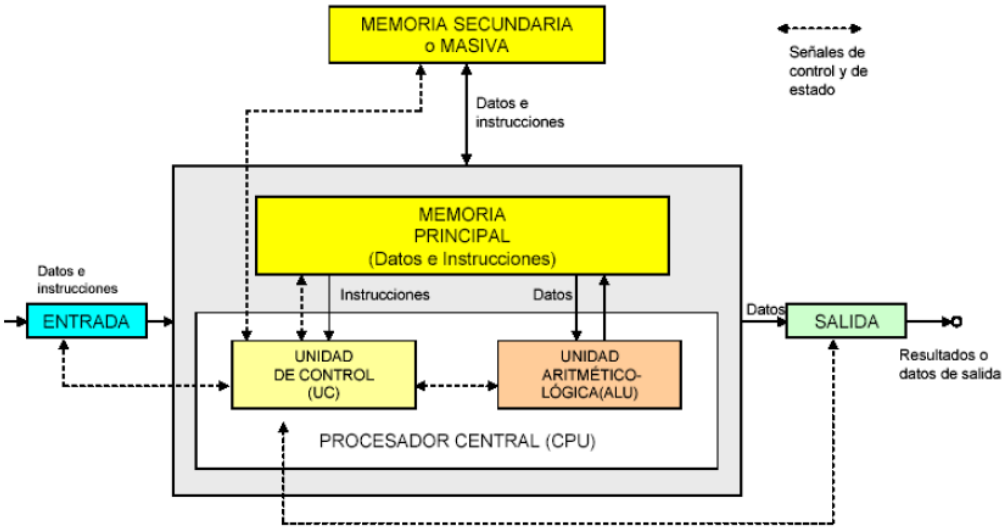
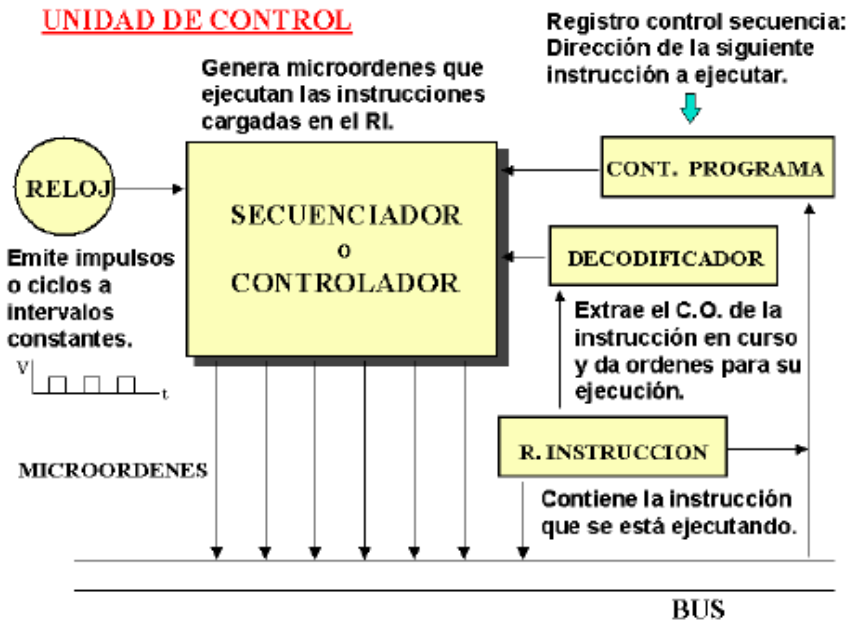
Unidades funcionales del PC:

Esquema básico de los componentes del PC:

