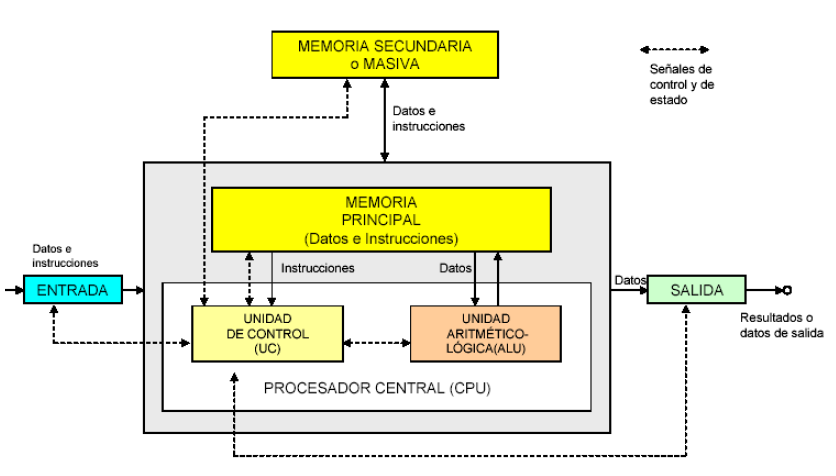
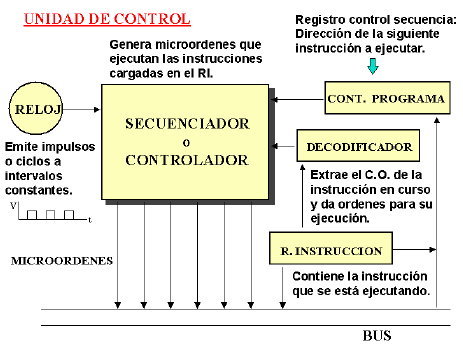
# Unidades funcionales del PC:

Esquema básico de los componentes del PC:

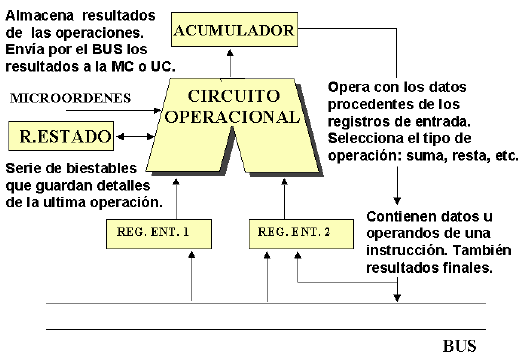


**CPU:** se compone principalmente de:

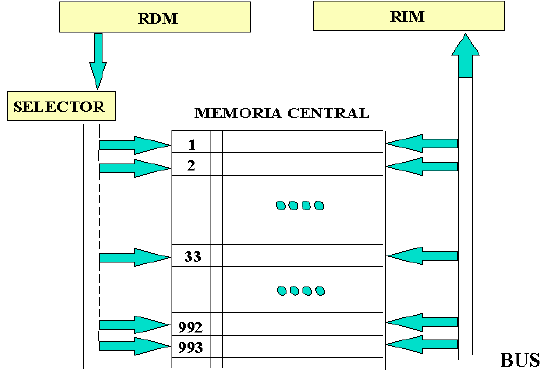
**-Unidad de control:** encargada de buscar e interpretar las instrucciones.



**-Unidad Aritmético Lógica:** se encarga de realizar todas las operaciones de cálculo.



**Memoria Principal (RAM):** es la unidad donde se almacenan todas las instrucciones y datos necesarios en ejecución de un proceso. Se divide en:



**-Selector de memoria:** posibilita la transferencia de datos entre el RDM y el RIM en ambos sentidos, ya sea para realizar una operación de Entrada o de Salida

**-RDM(Registro de Dirección de Memoria):** es la dirección de la celda donde se va a leer/escribir datos.

**-RIM(Registro de Intercambio de Memoria):** si la operación es de lectura: recibe el dato señalado por el RDM; y si es de escritura: almacena la información que hay que grabar procedente de cualquier unidad funcional.

