Entrada/Salida

Sistemas Operativos (SO)
Departament d'Arquitectura de Computadors
Facultat d'Informàtica de Barcelona
Universitat Politècnica de Catalunya

Licencia Creative Commons

Licencia Creative Commons

Índice

Índice

- Conceptos y objetivos
- Uso
- Implementación
  - Estructuras Básicas
  - E/S Síncrona
  - E/S Asíncrona
  - Ejemplos
    - Linux
    - Windows

E/S: ¿Qué es?

- ¿Verdadero o falso?
  - Entrada / Salida
    - Verdadero
  - Input / Output
    - Verdadero
  - I/O
    - Verdadero .... :-)
  - Recepción/envio de información desde/hacia periféricos
    - WARNING: Es más que eso!

E/S: ¿Qué es?

- Se entiende por E/S la transferencia de información hacia/desde un proceso
  - En general, un proceso sólo puede leer/escribir en la memoria que se le ha asignado (su espacio lógico de direcciones)
    - ¿Cómo podemos comunicarnos con un proceso?
  - En general, los procesos tienen espacios de direcciones disjuntos
    - ¿Cómo podemos comunicar procesos entre sí?

Es necesario que un proceso pueda comunicarse con su exterior

E/S: ¿Qué es?

- Entrada de datos al proceso desde
  - Teclado, fichero, red, salida de otro proceso, ...
- Salida de datos del proceso hacia
  - Pantalla, fichero, red, entrada de otro proceso, ...

E/S: Observaciones

- Existen muchos tipos de dispositivos
  - difíciles de programar
  - con características muy variadas

Características de los dispositivos

- Velocidad de transferencia
- Unidad de transferencia
  - bloque o carácter
- Representación de los datos
- Operaciones permitidas
- Modos de trabajo
  - echo/noecho, spool/nospool
- Tipos de error

Función del SO

- Función del S.O.: gestionar los dispositivos de E/S
  - Cada dispositivo require un tratamiento especial
    - write_printer
    - write_file
    - write_screen
  - El S.O. oculta las diferencias y proporciona una interficie común independiente del dispositivo
    - write
    - Operaciones sobre *dispositivos virtuales*

E/S: Observaciones

- Puede interesar ejecutar un mismo programa usando diferentes dispositivos en cada ejecucion

  prompt_$ ls

  Normalmente saca la salida del comando por pantalla

  prompt_$ ls > fichero

  Deja la salida del comando ls en un fichero

- No siempre se usa un periférico

  prompt_$ ls | more

  La | (pipe) es un dispositivo lógico (no físico) creado por el SO que sólo existe en memoria

E/S: Conceptos

- Canales de Entrada/Salida/Error estándar
- Redirección
- Código independiente del dispositivo
  - Usa dispositivos virtuales
    Tambien llamados canales virtuales (o sencillamente canales)
- El SO asocia un dispositivo virtual a un dispositivo ...
  - Lógico o Físico

Tipos de dispositivos

- Físicos
  - Dispositivos hardware (disco, teclado, ...)
- Lógicos
  - Dispositivos que:
    - no tienen dispositivo hardware asociado (Nul , Random , ...)
    - añaden funcionalidades sobre el dispositivo hardware (ficheros)
    - agrupan varios dispositivos hardware (Consola)
- Virtuales
  - Canal: Abstracción del Sistema con la que trabajan los procesos

Objetivos de la E/S

- Permitir comunicación de un proceso con el exterior
- Facilitar el uso de dispositivos con características muy diferentes
  - teclado, pantalla, ratón, disco, CD-ROM, ..., ficheros, ... nulo
- Permitir que un mismo código pueda ser usado con diferentes dispositivos
  - Código de usuario independiente del dispositivo
  - Redirección de dispositivos de E/S

Principios de diseño

- Uniformidad en las operaciones
  - Mismas *Llamadas a SO* para todos los dispositivos
    - Misma semántica sobre todos los dispositivos
  - El sistema comprueba si la operación es válida sobre el dispositivo
  - Aumenta la portabilidad y simplicidad de los procesos de usuario

Principios de diseño (II)

- Dispositivos virtuales
  - El proceso trabaja sobre dispositivos virtuales
  - El sistema proporciona mecanismos para asociar/desasociar un dispositivo virtual a uno físico/lógico
    - open,close

Principios de diseño (III)

