**TEMA 5: GESTIÓN DE ENTRADAS/SALIDAS**

1. **HARDWARE DE E/S**

Elementos hardware del sistema de E/S y arquitectura:

* Bus
* Puerto
* Controlador
* Dispositivo

Comunicación con el hardware de E/S:

* Sondeo
* Interrupciones
* DMA
* Dispositivos proyectados en memoria

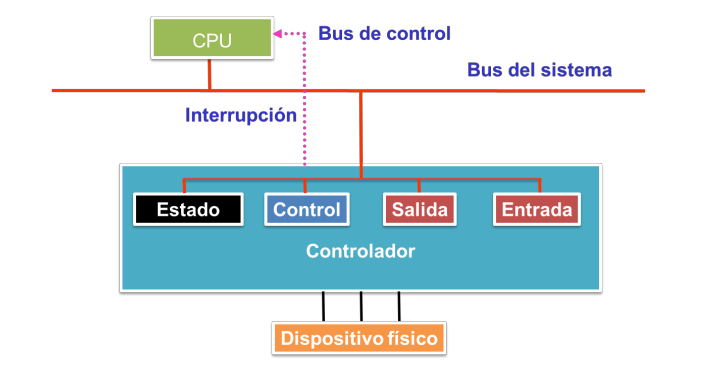
**ARQUITECTURA HARDWARE**

La interfaz hardware asociada con un dispositivo de E/S consta de cuatro elementos básicos:

* Un bus para comunicarse con la CPU y que es compartido entre varios dispositivos.
* Un puerto que consta de varios registros:
* Estado →  indica si está ocupado, los datos están listos, o ha ocurrido un error.
* Control →  operación que ha de realizar.
* Datos\_entrada →  datos a enviar a CPU.
* Datos\_salida →  datos recibidos de la CPU.

Un controlador (driver) es un software que controla un dispositivo. Recibe órdenes del bus del sistema, traduce órdenes en acciones del dispositivo, y lee/escribe datos desde/en el bus del sistema.  Los controladores actúan como “puentes” entre las aplicaciones y los dispositivos, encargándose de que ambos interactúen.

**ESQUEMA**



Tipos de gestión de la E/S:

**i) COMUNICACIÓN: SONDEO (E/S PROGRAMADA)**

Los pasos a seguir en el sondeo:

1. CPU espera hasta que el estado sea libre.
2. La CPU ajusta el registro de órdenes y datos de entrada o salida.
3. La CPU ajusta el estado a orden-preparada.
4. El controlador reacciona a orden-preparada y pone estado a ocupado. Lee registro de órdenes y ejecuta orden, pone un valor en registros de datos si es necesario. Pone estado a listo.
5. Suponiendo que la orden tiene éxito, el controlador cambia el estado a ocioso.
6. La CPU observa el cambio a ocioso y lee los datos si es una operación de salida.

**ii) COMUNICACIÓN: INTERRUPCIONES (E/S DIRIGIDA POR INTERRUPCIONES)**

El sondeo es una buena elección si los datos van a ser manejados al instante (un modem o teclado), ya que los datos se perderían si no se retiran del dispositivo lo suficientemente rápido, pero ¿qué ocurre si el dispositivo es lento comparado con la CPU? En lugar de tener la CPU ocupada esperando la disponibilidad del dispositivo, el dispositivo interrumpe a la CPU cuando ha terminado una operación de E/S. Cuando se produce la interrupción de E/S:

1. Hay que determinar qué dispositivo la provocó para poder asignar la RSI (rutina de servicio de interrupción) correspondiente.
2. Se determina el estado de finalización. Si la última orden fue una operación de entrada, hay además que recuperar los datos del registro del dispositivo.
3. Inicia la siguiente operación para el dispositivo.