**Bus del sistema:** es un conjunto de circuitos integrados que se encargan de la conexión y comunicación entre la CPU y el resto de unidades de la máquina, se divide en varias líneas eléctricas para permitir la transmisión en paralelo. Las líneas de un bus pueden ser de uno de los siguientes tipos:

**-Líneas de datos:** son los “caminos” físicos por donde se transmiten los datos.

**-Líneas de dirección:** se emplean para seleccionar la fuente o destino de la información que hay sobre el bus de datos, determina la capacidad de direccionamiento segmentando y codificando la dirección. Solo transfieren direcciones.

**-Líneas de control:** gestionan el uso y acceso a los buses de datos y de dirección; se encargan de transmitir órdenes o información de temporización entre los módulos del sistema.

# Instrucciones:

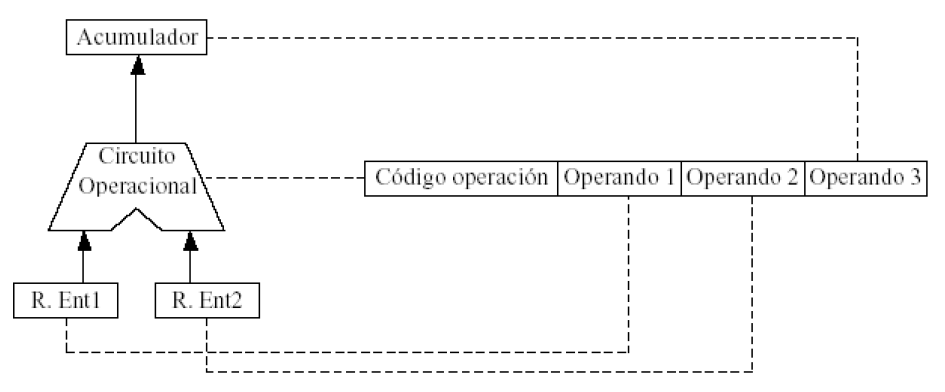
Se componen de:

**-Código de Operación(CO):** indica la operación que se realizará.

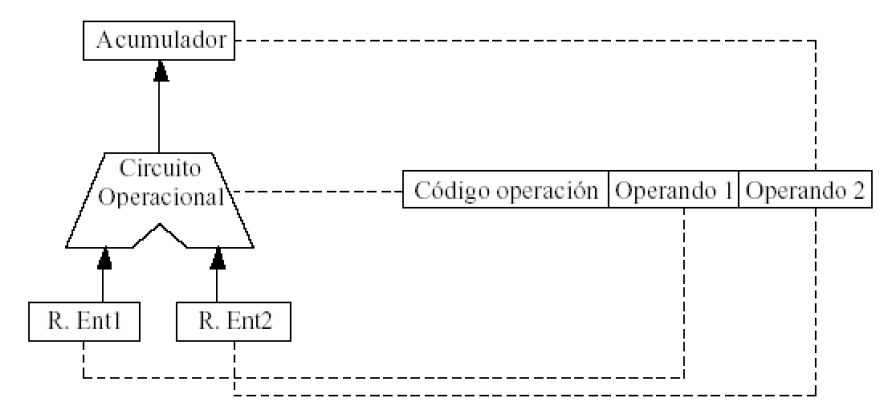
**-Operandos:** Los valores que se usarán para realizar la operación.