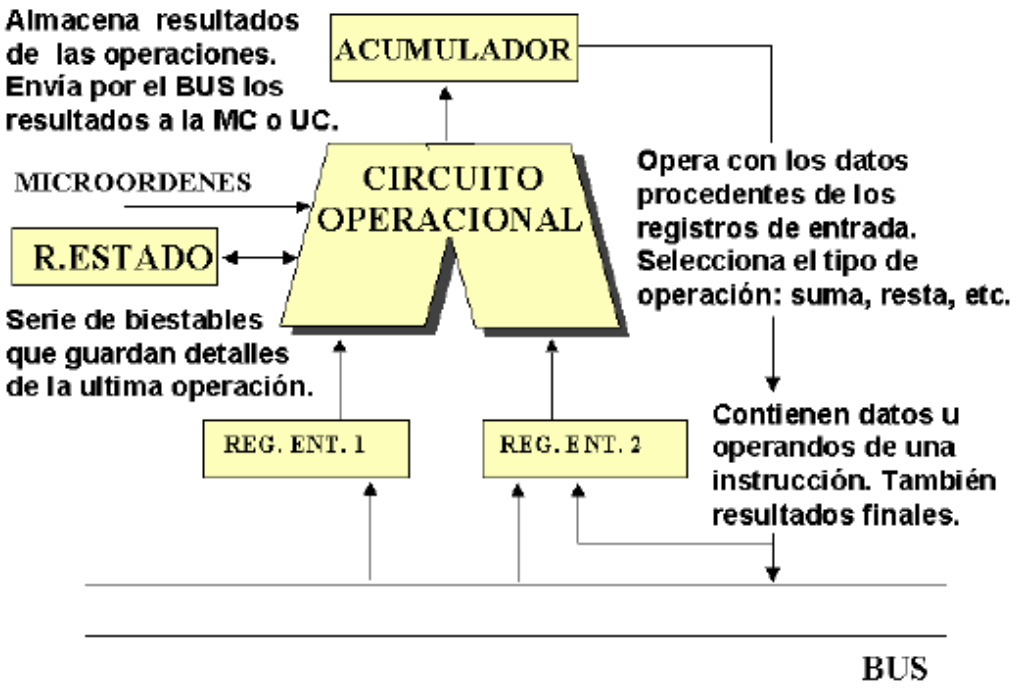


CPU:

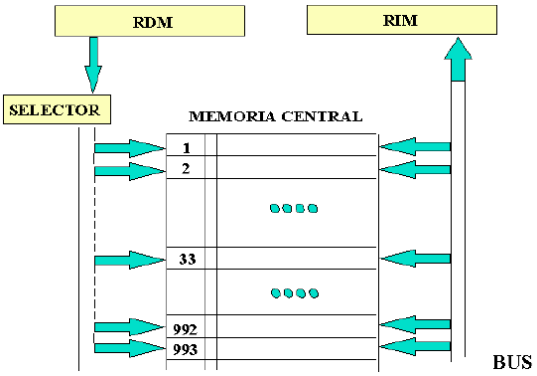
-Unidad de control: contiene la instrucción que se está ejecutando en ese instante, es la que está siendo interpretada o decodificada. Su ejecución está siendo "atendida por el Kernel"



-Unidad Aritmético Lógica:



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



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

-RDM (Registro de Dirección de Memoria): es la dirección de la celda o posición a partir de la que 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:

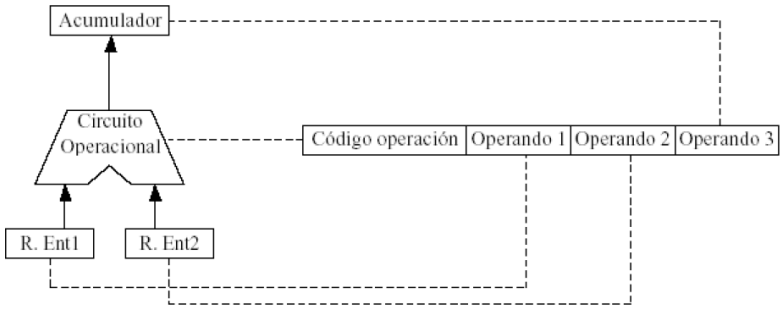
Se componen de:

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

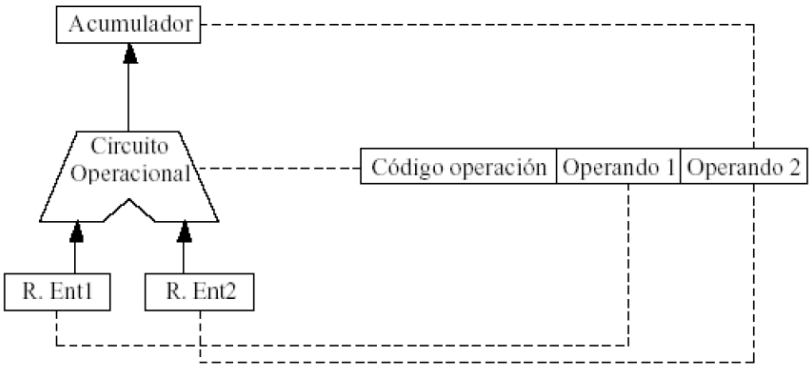
-Operandos: Los valores que se usarán para realizar la operación.

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

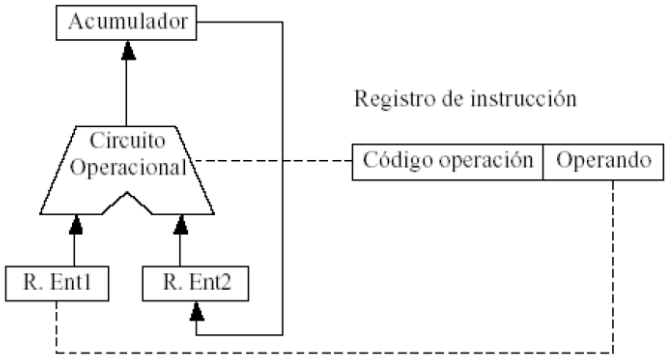
-Instrucciones de 3 operandos (I+O+O+O):



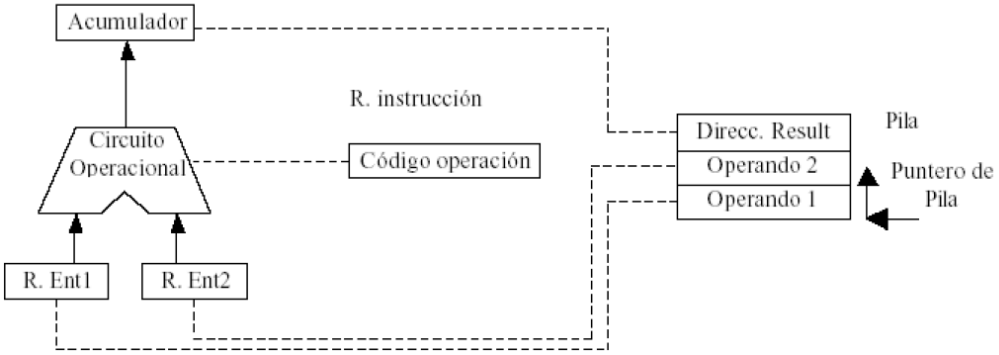
-Instrucciones de 2 operandos (I+O+O):



-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 contiene la dirección del dato implicado sino la de una posición de memoria que contiene la dirección de ese dato.

-Direccionamiento relativo: la dirección es calculada. La dirección se obtiene sumando la dirección contenida en la propia instrucción con una magnitud fija contenida en un registro especial. (Magnitud fija nos sitúa en el espacio de direcciones de memoria del proceso.)

Proceso:

Es un programa en ejecución, una instancia de un fichero ejecutable.

Un proceso implica:

- **Su imagen de memoria, su espacio de direcciones de memoria donde se va a ejecutar, este bloque o espacio de memoria es independiente de todas las demás espacios de los distintos procesos. Cada proceso posee o tiene asignado su propio espacio de direcciones de memoria independiente.**
- **Estado del proceso :**

Conjunto de valores de los registros de la unidad de control y de la Unidad Aritmética Lógica

Valores de: registro de instrucción, registro contador de programa, registros en la U.A.L.

- **Momento del proceso podemos decir que es el VALOR DE EL REGISTRO CONTADOR DE PROGRAMA, contiene la dirección de memoria de la siguiente instrucción a ejecutarse.**

Programa vs Proceso.