- Redireccionamiento
  - El S.O. permite cambiar la asignación de los dispositivos virtuales antes de iniciar un proceso
  - Permite que el mismo programa se ejecute sobre diferentes dispositivos sin cambiar el código
    - UNIX: ls > llistat.dat
    - VMS: assign sys$output llista.dat; dir

Independencia del dispositivo

- Incrementa la portabilidad de los programas
- Independencia del dispositivo
  - Dispositivos virtuales
- Independencia de la unidad de transferencia
  - Uniformidad de las operaciones
- Incrementa la reusabilidad de los programas
  - Redireccionamiento

Índice

- Conceptos y objetivos
- Uso
- Implementación
  - Estructuras Básicas
  - E/S Síncrona
  - E/S Asíncrona
  - Ejemplos
    - Linux
    - Windows

E/S: Uso

- El SO proporciona llamadas a sistema para la realización de operaciones de E/S
- Vamos a enumerar algunas llamadas del SO UNIX.
- En el laboratorio se iran viendo las llamadas más habituales con detalle y experimentaremos con ellas.

E/S: Uso

- El SO proporciona llamadas a sistema para la asociación de canales virtuales a dispositivos de E/S
  - open, creat, pipe, socket
- Una vez se dispone de un canal, se opera con operaciones genéricas de lectura y escritura sobre él
  - read, write
- Se puede finalizar la asociación entre el canal virtual y el dispositivo (físico y/o lógico)
  - close

E/S: Uso

- Se pueden duplicar canales de E/S
  - dup, dup2
- Se pueden gestionar características de los dispositivos
  - ioctl, fcntl
- Se cambia el puntero de lectura/escritura
  - lseek
    (no disponible para ciertos dispositivos)

Índice

- Conceptos y objetivos
- Uso
- Implementación
  - Estructuras Básicas
  - E/S Síncrona
  - E/S Asíncrona
  - Ejemplos
    - Linux
    - Windows

Tabla de canales

- Guarda las relaciones entre los dispositivos virtuales y los dispositivos físicos/lógicos
  - Indexada por nº de canal o nombre del dispositivo virtual
  - Una por proceso
    - es compartida por todos los flujos (threads) del proceso
    - puede ser compartida entre *procesos con recursos compartidos* (ej. clones)
  - S.O. puede asignar los canales (primero libre) o puede dejar que los decida el usuario

Niveles de E/S

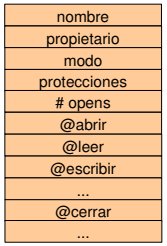
- El código del SO que implementa la E/S tiene dos niveles:
  - Independiente
    - Funciones de E/S comunes a todos los dispositivos
  - Dependiente
    - Funciones de E/S especifícas de cada dispositivo
    - Añadir un nuevo dispositivo significa modificar este nivel

Descriptor de dispositivo

- Estructura que contiene la información relacionada con un dispositivo
  - Punteros a las funciones dependientes específicas del dispositivo
  - Añadir un dispositivo es añadir un descriptor y sus funciones relacionadas
  - Tiene la información necesaria para gestionar el dispositivo:
    - Ejemplo
      - nombre, apuntadores a las rutinas específicas, Nº de opens, características, offset

Descriptor de dispositivo

- Ejemplo de Descriptor de Dispositivo (DD)



Device Driver

- Software que se comunica directamente con el hardware (device controller) a traves de los registros del dispositivo
- La comunicación entre el driver y el controlador puede ser de dos maneras:
  - por encuesta (polling)
  - por interrupciones

Los conceptos básicos de soporte hardware y funcionamiento de encuesta e interrupción se han visto en IC, EC1 y EC2

Encuesta

- La CPU esta constantemente consultando el dispositivo para ver si la operación ya se ha realizado

```
preparar E/S
while ( consultar_dispositivo != FINALIZADO );
finalizar E/S
```

  - Sencillo
  - Muy poco eficiente desde el punto de vista del sistema
  - Solo se ha de usar cuando no se pueda hacer de otra manera

Interrupciones

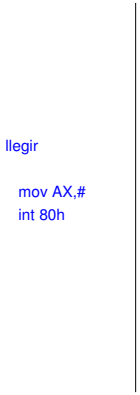
- La CPU programa la E/S y recibe una interrupción cuando esta ha finalizado
  - El proceso se bloquea y cede la CPU hasta que recibe la interrupción
    - mejor uso de la CPU en el sistema
  - Hay que minimizar siempre el trabajo que se hace en la rutina que atiende a la interrupción

