

Correo: alecu@ugr.es
Tutorías: Departamento 19 3^a planta

Notes

open

read(fd, buffer, size)

char buffer[1024]

1. Sistema Informático

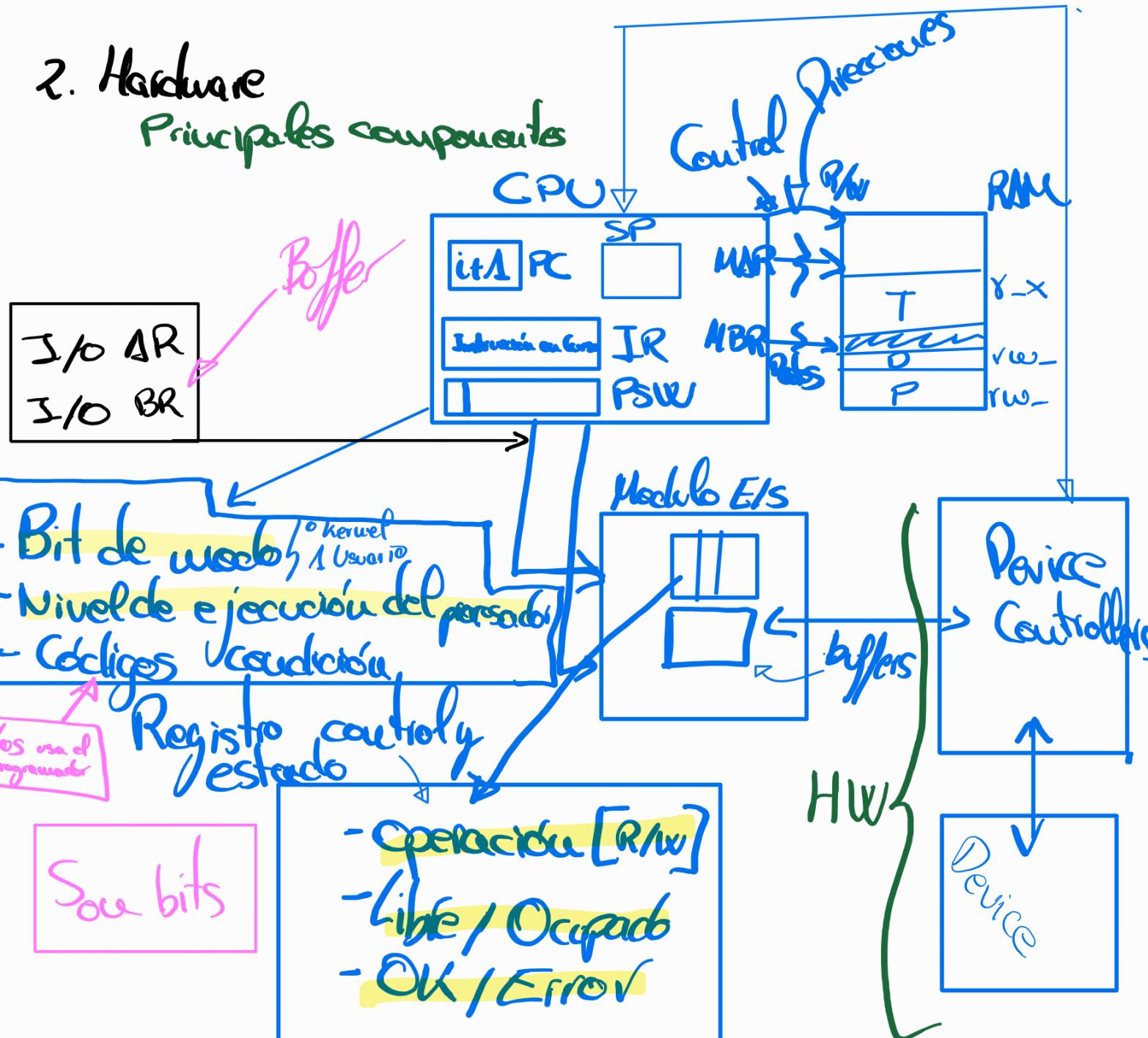
Device Controller → Hardware

Device Driver → Software

id-disp

2. Hardware

Principales componentes

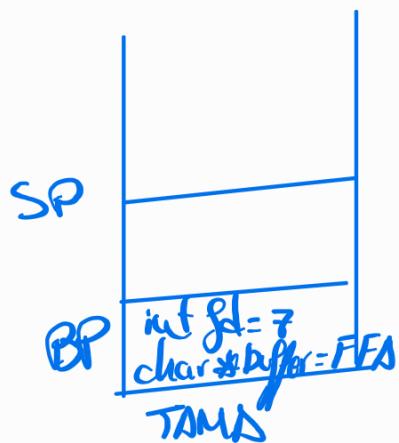


CPU

Registros de propósito general (En IA-32 son 32 bits)

- Accesibles por programas y clasificables en register de datos y registros de direcciones.
- Algunos importantes son: Registro/indice, puntero a segmento y puntero a pila (sp)

Puntero de pila (SP) → Clavóca a función
int read(int fd, char * buffer, int tama)



Registros internos

- **MAR** (Registro de dirección de memoria): contiene la dirección de la siguiente R/w
- **MBR** (Registro de buffer de memoria): contiene datos que van a escribirse en memoria o información que se recibe los datos que se leerán de memoria. Dirección
- **I/O DR** (registro de direcciones de entrada - salida)
- **I/O BR** (registro de buffer de E/S)

Otros registros

- PC: contiene la dirección de la siguiente ejecución a ir a buscar (fetch)
- IR: Contiene la última dirección que se fue a buscar (fetched)
- PSW: Contiene los códigos de condición, información para interrupciones y modo de ejecución del procesador

FLAGS

Este tipo de banderas de estado se encuentran en la palabra de estado o PSW, el cual tiene el tamaño de los registros de propósito general. Cada bit de dicha palabra es una bandera; bit activado = bandera activa

Ejemplo:

- I/O Privilege Level: Cuando este activa permite la entrada y salida privilegiada.
- Interrupt Enable Flag: Cuando está activa, permite que haya interrupciones.
- Carry Flag: Cuando está activa, los procesos se realizan en modo Kernel.

