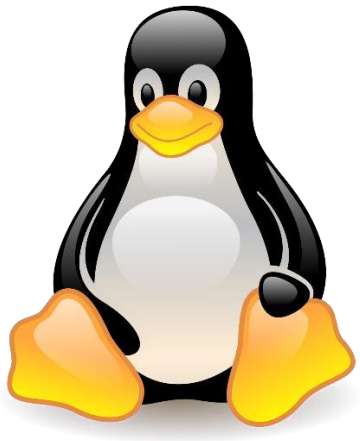
GESTIÓN DE INTERRUPCIONES Y DE PROCESOS EN LINUX

Me llamo Héctor Toribio González con DNI 12437502E y en este trabajo hablaré de lo dicho en el título. He escogido este tema porque me interesa profundizar en él ya que siempre me ha parecido un tema complejo, pero fundamental para el funcionamiento de un sistema operativo (Linux en este caso). A continuación, proporcionaré un índice de los temas específicos que voy a tratar en el informe:



INTRODUCCIÓN

¿Qué es un proceso?

Un proceso en Linux es un concepto muy general, prácticamente todo lo que se ve en el sistema operativo es un proceso o es fruto de uno. Un proceso, en pocas palabras, es una instancia de un programa en ejecución y como bien sabemos prácticamente todo lo que ocurre en un O.S es un programa.

¿Pero esta información para que nos sirve? Principalmente para darnos cuenta de lo importante que son los procesos en un sistema operativo y, por qué no decirlo, para incitarnos a indagar más en el concepto de proceso en Linux y como funciona.

Los procesos están formados por las instrucciones de un programa, por el estado actual de ese proceso, la memoria asignada a ese proceso y toda la información que le permite al S.O. su planificación y gestión ya que, efectivamente, estos están gestionados por el sistema operativo Linux. Cada vez que se crea un proceso, a ese proceso se le asigna un PID (Identificador de proceso), también hay que tener en cuenta que existirá el PPID (ID del proceso padre).

Un sistema utiliza varias iniciales para indicar en qué estado se encuentra el proceso:

- R → Running.
- S → Interruptible sleep (en espera de que ocurra algún evento en el sistema).
- D → Uninterruptible sleep (normalmente en espera de un dispositivo entrada salida).
- T → Stopped Z → Zombie process

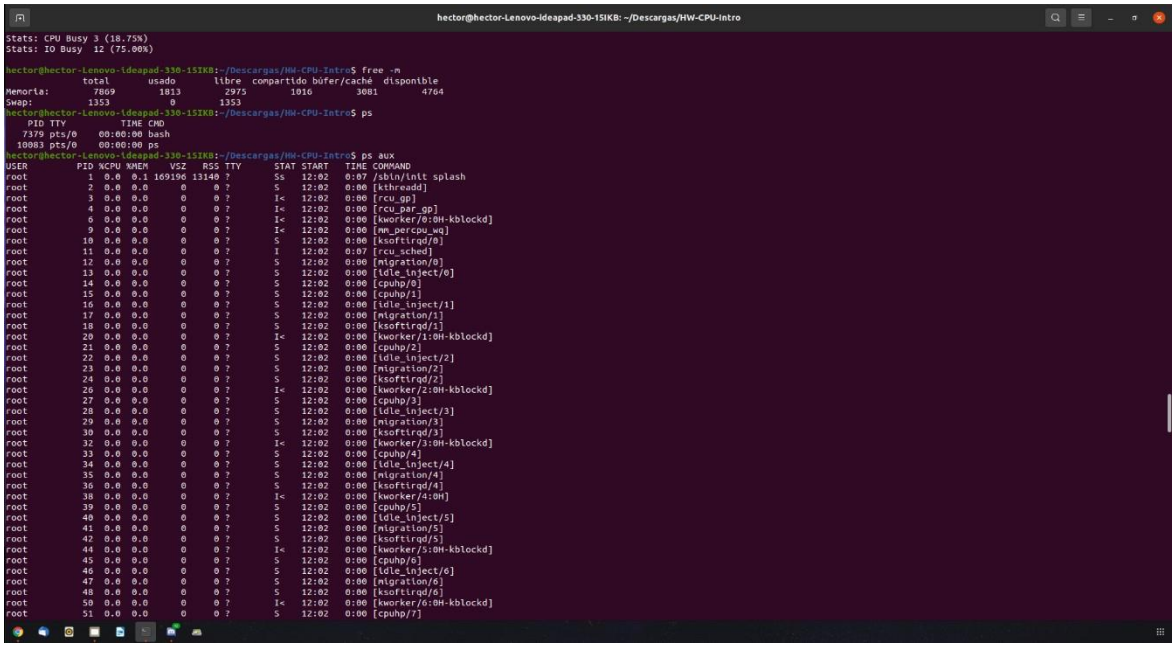
¿Qué es un "proceso zombie"?