Tipos de Entrada/Salida

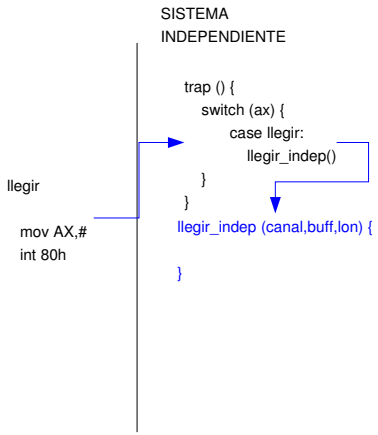
- Síncrona
  - El proceso de usuario se queda bloqueado hasta que finaliza la operación de E/S
- Asíncrona
  - El proceso de usuario se ejecuta concurrentemente mientras se realiza la E/S
    - El Sistema Operativo notifica al proceso cuando esta finaliza

      y/o
    - El proceso dispone de una interficie para consultar el estado de las operaciones pendientes
      - esperar, cancelar, estado_es
  - Programación más compleja

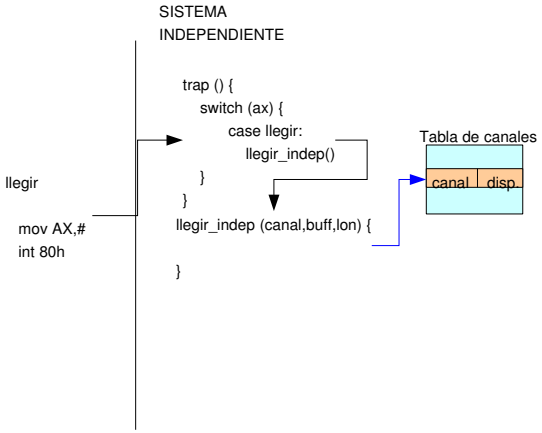
E/S Síncrona



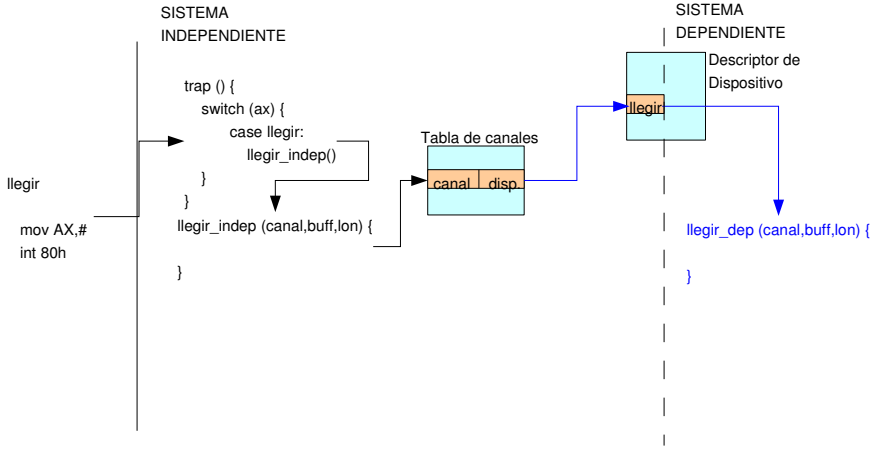
E/S Síncrona



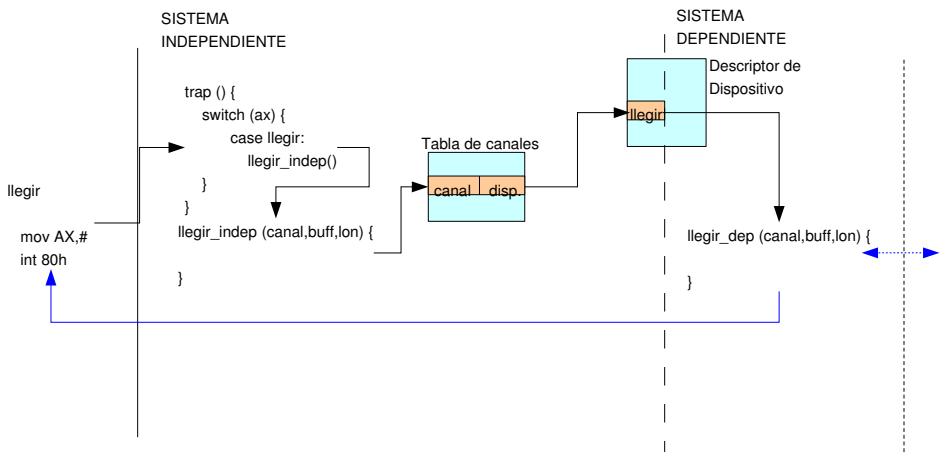
E/S Síncrona



E/S Síncrona



E/S Síncrona



Mejoras de la E/S

- Buffering
  - El dispositivo dispone de un buffer donde guarda los datos a enviados/recibidos
    - El buffer se va llenando/vaciando mientras los procesos trabajan
    - Permite evitar bloqueos
      - evitando picos de E/S
    - Permite evitar la perdida de información
  - Doble buffering
    - Permite que se produzca a la vez movimiento de datos entre usuario – sistema y sistema - dispositivo
  - Buffering circular
    - Generalización del doble buffering

Mejoras de la E/S (II)

- Caching
  - El dispositivo dispone de una cache donde guarda los resultados de operaciones anteriores
    - Si una operación esta en la en cache se sirve el resultado de está directamente
    - Aumenta la eficiencia de la E/S si la política es buena
  - Ejemplo
    - Memoria usada como cache de disco

Mejoras de la E/S (II)

- Spooling
  - La E/S se realiza sobre un dispositivo intermedio
    - El sistema posteriormente la realizará sobre el dispositivo final
    - Permite compartir dispositivos *no compartibles*
    - El dispositivo intermedio suele ser más rápido
  - Ejemplo
    - Impresora: dispositivo no compartible
      - Mientras se esta imprimiendo un documento no se puede imprimir otro
    - Disco: dispositivo compartible