En función de si las clasificamos según su formato y el número de operandos, nos encontramos los siguientes tipos:

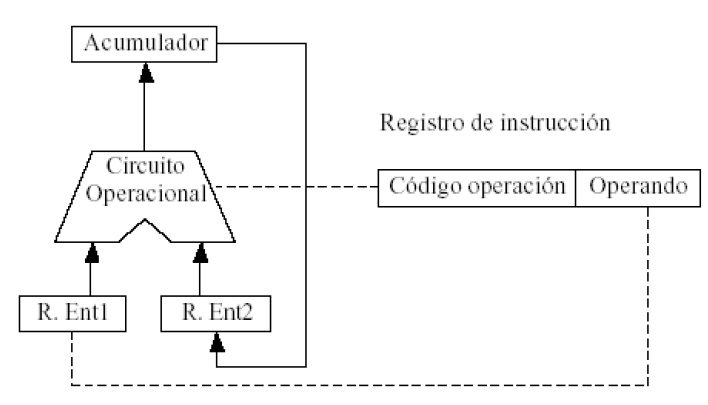
**-Instrucciones de 3 operandos (I+O+O+O):**



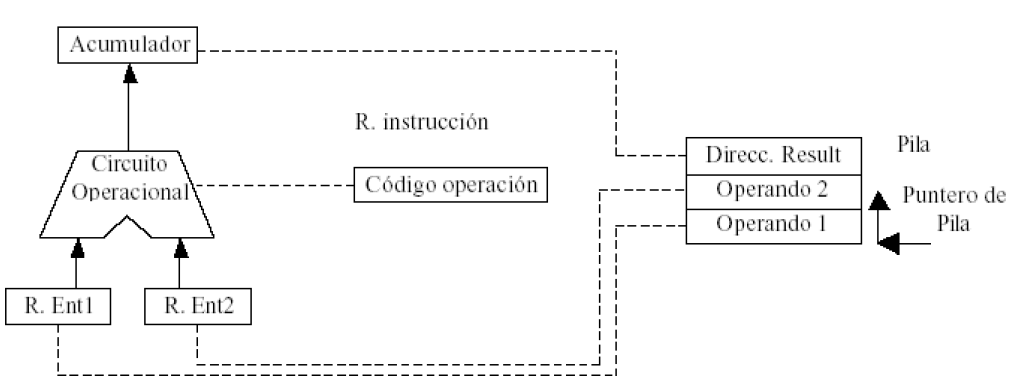
**-Instrucciones de 2 operandos (I+O+O):**



**-Instrucciones de un operando (I+O):**



**-Instrucciones sin operandos (I):**



# Métodos de direccionamiento:

Un modo de direccionamiento de una instrucción es el modo que se utiliza en la misma para indicar la posición de memoria en qué está el dato o datos que intervienen en la instrucción.

**-Direccionamiento inmediato:** la instrucción contienen el dato que hay que emplear, no siendo necesario el acceso a memoria.

**-Direccionamiento directo:** la instrucción contiene la dirección de memoria central donde se encuentra el dato.

**-Direccionamiento indirecto:** la instrucción no es la del dato implicado sino la de una posición de memoria que contiene la dirección de ese dato.

**-Direccionamiento relativo:** la dirección es calculada, por lo que no se encuentra en ningún lugar. La dirección se obtiene sumando la dirección contenida en la propia instrucción con una magnitud fija contenida en un registro especial.

# Procesos:

Es un programa en ejecución, una instancia de un fichero ejecutable. Tiene un tiempo de ejecución y datos de proceso.

Un proceso **requiere**:

**-Su propio espacio de direcciones de memoria interna**. (Resevado por el S.O al arrancar el programa)

**-Valor de contador de programa**, que indica donde va el programa. Guarda la dirección de memoria de la siguiente instrucción a ejecutar

**-El estado del procesador:** es el conjunto de valores de los distintos registros del procesador.

**-Estados de un proceso:**

Desde que se ejecuta un proceso hasta que finaliza, pasa por diferentes estados, producido por la intervención del S.O:

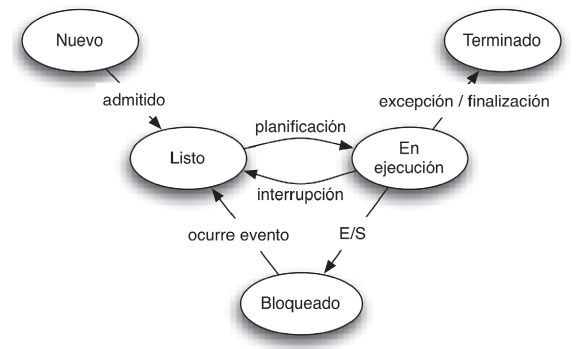
**-Nuevo:** está siendo creado a partir del fichero ejecutable. Solo se encuentra en este estado 1 vez.