Un proceso zombie es un proceso que ha muerto, pero su padre sigue esperando una respuesta por su parte. Esto provoca que el padre siga contando con que ese proceso está vivo, lo que causa que ese proceso siga en la tabla de procesos. Por este motivo se le llama zombie, porque parece que está vivo, pero en realidad está muerto.

Todo esto está muy bien, pero ¿cómo comprobamos nosotros en qué estado se encuentran los procesos y sus correspondientes PID? Todo esto y mucho más se logra gracias al comando "ps".

Comando "ps":

El comando ps es el comando más importante para la gestión de procesos en Linux ya que nos permite ver todos los procesos que se están llevando a cabo en el sistema en ese preciso momento. A continuación, una imagen de lo que muestra este comando y una breve explicación de toda la información que da.



UID: Número de identificación del usuario.

PID: Número de identificación del proceso padre.

PGID: Número del grupo de un proceso.

PRI: Prioridad de cada proceso.

VSZ: Memoria virtual del proceso (en Kb).

RSS: Memoria física del proceso (en Kb).

TTY: Terminal asociada al proceso.

STAT: Estado del proceso.

START: Momento en el que se inició el proceso.

TIME: Tiempo que lleva el proceso en ejecución.

COMMAND: Comando que provocó la ejecución del proceso.

Comando “kill”:

El comando “kill” nos permite finalizar un proceso utilizando su PID con la sintaxis “kill PID”.

El comando kill tiene muchas señales con diferentes opciones, pero las más usadas son “kill -9 PID” y “kill -15 PID”. Si usamos simplemente el comando “kill” con un PID estaremos ejecutando por defecto la señal “-15”.

La señal “-9” nos sirve básicamente para matar uno o varios procesos forzosamente mediante la siguiente sintaxis “kill -9 PID1 PID2 PID3…”.

El **comando “renice”** nos sirve para modificar la prioridad de un proceso.

¿Qué es una interrupción?

Una interrupción es una parada momentánea de la ejecución de un proceso para realizar otra rutina con mayor prioridad o que necesite utilizar la CPU en ese momento.

Estas interrupciones son también gestionadas por el sistema operativo ya que éste las envía al procesador cuando:

1. Es necesario sincronizar E/S: Cuando un dispositivo de E/S está preparado para ser utilizado se lo indica a la CPU, con lo que consigue activar la línea de bus de control llamada “LINEA DE PATICIÓN DE INTERRUPCIÓN”. Una vez ha llegado la petición de interrupción se pone en marcha una Rutina de Tratamiento de Interrupciones que se encarga de atender al dispositivo de E/S correspondiente.

2. Es necesario sincronizar otros procesos con mayor prioridad.

3. Un fin de Reloj. En ocasiones hay un reloj que se encarga de ir turnando a los procesos en la CPU. Cuando un proceso ha estado el máximo de tiempo posible en CPU y estamos en planificación expropiativa, el S.O expulsará al proceso que está en ese momento en ejecución de la CPU mediante una interrupción para dar paso a otro proceso.