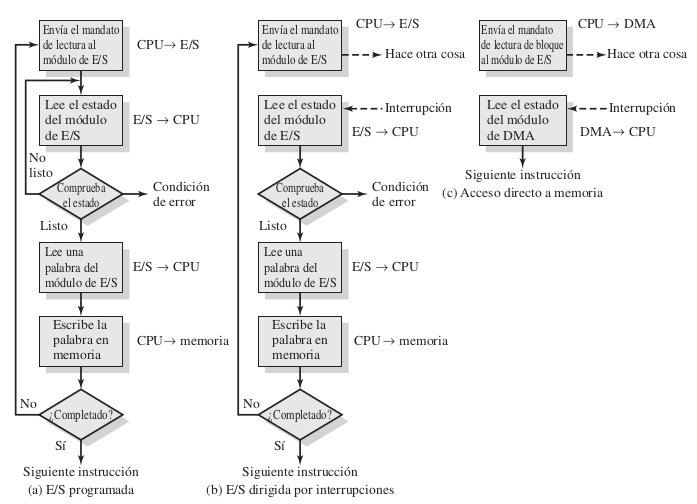
**iii) COMUNICACIÓN: DMA (ACCESO DIRECTO A MEMORIA)**

Recuperar información byte a byte no adecuado para dispositivos que transfieren grandes cantidades de datos.

**DMA** es el controlador de dispositivo que puede leer/escribir directamente en memoria. En lugar de registros de E/S, tiene un registro de dirección:

La CPU indica al DMA la ubicación de la fuente/destino de la transferencia.

1. DMA opera el bus e interrumpe a la CPU (robo de ciclo) cuando se completa la transferencia.
2. DMA y CPU compiten por el bus de memoria.



1. **ARQUITECTURA**

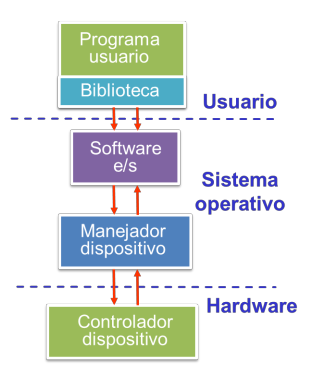
**SERVICIOS DE E/S**

Mecanismo para nombrar archivos, da control de acceso. Se utiliza el buffer para hacer referencia a los metadatos del archivo. En todos los dispositivos se separan las dependencias del sistema, se aísla en el manejador de dispositivo y este sabe tratarlo según el tipo de dispositivo que sea. No hace falta cambiar el software de E/S.

* Designación de archivos y dispositivos.
* Control de acceso (protección).
* Operaciones adecuadas para archivos y para dispositivos.
* Asignación de dispositivos.
* Buffering, caché, y spooling, para suministrar una comunicación eficiente con el dispositivo.
* Planificación de E/S.
* Gestión de errores y recuperación de fallos asociados con el dispositivo.
* Aislar en un módulo las características y conducta específica del dispositivo.

Podemos estructurar el software de E/S en capas:

* Manejadores de dispositivos
* Software de E/S independiente del dispositivo
* Software a nivel de usuario.



1. **ARCHIVOS DE DISPOSITIVOS**

**SOFTWARE INDEPENDIENTE DEL DISPOSITIVO**

Realiza tareas comunes a todos los dispositivos y suministra una interfaz común al usuario.

Sus principales funciones son:

* Suministra interfaz uniforme a los manejadores.
* Realiza la designación de dispositivos.
* Implementa la protección de dispositivos.
* Establece el tamaño de bloque independiente del dispositivo y asigna almacenamiento (para dispositivos de bloque).
* Informar de errores producidos en las E/S.

Cuando se desea hacer una lectura, en todas las operaciones se pone el tamaño a leer deseado, que es independiente del tamaño del archivo, el cual se establece cuando se formatea  el sistema de archivo.

**stat**, además de para obtener los metadatos de un archivo, da un consejo del tamaño de hardware a utilizar.

**DESIGNACIÓN**

El espacio de nombre de dispositivos define cómo identificar y nombrar los dispositivos.

