TEAM GETPPID()
www.wuolah.com/student/TEAM-GETPPID()

TEMA 1.pdf

Resumen Tema 1

2º Sistemas Operativos

Grado en Ingeniería Informática

Escuela Politécnica Superior de Córdoba
UCO - Universidad de Córdoba

SISTEMAS OPERATIVOS. RESUMEN TEMA 1.

Carlos de la Barrera Pérez

NEWS

WUOLAH

LUN 19 nov

Noticias para el mundo universitario.

nº 11. Semana del 19 al 25

6 sitios web para descargar libros gratis de forma legal

Fuente: noticiasuniversia.es

1. Editorial Traficantes de Sueños
La editorial Traficantes de Sueños y un símbolo de los movimientos sociales en Madrid. Todas sus publicaciones tienen copyleft (distribución de copias y versiones modificadas de una obra u otro trabajo) y pueden descargarse en formato .pdf.

2. Bubok
Bubok también es una editorial española que publica bajo demanda y brinda muchos libros gratuitos, y otros .pdf de pago, por deseo de sus autores. Busca en las diferentes categorías y encuentra lo que quieres.

3. Proyecto Gutenberg
Proyecto Gutenberg es una biblioteca de ebooks de dominio público gratuitos extremadamente conocida. En general ofrece libros en formato .epub y .mobi (Kindle).

4. Europeana
Europeana es la gran biblioteca digital de Europa y ofrece numerosos libros en formato digital y en general de dominio público. Todas las obras pueden encontrarse en cada uno de los idiomas oficiales de la Unión Europea.

5. Amazon
Amazon es una de las tiendas de venta online más extensas y respetadas a nivel mundial también ofrece los ebooks gratuitos para Kindle, principalmente las obras clásicas de la literatura en castellano, inglés y francés.

6. Anarres
La editorial argentina Anarres difunde la ideología anarquista de manera gratuita, en formato .pdf.

Las mejores aplicaciones Android para aprender inglés

Fuente: noticiasuniversia.es

Duolingo
Esta app permite a sus usuarios aprender inglés, entre otros idiomas, de forma divertida y completamente gratis.
La aplicación es interactiva y muy original, ya que funciona como un juego en el que perderás vidas por cada respuesta incorrecta, mientras que los aciertos te reportarán puntos con los que subir de nivel.

Busuu
Quizás, lo mejor de Busuu sea su inmensa comunidad formada por personas nativas con las que practicar tu inglés.
Los usuarios también pueden acceder a prácticas de audición, lectura, escritura y conversación con las que poner fin a esa asignatura pendiente con los idiomas.

Babbel
La sencillez y éxito de Babbel reside en la categorización de un máximo de 3.000 palabras en diferentes temáticas.
Además de la posibilidad de ampliar vocabulario, la aplicación permite mejorar la pronunciación mediante diversos ejercicios prácticos.

Voxy
Especialmente destinada a quienes estén preparando las pruebas de certificación de nivel de inglés.
Voxy ofrece ayuda online de tutores nativos y ejercicios para mejorar tu fluidez y pronunciación en tiempo récord.

British Council App
El Instituto Británico ofrece también diversas aplicaciones para aprender inglés según el nivel. El prestigio académico de la institución es una garantía de la calidad de las herramientas.

WUOLAH GIVEAWAY

Con el Black Friday a la vuelta de la esquina nos hemos vuelto un poco locos. A qué estás esperando, ¡entra en Wuolah.com y participa!

1. ¿Qué es un Sistema Operativo?

Un Sistema Operativo es un programa en ejecución que controla el hardware de una computadora, siendo todo lo demás programas del sistema o de aplicación.
De forma genérica, un proceso es un programa cargado en la memoria principal, sin embargo, el procesador no distingue al sistema operativo del resto de procesos.
Encontramos dos tipos de programa:

-Programas del sistema que controlan operaciones propias de la computadora.
-Programas de aplicación que resuelven problemas específicos de los usuarios.

El objetivo de un sistema operativo es ocultar la complejidad del hardware proporcionando al usuario una interfaz que le permite gestionar y utilizar los servicios del sistema, el usuario tiene, por tanto, una visión de una máquina virtual con la que es fácil comunicarse.

1.1. El sistema operativo como gestor/asignador de recursos

Una responsabilidad clave de un S.O. es gestionar los recursos del sistema y planificar el uso de los mismos por parte de los procesos activos. El S.O. ha de decidir como asignar estos recursos a programas y usuarios de forma eficiente.

A la hora de asignar recursos se deben tener en cuenta tres factores:

-Equitatividad: Todos los procesos deben de poder acceder al recurso por el que compiten, evitando así la llamada "inanición" de procesos.

-Respuesta diferencial: El S.O. debe asignar los procesos según sus diferentes requisitos de servicio. Por ejemplo, si un proceso está esperando un dispositivo de E/S, el S.O. puede planificar los procesos que están utilizando dicho dispositivo para que terminen lo antes posible.

-Eficiencia: Se debe maximizar la productividad minimizando el tiempo de respuesta y acomodar a tantos usuarios como sea posible en caso de S.O multiusuario.

1.2. El sistema operativo como máquina virtual

Una de las funciones del sistema operativo es presentar al usuario una máquina virtual, más fácil de utilizar que el hardware subyacente.
Una máquina virtual es aquella que, basada en hardware elemental, presenta una mayor facilidad de uso de la misma de esta forma el usuario ve una abstracción del hardware que entiende órdenes a través de esa máquina virtual.
El S.O. como máquina virtual proporciona los siguientes servicios:

-Creación de nuevos programas: Permite utilizar compiladores, depuradores, editores, etc.… para crear nuevos programas.

-Ejecución de programas: Gestiona todo lo necesario para la ejecución de programas: Cargar el código, los datos en memoria, preparar los dispositivos E/S…

-Operaciones E/S: Controlar todos los dispositivos de E/S de forma transparente al usuario.

-Manipulación y control del sistema de archivos: El sistema operativo tiene una forma y estructura propia de guardar la información además de proporcionar mecanismos para su control.

-Detección de errores: Debe ser capaz de detectar, solucionar y/o minimizar el impacto en el sistema.

-Control de acceso al sistema: En sistemas de acceso compartido se debe vigilar quien accede y con qué permisos accede.

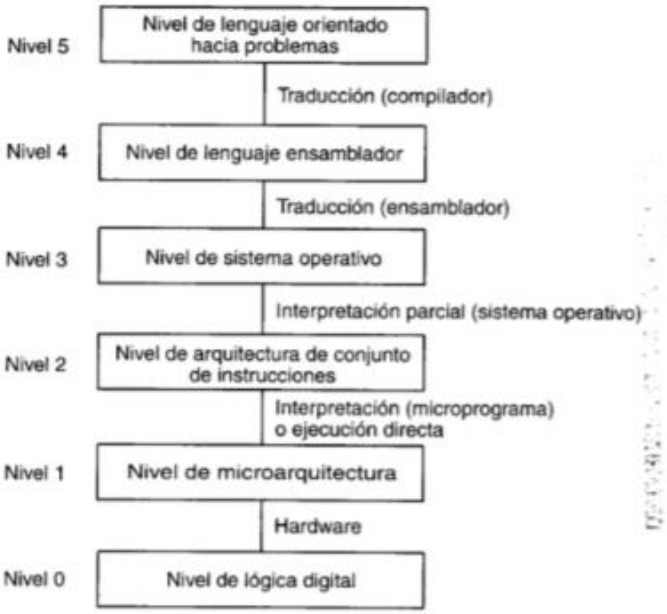
-Monitoreo: Proporcionar información sobre el grado de utilización de los recursos configuraciones y tiempo de respuesta.