Orígenes:

En cuanto a la historia del concepto “proceso” en la informática solo hace falta remontarse a la propia definición de “informática”. La informática es la ciencia que estudia la administración de métodos y procesos con el fin de manejar información y datos en formato digital. Con esto podemos deducir que los procesos ya estaban ahí, es algo que ya existía y viene de la mano con el nacimiento de la informática.

En cuanto a las interrupciones ya sabemos que nacieron de la necesidad de controlar estos procesos y se perfeccionaron gracias al Polling del cual hablaré más adelante.

LINUX

Gestión de procesos en Linux:

Para gestionar procesos en Linux primero tenemos que tener un proceso, el cual se crea simplemente ejecutando un programa. Con la llamada a "fork" conseguimos crear un proceso hijo igual al proceso padre.

Los procesos se pueden ejecutar en dos modos:

- **Modo usuario:** En este modo el proceso no tiene ningún privilegio. Es el "modo normal" por decirlo de alguna manera. Con este modo el proceso solo puede acceder a las zonas de memoria que se le han asignado, tiene varias instrucciones prohibidas como las que afectan al control de la máquina o de E/S, si el proceso quiere acceder a los recursos de la máquina solo puede hacerlo con una llamada al sistema pasando de esta manera al modo sistema y después sigue ejecutándose. Un proceso en modo usuario puede ser interrumpido en cualquier momento.

- **Modo núcleo:** Este modo es el "privilegiado", es decir, no tiene ninguna restricción. El proceso puede acceder directamente a los recursos y memoria del sistema, usar todas las instrucciones disponibles en el procesador y comunicarse con todos los dispositivos E/S.

  Linux es un Kernel reentrante, esto significa que puede haber varios procesos ejecutándose en modo núcleo al mismo tiempo. A pesar de esto, si ocurre una interrupción el cualquier momento, el kernel reentrante es capaz de suspender el proceso aun que se esté ejecutando en modo núcleo lo cual es una gran ventaja.

Generalmente cuando ejecutamos un proceso en Linux, este se ejecuta en modo usuario y cuando necesita algo que solo le puede ofrecer el núcleo cambia a modo núcleo. También existen otras situaciones en las que es necesario cambiar de modo, las más destacadas son cuando un proceso invoca una llamada al sistema, cuando se produce una excepción o cuando un dispositivo lanza una interrupción.

También hay diferentes maneras de clasificar los procesos, sobre todo de cara al Scheduling que explicaré más adelante:

- **Procesos Batch**: No necesitan de interacción con el usuario por lo que se ejecutan en segundo plano la mayoría de las veces.

- **Procesos de tiempo real**: Tienen un tiempo de respuesta muy corto, nunca deben ser bloqueados por procesos con menor prioridad.

- **Procesos interactivos**: Como dice el nombre son procesos que interactúan casi todo el tiempo con el usuario, esperando que pulse teclas, que haga clics… En cuanto uno de estos inputs se producen el proceso debe despertarse muy rápidamente (normalmente entre 50 y 150 milisegundos). Suelen tener prioridad alta.

Linux es un sistema que crea la ilusión de estar ejecutando varios procesos al mismo tiempo. Esto es gracias al "time sharing", que es la capacidad de dividir el tiempo de ejecución e ir ejecutando poco a poco varios procesos. Esto explica la palabra "ilusión", la cual es porque realmente no se ejecutan todos los procesos a la vez, sino que se ejecutan poco a poco intercalándose en la CPU. El encargado de ir intercalando estos procesos es el sistema operativo. Los que no se están ejecutando se quedan esperando en memoria gracias a la multiprogramación, que es la capacidad de que varios procesos estén preparados en memoria al mismo tiempo para ser ejecutados.

A continuación, hablaremos del **"Scheduling"**, proceso el cual analiza cuando intercalar los procesos y que proceso elegir a la hora de cambiarlos.

Scheduling:

El Scheduling es una política de planificación para saber en qué momento tenemos que cambiar de programa en el time sharing y que proceso elegir para intercambiar con el que se esté ejecutando en ese momento en la CPU. Todo esto lo tiene que conseguir sin perjudicar al rendimiento, la eficiencia, el tiempo de respuesta y evitando la inanición.