Existen diferentes espacios de nombre:

* Espacio de nombres hardware →  especifica el dispositivo por el controlador al que está ligado y el número de dispositivo lógico dentro del controlador.
* Espacio de nombres kernel →  utilizado por el núcleo, suele basarse en el anterior.
* Espacio de nombres de usuario →  debe ser un esquema sencillo y familiar.

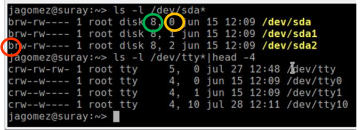
El SO debe llevar esa correspondencia de nombres.

**DESIGNACIÓN: LINUX**

El sistema de E/S independiente del dispositivo, define las semánticas de los espacios de nombres kernel y usuario, y establece las correspondencias entre ellas.

En Unix, el espacio de nombres kernel identifica un dispositivo por:

* Número principal (mayor) →  identifica el controlador O
* Número secundario (minor) →  instancia del dispositivo O
* Tipo de dispositivo →  de caracteres o bloques  O



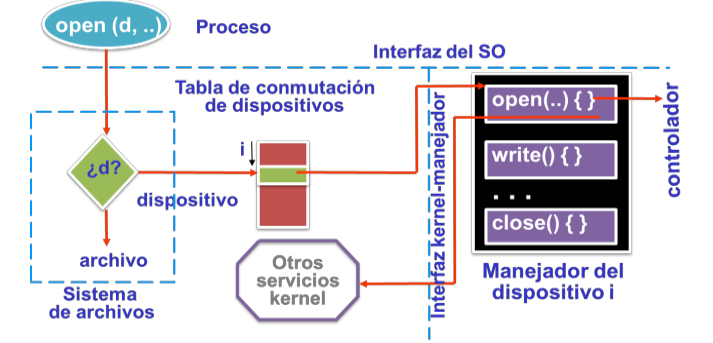
**ESPACIO DE NOMBRES DE USUARIO**

La forma de designación más extendida es la integración del espacio de nombres de dispositivos en el de archivos.

El concepto central es el archivo de dispositivo, el cual permite manejar de la misma forma archivos y dispositivos (mismo conjunto de llamadas al sistema), y aplicarles los mismos mecanismos de protección.

En UNIX, un archivo de dispositivo es un archivo especial que no contiene datos (el campo tamaño ha sido sustituido por los números principales y secundarios), es una interfaz del controlador (Ejemplo: leer del archivo dispositivo terminal se traduce en una llamada a leer directamente del terminal).

**DESIGNACIÓN E INDEPENDENCIA DEL DISPOSITIVO**



**BUFFERING Y CACHÉ DE BÚFERES**

Los dispositivos suelen tener una pequeña memoria en la tarjeta para almacenar datos temporalmente antes de transferirlos a/desde la CPU. ¿Por qué tener búferes en el SO? Para:

* Acoplar la diferencia de velocidades entre la CPU y el dispositivo.
* Hacer frente a la diferencia de tamaños de transferencia de datos entre dispositivos.
* Minimizar el tiempo que el proceso de usuario está bloqueado en una escritura.

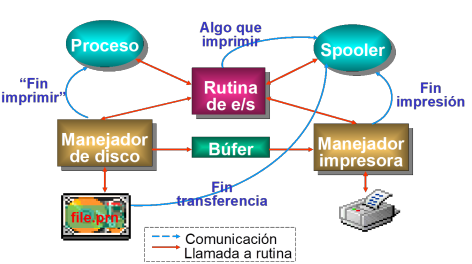
Mantener los búferes en una caché mejora el rendimiento del disco reduciendo el número de acceso al disco. Mantener bloques de datos recientemente usados en memoria después de que se complete la llamada de E/S que los cargó mejora el rendimiento del disco reduciendo el número de acceso al disco.

