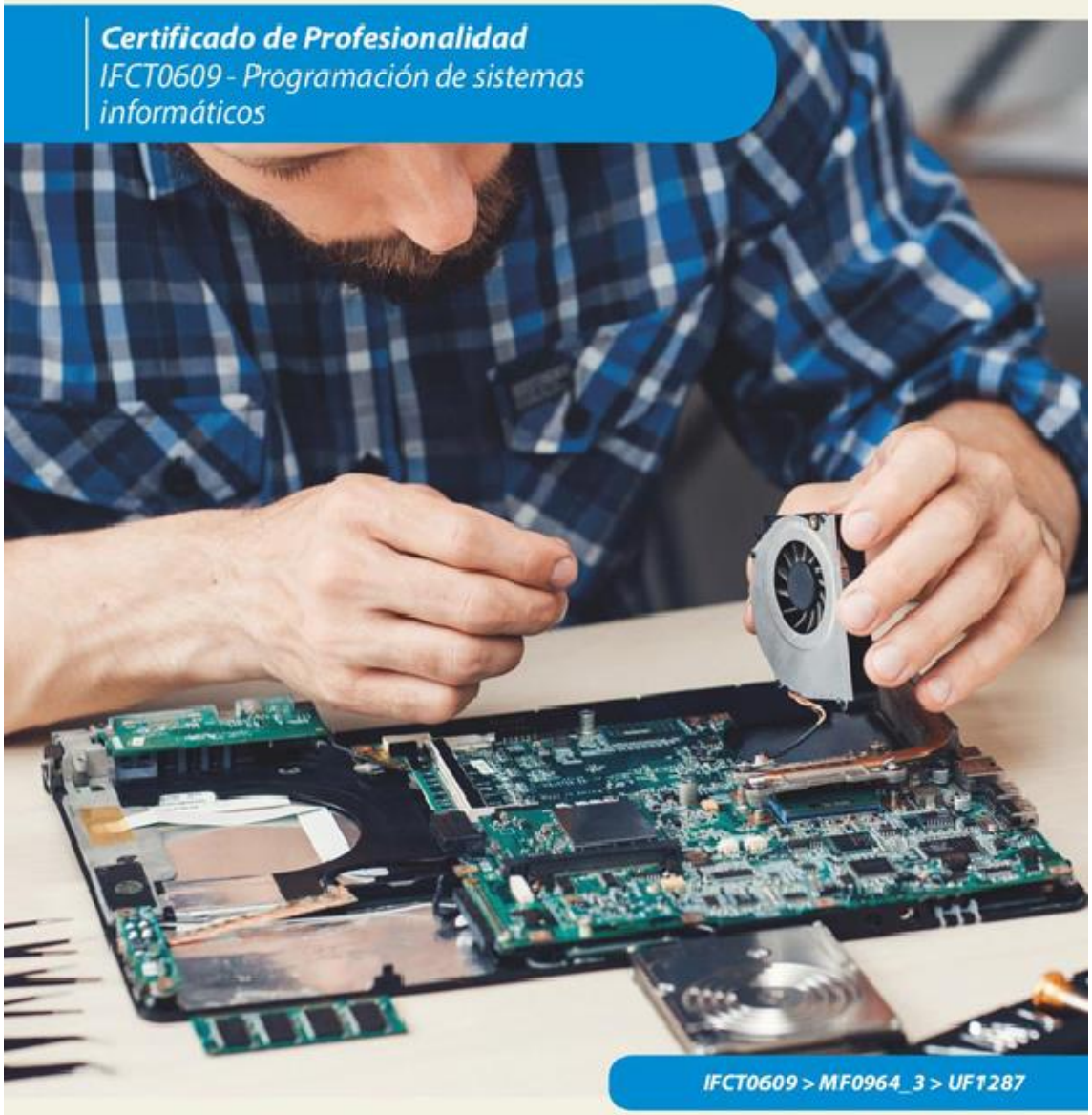
UF1287: Desarrollo de componentes software para el manejo de dispositivos (Drivers)

Certificado de Profesionalidad
IFCT0609 - Programación de sistemas informáticos



IFCT0609 > MF0964_3 > UF1287

ic editorial

Mª Josefa Díaz Coca

Desarrollo de componentes *software* para el manejo de dispositivos *(Drivers)* IFCT0609

Mª Josefa Díaz Coca

ic editorial

Desarrollo de componentes *software* para el manejo de dispositivos *(Drivers)*. IFCT0609
© Mª Josefa Díaz Coca

2.ª Edición

Editado por: IC Editorial
c/ Cueva de Viera, 2, Local 3
Centro Negocios CADI
29200 Antequera (Málaga)
Teléfono: 952 70 60 04
Fax: 952 84 55 03
Correo electrónico: iceditorial@iceditorial.com
Internet: www.iceditorial.com

ISBN: 978-84-1103-950-5

Presentación del manual

El **Certificado de Profesionalidad** es el instrumento de acreditación, en el ámbito de la Administración laboral, de las cualificaciones profesionales del Catálogo Nacional de Cualificaciones Profesionales adquiridas a través de procesos formativos o del proceso de reconocimiento de la experiencia laboral y de vías no formales de formación.

El elemento mínimo acreditable es la **Unidad de Competencia**. La suma de las acreditaciones de las unidades de competencia conforma la acreditación de la competencia general.

Una **Unidad de Competencia** se define como una agrupación de tareas productivas específica que realiza el profesional. Las diferentes unidades de competencia de un certificado de profesionalidad conforman la **Competencia General**, definiendo el conjunto de conocimientos y capacidades que permiten el ejercicio de una actividad profesional determinada.

Cada **Unidad de Competencia** lleva asociado un **Módulo Formativo**, donde se describe la formación necesaria para adquirir esa **Unidad de Competencia**, pudiendo dividirse en **Unidades Formativas**.

El presente manual desarrolla la Unidad Formativa **UF1287: Desarrollo de componentes software para el manejo de dispositivos (Drivers)**,

perteneciente al Módulo Formativo **MF0964_3: Desarrollo de elementos software para gestión de sistemas**,

asociado a la unidad de competencia **UC0964_3: Crear elementos software para la gestión del sistema y sus recursos**,

del Certificado de Profesionalidad **Programación de sistemas informáticos**.

Índice

Salida por pantalla 12438 se ejecutan verticalmente
4 semaforos s1,s2,s3,s4 inicialmente a 0
proceso 1 S1, s2, s3 =1 (signal incrementa)
proceso 2 s1=0, s3=0, S4=1 s3=-1
proceso 3 s2=0, S4=0 S3=0

Capítulo 1

El núcleo del sistema operativo

Contenido

1. Introducción

Un sistema informático puede dividirse, a grandes rasgos, en los siguientes componentes: el ***hardware*** (la máquina física y sus componentes electrónicos), el ***software*** (el sistema operativo y los programas de aplicación) y los usuarios.

El papel que juega el sistema operativo dentro de este sistema es controlar y coordinar el uso del *hardware* entre los diversos programas de aplicación que utilizan los distintos usuarios.

El sistema operativo puede definirse como un *software* que gestiona el *hardware* del ordenador y proporciona un entorno para ejecutar los programas de aplicación.

El núcleo, también llamado *kernel*, constituye el nivel más bajo del sistema operativo y proporciona una interfaz entre el *hardware* y el resto de niveles del sistema operativo.

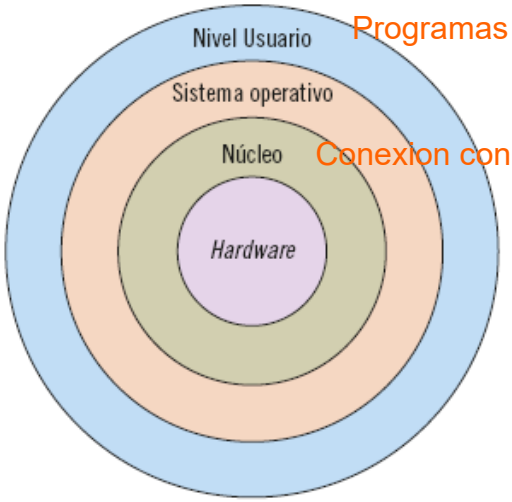
El objetivo de este capítulo es definir, de forma detallada, qué es el núcleo de un sistema operativo y describir cómo gestiona los recursos básicos del sistema, para proporcionar los servicios esenciales a los programas de aplicación y usuarios.

2. Arquitectura general del núcleo

El núcleo *(kernel)* es la parte esencial del sistema operativo, que siempre está residente en la memoria real (en la memoria de acceso aleatorio RAM). Esta parte es cargada con la ejecución de las tareas más importantes del sistema.

