pila que se almacena en su pila para que no se encuentre con la ejecución de B o el resto de las hebras.

El resto de estructuras que monta el kernel y datos globales dentro de nuestro programa si van a ser compartidos.

El primer SO que implementó las hebras mas completas fue Solaris 2 y es de las implementación más completas que hay.

En un proceso tradicional la prioridad, las señales, registros y pila kernel solo esta una vex cuando hay una hebra, cuando mas de una, se tiene que replicar para cada hebra.

Hebra = unidad de asignación de la CPU(de planificación)

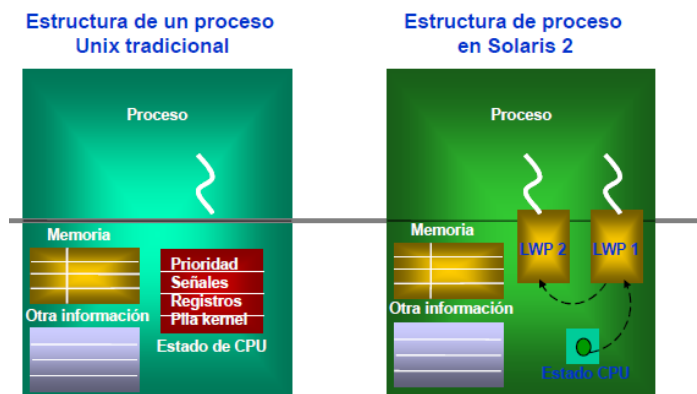
cada hebra necesita su contador de programa para saber por dónde va la ejecución del programa.

Tiene su propio contexto hardware:

- Su valor del contador de programa
- los valores de los registros
- su pila de ejecución, porque es parte del flujo de control de ejecución

Solaris fue el primer SO precursor del uso de hebras

Implementación de LWP



Tipos de hebras

Las hebras se distinguen a la hora de programarlas en base al nivel en el que se encuentran. Cada nivel da lugar a un tipo de hebra:

- **Hebras kernel:** implementadas en el mismo kernel. Conmutación entre hebras rápidas. El kernel es capaz de crearlas y controlarlas.
- **Hebras de usuario:** hebras que se ejecutan en el espacio de direcciones que le corresponda al proceso siempre que esté en modo usuario. Implementadas a través de una biblioteca de usuario que actúa como un pequeño kernel. La conmutación entre ellas es muy rápida, porque no pregunta al kernel y puede crear hebras, modificarlas, asignarlas...
- **Enfoque híbridos:** implementan hebras kernel y de usuario y procesos ligeros. Hay hebras a nivel de usuario, el sistema es capaz de reconocer hebras a nivel de usuario asociadas a un mismo proceso, y además les puede asociar hebras de tipo kernel. Esa asociación se suele denominar procesos ligeros (p.e Solaris 2).

Hebras de usuario

- Alto rendimiento porque se ejecutan nada más en modo usuario y no consumen recursos de kernel. El kernel no las reconoce entonces no consumen mas que un proceso tradicional, aunque son varias hebras. Cuando a ese proceso se le acabe el quantum el kernel la detiene. No hace llamadas al sistema, para gestionar que una hebra se bloquee o pasa a listo, no hace falta saltar al kernel, lo da la biblioteca asociada.
 - Para crear una hebras tiene que haber un tamaño critico (700-800 instrucciones), habitualmente se ha dicho que crear una hebra para ejecutar 200 instrucciones no interesa.
 - Tiene que tardar menos que el propio proceso en ejecutar.
 - Al no conocer el kernel su existencia:
 - No aplica protección entre ellas.
 - Problemas de coordinación entre el planificador de la biblioteca y el SO. La biblioteca y el SO tienen su propio planificador y puede ser que no estén coordinadas las dos planificaciones.
 - Si una hebra se bloquee pidiendo datos al kernel por ejemplo, bloquea la tarea completa. Por qué? Cual es el problema? Que el kernel no está viendo que están planificadas varias hebras que se están conmutando, el sabe que si le han pedido unos datos, entonces el kernel bloquea la hebra hasta que pueda darle los datos necesarios
- Un proceso con hebras en modo usuario, el kernel ve el proceso como un proceso tradicional de una sola hebra, el no ve que es un programa avanzado. La hebra que ha pedido datos de un fichero, queda bloqueada por el kernel, lo suyo es que el planificador ahora planifique otra hebra en modo usuario, pero no puede porque el kernel lo ha bloqueado todo. El kernel no ha dejado retornar el proceso del modo usuario, le ha quitado la CPU.

Estándares de hebras

El estándar de las hebras normalmente son POSIX, entonces para las hebras que están muy ligadas al SO también utilizan este estándar y casi todos los sistemas UNIX tienen una biblioteca de hebras

- POSIX (Pthreads) -ISO/IEEE estándar
 - API común
 - Casi todos los UNIX tienen una biblioteca de hebras.
- Win32
 - Muy diferente de POSIX
 - Existen bibliotecas comerciales de POSIX
- Solaris
 - Anterior a POSIX
- Hebras Java

Todos los sistemas prácticamente tienen sus bibliotecas de hebras, por ejemplo Microsoft siempre ha tenido Win32, que es bastante diferente a POSIX pero en Windows también existen hebras de POSIX, de las cuales podemos hacer uso. Solaris también tiene su implementación de hebras, en las que se basan POSIX.

Solaris 2.x

Es una versión multihebra de Unix, soporta multiprocesamiento

Tareas hebras y LWP

Modelo de hebras de Solaris

Tarea1: ejemplo de tres hebras de usuario (naranjas) son soporte de dos hebras de kernel (azules) (todo esto es virtual, son conceptos lógicos). Dos hebras de usuario comparten una kernel por lo que tiene que haber una planificación en modo usuario que permita a ambas hebras compartir una misma hebra de kernel y se implemente como un único proceso ligero.

Tarea2: proceso ligero

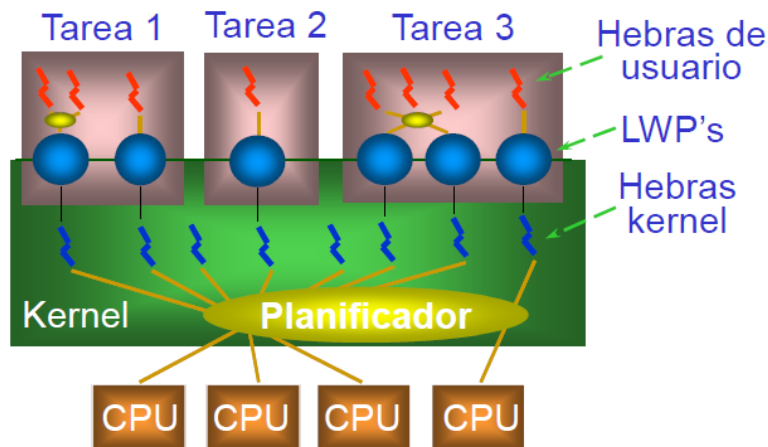
Tarea3: dos hebras en modo usuario que comparten dos hebras en modo kernel.

El planificar es el que decide cuando la virtualización se convierte en la realidad, cuando a una de esas hebras se le asigna una CPU y avanza en la ejecución.

Como se llama la hebra de usuario que tiene dedicada exclusivamente una hebra de kernel?
hebras ligadas

Por que ahora los procesos pasan a llamarse tareas y crean estructuras de tareas?

Los sistemas entienden que el concepto de tarea va más allá que el concepto de proceso que es un flujo de ejecución. Ahora si hay varias hebras, hay más flujos de ejecuciones.



segunda parte de la clase**

9. Hebras en Linux

Depende del tipo de versión de Linux, la implementación de las bibliotecas POSIX tenían semánticas diferentes.

Hoy en día, Linux soporta hebral kernel pero está implementada en labores del propio sistema y también tienen bibliotecas de hebras de usuario.

- Linux soporta:
 - Hebras kernel en labores de sistemas: ejecutan una única función en modo kernel: limpieza de cachés de disco, intercambio, servicio de conexiones de res, etc
 - Bibliotecas de hebras de usuario
- No todas las bibliotecas del sistema son reentrantes

Hay que tener en cuenta que no solamente lo que programamos tenemos que asegurarnos de que son realmente bibliotecas thread-safe(reentrantes)

APIs de Unix y Win32

Operación	Unix	Win32
Crear	<code>fork()</code> <code>exec()</code>	<code>CreateProcess()</code>
Terminar	<code>_exit()</code>	<code>ExitProcess()</code>
Obtener código finalización	<code>wait</code> <code>waitpid</code>	<code>GetExitCodeProcess</code>
Obtener tiempos	<code>times</code> <code>wait3</code> <code>wait4</code>	<code>GetProcessTimes</code>
Identificador	<code>getpid</code>	<code>GetCurrentProcessId</code>
Terminar otro proceso	<code>kill</code>	<code>TerminateProcess</code>

Como podemos ver en la imagen anterior, Linux no realiza todo en una ejecución como lo hacer Windows normalmente.

Aquí podemos ver las hebras que se ejecutan en Unix y Windows

Operación	Pthread	Win32
Crear	Pthread_create	CreateThread
Crear en otro proceso	-	CreateRemoteThread
Terminar	Pthread_exit	ExitThread
Código finalización	Pthread_yield	GetExitCodeThread
Terminar	Pthread_cancel	TerminateThread
Identificador	-	GetCurrentThreadId

***** Parte2 del tema *****

10. Planificación del Sistema Operativo (Planificación de CPU)

El planificador es un módulo del sistema operativo que controla como se utiliza la CPU.

Planificador:

- **Definición:** es un módulo del SO que controla la utilización de recursos, cuando se habla de planificador, se entiende planificar de Cpu, pero un SO tiene planificadores para por ejemplo implementar algoritmos en el acceso a Disco, acceder a dispositivos E/S, etc...
- Hay dos planificadores básicos, a largo a plazo y corto. A largo plazo quiere decir cuando un programa ejecutable realmente el SO lo va a convertir en proceso. A corto plazo: en cada momento va a asignando el cambio de contexto de un proceso a otro, es decir, va a asignando la cpu a cada momento.
- Hoy en día ya no existen, existen variantes como el demonio cron y at(cliente, atd:demonio)

Planificador a corto plazo:

- Selecciona que procesos entran a la CPU.
- Ocurre en espacios muy cortos de tiempo.
- Se invoca con mucha frecuencia (del orden de mseg.) por lo que debe ser rapido en su ejecución.

Planificador a largo plazo:

- Consiste en seleccionar que tareas/procesos van a la cola de preparados. Ahora mismo no le estamos asignando la CPU, si no que se le va metiendo en la cola para que se le asigne.
- se invoca con menor frecuencia (min o seg) por lo que puede ser lento.
- influye en el grado de multiprogramación, ya que influye en cuántos programas hay a la vez ejecutándose.

Hoy en día, existe una variante ya que cualquier usuario que quiera iniciar una aplicación, el sistema le va a permitir el acceso y va a crear el proceso y no crea una cola como tal.

Cual es la variante a la planificación a largo plazo? Los demonios son una planificación a largo plazo pero lo está decidiendo el usuario y no la máquina, y por tanto es una variación.

