TEMA1-FSoftware.pdf

Airin86

Fundamentos del Software

1º Doble Grado en Ingeniería Informática y Matemáticas

Facultad de Ciencias
Universidad de Granada

Fundamentos del Software Irene Pérez Martínez

TEMA 1: Sistemas de cómputo

1. Componentes de un Sistema de cómputo

Definiciones

Tenemos distintos conceptos que trataremos mucho en esta asignatura (bit, computador, hardware y software, Sistema Informático...). Veamos algunos de ellos:

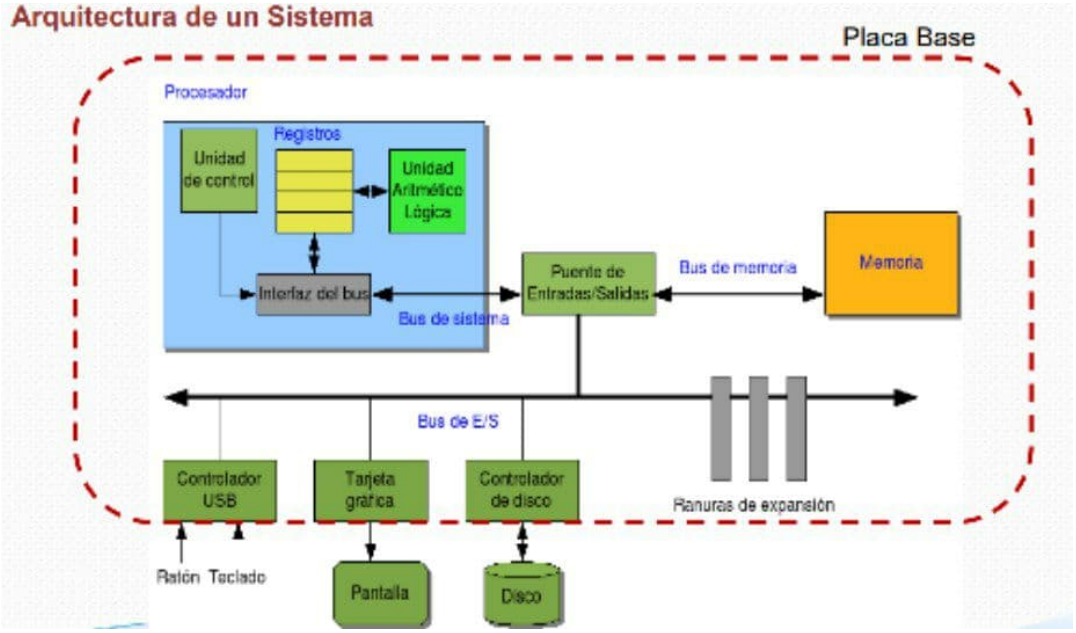
- **Bit**: unidad mínima de información. Representa un nivel alto de tensión o nivel bajo. Se asimila, según la lógica booleana, con los estados falso y verdadero. Por cada n bits podemos representar 2^n elementos
- **Byte**: es la unidad mínima para direccionar, y lo conforman 8 bits

Multiples of Bits		
Unit (Symbol)	**Value (SI)**	**Value (Binary)**
Kilobit (Kb) (Kbit)	10^3	2^{10}
Megabit (Mb) (Mbit)	10^6	2^{20}
Gigabit (Gb) (Gbit)	10^9	2^{30}
Terabit (Tb) (Tbit)	10^{12}	2^{40}
Petabit (Pb) (Pbit)	10^{15}	2^{50}
Exabit (Eb) (Ebit)	10^{18}	2^{60}
Zettabit (Zb) (Zbit)	10^{21}	2^{70}
Yottabit (Yb) (Ybit)	10^{24}	2^{80}

- **Instrucción**: conjunto de símbolos insertados en un secuencia estructurada o específica que el procesador interpreta y ejecuta.
- **Datos**: secuencia de uno o más símbolos que representan una unidad de información, es decir, a los que se les da un significado.
- **Lenguaje natural**: lenguaje que se utiliza para hablar entre personas, el español, francés, chino...
- **Lenguaje de programación de alto nivel**: lenguaje utilizado para programar de manera que el computador lo entiende y que es bastante cercano al lenguaje natural (C++, java...). Suelen usar palabras de algún lenguaje natural (if, while, else...)
- **Lenguaje Ensamblador**: es un lenguaje de más bajo nivel, con una estructura diferente al de los lenguajes superiores. Suele ser el algo intermedio entre el lenguaje de alto nivel y el lenguaje máquina.
- **Lenguaje máquina**: es el lenguaje que entiendes los computadores, basados en 1s y 0s.
- **Compilador e Intérprete**: son los traductores entre lenguajes de alto y bajo nivel.
    - El **compilador** tiene dos pasos: primero toma el programa fuente (en lenguaje de alto nivel) y genera un programa equivalente (en un lenguaje de bajo nivel), y en segundo lugar, si no hay errores en el código y a partir del programa generado con el nuevo