      - Se pueden ir alternando accesos a diferentes ficheros para diferentes procesos
    - Se pueden guardar peticiones de impresion en ficheros temporales. Se usa una cola para gestionar las peticiones. Se imprimen de uno en uno.

Mejoras de la E/S (III)

- Algoritmos eficientes de acceso
  - Reordenar peticiones para mejorar la eficiencia en el acceso
  - Ejemplo: politicas de planificación de acceso a disco
    - Según quien hace la petición
      FIFO, LIFO, random, prioridades
    - Según el contenido de la petición
      SSTF, SCAN, C-SCAN, N-step-SCAN, FSCAN
- Organización y uso del hardware
  - Ejemplo: RAID
    - Distribución de un fichero en diversos discos: acceso en paralelo
    - Replicación: aumento de la tolerancia a fallos

Gestores

- Proceso de sistema encargado de atender y resolver peticiones de E/S
  - Puede haber 1 o más gestores por dispositivo
  - pseudo-código

```
for ( ; ; ) {
        esperar petición
        recoger parámetros
        realizar E/S
        notificar finalización E/S
}
```


Gestores (I)

- Simplifican el acceso a las estructuras de datos
  - Reducen la necesidad de usar exclusiones mutuas
- Permiten planificar las peticiones
- Facilitan la implementación de E/S asíncrona

Gestores (II)

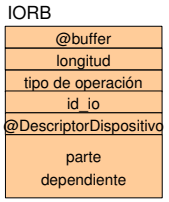
- Sincronización proceso de usuario/gestor
  - Mediante semáforos (operaciones wait / signal)
    Por el momento las podemos entender como
    wait: esperar_aviso
    signal: enviar_aviso
  - Notificación de una nueva petición de E/S
    - El gestor espera a recibir notificaciones (hace un wait sobre un semáforo)
    - La rutina de E/S avisa al gestor (hace un signal sobre el semáforo del gestor)
  - Notificación de finalización de E/S
    - La rutina de E/S espera mediante un wait sobre un semáforo
      - Cada operación de E/S tiene un semáforo própio
    - El gestor avisa de la finalización de la E/S (hace un signal sobre el semáforo)

Gestores (III)