2. Maquina multinivel o máquina virtual en capas

Un S.O. puede hacerse modular mediante una estructura en niveles en la que se divide en una serie de capas. Desde el Nivel 0 que sería el Hardware, hasta el nivel más alto que sería la interfaz de usuario.

Cada uno de estos niveles es una implementación de un objeto abstracto formado por unas series de datos y por las operaciones que permiten manipular dichos datos. Surge entonces lo que llamamos maquina multinivel donde cada capa se apoya en la que hay por debajo de ella.

El método multinivel trae como ventaja más simplicidad en la construcción ya que se van depurando desde el más bajo hasta el más alto y aunque los de arriba dependan del que tienen debajo, ya se ha verificado el correcto funcionamiento del mismo.



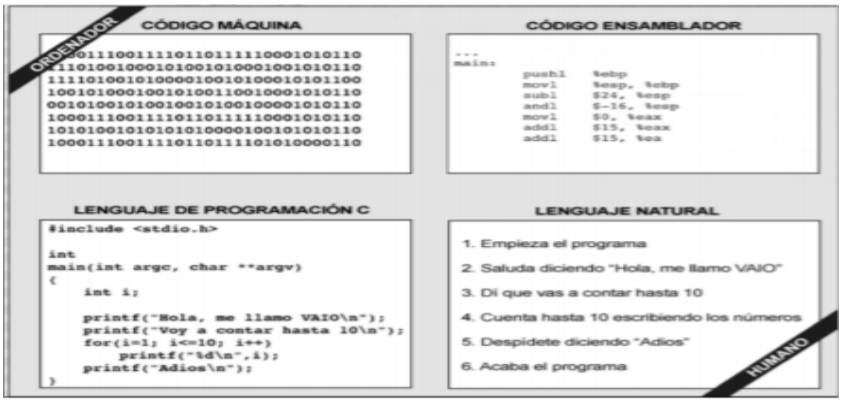
Cada nivel se implementa utilizando las operaciones proporcionadas por los niveles inferiores, por tanto, cada nivel oculta a los niveles superiores la existencia de determinadas estructuras de datos, operaciones y hardware.

Es necesario planificar cuidadosamente los niveles, implementar niveles de manera excesiva puede provocar tiempos de respuesta largos ya que las instrucciones tendrán que pasar por más niveles. Se puede decir que cada nivel añade una carga de trabajo adicional a las llamadas del sistema.

2.1. Paso entre niveles

Las maquinas interpretan un “alfabeto” basado en 0 y 1 que se corresponde a la presencia o ausencia de voltaje, las instrucciones se ejecutan precisamente en forma de colecciones de bits, sin embargo, para un programador sería muy tedioso escribir un programa a base de 0 y 1.

Para evitar esto surgen los lenguajes de programación de alto nivel, y los compiladores que traducen esos lenguajes a un lenguaje de bajo nivel.



En la figura anterior se muestra como un lenguaje de alto nivel (C) se traduce a ensamblador que, si bien es de nivel 4, aun no es entendido por la máquina. El Ensamblador traducirá el código a "código maquina" que está en los niveles 3 y 2.

El código maquina tiene el conjunto de instrucciones que determinan las acciones que deberá realizar la máquina.

Es el código máquina del sistema operativo, el que, por ejemplo, se encarga de arrancar la computadora y el usuario normal no puede acceder a este, ya que se ejecuta en modo "kernel". El usuario podrá, como mucho, hacer cambios a nivel 4.

3. Elementos básicos y organización de un computador

A más alto nivel (Arquitectura de Von Neumann), un computador consta de:

-**CPU**: La CPU controla las funciones del ordenador y realiza las funciones de procesamiento de datos, contiene la ALU, registros, contadores…

-**Memoria principal**: Es una memoria volátil, donde se cargan datos e instrucciones.

-**Módulos de E/S**: Transfieren datos entre el computador y el exterior. Estos módulos contienen buffers (zonas de almacenamiento internas) que mantienen temporalmente los datos hasta que se pueden enviar

Estos componentes se interconectan utilizando los buses del sistema que comunican todo lo anterior.

La CPU y las controladoras de dispositivos pueden funcionar de forma concurrente compitiendo por la memoria principal, con lo que para asegurar su acceso se proporciona una controladora de memoria para sincronizar el acceso a la misma.

3.1. El procesador y sus registros

La CPU tiene un tipo de memoria más rápida y de menor capacidad que la memoria principal llamada registro.

-Registro de dirección de memoria (RDIM): Almacena una dirección de memoria para lectura/escritura.

-Registro de datos de memoria (RDAM): Almacena datos que se reciben de la memoria o que se pueden escribir en ella.

-Registro de dirección de E/S (RDIE/S): Que especifica un determinado dispositivo de E/S.
-Registro de datos de ES (RDAE/S): Intercambia datos entre el módulo de E/S y el procesador.

Además de los anteriores contamos con los siguientes para la ejecución de programas:

-Contador de programa: Dirección de la próxima instrucción a ejecutar.

-Registro de instrucción: Ultima instrucción leída.
-Todos los procesadores disponen de un registro conocido como "palabra estado del programa" que contiene un bit para habilitar/inhabilitar interrupciones, un bit de modo usuario/supervisor y bits de códigos de condición.

Se puede hablar de los registros en **función de si son visibles o no al usuario.**

-**Visibles**: Permiten minimizar las referencias a memoria principal optimizando así el uso de registros. En los lenguajes de alto nivel el compilador elegirá que variables se asignan a registros.
-**No visibles:** Los usa el procesador para controlar su operación y por rutinas del sistema operativo para controlar la ejecución de programas.

3.2. Estructura de almacenamiento

Los programas de la computadora se cargan en la memoria principal (RAM), esta memoria es el único área de almacenamiento de gran tamaño donde el procesador accede directamente.
Esta memoria consiste en una matriz de palabras de memoria, cada una con su propia dirección. Se interacciona con ella mediante la secuencia "load" -que mueve un registro hacia la CPU- y "store" -que mueve un registro de la CPU a la memoria principal-.

Un ciclo típico de ejecución extrae una instrucción de memoria, la almacena en un registro IR y de ella se extraen operandos que se almacenan en algunos registros internos, tras concluir las operaciones el resultado vuelve a la memoria.

Aunque sería deseable que los programas permanecieran en la RAM de forma permanente no es posible debido a que es una memoria volátil y de poco tamaño. Por tanto, se proporciona un sistema de almacenamiento secundario véase el disco duro por ejemplo que almacena los programas y luego se cargan en la RAM.

Otros sistemas de almacenamiento: Cache, CDROM, Pendrive, ….

3.3. Estructura de E/S

Gran parte del sistema operativo gestiona la entrada y la salida. La computadora consta de una o más CPU y múltiples controladoras de dispositivo que se conectan a través de un bus.
Cada controladora se encarga de un dispositivo especifico, aunque en algunos casos se puede encargar de más de un tipo de dispositivo como ocurre con la controladora USB.

La controladora de dispositivo transfiere los datos entre los periféricos que controla y su buffer local.

El software que se encarga del funcionamiento de los controladores se llama "**Driver**" o **controladora de dispositivo Software**, diferente del controlador Hardware (como puede ser una tarjeta de video).

