PROGRAMACION MULTIPROCESO

1.- Introducción: Aplicaciones, Ejecutables y Procesos.
2.- Ejecutables. Tipos.
3.- Multiprogramación o multitarea
4.- Ventajas de la multiprogramación:
5.- Multiprogramación y Multiproceso:
6.- Gestión de procesos.
6.1.- Introducción.
6.2.- Herramientas gráficas para la gestión de procesos.
6.3.- Estados de un Proceso.
6.4.- Planificación de procesos por el Sistema Operativo.
6.5.- Cambio de contexto en la CPU.
6.6.- Servicios. Hilos.
6.7.- Comandos para la gestión de procesos.

1.- Introducción: Aplicaciones, Ejecutables y Procesos.

A simple vista, parece que con los términos aplicación, ejecutable y proceso, nos estamos refiriendo a lo mismo.

Una aplicación es un tipo de programa informático, diseñado como herramienta para resolver de manera automática un problema específico del usuario.

Debemos darnos cuenta de que sobre el hardware del equipo, todo lo que se ejecuta son programas informáticos, que, ya sabemos, que se llama software. Con la definición de aplicación anterior, buscamos diferenciar las aplicaciones, de otro tipo de programas informáticos, como pueden ser: los sistemas operativos, las utilidades para el mantenimiento del sistema, o las herramientas para el desarrollo de software. Por lo tanto, son aplicaciones, aquellos programas que nos permiten editar una imagen, enviar un correo electrónico, navegar en Internet, editar un documento de texto, chatear, etc.

Recordemos, que un programa es el conjunto de instrucciones que ejecutadas en un ordenador realizarán una tarea o ayudarán al usuario a realizarla.

Nosotros, como programadores y programadoras, creamos un programa, escribiendo su código fuente; con ayuda de un compilador, obtenemos su código binario o interpretado. Este código binario o interpretado, lo guardamos en un fichero. Este fichero, es un fichero ejecutable, llamado comúnmente: ejecutable o binario.

Un ejecutable es un fichero que contiene el código binario o interpretado que será ejecutado en un ordenador.

Ya tenemos más clara la diferencia entre aplicación y ejecutable. Ahora, ¿qué es un proceso?

De forma sencilla, un proceso, es un programa en ejecución. Pero, es más que eso, un proceso en el sistema operativo (SO), es una unidad de trabajo completa; y, el SO gestiona los distintos procesos que se encuentren en ejecución en el equipo. En siguientes apartados de esta unidad trataremos más en profundidad todo lo relacionado con los procesos y el SO. Lo más importante, es que diferenciemos que un ejecutables es un fichero y un proceso es una entidad activa, el contenido del ejecutable, ejecutándose.

Un proceso es un programa en ejecución.

Un proceso existe mientras que se esté ejecutando una aplicación. Es más, la ejecución de una aplicación, puede implicar que se arranquen varios procesos en nuestro equipo; y puede estar formada por varios ejecutables y librerías.

Al instalar unaaplicacion en el equipo, podremos ver que puede estar formada por varios ejecutables y librerías. Siempre que lancemos la ejecución de una aplicación, se creará, al menos, un proceso
nuevo en nuestro sistema.

2.- Ejecutables. Tipos.

En sistemas operativos Windows, podemos reconocer un fichero ejecutable, porque su extensión, suele ser .exe. En otros sistemas operativos, por ejemplo, los basados en GNU/Linux, los ficheros ejecutables se identifican como ficheros que tienen activado su permiso de ejecución (y no tienen que tener una extensión determinada).

Según el tipo de código que contenga un ejecutable, los podemos clasificar en:

Binarios. Formados por un conjunto de instrucciones que directamente son ejecutadas por el procesador del ordenador. Este código se obtiene al compilar el código fuente de un programa y se guarda en un fichero ejecutable. Este código sólo se ejecutará correctamente en equipos cuya plataforma sea compatible con aquella para la que ha sido compilado (no es multiplataforma). Ejemplos son, ficheros que obtenemos al compilar un ejecutable de C o C++.