Un programa es un fichero BINARIO EJECUTABLE en disco.

Todo proceso es creado por el Sistema Operativo, la orden de instanciar un ejecutable es atendida por el planificador de procesos del sistema operativo. Por eso, todos los procesos existentes en un instante dado en un equipo **constituyen lógicamente un árbol de procesos**, donde el nodo raíz es el sistema, proceso 0. La parte del planificador de procesos encargada de crear los procesos se corresponde un con una política de planificación a largo plazo.

Un proceso comporta o supone la existencia de:

-Su propio espacio de direcciones de memoria interna. Cada proceso tiene su propia **imagen de memoria .Por eso podemos definir proceso como el espacio de direcciones de memoria donde se está ejecutando un fichero ejecutable.**

-Valor de contador de programa en un registro de la unidad de control, que indica el momento del proceso. Guarda la dirección de memoria de la siguiente instrucción a ejecutar

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

Para identificar los procesos, el SO utiliza 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.

-Estados de un proceso:

Desde que se da la orden de crear un proceso hasta que este finaliza, el proceso pasa por diferentes estados:

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

-Estado Listo: No hay nada que impida que el proceso esté en ejecución, pero sin embargo, no se encuentra en ejecución y está esperando preparado para hacerlo. El S.O. no le ha asignado ningún procesador para su ejecución. Es el planificador de procesos del S.O el que se encarga de poner un proceso en estado de ejecución.

En un instante dado puede haber n procesos en estado listo. Todos los que están lógicamente en la cola de procesos preparados.

La cola de procesos preparados es tratada como una lista FIFO (FIRST IN FIRST OUT), para procesos con la misma prioridad.

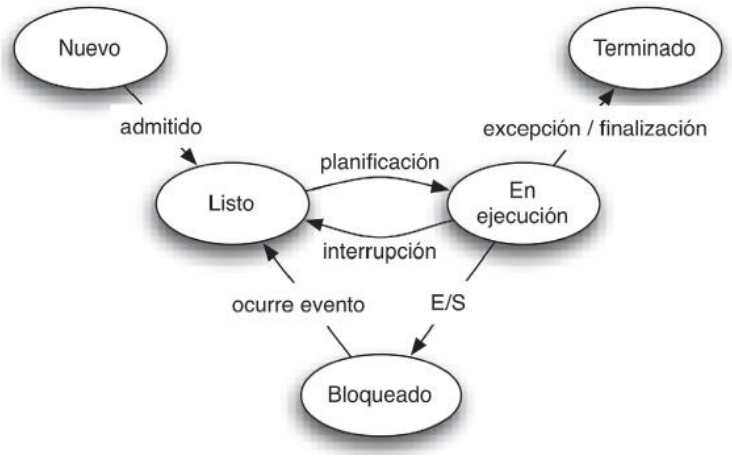
-En ejecución: La única "puerta de entrada a este estado" es estar al comienzo de la cola de procesos preparados. En un instante dado la única instrucción almacenada en el registro de instrucción de la Unidad de Control es **la del único proceso que está estado de en ejecución.** El S.O (Kernel) "atiende, está a la escucha" de la ejecución de la instrucción. El Kernel utiliza un mecanismo de interrupciones para comunicarse con la instrucción en ejecución, de este modo controla y atiende a la misma. Si el proceso demanda un recurso hará la correspondiente llamada al sistema mediante una interrupción. **Decimos que el kernel atiende a la instrucción en curso.** La interfaz que proporciona el kernel para atender a cualquier requerimiento es mediante un mecanismo de interrupciones.

Una interrupción -→ genera una respuesta concreta. Una rutina que la atiende.

- **Estado 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 ser planificado nuevamente por el sistema, por lo que pasará a la cola de procesos preparados. En este estado puede encontrarse en más de una ocasión. El proceso solo se bloquea desde el estado en ejecución.