Si existe un quantum y un proceso no ha terminado antes del quantum, puede producirse un cambio de proceso. El time sharing es transparente a los procesos porque se basa en interrupciones de reloj (de esto hablaremos más tarde).

En el caso de que exista prioridad entre procesos, el Scheduling lo respeta y lo gestiona. Esto lo hace porque cada proceso tiene un valor de prioridad, estos valores los recibe el "Scheduler" que es el encargado de aplicar esta política. Una vez tiene los valores el Scheduler los utiliza para saber que procesos tienen que pasar o salir de CPU. En Linux ocurre una cosa un poco distinta, el Scheduler va modificando las prioridades de los procesos según va viendo que están haciendo. De esta manera se consigue que los procesos que llevan mucho tiempo sin entrar en CPU cambien su prioridad para entrar y que no se produzca la inanición.

Esto nombrado anteriormente es debido a que hay varios tipos de prioridad:

- Prioridad estática: Es la prioridad que se mantiene con el tiempo. Los procesos con prioridad estática siempre tienen la misma prioridad y pueden dar lugar a inanición.

- Prioridad dinámica: La prioridad dinámica es la que busca el Scheduler al buscar un proceso para ejecutarlo. Puede variar dinámicamente como se ha anotado anteriormente. En el Scheduling, la prioridad dinámica puede recibir un bonus entre 0 y 10 que depende de cómo se ha ido ejecutando el proceso y su tiempo de sleep.

  El tiempo de sleep promedio es el numero promedio de nanosegundos que el proceso pasa durmiendo. Los diferentes estados en los que duermen los proceso afectan distiendo al TSP. Si un proceso duerme en modo INTERRUMPIBLE afecta diferente que un proceso que duerme en modo UNINTERRUMPIBLE. El TSP nunca puede ser mayor que un segundo. El bonus asociado a cada TSP viene en la siguiente tabla:

Tiempo de sleep promedio	*Bonus*	*Granularidad*
Entre 0 y 99 ms	0	5120
Entre 100 ms y 199 ms	1	2560
Entre 200 ms y 299 ms	2	1280
Entre 300 ms y 399 ms	3	640
Entre 400 ms y 499 ms	4	320
Entre 500 ms y 599 ms	5	160
Entre 600 ms y 699 ms	6	80
Entre 700 ms y 799 ms	7	40
Entre 800 ms y 899 ms	8	20
Entre 900 ms y 999 ms	9	10
1 segundo	10	10

  Ambas prioridades se relacionan por la siguiente fórmula:

$$\textit{prioridad dinámica} = \max(100, \min(\textit{prioridad estática} - \textit{bonus} + 5, 139))$$

En cuanto a los tipos de procesos, si hablamos de procesos tipo Batch como son procesos que no necesitan una respuesta rápida, están penalizados por el Scheduler. Pueden ser acotados por E/S o por CPU. Si hablamos de los procesos tipo Tiempo Real tienen requerimientos de Scheduling muy estrictos ya que tienen que tener respuesta muy rápida. En cuanto a los interactivos no son fáciles de diferenciarlos con los Batch, aún que el Scheduler tiende a favorecer a los interactivos.

El **desalojo de procesos** es un parte muy importante del scheduling ya que es la herramienta que nos permite expulsar a procesos de CPU si no deben estar ahí (otro proceso tiene mayor prioridad), o si se termina su quantum. La duración de un quantum no debe ser ni muy excesiva ni muy baja ya que puede ocasionar que ningún proceso se ejecute o una sobrecarga del sistema. De esta manera desalojamos al proceso, no lo suspendemos, porque permanece en estado TASK_RUNNING, aunque no esté en CPU.