Interpretados. Código que suele tratarse como un ejecutable, pero no es código binario, sino otro tipo de código, que en Java, por ejemplo se llama bytecode. Está formado por códigos de operación que tomará el intérprete (en el caso de Java, el

intérprete es la máquina virtual Java o JRE). Ese intérprete será el encargado de traducirlos al lenguaje máquina que ejecutará el procesador. El código interpretado es más susceptible de ser multiplataforma o independiente de la máquina física en la que se haya compilado.

Un tipo especial de ejecutables interpretados, son los llamados scripts. Estos ficheros, contienen las instrucciones que serán ejecutadas una detrás de otra por el intérprete. Se diferencian de otros lenguajes interpretados porque no son compilados. Por lo que los podremos abrir y ver el código que contienen con un editor de texto plano (cosa que no pasa con los binarios e interpretados compilados). Los intérpretes de este tipo de lenguajes se suelen llamar motores. Ejemplos de lenguajes de script son: JavaScript, php, JSP, ASP, python, ficheros .BAT en MS-DOS, Powershell en Windows, bash scripts en GNU/Linux,

Librerías. Conjunto de funciones que permiten dar modularidad y reusabilidad a nuestros programas. Las hemos incluido en esta clasificación, porque su contenido es código ejecutable, aunque ese código sea ejecutado por todos los programas que invoquen las funciones que contienen. El conjunto de funciones que incorpora una librería suele ser altamente reutilizable y útil para los programadores; evitando que tengan que reescribir una y otra vez el código que realiza la misma tarea.

Ejemplo de librerías son: las librerías estándar de C, los paquetes compilados DLL en Windows; las API (Interfaz de Programación de Aplicaciones), como la J2EE de Java (Plataforma Java Enterprise Edition versión 2); las librerías que incorpora el framework de .NET; etc.

3.- Multiprogramación o multitarea

M**ultiprogramación** o **multitarea** es la técnica que permite que dos o más procesos ocupen la misma unidad de memoria principal y que sean ejecutados al "mismo tiempo" (seudo-paralelismo, ya que en una única CPU sólo puede haber un proceso a la vez) en la unidad central de proceso o CPU, es decir, permite la ejecución de varios procesos al mismo tiempo corriendo sobre un único procesador.

4.- Ventajas de la multiprogramación:

Permite la existencia de varios procesos en ejecución.

Permite el servicio interactivo simultáneo a varios usuarios de manera eficiente.

Aprovecha los tiempos que los procesos pasan esperando a que se completen sus operaciones de E/S, por lo que aumenta el uso de la CPU.

Las direcciones de los procesos son relativas, el programador no se preocupa por saber en dónde estar á el proceso dado que el sistema operativo es el que se encarga de convertir la dirección lógica en física.

5.- Multiprogramación y Multiproceso:

Multiprogramación es distinto a multiproceso.

El multiproceso implica la existencia de varios procesadores en el mismo sistema, ejecutando por ejemplo simultáneamente diferentes trabajos, pudiendo ser usados en multiprogramación.

En cambio un sistema de multiprogramación no podría ser utilizado en multiproceso si sólo tuviera un procesador.

El grado de multiprogramación viene determinado por el número de procesos activos.

La ejecución aparentemente simultánea de varios procesos en un mismo sistema, requiere repartir el tiempo de CPU entre los objetos a ejecutar. Eso implica expulsar de la CPU, al proceso en ejecución y asignarla a un proceso preparado.

Esta actividad se conoce como cambio de contexto.

Los cambios de contexto no son trabajo útil e implican una sobrecarga importante si se hacen con frecuencia reducen la utilización.

Tipos de multitarea :

Multitarea Nula: es aquel SO que carece de multitarea, por ejemplo MS-DOS.

Multitarea Corporativa, Cooperativa o No apropiativa: es aquella donde los procesos de usuario son los que ceden la CPU al SO a intervalos regulares. Es el tipo de multitarea de los SO Windows anteriores a 1995 como Windows 3.11.

Multitarea Preferente o Apropiativa: es aquella en donde el SO se encarga de administrar el/los procesador/es, repartiendo el tiempo de uso del mismo entre los distintos procesos que esperan utilizarlo. Si hay un solo procesador, cada proceso lo utiliza en periodos cortísimos de tiempo, dando la sensación de que se ejecutan al mismo tiempo. Es el tipo de multitarea de los SO UNIX (y sus clones Linux), Windows NT, etc.