**E/S EN ESPACIO DE USUARIO**

Las bibliotecas estándar de E/S permiten realizar las llamadas al sistema de E/S para:

* Gestión de formatos (Ejemplo: printf)
* Control de los dispositivos (Ejemplo: iocntl)

El spooling es una técnica para manejar dispositivos dedicados en sistemas multiprogramados (Ejemplo: no asignamos impresora sino que generamos la impresión en un archivo. Un proceso especial manda los archivos a impresión. Igual que el correo electrónico).



1. **MANEJADORES**

**MANEJADOR (DEVICE DRIVER)**

**Manejador del dispositivo**: contiene todo el código dependiente del dispositivo. Cada manejador gestiona un tipo o clase de dispositivo y acepta peticiones “abstractas” de la capa de software E/S y controla que la petición se realiza:

* Traduce petición abstracta en órdenes para el controlador del dispositivo.
* Se bloquea o no, según tipo de operación.
* Si no hay errores, da respuesta si es necesario, y retorna al llamador.

**VISIÓN DEL PROGRAMADOR**

El SO suministra una interfaz de los dispositivos que simplifica el trabajo del programador:

* Interfaz estándar para diferentes dispositivos relacionados.
* El manejador del dispositivo encapsula las dependencias del dispositivo.
* El SO puede soportar nuevos dispositivos simplemente con suministrar el manejador del dispositivo.

**DISPOSITIVO**

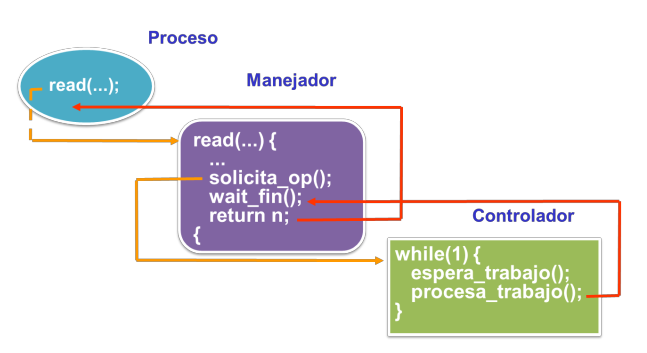
Características de los dispositivos:

* Unidad de transferencia: carácter/bloque
* Método de acceso: secuencial/aleatorio
* Temporización: síncrona/asíncrona. Observar que la mayoría de los dispositivos son asíncronos, mientras que las llamadas al sistema de E/S son síncronas. El SO implementa E/S bloqueantes.
* Compartido o dedicado.
* Velocidad.
* Operación: entrada, salida, o ambas.

**RECAPITULACIÓN**

En lo visto hasta el momento, cuando el usuario solicita una lectura de dispositivo, los pasos eran:

1. Si los datos están en un búfer, salta a paso 4. Si no, paso 2.
2. El SO instruye al dispositivo y “espera”.
3. Cuando el dispositivo tiene los datos interrumpe a la CPU, que los transfiere a un búfer del SO.
4. El SO transfiere los datos al proceso de usuario y lo desbloquea.
5. Cuando el proceso vuelve a ejecutarse, sigue la ejecución después de la llamada al sistema de E/S.

**ESQUEMA**

**PROBLEMA**

El funcionamiento descrito muestra cómo el manejador obedece, por una parte, órdenes del resto de módulos del SO y, de otra, del controlador (driver) del dispositivo. Es el problema que denominamos “un esclavo con dos amos”.

Siendo esto así ¿cómo realiza la espera, wait\_fin(), el manejador en el paradigma síncrono de programación?

El paradigma síncrono de programación, que es el más usado, supone que una vez que la operación de E/S se ha realizado, los datos están listos para usarse. Esto supone que debemos bloquear al proceso en su contexto mientras la operación de E/S finaliza.