La finalidad principal del núcleo es construir un entorno adecuado en el que se puedan ejecutar los procesos, transformando los recursos reales de la máquina en recursos estándares y cómodos de usar.

Estructura genérica de un sistema operativo



Programas que utiliza el usuario

Conexion con el hardware

El **proceso** es la unidad fundamental de trabajo de un sistema operativo.

Proporcionar el entorno adecuado para que se ejecuten correctamente los procesos implica:

- Gestionar los recursos básicos del sistema:
  - Realizar la gestión de la memoria, haciendo un seguimiento de qué partes de la misma están siendo usadas y por quién.
  - Creación, borrado, sincronización y planificación de procesos.
  - Mecanismos para la intercomunicación entre procesos.
  - Mecanismos básicos de entrada/salida.
- Proporcionar servicios esenciales a los programas de aplicaciones y usuarios:
  - Autentificación de usuarios, ocupándose de la protección y seguridad del propio sistema operativo y de los usuarios.

- Control de acceso a los recursos por parte de los procesos.
- Gestión de ficheros, proporcionando sistemas de archivos para representar archivos, directorios y gestionar el espacio en los dispositivos de almacenamiento masivo.

El núcleo está constituido directamente sobre el *hardware*, siendo la parte del sistema operativo que depende de la máquina, por esto debe contener el código ensamblador.

Nota

En el lenguaje ensamblador se establece una relación uno a uno entre cada instrucción en código máquina y una palabra mnemotécnica.

El resto del sistema operativo puede estar programado en un lenguaje de más alto nivel, lo que hace que el desarrollo del código y su posterior mantenimiento sea más fácil.

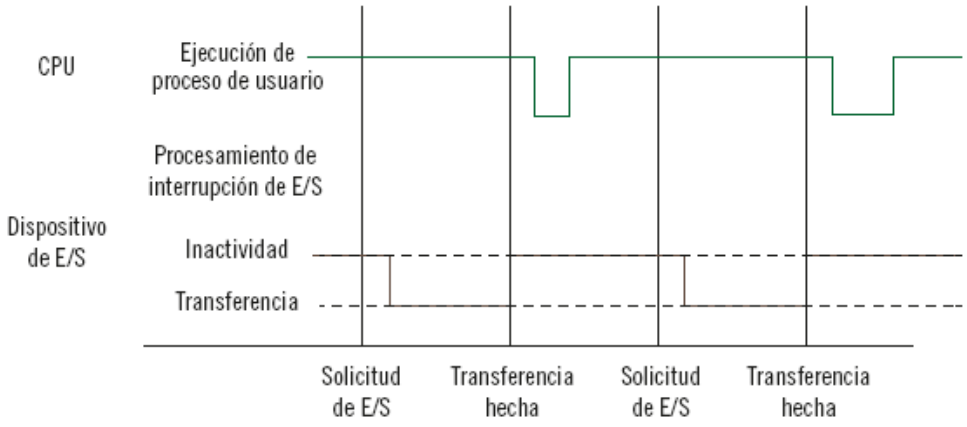
Sabía que...

Desde la aparición del sistema operativo *Unix*, los sistemas operativos suelen desarrollarse en lenguaje C.

El núcleo necesita unos requisitos mínimos de *hardware* para realizar su cometido. Estos requisitos son:

- **Mecanismo de interrupciones**. El *hardware* debe proporcionar un mecanismo para interrumpir el funcionamiento normal de la CPU. Cuando ocurre un suceso, se indica mediante una **interrupción**, que puede ser *hardware* (enviándole una señal a través del bus del sistema) o *software* (el *software* puede ejecutar una operación especial denominada **llamada del sistema**). Las **interrupciones** permiten mejorar el aprovechamiento de la CPU.

Diagrama de tiempos de una interrupción para un proceso de salida



- **Protección de memoria**. Al producirse la ejecución de forma concurrente de varios procesos, es necesario proteger el uso de la memoria de cada uno de ellos y evitar el acceso no autorizado.
  Definición. Procesos concurrentes: aquellos que son ejecutados al mismo tiempo y que potencialmente pueden interactuar entre sí.
- **Un conjunto de instrucciones reservadas**. Para evitar las interferencias entre procesos concurrentes, parte de las instrucciones del computador se reservan de forma exclusiva para el sistema operativo. Estas instrucciones son utilizadas para:

  - Habilitar/deshabilitar interrupciones.
  - Acceso a registros usados por el *hardware* de protección de memoria.
  - Realizar operaciones de entrada/salida.
  - Hacer que la CPU cambie de proceso.

- **Reloj de tiempo real**. Para planificar la ejecución de los procesos, es esencial disponer de un reloj *hardware* que produzca interrupciones a intervalos fijos.

3. Subsistemas del núcleo

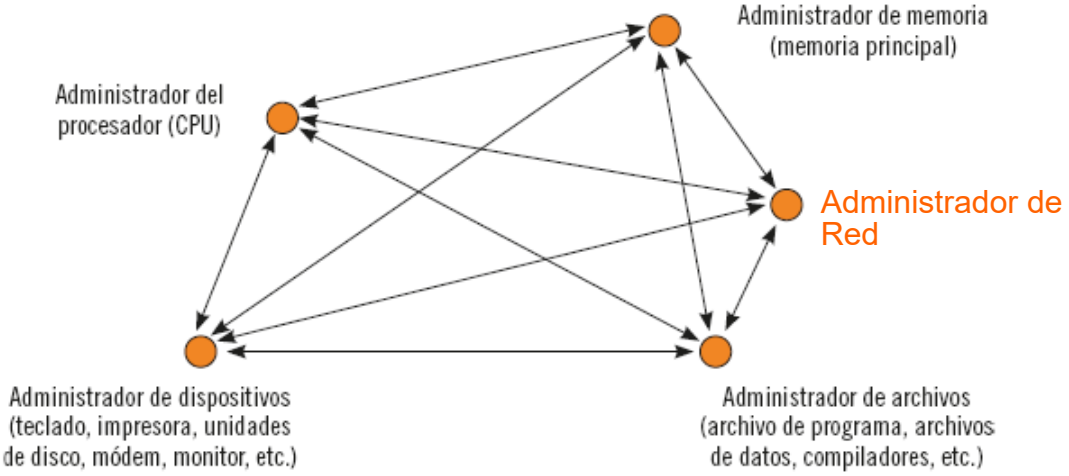
En la imagen siguiente, se va a mostrar una representación abstracta de las partes más importantes de un sistema operativo y cómo estas partes interactúan entre ellas.

Se van a mostrar los administradores esenciales de todo sistema operativo, que son: el administrador de memoria, el administrador del procesador, el administrador de dispositivos y el administrador de archivos. Además, aparece un administrador de red, ya que la mayor parte de los sistemas operativos actuales lo incorporan.

Cada uno de los administradores realiza las siguientes tareas:

- Supervisar continuamente sus recursos.
- Planificar (asignando y desasignando) qué procesos utilizan —y de qué manera — los recursos.

Representación abstracta de las partes más importantes del sistema operativo

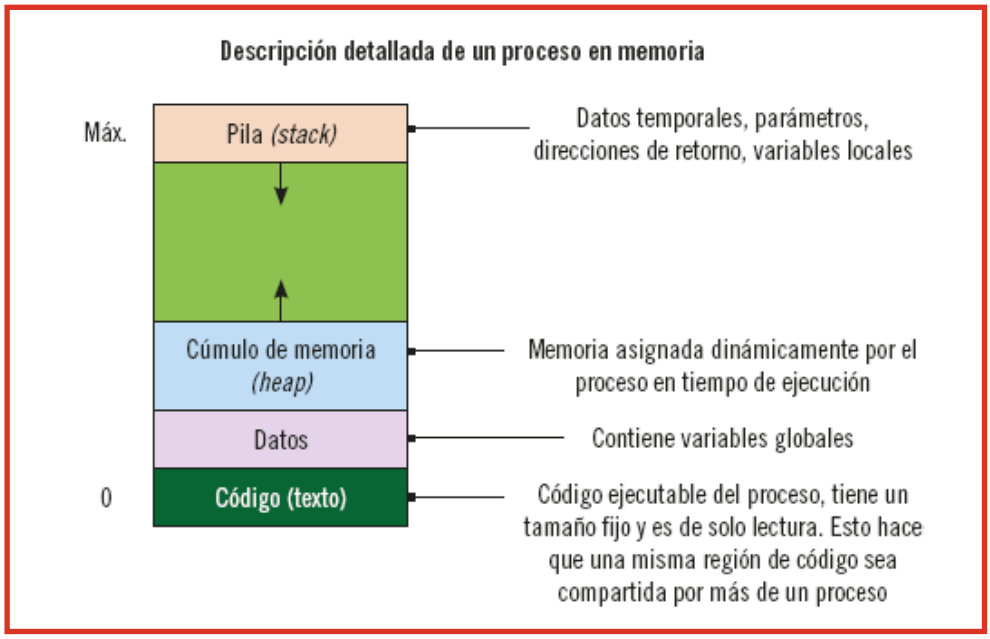


A continuación, se detallan los subsistemas del núcleo.