Cuando se inicia una operación de E/S:

1) El driver carga los registros apropiados en la controladora Hardware.
2) Esta examinara el contenido de los registros para conocer la acción a realizar.
3) La controladora transfiere los datos a su buffer local e informa al Driver mediante una -interrupción- de que ha terminado la operación.
4) El Driver da el control al sistema operativo devolviendo los datos necesarios.

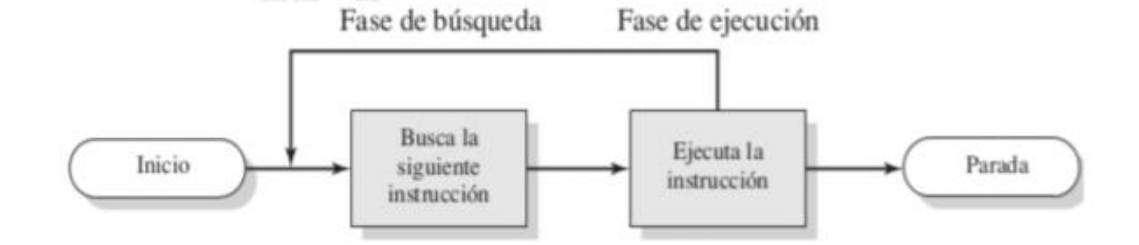
El método descrito anteriormente (Conocido como E/S por interrupción) es perjudicial cuando se trata de un movimiento masivo de datos, este problema se soluciona con el acceso directo a memoria (DMA) que consiste en

que la controladora hardware transfiere datos a la memoria principal sin que intervenga la CPU lo cual libera a la misma para realizar otros trabajos.

4. Ejecución de instrucciones

Un programa que va a ejecutarse almacena un conjunto de instrucciones en la memoria RAM. De forma que el procesador busca estas instrucciones y las ejecuta, por tanto, la ejecución completa consiste en repetir el proceso de búsqueda y ejecución hasta acabar con todas.

A cada una de estas búsquedas y ejecuciones se le llama Ciclo de Instrucción, es decir cada vez que se busca y se ejecuta una instrucción es un ciclo distinto.



Como se ve en la figura, dicho ciclo consta de la Fase de Búsqueda y la Fase de ejecución.

La ejecución de un proceso no se detendrá a menos que se produzca un error, se apague la maquina o se de alguna orden que termine el proceso.

4.1. Búsquedas y ejecución de una instrucción

Las instrucciones leídas de la memoria principal se cargan en un registro del procesador conocido como registro de instrucción (IR), la siguiente instrucción se guardará a continuación de forma que se leerán de forma secuencial.

Estas instrucciones contienen bits que especifican la acción a realizar. Se pueden dividir en cuatro categorías:

-Procesador-memoria: Se pueden transferir datos desde el procesador a la memoria o viceversa.

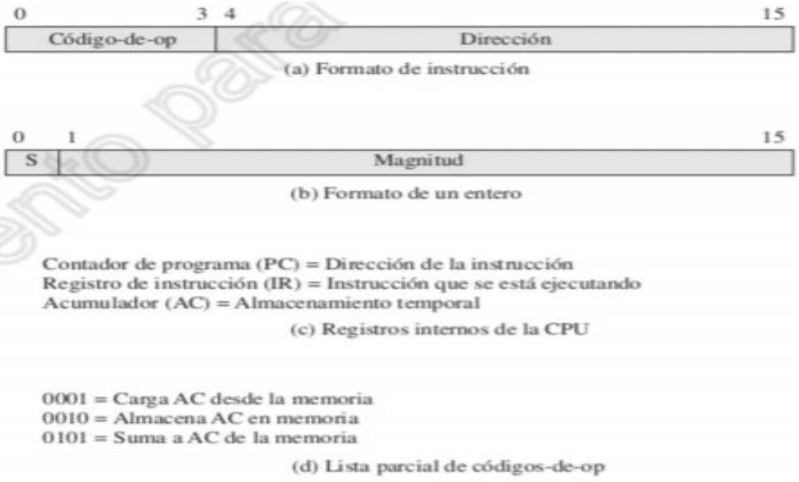
-Procesador-E/S: Se envían datos a un dispositivo periférico o se reciben desde el mismo.

-Procesamiento de Datos: El procesador puede realizar algunas operaciones aritméticas o lógicas.

-Control: Una instrucción puede especificar si se va a alterar la secuencia de ejecución.

4.2. Ejemplo de ejecución de instrucciones

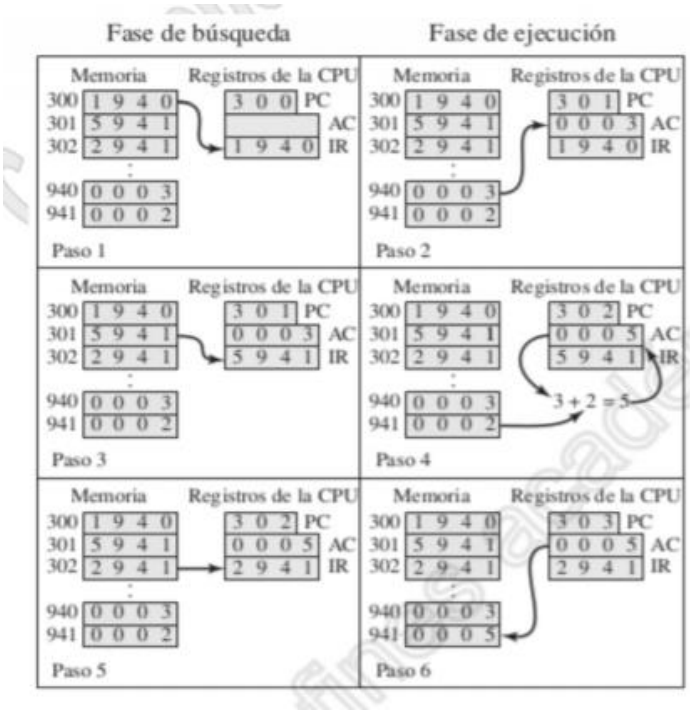
Supongamos tener una máquina de 16 bits con las características de la figura:



El procesador contiene un único registro de datos llamado acumulador AC, las instrucciones y los datos tienen una longitud de 16 Bits estando la memoria organizada como una secuencia de palabras de 16 bits.

Las instrucciones tienen 4 bits para el código de operación y permiten 16 códigos de operación diferentes. Con los 12 bits restantes se pueden direccionar hasta 4.096 palabras de memoria.

La siguiente figura representa la ejecución parcial de instrucciones:



1) El PC contiene el valor 300 que es la dirección de la primera instrucción (con valor 1940 hexadecimal) esta se carga en el registro IR y se incrementa el PC, este proceso involucra a los registros RDIM y RDAM.
2) El primer digito hexadecimal del IR indica que en el AC se va a cargar un valor leído de la memoria. Los restantes 3 dígitos hexadecimales indican que la dirección de memoria es la 940.
3) Se lee la siguiente instrucción: 5941 de la posición 301 del PC y se incrementa este.
4) El contenido previo del AC y el de la posición 941 se suman y el resultado va al AC.
5) SE lee la siguiente instrucción 2941 de la posición 302 y se incrementa el PC.
6) Se almacena el contenido del AC en la posición 941.

Hemos hecho 3 ciclos cada uno con fase de búsqueda y ejecución.

5. Interrupciones y manejador de interrupciones.

El procesador se puede interrumpir mientras realiza fases de búsqueda-ejecución por un suceso determinado. Estas interrupciones pueden ser software o hardware.