-Estado Terminado: el proceso finaliza su ejecución, el SO libera su imagen de memoria. El proceso puede terminar debido a que finaliza y él mismo hace una llamada al sistema para finalizarse o el propio sistema puede finalizarlo mediante una excepción (interrupción especial). **Si el proceso forma parte de un árbol de procesos cooperantes,**

también podrá ser finalizado por su proceso padre. 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.

Un proceso deja de estar en estado de ejecución:

-Agota su cuánto de ejecución y el planificador de procesos lo interrumpe--> El proceso pasa a estado Listo.

-El proceso se auto interrumpe al demandar un recurso, una operación de E/S--> El proceso pasa a estado bloqueado.

De estado bloqueado pasará a estado listo cuando suceda el evento que lo desbloquea, no puede pasar directamente de bloqueado a ejecución, el proceso deberá ser planificado de nuevo desde la cola de procesos preparados.

-Aparece en la cola de procesos preparados, es decir en estado listo, un nuevo proceso con mayor prioridad que todos los existentes y que él que está en ejecución, este nuevo proceso se sitúa al comienzo de la cola de procesos preparados y se desaloja el que está en ejecución

-Llega al estado terminado porque el proceso finaliza realizando la operación exit (), se lanza una excepción que hace que el S.O lo finalice, el proceso padre lo finaliza realizando la operación destroy ().

-Colas de procesos:

-Cola de procesos: contiene lógicamente a todos los procesos existentes y que el sistema está planificando y proporcionando su ejecución concurrente.

-Cola de procesos preparados: contiene lógicamente a todos los procesos que están en estado "Listo".

-Colas de dispositivos: 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.

PLANIFICACIÓN DE PROCESOS:

-Políticas de planificación de procesos a corto plazo:

-Planificación sin desalojo o cooperativa: sólo se cambia de proceso en ejecución si dicho proceso se bloquea o termina. El proceso que pasa a estado de ejecución es el que está al principio de la cola de procesos preparados, se restaura su estado en el procesador.

- Planificación APROPIATIVA o con desalojo: sí en cualquier momento en que un proceso se está ejecutando aparece en estado listo otro proceso de mayor prioridad el proceso en ejecución se interrumpe o desaloja.

Esta política que atiende a la prioridad de los procesos trata la cola de procesos preparados como un conjunto de pilas (lista LIFO) Last In first Out

-Tiempo compartido: se desaloja el proceso en ejecución porque finaliza su cuanto de ejecución.

Cuanto de ejecución de un proceso es la cantidad máxima de tiempo que un proceso puede estar en ejecución **ininterrumpidamente.**

Estas políticas de planificación **son procesos que se ejecutan concurrentemente**. La ejecución concurrente de todas ellas, rutinas que constituyen el planificador de procesos, son las que **proporcionan multiprogramación al sistema, es decir proporcionan LA EJECUCIÓN CONCURRENTE DE TODOS LOS PROCESOS.**

--Políticas de planificación de procesos a largo plazo:

A largo plazo, el planificador instancia o crea los nuevos procesos. Selecciona qué procesos nuevos deben pasar a la cola de procesos preparados. **Controla el nº de procesos existentes y por lo tanto controla el grado de multiprogramación.**

CAMBIO DE CONTEXTO:

-Cada vez que el planificador de procesos desaloja un proceso, es decir interrumpe el proceso en ejecución, otro proceso pasa a ejecución. Esto es posible porque el Sistema Operativo **realiza el cambio de contexto.**

El cambio de contexto se hace muy frecuentemente, de modo que la cola de procesos preparados es una “cola en movimiento”

El sistema debe salvaguardar el contexto del proceso actual y restaurar el contexto del proceso que el planificador pone en ejecución.

Un proceso se interrumpe o desaloja y otro proceso se restaura o pasa a ejecución.

Se conoce como **contexto de un proceso** a:

-Estado del proceso, 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 para ningún proceso durante ese tiempo. Ese tiempo es solo útil para permitir la multiprogramación, programación concurrente de varios procesos, y su duración depende de la arquitectura del procesador.