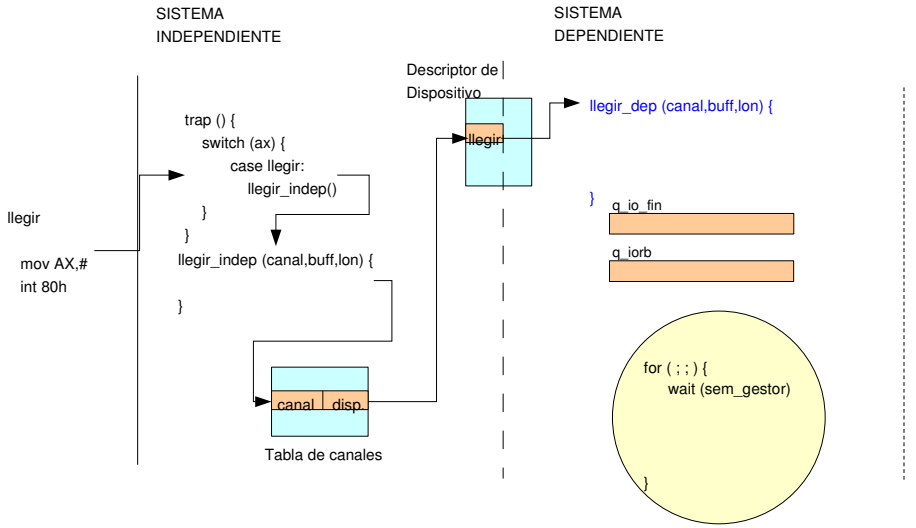
- Paso de parámetros
  - Mediante la estructura IORB (Input/Output Request Block)
    - Las rutinas de E/S rellenan y encolan los IORBs
    - Cada gestor/dispositivo tiene una cola de IORBs con las peticiones pendientes
    - El contenido de los IORBs varía según el dispositivo
- Retorno de resultado
  - Mediante la estructura io_fin
    - Contiene el identificador de E/S y su resultado
    - Una cola de resultados por dispositivo
    - El gestor encola el io_fin y la rutina de E/S lo recoge

Implementación: estructuras

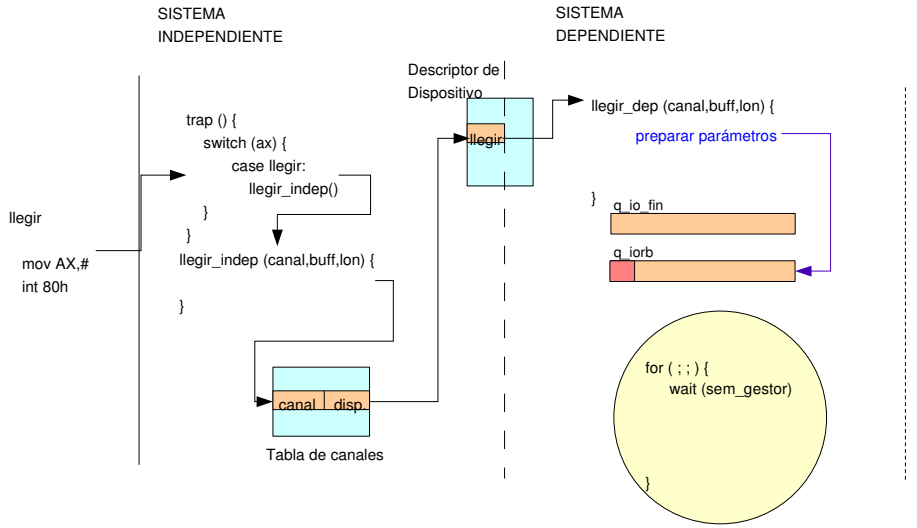
- IORB
  - Buffer:
    - buffer de usuario donde estan o donde se dejan los datos
  - Tipo de operación:
    - Lectura o escritura?
  - id_io:
    - Identificador de la E/S que representa el IORB