3.1. Gestión de procesos

Se define un proceso de forma informal como un programa en ejecución, pero un proceso es más que el código de un programa, ya que además incluye:

- **Espacio de direcciones**. En la siguiente imagen aparece representado un proceso en memoria, es decir, el espacio de direcciones de un proceso.



- **Actividad actual**. Está representada por el registro **contador del programa** y los contenidos del resto de los registros de la CPU.

Nota

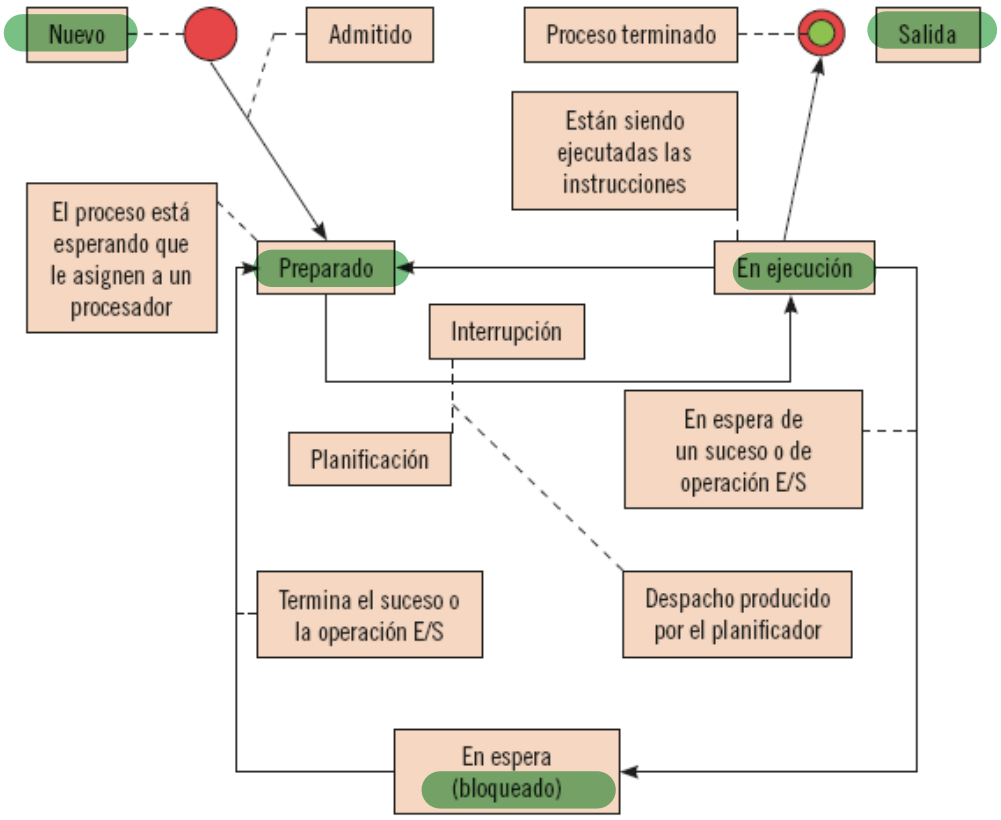
La ejecución de un programa es una sucesión de instrucciones que se almacenan en memoria. En el contador de programa se indica cuál es la siguiente instrucción a ejecutar.

Estado de proceso

Un proceso puede encontrarse en tres estados básicos. A medida que se va ejecutando el proceso va cambiando de estado.

En el diagrama siguiente se aprecia el paso por los distintos estados de un proceso.

Diagrama detallado de estados de un proceso



Importante

Un proceso puede ser creado por los siguientes motivos:

- Un nuevo usuario se conecta.
- El sistema operativo crea un proceso para realizar un servicio concreto.
- Un proceso creado por otro.
- Es el siguiente trabajo de un procesamiento por lotes *(batch)*.

Un proceso puede terminar por los siguientes motivos:

- El proceso ejecuta una llamada al sistema para indicar que ha terminado.
- El proceso necesita más memoria de lo que se le ha reservado.
- Se excede de un límite de tiempo.
- Errores de protección, aritméticos, errores en el uso de instrucciones por parte del usuario.

- El administrador del sistema decide terminar el proceso.

Un proceso puede ser suspendido por:

- El sistema operativo lleva el proceso a disco *(swapping)* para permitir que se ejecuten otros procesos.
- El sistema operativo suspende el proceso por interbloqueo.
- El usuario suspende el proceso para depurarlo.
- Un proceso que se ejecuta de forma periódica y que está a la espera del siguiente intervalo.
- Un proceso hijo quedará suspendido a causa de su proceso padre.

Los procesos se representan en el sistema operativo mediante un **Bloque de Control de Proceso** (PCB, *Process Control Block*).

Una **PCB** es una estructura de datos que contiene la información relativa a cada proceso:

- Identificador de proceso o pid *(process identifier)*.
- Estado del proceso.
- Valores de los registros de CPU (contador del programa, etc.).
- Datos para la gestión de recursos:
  - Memorias (tablas de páginas, etc.).
  - E/S (demandas, tablas de dispositivos asignados, etc.). E/S se refiere a los dispositivos de Entrada/Salida, es decir, cuando se produce un intercambio de información de entrada o de salida.
  - Procesador (prioridades, punteros a colas, etc.).
- Datos de contabilización (tiempo de uso del procesador, etc.).

Sabía que...

Existen operaciones para ver un listado de los procesos activos, así como parte de los atributos de los procesos que se han visto, por ejemplo, el identificador o pid.

Para *Windows*, se puede utilizar, desde la consola el comando *tasklist*.

Para sistemas *Unix*, se puede utilizar, desde terminal el comando *top*.

Aplicación práctica

Indique en qué estado están los siguientes procesos en las situaciones que se describen a continuación:

- **Un proceso está en ejecución y se produce una interrupción.**
- **Un proceso acaba de terminar de ejecutar todas sus instrucciones.**
- **Un proceso se está ejecutando y necesita realizar una operación de E/S. El dispositivo que necesita tiene varios procesos antes de él.**
- **Un proceso se está ejecutando, crea un proceso hijo, este necesita realizar una operación E/S y queda en estado bloqueado, ¿en qué estado queda el padre?**
- **Un proceso está en ejecución y realiza una operación que provoca un fallo de memoria, escribiendo en una dirección de memoria no permitida.**

SOLUCIÓN

Un proceso está en ejecución y se produce una interrupción: el proceso vuelve al estado preparado.

Un proceso acaba de terminar de ejecutar todas sus instrucciones: el estado del proceso es terminado.

Un proceso se está ejecutando y necesita realizar una operación de E/S, el dispositivo que necesita tiene varios procesos antes de él: el proceso pasa a estado bloqueado.

Un proceso se está ejecutando, crea un proceso hijo, este necesita realizar una operación E/S y queda en estado bloqueado, ¿en qué estado queda el padre?: el padre sigue en ejecución.

Un proceso está en ejecución y realiza una operación que provoca un fallo de memoria, escribiendo en una dirección de memoria no permitida: el proceso pasa a estado terminado.

Planificación de procesos

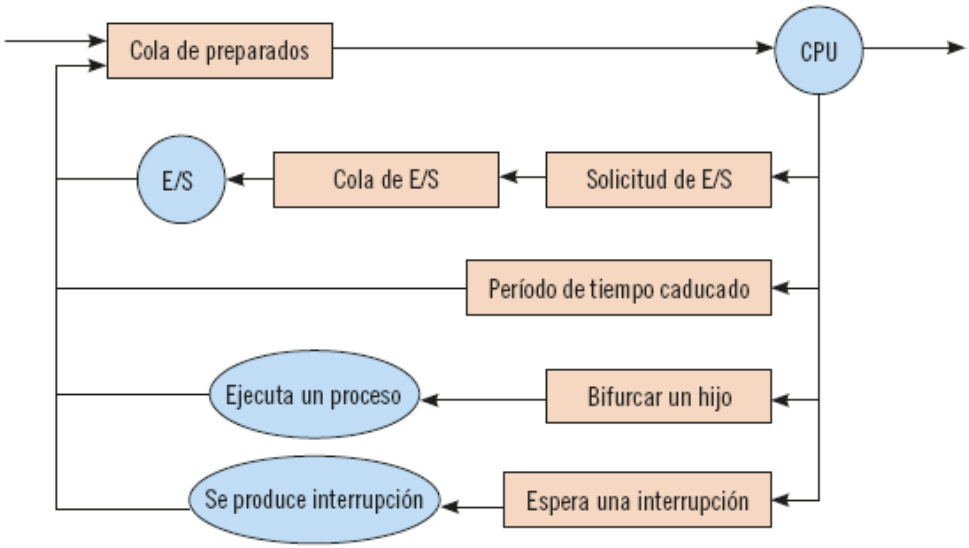
Una de las funciones básicas del núcleo es planificar los procesos que se van a ir ejecutando. Para esto, dentro del núcleo se encuentra:

- Planificador de bajo nivel o ***dispatcher:*** módulo del núcleo que se encarga de asignar CPU al proceso que está en cabecera, en la cola de procesos en estado preparado.