Algoritmos de Scheduling:

Los algoritmos de scheduling de las últimas versiones de Linux son mucho más sofisticados que en las primeras versiones. Los algoritmos actuales seleccionan el tiempo a correr en tiempo constante, independientemente del número de procesos que se encuentren preparados para ser ejecutados. También se ajustan correctamente al número de procesadores porque cada CPU tiene una cola de procesos. Como consecuencia de esto, en Linux se percibe que los programas interactivos son mucho mejor llevados que en versiones anteriores e incluso que en otros sistemas como Windows. A continuación, explicaré las tres principales clases de scheduling:

- **SCHED_FIFO:**

  Cuando el scheduler asigna a un proceso en CPU, deja el descriptor en la posición en la que está en ese momento de la cola de listos. Si no hay otros procesos de mayor prioridad listos, entonces el proceso continúa usando CPU, aun existiendo procesos con la misma prioridad.

- **SCHED_RR:**

  Round Robin. Cuando el scheduler asigna CPU a un proceso, pone el descriptor al final de la cola. Este algoritmo proporciona una compartición justa de la CPU a todos los procesos de tiempo real con este algoritmo.

- **SCHED_NORMAL:**

  Es el proceso convencional de tiempo compartido. El algoritmo se comporta muy diferente con los procesos convencionales que con los de tiempo real.

Funciones que usa el scheduler:

scheduler_tick()	Mantiene el contador *time_slice* de *current* actualizado
try_to_wake_up()	Despierta a un proceso dormido
recalc_task_prio()	Actualiza la prioridad dinámica de un proceso
schedule()	Selecciona un nuevo proceso para ser ejecutado
load_balance()	Mantiene balanceadas las colas de un sistema multiprocesador

Con esto termino con la gestión de procesos y paso a la gestión de interrupciones, la cual es necesaria para todo lo que hemos hablado.

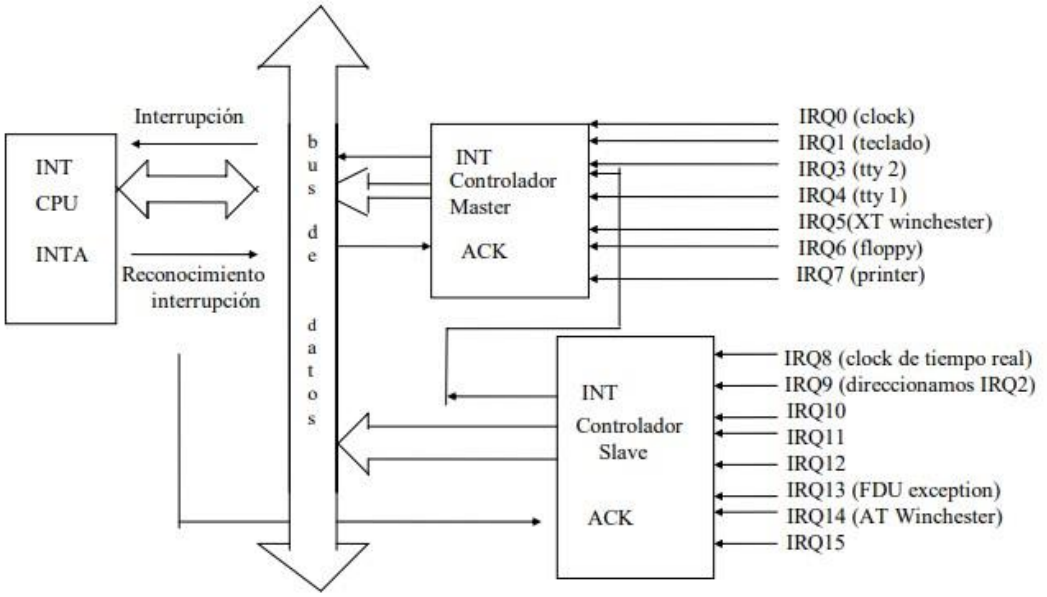
Gestión de interrupciones en Linux:

Las interrupciones son necesarias para la gestión de procesos. Son lo que permite expulsar procesos de CPU, parar el sistema cuando se necesita una entrada/salida o abortar procesos si se produce un error inesperado. Es prácticamente la base de la gestión de cualquier sistema operativo actualmente y se producen miles de ellas por cada operación que hacemos con el ordenado (presionar una tecla, abrir un programa, etc.).