La multiprogramación: varios procesos ejecutándose concurrentemente en un equipo no acorta el tiempo de ejecución de cada proceso.

Su ventaja es Optimizar el uso del procesador del equipo.

-ÁRBOL DE PROCESOS:

Un proceso se crea siempre a petición de otro proceso, el S.O es el responsable de la creación de cualquier proceso, por ser el único que puede acceder a los recursos hardware. Por eso, decimos que es el padre de todos los procesos.

Todos los procesos son nodos hijos del S.O. Nodos de hijo del nodo raíz.

Definimos una **aplicación multiproceso como un árbol n-ario de procesos**, todos los procesos de este árbol se dicen que **son cooperantes**, porque cooperan en la realización de una tarea común.

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

Es una visión lógica desde nuestra aplicación multiproceso, para el sistema operativo todos los procesos de nuestra aplicación multiproceso son procesos independientes como todos los demás en la cola de procesos que tiene que planificar y que compiten entre ellos por uso de la CPU.

Para la realización de esa tarea común, los procesos del mismo árbol de procesos deberán comunicarse, en esa comunicación hay una necesidad de sincronizarse.

En general, todos los procesos que se están ejecutando concurrentemente en un equipo, son procesos independientes, que se ejecutan asincrónicamente y que no se comunican. Es el planificador de procesos del SO el que proporciona esta ejecución (multiprogramación).

Los procesos que forman parte de la misma aplicación multiproceso, es decir forman parte del mismo árbol de procesos, se pueden ejecutar asincrónicamente o sí se tienen que comunicar entonces se ejecutan sincrónicamente.

La comunicación entre procesos padre—hijo es una forma de sincronización.

La comunicación entre procesos del mismo árbol, se establece a través de flujos o streams de bytes **de información. Es tarea del programador realizar un algoritmo correcto que permita la comunicación y sincronización entre proceso padre y sus procesos hijo**

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" 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 los procesos que existen en un instante dado “no se ven”, no pueden acceder al espacio de memoria de otro proceso, 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 proceso que ejecuta el método start() referenciado a otro proceso se convierte en el proceso padre del proceso refernciado.

En la mayoría de los S.O los procesos hijo nacen con sus buffers (stdin, stdout, stderr) 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.

Cuando un proceso arranca uno o varios procesos hijos, se convierte en proceso padre, y habitualmente necesita comunicarse y sincronizarse con ellos, lo hace estableciendo tuberías de comunicación entre padre-hijo, abriendo flujos de entrada/salida.

ProcessBuilder.start() tipo de retorno objeto de la clase Process , el método 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().

.start() devuelve la referencia de un objeto de la clase Process, **el nuevo proceso está en estado listo.**

Al arrancar un proceso pueden surgir diversos problemas:

-No encuentra el ejecutable debido a la ruta indicada.

-No tener permisos de ejecución

-La especificación del fichero no ser un ejecutable válido en el sistema.

-COMUNICACIÓN DE PROCESOS:

Dentro del espacio de direcciones de memoria de cada proceso, por defecto están definido estos 3 buffers de datos.