Multitarea Real: es aquella en donde el SO ejecuta los procesos realmente al mismo tiempo, haciendo uso de múltiples procesadores. En el caso de que los procesos o tareas sean más que la cantidad de procesadores, éstos comienzan a ejecutarse como en la multitarea preferente. Es el tipo de multitarea de los SO modernos. En este caso tenemos un sistema multiproceso.

6.- Gestión de procesos.

Como sabemos, en nuestro equipo, se están ejecutando al mismo tiempo, muchos procesos. Por ejemplo, podemos estar escuchando música con nuestro reproductor multimedia favorito; al mismo tiempo, estamos programando con Eclipse; tenemos el navegador web abierto, para ver los contenidos de esta unidad; incluso, tenemos abierto el Messenger para chatear con nuestros amigos y amigas.

Independientemente de que el microprocesador de nuestro equipo sea más o menos moderno (con uno o varios núcleos de procesamiento), lo que nos interesa es que actualmente, nuestros SO son multitarea; como son, por ejemplo, Windows y GNU/Linux.

Ser multitarea es, precisamente, permitir que varios procesos puedan ejecutarse al mismo tiempo, haciendo que todos ellos compartan el núcleo o núcleos del procesador.

Pero, ¿cómo?

Imaginemos que nuestro equipo, es como nosotros mismos cuando tenemos más de una tarea que realizar. Podemos, ir realizando cada tarea una detrás de otra, o, por el contrario, ir realizando un poco de cada tarea. Al final, tendremos realizadas todas las tareas, pero para otra persona que nos esté mirando desde fuera, le parecerá que, de la primera forma, vamos muy lentos (y más, si está esperando el resultado de una de las tareas que tenemos que realizar); sin embargo, de la segunda forma, le parecerá que estamos muy ocupados, pero que poco a poco estamos haciendo lo que nos ha pedido.

Pues bien, el micro, es nuestro cuerpo, y el SO es el encargado de decidir, por medio de la gestión de procesos, si lo hacemos todo de golpe, o una tarea detrás de otra.

En este punto, es interesante que hagamos una pequeña clasificación de los tipos de procesos que se ejecutan en el sistema:

Por lotes. Están formados por una serie de tareas, de las que el usuario sólo está interesado en el resultado final. El usuario, sólo introduce las tareas y los datos iniciales, deja que se realice todo el proceso y luego recoge los resultados. Por ejemplo: enviar a imprimir varios documentos, escanear nuestro equipo en busca de virus,...

Interactivos. Aquellas tareas en las que el proceso interactúa continuamente con el usuario y actúa de acuerdo a las acciones que éste realiza, o a los datos que

suministra. Por ejemplo: un procesador de textos; una aplicación formada por formularios que permiten introducir datos en una base de datos; ...

Tiempo real. Tareas en las que es crítico el tiempo de respuesta del sistema. Por ejemplo: el ordenador de a bordo de un automóvil, reaccionará ante los eventos del vehículo en un tiempo máximo que consideramos correcto y aceptable. Otro ejemplo, son los equipos que controlan los brazos mecánicos en los procesos industriales de fabricación.

6.1.- Introducción.

En nuestros equipos ejecutamos distintas aplicaciones interactivas y por lotes.

Como sabemos, un microprocesador es capaz de ejecutar miles de millones de instrucciones básicas en un segundo (por ejemplo, un i7 puede llegar hasta los 3,4 GHz). Un micro, a esa velocidad, es capaz de realizar muchas tareas, y nosotros (muy lentos para él), apreciaremos que solo está ejecutando la aplicación que nosotros estamos utilizando. Al fin y al cabo, al micro, lo único que le importa es ejecutar instrucciones y dar sus resultados, no tiene conocimiento de si pertenecen a uno u otro proceso, para él son instrucciones. Es, el SO el encargado de decidir qué proceso puede entrar a ejecutarse o debe esperar. Lo veremos más adelante, pero se trata de una fila en la que cada proceso coge un número y va tomando su turno de servicio durante un periodo de tiempo en la CPU; pasado ese tiempo, vuelve a ponerse al final de la fila, esperando a que llegue de nuevo su turno.