Memoria principal

- Modelo abstracto. Tabla o array lineal compuesto por un número de elementos: palabras de memoria con un tamaño.
- Dichas palabras son direcciones considerando en la posición 0 (cada número es una dirección).

• **Espacio de direcciones**: Conjunto de números que representa las direcciones de memoria.

OPERACIONES

→ **Lectura**: La memoria recibe una dirección y devuelve su contenido.

→ **Escritura**: La memoria recibe una dirección, un byte y su información y sobreescribe en esa posición.

DIRECCIONES Y TAMAÑO (2^x)

Si usamos x bits tenemos 2^x direcciones (Esp. de direcciones)

Tamaño = direcciones \times

El ancho de bus de dir. determina el tamaño de bits que contiene la palabra

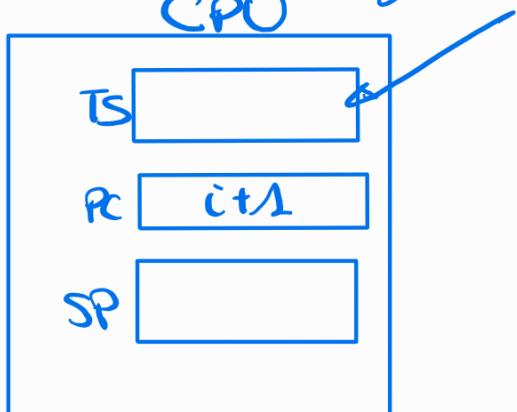
Modelos DE MEMORIA

→ Los programas no acceden directamente a la memoria física.

► Flat memory model: Consiste en tener un espacio de memoria扁平的, como un array de 2^n posiciones.

Segmented memory model: Los programas ven el espacio de memoria como grupos de memoria independientes llamados segmentos. Es la segmentación de TS, por tanto se usa un sistema de direccionamiento bidimensional.

Trabajar en Segmented Model



Selector de segmento
(indica el inicio del segmento)

Hace uso de Flat Memory Model.

Consejo de usar

una tabla de

segmentos de tamaño

en donde se guardan

certas cosas sobre

cada segmento.

Real-address mode memory model:

Es una implementación de segmentación

con limitaciones en el tamaño

de los segmentos 64 kB, y

en el espacio de memoria final

accesible, 2^{20} bytes.