lenguaje, establece una salida usando las entradas que el usuario proporciona. Llamemos a las fases ""análisis" y "generación".
    - El **intérprete**, por otro lado, se diferencia en que hace lo mismo que el compilador pero en un solo paso. Toma a la vez el código y la entrada, para proporcionar la salida.

- **Ensamblador**: permite que el compilador no tenga que traducir desde un lenguaje de alto nivel hasta lenguaje máquina, si no que pase de alto nivel a lenguaje ensamblador.



- **Sistema Informático**: sistema que permite almacenar y procesar información. Es el conjunto de partes interrelacionadas: hardware, software y usuario.
- **Hardware**: todo elemento físico que compone un computador, incluyendo tanto los elementos exteriores (torre, teclado, pantalla...) como los interiores (procesador, microchips, baterías, ventilador...).
- **Software**: conjunto de programas, instrucciones y reglas informáticas que permiten ejecutar distintas tareas en un computador
- **Firmware**: software más cercano al hardware. Es el bloque de instrucciones máquina para propósitos específicos, grabado en una memoria (normalmente acciones de lectura/escritura), que establece la lógica de más bajo nivel que controla los circuitos electrónicos de un dispositivo de cualquier tipo. Está relacionado con el concepto de la **BIOS**: parte del firmware, que tiene como objetivo encender el computador y preparar su entorno.

2. Capa Hardware

2.1 Estructura de un ordenador

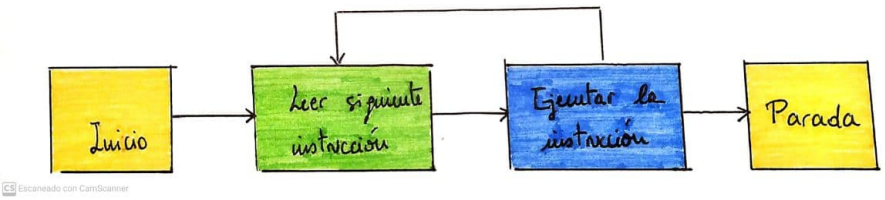
- **Registro de control y de estado**: registros de memoria en los que se deja constancia de algunas condiciones que se dieron en la última operación realizada y que normalmente son relevantes para el desarrollo de la ejecución que se está llevando a cabo. Tenemos algunos tipos:
    - **Contador de programa (PC)**: contiene la dirección de la próxima instrucción que se va a leer de memoria.
    - **Puntero a pila (SP)**: se utiliza en muchos lenguajes de programación para realizar consultas de datos. Pueden ser FIFO (los datos se leen en el orden en el que se almacenan), LIFO (se lee desde el último en ser añadido hacia el primero)... La pila se suele usar para casos donde llamamos a una función auxiliar desde dentro de nuestra función principal.
    - **Registro de instrucción (IR)**: contienen la última instrucción leída, es decir, la última captada de memoria y que se ha ejecutado.
- **Registro de estado (bit informativos)**: permiten almacenar si una instrucción se ha ejecutado con éxito o no, y establecer condiciones para decidir si se ejecuta o no una instrucción. Suele estar ligado a un conjunto de registros (conjunto de estados), que contienen códigos de condición para determinar cuándo se puede o no ejecutar una instrucción (también tienen bits informativos, determinan si el usuario tiene permiso para ejecutar).
- **Registros visibles para el programador**: son claves para poder tener un acceso más rápido a la información que necesita un programa (unidad de control, unidad aritmetico-lógica). Por ejemplo, si sabemos que hay una serie de datos que van a ser usados de forma muy recurrente en un programa, el programador puede decidir almacenar eso en los registros visibles y así reducir el tiempo de la ejecución porque no se tiene que acceder a memoria principal tantas veces.

