

## ARQUITECTURA

Chip	Introduction	Data bus	Address bus
4004	1971	4	8
8008	1972	8	8
8080	1974	8	16
8085	1977	8	16
8086/88	1978	16/8	20
80186/188	1982	16/8	20
80286	1983	16	24
80386DX	1986	32	32
80386SX	1988	16	24
80486DX (With coprocessor)	1989	32	32
80486SX (Without coprocessor)	1989	32	32
Pentium	1993	64	32

Table 1.1 80X86 family tree

Arquitectura de IA 32 de INTEL, IA 32 es el nombre que tiene la arquitectura del procesador PENTIUM, procesador tiene una arquitectura IA 32, el y los subsiguientes, recién cambio para los procesadores de 64 bits, donde la arquitectura es IA 64.

La imagen tiene un poco la historia como ha ido avanzando, puse este gráfico porque acá vamos a ver el procesado 8088 el cual era el de la PC original, si les cuento esto porque ya vamos a ver en un ratito que PENTIUM, cualquier IA 64 cuando arranca, tiene dos o tres modos de funcionamiento por compatibilidad hacia atrás, tiene un modo de funcionamiento que se llama *modo real* y por más que sea un procesador

nuevísimo funciona como el 8088, donde tiene el BUS de direcciones de 20 bits nada más, con lo cual puede direccionar 1MB, y lo más triste que vamos a ver un poquito más adelante que ni siquiera es continua.

Es IA 32 porque el BUS de direcciones tiene 32 bits, si bien el BUS de datos es de 64 yo puedo acceder a palabras de 64 bits de ancho, el BUS de direcciones es de 32. Entonces yo puedo direccionar 4 GB de RAM.

Figure 2-1 shows a block diagram of the Pentium processor.

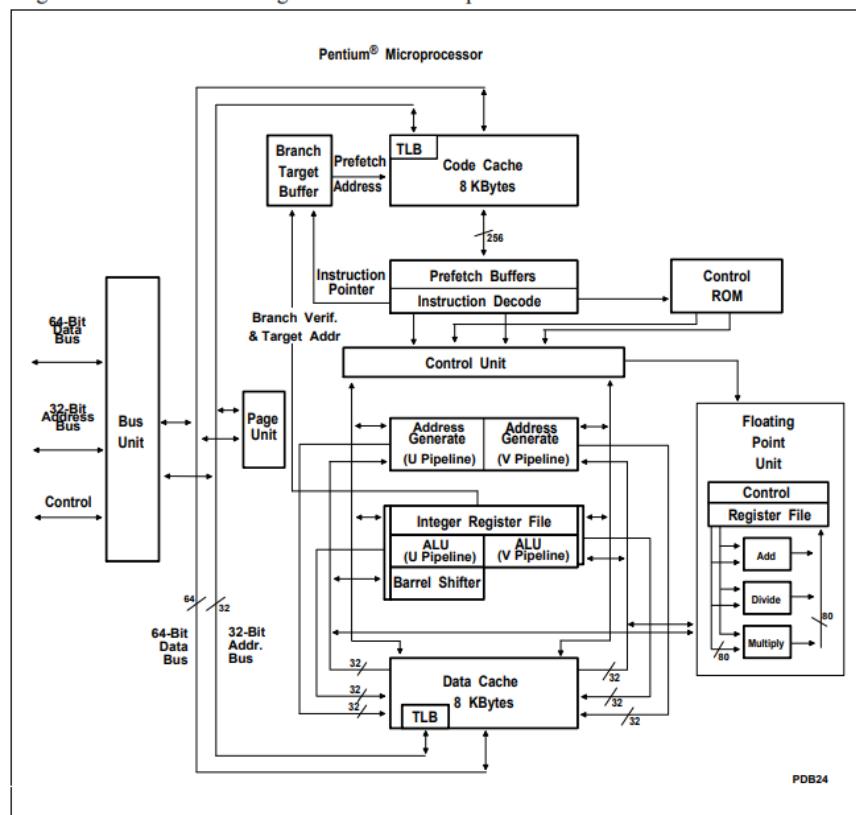
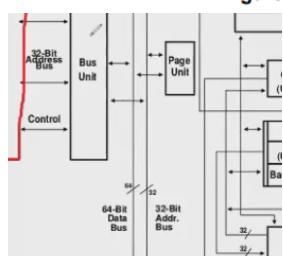


Figure 2-1. Pentium® Processor Block Diagram

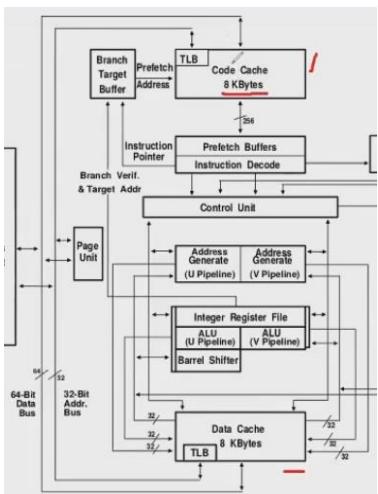
Esta es la arquitectura interna, en forma genérica, los nuevos agregan algunas funcionalidades, pero lo más importante es que en la pastilla no hay 1 sólo de estos, sino que hay 2, 4 u 8, los CORE que quieran, básicamente eso, y la lógica para que no se peleen por la misma dirección de memoria y ese tipo de cosas, pero básicamente la estructura interna sigue siendo ésta.