**-Listo:** el proceso aún no se encuentra en ejecución, pero está preparado para hacerlo, es decir, el S.O. no le ha asignado ningún procesador para ejecutarlo. El planificador del S.O se encarga de seleccionar que proceso está en ejecución, por lo que indica cuando pasa a ejecución. Ya está cargado en RAM.

**-En ejecución:** en este instante la única instrucción en el registro de instrucción es la de ese proceso. El S.O utiliza el mecanismo de interrupciones para controlar su ejecución. Si el proceso demanda un recurso hará la correspondiente llamada al sistema.

**-Bloqueado:** el proceso está bloqueado esperando a que ocurra algún suceso. Cuando ocurre el evento que lo desbloquea, no pasa directamente a ejecución, sino que tiene que estar planificado nuevamente por el sistema, por lo que estará en la cola. En este estado puede encontrarse en más de una ocasión. El proceso solo se bloquea desde el estado listo y en ejecución.

**-Terminado:** el proceso finaliza su ejecución y libera su imagen de la memoria. El proceso puede terminar debido a que él mismo hace una llamada al sistema para finalizarse o el propio sistema puede finalizarlo mediante una excepción (interrupción especial). Solo puede estar en este estado 1 vez.



Este esquema representa los diferentes estados de un proceso, y la forma en la que éste puede cambiar de estado.

**-Mecanismos de interrupción de procesos:**

-Agota su cuánto de ejecución y el S.O lo interrumpe-->Pasa a Listo.

-Autointerrupción de error. -->Pasa a estado bloqueado.

-Se autointerrumpe al demandar un recurso--> Pasa a estado bloqueado.

De estado bloqueado pasa a estado listo, no puede directamente pasar a ejecución debido a que no lo replanificó de nuevo el sistema.

El S.O se encarga de controlar la ejecución del único proceso en ejecución en ese instante, y planificar la ejecución de todos los procesos.

Llega al estado terminado porque se finaliza o se lanza una excepción porque el S.O lo finaliza o porque su proceso padre lo finaliza.

**-Colas de procesos:**

**-Cola de procesos:** contiene todos los procesos de sistema.

**-Cola de procesos preparados:** contiene el identificador de todos los procesos que están en estado "Listo".

**-Colas de dispositivo:** son las colas en las que se encuentran los procesos en estado bloqueado que están a la espera de la demanda de alguna operación de E/S.

**PLANIFICACION DE PROCESOS:**

**-A corto plazo:** El S.O a corto plazo selecciona qué proceso de la cola de preparados pasa a ejecución.

**-Planificación sin desalojo o cooperativa:** solo se cambia de proceso en ejecución si dicho proceso se bloquea o termina.

**-APROPIATIVA:** además de lo anterior si se cambia el proceso en ejecución si en cualquier momento en que un proceso se está ejecutando, otro proceso tiene mayor prioridad.

**-Tiempo compartido:** se desaloja el proceso en ejecución porque finaliza su cuanto de ejecución .En este caso las prioridades se consideran iguales.

Estas planificaciones funcionan de forma que se complementan entre las 3.

**-A largo plazo:** selecciona qué procesos nuevos deben pasar a la cola de procesos preparados. Controla el nº de procesos que están en la cola de y por lo tanto, en memoria interna (es decir, controla el grado de multiprogramación).

**CAMBIO DE CONTEXTO:**

-Cuando el procesador pasa a ejecutar otro proceso, el sistema debe guardar el contexto del proceso actual y restaurar el contexto del proceso que el planificador a corto plazo ha elegido ejecutar. El contexto se guarda en el momento que se produce una interrupción.