T. Segmentos

Selector
de segmento

Dir	TAM	Permisos
-----	-----	----------

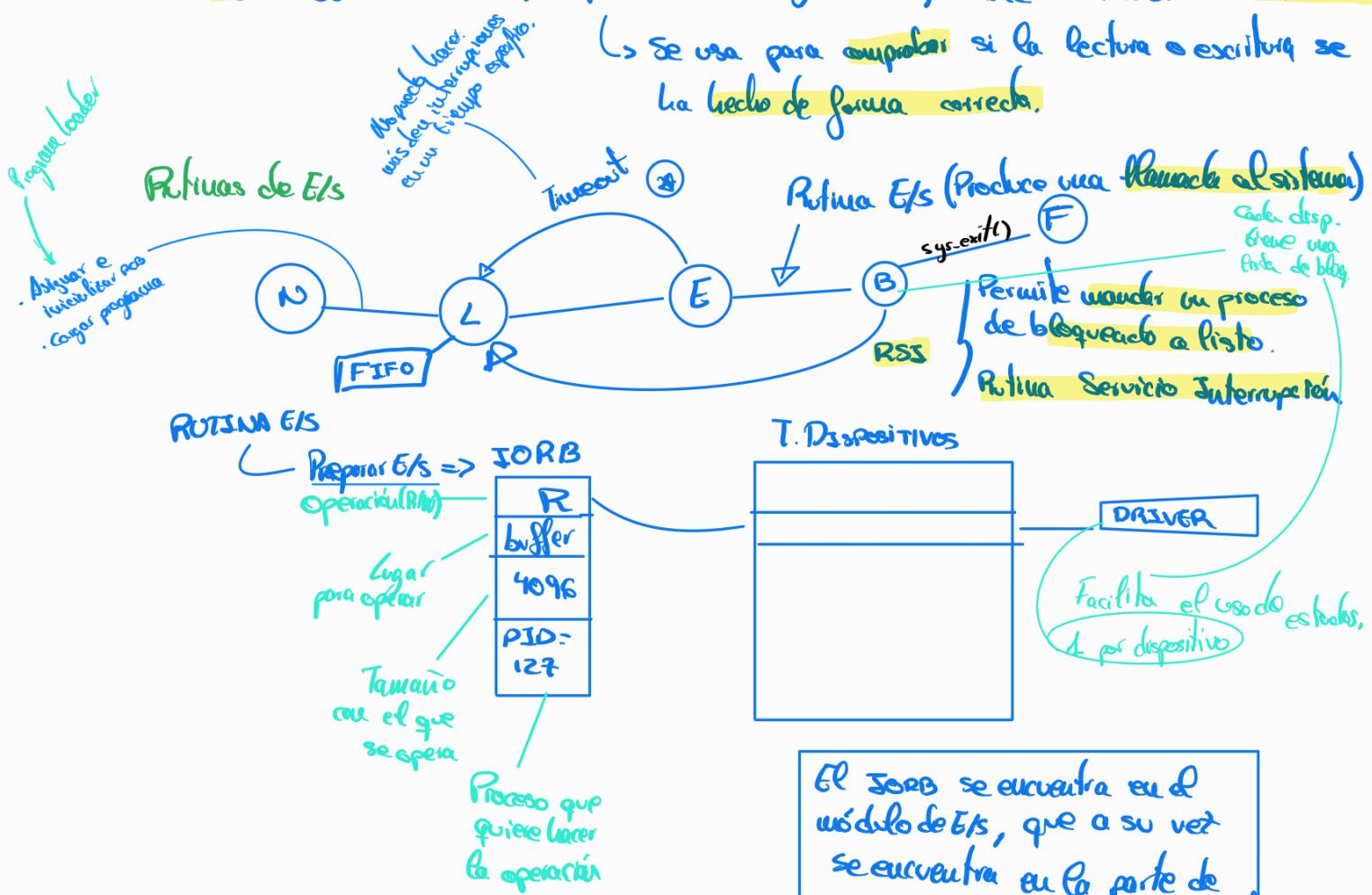
Se asocia a
fragmentación

Es muy similar a paginación.

Módulo de E/S

Funciones (cada)

- Control y temporización
- Comunicación procesador-dispositivos
- Almacenamiento temporal de datos (buffers). La velocidad dispositivos > velocidad E/S < procesador/memoria, procesador/módulo → memoria/módulo. Si hay DMA tendríamos comunicación directa entre E/S - memoria RAM pero solo hay ciertas zonas para uso de DMA.
Sustituir en pos. bajas
- Detección de errores (bit paridad o códigos CRC). Sigue lazo del Device Controller.



IORB - Input Output Request Block

RSS - Routine Service Interrupción (hay uno para DMA y otro para E/S dirigida por interr.)

FIFO - First In, First Out (se encarga de manejar el intercambio de procesos)

En todo proceso que usa memoria hay una zona de kernel y una zona de usuario.

Cambios de contextos

Tras la rutina de E/S hay un cambio de contexto donde entra en juego el planificador de CPU que da los PSD de los procesos listos. Tras obtener un nuevo PSD para ejecutar se ejecuta el DISPATCHER, almacenará el estado del proceso actual en el PCB.

PCB (Se encuentra en la parte Kernel de la memoria)

RS3

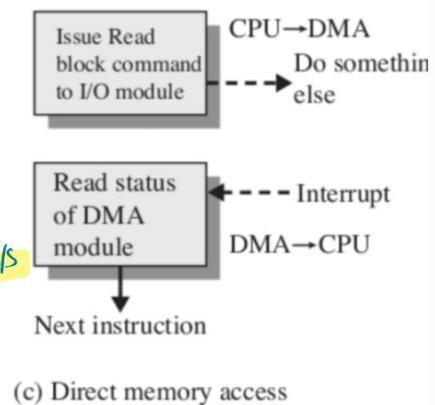
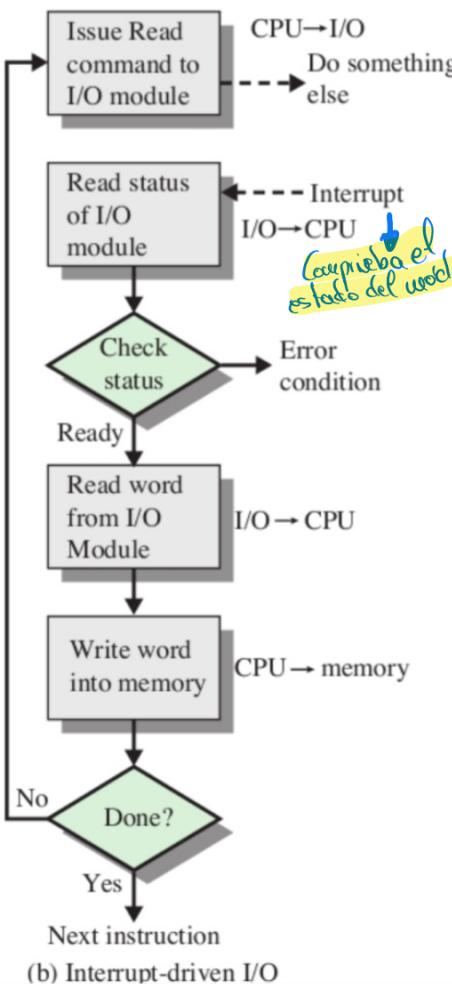
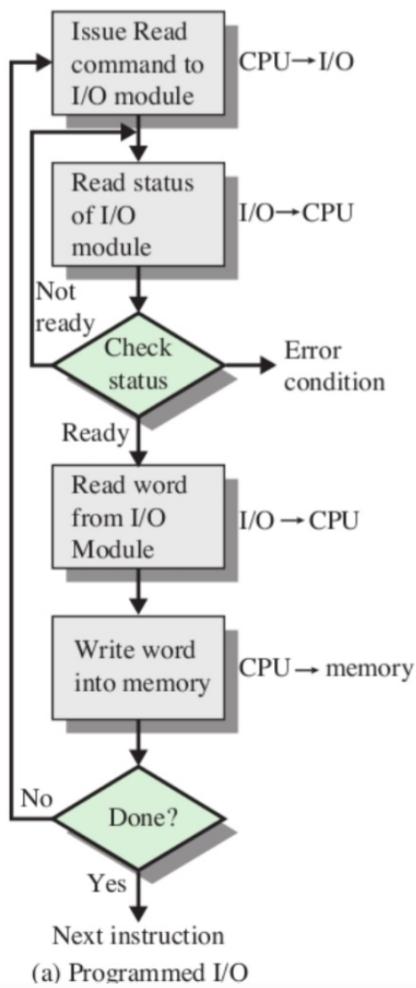
- PSD proceso
- Contexto registros } -PC
-PSW
-SP
- Memoria
- Estado

- ok/error
- Transferir páginas → RAM
- $PSD_{dispG} \cdot Estab = 'L'$
Ejecutar (PSD-dispG, LS, TOS)

Cambio de contexto (Es una función que realiza un cambio de proceso)

Ejecuta el planificador de CPU (de el siguiente proceso a ejecutar)
 DISPATCHER (PSD_{cou} , PSD) (Guarda la información del proceso que se está ejecutando en el PCB y que el Kernel del buffer de memoria)

Formas de E/S.