Para empezar, conviene hablar de los tipos de interrupciones ya que no les he detallado antes. Existen tres tipos de interrupciones en Linux:

1. **Interrupciones software**: Son las que se producen cuando un usuario solicita una llamada al sistema.
2. **Interrupciones hardware**: Causadas por dispositivos de entrada/salida.
3. **Excepciones**: Causadas por la propia CPU.

Para que se produzca una interrupción hardware primero llega una petición a un controlador de interrupciones:



Este controlador envía la petición a la CPU y esta actica la línea de interrupciones del procesador. El procesador activará la línea de reconocimiento de interrupción para leer la interrupción correctamente. Una vez activada esta línea el controlador de interrupciones del dispositivo mandará el código asociado a la línea de interrupción activada. El procesador utilizará este valor para indexar la tabla de interrupciones del descriptor y, con esto, logrará encontrar la rutina del SO que cubrirá la interrupción.

En cuanto a la rutina que cubre la interrupción, el procesador cambiará a una pila diferente. Esta pila será la correspondiente al nuevo nivel de privilegio de la interrupción. El procesador también cargará el nuevo segmento de código y el nuevo puntero a la instrucción. A parte de esto deshabilitará las interrupciones del procesador.

Cuando se retorna de la instrucción el procesador restablecerá todos estos valores para seguir el funcionamiento correcto del sistema.

Todas las interrupciones de Linux tienen como punto de entrada código producido por el bucle entry.s. Este simplemente pone en la pila el número de interrupción menos 256. El proceso de la interrupción se realiza en C. Cuando se termina la interrupción se retorna el código que se había interrumpido por la función ret_from_intr escrita en C.

Cada manejador de interrupciones tiene una estructura con **varios campos**:

- Handler: Apunta a la función que servirá esa interrupción.
- Flag: Caracteriza el manejador.
- Mask: No se usa siempre.
- Name: Donde está el nombre del manejador.
- Dev_id: Es un identificador único.
- Next: Apunta a la siguiente acción a realizar.
- Irq: línea de interrupción en la que está el manejador.

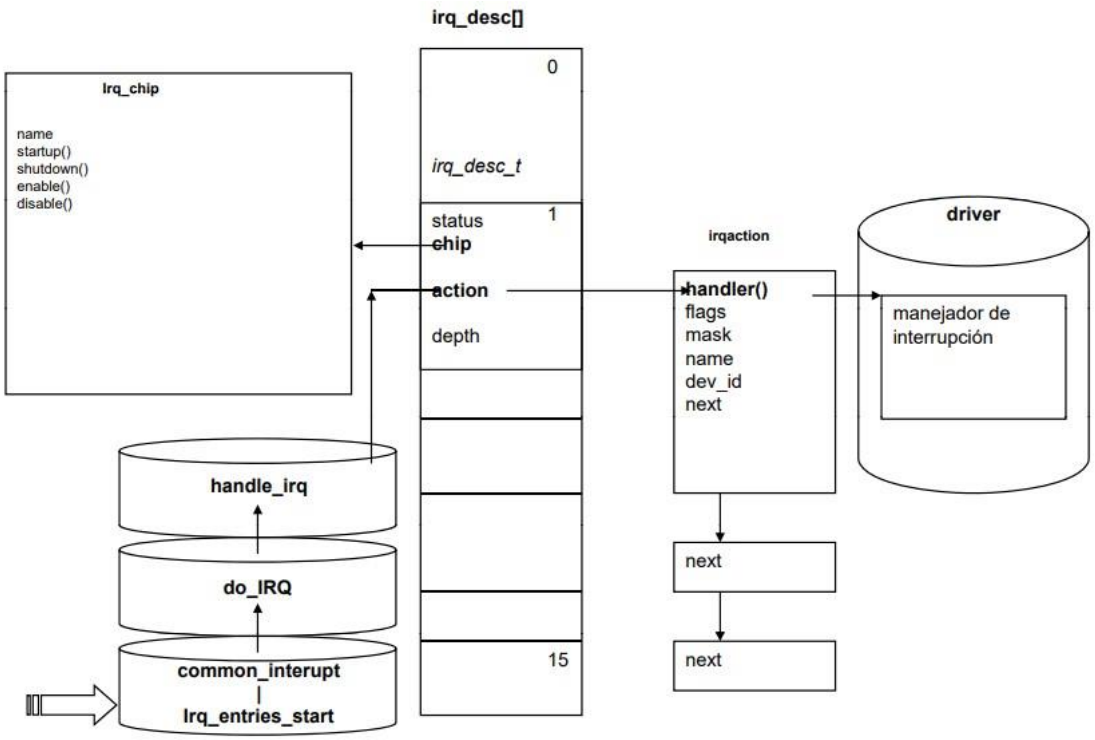
- Dir: Apunta al descriptor correspondiente a la interrupción irq (interrupt request, notificaciones de interrupción enviadas desde E/S a CPU).