Planificador a medio plazo: hoy en día se usa mas que el de largo plazo. Se encarga de la parte que tiene q ver con el intercambio (swapping). Se encarga de las transición que hemos visto anteriormente que a veces si un proceso estaba bloqueado por falta de memoria y tenia que dejar el sistema libre, son unos procesos candidatos que tiene que ocurrir el desbloqueo (imagen de abajo). Es un gestor de memoria pero debería estar coordinado con el planificador del sistema.

11.Ráfagas de CPU



Planificadores de procesos

- **Planificador** – módulo del SO que controla la utilización de un recurso.
- **Planificador a largo plazo** (o *de trabajos*) – selecciona procesos que deben llevarse a la cola de preparados.
- **Planificador a corto plazo** (o *de la CPU*) – selecciona al proceso que debe ejecutarse a continuación, y le asigna la CPU.

Una cuestión importante para los planificadores es cómo pueden tener diferentes parámetros de cómo se están comportando los procesos en el sistema si tiene que tener en cuenta que hay diferentes tipos de procesos en el sistema.

Ej: un proceso interactivo va a necesitar muchas operaciones de E/s y van a ocurrir múltiples interrupciones.

Los procesos acotados a entradas y salidas

- La ejecución de un proceso consta de ciclos sucesivos ráfagas de CPU-E/S
- Procesos acotados a E/S- muchas rafagas cortas de CPU
- Procesos acotados a la CPU - ráfagas mas largas y con menos frecuencia.
- Procesos de tiempo real: ejecución definida por (repetidos) plazos (deadline). El procesamiento por plazo debe ser conocido y acotado.

12/11/20

Segunda parte

Planificador

Es un módulo del SO que controla la utilización de un recurso. Un SO tiene planificadores para implementar algoritmos.

Hoy en día existen variantes de planificadores, como el cron. Mediante crontab (programa cliente) proporcionamos al sistema el demonio cron.

Cual es la diferencia entre el planificador generico a largo plazo y el cron?

Antes quien decidía cuando entraban los procesos en el sistema era el propio sistema, pero el cron permite planificar que el propio usuario especifique cuando se ejecuta dicho programa.

Aparte de cron esta at(programa cliente) y atd (el demonio).

Cuando planificar?

Políticas de planificacion

Si un proceso que era importante estaba bloqueado y pasa a preparado se invoca al planificador para ver si se ejecuta o no. y también cuando un proceso que se está ejecutando termina.

- Las decisiones de planificación pueden tener lugar cuando:
 - Estado ejecutándose a preparado
 - Estado ejecutándose a preparado
 - Estado de bloqueado a preparado
 - Estado ejecutándose a finalizado
- Es decir siempre que un proceso abandona la CPU o se inserta un proceso en la cola de preparados.

Planificación apropiativa (preemptive): El SO puede quitar la cpu al proceso, invocando al planificador para decidir si se cambia un proceso en ejecución por otro.

Planificación no apropiativa: No se puede retirar al proceso de la CPU, esté la libera voluntariamente al bloquearse o finalizar.

Muchas veces clasificar a un proceso con una planificación u otra no es totalmente exacta. La elección entre ambas depende del comportamiento de la aplicación (de lo que queremos hacer) y del diseño que queremos hacer del sistema.

Apropiación frente a no apropiación

Apropiación nos asegura que un trabajo no bloquea (si no q no lo tiene en espera) a otro igualmente importante

Cuestiones a tener en cuenta:

- Cuando apropiar? en tiempo de interrupción?
- Tam de la fracción de tiempo?

La planificación no apropiativa requiere que los trabajos invoquen explícitamente al planificador.

Un trabajo erróneo puede tirar el sistema.

Simplifica la sincronización de hebras/ procesos.

Despachador(dispatcher)

Hace referencia al mecanismo que hace un SO de lo que es la planificación. El despachador da el control de la CPU al proceso seleccionado por el planificador. Realiza lo siguiente:

- Se encarga de llevar a cabo el cambio de contexto(en modo kernel)
- Conmutación a modo usuario
- Salto a la instrucción del programa para su reanudación.

Latencia de despacho: tiempo que emplea el despachado en detener un proceso y comenzar a ejecutar otro. Interesa que sea baja.

Bucle de despacho

■ En pseudocódigo:

```
while (1) {    /* bucle de despacho */
    ejecutar un proceso un rato;
    parar al proceso y salva su estado;
    carga otro proceso;
}
```

Cómo obtiene el despachador el control?

Hay dos formas y un mismo sistema puede usar ambas:

- Síncrona: un proceso cede la CPU. Cuando un proceso se bloquea o termina.
- Asíncrona: iniciado por una interrupción u ocurrencia de un evento que afecta a un proceso (por ej. fin de e/s, liberación de recurso, etc).

Gestor de interrupciones revisado

Realiza las siguientes operaciones:

1. Salva el contexto del proceso en ejecución
2. Determina el tipo de interrupción y ejecute la rutina de servicio de interrupción adecuada.
3. Selección del proceso que se ejecuta a continuación.
4. Restaura el contexto salvado del seleccionado para ejecutarse.

Implementación de tiempo compartido

Hay que tener en cuenta que hay una rutina de servicio de interrupción de reloj, rutina que puede programar el propio núcleo para que ocurra cada 10 ms por ejemplo, entonces es una forma de tomar de forma controlada el núcleo el control para decidir en esos momentos si hay que planificar otro proceso

El SO asigna la CPU a un proceso y le asocia una fracción de tiempo o quantum

Mecanismo VS Política

Un mecanismo es el código (a menudo a bajo nivel) que manipula un recurso.

- CPU: cambio entre procesos
- Memoria: asignar, liberar,...
- Disco: leer, escribir

Una política decide “ como , quien cuando y porque ”:

- Cuanta CPU obtiene un proceso
- Cuanta memoria le damos
- Cuando escribir en disco

Criterios de planificación

- **Utilización:** mantener la CPU tan ocupada como sea posible
- **Productividad:** nº de procesos que completan su ejecución por unidad de tiempo.
- **Tiempo de retorno(tiempo de servicio, tiempo de ejecución):** cantidad de tiempo necesaria para ejecutar un proceso dado. Desde que se lanza hasta que termina.
- **Tiempo de espera:** tiempo que un proceso ha estado esperando en la cola de preparados
- **Tiempo de respuesta:** tiempo que va desde que se remite una solicitud hasta que se produce la primera respuesta (no salida).

Métricas de planificación

La elección depende del tipo de aplicaciones y el uso del SO. Máxima utilización:

- Máx producción
- mín tiempo de retorno
- mín tiempo de espera
- mín tiempo de respuesta

En la mayoría de los casos, algunos de estos criterios son contrapuestos.

Primero en Llegar (FIFO)



Primero en Llegar, Primero en Servir (FIFO)

■ Para los procesos de la tabla, construimos el Diagrama Gantt.

■ Tiempos de espera:

■ $P_1 = 0$

■ $P_2 = 24$

■ $P_3 = 27$

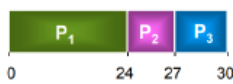
■ Tiempo medio de espera (t_e medio):

$$(0+24+27)/3 = 17$$

a)

Proceso	T. ráfaga	T. llegada
P_1	24	0
P_2	3	0
P_3	3	0

b)



El sistema se dedica a ejecutar los procesos en el orden que llegan.

Tiene un efecto escolta, que consiste en que los procesos cortos se quedan esperando mas tiempo a q terminen los procesos largos, por lo que el tiempo medio de espera se verá aumentado, este efecto hace que este algoritmo no se muy bueno, ese efecto escolta se puede quitar haciendo que los procesos mas cortos se ejecuten antes que los largos.



Efecto escolta

• Si cambiamos el orden de ejecución de los procesos, t_e medio es mejor:

$$(6+0+3)/3 = 3$$

• **Efecto escolta** – los procesos cortos esperan a los largos.



Tiempos de espera:

$P_1 = 6$

$P_2 = 0$

$P_3 = 3$

Para los procesos de la tabla, construimos el Diagrama de Grantt.

Tiempos de espera:

- $P_1 = 0$
- $P_2 = 024$
- $P_3 = 27$

Tiempo medio de espera(t_e medio): $(0+24+27)/3=17$

Primero el Mas Corto (SJF)

Se planifica el proceso cuya siguiente ráfaga es la más corta. Tiene dos versiones:

- **Apropiativa**(Primero el de tiempo restante menor - **SRTF**): cuando llega un nuevo proceso si el nuevo necesita menos de lo que le queda al proceso actual, le quita la cpu porque a él le queda más tiempo de ejecución que el que tarda el nuevo proceso. Un proceso con ráfaga más corta que el tiempo restante del proceso en ejecución, apropiada al proceso actual.
- **No apropiativo**: el proceso en ejecución no se apropia hasta que complete su ráfaga.

Característica de SJF

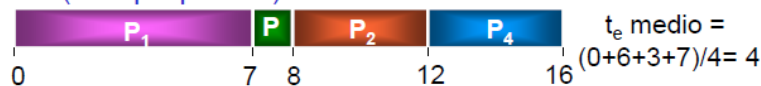
- Minimiza el t_e medio para un conjunto dado de procesos (no así, el tiempo de respuesta)
- Se comporta como un FIFO , si todos los procesos tienen la misma duración de ráfaga.
- Actualmente se utilizan variantes de este algoritmo para planificación de procesos en tiempo-real

EJ de SJF:

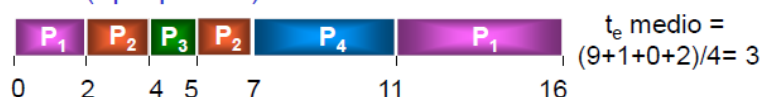


Proceso	T. Ráfaga	T. Llegada
P ₁	7	0
P ₂	4	2
P ₃	1	4
P ₄	4	5

□ SJF (no apropiativo)



□ SRTF (apropiativo)



No apropiativa

Para calcular el tiempo medio de cada proceso, se coge el instante en el que llego y se le resta el instante de T. Llegada:

- P1= 0
- P2= 8-2=6
- P3=7-4=3
- P4=12-5=7

En caso de igualdad de T. de Llegada se coge el que menos T rafaga tenga

La versión apropiativa mejora en comparación a la versión no comparativa.

Apropiativa

P2 necesita 4u, a p1 le queda 7-2= 5 cuando llego p2, por tanto el sistema planifica p2 y le quita la cpu a P1.

Con P3 igual, P3 solo tiene 1u de rafaga y a P2 le quedan aun 2, por tanto P3 le quita la cpu a P2

$$P1=11-2 = 9$$

$$P2=5-4 =1$$

P3=0

P4=7-5=2

instante en el que se ejecutó - instante en el que llegó



Estimación tiempo de ráfaga

- Estimamos su duración con ráfagas previas.
- Sean:
 - T_n =duración actual de la n-ésima ráfaga
 - Ψ_n =valor estimado de la n-ésima ráfaga
 - Un peso W , donde $0 \leq W \leq 1$
 - Definimos: $\Psi_{n+1} = W * T_n + (1-W) \Psi_n$
- $W=0$; $\Psi_{n+1}=\Psi_n$, no influye historia reciente.
- $W = 1$; $\Psi_{n+1} = T_n$, sólo cuenta la ráfaga actual.