2.2 Ejecución de instrucciones (IMPORTANTE)

Para llevar a cabo la ejecución de una serie de instrucciones se deben realizar dos pasos en bucle. Los pasos para **procesar una instrucción**:

1. **Lectura**: El procesador lee (busca) instrucciones de la memoria, una cada vez.
2. **Ejecución**: Una vez sabe qué debe realizar, el procesador ejecuta la instrucción

Para llevar a cabo el procesamiento, el **proceso a seguir** tiene tres pasos. En primer lugar, está claro, se leerá la instrucción pedida, cuya dirección está almacenada de el PC (contador del programa, nos indica por qué instrucción estamos). Así pues, se busca en la dirección indicada para saber qué dice la instrucción y por tanto qué debemos hacer. En segundo lugar, se incrementa el PC en una unidad para que cuando acabe el ciclo sepamos qué instrucción debemos ejecutar. Por último, se ejecuta la instrucción. Acabado el ciclo, se estudia la siguiente instrucción de la misma manera hasta que se nos indique que hemos acabado.



Que quede claro que **procesar una instrucción** se hace en **dos pasos**, mientras que el **proceso a seguir** tiene **tres pasos**.

```
NOTA: Es muy factible que caiga este apartado como pregunta tanto de desarrollo
como de tipo test. En desarrollo explicar cada uno de los pasos de procesar una
instrucción, explicar los tres pasos del proceso a seguir y si es posible hacer
el dibujo (esquema) de la imagen.
```

2.3 Técnicas de comunicación de E/S (entrada/salida)

Contamos con tres técnicas que indican cómo pueden interactuar los dispositivos de E/S. La primera de ellas será algo mala, la segunda regular y la tercera la mejor opción de las que estudiamos. Despúes de verlas definiremos lo que son las interrupciones y las excepciones.

Caso 1: E/S programada

Es el método más intuitivo. El procesador se comunica con el dispositivo de entrada/salida y ve si está libre (disponible), por tanto tiene un **papel activo**. Si está disponible, el procesador lee el primer dato (o grupo de datos, en función del tamaño del bus) que se desea enviar al dispositivo E/S, lo manda y vuelve a repetir lo mismo con los datos siguientes.

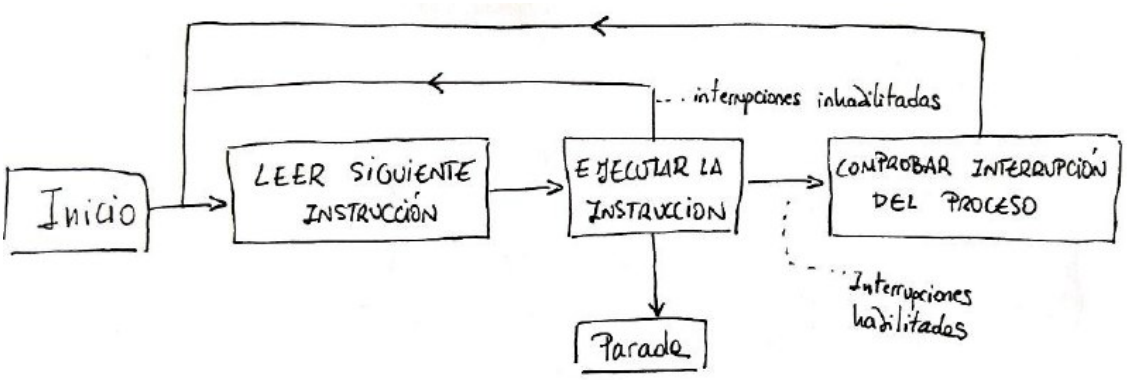
Su problema es que dedica demasiado tiempo esperando a que el dispositivo de E/S acabe lo que esté haciendo. Si está ocupado el dispositivo de E/S, el procesador parará todo su trabajo hasta que esté disponible y entonces reanudar el envío de datos. Una solución sería que el procesador se ocupase de otras tareas mientras el módulo de E/S está ocupada.

Caso 2: E/S dirigida por interrupciones

Cada una de las distintas interrupciones que se pueden dar en el ordenador tienen un registro asociado que dice cómo resolverlas. Cuando se da una interrupción, el procesador almacenará el estado del proceso que esté ejecutando en ese momento, buscará en función del caso el registro de la interrupción, realizará lo que indique el registro y volverá a ejecutar el proceso inicial. El procesador tiene un **papel semiactivo**. En resumen, el procesador estará ejecutando cosas hasta que haya una interrupción; entonces parará lo que está haciendo para "resolver" la interrupción en función de lo que indique el registro asociado.

```
APUNTE
-> Interrupción: se definirá concretamente más adelante, pero son acciones de
parada de ejecución de dispositivos de E/S
-> Registros: almacenados en memoria principal
```

Cada vez que se da una interrupción, se ejecuta un determinado programa de memoria en función del tipo de parada que se ha dado. Existen registros que almacenan qué programa ha de ejecutarse en función del tipo de interrupción que se puede dar. Son eventos asíncronos, por lo que cuando se de el error se almacena el estado actual del programa, se lleva a cabo la ejecucion del programa de error y luego se vuelve a la ejecución inicial.