Una interrupción hardware se puede activar en cualquier momento enviando una señal a la CPU, hay distintos tipos:

-Interrupción de E/S: Se genera por un controlador de dispositivos E/S y señala la conclusión de una operación o un error.

-Interrupción de fallo de Hardware: Cortes de energía o zonas corruptas de memoria.

Por otro lado, las interrupciones por Software son:

-**Interrupciones de programa (Excepciones):** Se producen por la ejecución de una instrucción que genera una condición determinada: Desbordamiento aritmético, división por cero, referencias a espacios de memoria no permitidos…

-**Interrupción por temporizador:** Permiten al sistema operativo generar ciertas funciones de forma regular.

Una interrupción puede venir cuando un proceso agote su cuota de CPU y tenga que dejar paso al siguiente.

Cada vez que se interrumpe la CPU, esta deja lo que está haciendo y transfiere la ejecución a una posición de la memoria RAM ya fijada, dicha posición contiene la dirección donde se encuentra la rutina al servicio de interrupción "ISR" la cual al ejecutarse trata la interrupción para que después la CPU reanude su trabajo.

Aunque cada computadora es diferente, suele haber un método común:

1) Se invoca la rutina ISR genérica

2) Dicha rutina invoca a otra rutina especifica de tratamiento de interrupción.

Este método es algo lento y el tratamiento de interrupciones debe ser rápido, para lo cual existe otra forma: **Utilizar una tabla de punteros a las diferentes rutinas de interrupción (Vector de interrupciones).**

Este vector se ubica en la memoria principal y cada elemento es una dirección de memoria de rutinas ISR. El vector además va indexado con un numero de interrupción que se proporciona en la propia solicitud de interrupción con el que podemos obtener la dirección ISR específica para una interrupción concreta.

IMPORTANTE: **Cada vez que se invoca a un ISR se almacena la dirección de la instrucción interrumpida correspondiente al proceso actual, es decir la siguiente instrucción que se iba a ejecutar cuando se produjo la interrupción. Tras atender la interrupción la dirección de retorno guardada se carga en el contador del programa y el programa interrumpido se reanuda.**

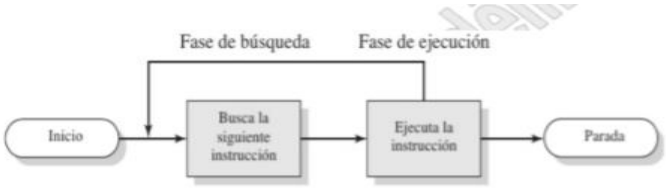
Cuando hay una interrupción activa el resto están deshabilitadas, por ello hay que priorizar las instrucciones.

IMPORTANTE: Cuando hablamos de interrupción la "dirección" va al procesador, es decir, el procesador interrumpe una tarea que está realizando. Otra cosa es invocar una operación de E/S que resulta en una llamada al sistema, lo cual no es una interrupción.

5.1. Adición de la fase de interrupción

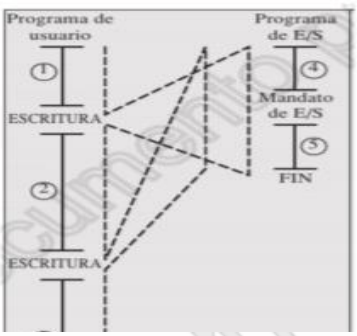
El tratamiento de instrucciones E/S es muy importante porque estos dispositivos son más lentos que el procesador.

Vamos a suponer un proceso en el cual se transfieren datos a una impresora utilizando el esquema del ciclo de instrucción



Después de cada petición de escritura a la impresora, el procesador para y está inactivo hasta que la impresora vacíe su buffer local e informe de ello. Este tiempo es un enorme desperdicio de procesador.

Si se realizan una serie de peticiones de impresión intercaladas con el resto del código del programa:

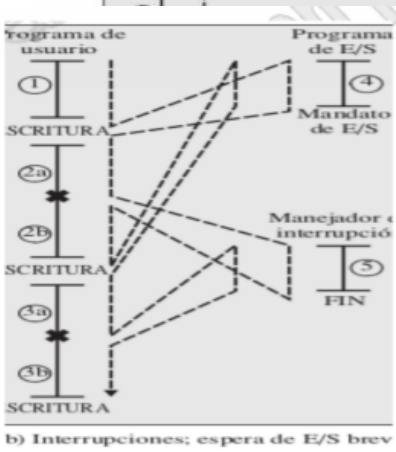


El circulo 4 consiste en preparar la operación E/S (Comprobar si la impresora esta libre y enviar la información a sus buffers).

El circulo 5 es el vaciado de los buffers de la impresora para que realice la impresión.

Entre el 4 y 5 el procesador se queda esperando desperdiciándose por tanto ciclos de CPU.

Lo que se hace por tanto es hacer que mientras la impresora vacía sus buffers el procesador realice otras tareas como se ve en la figura.



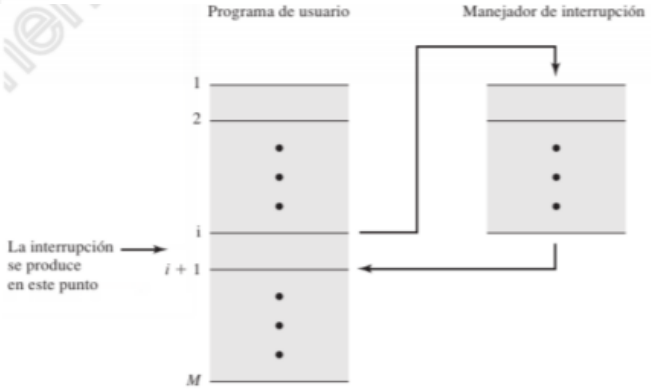
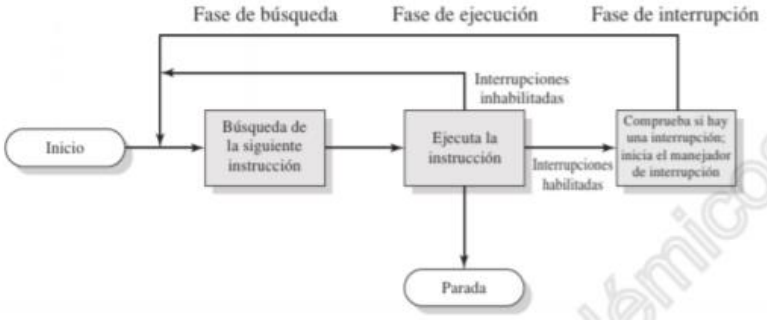


Figura 1.6. Transferencia de control mediante interrupciones.

el código del
ntroladora una
anuda la ejecución

errupción, sino que

Para completar los conceptos, añadimos una fase de interrupción al ciclo de instrucción



Con lo cual, aclarando conceptos:

-El procesador comprueba si se ha producido una interrupción (Comprobación hardware) hecho indicado por una señal de interrupción en un registro del procesador y enviada por la controladora de E/S en este caso.

-Si está pendiente de una interrupción el procesador suspende el proceso actual para ejecutar la rutina ISR y después se reanuda el proceso

-Si no hay interrupciones el procesador continua la búsqueda y lee la siguiente instrucción del proceso actual.