- ***Scheduler:*** se encarga de la selección del siguiente proceso.

La planificación de los procesos va a necesitar de varias colas de almacenamiento. En primer lugar, cuando un proceso entra al sistema ingresa en una **cola de trabajo**. Cuando el proceso es admitido y pasa a estado preparado, la cola a la que pasa es la **cola de procesos preparados**, en la que los procesos están dispuestos en un orden definido. Además, cada dispositivo de E/S tiene su propia **cola del dispositivo**, en la que esperan los procesos que necesitan ese recurso.

Diagrama de colas para la planificación de procesos



Hebras (threads)

Una hebra es la unidad básica de utilización de la CPU que puede planificarse y ejecutarse. La manipulación de las hebras consume menos tiempo que la manipulación de los procesos, ya que estos son más complicados.

Una hebra está constituida por:

- Identificador de hebra.
- Contador de programa.
- Conjunto de registros.
- Pila de ejecución.

Varias hebras de un mismo proceso comparten la sección del código, la sección de datos y otros recursos del sistema operativo.

Un proceso tradicional tiene una sola hebra de control. Si un proceso tiene varias hebras de control, puede realizar más de una tarea a la vez.

Con la utilización de multihebra se incrementa la capacidad de respuesta, ya que permite compartir recursos y disminuir la sobrecarga.

Una biblioteca de hebras proporciona a los programadores una API, que crea y gestiona hebras. Las tres principales son:

- Posix *(Pthread)*. Hay muchos sistemas operativos que la implementan: *Linux, macOS* y *Unix*.
- Posix *(Portable Operating System Interface)* representa un conjunto de estándares implementados principalmente en sistemas operativos basados en Unix. *Linux, Solaris* y *macOS* son ejemplos de sistemas compatibles con Posix.
- Win32, disponible en los sistemas *Windows*.
- Java. Las hebras son los modelos de ejecución de programas fundamentales para los programas en Java. Lenguaje de programación con una rica API y soporte integrado para la creación y gestión de hebras. Los programas de Java se ejecutan en cualquier sistema operativo que permita el uso de una máquina virtual Java (JVM, *Java Virtual Machine)*.

Definición

***Application Programing Interface* (API)**
Conjunto de funciones y procedimientos contenidos dentro de una biblioteca para ser usados por otro *software*.

A continuación, se enumeran las diferentes llamadas al sistema y funciones en *Unix* y *Windows* para operar con procesos y con hebras.

- API de procesos:

Api de procesos

Operación	Unix	Win32
Crear	fork() exec()	CreateProcess()
Terminar	_exit()	Exitprocess()
Obtener código de finalización	wait() witpid()	GetExitCodeProcess()
Identificador	getpid()	GetCurrentProcessId()
Terminar otro proceso	kill	Terminateprocess()

- API de hebras:

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

Aplicación práctica

Con el siguiente trozo de un programa en C, se pretende que un proceso padre cree una hebra cuya misión es escribir un mensaje:

```
main()
{
  int tNum[numThreads];
  HANDLE hThread[numThreads];
  for (int i = 0; i < numThreads; i++)
  {
      tNum[i] = i;
     hThread[i] = CreateProcess(NULL, 0, threadFunc, &tNum[i],0,
NULL);
}
```

La función que aparece en negrita es la que se ha utilizado para crear la hebra. ¿Se está usando la función correcta?

SOLUCIÓN

No, porque la función que hay que utilizar, tal como aparece en el cuadro de funciones de la API, es la siguiente:

```
main()
{
  int tNum[numThreads];
  HANDLE hThread[numThreads];
  for (int i = 0; i < numThreads; i++)

      tNum[i] = i;
    hThread[i] = CreateThread(NULL, 0, threadFunc, &tNum[i],0,
NULL);
}
```

3.2. Gestión de memoria

La memoria principal puede definirse como una matriz de celdas que almacenan datos e instrucciones. Cada celda está identificada por un número o dirección de memoria y la información que se almacena en cada celda es una palabra o byte.

Definición

Byte
Número de bits utilizados para representar un carácter en un sistema de codificación dado.

Un ejemplo de cómo se podría definir una dirección que haga referencia a conjunto de celdas de memoria principal sería: de la 000F00h a la 000FFFh.

La memoria principal es compartida por la CPU y por los dispositivos de E/S. Para que un programa se ejecute, su código y datos deben estar cargados en memoria, generando direcciones absolutas, de manera que, cuando este termina, su espacio de memoria queda libre. También deben residir en memoria las rutinas del sistema operativo.

Por otro lado, para que la CPU procese datos de un dispositivo, por ejemplo de un disco, estos deben transferirse primero a la memoria principal, a través de las llamadas de E/S que genera la CPU. Además, estas instrucciones deben estar en memoria para que sean ejecutadas.

La gestión de la memoria debe incidir en estos aspectos fundamentales:

- Permitir la ejecución concurrente de los procesos sin que existan problemas por el uso común de la memoria.
- No permitir el uso de la zona de memoria destinada al sistema operativo.
- El espacio disponible se debe poder utilizar en su totalidad.
- La gestión de la memoria no debe penalizar el rendimiento total.

Conversión de direcciones (reubicación)

Los procesos recorren una serie de pasos antes de llegar a ser ejecutados. A lo largo de estos pasos, las direcciones pueden representarse de distinta forma.

Las direcciones del programa fuente son simbólicas. El compilador se encargará de reasignar estas direcciones a direcciones **reubicables** (direcciones **binarias**). El cargador se encargará de **reasignar** estas direcciones a **absolutas**.

En la imagen que se muestra a continuación, se pueden ver los distintos pasos en el procesamiento de un programa de usuario.

Importante

Dirección física: indica la posición de una celda de la memoria física.

Dirección lógica o virtual: estas direcciones son las que utilizan los programas, son simbólicas.

Unidad de Manejo de Memoria (MMU): para traducir las direcciones simbólicas o virtuales a las absolutas o físicas, se necesita este dispositivo. Se encarga de:

- Convertir las direcciones lógicas emitidas por los procesos en direcciones físicas.
- Comprobar que esta conversión se puede realizar.

❙ Comprobar que el proceso intenta acceder a una dirección de memoria permitida.

Espacio de direcciones lógicas: conjunto de las direcciones virtuales que utiliza un programa.

Espacio de direcciones físicas: conjunto de direcciones que resultan de traducir el espacio de direcciones lógicas.

Es posible encontrar dos mecanismos que contribuyen a mejorar la utilización del espacio de la memoria:

- **Carga dinámica:** con este mecanismo, cuando se produce una llamada a una rutina, se carga en memoria, de manera que, si no se produce la llamada, no se cargará nunca.
- **Enlace o montaje dinámico:** mecanismo similar a la carga dinámica, solo que lo que se pospone no es la carga, sino el enlace o montaje. Esto se utiliza en el caso de bibliotecas dinámicas (DLL).

Sabía que…

Gracias al enlace o montaje dinámico, se puede sustituir una biblioteca por una nueva versión y todos los programas que hacen referencia a dicha biblioteca emplean automáticamente esta última versión. De lo contrario, habría que enlazar o montar de nuevo la biblioteca.

Intercambio

Los procesos pueden ser intercambiados temporalmente por distintas causas, desasignándoles la memoria *(swap out)*, almacenándolos en una zona de respaldo y cargándolos de nuevo para continuar su ejecución *(swap in)*.

Normalmente, un proceso descargado volverá a cargarse en el mismo espacio de memoria que ocupaba, aunque esto depende de si la reasignación de direcciones se realiza en el tiempo de ensamblado o de carga.

Para este mecanismo, se requiere un espacio de almacenaje de respaldo que sea lo suficientemente rápido y grande.

Es necesario que el proceso que se quiere intercambiar esté inactivo, por ejemplo que no esté pendiente del acceso a una operación de E/S.