Se conoce como **contexto** a:

**-Estado del proceso.**

**-Estado del procesador**, es decir, los valores de los diferentes registros del procesador.

**-Información de gestión de memoria:** espacio de memoria reservada para el proceso.

El **cambio de contexto** es tiempo perdido, pues el procesador no hace un trabajo útil durante ese tiempo. Ese tiempo es solo útil para permitir la multiprogramación y su duración depende de la arquitectura del procesador.

**-ÁRBOL DE PROCESOS:**

Un proceso se crea siempre a petición de otro proceso, aunque el S.O sea el responsable del proceso de creación por ser el único que puede acceder a los recursos hardware.

Debido a que un proceso se crea petición de otro proceso, se crea un árbol de procesos porque se establece un vínculo entre un proceso padre (el que hace la petición de crear otro proceso) y su/s procesos hijos.

Para identificar los procesos, los S.O´s suelen utilizar un identificador de proceso **(PID)**, es un número entero único en el sistema. La utilización del PID es básica para gestionar procesos, ya que es la forma que tiene el sistema para referirse a los procesos que gestiona.

La comunicación entre procesos se establece a través de los **buffers de comunicación.**

Cuando un proceso padre arranca un proceso hijo a la operación se la denomina "CREATE"; si el proceso padre necesita esperar hasta que el proceso hijo termine su ejecución para poder continuar la suya con los resultados obtenidos por el hijo, lo hace mediante la operación "WAIT" que se debe diferenciar de la operación "SLEEP" que bloquea el proceso hijo y no el padre. Y si es necesario terminar un proceso, aunque lo normal es que se termine a sí mismo mediante la operación "EXIT", el proceso padre lo termina mediante la operación "DESTROY".

Como entre procesos no se ven, para poder intercambiar información necesitan compartir recursos que les permita intercambiar dicha información. Estos recursos pueden ser tanto ficheros abiertos como espacios de memoria compartida. La memoria compartida es una región de memoria a la que pueden acceder distintos procesos cooperativos para compartir la información. Los procesos se comunican escribiendo y leyendo datos de dicha región. El S.O solo interviene a la hora de crear y establecer los permisos de qué procesos pueden acceder a dicha zona. Y así se establecen los buffers o streams de comunicación entre procesos.

Al terminar la ejecución de un proceso, es necesario avisar al S.O de su terminación, para que así éste pueda liberar el espacio de memoria asignado al proceso.

El proceso que ejecuta un metodo start() correctamente se convierte en proceso padre, que puede arrancar más procesos hijos.

En la mayoría de los S.O los procesos hijo tienen sus buffers (stdin,stdout,stderr) son como una copia de los buffers estándar del padre.

Los procesos padre e hijo/s son procesos cooperativos, porque cooperan en la realización de una determinada tarea.

Los procesos cooperativos (cuando un proceso (que se convierte en padre) arranca uno o varios procesos hijo) necesitan sincronizarse, para ello, lo hacen a través del establecimiento de la comunicación entre los procesos(abriendo flujos de entrada/salida).

**-CREACIÓN DE PROCESOS:**

**-Process ProcessBuilder.start():** inicia un nuevo proceso utilizando los atributos indicados en el objeto. El nuevo proceso ejecuta el comando y los argumentos indicados en el método command(), ejecutándose en el directorio de trabajo especificado por el método directory().

Al arrancar un proceso pueden surgir diversos problemas:

-No encuentra el ejecutable debido a la ruta indicada.

-No tener permisos de ejecución

-No ser un ejecutable válido en el sistema.

-Etc

**-COMUNICACIÓN DE PROCESOS:**

Dentro del espacio de direcciones de cada proceso, todos tienen por defecto definidos estos 3 flujos de datos.

**-Entrada estándar(stdin):** lugar donde el proceso lee los datos de entrada que requiere para su ejecución. No se refiere a los parámetros de ejecución del programa. Por defecto suele ser el teclado, pero podría recibirlos de un fichero, de la tarjeta de red o hasta a la salida estándar de otro proceso.

La lectura de datos a lo largo de un programa leerá los datos de su entrada estándar.