Problemas del algoritmo SJF

- Si siempre llegan procesos más cortos, los de rafagas más largas no llegan a ejecutarse, inanición.
Inanición: un proceso que se ve indefinidamente retardado porq hay otros procesos que se adelantes, y por tanto nunca obtiene la cpu
- El tiempo de espera no se puede conocer de antemano

Clase 19/11/20

Planificación por prioridades

Para estos sistemas no todos los procesos son tan importantes

Puede ser: apropiativa, es decir, el siguiente proceso a ejecutar el sistema ha cogido el más importante, si llega un proceso más importante el sistema quita la cpu al proceso que se estaba ejecutando y se la da al nuevo. Esto puede dar lugar a :

- Inanición: un proceso se va a ver retardado indefinidamente, en caso de que están llegando todo el rato procesos más importantes
- Para evitar esto, habitualmente se usa técnica de envejecimiento: consiste en bajar al importante q se están ejecutando e incrementar las a los que no se están ejecutando. La prioridad de un proceso puede cambiar dinámicamente.

Planificación Round-robin(RR) (Tiempo compartido, por turnos)

Se usan en planificadores de tiempo compartido o por turnos, a cada proceso se le asigna un quantum de CPU. Pasado ese tiempo si no ha finalizado ni se ha bloqueado, el SO se apropia al proceso y lo pone al final de la cola de preparados.

Hoy en dia en linux los quantum suelen ser de menos de 20 ms, el mismo sistema puede asignar quantums diferentes, cuando se pasa ese tiempo, no pasa el proceso a bloqueado, se quedará preparado y le da el turno al siguiente proceso que este en la cabecera de la cola. El kernel garantiza el control del cuántum con las interrupciones del reloj.

Características RR:

- Realiza una asignación imparcial de la CPU entre los diferentes trabajos. Todos los procesos tienen la misma importancia.
- Tiempo de espera medio:
 - bajo si la duración de los trabajos varía
 - alto si la duración de los trabajos es idéntica
- El rendimiento depende del tamaño de quantum
 - quantum grande \rightarrow RR tiende a comportarse como FIFO
 - quantum pequeño \rightarrow mucha sobrecarga si q no es grande respecto a la duración del cambio de contexto.

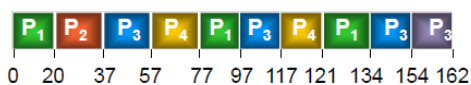
Tiene un mayor tiempo de retorno SRTF, es la versión apropiativa de primero el trabajo más corto. Típicamente, tiene un mayor tiempo de retorno que SRTF, pero mejor respuesta.



Ejemplo de RR con $q = 20$

Proceso	T. Ráfaga	T. Llegada
P ₁	53	0
P ₂	17	0
P ₃	68	0
P ₄	24	0

El diagrama de Gantt es



Típicamente, tiene un mayor tiempo de retorno que SRT, pero mejor *respuesta*.

P2 tiene un tiempo de ráfaga menor que el quantum, en cuanto termina de ejecutarse, el sistema pasa a ejecutar el siguiente proceso (no se espera a que se termine el quantum).

En la imagen pone SRT, pero se refiere a SRTF (SFJ apropiativo)

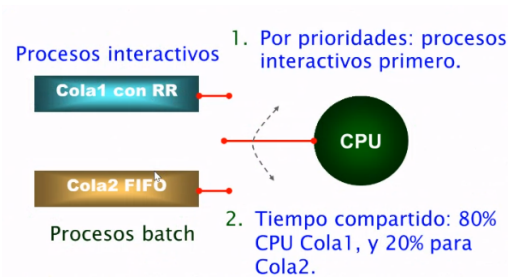
Colas múltiples

La cola de preparados se fraccionó en varias colas; cada cola puede tener su propio algoritmo de planificación.

El sistema divide a los procesos preparados en varias colas, estas divisiones pueden ser por tema de prioridad, tipos de procesos...

Siempre que haya procesos en la cola de más prioridad el sistema lo ejecuta, solo pasa a la otra cola cuando la cola de más prioridad no tiene más procesos esperando.

O puede ser q sea de tiempo compartido, a una cola le puede dar el 80% de tiempo de CPU y a otra cola darle el 20% de tiempo, por q sean de otro tipo.. o lo q sea



Cola múltiples con realimentación

La realimentación justamente es que los procesos se pueden mover dinámicamente a lo largo de la ejecución entre diferentes colas. En estos casos intentamos aproximarnos a la técnica de envejecimiento. Los parámetros necesarios son, colas totales, algoritmo para cada cola y algoritmos que van a determinar cuando un proceso se cambia, asciende o desciende en una cola.

Se utilizan mucho en sistemas interactivos.

CMR en sistemas interactivos

- Planificación entre colas por prioridad
- Estos sistemas asignan prioridad basándose en el uso que hacen de CPU- E/S que indica cómo estos “cooperan” en la compartición de la CPU:
 - + prioridad cuando se realizan más E/S
 - - prioridad cuanto más uso de CPU

Planificación en multiprocesadores

Con varias CPUs, la planificación es más compleja: puede interesar que una hebra se ejecute en un procesador concreto (puede tener datos en caché) o que varias hebras de una tarea planificada simultáneamente.

El sistema considera que cualquier CPU que se quede libre se le asigna al siguiente proceso. Dos técnicas para repartir el trabajo:

Equilibrado de carga: otro nivel de planificación a parte del planificador general, reparto uniforme de la carga entre las diferentes CPUs.

Distribución de carga: repartir la carga entre CPUs para no tener ninguna ociosa.

Linux permite que una hebra esté asignada a un único núcleo de la CPU, al ver el listado de sistemas se ve por qué el sistema lo muestra con una barra y el número de núcleo que le corresponde, siempre que el kernel haya decidido que esa hebra se ejecute en un kernel concreto.

Planificación Tiempo-Real

- Sistemas de tiempo-real duros (hard) - necesitan completar una tarea crítica dentro de un intervalo de tiempo garantizado. Necesitan reserva de recursos.
- Sistemas de tiempo-real blandos (soft) - requieren que los procesos críticos reciban prioridad sobre los menos críticos. Basta con que los procesos de tiempo-real tengan mayor prioridad y que esta no se degrade con el tiempo.

Es importante reducir la latencia de despacho para acortar el tiempo de respuesta:

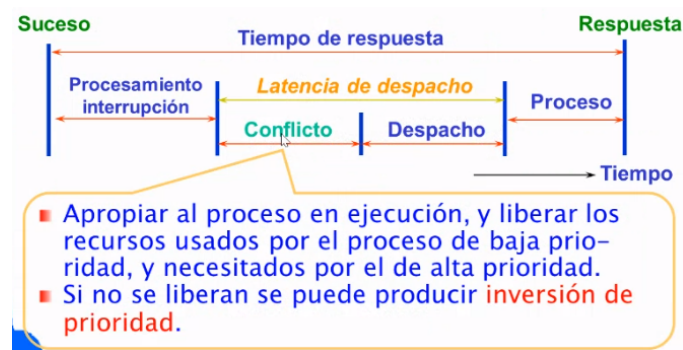
Como algoritmos de planificación se suelen utilizar variaciones del SJF:

- EDF(earliest-deadline First): Divide los trabajos por plazo selecciona primero el trabajo con el plazo más próximo.
- Razón monótona: asigna prioridades inversamente al periodo, asigna más prioridad a los procesos que entran en funcionamiento en un periodo más corto

Latencia de despacho

el tiempo que le lleva al sistema hacer el cambio de contexto conflicto por los recursos en uso.

despacho



Inversión de prioridad

no tiene una solución total actualmente, pero es un efecto que produce que va en contra de la propia planificación, si estamos suponiendo por ejemplo una política de planificación por prioridades

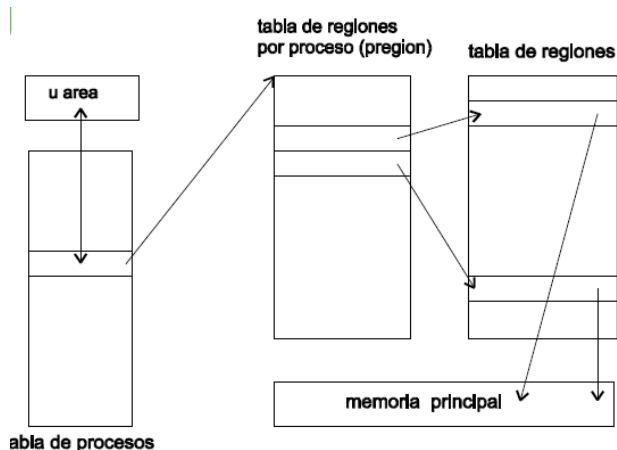
- Fenómeno producido cuando un proceso o hebra, P2, de prioridad p_2 , se ejecuta antes que P1, de prioridad $p_1 > p_2$ debido a que P1, espera por recurso que tienen bloqueado a P3
- Una solución: herencia de prioridad, P1, lega su prioridad a P3, (que bloquea el recurso)

Control de procesos

- Llamadas al sistema para procesos:

Relacionadas con Gestión de Memoria				Relacionadas con sincronización		
<i>fork</i>	<i>exec</i>	<i>brk</i>	<i>exit</i>	<i>wait</i>	<i>signal</i>	<i>kill</i>
algoritmos internos (<i>sleep</i> , <i>wakeup</i> , ...)						

12. Estructura de datos



regiones(segmento) En la tabla de regiones por proceso mínimo habrá 4 entradas: seg de código, datos globales, pila y heap (montón del proceso). Se gestiona dinámicamente

tablas de regiones globales: se supone que es la que deriva a buscar en memoria el seg de código, las var globales etc

¿Objetivo la división de la tabla de regiones (global y otra específica por proceso) ?

Tabla de procesos

- estado del proceso
- campos para localizar el proceso y su u-área, e información sobre el tam del proceso
- Relaciones entre procesos
- Descriptor de eventos que espera un proceso
- Parámetros de planificación
- Señales recibidas de otros procesos pero no tratadas aun
- Temporizadores indicando el tiempo de ejecución del proceso y el tiempo de uso de los recursos
- Punteros para enlazar el proceso en la cola del planificador, o si está bloqueado, en la cola de bloqueados
- Punteros para las colas hash en base a su PID.

Tabla de procesos o U-área

- Puntero a la entrada de la tabla de procesos.
- UIDs y GIDs real y efectivo.
- Tiempos que los procesos y sus descendientes gastan ejecutándose en modo usuario y modo supervisor
- Matriz que almacena la reacción ante las señales
- Terminal asociado al proceso, si existe
- Campos de error y del valor devuelto (para las llamadas al sistema)
- Tabla de descriptores de archivos, los archivos abiertos por un proceso, por ejem
- Parámetros de las operaciones E/S
- Directorio actual y raíz
- Tamaño limite de procesos y archivos
- Campo modo de permisos(umask)
- Pila del núcleo

Pregion

Puede estar en distintos lugares depende de la implementación:

- en la tabla de procesos o u-area
- en una zona de memoria asignada separadamente

Contenido de cada entrada

- puntero a la correspondiente entrada de la tabla de regiones(estructura global. lo q esta haciendo el sistema es dividir en dos regiones
- dirección virtual de comienzo de la región
- campos de permiso para el tipo de acceso del proceso

El concepto de región es independiente de las políticas de gestion de memoria utilizadas

Tabla de regiones

tiene un puntero al i-nodo del archivo ejecutable desde el cual viene el programa actual que se está ejecutando.