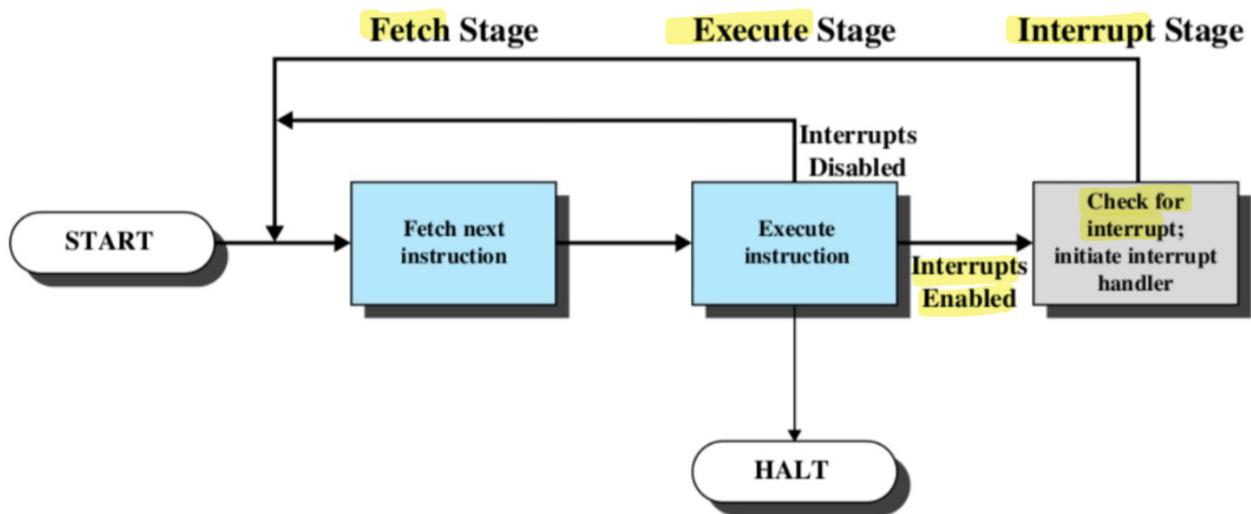


Po same thing else: cambio de contexto

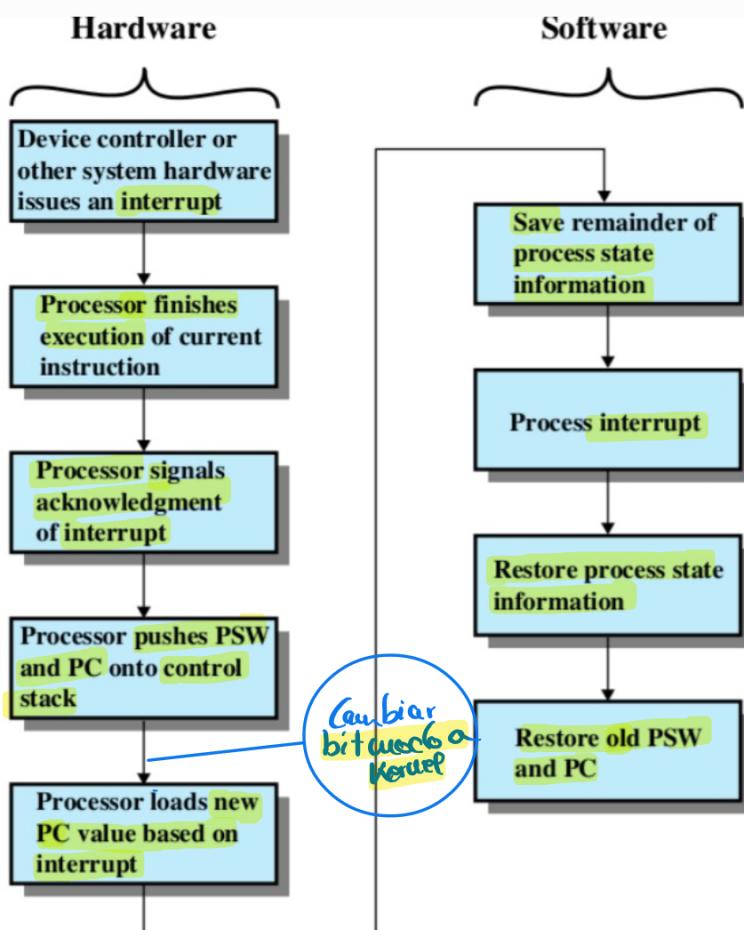
PSD.Gtrapeze
 $PSD_{cou}.Etab = 'B'$
 PSD-PlanifCPU
 DISPATCHER(PSD, PSD_{cou})
 Ejecutar(PSD_{cou}, Cou, SSDPS)
 LBSI

Funcionan igual pero en la otra se realizan comprobaciones mucho mas leves.

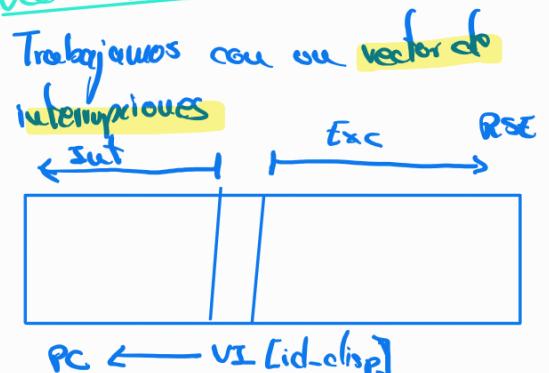
Ciclo de ejecución de instrucción con interrupciones.