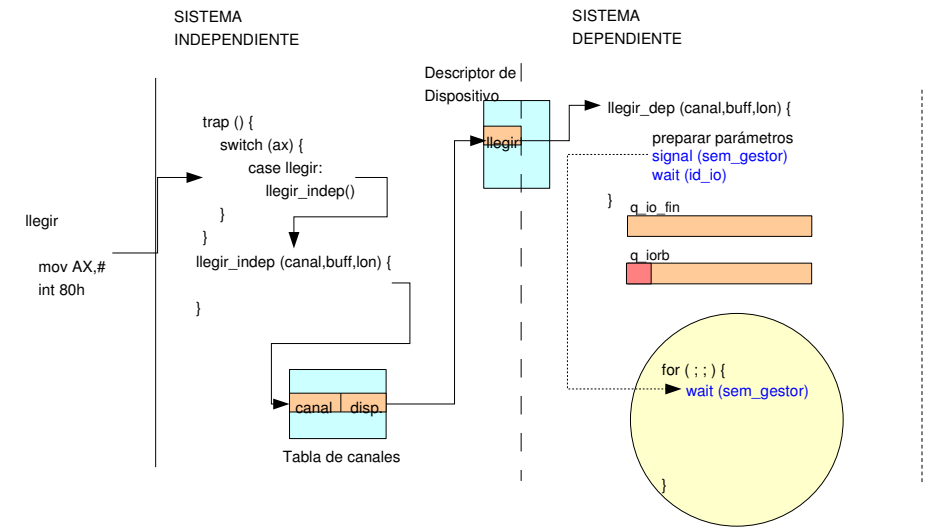
E/S Síncrona con gestor



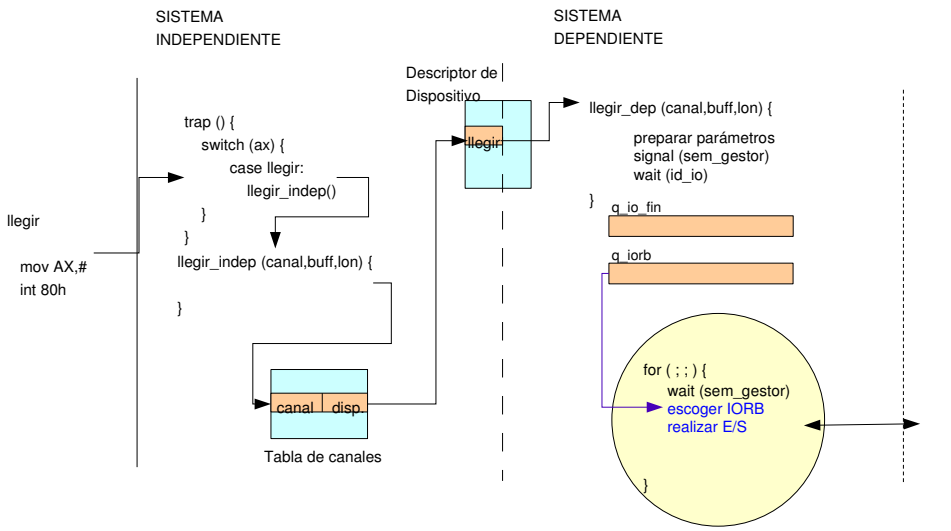
E/S Síncrona con gestor



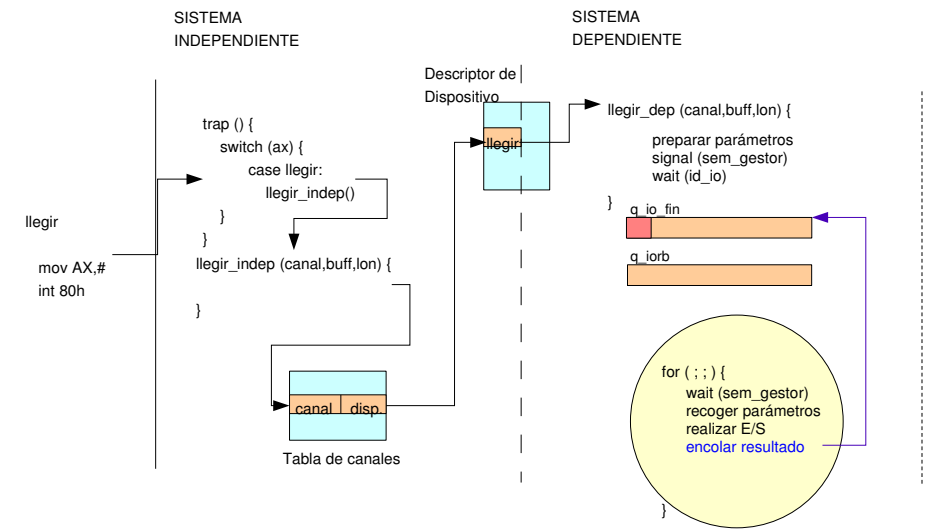
E/S Síncrona con gestor



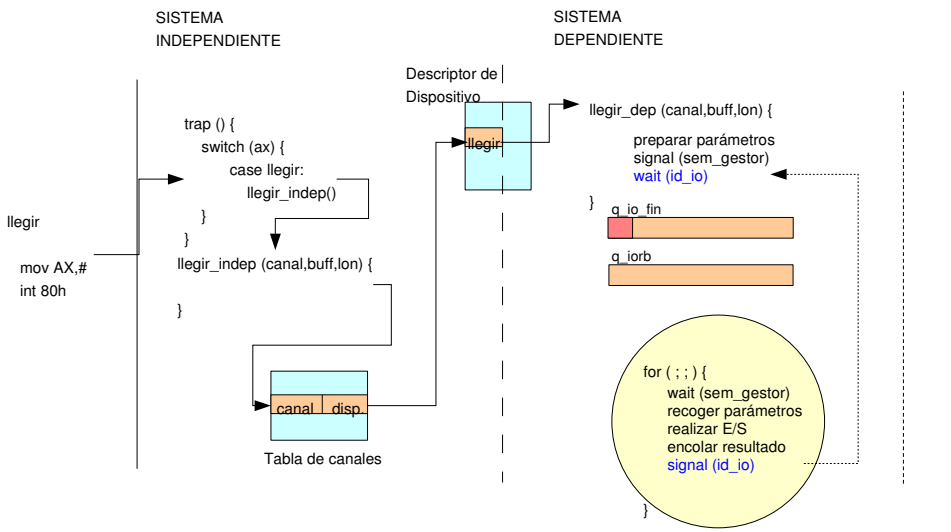