Tipo de región: texto, datos compartidos , pila,,,

Tamaño de región

localización de la región en memoria

Estado de la region (bloqueada, en demanda, cargando en memoria o válida)

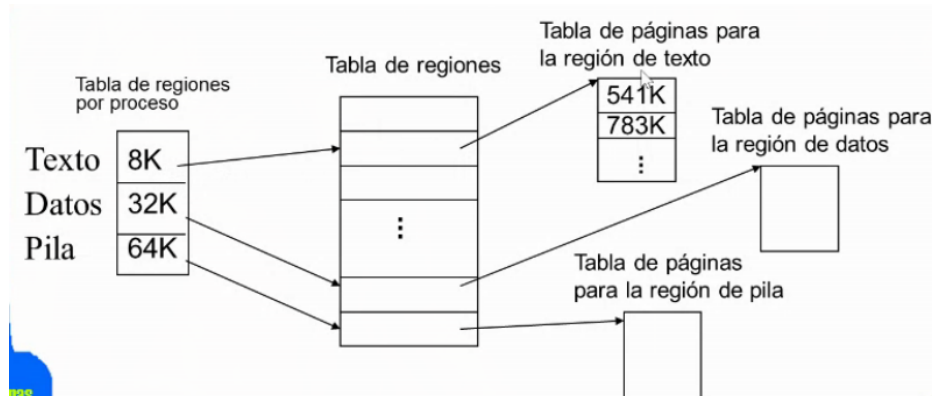
Contador de procesos que referencia esta región

La tabla de regiones por proceso suele tener cuatro entradas: una a la pila, otra al hit(el montón), otra a los datos y otra.. Luego suele haber más entradas.

Diseño de Memoria

Se utiliza un sistema de paginación por demanda:

cada región tiene asociada una tabla de páginas con información de las páginas que la forman



Si hay otro proceso, tendrá su tabla de regiones de procesos, su primera entrada esta apuntando en la 10K(por ej.) y acaba apuntando al mismo segmento que 8k(por ej,) si están compartiendo la misma región de texto (y la información)

¿Cuándo se podría dar este caso?

- Cuando son hebras del mismo proceso, directamente la tabla de regiones por proceso es la misma
Así que no, no es este caso
- Esta situación se da cuando un proceso padre crea a otro proceso hijo, con fork()

13.Creación de procesos

Los pasos a seguir por el SO:

1. Asignarle un PCB
2. Establecer su contexto de memoria (espacio de direcciones)
3. Cargar imagen (ejecutable) en memoria
4. Ajustar su contexto de CPU (registros)
5. Marcar la tarea como ejecutable
 - a. saltar al punto de entrada inicial o
 - b. ponerlo en la cola de preparados

Posibilidades

Un proceso puede crear otros procesos, y :

- Formar un árbol de procesos(**UNIX**) - relación de parentesco entre procesos
ó
- No mantener una jerarquía (**win 2000**, windows no tiene esa jerarquía)

¿Compartir recursos entre el creador y el creado?

- Comparten todos los recursos, o un subconjunto
ó
- Creado y creado no comparte recursos

Respecto a la ejecución

- Creador/creado se ejecutan concurrentemente
- Creador espera al que el creado termine

Sus espacios de direcciones son:

- Clonados: se copia el espacio del creador para el creado: comparten los mismos códigos, datos, .. ej Unix
- Nuevos: el proceso creado inicia un programa diferente al del creado, p.e: Windows2000

Llamada la sistema fork()

- Crea un nuevo proceso, copia casi idéntica del padre.

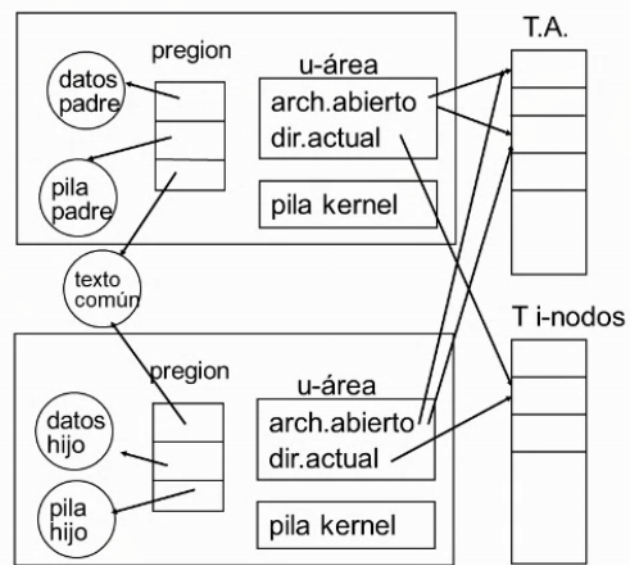
-Devuelve al padre el PID del hijo y al hijo un 0

-Sintaxis:
pid = fork();

- Después del fork, ambos se ejecutan concurrentemente.

Llamadas la sistema exec()

- Ejecuta el programa que se le pasa como argumento
- Existen seis funciones exec que se diferencia únicamente en la forma de pasa los argumentos (execl, execl, execl en vez de dar una lista en ejecución de cada parámetro, el segundo es un array que agrupa las cadenas), execvp(si no se encuentra el camino,vaya a la var path e intente localizar el ejecutable),...
- Si la llamada tiene éxito, el espacio de direcciones del proceso que la inició se ha sustituido completamente por un nuevo programa.
- Hay que asegurar que el nuevo programa se ejecutará correctamente. P.e. puede ser necesario redirigir al entrada o la salida o ambas.



Llamadas exit y wait

- **exit**
 - Pone fin a la ejecución de un proceso
 - Devuelve el estado de finalización al padre y el proceso pasa a estado zombie
- **wait**
 - Espera la terminación del primer hijo
 - Si no ha terminado ningún hijo el proceso (padre) se bloquea

- Si un padre finaliza antes que sus hijos, los hijos conectan al proceso init, no hay problema, porque init es un proceso diseñado para cuando el proceso hijo enganchado, cuando termina el sistema una de las señales envía un señal sit child(algo así) al padre. Y en el manejo de esa señal, es un wait internamente.
- Si un hijo finaliza sin que el padre ejecuta wait, se queda en estado zombie

15. Manejador del reloj

Una de las rutinas que entran en funcionamiento de forma periódica en el SO

Toda máquina unix tiene un reloj hardware que interrumpe al sistema en intervalos de t^o fijos (típicamente se establece el tick de reloj en 10 ms)

Las funciones del manejador de la interrupción de reloj son (no todas en cada tick):

- Reiniciar el reloj (si es necesario)
- Planificar funciones internas del núcleo basadas en temporizadores
- Reunir estadísticas del sistema y de los procesos
- Mantener la hora
- Enviar las señales de alarma a los procesos
- COntrolar la planificación de procesos
- Despertar a otras tareas como al intercambiador y stealer periódicamente

16. Planificador Unix

Colas múltiples con realimentación, la cola se gestiona por RR y entre colas, por prioridad.

Si existe más de un proceso elige el más antiguo , si no hay procesos espera a la interrupción planificarse.

Se introducen clases de planificación:

Hay dos

1) Tiempo compartido (0-59):

- Cambiar prioridad de un proceso en respuesta a eventos específicos relacionados con ese proceso.
- Existe una tabla de parámetros del panificador que define como los distintos eventos cambian la prioridad de un proceso.
- EL quantum depende de la prioridad, suele ser así. Aa mayor prioridad menor quantum porq el sistema entiende q lo procesos de mayor prioridad son aquellos eq entran en funcionamiento menos tiempo+
- Cuando un proceso se bloquea, se el asigna una prioridad a nivel de núcleo(60-99)
- Valor nice entre -20 a +19 (por defecto, 0)

2) Tiempo real (100- 159)

- Los procesos tienen prioridad y quantum fijos especificados explícitamente.
- Solo el superusuario puede tener procesos en esta clase.

Planificador SVR4

17.Señales

- Informan a los procesos de la ocurrencia de eventos
- Los procesos y el núcleo pueden enviar señales
- Categoría
 - 1. Eventos síncronos: errores generados por la ejecución de un proceso. P.e. violación de segmento (SIGSEGV) o instrucción ilegal (SIGILL)
 - 2. Eventos asíncronos: ocurren externamente a la ejecución del proceso pero que tienen alguna relación. P.e terminación de un hijo o bloqueo desde el manejador del terminal tty (SIGHUP)
- Cada señal tiene asociado un número entero positivo
- Envío de señal a un proceso -> el núcleo activa un bit del campo de señales en entrada tabla de procesos
- Un proceso puede bloquearse de dos formas:
 - interrumpible: el suceso que espera no se sabe cuando ocurrirá o si ocurrirá. (ej pulsación de una tecla). Si el proceso está bloqueado de forma interrumpible -> la recepción de señal provoca su desbloqueo
 - no interrumpible: el suceso ocurrirá pronto (ej, fin E/S)
- Un proceso puede recordar diferentes tipos de señales, pero no cuántas de cada tipo ha recibido.
- Hay dos fases en el proceso de señalización:
 - 1) Generación: el núcleo genera señales en base a varios eventos:
 - Excepciones
 - interrupción de terminal
 - control de tareas
 - alarmas
 - notificaciones solicitadas por un proceso
 - 2) Reparto o manejo: cuando el proceso ejecutándose va a pasar de modo supervisor a modo usuario comprueba señales pendientes. Posibles acciones:
 - Abortar o terminar el proceso
 - ignorar
 - ejecutar una función concreta
 - suspender al proceso
 - Reanudar un proceso suspendido
 - Cada señal tiene una acción por defecto que lleva a cabo el núcleo si no se especifica una alternativa.

- Hay señales especial: SIGKILL o SIGTOP
- Para especificar la acción deseada:
 - funcion_anterior = signal(num_señal, funcion)
 - función puede ser:
 - dirección de la función invocar
 - ignorar la ocurrencia de la señal
 - terminar el proceso
- Para enviar una señal: kill(pid, señal)
 - Dependiendo del valor de pid, el núcleo envía la señal a:
 - pid>0 : al proceso con ese nº de pid
 - pid = 0: a todos los procesos del grupo del emisor
 - pid = -1 : a todos los procesos cuyo UID real = UID efectivo del emisor. Si el emisor tiene un UID efectivo de superusuario se manda a todos excepto al proceso 0 y 1.

Planificación Unix

- Cada proceso activo tiene una prioridad de planificación
- El planificador utiliza la política de colas múltiples con realimentación, donde cada cola se gestiona por Round Robin
- El algoritmo que se sigue es:
 - selecciona el proceso de más alta prioridad de aquellos que están en estado apropiado o preparado para ejecutarse en memoria
 - Si existe más de un proceso, elige el más antiguo
 - Si no hay procesos, espera a la siguiente interrupción y después de su tratamiento, intenta planificar un proceso.