El principal problema que tiene esta técnica es que debemos almacenar el estado de programa siempre que haya una interrupción, lo que hace perder mucho espacio de almacenamiento y mucho tiempo guardando esa información.

Caso 3: Acceso directo a memoria (DMA)

En lugar de que sea el procesador el que se encargue de gestionar cuándo está o no ocupado el modulo de E/S, existe un módulo aparte que se encarga de gestionarlo. El procesador manda los datos al DMA y es él el que se encarga de gestionar el envío al módulo de E/S. El procesador **no tiene papel**, o tiene un papel pasivo, ya que no realiza nada directamente sino que le deja el trabajo completo al DMA. Así pues el procesador no pierde tiempo esperando.

Cuando la CPU tiene que realizar alguna tarea relacionada con un módulo de entrada/salida manda la orden al DMA y mientras puede realizar otras tareas. Cuando acaba el DMA, avisa a la CPU mediante una interrupción para que este lea el estado del módulo de DMA y continúa con la siguiente instrucción.

```
NOTA: Apartado muy factible de ser preguntado en examen. Posibles preguntas de
examen serían las siguientes:
        ¿Cuál es la técnica más eficiente?
            R: la tercera, pierdes menos tiempo en la CPU
        ¿Qué opciones (técnicas) tenemos para comunicarnos con E/S?
            R: explicación de las tres vistas
        ¿En qué se diferencian las distintas técnicas?
            R: Pues en varias cosas, algunos ejemplos serían
                - La tercera hace que CPU y E/S no interactúen de manera directa
                - En la segunda y en la tercera necesitamos interrupciones
                - En cada caso la CPU tiene una actuación diferente (activa,
semiactiva o pasiva)
                - La productividad de la CPU es diferente en cada caso (caso 1 <
caso 2 < caso 3)
                etc.
```

Interrupciones

Una interrupción es una **suspensión temporal** de la ejecución de un proceso, para pasar a ejecutar una subrutina de servicio de interrupción, la cual, por lo general, no forma parte del programa, sino que pertenece al SO o a la BIOS. Cuando se acaba la ejecución de dicha subrutina, se reanuda la ejecución del programa. Las interrupciones se generan por dispositivos de E/S con la finalidad de llamar la atención de la CPU por alguna cuestión (lo más usual, que ya han realizado el trabajo que debían y tienen que avisar). Gracias a estos mecanismos se pueden realizar otras tareas en la CPU mientras se está usando el módulo de E/S (como hemos visto en el caso 2 de antes, aunque en el caso 3 también se usan pero entre el DMA y el dispositivo de E/S).

Excepciones

Es un **evento inesperado** generado por alguna condición que ocurre durante la ejecución de una instrucción máquina (como por ejemplo división entre 0, dirección inválida, no poder realizar una instrucción por no tener permiso...). Son interrupciones producidas por la propia CPU. Básicamente es un aviso y actuación por parte de la CPU ante una situación fuera de lo usual, pero no por ello debe estar relacionado con errores. Es un evento síncrono.

Puede ser gestionado por un programa durante su ejecución, si tiene esa capacidad, o si es un conjunto predefinido de excepciones las puede resolver el Sistema Operativo. Cuando el SO trata la excepción (rutina de tratamiento de excepción) se reinicia la instrucción donde se produjo.

2.4 Protección

Vamos a ver la protección en tres elementos diferentes de la computadora: el procesador, los dispositivos de entrada/salida y la memoria.

Protección del procesador

Tenemos un mecanismo hardware, el **funcionamiento en modo dual**, que permite una correcta operación del SO. Para ello, agrega un bit llamado bit de modo, que indica dos modos posibles de operación (monitor (0) ó usuario (1)). Eso permite identificar los programas incorrectos, lo que propicia que el SO no ejecute de forma incorrecta otros programas y se cree una cadena eterna de errores (por ejemplo, que una instrucción privilegiada actúe mal y afecte a instrucciones no privilegidas).