5.2. Interrupciones múltiples

Es posible que se produzca una interrupción mientras se está produciendo otra. Se pueden considerar en este caso dos alternativas:

-La primera consiste en inhabilitar las interrupciones mientras que se procesa una interrupción. **Una interrupción inhabilitada significa que el procesador ignorará cualquier nueva señal de interrupción**.
Las interrupciones que se producen mediante este tiempo quedaran pendientes de ser procesadas y el procesador la atenderá después de que se rehabiliten las instrucciones. Las interrupciones se manejan en orden secuencial.
La desventaja de esta estrategia consiste en que no se tienen en cuenta las prioridades de las interrupciones.

-La segunda estrategia **consiste en definir prioridades** para las interrupciones y permite que una interrupción de más prioridad cause que se ponga por delante de una interrupción menos urgente.

6. Arquitectura de un sistema informático desde el punto de vista del procesador.

Los sistemas informáticos se organizan de varias maneras en función del número de procesadores.

6.1. Sistemas de un solo procesador

Consisten en una CPU de un solo núcleo, este es un procesador que ejecuta instrucciones de propósito general, también se dispone de procesadores de propósito especial que pueden venir en los dispositivos (microprocesador en una controladora de disco, dispositivo E/S, etc.…) que quitan carga al procesador principal pero no ejecutan programas de usuario lo cual es tarea de la CPU principal, el S.O. los gestiona y monitoriza.
El disponer de estos procesadores especiales liberan a la CPU de carga y de tiempos ociosos.

6.2. Sistemas Multiprocesador.

Sistemas cuya CPU tiene más de un núcleo que realizan tareas de forma paralela.
Estos procesadores se impulsaron debido a los problemas provocados por la temperatura en las CPU mono núcleo al querer meter cada vez más nano transistores.
Esto implica que el S.O. tenga que estar preparado para gestionar un procesador multinucleo.

Obtenemos las siguientes ventajas:

-Mayor rendimiento: Al aumentar el número de procesadores se realizan más trabajos independientes en menos tiempo. Sin embargo, la mejora de rendimiento con N procesadores es menor que N ya que al tener varios procesadores cooperando parte de la carga de trabajo se invierte en conseguir que todas las partes funcionen correctamente.
Si a esto añadimos la competición por los recursos se reduce la ganancia esperada por añadir procesadores adicionales, lo cual no pasa con un procesador multinúcleo.
-Economía a escala: Los sistemas pueden compartir periféricos, almacenamiento, etc.

6.2.1. Tipos de sistema multiprocesador

ASIMETRICO: A cada procesador se le asigna una tarea específica con un procesador maestro controlando el sistema, el resto esperan a que este les dé instrucciones, también pueden tener asignadas tareas predefinidas.
Aunque es un sistema muy simple, presenta serias desventajas:

-Si falla el maestro falla todo el sistema.
-Uso deficiente de los recursos, se tiene que esperar a que el maestro asigne trabajo.
-Incrementa el número de interrupciones ya que el maestro se ve interrumpido por el resto de procesadores lo que crea largas colas y cuellos de botella.

SIMETRICO(SMP): Cada procesador realiza las tareas correspondientes al SO, sin relación maestro-esclavo. Tienen las siguientes características:

-Comparten las mismas utilidades de la memoria principal y E/S interconectadas por un bus.

-Todos los procesadores realizan las mismas funciones.

-Los procesadores son gestionados por el SO y por tanto permanecen transparentes al usuario.

Además, tiene las siguientes ventajas:

-Rendimiento y economía a escala.
-Mas confiable ya que el fallo de un procesador no supone el fallo del resto.
-Aumento de la disponibilidad, sin colas y sin cuellos de botella

6.3. Sistemas en clúster

Son sistemas formados por un conjunto de sistemas acoplados que comparten almacenamiento y se conectan entre si mediante red LAN, este conjunto visto como un solo ordenador alcanza mucha más potencia que un equipo de escritorio y su principal uso es mejorar el rendimiento y la disponibilidad.
Pueden ser:

-Homogéneos: Mismo hardware y SO
-Semihomogeneo: Diferente hardware y mismo SO.
-Heterogéneo: Diferente hardware y SO.

En función de su propósito pueden ser:

-**Alto rendimiento:** Ejecuta tareas que requiere de gran capacidad computacional y/o memoria.
-Alta disponibilidad: Tratan de dar un servicio que este siempre disponible además de permitir la recuperación de fallos como los sistemas RAID de backup.
-Alta eficiencia: Tienen el objetivo de ejecutar la mayor cantidad de tareas en el menor tiempo posible, no necesitan que una tarea termine para comenzar otra.

7. Multiprogramación

Incrementa el uso de CPU de forma que no pueda estar ociosa.

El sistema operativo mantiene en memoria principal varios trabajos de forma simultánea y ejecuta uno de ellos. A lo mejor el trabajo tiene que esperar a que finalice otra tarea.

Si el sistema no fuera multiprogramado la CPU se quedaría inactiva hasta que dicha tarea concluyese. Sin embargo, en un sistema multiprogramado el SO cambia de trabajo y ejecuta otro mientras el anterior trabajo espera.

Siempre que la CPU tiene que esperar, se lanza otro trabajo y así sucesivamente, a que trabajo se cambia es decisión del planificador que el sistema utilice.

8. Multiprocesamiento

Este se da en sistemas con más de un procesador, la multiprogramación sin embargo puede darse en sistemas con uno o más.
Se define como el uso o ejecución de múltiples programas concurrentes (al mismo tiempo) en un sistema, en lugar de un único proceso en un instante determinado (mono procesamiento).
En resumen: Con la multiprogramación solo un proceso se puede ejecutar a la vez mientras el resto esperan, con multiproceso más de un proceso puede ejecutarse simultáneamente cada uno de ellos en un procesador diferente.
IMPORTANTE: ¡En un sistema multiprocesamiento también hay multiprogramación!!!!

9. Modo dual (usuario y kernel)

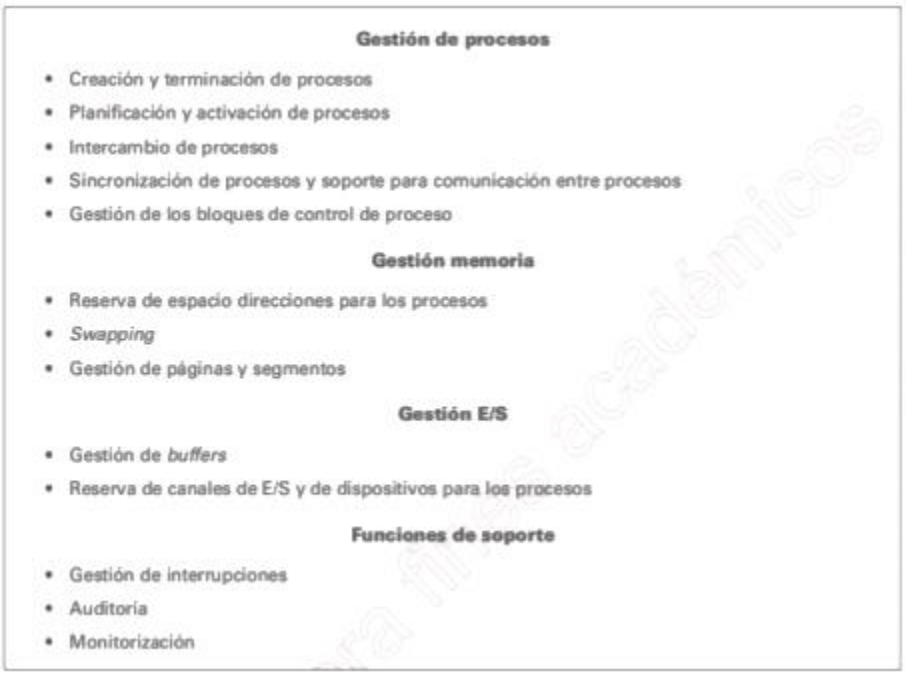
El usuario no debe poder cambiar a modo KERNEL, a través de ningún programa de aplicación, rutinas o instrucciones máquina que rigen el comportamiento del SO y que interactúan con el hardware ya que afectaría al comportamiento del sistema. Para evitar esto los diseñadores han buscado una solución basada en un modo dual de operación.