Tema 3: Gestión de Memoria

MMU

Intercambio (Swapping)

- Intercambiar procesos entre memoria y almacenamiento auxiliar.
- El almacenamiento auxiliar debe ser un disco rápido con espacio para albergar las imágenes de memoria de los procesos de usuario.
- El factor principal en el tiempo de intercambio es el tiempo de transferencia.
- El intercambiador tiene las siguientes responsabilidades:
 - Seleccionar procesos para retirarlos de MP
 - Seleccionar procesos para incorporarlos a MP
 - Gestionar y asignar el espacio de intercambio

Localización del espacio de intercambio

El espacio de intercambio, hay dos soluciones en un sistema de archivo existentes, que contenga un archivo especial, como windows el swap, que es un archivo que se dedica únicamente a albergar contenidos en memoria, páginas de los procesos que están ejecutándose y no caben en memoria física, y el otro es que exista realmente un espacio de intercambio, que suele ser una partición dedicada al intercambio, su sistema de archivos es muy básico, de división de bloques, no hay casi nada de meta información y solo alberga las páginas que no caben en memoria física.

2. Organización de memoria virtual

El tamaño del programa, los datos y la pila puede exceder la cantidad de memoria física disponible para él.

Se usa un almacenamiento a dos niveles:

- Memoria oral: parte del proceso necesario en un momento dado.
- Memoria secundaria: espacio de direcciones completo del proceso

Aunque hubiera que ejecutar un único programa, cuando éste se convierta en proceso, incluso si no cupiera en memoria física, la idea es que el sistema sería capaz de ejecutarlo, este ejemplo es extremo.

El sistema da la sensación a los procesos de que tienen memoria infinita. Realmente lo que se está haciendo es usar memoria a dos niveles.

Es necesario:

- saber que se encuentra en memoria física
- una política de movimiento entre MP y MS

Además, la memoria virtual:

- resuelve el problema del crecimiento dinámico de los procesos
- puede aumentar el grado de multiprogramación.

Se resuelve más cosas, resuelve problemas para que los procesos puedan crecer. Además el núcleo nos da para en ciertos segmentos, por ejemplo para el segmento de datos o el heap, podremos aumentar y pedirle al kernel de forma explícita una ampliación de un segmento.

Unidad de Gestión de Memoria

- La MMU (Memory Management Unit) es un dispositivo hardware que traduce direcciones virtuales a direcciones físicas. Este dispositivo está gestionado por el SO.
- En el esquema MMU más simple, el valor del registro base se añade a cada dirección generada por el proceso de usuario al mismo tiempo que es enviado a memoria.
- El programa de usuario trata sólo con direcciones lógicas; éste nunca ve direcciones reales

Es un dispositivo hardware implementado para la traducción rápida de una dirección virtual en el momento en el cual el sistema está accediendo a una dirección de memoria, al estar en el hardware es más rápida. Y ahí también se implementará parte de la protección, el núcleo tiene dos rutinas que lanza, dispara el manejador de falta de protección con motivos de excepción o manipular la falta de validez, es decir la ausencia de una página a la cual va a acceder un proceso, que no está en memoria física.

Segmentación y paginación

Tabla de regiones por procesos, es la tabla de segmento, tiene una entrada por cada segmento que tiene un proceso. Por ej: de texto, que es el código

Desde la tabla de regiones globales, para cada segmento existe una tabla de páginas. A través de las entradas de regiones globales, cada entrada tiene su parte de tablas de páginas.

El bit de presencia, o bit de validez, es que realmente esa página está cargada y operativa en memoria física. Si una página no tiene marco de página, es que esa página no está accesible en memoria física, si se intentara acceder se produciría una falta de validez.

Cual es el problema de la paginación?

Supongamos una arquitectura de 32 bits, lo típico es que el tamaño de página sea de 4 Kbytes (2^{12} bytes), quiere decir de los 12 últimos de esos 32 se dedican a direccionar dentro de una página.

- tamaño del campo de desplazamiento = 12 bits
- tamaño número de páginas virtual = 20 bits

Los otros 20 son los que se dedican al número de páginas, eso quiere decir que un proceso puede llegar a tener como máximo 2^{20} páginas = 1 048 576, 1GB, lo que es muy pesado.

Solución: paginación multinivel

De los 20 bits se dividen en dos subcampos, los de la izq son referentes a una paginación a primer nivel y los de la dcha a una página a segundo nivel,

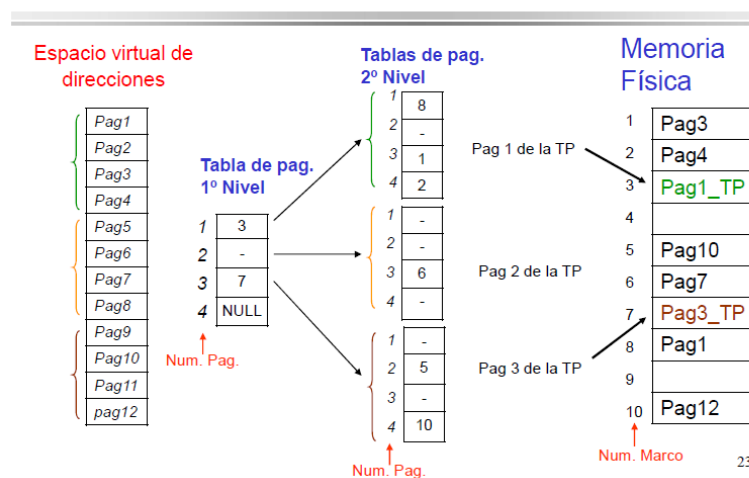
Los primeros bits llevan a una tabla, localizamos la entrada a la que tenemos que acceder y esa nos lleva a otra tabla a un segundo nivel, cuyos bits ayudarán a localizar la entrada correspondiente de esta subtabla

Cuando para una entrada a primer nivel no haya ninguna entrada válida para direccionar memoria de un proceso, la segunda subtabla no hay que asignarla.

Y así ajustamos el tam de página al tam de memoria que requiere cada proceso.

Ej:

Ejemplo: Esquema de paginación a dos niveles



2.3 Segmentación

Esquema de organización de memoria que soporta mejor la visión de memoria del usuario: un programa es una colección de unidades lógicas- segmentos- p.ej. procedimientos, funciones, pila, tabla de símbolos matrices etc

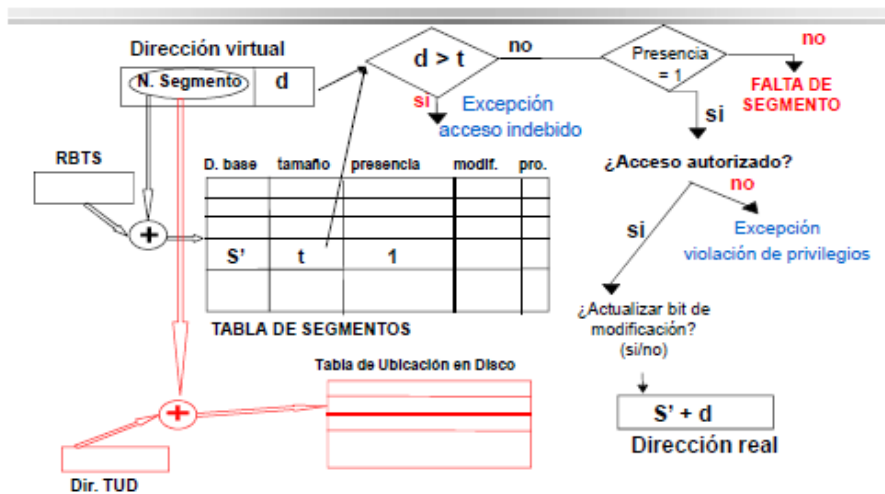
Tabla de segmentos

- Una dirección lógica es un tupla:
 - <num_de_segmento, desplazamiento>
- La tabla de Segmentos aplica direcciones bidimensionales definidas por el usuario en direcciones físicas de una dimensión. Cada entrada de la tabla tiene los siguientes elementos (aparte de presencia, modificación y protección):
 - **base:** dirección física donde reside el inicio del segmento de memoria
 - **tamaño:** longitud del segmento

Implementación de la tabla de segmentos

- La tabla de segmentos se mantiene en memoria pral.
- El Registro Base de la Tabla de segmentos (RBTS) apunta a la tabla de segmentos (suele almacenarse en el PCB del proceso)
- El Registro Longitud de la Tabla de Segmentos (STLR) indica el número de segmentos del proceso; el nº de segmento s , generado en una dirección lógica, es legal si $s < \text{STLR}$ (suele almacenarse en el PCB del proceso)

Esquema de traducción



28

Segmentación Paginada

- La variabilidad del tamaño de los segmentos y el requisito de memoria contigua dentro de un segmento complica la gestión de MP y MS.
- Por otro lado, la paginación simplifica eso pero complica más los temas de compartición y protección (estos van mejor en segmentación)
- Algunos sistemas combinan ambos enfoques, obteniendo la mayoría de las ventajas de la segmentación y eliminando los problemas de una gestión de memoria compleja

3. Labores de gestión de memoria de virtual que hacen el núcleo de los SO

Introducción

Gestión de memoria virtual con paginación

Criterios de clasificación respecto a :

- Políticas de asignación de marcos de páginas en memoria pral a procesos:
 - Fija
 - Variable
- Políticas de búsqueda:
 - Paginación por demanda: cuando un proceso necesita una página es en ese momento cuando el sistema se pone en marcha, como consecuencia de la demanda.
 - Paginación anticipada:

- Políticas de sustitución: cuando una página tenga que venir a memoria principal y no haya marco de página suficiente, hay que reemplazar algunos de los contenidos de algún marco de página por otros contenidos de la nueva página.

Independientemente de la política de sustitución utilizada existen ciertos criterios que siempre deben cumplirse:

- Páginas "limpias" frente a "sucias": Se encarga de que si una página estuvo en disco, se trajo a memoria principal y no ha sufrido cambios respecto a la copia que hay en disco, no hace falta volver a transferirla a disco. Pretende minimizar el coste de transferencia.
- Páginas compartidas: se pretende reducir el número de faltas de página
- Páginas especiales: algunos marcos pueden estar bloqueados (ej: buffers de E/S mientras se realiza una transferencia)

Asignación fija

A los procesos se les va a asignar un número de marcos.

- Asignación por igual:
 - Se asignan el mismo número de marcos a todos los procesos
 - Si hay m marcos, y n procesos. A cada proceso se le asignan m/n marcos
- Asignación según tamaño:
 - Proporcional al tamaño del proceso
 - s_i = tamaño de p_i
 - S = sumatoria de s_i
 - m = número total de marcos
 - la asignación, a_i para p_i es:
 - $a_i = (s_i/S) * m$

Paginación por demanda vs anticipada

es más interesante la paginación por demanda, ventajas:

- Se garantiza que en MP solo están las páginas necesarias en cada momento
- La sobrecarga de decidir qué páginas llevar a MP es mínima
- Las ventajas de la paginación anticipada son:
 - Se puede optimizar el tiempo de respuesta para un proceso pero los algoritmos son más complejos y se consumen más recursos.