Trafamiento de interrupciones



VectORIZACIÓN



Lo que se guarda en cada posición la dirección de memoria de la RSS correspondiente a ese dispositivo de E/S. Concretamente guarda la dirección de la primera instrucción de la RSS.

Cada instrucción final de una RSS tiene un "iret" que realiza un cambio de modo.

Lo mismo ocurre en RSE y manda al sistema

RSE - Routine Service Exception

Solo puedes ejecutar código del SO como motivo de:

- Llamada al sistema
- Superpuesto
- Excepción

Cambio de modo

- Como respuesta a una **interrupción**, el HW apila el PSW y el PC actuales \Rightarrow activa modo Kernel y carga el PC con el contenido del **vector de interrupciones** correspondiente.
- Como respuesta a una **excepción** (como resultado de la ejecución de una instrucción) el HW apila el PSW y PC actuales, activa el modo Kernel y carga el contenido de la **dirección** de la primera instrucción correspondiente a ese **error** en el PC.
- Como respuesta de una llamada al sistema solicitada por un programa en ejecución, el HW apila PC y PSW actuales, activa modo Kernel y carga el PC con el contenido del vector correspondiente al **gestor general de llamadas**.

Estructura del sistema operativo

Consiste en establecer la **funcionabilidad** del SO usando distintos módulos que interactúan entre ellos.

Funcionamiento de procesos

Un proceso es un **concepto** que permite **monitorear** y **controlar** sistémicamente la ejecución de varios programas en el procesador. Debe tener asociados los elementos:

- Programa a ejecutar
- PCB
- Consideraciones (multiprogramming, Time Slicing)

RSI (simple en uno)

```
check aux_error;  
PSD.Esto=0='1';  
Encero (PSD,CSSTOS);  
iret;
```



Sistemas de tiempo compartido

Son sistemas **multiprogrammado** y **multitarea** donde se maneja con **mejorado reloj**. De este manera el SO **toma el control** cada **cuadro**. Esto se hace a través de una **RSI-reloj()**.

```
RSI-reloj() {  
    Tiempo  
    cout<<cout++;  
    if (PSD.out>0)  
        PSD.out=0;
```



Usa ciertos **trucos** para tomar el control y poner la ejecución de un proceso pasándolo de nuevo al otro. Cuando se ejecuta el **RS3-ready()** se ejecuta un cambio de contexto con la posibilidad de pasar los procesos a otro.

Funcionalidad para procesos.

- **Creador del PCB** asociado a un **programa** que va a ejecutarse. **Eliminación del PCB** una vez finalizada la ejecución.
- **Bloqueo (sleep)** y **desbloqueo (wakeup)** de los procesos dependiendo de los **eventos** por los que debe esperar un programa para poder continuar su ejecución.
- Proporcionar **mecanismos** que permitan la **comunicación** y la **sincronización** de procesos.

Funcionalidad para memoria.

- **Protección de la página de memoria principal ocupada por el Kernel y protección de las regiones de memoria principal ocupadas por los programas.**

- **Compartirán espacio de las regiones ocupadas por los programas para facilitar la comunicación entre estos.**
- **Gestión automática** de la asignación/liberación de la memoria disponible para un programa en cualquier nivel de la jerarquía de memoria y de forma transparente al programador.
- **Mantener información** sobre la memoria asignada a los procesos, vínculos, etc. y la memoria libre en cualquier nivel de la jerarquía.

- **Implementar algoritmos** para decidir cuanta memoria asignar a cada proceso y cuando debe ser liberada toda la memoria asignada a un proceso manteniéndola en almacenamiento secundario.

Señalizar de T-páginas.
Marco, páginas.
También una lista de
ubicación física
para saber en qué
parte del disco estab
el programa.

Funcionalidad para archivos y directorios

- Un **archivo** es una colección de información identificada por nombre que reside en almacenamiento secundario.
- La funcionalidad relacionada con los archivos está asociada al sistema de archivos, el cual permite:
 - Crear, eliminar, copiar, renombrar,... archivos y directorios. **Operaciones** con →
 - Mantiene los **atributos de archivo** en una estructura de datos (**inode** en UNIX/Linux).
 - Abrir y cerrar sesiones de trabajo con archivos, leer, escribir, variar el puntero de lectura/escritura.
- Gestionar la asignación y liberación de **espacio en disco** para mantener la información de los archivos y directorios.
- Mantener la ubicación en **disco** de los **datos** asociados a archivos y directorios así como los **metadatos del Sistema de Archivos**.

Funcionalidad para recurso del SO

Planeación y gestión de recursos

- **Equitatividad.** El acceso a los recursos debe estar garantizado para todos los procesos.
- **Respuesta diferencial.** El SO debe **planificar** la asignación de los recursos teniendo en cuenta características diferenciales de los clientes, procesos u otras entidades (p.e. IORBs).
- **Eficiencia.** El SO debe **maximizar** la productividad, minimizar el tiempo de respuesta y, en sistemas de tiempo **compartido**, permitir el trabajo simultáneo de tantos usuarios como sea posible.

Funcionalidad para disp. de E/S

- Controlar el funcionamiento de los dispositivos de E/S.
- Proteger su utilización de forma directa por parte de los programas de aplicación: acceso mediante API de llamadas al sistema.
- Proporcionar una **interfaz independiente** de las particularidades de los dispositivos al resto del SO y a los programadores: interfaz estándar para todos los dispositivos.
- Manejador de dispositivo (**Device Driver**). Módulo software que gestiona un determinado dispositivo y se encarga de implementar las operaciones del interfaz, implementar el manejo de interrupciones,

Componentes generales de un SO.