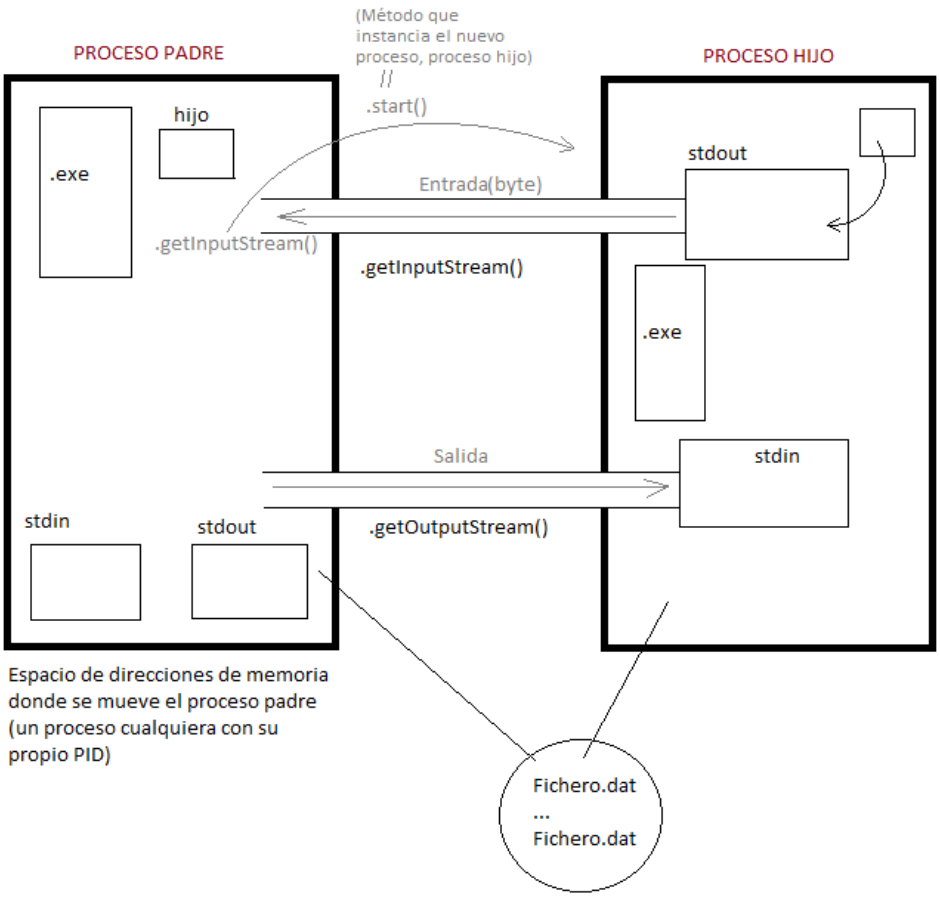
-Buffer de entrada estándar (stdin): buffer (espacio de memoria interna) donde el proceso lee los datos de entrada que requiere para su ejecución. No se refiere a los parámetros de ejecución del programa. Por defecto suele estar asociado al teclado, pero podría recibirlos de un fichero, de la tarjeta de red o hasta a la salida estándar de otro proceso.

El proceso leerá los datos de su entrada estándar.

-Buffer de salida estándar (stdout): espacio de memoria donde el proceso escribe los datos de salida que obtiene. Por defecto está asociado a pantalla, pero puede estar asociado a la impresora, a un fichero en disco, o a la entrada estándar de otro proceso que necesite esos datos como datos de entrada.

El proceso escribirá los datos en su salida estándar

-Buffer de errores (stderr): buffer de salida**,** espacio de memoria donde el proceso envía los mensajes de error. Por defecto conectada a la salida estándar



Esquema de comunicación entre procesos.

.getInputStream() devuelve un flujo de bytes de entrada (al padre), el flujo de bytes está conectado a la salida estándar **(stdout) del proceso hijo referenciado.** Es un flujo **unidireccional**. En este flujo el padre realizará operaciones de lectura para acceder a los datos que el proceso hijo escribe en su salida estándar.

.getOutputStream() devuelve un flujo de bytes de salida (al padre), que está conectado a la entrada estándar **(stdin) del proceso hijo referenciado** . Es un flujo unidireccional. En este flujo el padre realizará operaciones de escritura para enviar información a la entrada estándar del hijo.

.getErrorStream() devuelve al padre un flujo de bytes de entrada. El stream está conectado por un pipe a la stderr (salida estándar de errores) del proceso hijo.

.**destroy()** finaliza el proceso hijo referenciado. El proceso referenciado pasa a estado terminado.

.**waitFor()** bloquea a el padre hasta que el proceso hijo finaliza.

.exitValue() obtiene el valor de retorno del proceso hijo ,el valor de la operación exit que realiza el proceso hijo al finalizar.

Otros conceptos comentados en clase:

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. El código fuente o programa fuente está escrito en un determinado lenguaje de programación y contiene el algoritmo de resolución de una determinada tarea.

Tipos de traductores: compilador, intérprete; ambos traducen el código fuente a lenguaje máquina.