Tiene 2 ALUs, la "U" y la "V", la V permite hacer cualquier la operación incluyendo operaciones de punto flotante, la U algunas operaciones de punto flotante sencillas, no todas.

Tiene un PIPELINE o una tubería de varias etapas.



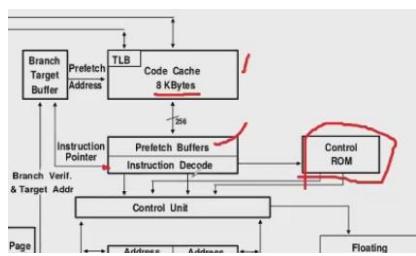
La memoria RAM estaría a la izquierda afuera, donde tengo un BUS de direcciones de 32 como hemos visto que se puede direccionar 4 GB y el BUS de datos de 64.



Tengo memoria caché interna de 8KBytes, es un procesador viejo ahora son mucho más grandes, tengo separada la caché para datos y para códigos sea para instrucciones, cuando va a buscar una instrucción, busco si está en la caché, no me hace falta ir a buscar a la memoria principal por lo tanto es más rápido, al igual que los datos, si necesito acceder a un dato lo busco en la caché y si no está recién voy por abajo por dirección y dato a acceder a la memoria principal.

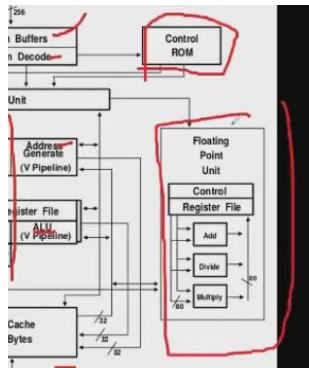
TLB es para hacer una translación de direcciones pero cuando utilizamos la memoria que está en la unidad de paginación eso por ahora no lo vamos a ver, existe tanto para la caché de instrucción como para la de datos.

Tengo un BUFFER de pre-búsqueda, necesito tener un mínimo de dos instrucciones porque voy a alimentar el camino de datos de la izquierda que es de la ALU U y al de la ALU V, por eso necesito tener al menos 2, pero esta CPU tenía lo que se llama un algoritmo de búsqueda de saltos, entonces cuando yo codifico una instrucción y sé que puede ser un salto condicional lo que hace es ir a buscar también a la memoria la otra posible dirección donde va a saltar, entonces si o si éste BUFFER de pre-búsqueda va a tener lugar para 2 instrucciones para un PIPELINE, el U, y otras dos instrucciones para el V.



Las instrucciones se obtienen de la caché o de la memoria principal y eso alimenta al decodificador de instrucciones, la cual es micro-programada, La ROM de control tiene un micro código, una vez que se decodifica, tiene dos etapas la decodificación de la instrucción y de la generación del código, una vez que ya decodifique la instrucción lo que hago es efectivamente pasársela a la unidad ALU para que haga la operación, tanto para el flujo V o para el U, y una vez se tiene la operación se almacena de nuevo en los registros internos o en la memoria que

se accederá por abajo, la memoria principal.



Esta es la unidad de punto flotante por hardware, esto me permite en el mejor de los casos ejecutar dos instrucciones por ciclo de reloj, se llaman súper escalares cuando pueden ejecutar más de una, así que cae en esas características esta arquitectura.

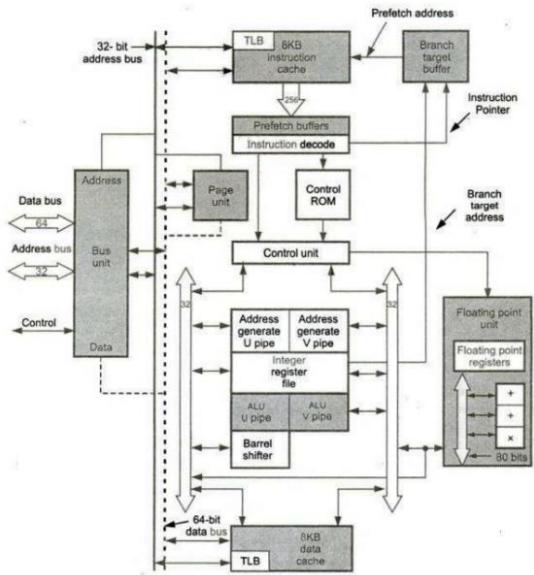


Fig. 1.2 Pentium architecture block diagram

[Microprocessor\\_Microcontroller\\_3ed.pdf](#)