Rendimiento de la paginación por demanda

- Sea p la tasa de falta de página; $p=0$ no hay faltas de páginas o $p=1$, toda referencia es una falta. Por tanto: $0 \leq p \leq 1$
- $TAE = (1-p) * \text{acceso_a_memoria} + p * \text{acceso_a_disco}$

(sobrecarga falta de página + sacar fuera una página + traer la página + sobrecarga de rearranque)

Influencia del tamaño de página

- Cuanto más pequeñas
 - Aumento del tam de las tablas de páginas
 - Aumento del nº transferencias entre MP y Disco
 - Reducen la fragmentación
- Cuanto más grandes
 - Grandes cantidades de información que no serán usadas están ocupando MP
 - Aumenta la fragmentación interna
- Búsqueda de un equilibrio

3.2 Algoritmos de sustitución

- Podemos tener las siguientes combinaciones
 - asignación fija y sustitución local
 - asignación variable y sustitución local
 - asignación variable y sustitución global
- Veremos distintos algoritmos de sustitución y nos basaremos (por simplicidad) en que se utiliza una política de asignación fija y sustitución local
- Cadena de referencia $w = r_1, r_2, r_3 \dots$ secuencia de número de páginas referenciadas por un proceso durante su ejecución.

A) Algoritmo Óptimo

Se sustituye la página que no será objeto de ninguna referencia posterior o que se referencia más tarde.

- 4 marcos de página
- Problema: debemos tener un conocimiento perfecto de la cadena de referencia
- Se utiliza para medir cómo de bien se comportan otros algoritmos
- Faltas de páginas: 6

1,2,3,4,1,2,5,1,2,3,4,5

[illegible]

B) Algoritmo FIFO

- Se sustituye la página por orden cronológico de llegada a MP (la página más antigua)
 - 4 marcos de página
 - Sufre de la Anomalía de Belady: “más marcos no implican menos faltas de páginas
 - Faltas de páginas: 10

1,2,3,4,1,2,5,1,2,3,4,5

1	1	1	1	1	1	5	5	5	4	4
2	2	2	2	2	2	1	1	1	1	5
3	3	3	3	3	3	2	2	2	2	
4	4	4	4	4	4	3	3	3		

* * * * * * * * * *

C) Algoritmo LRU

- Se sustituye la página que fue objeto de la referencia mas antigua (Least Recently Used)
 - 4 marcos de página
 - Falta de página: 8
 - Mayor coste
 - Implementación del algoritmo
 - Con contadores
 - con pila

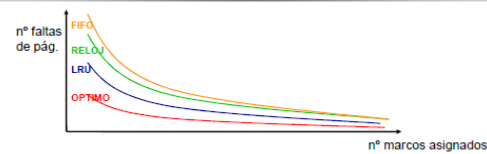
Implementación

- **LRU con contador**
 - Cada entrada de la tabla de páginas tiene un contador. Cada vez que se referencia la página, se copia el tiempo del reloj en el contador.
 - Cuando necesitemos cambiar una página, se miran todos los contadores y se elige la que tiene el menor tiempo.

D) Algoritmo del reloj

- Cada página tiene asociado un bit de referencia R
- Cuando una página llega a memoria se establece a 1; en cada referencia el hardware lo pone a 1.
- Los marcos de página se representan por una lista circular y un puntero a la página visitada hace mas tiempo
- Selección de una página: Consultar un bit R del marco actual R=1?
 - Si, R=0, ir al siguiente marco y vuelve a consultar
 - No, seleccionar para sustituir e incrementar posición

Comparación



- Conclusión:
 - » Infiere más la cantidad de MP disponible que el algoritmo de sustitución usado

3.3 Comportamiento de los programas

- Viene definido por la secuencia de referencias a páginas que realiza el proceso
- Importante para maximizar el rendimiento del sistema de memoria virtual(TLB, alg, sustitución,..)

Propiedad de localidad

- Distintos tipos
 - **Temporal:** una posición de memoria referenciada recientemente itne un probabilidad alta de ser referenciada en un futuro próximo (ciclos, rutinas, variable globales,..)
 - **Especial:** Si cierta posición de memoria ha sido referenciada es altamente probable que las adyacentes también lo sean (array, ejecución secuencia....

Conjunto de trabajo

- Observaciones
 - Mientras el conjunto de páginas necesarias puedan residir en MP, el nº de faltas de páginas no crece mucho
 - Si eliminamos de MP páginas de ese conjunto , la activación de paginación crece mucho
- Conjunto de trabajo (Working Set) de un proceso es el conjunto de páginas que son referenciadas frecuentemente en un determinado intervalo de tiempo

Se define el conjunto de trabajo en el momento t con "ancho de ventana" Δ como:

$$WS(t, \Delta) = \text{páginas referenciadas en el intervalo de tiempo } (t - \Delta, t]$$

47

- Propiedades
 - Los conjuntos de trabajo son transitorios
 - No se puede predecir el tam futuro de un conjunto de trabajo
 - Difieren unos de otros sustancialmente

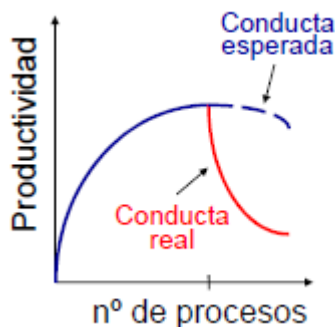
Teoría del Conjunto de Trabajo

- Un proceso solo puede ejecutarse si su conjunto de trabajo está en memoria pral.
- Una página no puede retirarse de memoria pral si está dentro del conjunto de trabajo del proceso en ejecución

3.4 Hiperpaginación

Al aumentar el nº de procesos existe un valor a partir del cual se produce una caída brusca de las prestaciones:

- se observa que ningún proceso adquiere tiempo de CPU
- hay un fuerte aumento del nº de intercambio de págs



- Formas de evitar la hiperpaginación:
 - Asegurar que cada proceso existente tenga asignado un espacio en relación a su comportamiento -> Algoritmo de asignación de variables
 - Actuar directamente sobre el grado de multiprogramación -> Algoritmos de regulación de carga

3.5 Algoritmos de asignación variable y sustitución global:

A) Algoritmo basado en el modelo del WS

Parámetro del algoritmo: ancho de ventana V

En cada momento t en que se hace referencia a una posición de memoria, se determina el conjunto de trabajo de la siguiente forma:

WS = páginas referenciadas en el intervalo (v-V, t]

** solo estas páginas se mantienen en memoria pral

B) Algoritmo FFP(Frecuencia de Falta de Página)

Definición: intervalo entre dos faltas de página

```
Tiempo actual – tiempo de la falta de página anterior
IF intervalo > valor umbral L // L es un parámetro del algoritmo
THEN //el proceso tiene demasiados marcos de página, hay que quitarle.
se retiran de memoria principal todas las páginas no referenciadas entre el
momento actual y el momento de la falta de página anterior

ELSE // demasiadas faltas de página, el proceso tiene pocos marcos.
se asigna al proceso un marco de página más para albergar la página que
ha originado esta falta de página
```

LA VELOCIDAD DE FP SIRVE PARA CONOCER SI EL Nº DE MARCOS ASIGNADOS AL PROCESO ES GRANDE O PEQUEÑO

Tema 4 : Gestión de Archivos

Desde el punto de vista del usuario, los sistemas de ficheros son una de las partes más importantes de los SO. El sistema de ficheros permite a los usuarios crear ficheros (colecciones de datos) con propiedades deseables:

- **Existencia a largo plazo:** los ficheros no desaparecen cuando un usuario se desconecta, sino que se guardan en disco u otro almacenamiento secundario
- **Compartible entre procesos:** los ficheros tienen nombres y puede tener permisos asociados que permitan gestionar la compartición.
- **Estructura:** los ficheros se pueden organizar en estructuras jerárquicas para reflejar las relaciones entre los mismos, dependiendo de su estructura interna.

Archivo

Concepto

Colección de información relacionada y almacenada en un dispositivo de almacenamiento secundario

Espacio de direcciones lógicas contiguas

- Estructura interna (lógica)
 - Secuencia de bytes: el tipo del archivo determina su estructura (texto -> caracteres, líneas y páginas, código fuente -> secuencia de subrutinas y funciones)
 - secuencia de registros de longitud fija
 - secuencia de registros de longitud variable
- Tipos de archivos: regulares, directorios, de dispositivos
- Formas de acceso: secuencial, aleatorio, otros

Atributos (metadatos)

- Nombre : única información en formato legible
- Tipo: cuando el sistema soporta diferentes tipos
- Localización: información sobre su localización en el dispositivo
- Tamaño: tamaño actual del archivo
- Protección: controla quién puede leer, escribir y ejecutar
- Tiempo, fecha e identificación del usuario: necesario para protección , seguridad y monitorización

Operaciones sobre archivos

Cualquier sistema de ficheros debe proporcionar una forma de almacenar los datos organizados como ficheros además de un serie de funciones que se pueden llevar a cabo sobre ficheros

- **Gestión sobre archivos**
 - **Crear:** se define un nuevo fichero y se posiciona en la estructura de ficheros.
 - **Borrar:** se elimina un fichero de la estructura de ficheros y se destruye
 - Renombrar
 - Copiar
 - Establecer y obtener atributos

- **Procesamiento**
 - Abrir: un fichero existente se declara abierto por un proceso, permitiendo que dicho proceso realice funciones sobre él.
 - Cerrar: un proceso cierra un fichero, de forma que no puede realizar ciertas funciones sobre él (a no ser que vuelva a abrirlo)
 - Leer: Un proceso lee todos los datos de un fichero o bien una parte de ellos.
 - Escribir (modificar, insertar, borrar información): un proceso actualiza un fichero, bien añadiendo nuevos datos y, por tanto, expandiendo el tamaño, o bien cambiando valores de datos existentes.

Un sistema de ficheros suele mantener unos atributos asociados al fichero. Estos incluyen propietario, tiempo de creación, tiempo de última modificación, privilegios de acceso, etc

Estructura de un fichero

- **Campo:** es el elemento básico de los datos. Un campo individual contiene un único valor que se caracteriza por su tipo de dato y longitud. Dependiendo del diseño del fichero, el campo puede tener una long fija o variable. Si la longitud es variable el campo estará formado por dos o tres subcampos:
 - Valor real almacenado.
 - nombre del campo
 - longitud del campo (en algunos casos)

Si no existe este tercer subcampo, la longitud del campo se indicará mediante símbolos de demarcación especiales entre campos.

- **Registro:** colección de campos relacionados que se pueden tratar como una unidad por determinadas aplicaciones. Pueden tener longitud fija o variable. Si la longitud es variable, alguno de los campos tiene longitud variable o bien el número de campos puede cambiar. En este caso, cada campo se acompaña normalmente de un nombre de campo. El registro completo incluye normalmente un campo longitud.
- **Fichero:** colección de campos similares. Un fichero se trata como una entidad única por parte de usuario y aplicaciones. Los ficheros se pueden crear, borrar y referenciar por nombre. Las restricciones de control de acceso se aplican normalmente a nivel de fichero, es decir, solo los usuarios autorizados podrán acceder y/o manipular el archivo. En sistemas más sofisticados, estos controles se pueden realizar a nivel de registro o incluso a nivel de campo.
- **Bas de datos:** colección de datos relacionados. Esta relación debe ser explícita y la BD será utilizada por varias aplicaciones diferentes. Está formada por uno o más tipos de ficheros, por lo que, normalmente, hay un sistema de gestión de base de datos separado del SO, aunque utiliza los programas de gestión de ficheros.