**-La salida estándar(stdout):** sitio donde el proceso escribe los datos que obtiene. Por defecto por pantalla, pero puede ser por impresora, o hasta la entrada estándar de otro proceso que necesite esos datos como datos de entrada.

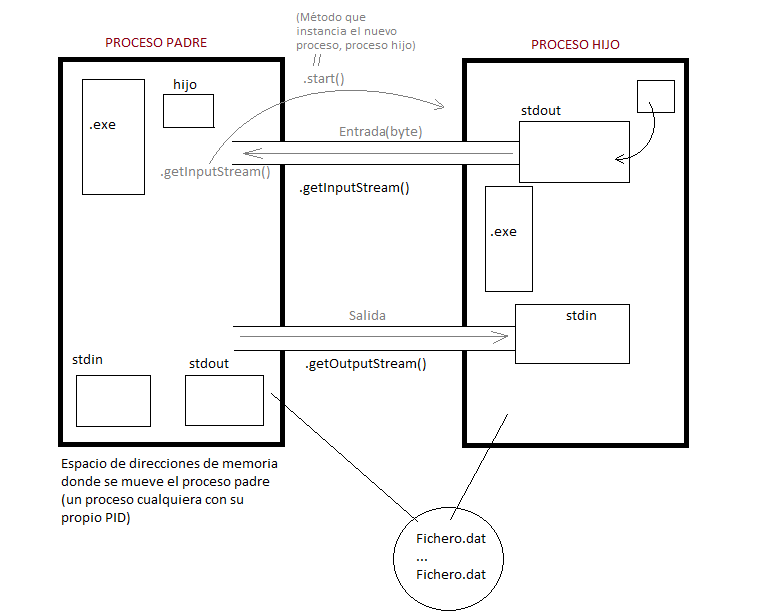
**-La salida de error(stderr):** sitio donde el proceso envía los mensajes de error. Por defecto conectada a la misma que la salida estándar (pantalla), pero también puede un fichero.

Todas las salidas y entradas de información se redirigen al proceso padre a través de los siguientes flujos o streams:

**-OutputStream:** flujo de bytes de salida del proceso hijo, está conectado al stdin(entrada estándar) del proceso hijo.

**-InputStream:** devuelve(al padre) un flujo de bytes de entrada, que está conectado al stdout(salida estándar) del hijo.

**-ErrorStream:** devuelve al padre un flujo de bytes de salida. El stream está conectado por un pipe a la stdout(salida estándar)del proceso hijo.



**Esquema de comunicación entre procesos.**

Cuando se arranca un nuevo proceso hijo, éste arranca "mudo", es decir, no tiene asociados los buffers de ENTRADA/SALIDA.

**CLASE PROCESS:**

**-getInputStream():** devuelve un flujo de bytes de salida (al padre), está conectado al stdin(entrada estándar) del proceso hijo.(Sólo es unidireccional)

**-getOutputStream():** devuelve(al padre) un flujo de bytes de entrada, que está conectado al stdout(salida estándar) del hijo. (Sólo es unidireccional)

**-getErrorStream():** devuelve al padre un flujo de bytes de salida. El stream está conectado por un pipe a la stdout(salida estándar)del proceso hijo.(Sólo es unidireccional)

**-destroy():** para finalizar el proceso hijo.

**-waitFor():** bloquea el padre hasta que obtiene el retorno del proceso hijo.

**-exitValue():** obtiene el valor de retorno del proceso hijo (el valor del "Exit" que se le da en el hijo).

# Otros conceptos:

**Código fuente:** (TEXTO, ASCII). Contiene el algoritmo de resolución de una tarea. Para pasar de código fuente a código objeto es necesario un software encargado de traducirlo. Está escrito en un determinado lenguaje de programación que contiene el algoritmo de resolución de una tarea. Tipos de SW traductores: compilador, intérprete; ambos traducen el código fuente a lenguaje máquina.