Importante

Para llevar a cabo el intercambio o *swapping* es necesario tener en cuenta lo siguiente:

- Es necesario un proceso intercambiador.
- Hay que definir los criterios para elegir el proceso a descargar (política de *swapping out)*.
- Hay que definir los criterios para elegir el proceso a cargar (política de *swapping in)*.
- Debe haber espacio en disco para almacenar las imágenes de los procesos.
- Es necesario definir los criterios para la gestión del espacio de intercambio.

Gestión de memoria contigua

La memoria principal debe contener tanto el sistema operativo como los diversos procesos de usuario, de forma eficiente. Para ello, uno de los métodos más utilizados es la asignación de memoria contigua.

Uno de los métodos más simples consiste en dividir la memoria en varias particiones de tamaño fijo. Cada partición puede contener un proceso, de manera que el grado de multiprogramación está limitado al número de particiones disponibles. Cuando una partición está libre, se selecciona otro proceso de la cola de entrada y se carga en ella. Este método se denomina método de **particiones múltiples (MTF)**.

Una generalización de este método es el método de **particiones de tamaño variable (MVT)**.

Los mecanismos para gestionar la memoria contigua son:

- El sistema operativo mantiene una tabla que indica qué partes de la memoria están ocupadas o libres, denominada **Tabla de Descripción de Particiones (TDP)**.
- El sistema operativo gestiona una lista de huecos libres de memoria (también llamados agujeros) y selecciona qué procesos pueden cargarse en memoria para ser ejecutados.
- La existencia de unas primitivas internas para pedir y liberar memoria.

Las estrategias más utilizadas para seleccionar el agujero libre entre un conjunto de agujeros disponibles son:

- Primer ajuste: se asigna el primer agujero lo suficientemente grande.

- Mejor ajuste: se asigna el agujero más pequeño de los que tengan el tamaño suficiente.
- Peor ajuste: se asigna el agujero más grande de los que tengan el tamaño suficiente.

Las estrategias de primer y mejor ajuste tienen un problema, denominado **fragmentación externa**, que consiste en que, a medida que se cargan los procesos y se libera ese espacio, la memoria libre se va descomponiendo en pequeños fragmentos.

Hay dos técnicas complementarias que permiten solucionar este problema: la **paginación** y la **segmentación**.

Paginación

La paginación es una técnica para el aprovechamiento de la memoria y para solucionar la fragmentación externa. La técnica consiste en dividir la memoria física en unos bloques de tamaño fijo, denominados **marcos**. La memoria virtual se divide de la misma manera en bloques denominados **páginas**. El espacio de memoria virtual de un proceso en ejecución se va a dividir en páginas que encajarán en los marcos de memoria que haya disponibles.

Las direcciones lógicas se componen de dos partes, el número de página y el desplazamiento.

La MMU se encarga de asociar el número de página lógico con el marco de página asignado, para lo que emplea una **tabla de páginas**.

Actividades

1. Busque un diagrama que muestre un esquema sencillo de paginación.

Segmentación

La segmentación es un esquema de gestión de la memoria de manera que se asemeja a la forma en la que están construidos los programas, es decir, un programa se va a descomponer en varios segmentos de memoria: código, datos, pila, etc.

Un espacio lógico de direcciones es una colección de segmentos. Cada uno tiene un nombre y una longitud y, por lo tanto, un usuario especificaría cada dirección con dos valores: un nombre del segmento y un desplazamiento.

Con el *hardware* adecuado, se pueden ubicar esos segmentos en zonas de memoria no contiguas, para lo que se utiliza una **tabla de segmentos**.

Actividades

2. Busque un diagrama que muestre un esquema sencillo de segmentación.
3. ¿Por qué se combinan en ocasiones en un único esquema los mecanismos de segmentación y paginación?

3.3. Sistema de ficheros

El sistema de ficheros proporciona los mecanismos para administrar los dispositivos de almacenamiento masivo (por ejemplo discos).

Está compuesto de dos partes diferenciadas: una colección de ficheros (cada uno de ellos almacena información relacionada) y una estructura de directorios, que organiza todos los ficheros del sistema y da información sobre ellos. Además, es necesario controlar quién y cómo puede acceder a un fichero, utilizando determinadas técnicas de protección y seguridad.

Conceptos

Un fichero se define como una colección de información relacionada con un nombre que se graba en el almacenamiento secundario. Normalmente, los ficheros representan programas y datos.

Nota

El almacenamiento secundario se lleva a cabo casi exclusivamente sobre discos.

Un fichero tiene una determinada estructura definida, que dependerá del tipo de fichero que sea.

Los atributos de un fichero, en términos generales —ya que varían según el sistema operativo—, son:

- **Nombre:** nombre simbólico establecido por el usuario.
- **Identificador:** identifica de forma única el fichero dentro del sistema de ficheros.
- **Tipo:** para sistemas que permiten varios tipos.
- **Ubicación:** puntero a un dispositivo y su posición en él.
- **Tamaño:** tamaño actual del fichero en bits, bytes o bloques.

- **Posición actual:** puntero a la posición actual en el fichero si se está realizando una operación de lectura o escritura.
- **Protección:** información sobre el control de acceso.
- **Recuento de uso:** valor que indica el número de procesos que están usando el fichero, es decir, que lo han abierto.
- **Hora, fecha e identificación del usuario:** información referente a la creación, última modificación y último uso del fichero. Estos datos son útiles desde el punto de vista de la protección, seguridad y monitorización del uso de fichero, modificación y último acceso.

La estructura de directorios almacena la información acerca de los ficheros y también se almacena en el almacenamiento secundario. Una entrada de directorio está compuesta por:

- Nombre del fichero.
- Identificador (este es único y permite encontrar el resto de los atributos del fichero).

El sistema operativo puede realizar las siguientes operaciones básicas relacionadas con ficheros:

- **Creación:** tiene que realizar dos pasos, en primer lugar, encontrar espacio para el fichero dentro del sistema de ficheros y, en segundo lugar, incluir en el directorio una entrada para el nuevo fichero.
- **Escritura:** hay que realizar una llamada al sistema, especificando el nombre del fichero y la información que hay que escribir en él. El sistema debe mantener un puntero de escritura.
- **Lectura:** hay que realizar una llamada al sistema que especifique el nombre del fichero y dónde debe colocarse el siguiente bloque del fichero. El sistema necesita mantener un puntero de lectura.
- **Búsqueda dentro de un fichero:** se explora dentro del fichero para encontrar la entrada y se reposiciona el puntero de posición actual dándole el nuevo valor.
- **Borrado:** se busca en el directorio el fichero indicado y se libera esa entrada.
- **Truncado:** el usuario puede borrar parte de un fichero. Esta función permite mantener los atributos originales del fichero (excepto la longitud).

Importante

Cuando se está trabajando con ficheros, es importante realizar dos operaciones: la apertura del fichero y el cierre de este.

Un ejemplo de estas operaciones en C es: open() y close().

Aplicación práctica

Se tiene un sistema gestionado por el método de particiones de tamaño variable (MVT). La memoria física de la que se dispone tiene 4.200 bytes. Al principio, la memoria está ocupada por 3 procesos de esta forma:

Número de proceso	Dirección inicial	Longitud del proceso
P1	1.000	150
P2	2.900	500
P3	3.400	600

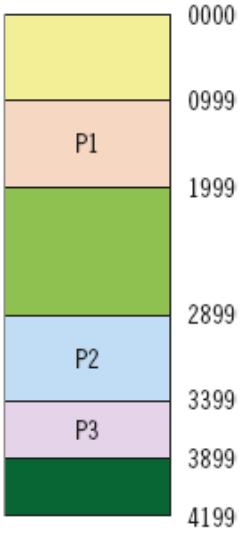
La estrategia que se va a seguir cuando se carga un proceso nuevo es la del mejor ajuste, que consiste en que se empieza buscando hueco por la dirección más baja, hasta encontrar el hueco con tamaño suficiente.

Hay que cargar tres procesos de 500 (P4), 1.000 (P5) y 200 (P6), en este orden. Después hay que describir el contenido de la memoria.

¿Ocurre algo destacable en la memoria tras esta última carga de procesos?