No todos los archivos utilizan esta estructura. Por ejemplo, un programa se almacena como un fichero, pero no tiene campos físicos, registros y otras estructuras.

Directorios

Los directorios contienen información sobre los ficheros, incluyendo atributos, ubicación y propiedad. Gran parte de esa información, especialmente la que concierne al almacenamiento, es gestionada por el SO. El directorio es a su vez un fichero accesible por varias rutinas de gestión de ficheros. Aunque parte de la información de los directorios está disponible para usuarios y aplicaciones, se suele proporcionar indirectamente por las rutinas del sistema.

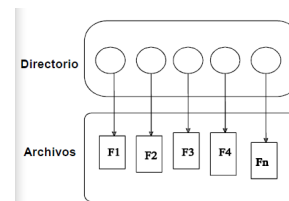
Desde el punto de vista del usuario, el directorio proporciona una proyección entre los nombre de ficheros, conocidos por los usuarios, las aplicaciones y los propios ficheros. Por tanto, cada entrada de fichero incluye su nombre. Prácticamente todos los sistemas tratan con diferentes tipos de ficheros y distintas organizaciones de estos. Esta información también se proporciona. Una importante categoría de información sobre cada fichero trat sobre el almacenamiento, incluyendo su ubicación y tamaño. En sistemas compartidos, es también importante dar información necesaria para el control de acceso a los ficheros. Típicamente, un usuario es el propietario del fichero y puede conceder ciertos privilegios a otros usuarios. Finalmente, la información de uso se utiliza para gestionar la utilización actual del fichero y registrar el historial de uso.

Estructura

- Colección de nodos conteniendo información acerca de todos los archivos ->

Organización

- Tanto la estructura de directorios como los archivos residen en el almacenamiento secundario.

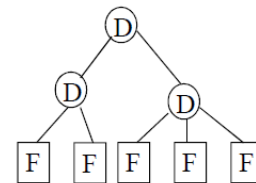
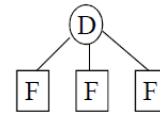


La organización (lógica) de los directorios debe proporcionar:

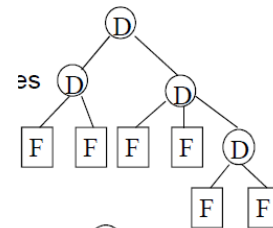
- Eficiencia: localización rápida de un archivo
- Denominación: nombrar los ficheros, y poder organizarlos como mejor le venga a los usuarios
 - Dos usuarios pueden tener el mismo nombre para diferentes archivos
 - El mismo archivo puede tener varios nombres
- Agrupación: basada en que los directorios a su vez pueden contener otros subdirectorios. Agrupar los archivos de forma lógica según sus propiedades (Ej: todos los programas en C)

Casos

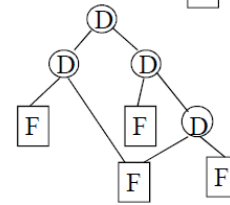
- A un único nivel
 - Problema de denominación
 - Problema de agrupación
- A dos niveles
 - Aparece el concepto de, nombre de camino
 - Diferentes usuarios puede tener archivos con igual nombre
 - No hay posibilidad de agrupación



- En árbol
 - Necesidad de búsquedas eficientes
 - Posibilidad de agrupación
 - Directorio actual (de trabajo)
 - Nombres de camino absolutos y relativos



- En grafo
 - Compartición de subdirectorios y archivos
 - Estructura más evolucionada
 - Más flexibles y complejos
 - Permite compartir directamente un archivo sin tener que recorrer su localización original



Protección

Básicamente consiste en proporcionar un acceso controlado a los archivos

- lo que puede hacerse
- por quién

Tipos de acceso

- Leer
- Escribir
- Ejecutar
- Añadir
- Borrar
- Listar

Protección : Listas y Grupos de Acceso

- Principal solución a la protección: hacer el acceso dependiente del identificativo del usuario
- Las listas de acceso de usuarios individuales tiene el problema de la longitud
Para cada archivo tiene un lista de lo usuarios individuales que pueden acceder a ese archivo, y con que permisos, estas listas pueden tener un problema de longitud.
¿Qué hace Unix? condensa esa lista, la reduce a clasificar a los usuarios entre grupos pral:

- Solución con clases de usuario
 - propietario
 - grupo
 - público

Después hay que hacer una gestión adicional, gestionando los grupos que usuarios pertenecen a cada grupo.

Un archivo no puede pertenecer a más de grupo, un usuario si puede pertenecer a más de un grupo.

- Propuesta alternativa: Asociar un password con el archivo. Problemas:
 - Recordarlo
 - Si solo se asocia un password -> acceso total o ninguno

Semánticas de consistencia

Especifican cuándo las modificaciones de datos por un usuario se observan por otros usuarios

Ej:

1. Semántica de Unix

- La escritura en un archivo es directamente observable
 - Unix permite que dos usuarios que acceden a un archivo puedan compartir el puntero de escritura y lectura.
 - El núcleo aplica bloqueos a archivos compartidos. Hay ciertos momentos en los que no puede permitir que se entremezclan acciones, ¿Entonces cuál es la semántica de unix? Dos operaciones que ocurren concurrentemente sobre el mismo archivo, o parte del archivo, son llamadas a sistema. Entonces, en principio hasta que la primera llamada no haya acabado no se atiende la segunda.
 - Si queremos hacer varias operaciones de lectura y escritura consecutivas sobre un archivo, de forma que todas sean consistentes, como se consigue?
- Con cerrojos.**
- Existe un modo para que los usuarios compartan el puntero actual de posicionamiento en un archivo

2. Semánticas de sesión (Sistemas de archivos de Andrew)

- La escritura en un archivo no es directamente observable
- Cuando un archivo se cierra, sus cambios sólo se observan en sesiones posteriores.

3. Archivos inmutables

- Cuando un archivo se declara como compartido, no se puede modificar.

Sistemas de archivos nfs(sistema de archivos en red).

Funciones básicas del Sistema de Archivo

- Tener conocimiento de todos los archivos del sistema
- Controlar la compartición y forzar la protección de acceso a los archivos.
- Gestionar el espacio del sistema de archivos: El sistema tiene que controlar qué espacios hay libres en el sistema de archivos para poder asignarlos.
 - Asignación/liberación del espacio en disco
- Traducir las direcciones lógicas del archivo en direcciones físicas del disco.
 - Los usuarios especifican las partes que quieren leer/Escribir en términos de direcciones lógicas relativas al archivo.

Supongamos que el sistema de archivos tiene bloques de 4KB, 4096 byte, supongamos que accedemos la byte 5000, el sistema tiene que saber que este cae en el segundo bloque, hay que traducir el número de la posición lógica del byte que referenciamos al bloque físico que le corresponde en disco.

Estructura del Sistema de Archivos

Un sistema de archivos posee dos problemas de diseño

1. Definir la interfaz que debe ver el usuario del sistema de archivos:
 - a. definir un archivo y sus atributos
 - b. definir las operaciones permitidas sobre un archivo
 - c. definir la estructura de directorios
 - d. quien va a crear enlaces. **Hay enlaces que solo pueden crear root, por que o cuales?** Enlaces duros a directorios, porque es algo peligroso, muchas de las herramientas que buscan archivos en directorios y hay gestiones sobre ellos, el sistema puede entrar en ciclos infinitos, por eso root es el único que tiene el permiso para hacerlo.
2. Definir los algoritmos y estructuras de datos que deben crearse para establecer la correspondencia entre el sistema de archivos lógicos y los dispositivos físicos donde se almacenan.

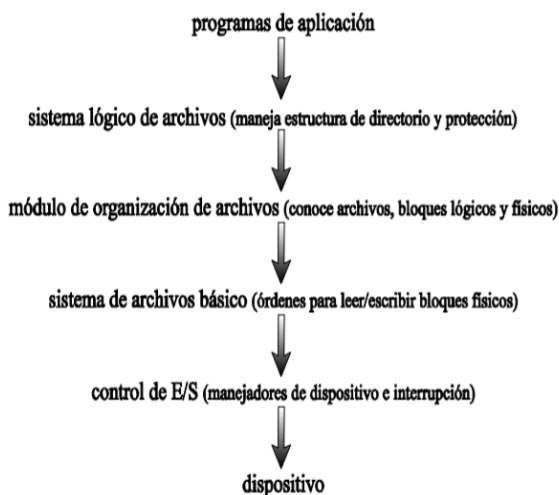
● **Organización en niveles (capas)**

El subsistema de archivos, es el subsistema dentro del núcleo de Unix que tiene más código. Normalmente se estructura por capas. Los **programas de aplicación** lo primero que interaccionan a través de la llamada al sistema es con la parte del sistema lógico, es la que tiene que ver con la estructura de directorio, comprobar temas de protección...

La segunda capa, **módulo de organización de archivos**, conoce los metadatos de un archivo, como manejarlos, y sabe también de la conversión de bloques lógicos a físicos. En esta capa también conoce el sistema cual es el espacio libre en el almacenamiento, conoce los archivos y la parte que no está libre.

Sistema de archivos básicos, genera todo lo que tiene que ver con interaccionar con el manejador del dispositivo, emitir órdenes en base a una especificación. Estos sistemas incluyen también una política de planificación de mejora de rendimiento en el acceso a lo que es emitir operaciones que obligan al acceso y a llevar a cabo acciones sobre el dispositivo.

Igual que hemos dicho q en procesos q es una abstracción el sistema tiene que mantener datos, metadatos de un proceso, manejar señales, procesos hijos, hebras, pues aquí como archivos es realmente una abstracción equivalente a procesos equivalente a procesos pero referente a almacenamiento, también tiene un bloque de control de archivo. **Como se accede al bloque?** Por un descriptor de archivo que es lo que devuelve el sistema cuando hacemos la primera aplicación típica sobre un archivos regular que es, abrir el archivo, que devuelve un entero, q es el identificador para ese proceso, que le lleva al bloque de control de ese archivo.



- Por eficiencia, el SO mantiene una tabla indexada (por descriptor de archivo) de archivos abiertos

- Bloque de control de archivo: estructura con información de un archivo en uso

Métodos de Asignación de espacio: Contiguo

De las labores más importante que hace el sistema es la asignación de espacio a los archivos. Hay varios métodos, que se suele clasificar en método contiguos y no contiguos.

Contiguo

Todos los bloques que se asignan a un archivo para que albergue sus datos, en el dispositivo está dispuesto de forma contiguos.

- Ventajas
 - Asignar ahí espacio es sencillo, solo se necesita saber dónde comienza el archivo(nº bloque) y su tamaño.
 - Este método es muy bueno para un acceso secuencial y accesos directos.
- Desventajas
 - Inicialmente no se suele conocer su tam, por lo que tenemos que garantizar que donde vaya el archivo hay suficiente espacio para almacenar todos sus datos. Por tanto es posible que desaproveches espacio, provocando fragmentación externa. Para evitar esto se hace la compactación.
- Asociación lógica a física

Calcular la dirección física que el corresponde a un lógica, es esa dir lógica la dividimos por el tam de bloque, el cociente es el num de bloques donde cae el byte que referenciamos y el resto es el desplazamiento.