**Código objeto:** es la traducción del código fuente a lenguaje máquina interno.

**Código ejecutable:** es un fichero que contiene todo lo necesario para crear un proceso.

**Ejecutable:** contiene la información necesaria para crear un proceso a partir de los datos almacenados de un programa.

**Palabra de memoria:** cantidad de información que puede se puede leer o escribir de la memoria central de una sola vez. (16, 32 o 64 Bits)

**Desbordamiento de pila:** cuando las instrucciones a pasar a la pila sobrepasa el límite del espacio de la propia pila.

**Demonio:** es un proceso que está siempre en ejecución, no es interactivo, está "por debajo" realizando una tarea concreta, normalmente prestando un servicio para otros procesos que lo requieren.

**Kernel(núcleo del S.O):** funciona en base a las interrupciones(suspensión temporal de la ejecución de un proceso), de forma que llama al Kernel, que en función del tipo de interrupción ejecuta determinada rutina. Cuando el Kernel está atendiendo a una interrupción deshabilita la llegada de más interrupciones, debido a que en cada instante solo se puede atender a una. Cuando la ejecución de la rutina termina, se reanuda la ejecución del proceso en el mismo instante que estaba cuando fue interrumpido. Al Kernel no se le puede denominar como demonio, pues no ofrece un servicio cuando lo demanda el proceso, sino que solo se le llama en base a interrupciones que se producen en los procesos.

# Programación concurrente:

Al ejecutar un programa varias veces se están ejecutando concurrentemente distintos procesos en una misma máquina no conectada en red.

Un proceso incluye un espacio de direcciones de memoria interna y son independientes los espacios de memoria interna, gracias a eso podemos tener varios procesos ejecutados concurrentemente. (No simultáneamente, pues cada uno tiene su tiempo de ejecución).

Los procesos se pueden ejecutar concurrentemente porque son totalmente autónomos, los espacios de direcciones no permiten que se vean los procesos entre sí.

En este caso tenemos un único procesador (con una sola unidad de control): solo puede haber un proceso ejecutado en cada instante, pero el S.O se encarga de planificar los distintos procesos, asignándoles a cada proceso un tiempo de ejecución.

El tiempo de UCP se "comparte" entre los distintos procesos, denominado "TIEMPO COMPARTIDO".

Multiprogramación o tiempo compartido: se consigue porque el S.O planifica la ejecución de los distintos procesos.

Cuánto de ejecución: es la cantidad máxima de tiempo que un proceso está ocupando la UCP.

El programa es cargado en RAM, y las instrucciones del código se pasan a la Unidad de control.

El S.O le asigna a cada proceso un cuánto de ejecución, cuando un proceso consume su cuánto de ejecución es detenido por el S.O.

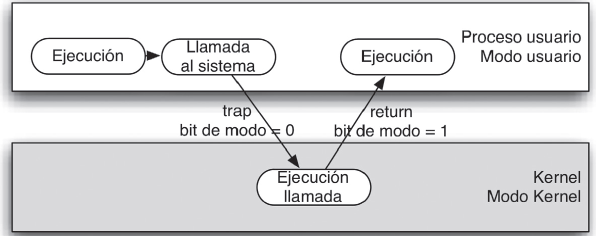
En caso de tener **un procesador con varios núcleos**: cada núcleo atiende a una tarea, refiriéndose a tareas del mismo proceso, dividiendo así el proceso en diferentes hilos de ejecución, haciéndolo más rápido.

Permite programación multitarea(tarea=hilo de ejecución): simultáneamente se ejecutan varios procesos.

**Modo dual:** característica del Hardware que permite al Kernel "protegerse", tiene 2 modos de funcionamiento identificados por 1bit.

-Modo usuario: cuando la demanda es por parte de un programa de usuario.

-Modo Kernel: cuando se produce la interrupción y ejecuta la rutina correspondiente.



Es decir, cuando se está ejecutando el proceso, está activado el modo usuario, cuando se produce la interrupción y se hace la llamada al sistema, cambia a modo Kernel, ejecuta la rutina y vuelve al modo usuario.