-Modo núcleo, Kernel o privilegiado: Asociado al procesamiento de rutinas e instrucciones del SO. Solo se puede acceder a él a través de llamadas al sistema, que son rutinas o funciones relativas a nivel de núcleo.

El núcleo es la parte del SO que engloba las funciones y servicios más importantes y que deben protegerse, reside en lenguaje maquina en la memoria principal de forma continua mientras que el resto del SO se carga solo cuando es necesario (si existe kernel modular).

-Modo usuario: asociado al procesamiento de rutinas e instrucciones de los programas de usuario. Es decir, ejecuta rutinas cuyo código maquina está situado en la parte de la memoria principal reservada a dichos programas. ***Si un programa de usuario quiere acceder al núcleo debe hacer una llamada al sistema.***

Funciones básicas del núcleo:



Teniendo el núcleo aislado además conseguimos que si se cae un programa lo haga solo para el usuario en lugar de caerse el SO entero.

¿Cuándo se cambia de modo?

-Llamadas al sistema

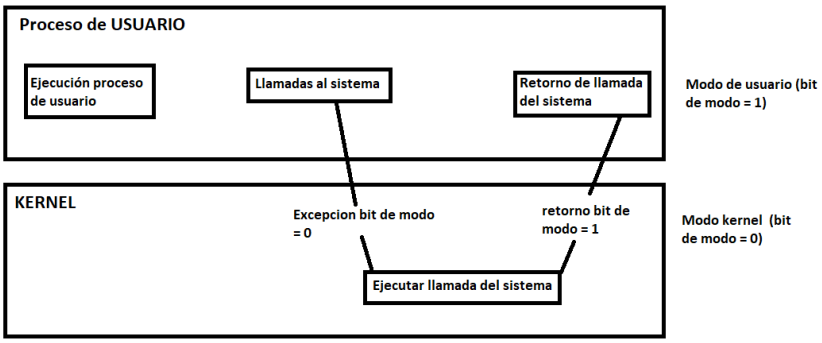
-Cuando una aplicación provoca una excepción.

-Cuando un dispositivo provoca una interrupción.

¿Cómo se cambia de modo?

Existe un bit en la palabra de estado del programa que indica el modo de ejecución, este bit cambia en determinados eventos como puede ser una interrupción o llamada al sistema.

Así si un programa intenta ejecutar una instrucción privilegiada de núcleo, el hardware envía una **excepción** al sistema operativo, esta se trata igual que cuando se produce una interrupción y el SO da un mensaje de error apropiado.



10. Llamadas al sistema

Una llamada al sistema es utilizada por una aplicación de usuario para solicitar un servicio al sistema operativo, por tanto, proporciona una interfaz entre dicha aplicación y el SO.

Se hace constantemente uso de las llamadas al sistema, haciendo por lo general miles de llamadas por segundo.

Los programadores pueden usar las llamadas al sistema de dos formas.

-**API:** Es utilizada por los programadores para implementar sus programas.

Una API es un conjunto de funciones que el programador de aplicaciones puede usar.

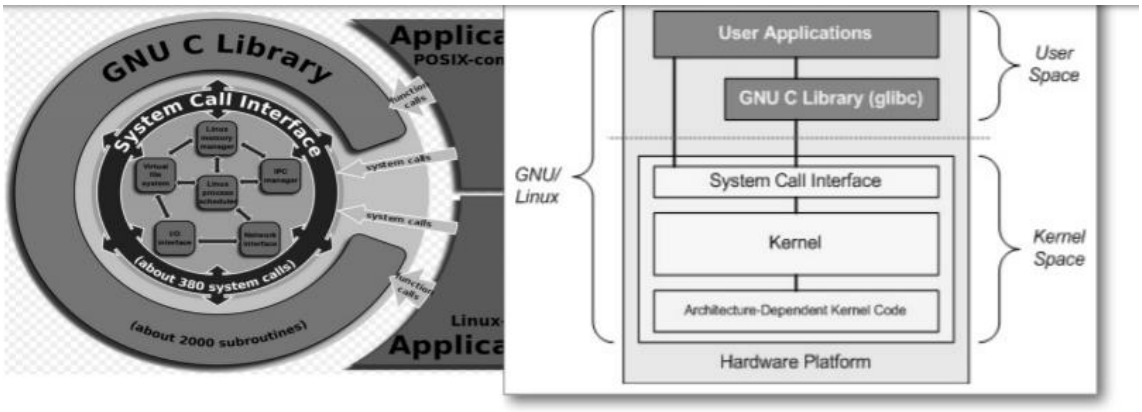
Las funciones que conforman una API de alto nivel invocan a las llamadas al sistema internamente (se les llama **wrappers** o envolventes de funciones nativas del núcleo).

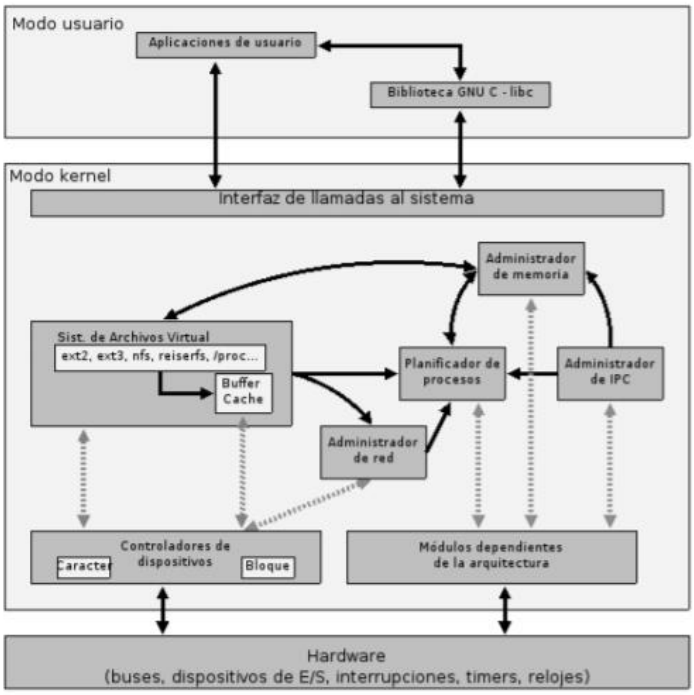
Ejemplo de Apis: glibc, Win32

La propia función invocada no es la llamada al sistema en sí, lo que ocurre es que por debajo se encuentra una, o un conjunto de llamadas al sistema nativas.

Ejemplo: La función CreateProcess () de Win32 llama a la función nativa del sistema NTCreatePRocess () del kernel de Windows.

-Acceso mediante la interfaz ofrecida por el núcleo del SO: Esta definida en lenguaje Ensamblador.
Es una forma más compleja ya que el programador tiene que utilizar lenguaje ensamblador para invocar una llamada al sistema.
Dependen del hardware sobre el que se esté ejecutando el SO.





Funciones nativas implementadas en ensamblador (lo de arriba): Son invocadas mediante código de identificación, previo paso a modo núcleo.