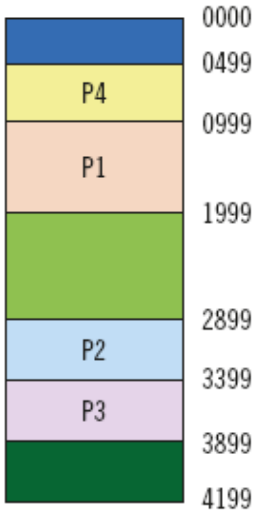
SOLUCIÓN

Inicialmente, la memoria principal está así: 0999

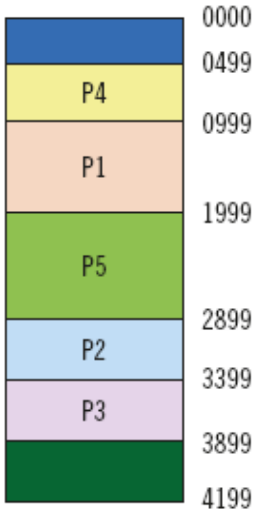


Después de cargarse, los procesos siguientes quedarían:

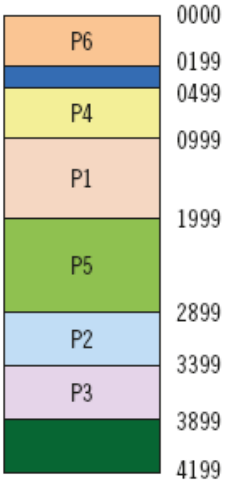
- P4



- P5



P6



La memoria empieza a fragmentarse en huecos pequeños, lo que se denomina fragmentación externa.

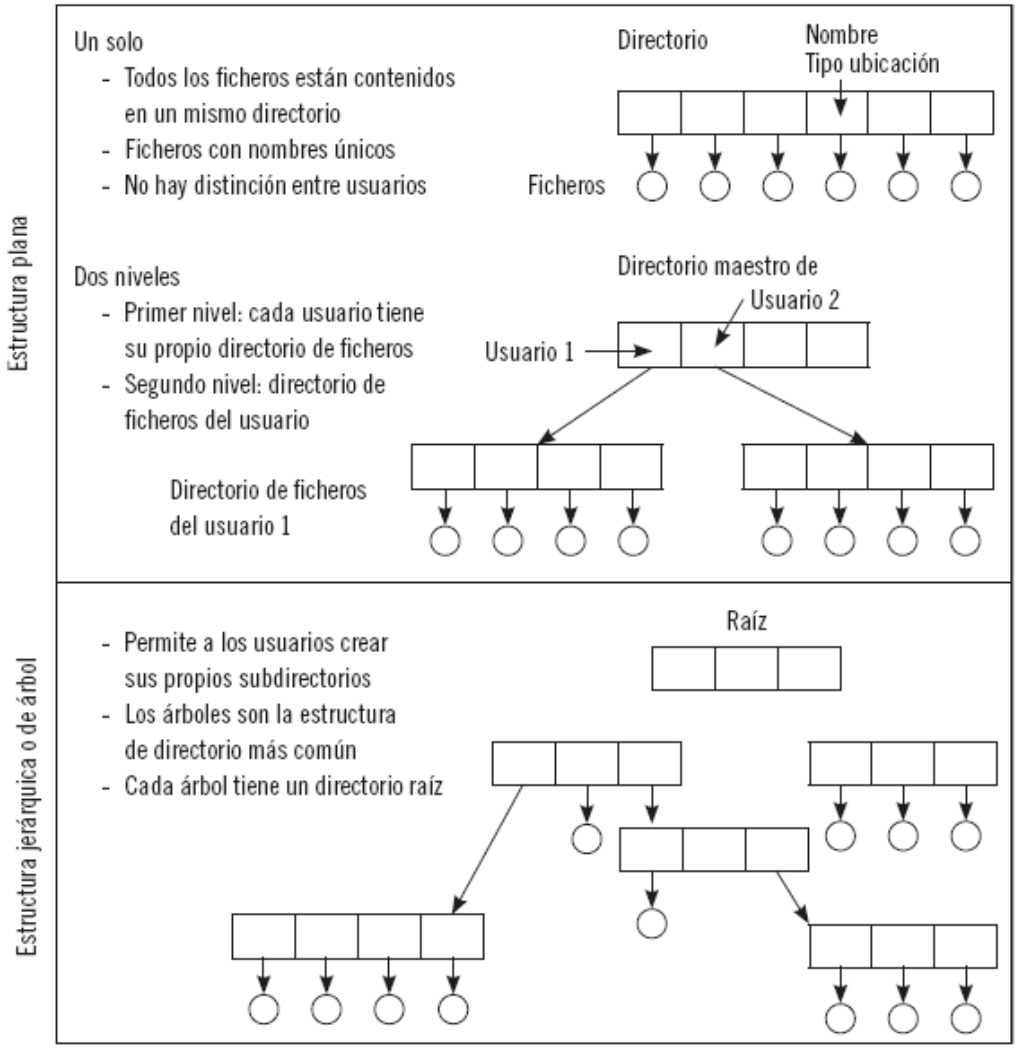
Estructura de directorios

Los sistemas de ficheros tienen una gran complejidad y, por lo tanto, se necesita organizar esta información de una forma rápida y eficiente. Para eso se utilizan los directorios.

Se puede considerar un directorio como una tabla de símbolos con la que se traducen los nombres de los ficheros a sus correspondientes entradas de directorio. Puede organizarse de muchas maneras, pero las operaciones que se van a realizar son las siguientes: buscar, crear y borrar un fichero, listar un directorio, renombrar un fichero y recorrer el sistema de ficheros.

Los esquemas más comunes para definir la estructura lógica de un directorio son los siguientes:

Diagrama para definir las principales estructuras de directorios



Protección

Es necesario proporcionar mecanismos de protección a los ficheros ya que son el principal mecanismo para el almacenamiento de la información.

Para ello, se controla la forma en la que se accede al fichero, de manera que se tienen los siguientes tipos de acceso: lectura, escritura, ejecución, adicción o inserción de nueva información, borrado, etc.

La protección de ficheros puede realizarse mediante contraseñas, lista de acceso de usuarios al fichero en cuestión o mediante otras técnicas.

Actividades

4. Busque información sobre la estructura de directorio tipo grafo.

3.4. Control de dispositivos

Además de controlar procesos, espacio de direcciones y ficheros, un sistema operativo también controla todos los dispositivos de E/S.

Debido a la gran cantidad de dispositivos que hay, se necesitan diversos métodos para controlarlos, dichos métodos forman parte del subsistema de E/S del núcleo o *kernel*.

Debe proporcionar comandos a los dispositivos que les permitan la utilización de interrupciones, controlar e identificar los errores.

Conceptos importantes

Las unidades E/S consisten en un componente mecánico y uno electrónico. A la parte electrónica se le llama **controlador de dispositivo** o **adaptador**. La parte mecánica es el dispositivo en sí. Los controladores de dispositivo proveen al subsistema de E/S de una interfaz uniforme de acceso a los dispositivos.

Desde el punto de vista del *hardware*, los dispositivos se comunican enviando señales a través de un cable o por el aire, mediante un punto de conexión o **puerto**. Si utilizan un conjunto de hilos común, la conexión se denomina **bus**. Este tipo de conexiones van acompañadas de un protocolo que dirige los mensajes que se pueden enviar a través de los hilos.

Una **controladora** es una colección de componentes electrónicos que permite controlar un puerto, un bus o un dispositivo. El paso de comandos y datos entre el procesador y la controladora, para llevar a cabo una transferencia de E/S, se realiza a través de uno o más registros para los datos y las señales de control.

La controladora de dispositivo puede soportar una E/S mapeada en memoria, es decir, que estos registros de control del dispositivo están mapeados en el espacio de direcciones del procesador.

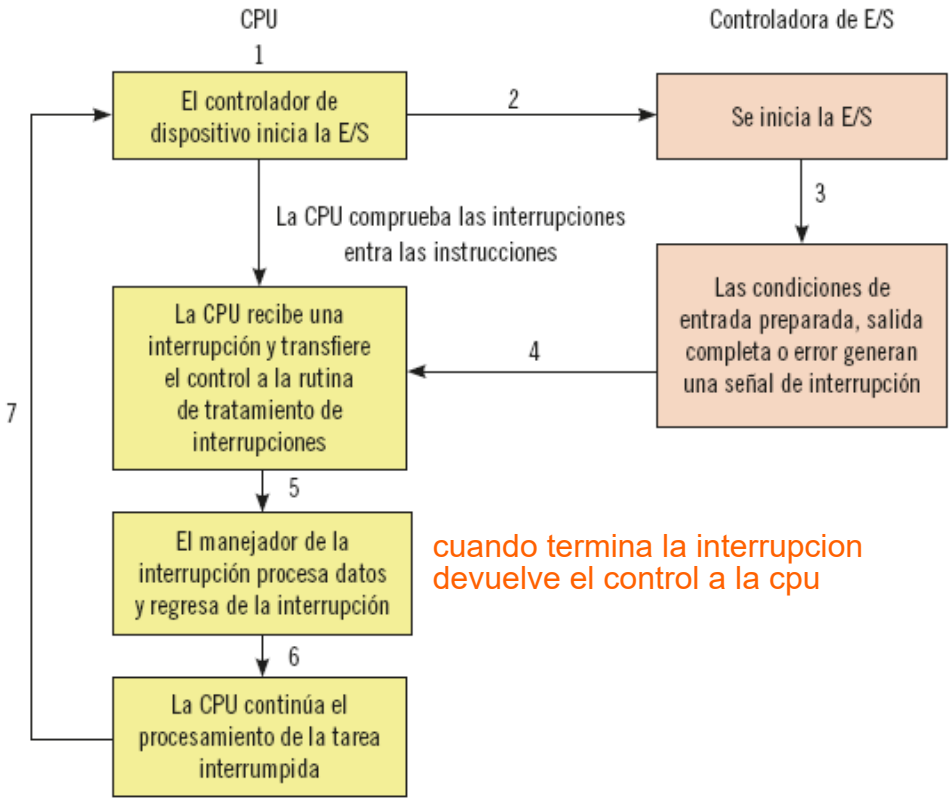
Definición

Mapa de memoria
Organización de las distintas unidades de memoria en el espacio de direcciones de un procesador.