Vamos a ver cómo el SO es el encargado de gestionar los procesos, qué es realmente un programa en ejecución, qué información asocia el SO a cada proceso. También veremos qué herramientas tenemos a nuestra disposición para poder obtener información sobre los procesos que hay en ejecución en el sistema y qué uso están haciendo de los recursos del equipo.

Los nuevos micros, con varios núcleos, pueden, casi totalmente, dedicar una CPU a la ejecución de uno de los procesos activos en el sistema. Pero no nos olvidemos de que además de estar activos los procesos de usuario, también se estará ejecutando el SO, por lo que seguirá siendo necesario repartir los distintos núcleos entre los procesos que estén en ejecución.

6.2.- Herramientas gráficas para la gestión de procesos.

Tanto los sistemas Windows como GNU/Linux proporcionan herramientas gráficas para la gestión de procesos. En el caso de Windows, se trata del Administrador de tareas, y en GNU/Linux del Monitor del sistema. Ambos, son bastante parecidos, nos ofrecen, al menos, las siguientes funcionalidades e información:

Listado de todos los procesos que se encuentran activos en el sistema, mostrando su PID, usuario y ubicación de su fichero ejecutable.

Posibilidad de finalizar procesos.

Información sobre el uso de CPU, memoria principal y virtual, red,

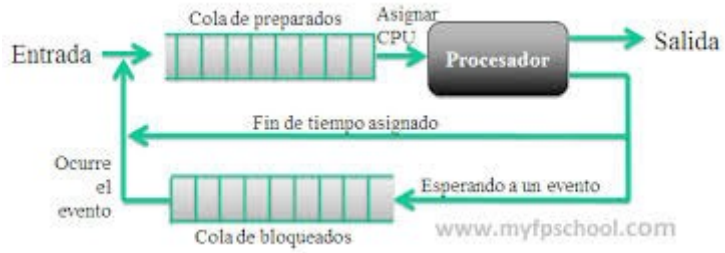
Posibilidad de cambiar la prioridad de ejecución de los procesos.

6.3.- Estados de un Proceso.

Si el sistema tiene que repartir el uso del microprocesador entre los distintos procesos, ¿qué le sucede a un proceso cuando no se está ejecutando? Y, si un proceso está esperando datos, ¿por qué el equipo hace otras cosas mientras que un proceso queda a la espera de datos?

Veamos con detenimiento, cómo es que el SO controla la ejecución de los procesos. Ya comentamos en el apartado anterior, que el SO es el encargado de la

gestión de procesos. En el siguiente gráfico, podemos ver un esquema muy simple de cómo podemos planificar la ejecución de varios procesos en una CPU.



En este esquema, podemos ver:

1. Los procesos nuevos, entran en la cola de procesos activos en el sistema.
2. Los procesos van avanzando posiciones en la cola de procesos activos, hasta que les toca el turno para que el SO les conceda el uso de la CPU.
3. El SO concede el uso de la CPU, a cada proceso durante un tiempo determinado y equitativo, que llamaremos quantum. Un proceso que consume su quantum, es pausado y enviado al final de la cola.
4. Si un proceso finaliza, sale del sistema de gestión de procesos.

Esta planificación que hemos descrito, resulta equitativa para todos los procesos (todos van a ir teniendo su quamtum de ejecución). Pero se nos olvidan algunas situaciones y características de nuestros los procesos:

Cuando un proceso, necesita datos de un archivo o una entrada de datos que deba suministrar el usuario; o, tiene que imprimir o grabar datos; cosa que llamamos 'el proceso está en una operación de entrada/salida' (E/S para abreviar). El proceso, queda bloqueado hasta que haya finalizado esa E/S. El proceso es bloqueado, porque, los dispositivos son mucho más lentos que la CPU, por lo que, mientras que uno de ellos está esperando una E/S, otros procesos pueden pasar a la CPU y ejecutar sus instrucciones.

Cuando termina la E/S que tenga un proceso bloqueado, el SO, volverá a pasar al proceso a la cola de procesos activos, para que recoja los datos y continúe con su tarea (dentro de sus correspondientes turnos).

Sólo mencionar (o recordar), que cuando la memoria RAM del equipo está llena, algunos procesos deben pasar a disco (o almacenamiento secundario) para dejar espacio en RAM que permita la ejecución de otros procesos.