No contiguo

- **Enlazado:** los bloques no tienen que estar contiguos en disco, pero para localizar todos los bloques hay que guardar información de cuáles son todos los archivos que pertenecen al bloque. Esto se monta como una lista enlazada. Cada bloque tiene una reserva de memoria para guardar un puntero que apunta al siguiente bloque físico donde está el siguiente bloque lógico del archivo. El último bloque del archivo tendrá un valor especial que indicará que es el final del archivo.

Ventajas:

- Evita fragmentación externa.
- El archivo puede crecer dinámicamente cuando hay bloques de disco libres
-> no es necesario compactar
- Basta almacenar el puntero al primer bloque del archivo

Desventajas:

- El acceso directo no es efectivo, hay que recorrer secuencialmente todos los bloques hasta encontrar el byte que estamos buscando, porque no sabemos de partida cuál es la referencia del bloque físico que corresponde la dirección lógica que estamos buscando.
- El espacio requerido para los punteros.. Solución: agrupación de los bloques físico (clusters). Un cluster (o bloque lógico, Unix) actualmente suelen manejar un tamaño de 4KB, suponiendo un disco estándar de 512 bytes el sector, son 8 bloques físicos.
- Si cae el sistema y se modifica el puntero perderíamos parte de los archivos. Solución: construir lista doblemente enlazadas.

- **Asociación lógica a física**

Dirección lógica de un byte en concreto

Dirección lógica (DL)/511 -> cociente: número del bloque dentro de la cadena, resto+1: desplazamiento

El primer byte se reserva para el puntero

******La información de los punteros se guardan en una estructura en el sistema de archivos, a su comienzo, en una tabla específica reservada para albergar esta información, no dentro de cada bloque.

Quién utiliza métodos enlazados: sistema de archivos que tiene que ver con Microsoft, por ejemplo. Lo que es el sistema de archivos FAT se basa en un método enlazado, es una variación. Qué hace la FAT:

- Reserva una sección del disco al comienzo de la partición para la FAT
- Contiene una entrada por cada bloque del disco y está indexada por número de bloque de disco
- Simple y eficiente siempre que esté en caché.
- Para localizar un bloque solo se necesita leer en la FAT -> se optimiza el acceso directo.
- Problema: pérdida de punteros -> doble copia de la FAT.

¿Cómo de grande tiene que ser la FAT, cuantas entradas tiene que tener?

La partición no solo tiene bloques de datos, consta también de otro espacio que tiene tantas entradas como bloques asignados en la partición.

Una fat de 32, dice que cada entrada son 32 bits, las direcciones son de 4 byte. Entonces, una tabla FAT de 32 es el número de entradas del bloque asignado en esta partición por el tam de cada partición, 4 byte.

- **Indexado**

- Consiste en un bloque índice, un bloque para cada archivo, que contiene los punteros que llevan a los bloques de datos de ese archivo.
- El directorio tiene la localización a este bloque índice y cada archivo tiene asociado su propio bloque índice
- Para leer el i-esimo bloque buscamos el puntero en la i. esima entrada del bloque indice
- Ventajas:
 - Mejor acceso directo que el método enlazado
 - No produce fragmentación
 - Se utiliza actualmente para todos los sistemas de archivos grandes

Los indexados son más potentes, mejor estructurados, permiten montar listas de control de acceso para el tema de protección. Pero es más costoso.

Desventajas:

- posible desperdicio de espacio en los bloques de índices: para cada archivo que tam le damos a este bloque? Si le damos un pequeño , con pocas entradas, el archivo no puede ser muy grande. Si tiene un tam de bloque más grandes, con muchas entradas, y el sistema tiene archivos pequeños, que ocupan solo las primeras entradas y el resto están libres, con lo cual provoca fragmentación internas. Entonces cuál es el tamaño ideal? Es un dilema a resolver.
 - Para sol: se montan bloques de índices enlazados, donde la última entrada apunta a otro bloque de índices.
 - O bloques índices con estructura multinivel:
 - El acceso a disco es mayor. Algunos sistemas dejan algunos bloques índices cacheados en memoria oral. Unix usa un esquema combinado

Unix(s5fs)

Dentro del I-nodo: hay una serie de campos:

- cuando se creó
- última modificación
- último acceso
- propietario
- tam de byte
- grupo
- permisos de propietario y grupo

El resto de lo que ocupa este i nodo tiene que ver con las entradas para localizar su bloque de datos. Y utiliza un método combinado porque reserva las diez primeras entradas para direccionar directamente bloques de datos, cada puntero llega a un bloque físico donde están los datos del archivo. La onceava lleva a otro bloque índice, que a su vez llevan a otros bloques de datos.

Ventajas:

- los bloques índice se asignan cuando son necesarios.
- Llamada al sistema para acceder al i-nodo: stat.st_mode tipo de archivo y los 9 permisos que tiene que ver con el propietario, grupo y permisos.

Gestión de espacio libre

- El sistema mantiene una lista de los bloques que están libres: lista de espacio libre(aunque no siempre se implementa como una lista)
- La fat no necesita ningún método: porque todos los no asignados tendrán también un valor especial, entonces solo hay que buscar en la fat secuencialmente esos valores, que indican que no están asignados.
- A pesar de su nombre, la lista de espacio libre tiene diferentes implementaciones:
 1. Mapa o vector de bits:
 - Cada bloque se representa con un bit (0 bloque libre; 1 bloque ocupado)
 - Fácil encontrar un bloque libre con bloques libres consecutivos. Algunas máquinas tienen instrucciones específicas
 - Fácil tener archivos en bloques contiguos
 - Ineficiente si no se mantiene en memoria principal.
 2. Listas enlazada
 - Enlaza todos los bloques libres del disco, guarda un puntero al primer bloque en un lugar concreto
 - No desperdicia espacio, el propio espacio libre es el que alberga los punteros al siguiente bloque libre.
 - Relativamente ineficiente. hay que pasar de un bloque a otro secuencialmente.
 3. Lista enlazada con agrupación
 - Cada bloque de la lista almacena n-1 direcciones de bloques libres.
 - Obtener muchas direcciones de bloques libres es rápido
 4. Cuenta
 - Cada entrada de la lista: una dirección de bloque libre y un contador del nº de bloques libres que le sigue.

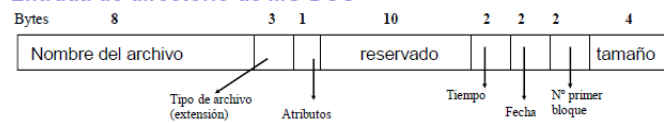
Implementación de Directorios

Son archivos especiales, albergan metadatos .
Contenido de una entrada de directorio

Casos:

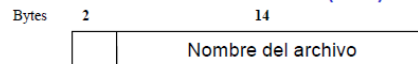
(a) Nombre de Archivo + Atributos + Dirección de los bloques de datos (DOS)

Entrada de directorio de MS-DOS



(b) Nombre de Archivo + Puntero a una estructura de datos que contiene toda la información relativa al archivo (UNIX)

Entrada de directorio de UNIX (s5fs)



Esta estructura es variable, delante hay un campo que indica el tam total de la entrada, y otra que indica el tam en byte del campo nombre. Y ocupa 4K

Cuando se abre un archivo:

- El SO busca en su directorio la entrada correspondiente
- Extrae sus atributos y la localización de sus bloques de datos y los coloca en un tabla me memoria pral

Implementación de Directorios

- Posibilidad respecto a la implementación
 - Lista línea:
 - Sencillo de programar
 - Consume tiempo en las creaciones, búsquedas, .. si no se utiliza una caché software.
 - Tabla hash
 - Decrementa el tiempo de búsqueda
 - Dificultades:
 - Tam fijo de la Tabla hash.
 - Dependencias de la función hash sobre el tam de la tabla.
 - Necesita previsión para colisiones.
- Implementación de archivos compartidos (o enlace):
 - **Enlaces simbólicos:** rutas hacia un archivo. Se construye en base a que el sistema al dar de alta un enlace estamos especificando un ruta que apunta a un nuevo nombre de archivo, que va a apuntar a otro archivo etc etc **Donde se guarda esta ruta?** Son archivos especiales del sistema, codificados en el campo `st_mode`, hay cierta codificación de los bits que especifican que es un enlace simbólico, lo que quiere decir que no tiene datos normales, si no que lo que guardan es un metadato.
 - Se crea una nueva entrada en el directorio, se indica que es de tipo enlace y se almacena el camino de acceso absoluto o relativo del archivo al cual se va a enlazar.

- Se puede usar en entornos distribuidos.
- Gran número de acceso.

Enlaces simbólicos rápidos, que empezó a implementar Linux

- f
 - Enlace duro:
 - Se crea una nueva entrada en el directorio y se copia la dirección de la estructura de datos con la información del archivo.
 - Problema al borrar los enlaces: solución -> Contador de enlaces.

En linux cada fichero y cada carpeta del SO tiene asignado un numero entero unívoco, llamado inodo. la información que almacena cada uno de los inodos es la siguiente:

- Los permisos del archivo o carpeta
- El propietario del fichero y carpeta
- La pos/ubicación del archivo dentro de nuestro disco duro
- La fecha de creación del archivo o directorio
- etc

Podemos decir que un enlace duro es un archivo que apunta al mismo contenido almacenado en disco que el archivo original.

Por tanto los archivo originales y los enlaces duros dispondrán del mismo inodo y consecuentemente ambos estarán apuntando hacia el mismo contenido almacenado en el disco duro. Un enlace duro no es más que una forma de identificar un contenido almacenado en el disco duro con un nombre distinto al del archivo original. Se podrá realizar un enlace duro si el archivo está en la misma partición del disco duro que pretendemos crear el enlace-

Eficiencia y Rendimiento

Los discos suelen ser el principal cuello de botella del rendimiento del sistema.

La eficiencia depende de la asignación de disco y de la implementación de directorios utilizados.

Para proporcionar mejor rendimiento:

- Caché de disco: secciones de M.P con bloques usados.
- Discos virtuales o discos RAM: almacén temporal. Su contenido es controlado por el usuario.

El propio núcleo también construye en memoria pral un buffer caché dedicada a bloques de disco,q se utilizan para bloques de disco que contienen metadatos de los archivos, los datos de ese inodo se incorporan como archivos mapeados a memoria: los datos del archivo van a páginas especiales del segmento que corresponde a datos que puedan ser compartidos con otros procesos pero que son páginas que se mapean al espacio de direcciones del proceso.

Recuperación

Como los archivos y directorios se mantienen tanto en MP como en disco, el sistema debe asegurar que un fallo no genere pérdida o inconsistencia de datos:

- Distintas formas:
 1. Comprobar consistencia:
 - Compara los datos de la estructura de directorios con los bloques de datos en disco y trata cualquier inconsistencia.
 - Más fácil en listas enlazadas que con bloques índices.
 2. Usar programas del sistema para realizar copias de seguridad (backup) de los datos de disco a otros dispositivos y de recuperación de los archivos perdidos.

Situación de inconsistencia: supongamos que tenemos un método enlazado y el contador de bytes del archivo no coincide con los bloques que tiene asignado.

Un bloque de datos este asignado a un archivo y a su vez en la lista de libre

Estructura de Disco

El sistema intenta verlo como un sucesión de bloques, luego realmente si es un disco tradicional