- La funcionalidad descrita previamente se suele agrupar en los siguientes componentes fundamentales.
 - Gestor de procesos.
 - Gestor de memoria.
 - Gestor de archivos.
 - Gestor de E/S.
 - Gestor de comunicaciones.

Protección y seguridad

- **Protección**. Cualquier mecanismo para controlar el acceso de procesos o usuarios a recursos del SO (ej. control de acceso a archivos en el Filesystem).
- **Seguridad**. Defensa del sistema frente a ataques internos o externos (ej. denegación de servicio, worms, viruses, robo de identidad).
- Los sistemas distinguen usuarios para determinar quién puede hacer que. UID y GID se asocian con procesos y archivos para determinar control de acceso.

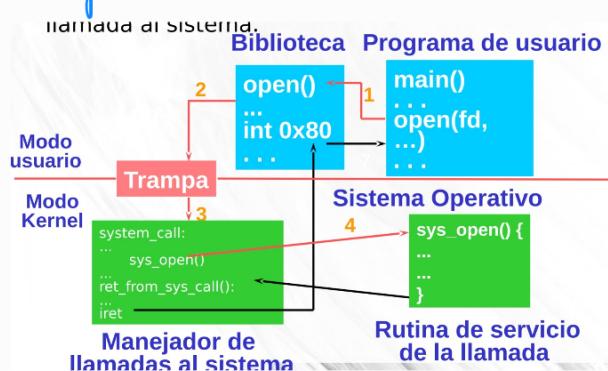
Interfaces entre usuario y SO.

- **Command Line Interface** (CLI). A veces implementado en el kernel y la mayoría mediante shells.
- Los shells permiten ejecutar órdenes built-in (implementadas en el propio programa shell) o órdenes (programas) que residen en el FS.
- **Graphic User Interface** (GUI). Interfaz que utiliza la metáfora de escritorio (Desktop metaphor). Usa teclado/ratón.
- **System calls** (Llamadas al sistema). Interfaz de programación a los servicios proporcionados por el SO. Los programas lo utilizan mediante una API en lugar de acceso directo a la system call.

Llamadas al sistema

se solicita la ejecución del kernel para ese específico, en diferencia con la ejecución de un proceso que se produce como resultado de un fallo en una instrucción o para arrojar del flujo de un programa de E/S.

Hay una serie de componentes implicados:



o Paso de parámetros:

- En registros de la CPU
- Parámetros en memoria para la dirección del buffer en el registro
- Parámetros en pila

Implementación

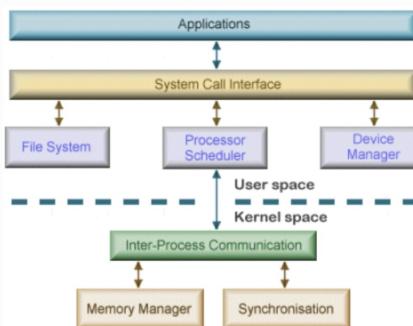
- o Se asocia un número con cada llamada al sistema proporcionada por el SO.
- o El ruteador de llamadas rastrea la llamada requerida por el programador y devuelve el estado de finalización y los valores de retorno.

Arquitecturas monolíticas

- El **SO** es el único programación que se ejecuta en el **cuadrado privilegiado** del procesador (ring 0)
- La **dependería** entre los distintos módulos son **complejas** salvo para algunos elementos **bien establecidos**
- El **modelo de obtención de servicios** es la **llamada a procedimiento**
- La fuerte dependencia entre módulos provoca **dificultades para comprender el código** y realizar **modificaciones**.
- Al ser **un solo programación** cargado en **memoria**, el **fallo de un módulo** puede provocar la **caida del sistema**.

Arquitecturas microkernel

- Una **pequeña** parte de la funcionalidad del SO está implementada como **Kernel** y el resto de procesos de **usuario**.
- El **microkernel** soporta **memoria virtual de bajo nivel**, **creación de procesos** y, **comunicación y sincronización**



- El **módulo** de obtención de servicios es **por medio de mensajes entre procesos**.
- La app **solicita un servicio** al ser (server, &ui), y **envia la solicitud del servicio**, **recibe (serv, &ui)** → **se mandan órdenes al servidor (llamada al sys)** → **el kernel** **envia la solicitud de servicio** y **envia el mensaje al servidor Serv.**
- El **microkernel** **envia la solicitud de servicio** y **envia el resultado** al **kernel**.
- El **servidor** **obtiene el mensaje**, **genera el resultado** y **envia el resultado** al **kernel** para que este lo devuelva a la aplicación, r.
- En la arquitectura **microkernel** **no se usan los drivers**, sino que **los usuarios programan directamente** que tienen acceso a ellos.

Ventajas

Fiabilidad: Un **error de un módulo** solo hace que se caiga una parte del sistema que se puede **recuperar** levantando el proceso de servicio.

Extensibilidad: incluir **nuevos servicios** como **procesos de usuarios**

Desventajas

Pérdida redundante que la memoria porque se **usa más memoria de contexto y modo**

Virtualización

- Se usaba para referir a **maquinas virtuales** como un **duplicado** o **replicado** del **computador real**. 1960
- Puede usar los **recursos HW virtualizados** por el **VMM** (**Virtual Machine Monitor**) Hypervisor Capa SO
 - ↳ **Software que proporciona la abstracción de la máquina real al VMs.**

- **1974** → se dan **los requisitos** para que el **HW maneje una VM**:

- **Equivocación** (el programa debe ejecutarse igual que en la máquina real)
- **Control de recursos** (el **VMM** controla los recursos virtualizados)
- **Eficiencia** (la mayoría de las instrucciones se ejecutan solo si no interviene el VMM)