Todo proceso en ejecución, tiene que estar cargado en la RAM física del equipo o memoria principal, así como todos los datos que necesite.

Hay procesos en el equipo cuya ejecución es crítica para el sistema, por lo que, no siempre pueden estar esperando a que les llegue su turno de ejecución, haciendo cola.

Por ejemplo, el propio SO es un programa, y por lo tanto un proceso o un conjunto de procesos en ejecución. Se le da prioridad, evidentemente, a los procesos del SO, frente a los procesos de usuario.

Con todo lo anterior, podemos quedarnos con los siguientes estados en el ciclo de vida de un proceso:

Nuevo. Proceso nuevo, creado.

Listo. Proceso que está esperando la CPU para ejecutar sus instrucciones.

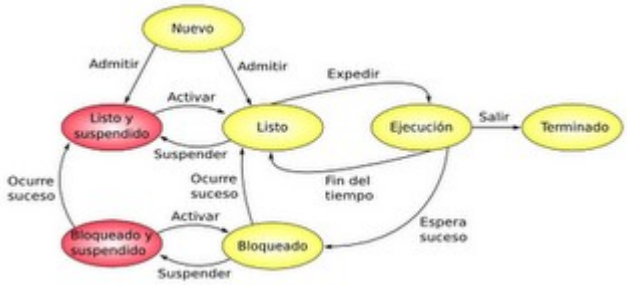
En ejecución. Proceso que actualmente, está en turno de ejecución en la CPU.

Bloqueado. Proceso que está a la espera de que finalice una E/S.

Suspendido. Proceso que se ha llevado a la memoria virtual para liberar, un poco, la RAM del sistema.

Terminado. Proceso que ha finalizado y ya no necesitará más la CPU.

El siguiente gráfico, nos muestra las distintas transiciones que se producen entre uno u otro estado:



6.4.- Planificación de procesos por el Sistema Operativo.

Entonces, ¿un proceso sabe cuando tiene o no la CPU? ¿Cómo se decide qué proceso debe ejecutarse en cada momento?

Hemos visto que un proceso, desde su creación hasta su fin (durante su vida), pasa por muchos estados. Esa transición de estados, es transparente para él, todo lo realiza el SO. Desde el punto de vista de un proceso, él siempre se está ejecutando en la CPU sin esperas. Dentro de la gestión de procesos vamos a destacar dos componentes del SO que llevan a cabo toda la tarea: el cargador y el planificador.

El cargador es el encargado de crear los procesos. Cuando se inicia un proceso (para cada proceso), el cargador, realiza las siguientes tareas:

1. Carga el proceso en memoria principal. Reserva un espacio en la RAM para el proceso. En ese espacio, copia las instrucciones del fichero ejecutable de la aplicación, las constantes y, deja un espacio para los datos (variables) y la pila (llamadas a funciones). Un proceso, durante su ejecución, no podrá hacer referencia a direcciones que se encuentren fuera de su espacio de memoria; si lo intentara, el SO lo detectará y generará una excepción (produciendo, por ejemplo, los típicos pantallazos azules de Windows).

2. Crea una estructura de información llamada PCB (Bloque de Control de Proceso). La información del PCB, es única para cada proceso y permite controlarlo. Esta información, también la utilizará el planificador. Entre otros datos, el PCB estará formado por:

Identificador del proceso o PID. Es un número único para cada proceso, como un DNI de proceso.

Estado actual del proceso: en ejecución, listo, bloqueado, suspendido, finalizando.

Espacio de direcciones de memoria donde comienza la zona de memoria reservada al proceso y su tamaño.

Información para la planificación: prioridad, quamtum, estadísticas, ...

Información para el cambio de contexto: valor de los registros de la CPU, entre ellos el contador de programa y el puntero a pila. Esta información es necesaria para poder cambiar de la ejecución de un proceso a otro.