Entre el *host* y la controladora se produce una **negociación**, que determina en qué momento escribe el *host* y cuándo cambia los bits que indican el estado, pero para esto el *host* debe estar continuamente preguntando si el dispositivo está disponible.

Una forma más eficiente es el mecanismo de *hardware* que permite notificar eventos a la CPU, que se denomina **interrupción**.

Ciclo de E/S dirigido por interrupción



cuando termina la interrupcion devuelve el control a la cpu

Para mover datos entre los dispositivos y la memoria principal, la CPU utiliza una E/S programada o se descarga en una controladora de **acceso directo a memoria** o **DMA** *(Direct Memory Access)*.

Desde el punto de vista del *software*, el objetivo es que sea posible abstraerse del tipo de dispositivo E/S que se está tratando, ya que, como se ha mencionado ante-

riormente, existen muchos y muy diversos dispositivos. Para acceder a cada tipo de dispositivo, se utiliza un conjunto estándar de funciones, es decir, una **interfaz**.

A grandes rasgos, los dispositivos de E/S pueden clasificarse en los siguientes tipos:

- **Dispositivos de bloque:** almacenan información en bloques de tamaño fijo, cada uno con su propia dirección. Todo el intercambio de información se hace en bloques de uno o más bloques completos. Ejemplos de este tipo de dispositivo son los discos duros, CD-ROM, memorias USB, etc.
- **Dispositivos de caracteres:** realizan transferencias de información a través de una secuencia de caracteres. Esta información no es direccionable y no tiene la posibilidad de realizar operaciones de búsqueda. Ejemplo de este tipo de dispositivos son impresoras, interfaces de red, monitores, etc.
- **Otros dispositivos:** algunos dispositivos no encajan en los dos tipos anteriores porque no manejan bloques direccionables ni secuencia de caracteres, este es el caso de los relojes. Estos realizan interrupciones a intervalos de tiempo definidos.

Actividades

5. Defina las ventajas y desventajas de utilizar un mecanismo de E/S mapeado en memoria para los registros de control de dispositivos.

3.5. Comunicaciones

Una de las funciones principales del núcleo es la comunicación entre procesos, también llamada **IPC** *(Inter-Process Communication)*. El objetivo es establecer mecanismos que permitan a los procesos comunicarse y sincronizarse entre sí.

Los procesos que se pueden ver afectados por otros procesos que están ejecutándose en el sistema, también llamados **procesos cooperativos**, pueden compartir un espacio de direcciones lógico, tanto código como datos, o compartir los datos únicamente, a través de ficheros o mensajes.

Hay que tener en cuenta tres cuestiones:

- Cómo se pasan la información de un proceso a otro.
- Es necesario asegurarse de que dos o más procesos no invadan el sitio de otro.
- La secuencia adecuada de procesos cuando hay dependencias entre ellos.

Se denomina **condición de carrera** a una situación en donde varios procesos manipulan o acceden a los mismos datos de forma concurrente y el resultado de la ejecución depende de si el orden en el que se realizan los accesos es el correcto.

Cada proceso tiene un segmento de código llamado **sección crítica**. En esta sección el proceso puede acceder a memoria o ficheros compartidos. Para evitar que se dé el problema de la condición de carrera, se necesita seguir estos requerimientos:

- **Exclusión mutua:** dos procesos no pueden estar simultáneamente en sus regiones críticas.
- **Progreso:** si ningún proceso está ejecutando su sección crítica, los candidatos a entrar son los que no estén ejecutando el resto de sus secciones.
- **Espera limitada:** un proceso no puede esperar infinitamente a entrar en su sección crítica. Hay un límite de veces de espera.

Existen soluciones *software*, como la solución de Peterson, y soluciones *hardware*, mediante una simple herramienta, el cerrojo *(lock)*, o basadas en las instrucciones TestANSet() y Swap(), aunque son soluciones complicadas.

Semáforos

Una herramienta de sincronización *software* más sencilla se denomina semáforo. Se puede ilustrar cómo funciona un semáforo con una codificación en C que muestre sus funciones principales.

Se supone que el semáforo va a estar implementado por una variable entera a la que se va a llamar S. Esta variable solo puede ser modificada mediante las operaciones wait() y signal(). A continuación, se ve cómo sería el código de estas operaciones:

wait(S)	signal(S)

`wait(S)`
`{`
`while S<=0`
`;  // no-operación`
`S--;`
`}`
	`signal(S)`
	`{`
	`S++;`
	`}`

Cuando un proceso modifica el valor del semáforo, ningún otro proceso puede modificar simultáneamente el valor de dicho semáforo.

Los sistemas operativos diferencian entre **semáforos contadores** y **semáforos binarios o cerrojos mutex** (el valor solo puede ser 0 o 1, iniciado en 1). Este último puede servir en el caso de múltiples procesos; los n procesos comparten el semáforo.

Los semáforos contadores se usan para un número finito de instancias. Se inicializa con el número de recursos disponibles:

- Cada proceso que desea usar un recurso ejecuta wait() en el semáforo (decrementa la cuenta).
- Cuando un proceso libera un recurso, ejecuta signal() (incrementa la cuenta).
- Cuando la cuenta del semáforo llega a 0, todos los recursos están en uso. Después, los procesos que quieran usar un recurso se bloquean hasta que la cuenta sea mayor que 0.

Un proceso se bloquea cuando entra en el bucle wait(). Para evitar esta espera, cuando un proceso entra en el bucle signal() debe poder ser reactivado. Para ello se utiliza la operación wakeup(); (el estado del proceso pasa de en espera a preparado), y pasa a una cola de procesos preparados.

A continuación, se muestra cómo se codifican wait() y signal(), utilizando estas últimas operaciones. En esta ocasión, el semáforo va a definirse como una estructura que contendrá un número entero y una lista para almacenar los procesos en espera.

```
typedef struct
{
int value;
struct process *list;
} semaforo;
```

Una operación wait() añade el proceso a la lista de procesos en espera y llama a una operación block() que suspende el proceso que la ha invocado.

Una operación signal() elimina un proceso de la lista de procesos en espera y lo despierta wakeup(), dejando de estar en estado bloqueado.

wait(semaforo *S)	signal(semaforo *S)

wait (semaforo *S)
{
S->value--;
if (S->value <0)
{
añadir proceso a S->list;
block();
}}
	signal (semaforo *S)
	{
	S->value++;
	if (S->value <= 0)
	{
	eliminar un proceso P
	de S->list;
	wakeup(P);}}

Al implementar el semáforo con una cola de espera, se puede provocar que dos o más procesos esperen indefinidamente a que se produzca un suceso —operación signal()—. Se dice que estos procesos se han **interbloqueado**.

Actividades

6. Investigue por qué las interrupciones no son apropiadas para implementar la sincronización en los sistemas multiprocesador.