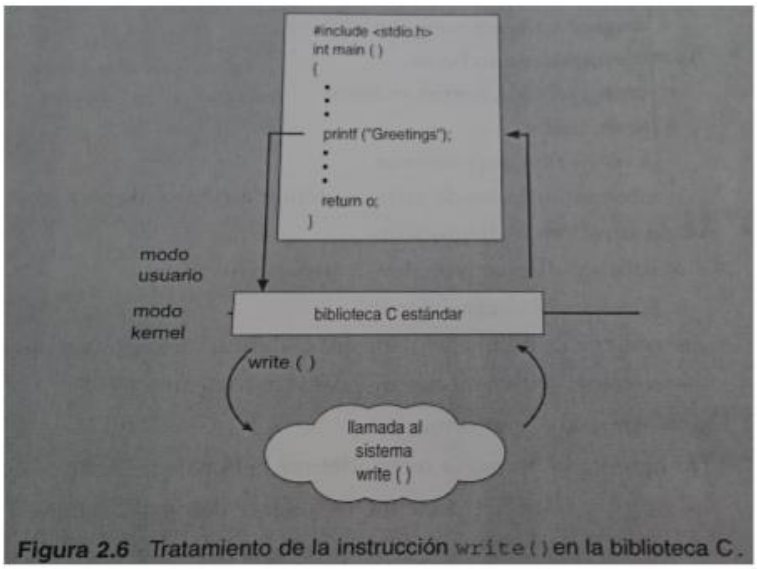


Figura 2.6 Tratamiento de la instrucción `write()` en la biblioteca C.

A su vez puede invocar otras subrutinas a nivel de núcleo.

Cuando realizamos una llamada al sistema puede ser necesario pasar parámetros al Kernel, para ello se emplean tres métodos.

-Pasar los parámetros a registros del procesador.

-Si hay más parámetros que registros se pasan a un bloque de memoria y la dirección del bloque se pasa como parámetro a un registro.

-**En otros casos el programa coloca los parámetros en la pila de memoria principal** reservada al usuario y el SO extraerá de la pila los parámetros a partir de una dirección de memoria principal proporcionada. Este método, llamado método(pila), junto al anterior (bloque), es el preferido por la mayoría de los sistemas operativos al no limitar la cantidad y longitud de parámetros que se pueden pasar.

Cada llamada al sistema tiene asociado un número y la interfaz de llamadas al sistema mantiene una tabla indexada en la memoria con dichos números. Con esta tabla, la interfaz de llamadas al sistema invoca a la llamada necesaria del kernel y devuelve el estado de ejecución de la llamada al sistema y los posibles valores de retorno. Esto se hace con funciones especiales denominadas trap.

¿Cómo conseguimos la portabilidad entre diferentes S? Os.?

Las llamadas el sistema se identifican con un identificador(numero) utilizado para indexar la tabla de llamadas al sistema, esta se conserva entre versiones, por tanto, dicha tabla es la que ofrece la compatibilidad entre diferentes versiones de SO.

¿Cómo implementar una nueva llamada al sistema?

Habría que añadir la llamada al fichero unistd.h comprobando que no exista el número de llamada que queremos añadir.

Tipos de llamada al sistema:

-Control de procesos.

-Manipulación de archivos

-Manipulación de dispositivos.

-Mantenimiento de información (fecha, hora, etc.…)

-Comunicaciones (mensajes, información de estado, etc.…)

10.1. Funciones de tipo trap para acceso al núcleo

¿Qué ocurre cuando se produce una invocación y como se realiza?

1. Si se produce a partir de la API, los parámetros asociados a la misma se cargarían en la pila de proceso a nivel de usuario o en registros dependiendo del sistema.

2. En la implementación interna de la API invocada, se ejecuta la función trap, esta se invoca a bajo nivel, pero aún no en ensamblador ya que estas funciones trap no forman parte de la API, sino que son las funciones del api las que se invocan internamente.

3. La función trap obtiene los parámetros almacenados y ejecuta una interrupción especial que cambian a modo núcleo invocamos a SYSTEMCALL.

4. Cada vez que se realiza una llamada al sistema se guarda el estado de ejecución (función SAVEALL) actual y el valor del PC ya que se va a saltar a una zona de memoria donde se encuentra la llamada al sistema invocada.

   La interrupción que invoca trap, llama a otras subrutinas para las operaciones de salvado, justo antes de ejecutar la llamada al sistema concreta a partir de su número de identificación.

5. Se examina la rutina invocada y sus parámetros y se busca en una tabla de rutinas si existe el número de identificación.

6. Tras identificar la rutina y verificar los parámetros se procede a ejecutarla.

7. Una vez ejecutada la llamada se invocan las subrutinas RETURN FROM TRAP, para devolver el código de estado de la llamada al sistema los posibles parámetros de retorno. En este punto se vuelve a modo usuario.

8. El sistema descarga la pila y comprueba el resultado de ejecución de la llamada al sistema de forma que el programa de usuario continua con la ejecución.

10.2. Llamadas al sistema en GNU/LINUX (Entra SEGURO)

En esta arquitectura se utiliza la interrupción especial int0x80 como manera de acceder a una llamada al sistema. Estas llamadas aparecen en el fichero include/asm-i386/unistd.h

La interrupción init0x80 se invoca a partir de a __init trap_init () -alojada en el fichero arch/x86/kernel/traps.c - invocada a su vez por start_kernel () que viene de una función a nivel de API.
El numero de la interrupción está definido por la constante SYSCALL_VECTOR que está en el fichero arch/x86/include/asm/irq_vectors.h

Lo siguiente es establecer el método de entrada al núcleo mediante la función set_system_gate (), que es la que invoca a int0x80 que da el salto de memoria a donde se encuentra system_call (), en este momento se pasa a modo núcleo y el puntero de pila apunta a la pila del núcleo del proceso donde está la función system_call ().

Una vez en modo núcleo, se suceden los siguientes pasos.

- La función system_call (), guarda el numero identificador de la llamada al sistema donde está situado el registro %eax.
- Acto seguido la macro SAVE_ALL guarda todos los registros del procesador y también se guardan en la pila los registros de la función interrumpida a nivel de usuario.
- Se comprueba que el número de llamada al sistema es válido mirando sys_call_table. De no serlo se ejecuta syscall_badsys metiendo un -1 en el registro %eax, el código de error se devuelve en ENOSYS (posteriormente erno) y salta al código de la subrutina resume_userspace.
- Si se puede realizar la llamada, se guarda el valor de retorno en la posición de la pila donde se sitúa el registro %eax. Este registro tenía hasta este momento el identificador de la llamada al sistema a invocar.
  La llamada al sistema tiene esta forma de prototipo: sys_nombre-de-lafuncion(parámetros). Los parámetros se recogen de system_call ()
- Tras la llamada se lanza la subrutina syscall_exit () y se comprueba el proceso que el usuario interrumpió para restaurarlo.
- Se ejecuta la macro RESTORE_ALL para restaurar los registros almacenados en SAVE_ALL
- Se ejecuta el retorno de interrupción iret. Restaurando los valores en la pila a nivel de usuario. En este punto se cambia a modo usuario.
- La biblioteca glibc obtiene el resultado de la llamada y sigue la ejecución del programa justo por donde iba.
- Si la llamada no hubiera sido exitosa el valor %eax devuelto seria -1, en ese caso se copiaría el error en ERRNO y se devuelve -1 a la pila de usuario.