E/S Síncrona con gestor



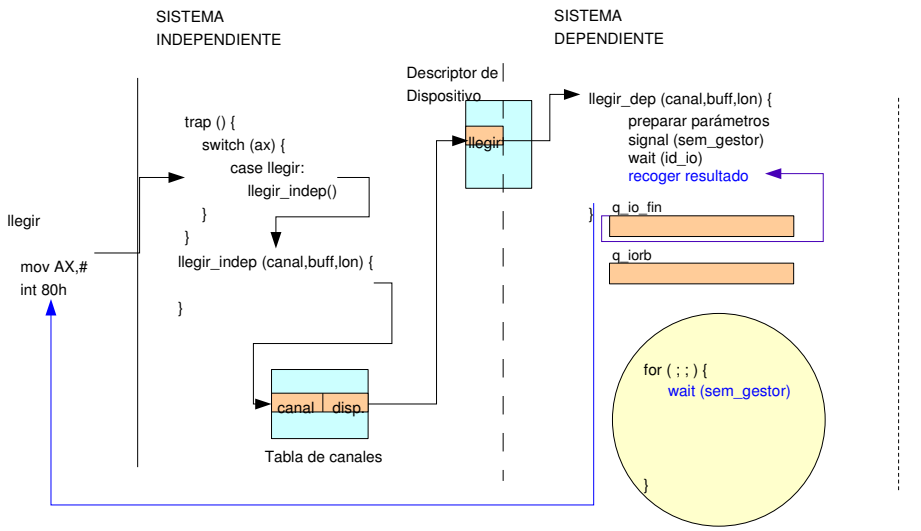
E/S Síncrona con gestor



E/S Síncrona con gestor



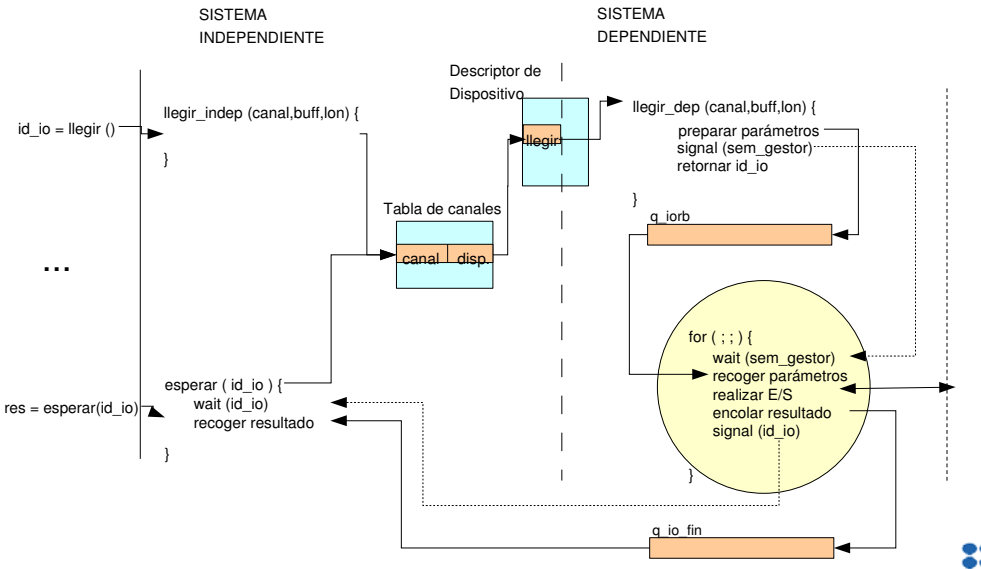
E/S Síncrona con gestor



Gestores (IV)