A destacar de estos campos son las diferentes **flags** que puede tener el campo Flag:

- **SA_INTERRUPT**: Dice si se deben habilitar o no el resto de interrupciones durante esta rutina de manejo.
- **SA_SHIRQ**: Indica si la rutina permite compartir la interrupción.
- **SA_SAMPLE_RANDOM**: Indica que la interrupción que se está llevando a cabo puede ser una fuente de entropía.

Estructuras de datos de interrupciones:

- **Irqaction**: Donde se encuentra la dirección de la función de manejo de interrupciones.
- **Irq_chip**: Donde se encuentran las funciones que manejan un controlador de interrupciones particular, es dependiente de la arquitectura.
- **Irq_desc**: Vector con una entrada para cada una de las interrupciones que pueden ser atendidas.



Si hablamos de estados, cada línea de interrupción puede tener varios estados. Entre estos estados están el estado "**IRQ_INPROGRESS**" que indica la interrupción que está

siendo procesada actualmente, "**IRQ_DISABLED**" que nos dice que la interrupción está deshabilitada y "**IRQ_PENDING**" que nos dice que la interrupción se le ha reconocido al controlador de interrupciones pero que todavía no se ha empezado a ejecutar el manejador.

El flujo de las instrucciones lo he explicado anteriormente.

CONCLUSIONES

Para terminar, quiero decir que este trabajo me ha ayudado a profundizar mucho más en éste tema. Sobre todo, con el scheduling que casi no sabía de su existencia y todo lo relacionado con las interrupciones, especialmente los algoritmos. Me ha parecido muy interesante ver cómo funciona la gestión en Linux y el extenso flujo de eventos que se produce con el simple hecho de pulsar una tecla. FIN 😊

Bibliografía:

https://bioinf.comav.upv.es/courses/unix/control_procesos.html

https://www.sololinux.es/que-son-los-procesos-en-linux-y-como-identificarlos/

https://es.wikipedia.org/wiki/Proceso_(informática)

http://www.fdi.ucm.es/profesor/mendias/512/docs/tema8.pdf

https://es.wikipedia.org/wiki/Informática

https://www.linuxito.com/docs/procesos-linuxito.pdf

https://juncotic.com/procesos-en-linux-estados-y-prioridades/

https://francisconi.org/linux/comandos/ps

https://www.hostinger.es/tutoriales/cancelar-proceso-comando-kill-linux

https://www.profesionalreview.com/2017/02/26/como-manejar-y-matar-procesos-desde-la-consola-en-linux/

https://es.slideshare.net/Tensor/clase-1-conceptos-bsicos-de-los-so

http://linux.linti.unlp.edu.ar/images/0/07/Concurrencia-sincronizacion-linux.pdf

http://cursos.delaf.cl/website/webroot/archivos/cursos/sistemas-operativos/material-de-apoyo/Scheduling%20de%20procesos%20en%20Linux%202.6.pdf

http://sopa.dis.ulpgc.es/ii-dso/leclinux/interrupciones/int_hard/int_hard.pdf

http://sop.upv.es/gii-dso/es/t6-1-interrupciones/tr2.html