Esto se traduce en dos modos de ejecución de instrucciones:

1. **Instrucciones privilegiadas**: En modo kernel o supervisor; instrucciones que pueden interferir en la ejecución de un programa cualquiera o del SO.
2. **Instrucciones no privilegiadas**: En modo usuario; su ejecución no representa un problema de seguridad para el resto de programas.

Protección de los dispositivos de E/S

Es necesario proteger los recursos de E/S, y para ello está preestablecido que todas las instrucciones máquina de acceso a los mismos solo pueden ejecutarse en modo kernel, es decir, son privilegiadas. Solo se puede acceder a ellos mediante peticiones al SO, por lo que será él el que lo regule.

Protección de memoria

Cada programa en ejecución requiere un espacio en memoria, por lo que hay que proteger cada zona asignada a programas, además de la zona de código del SO y sus datos (rutinas para interrupciones, por ejemplo). Para ello tenemos un hardware especial, la **unidad de gestión de memoria (MMU)**, que controla las regiones de memoria asignadas a los programas y vela por su protección.

3. El Sistema Operativo

El sistema operativo, o SO, tan nombrado hasta ahora en este tema es un **programa** o conjunto de programas que controla la ejecución de los programas de aplicación (básicamente todos los programas que tienen que ver con el trabajo del usuario, cosas que el usuario hace o pide). Actúa como interfaz entre el usuario y el hardware de la computadora.

SO visto como interfaz

Si vemos el Sistema Operativo como interfaz, vemos que presenta al usuario un entorno más sencillo de manejar que tratar directamente con el hardware. Es decir, actúa como homogeneizador ante las distintas acciones que se pueden realizar, ocultando las complejidades que subyacen. No es útil solo a nivel de usuario común, sino también para programadores: siempre es más sencillo programar a alto nivel (lenguajes de programación casi similares al lenguaje hablado) que en lenguajes de bajo nivel, muy cercanos al lenguaje ensamblador (c++ por ejemplo quedaría a mitad, ni de alto nivel ni de bajo).

Las típicas utilidades que el SO proporcionan son:

1. El desarollo de programas (editories de texto o compiladores)
2. La ejecución de programas

3. El acceso a dispositivos de E/S (como decíamos antes son intrucciones privilegiadas y sin el SO no podemos realizarlas)
4. El acceso al sistema (por ejemplo el acceso a la shell o la interfac gráfica)
5. La detección y respuesta a errores
6. Contabilidad (como rendimiento del sistema)

SO visto como administrador de recursos

Es claro que un computador es un administrador de recursos, y para gestionar tal cantidad de datos y tráfico de instrucciones necesitamos a alguien. ¿Quién será ese alguien? Obviamente el Sistema Operativo. El mecanismo que sigue para realizar esa gestión cubre dos aspectos:

- Las funciones del SO actúan de la misma forma que el resto del software, es decir, son programas ejecutados por el procesador.
- El SO frecuentemente cede el control y depende del procesador para volver a retomarlo.

Podemos sacar varias conclusiones de esto. En primer lugar, el SO dirige al procesador a la hora tanto de usar los recursos del sistema como del tiempo que dedica a la ejecución de otros programas. En segundo lugar, sacamos que la asignación de memoria principal (para poder ejecutar programas necesitamos darle un espacio para datos en la MM, de eso estamos hablando) la realizan conjuntamente el hardware de gestión de memoria del procesador y el SO. Por último, el que decide cuándo se usan los dispositivos de E/S y los ficheros es el SO.

Al final, el procesador es otro recurso más que tiene que gestionar el Sistema Operativo.

Características deseables en un Sistema Operativo

- Comodidad en el uso del computador.
- Eficiencia: Existen más programas que recursos. Hay que repartir los recursos entre los programas.
- Facilidad de Evolución: Un SO importante debe evolucionar en el tiempo por las siguientes razones:
    - Actualizaciones del hardware y nuevos tipos de hardware.
    - Mejorar y/o aportar nuevos servicios.
    - Resolución de fallos

4. Utilizades del Sistema

Se trata de un conjunto de programas de servicio que, en cierta medida, pueden considerarse como una ampliación del SO (Compactación de discos, compresión de datos, gestión de comunicaciones...). Su misión es facilitar la construcción de las aplicaciones de los usuarios, sea cual sea la naturaleza de éstas, tales como editores de texto, compiladores, enlazadores...