- Facilitan la implementación de E/S asíncrona
  - Separar petición de entrada salida de la espera del fin de la misma

E/S Asíncrona



E/S Asíncrona

- En un sistema que sólo proporciona E/S síncrona un proceso de usuario puede conseguir E/S asincrona si usa diversos flujos:
  - Un flujo (o varios) pueden realizar la(s) E/S (posiblemente bloqueándose)
  - Otro(s) flujo(s) pueden realizar los cálculos (sincronizándose cuando sea necesario con los flujos que realizan la E/S)

Ejemplos: UNIX/Linux

- Dispositivos accesibles a través del Sistema de ficheros
  - Ficheros especiales (normalmente situados en /dev)
    - /dev/hda1
    - /dev/audio0
    - /dev/nul
  - Se utilizan con las primitivas normales (open,read,write,...)
- Se crean mediante *mknod*
  - Asigna dos numeros especiales al fichero: *major* y *minor*
  - Asigna el tipo de entrada/salida: por bloques o por carácteres

Ejemplos: UNIX/Linux

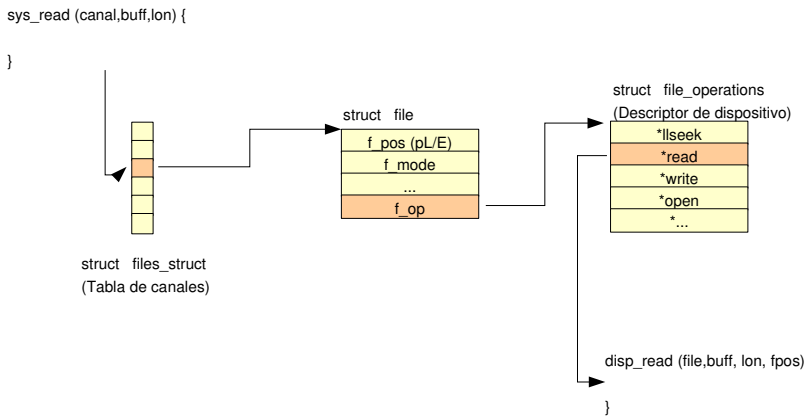
- El *major* establece la relación entre el fichero y el *driver* de dispositivo a utilizar
  - Los números específicos dependen de cada SO específico.
  - P.ej. en linux:
    - 2 -> pseudo terminales
    - 3 -> primer disco ide
    - 6 -> impresora
- El *minor* permite al driver distinguir entre diferentes dispositivos del mismo tipo
  - /dev/hda1, /dev/hda2, /dev/hda3, ...

Ejemplos: Linux

- Los *drivers* son ficheros objetos que se enlazan de forma dinámica con el kernel (módulos)
  - Sólo aquellos drivers que se vayan a usar están realmente en memoria
  - insmod, modprobe
- Al cargarse el modulo se ejecuta una función que registra las funciones del driver
  - Hay diferentes funciones de registro según el tipo de driver
    - int devfs_register_chrdev (unsigned int major, const char *name, struct file_operations *fops);
    - int devfs_register_blkdev (unsigned int major, const char *name, struct block_device_operations *bdops);
    - int register_netdevice(struct net_device *dev);
    - int register_filesystem(struct file_system_type *);
    - ...

Ejemplos: Linux

- Estructuras



Ejemplos: Windows

- HANDLE CreateFile(name, access, sharemode, security, creation, attributes, NULL)
- Función utilizada por el sistema operativo
  - No es independiente del tipo de fichero
  - El usuario ha de saber que tipo de fichero abrirá

Ejemplos: Windows

- Ejemplo:
  - Fichero normal abierto para leer:
    - CreateFile("\\prueba.txt", FILE_READ_DATA, FILE_SHARE_READ, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
  - Es equivalente a:
    - open("prueba.txt", O_RDONLY);

Ejemplos: Windows

- Ejemplo:
  - Abrir un dispositivo por su nombre lógico:
    - Createfile (("\\\\.\\PhysicalDrive0", 0, FILE_SHARE_READ | FILE_SHARE_WRITE, NULL, OPEN_EXISTING, 0, NULL);
    - Retorna un identificador con el cual se puede escribir físicamente en el disco duro