  (Las funciones se encuentran en el fichero unistd.h)

11. Diseño del sistema operativo

La estructuración del sistema se difiere según el número de capas y sus funcionalidades

11.1. Sistemas Monolíticos

El SO no se organiza en capas, sino como una colección de procedimientos enlazados entre sí en un solo programa binario. El SO es el único proceso privilegiado.

Cada rutina tiene la libertad de llamar a cualquier otra, como son miles de procedimientos que se pueden llamar entre si sin restricción es un sistema poco manejable y difícil de comprender, por ejemplo, MSDOS o la primera versión UNIX.

Estos sistemas tienen la ventaja de que son simples de diseñar y la velocidad de ejecución es muy rápida. Sin embargo, tienen graves problemas a la hora de modificar el sistema operativo ya que implica la modificación de todo el SO completo.

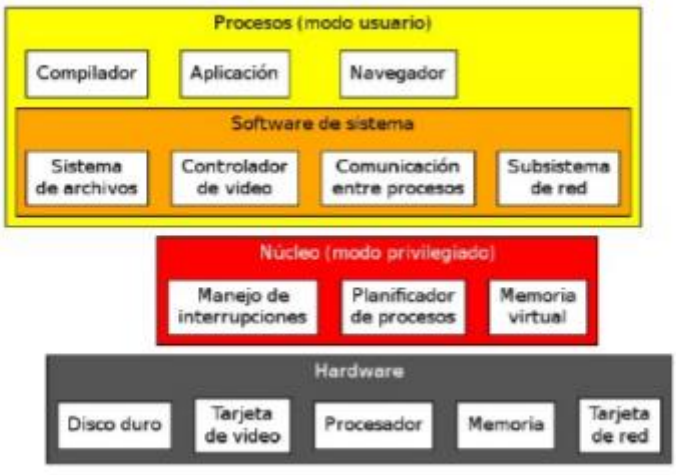
11.2. Sistemas Microkernels

Consiste en lograr una alta confiabilidad al dividir el SO en varios módulos pequeños y bien definidos, donde un solo modulo llamado microkernel se ejecuta en modo kernel y el resto se ejecuta como procesos de usuarios ordinarios.

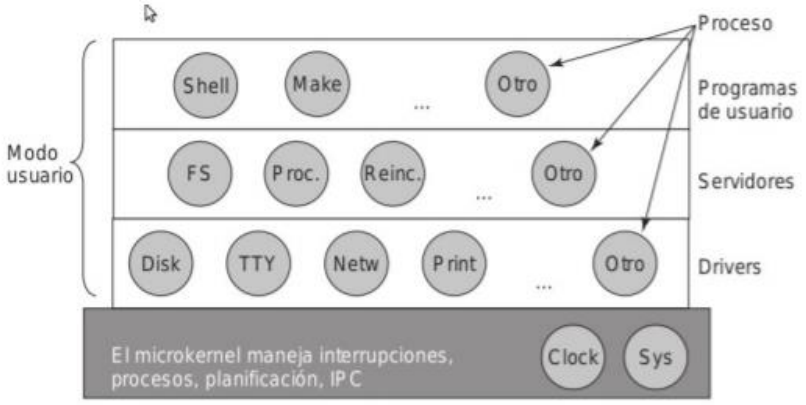
Se eliminan todos los componentes no esenciales del kernel y estos se implementan como programas de sistema y usuario, resultando un kernel muy pequeños.

Aunque el microkernel proporciona servicios mínimos lleva a cabo estas funciones:

- Cierta administración de la memoria
- Cantidad limitada de planificación y administración de procesos de bajo nivel.
- E/S de bajo nivel (gestión de interrupciones)



Un ejemplo de este SO lo podemos encontrar en Minix 3 donde los programas de usuario “servidores” realizan casi todo el trabajo del SO y otros programas llamados “drivers” se comunican con el microkernel.



El microkernel proporciona más fiabilidad porque los servicios se ejecutan en modo usuario con lo que de fallar algún servicio no se afecta el SO entero.

Sin embargo, los microkernel tienen mal rendimiento debido a una carga adicional de procesamiento impuesta por las funciones del sistema ya que los distintos componentes se ejecutan en espacios de memoria distintos y la activación de cada proceso requiere más tiempo

11.3. Sistemas basados en módulos

Utilizan la idea de la programación orientada a objetos para crear un kernel modular.
El kernel dispone de un conjunto de componentes fundamentales y enlaza dinámicamente los servicios adicionales en el arranque o en el tiempo de ejecución.

Permite que, para modificar el SO, solo tengamos que modificar determinados módulos en lugar del sistema entero como en los kernel monolíticos.

Al final, el resultado es un sistema de varios niveles muy flexible donde cualquier modulo puede llamar a cualquier otro modulo.
El módulo principal solo dispone de las funciones esenciales y las instrucciones para cargar y comunicarse con los módulos.

Los módulos que carga el kernel son utilizados para brindar soporte a nuevos dispositivos hardware, sistema de archivos y agregar llamadas al sistema. Si un módulo deja de ser requerido este puede descargarse y liberar memoria.

Cuando un sistema operativo no dispone de módulos cargables, el kernel tiene que acumular todas las funcionalidades, lo que repercute en una imagen del kernel mucho mayor que ocupa gran espacio en memoria, además los usuarios tendrían que recompilar el núcleo base cada vez que quieran agregar nuevas funcionalidades.

Los módulos cargables son un buen método para modificar el kernel en ejecución, pero puede ser abusado por atacantes del sistema, con lo que se opta con que solo un usuario “root” pueda cargar módulos”.

12. Interprete de comandos

Tambien llamado SHELL, ejecuta comandos introducidos por el usuario, su uso mas común es la manipulación de archivos.

Los comandos se implementan de dos formas:

- Uno de los métodos consiste en que la SHELL obtiene el código del comando que va a ejecutar.
- Otro método, muy utilizado por sistemas basados en GNU, implementa los comandos a través de una serie de programas del sistema. La SHELL no "entiende" el comando, sino que lo usa para identificar un archivo que hay que cargar en memoria y ejecutar.
  Esto permite que los programadores puedan añadir comando al sistema, creando nuevos archivos con los nombres apropiados.

13. Arranque del sistema

Cada vez que arranca la computadorita, ocurre el siguiente procedimiento.

- Se ejecuta el programa de inicio almacenado en memoria ROM o EPROM, es decir, la BIOS. Dicho programa identifica y diagnostica los dispositivos del sistema verificando su funcionamiento.
- Si las pruebas de diagnóstico de la BIOS son exitosas, la secuencia de arranque continua, la BIOS hace que se comience a ejecutar el gestor de arranque.
  Los cargadores de arranque se encuentran en el primer sector del disco duro, dicho sector recibe el nombre de MASTER BOOT RECORD o MBR para los amigos.
- El gestor de arranque localiza el kernel y lo carga en memoria principal para ejecutarlo, en este momento:
  - SE configura la RAM y memoria virtual(SWAPPER)
  - Configuración de E/S
  - Establecimiento del manejo de interrupciones.
  - Montaje del sistema de archivos del root en directorio raíz (/)
- Una vez ejecutado el Kernel, el SO establece el espacio del usuario. En sistemas GNU, se ejecuta el proceso init() que realiza tareas como:
  - Montaje de particiones
  - Inicialización de servicios de red
  - Inicio del entorno gráfico
  - Sesiones de usuarios

Ejemplo de cargadores de arranque: Lilo y Grub en GNU, NTLoader y Windows Boot Manager en sistemas Windows