Una vez que el proceso ya está cargado en memoria, será el planificador el encargado de tomar las decisiones relacionadas con la ejecución de los procesos. Se encarga de decidir qué proceso se ejecuta y cuánto tiempo se ejecuta. El planificador es otro proceso que, en este caso, es parte del SO. La política en la toma de decisiones del

planificador se denominan: algoritmo de planificación. Los más importantes son:

Round-Robin. Este algoritmo de planificación favorece la ejecución de procesos interactivos. Es aquél en el que cada proceso puede ejecutar sus instrucciones en la CPU durante un quamtum. Si no le ha dado tiempo a finalizar en ese quamtum, se coloca al final de la cola de procesos listos, y espera a que vuelva su turno de procesamiento. Así, todos los procesos listos en el sistema van ejecutándose poco a poco.

Por prioridad. En el caso de Round-Robin, todos los procesos son tratados por igual. Pero existen procesos importantes, que no deberían esperar a recibir su tiempo de procesamiento a que finalicen otros procesos de menor importancia. En este algoritmo, se asignan prioridades a los distintos procesos y la ejecución de estos, se hace de acuerdo a esa prioridad asignada. Por ejemplo: el propio planificador tiene mayor prioridad en ejecución que los procesos de usuario, ¿no crees?

Múltiples colas. Es una combinación de los dos anteriores y el implementado en los sistemas operativos actuales. Todos los procesos de una misma prioridad, estarán en la
misma cola. Cada cola será gestionada con el algoritmo Round-Robin. Los procesos de colas de inferior prioridad no pueden ejecutarse hasta que no se hayan vaciado las colas de procesos de mayor prioridad.

En la planificación (scheduling) de procesos se busca conciliar los siguientes objetivos:

Equidad. Todos los procesos deben poder ejecutarse.

Eficacia. Mantener ocupada la CPU un 100 % del tiempo.

Tiempo de respuesta. Minimizar el tiempo de respuesta al usuario.

Tiempo de regreso. Minimizar el tiempo que deben esperar los usuarios de procesos por lotes para obtener sus resultados.

Rendimiento. Maximizar el número de tareas procesadas por hora.

6.5.- Cambio de contexto en la CPU.

Una CPU ejecuta instrucciones, independientemente del proceso al que pertenezcan. Entonces, ¿cómo consigue unas veces ejecutar las instrucciones de un proceso y luego de otro y otro y otro..., sin que se mezclen los datos de unos con otros?

Un proceso es una unidad de trabajo completa. El sistema operativo es el encargado de gestionar los procesos en ejecución de forma eficiente, intentando evitar que haya conflictos en el uso que hacen de los distintos recursos del sistema. Para realizar esta tarea de forma correcta, se asocia a cada proceso un conjunto de información (PCB) y de unos mecanismos de protección (un espacio de direcciones de memoria del que no se puede salir y una prioridad de ejecuión).

Imaginemos que, en nuestro equipo, en un momento determinado, podemos estar escuchando música, editando un documento, al mismo tiempo, chateando con otras personas y navegando en Internet. En este caso, tendremos ejecutándose en el sistema cuatro aplicaciones distintas, que pueden ser: el reproductor multimedia VLC, el editor de textos writer de OpenOffice, el Messenger y el navegador Firefox. Todos ellos, ejecutados sin fallos y cada uno haciendo uso de sus datos.

El sistema operativo (el planificador), al realizar el cambio una aplicación a otra, tiene que guardar el estado en el que se encuentra el microprocesador y cargar el estado en el que estaba el microprocesador cuando cortó la ejecución de otro proceso, para continuar con ese. Pero, ¿qué es el estado de la CPU?

Una CPU, además de circuitos encargados de realizar las operaciones con los datos (llamados circuitos operacionales), tiene unas pequeños espacios de memoria

(llamados registros), en los que se almacenan temporalmente la información que, en cada instante, necesita la instrucción que esté procesando la CPU. El conjunto de registros de la CPU es su estado.

Entre los registros, destacamos el Registro Contador de Programa y el puntero a la pila.

El **Contador de Programa**, en cada instante almacena la dirección de la siguiente instrucción a ejecutar. Recordemos, que cada instrucción a ejecutar, junto con los datos que necesite, es llevada desde la memoria principal a un registro de la CPU para que sea procesada; y, el resultado de la ejecución, dependiendo del caso, se vuelve a llevar a memoria (a la dirección que ocupe la correspondiente variable). Pues el Contador de Programa, apunta a la dirección de la siguiente instrucción que habrá que traer de la memoria, cuando se termine de procesar la instrucción en curso. Este Contador de Programa nos permitirá continuar en cada proceso por la instrucción en dónde lo hubiéramos dejado todo.