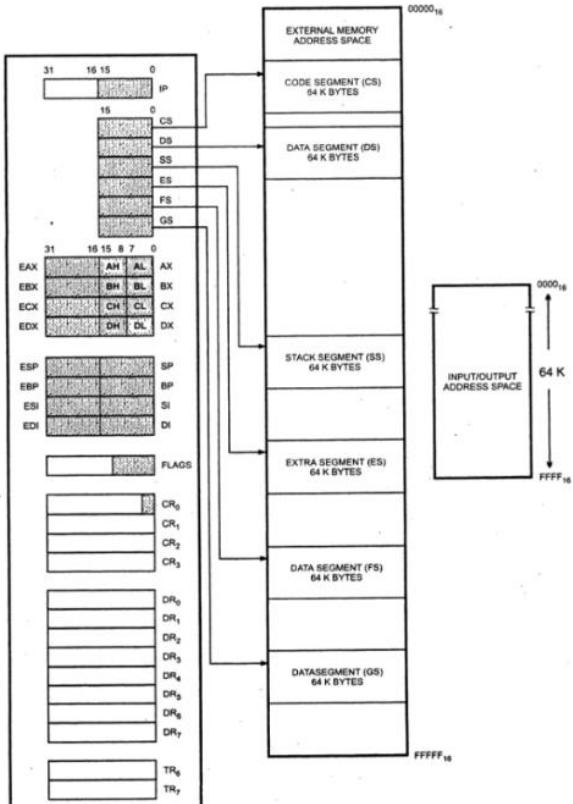


Fig. 1.6 Real mode programming model for Pentium processor

Es la misma imagen, solo que la anterior es la del manual INTEL y se ve un poco más clara.

Estos son todos los registros internos que tiene la IA 32, los que están a la izquierda. Tiene registros internos de 32 bits, la mayoría, pero se ven sólo 16, eso porque tiene compatibilidad hacia atrás cuando arrancan, después pasamos a algo que se llama modo protegido que tiene toda la potencia del procesador, la visibilidad de los registros es completa.

Les comento esto porque como les decía esto puede direccionar solamente 20 bits, y para el colmo, podemos ver que va de la dirección 00000h a la posición FFFFFh, en HEXA, son 20 bits, pero no es que lo direcciona de manera continua, no es que tenga un contador del programa que va de 00000h a FFFFFh, como en el registro del contador del programa tengo solo 16 bits en realidad yo podría direccionar solo 64K nada más, y ¿qué hago con el otro mega?, lo que hace INTEL, o decidió hacer en el 8088 y después quedó atado a esa compatibilidad hacia atrás, es que usa una serie de registro que se llaman registros de segmentos, por ejemplo el CODE SEGMENT (CS) que es el segmento de código o de programa, el DATA SEGMENT (DS), entonces lo que hace es lo siguiente, por ejemplo si yo al CODE SEGMENT (CS) le pongo un valor estoy fijando en qué parte de la memoria voy a trabajar esos 64K con mi contador del programa, mi contador de programa puede moverse desde 000 hasta 64K, porque tiene 16 bits, entonces a partir de un valor que yo ponga en CS es donde este segmento puede estar ubicado en el Mega de RAM, y este

segmento se puede mover en cualquier parte del Mega dependiendo del valor que coloque en CS para este caso en particular. Es tan libre de hacer esto que de hecho podría sin querer solapar segmentos y "CHAU".

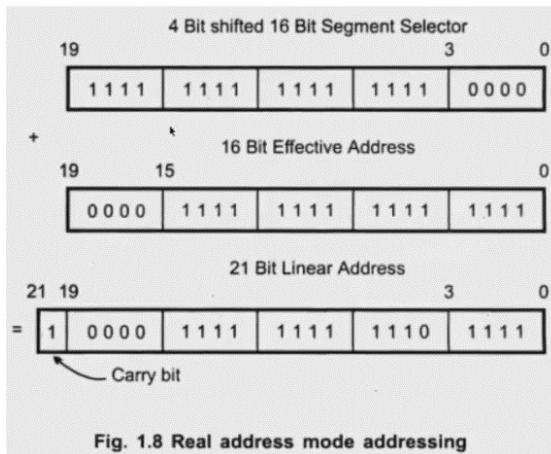
Éste es modo real, cuando arranca, no deberíamos usar procesador PENTIUM en modo real, porque sería como usar un 8088 que básicamente es una porquería no me permite usar multiprogramación, no puedo tener más de un programa a la vez, porque si en el mismo programa puedo hacer lio imagínese con muchos programas.