Aplicación práctica

Se dan tres procesos con el pseudocódigo que aparece abajo. Indique:

- **Si existe interbloqueo en su ejecución y por qué.**
- **Si existe una salida por pantalla y, en ese caso, cuál.**
- **El valor de los semáforos (hay 4 semáforos).**
- **Inicialmente, los semáforos están a cero:**

Proceso 1	Proceso 2	Proceso 3
imprime "1";	wait(&s1);	wait(&s2);
signal(&s3);	imprime "4";	wait(&s4);
imprime "2";	wait(&s3);	imprime "3";
signal(&s2);	signal(&s4)	imprime "8";
signal(&s1);	wait(&s3);	signal(&s3);

Salida por pantalla 12438 se ejecutan verticalmente
4 semaforos s1,s2,s3,s4 inicialmente a 0
proceso 1 S1, s2, s3 =1 (signal incrementa)
proceso 2 s1=0, s3=0, S4=1 s3=-1
proceso 3 s2=0, S4=0 S3=0

SOLUCIÓN

Comienza la ejecución del proceso 1:

- En pantalla aparece 1
  - s3=1;
  - s2=1;
  - s1=1;
  - En pantalla aparece 12
- Ejecución del proceso 2:
  - s1=0; P2 en cola;
  - En pantalla aparece 124;
  - s3=0; en la cola P2;
  - s4=1;
  - s3=–1 en la cola P2;
- Ejecución del proceso 3;
  - s2=0; en cola el proceso P3;
  - s4=0; en cola el proceso P3;
  - En pantalla aparece 1243;
  - En pantalla aparece 12438;
  - s3=0;

Ejecuciones terminadas.

Los tres procesos terminan, no se da interbloqueo.

La salida por pantalla es la siguiente: 12438.

Al final, todos los semáforos están a cero.

---

4. Aspectos de seguridad sobre el desarrollo de elementos del núcleo

Para garantizar el buen funcionamiento del sistema operativo hay dos conceptos fundamentales: la protección y la seguridad.

Los mecanismos de **protección** van a garantizar que solo los procesos autorizados por el sistema operativo puedan actuar sobre la memoria, la CPU y otros recursos.

Los mecanismos de **seguridad** van a evitar accesos sin autorización, que realicen modificaciones maliciosas o por que se produzca una introducción accidental de incoherencias.

La protección no es solo cuestión del diseñador del sistema operativo, también del programador de aplicaciones, que debe poner los medios adecuados para que su aplicación haga un buen uso de los recursos utilizados.

Se puede utilizar un principio que se usa frecuentemente y que dice que a los programas, usuarios y sistemas se les concedan solo los suficientes privilegios para llevar a cabo sus tareas. Este principio es el del **mínimo privilegio**. Por ejemplo, un sistema operativo que siga este principio implementará sus características, programas, llamadas al sistema y estructura de datos, de manera que si se produce un fallo se provoque el daño mínimo posible.

Un proceso va a actuar en un **dominio de protección** que define un conjunto de objetos y operaciones a los que tiene acceso.

Sabía que...

En sistema operativo *Unix*, un domino está asociado con el usuario, de manera que si se quiere tener acceso a otro dominio hay que cambiar la identidad del usuario.

En el sistema *Multics*, los dominios de protección están organizados jerárquicamente en una estructura de anillos concéntricos. Cada uno es un dominio.

Un mecanismo para proporcionar un control estricto es la **matriz de accesos**. Por ejemplo, una matriz de acceso tiene en sus filas un dominio y en sus columnas los objetos (que pueden ser *hardware* o *software)*. Los derechos tendrán esta forma (dominio, objeto).

Otro concepto a tener en cuenta es el de **roles**, que se definen como una colección de privilegios de acceso.

También se utiliza la protección basada en el lenguaje, que proporciona la posibilidad de hilar más fino. Por ejemplo, una JVM Java puede ejecutar varias hebras y aplicarle una protección diferente a cada una de ellas. Uno de los mecanismos de protección de Java se basa en aplicar a las solicitudes de recursos un mecanismo de inspección de la pila.

Se ha visto que existen medios para proteger el sistema por parte del sistema operativo y por parte de los desarrolladores de elementos del núcleo. Desde el punto de vista de los desarrolladores, lo ideal es el uso de buenas prácticas a la hora de la

codificación para modificar o desarrollar un elemento, ya que, como se ha visto a lo largo de todo el capítulo, el núcleo o *kernel* es la parte esencial del sistema operativo. Una mala modificación o un uso indebido pueden producir el bloqueo del sistema o exponerlo a vulnerabilidades.

Un ejemplo de malas prácticas por parte del desarrollador puede ser un error de programación que lleve al desbordamiento de la pila y del búfer. Este error puede producirse al olvidar comprobar los límites para un determinado caso de entrada, etc.

Escribir de forma aleatoria en una localización física de memoria puede, dependiendo de dónde se haya escrito, causar un mal funcionamiento o hacer que el sistema sea imposible de reiniciar. Por lo tanto, una buena práctica sería no probar directamente sobre la memoria.

Para evitar esto, el núcleo puede crear una interfaz virtual, entre la aplicación y la memoria, separando el espacio del usuario, del código y datos del núcleo, a través de un **direccionamiento virtual**.

Actividades

7. Defina cómo ayuda a la creación de sistemas de protección el principio del mínimo privilegio.
8. Investigue el mecanismo de protección que utiliza Java: mecanismo de inspección de la pila.

4.1. Consideraciones sobre compatibilidad de versiones del núcleo

Para hablar de las consideraciones que hay que tener en cuenta sobre la compatibilidad de versiones del núcleo, se va a tomar como referencia *GNU/Linux*, ya que es uno de los ejemplos más destacados de *software* libre. Además, todo el código fuente puede ser utilizado, modificado y distribuido.

En el sistema operativo del ejemplo, es posible compilar los núcleos o *kernels* modificados. Además, hay mucha documentación para ello. Pero es necesario considerar las consecuencias que puede tener. En este sentido, se va a destacar lo siguiente:

- Para los responsables de los sistemas operativos actuales, las actualizaciones, sobre todo las relacionadas con la vulnerabilidad, son lo prioritario. Gracias a ellas es posible mantener el sistema actualizado y protegido.
- Una distribución *GNU/Linux* es un sistema extraordinariamente complejo, formado por muchas piezas de *software* diferentes, desarrolladas de manera inde-

pendiente por programadores de todo el mundo. Es una labor complicada coordinar todas estas piezas, de manera que, dentro de cada distribución, por cada aplicación o paquete, se encuentran uno o más responsables (mantenedores).

- Los responsables se encargan de mantener seguros y actualizados los correspondientes programas y comprueban su compatibilidad y dependencia dentro del conjunto. El hecho de que se modifique un programa hace que haya que replantearse la estabilidad del conjunto, se renuncia a las actualizaciones y es necesario ser responsable de que esta parte funcione de forma correcta.
- En el caso del núcleo, al tratarse de la parte esencial del sistema operativo, los responsables del núcleo de cada distribución suelen ser programadores de reconocido prestigio dentro de la comunidad y se mantienen en permanente contacto con los desarrolladores del *kernel Linux*.
- Cambiar de *kernel* introduce el riesgo de que el equipo no funcione del todo bien, o incluso que ni siquiera arranque, por lo que se debe estar preparado previamente para este tipo de problemas.

En definitiva, cabe destacar lo delicadas que pueden ser las modificaciones de carácter individual que se hagan.

Actividades

9. Busque información en internet sobre la configuración e instalación de un nuevo núcleo en un sistema operativo basado en *Linux*.

5. Resumen

A lo largo de este capítulo, se ha tratado la importancia del núcleo del sistema operativo, de cómo esta parte esencial gestiona la unidad fundamental de trabajo de un sistema operativo, los procesos. También se han mostrado los requisitos que se necesitan para gestionarlos: el mecanismo de interrupciones, la protección de la memoria para los accesos concurrentes, el conjunto de instrucciones reservadas y el reloj de tiempo real.

Se han establecido también los subsistemas que lo componen.

En cuanto a la gestión de los procesos, se han analizado los estados por los que pasan los procesos, cómo se planifican y el concepto de unidad básica de uso de la CPU, la hebra.

En la gestión de la memoria, cómo cambian las direcciones a lo largo del ciclo de vida de un programa, transformándose de lógicas a físicas. El aprovechamiento de la memoria principal mediante intercambio de programas en memoria, la gestión de la memoria continua, la paginación y segmentación, que además solucionan el problema de la fragmentación externa.

En el sistema de ficheros, para gestionar los recursos de almacenamiento, cómo se estructura y protege dicho sistema.

En cuanto al control de dispositivos, esta tarea tiene una gran complejidad, debido a la cantidad de dispositivos que hay. Mediante este subsistema se tratan indistintamente de su tipo.

En la parte de comunicaciones se ha definido cómo se comunican los procesos, compartiendo datos, ficheros, y cómo hacer para que no interfieran entre ellos. Aquí cabe destacar el concepto de semáforo.

Por último, se ha hecho hincapié todo lo relacionado con la seguridad, protección del sistema, los accesos, roles, etc. Además, se han mostrado unas consideraciones a tener en cuenta a la hora de desarrollar los propios elementos del núcleo, siendo la más importante las buenas prácticas.