El **Puntero a Pila**, en cada instante apunta a la parte superior de la pila del proceso en ejecución. En la pila de cada proceso es donde será almacenado el contexto de la CPU. Y de donde se recuperará cuando ese proceso vuelva a ejecutarse.

La CPU realiza un cambio de contexto cada vez que cambia la ejecución de un proceso a otro distinto. En un cambio de contexto, hay que guardar el estado actual de la CPU y restaurar el estado de CPU del proceso que va a pasar a ejecutar.

6.6.- Servicios. Hilos.

Caso práctico

La conclusión que estamos sacando, es, que todo lo que se ejecuta en un equipo es un programa y que, cuando está en ejecución, se llama proceso. Entones ¿qué son los servicios? ¿y los hilos?

En este apartado, haremos una breve introducción a los conceptos servicio e hilo, ya que los trataremos en profundidad en el resto de unidades de este módulo.

El ejemplo más claro de hilo o thread, es un juego. El juego, es la aplicación y, mientras que nosotros controlamos uno de los personajes, los 'malos' también se mueven, interactúan por el escenario y quitan vida. Cada uno de los personajes del juego es controlado por un **hilo**. Todos los hilos forman parte de la misma aplicación, cada uno actúa siguiendo un patrón de comportamiento. El comportamiento es el algoritmo que cada uno de ellos seguirá. Sin embargo, todos esos hilos comparten la información de la aplicación: el número de vidas restantes, la puntuación obtenida hasta eso momento, la posición en la que se encuentra el personaje del usuario y el resto de personajes, si ha llegado el final del juego, etc. Como sabemos, esas informaciones son variables. Pues bien, un proceso, no puede acceder directamente a la información de otro proceso. Pero, los hilos de un mismo proceso están dentro de él, por lo que comparten la información de las variables de ese proceso.

Realizar cambios de contexto entre hilos de un mismo proceso, es más rápido y menos costoso que el cambio de contexto entre procesos, ya que sólo hay que cambiar el valor del registro contador de programa de la CPU y no todos los valores de los registros de la CPU.

Un proceso, estará formado por, al menos, un hilo de ejecución.

Un proceso es una unidad pesada de ejecución. Si el proceso tiene varios hilos, cada hilo, es una unidad de ejecución ligera.

Nota:

¿Sabes lo que es Hyper-Threading (HT)? Es una tecnología patentada por Intel, que incorporó en sus micros Pentium4 de un sólo núcleo para que el propio micro (hardware) simulara la existencia de 2 núcleos lógicos, para obtener mayor productividad en procesos de más de un hilo, ya que cada núcleo lógico gestionará cada hilo de forma casi independiente. Esta tecnología la eliminó en sus Core 2 Duo y Quad; ya que al existir más de un núcleo hardware no hacía falta simular la existencia de más de un núcleo por núcleo físico. Y lo ha vuelto a introducir en su familia de microprocesadores i7, i5 e i3. Estos últimos, por cada núcleo físico, simulan 2 núcleos lógicos. Buscan así incrementar la productividad del micro.

Un servicio es un proceso que, normalmente, es cargado durante el arranque del sistema operativo. Recibe el nombre de servicio, ya que es un proceso que queda a la espera de que otro le pida que realice una tarea. Por ejemplo, tenemos el servicio de impresión con su típica cola de trabajos a imprimir. Nuestra impresora imprime todo lo que recibe del sistema, pero se debe tener cuidado, ya que si no se le envían los datos de una forma ordenada, la impresora puede mezclar las partes de un trabajo con las de otro, incluso dentro del mismo folio. El servicio de impresión, es el encargado de ir enviando los datos de forma correcta a la impresora para que el resultado sea el esperado. Además, las impresoras, no siempre tienen suficiente memoria para guardar todos los datos de impresión de un trabajo completo, por lo que el servicio de impresión se los dará conforme vaya necesitándolos. Cuando finalice cada trabajo, puede notificárselo al usuario. Si en la cola de impresión, no hay trabajos pendientes, el servicio de impresión quedará a la espera y podrá avisar a la impresora para que quede en StandBy.