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

Ejecutable: fichero binario que ya contiene toda la información necesaria para crear un proceso

Palabra de memoria: cantidad de información que puede se puede leer o escribir de/en la memoria central de una sola vez, en una sola operación

Desbordamiento de pila: cuando las instrucciones /datos al ser almacenados en la pila (emplilar/ PUSH) 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. Demonio es un proceso que no interactúa con ningún otro proceso.

Kernel (núcleo del S.O): se comunica con la instrucción en curso en base a interrupciones (UNA INTERRUPCIÓN ES suspensión temporal de la ejecución de un proceso), de forma que el proceso se comunica o llama al Kernel, que en función del tipo de interrupción ejecuta una determinada rutina en respuesta. 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 considerar un proceso demonio porque sí interactúa con el proceso en ejecución mediante una interface o mecanismo de interrupciones.

El sistema está en modo Kernel **o modo seguro** cuando la instrucción en ejecución es una instrucción perteneciente a una rutina del Kernel, **indentificada por un bit.**

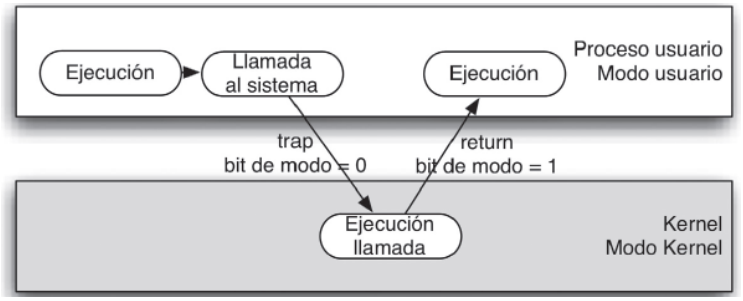
Sólo en modo seguro o modo kernel se podrán realizar determinadas operaciones en el sistema.

Modo dual: característica del Hardware que permite al sistema “protegerse", existen dos **modos de funcionamiento identificados por 1bit.**

-**Modo usuario**: cuando la instrucción en ejecución pertenece a cualquiera de los procesos en ejecución.

-Modo Kernel: cuando se ha producido una interrupción y se está ejecutando la rutina correspondiente a dicha interrupción.

La instrucción que en ese instante está siendo atendida por la UC, (está en el registro de instrucción) pertenece a una rutina del Kernel. 1 bit de de dicha instrucción lo señala **(bit 0, modo protegido o modo kernel)**



Es decir, cuando se está ejecutando un 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

Programación concurrente:

En un equipo se están ejecutando concurrentemente distintos procesos compartiendo tiempo de uso de la unidad de control. Una parte del sistema operativo son un conjunto de rutinas que proporciona multiprogramación, se llaman Planificador de procesos.

Cada proceso tiene su propio espacio de direcciones de memoria interna (imagen de memoria) y es independiente del resto de imágenes de memoria, gracias a eso podemos tener varios procesos que se están ejecutando concurrentemente. **No simultáneamente**, en un instante dado sólo un proceso está siendo atendido o en ejecución.

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

En cada instante solo puede haber un proceso en estado de ejecución, **el S.O se encarga de planificar los distintos procesos, asignándoles a cada proceso un tiempo de ejecución.**

El conjunto de rutinas que constituyen el planificador de procesos también se están ejecutando concurrentemente y conjuntamente nos proporcionan multiprogramación.

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

El programa es cargado en RAM, y **las instrucciones del código ejecutable son cargadas secuencialmente en el registro de instrucción de la Unidad de control, para ser interpretadas (ejecutadas).**

Si disponemos de un **procesador con varios núcleos**: **cada núcleo puede atender a una instrucción** pero del mismo proceso, **del único proceso que está en ejecución**, entonces es posible el paralelismo real o **multitarea** (tarea=hilo de ejecución); **simultáneamente se ejecutan varios hilos. Varias instrucciones una de cada hilo, todos del mismo proceso.**

Esto será objeto de estudio en la Unidad 2.