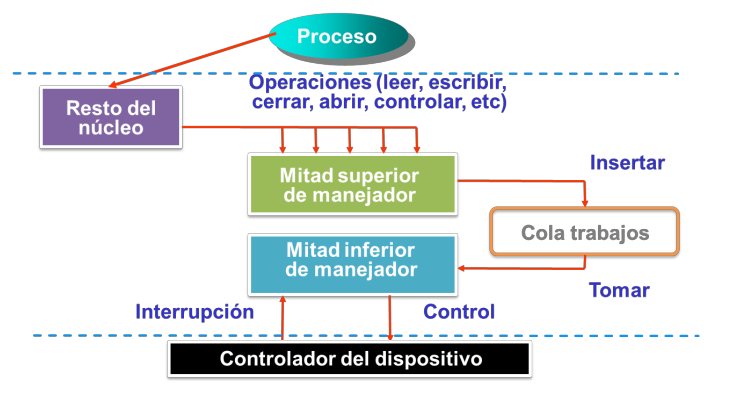
Por otro lado, la invocación de las rutinas de servicio de interrupción es asíncrona al proceso que espera y, por tanto, no debe utilizar el contexto del proceso para su ejecución.

**SOLUCIÓN**

El manejador se divide en dos mitades:

* La mitad superior contiene las rutinas síncronas, se ejecuta en el contexto del proceso y puede acceder a su espacio de direcciones. Puede bloquear al proceso.
* La mitad inferior contiene las rutinas asíncronas, no accede al contexto de proceso y normalmente no tiene relación con el proceso. No puede bloquearse, pues bloquearía a un proceso no relacionado con la interrupción.

**MANEJADORES DE 2 NIVELES**



**RENDIMIENTO**

Las E/S son costosas por varias razones:

* En disco duros electromecánicos (HDD) involucran movimientos físicos lentos(cabezal disco) o retardo por líneas de comunicaciones (teléfono-red).
* Los dispositivos de E/S son a menudo disputados por múltiples procesos.
* Las operaciones de E/S se suministran por medio de llamadas al sistema y gestión de interrupciones, que son lentas.

**PLANIFICACIÓN DE DISCO**

Planificación de disco: las peticiones para leer/escribir bloques de discos se pueden reorganizar en la cola de trabajos para hacer más eficiente el acceso al dispositivo (reducir la latencia de acceso al mismo).

* Reducir el número de veces que los datos son copiados manteniéndolos en caché.
* Reducir la frecuencia de interrupciones utilizando, si es posible, grandes transferencias de datos.
* Descargar computación de la CPU principal utilizando controladores DMA.
* Aumentar el número de dispositivos para reducir la contención de uno único, y así, mejorar el uso de CPU.
* Incrementar memoria física para reducir la cantidad de tiempo en paginación y por ello mejorar el uso de CPU.

Si interesa leer el disco duro como un flujo contiguo de bytes, cuando se hace una copia de seguridad de un disco, se lee entero. Se utilizan las mismas llamadas al sistema para manipular archivos y dispositivos.

Cuando se hace un open y se le da nombre, lo primero es ver si corresponde con un archivo o un dispositivo, conmuta a una tabla de dispositivos donde están todos numerados con su número principal y secundario.

Si se escribe sobre un dispositivo /dev, no tiene sentido ir al /dev de ese dispositivo, porque ese contenido va a terminal, por eso no me hace falta el tamaño.

El manejador va a traducir la petición al controlador del dispositivo.

¿Se bloquea? Si leo n caracteres, no necesariamente se han escrito, supongo que ha terminado, y sigo con la siguiente instrucción. En general el paradigma es bloqueante.

Esta interfaz es bloqueante y en la interfaz no bloqueante se suministra en SO porque hay un problema de concurrencia.

Supongamos que quiero leer de un descriptor n caracteres

n = read (fd, n) y m = read(fd, -..., m) . ¿Qué problema tiene este paradigma, si primero llega la información por el segundo read y por el primero no llega?

Genera una operación de sondeo sobre los descriptores que se van a pasar y el primero que responda, devuelve el descriptor con select. Después hay que seguir programando.

El proceso realiza una operación de E/S, el manejador inferior funciona de forma asíncrona y saca elementos de la cola. Después se reorganiza la cola.