Tipos de virtualización

de abstracción

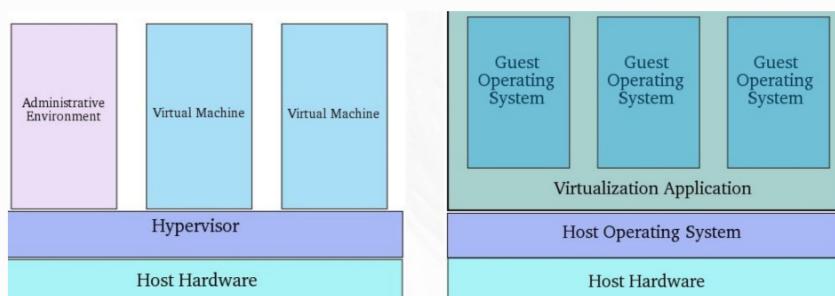
- **Hipervisor y maquinas virtuales:** **Capas software** que proporciona recursos virtuales a maquinas virtuales; que esté situada entre el software ejecutando el **HW real**. La VMM crea un entorno de trabajo para el software a ejecutar. La VMM puede ser SV, HV o FV y el pc donde se ejecuta es el **host machine** y la virtual machine es su **guest machine**.

Ahora el desarrollo y gestión de VMs se llama **server virtualization**.

La virtualización permite ejecutar varios S.O. en uno solo. Además, dispone de una serie de **recursos virtuales** proporcionados por la VMM y sobre ellos ejecuta el SO, las bibliotecas y las apps.

- Por último, el **ratio de consolidación** indica el nº de VMs que proporciona un determinado hipervisor.

- **Tipos de hipervisor:**
 - 1, **native**. Se ejecutan directamente en el host HW para proporcionar VM a los invitados.
 - 2, **hosted**. Se ejecutan sobre so conocido como los demás programas y el **SO invitado** se ejecuta en la abstracción proporcionada por el supervisor.



Hipervisor Tipo 1 (native o bare-metal) Hipervisor Tipo 2 (hosted)

Motivos de virtualización:

- Ejecutar **multiples SOs** en un **HW**
- **consolidación de servidores**
- **desarrollo de VMs**. En VM es más fácil de controlar y observar que en HW real

Evaluación de distintos

Un poco sobre implementación

- El hipervisor traduce las peticiones sobre los recursos virtuales que proporciona a peticiones sobre los recursos reales del hardware.
- Esta traducción provoca degradación en el rendimiento.
- Una VM se representa mediante archivos:
 - Archivo de configuración: CPUs, memoria, dispositivos E/S accesibles, etc.
 - Archivo de almacenamiento: discos virtuales y su correspondiente soporte mediante archivos reales.

Comparando hipervisores tipo 1 y tipo 2

- Los hipervisores tipo 1 obtienen mejor rendimiento que los tipo 2 ya que:
 - Disponen de todos los recursos para las VM.
 - Los múltiples niveles de abstracción entre el SO invitado y el HW real en el tipo 2 no permiten alto rendimiento de la máquina virtual.
- Los hipervisores tipo 2 permiten realizar virtualización sin tener que dedicar toda la máquina a dicho fin. Ejemplo, prueba de kernels o puesta a punto de servidores.

Paravirtualización

- Técnica en la que el hipervisor proporciona una API (hypercalls en Xen) y el SO que se ejecuta en una VM utiliza dicha API.
- Implicaciones de esta técnica:
 - El código del guest OS debe estar disponible.
 - Reemplazar instrucciones que necesitan ring 0 por hypercalls: ej. cli por vm_handle_cli.
 - Recompilar el guest kernel y usarlo.

HW-assisted virtualization: Ordenadores PC

Hoy en día

- Intel (VT-x) y AMD (AMD-V) presentan HW que soporta virtualización (CPU flag vmx en Intel)
 - ¿Cómo se entiende actualmente la virtualización? Cada elemento HW (ISA, I/O, interrupciones, memoria...) debe poder ser proporcionado por el hipervisor mediante una VM.
 - El hipervisor (Super/Hiper OS) tipo 1 debe proporcionar:
 - Almacenamiento independiente de SO para proporcionar recursos a las VM (también se les conoce como virtual environments, VE)
 - Switching de VEs para asignar recursos gestionados por el hipervisor.
- OS-level virtualization: Kernel permite muchas instancias aisladas a nivel de usuario de manera que un programa no verá todos los recursos del SO sino solo los asignados a la instancia

Containers

Proporcionan un entorno aislado para la ejecución de programas sobre el mismo SO. En el caso de Linux, el Kernel control group permite la agrupación de procesos y la gestión de recursos del sistema. En particular, permite que distintas jerarquías de procesos coexisten en el mismo SO; de manera que a cada jerarquía se le asigna un conjunto de recursos del sistema: memoria máxima, prioridad para CPU o prioridad para dispositivos de E/S.

Funciones del hipervisor

- Gestión de la ejecución de MVs:
 - Planificación de MVs para ejecución en CPU
 - Gestión de la memoria virtual para aislar VMs.
 - Cambio de contexto y emulación de timer e interrupciones.
- Emulación de dispositivos y control de acceso.
- Ejecución de instrucciones privilegiadas.
- Gestión de Máquinas Virtuales.
- Administración del hipervisor.

HW-assisted virtualization (actual)

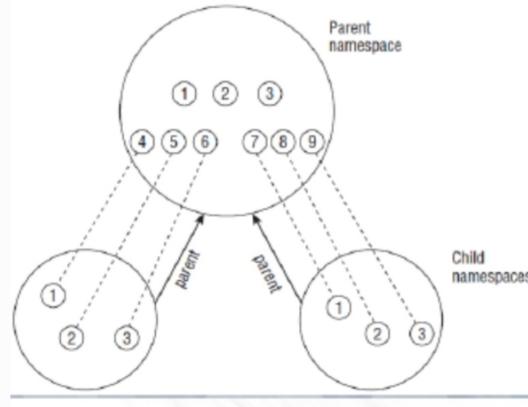
- Primer HW para asistir a la virtualización incluido en IBM System/370 de 1972 como soporte para VM/370 (1º SO que proporciona VM).
- Arquitectura x86 no soportaba HW-assisted virtualization por lo que se desarrollaron soluciones software. El problema es que los SO que se ejecutan sobre las máquinas virtuales necesitan ejecutarse en ring 0... Soluciones:
 - Paravirtualización
 - Full virtualization.