Como este, hay muchos servicios activos o en ejecución en el sistema, y no todos son servicios del sistema operativo, también hay servicios de aplicación, instalados por el usuario y que pueden lanzarse al arrancar el sistema operativo o no, dependiendo de su configuración o cómo los configuremos.

Un servicio, es un proceso que queda a la espera de que otros le pida que realice una tarea.

6.7.- Comandos para la gestión de procesos.

¿Comandos? Las interfaces gráficas son muy bonitas e intuitivas, ¿para qué quiero yo aprender comandos?

Es cierto que podemos pensar que ya no necesitamos comandos. Y que podemos desterrar el intérprete de comandos, terminal o shell. Hay múltiples motivos por los que esto no es así:

En el apartado anterior, hemos visto que necesitamos comandos para lanzar procesos en el sistema.

Además de las llamadas al sistema, los comandos son una forma directa de pedirle al sistema operativo que realice tareas por nosotros.

Construir correctamente los comandos, nos permitirá comunicarnos con el sistema operativo y poder utilizar los resultados de estos comandos en nuestras aplicaciones.

En GNU/Linux, existen programas en modo texto para realizar casi cualquier cosa. En muchos casos, cuando utilizamos una interfaz gráfica, ésta es un frontend del programa en modo comando. Este frontend, puede proporcionar todas o algunas de las funcionalidades de la herramienta real.

La administración de sistemas, y más si se realiza de forma remota, es más eficiente en modo comando. Las administradoras y administradores de sistemas experimentadas utilizan scripts y modo comandos, tanto en sistemas Windows como GNU/Linux.

El comienzo en el mundo de los comandos, puede resultar aterrador, hay muchísimos comandos, ¡es imposible aprendérselos todos! Bueno, no nos alarmemos,

con este par de trucos podremos defendernos:

1. El nombre de los comandos suele estar relacionado con la tarea que realizan, sólo que expresado en inglés, o utilizando siglas. Por ejemplo: tasklist muestra un listado de los procesos en sistemas Windows; y en GNU/Linux obtendremos el listado de los procesos con ps, que son las siglas de 'process status'.

2. Su sintaxis siempre tiene la misma forma: nombreDelComandoopciones Las opciones, dependen del comando en si. Podemos consultar el manual del comando antes de utilizarlo. En GNU/Linux, lo podemos hacer con "man nombreDelComando"; y en Windows, con "nombreDelComando /?"

Recuerda dejar siempre un espacio en blanco después del nombreDelComando y entre las opciones.

Después de esos pequeños apuntes, los comandos que nos interesa conocer para la gestión de procesos son:

1. Windows. Este sistema operativo es conocido por sus interfaces gráficas, el intérprete de comandos conocido como Símbolo del sistema, no ofrece muchos comandos para la gestión de procesos. Tendremos:

tasklist. Lista los procesos presentes en el sistema. Mostrará el nombre del ejecutable; su correspondiente Identificador de proceso; y, el porcentaje de uso de memoria; entre otros datos.

taskkill. Mata procesos. Con la opción /PID especificaremos el Identificador del proceso que queremos matar.

2. GNU/Linux. En este sistema operativo, todo se puede realizar cualquier tarea en modo texto, además de que los desarrolladores y desarrolladoras respetan en la implementación de las aplicaciones, que sus configuraciones se guarden en archivos de texto plano. Esto es muy útil para las administradoras y administradores de sistemas.

ps. Lista los procesos presentes en el sistema. Con la opción "aux" muestra todos los procesos del sistema independientemente del usuario que los haya lanzado.
pstree. Muestra un listado de procesos en forma de árbol, mostrando qué procesos han creado otros. Con la opción "AGu" construirá el árbol utilizando líneas guía y mostrará el nombre de usuario propietario del proceso.

kill. Manda señales a los procesos. La señal -9, matará al proceso. Se utiliza "kill -9 <PID>".

killall. Mata procesos por su nombre. Se utiliza como "killall nombreDeAplicacion".
nice. Cambia la prioridad de un proceso. "nice -n 5 comando" ejecutará el comando con una prioridad 5. Por defecto la prioridad es 0. Las prioridades están entre -20 (más alta) y 19 (más baja).