Full virtualization (poblar instrucciones para instrucción)

- Esta técnica fue implementada en la primera generación de VMM (hypervisor) en x86.
- La idea es traducción binaria para atrapar y virtualizar la ejecución de instrucciones sensibles (ring 0), ej. cli.
- Las instrucciones sensibles se detectan (estática o dinámicamente) y se reemplazan con trampas al VMM.
- Esta técnica puede provocar sobrecarga en el rendimiento con respecto a una VM que se ejecute en HW que soporta virtualización, IBM System/370.
- Solución final: HW-assisted virtualization

Linux Namespaces

Un espacio de nombres permite hacer visibles ciertos recursos del kernel de forma única dentro de ese espacio. Algunos usos son: System V IPC, sistemas de archivos montados, PID que sirve para identificadores de procesos, y por último, userid que permite tener recursos por usuario.



VNL

Su uso más típico es un kernel modificado que contiene extensiones para controlar múltiples máquinas virtuales cada una con su SO invitado.

KVM (Kernel-based Virtual Machine)

Consiste en una implementación "full virtualization" para Linux sobre HW x86 que contenga las extensiones correspondientes: Intel VT o AMD-V.

Sistemas operativos de propósito específico

Sistema de Tiempo Real

Sus usos más comunes consisten en sistemas de control para aplicaciones especializadas. Estos sistemas RTOS deben garantizar la correctitud no solo en el resultado final de la computación sino también la correcta utilización del tiempo para dar ese resultado.

Como problema, resulta complicado planificar las actividades para satisfacer todos los requisitos de tiempo, es decir, la planificabilidad. Ligado a esto, algunos procesos son clasificados como procesos de tiempo real, los cuales tienen como objetivo procesar eventos que se producen regularmente o no, en el sistema de control. Estos eventos ocurren en tiempo real por lo que el tiempo de respuesta tiene un límite que especifica cuándo se cumple la ejecución del proceso y cuándo se arroja la misma.

Características

1. Determinismo y reactividad. Grado en el que el sistema puede resolver todos los procesos RT cumpliendo los plazos.

- Tiempo de respuesta
- Determinismo → velocidad de respuesta del RTOS frente interrupciones (t^o cortado)
 - Reactividad → Tiempazo tarda el RTOS en la respuesta

2. Control de la prioridad de los procesos RT. El usuario debe poder controlar la prioridad del proceso RT

3. Fiabilidad = tolerancia a fallos. Considera el redundancia en los procesos. Algunos pueden ser muy peligroso; por tanto, peligra la estabilidad del sistema. El cual es estable cuando, aunque no cumple los requisitos siempre, los ciclos de los programas son más importantes.

Sistemas Operativos para Sistemas Empotrados

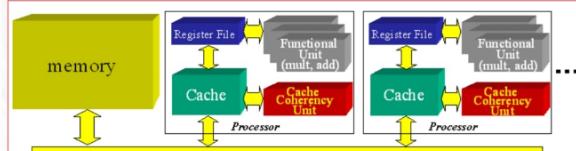
Este tipo de SO está especializado en usar computadoras incluidas en sistemas grandes; de manera que un sistema empotrado es un computador que forma parte de una maquinaria. Una desventaja de los SO es que tiene una funcionalidad muy limitada que un SO normal que viene determinada por el sistema empotrado.

Cada requisito, debe proporcionar la robustez en la ejecución de procesos, ya que debe facilitar con restrictiones de mejoras y protección de computadoras.

Computación de altas prestaciones

Las tareas en HPC suelen estar compuestas de un grupo de procesos fuertemente acoplados para resolver la tarea, luego necesita una conexión de alta velocidad entre nodos de cálculo; y estar compuestos de trabajos secuenciales que se puedan ejecutar individualmente en diferentes nodos de cálculo.

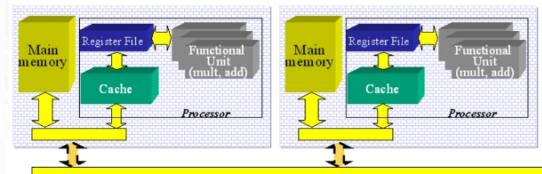
Multiprocessor Architecture



Cache coherency unit will intervene if two or more processors attempt to update same cache line

- All memory (and I/O) is shared by all processors
- Read/write conflicts between processors on the same memory location are resolved by cache coherency unit
- Programming model is an extension of single processor programming model

Multicomputer Architecture



- All memory and I/O path are independent
- Data movement across the interconnect is "slow"
- Programming model is based on message passing
 - Processors explicitly engage in communication by sending and receiving data

Sistemas Operativos para HPC

Se plantean dos alternativas de cara al procesamiento paralelo:

- Sistema: cada procesador ejecuta una copia idéntica del SO (baja redondeo)
- Arquitectura: Un procesador unívoco ejecuta el SO y los procesadores asíncronos ejecutan procesos de usuario (peor escalabilidad)

Con respecto a estos sistemas tienen algunas ventajas:

1. Soportan aplicaciones paralelas que quieren un aumento de velocidad de procesamiento complejas.
2. Necesitan primitivas básicas para dividir una tarea en múltiples actividades paralelas.
3. Proporciona una comunicación y sincronización eficiente entre esas actividades.
4. Pueden tener mechanismos de Tolerancia a Fallos.