Lo que se hace normalmente es, cuando arranca, se pone un modo protegido que ahí si tengo protección pero por hardware, no puedo solapar segmentos, un segmento con otro o ese tipo de cosas, entonces obviamente es mucho mejor, y además en vez de direccionar estos 64K que los puedo ir moviendo en cualquier parte del Mega nada más, puede direccionar hasta 4GB y podrían ser continuos inclusive.

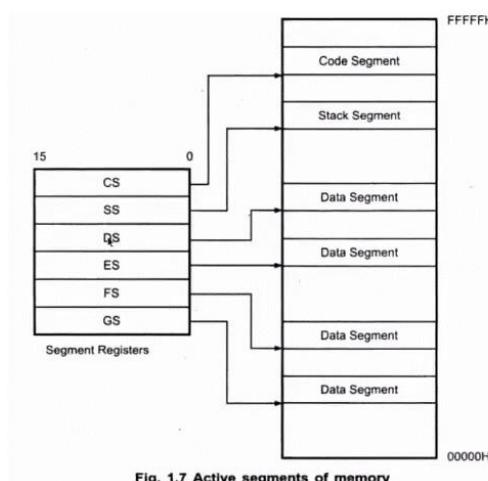
Les estoy contando esto porque la operación es parecida, por ejemplo, si yo quiero buscar en la memoria una instrucción, lo que voy a tener que hacer es ver dónde está el segmento del código, ósea veo el valor que tiene CS, DS, y después dentro de ese segmento de código con el contador de programa voy a tener el offset por así decir, si por ejemplo quisiera hacer un PUSH o un POP a una pila, va a estar en el segmento de pila y después con un registro STACK POINTER voy a poder ver o ir incrementando o decrementando dentro de la pila.

Esto se maneja de a pares de registros, por ejemplo el CODE SEGMENT y el contador de programa es para ubicar una instrucción, el STACK SEGMENT y el STACK POINTER para ubicar o para hacer un PUSH o POP de una pila, después por otro lado el DATA SEGMENT va a depender un poco de la instrucción que yo usé, si uso relativo, absoluto, etcétera, como adentro de ese segmento ubico mi dato.

Así de feo era INTEL, o sea y ¿por qué hicieron esto? porque en realidad ellos tenían un procesador que tenía 16 bits, pero si querías una pc en los 80', 64K de RAM era muy poquito entonces se les ocurrió hacer esta cosa tan linda que fue segmento y desplazamiento, ¿cómo se obtienen esos 20 bits?



El registro de segmento, no importa cual, depende de si es un instrucción, si quiero acceder a la pila o no, lo que hago es al segmento lo desplazó 4 bits (hacia la izquierda), como era 16 lo desplazó y me queda de 20, y le sumo el offset, por ejemplo este podría ser el CODE SEGMENT CS, lo desplaza y después lo sumo contador de programa INSTRUCCIÓN POINTER y eso mágicamente me da los 20 bits que estamos viendo en la imagen anterior de registros. Entonces con el CODE SEGMENT me puedo mover por todo el mega y con el contador de programa me muevo dentro de ese segmento, pero podría equivocarme y hacer lo que el STACK SEGMENT se superponga en alguna parte con CODE SEGMENT y ese es un problema.



Acá están más detallados los 6 registros de segmentos y como está ubicada la memoria, pero es solo un ejemplo esa disposición.

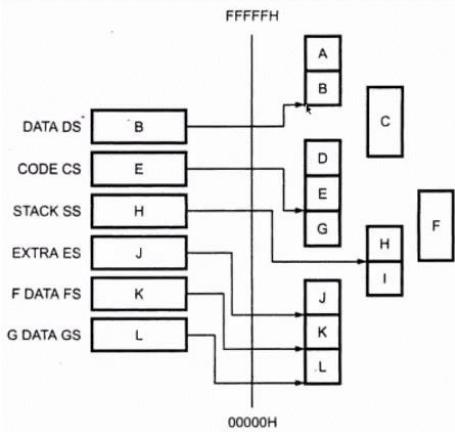


Fig. 1.9 Contiguous, adjacent, disjointed and

Les cuento esto porque cuando vemos el direccionamiento de modo protegido tiene que ver con esto, tiene un par de registros, el segmento y el desplazamiento pero no se suman así, nada que ver, pero se usan de a pares, o sea cuando voy a buscar en la memoria una instrucción voy a buscar él CODE SEGMENT y el contador programa, nada más que el procedimiento es otro, no es que se desplace y se sume, cuando voy a buscar algo en la pila va a buscar el STACK POINTER con el STACK SEGMENT, sigue haciendo eso solo que el mecanismo es totalmente distinto.

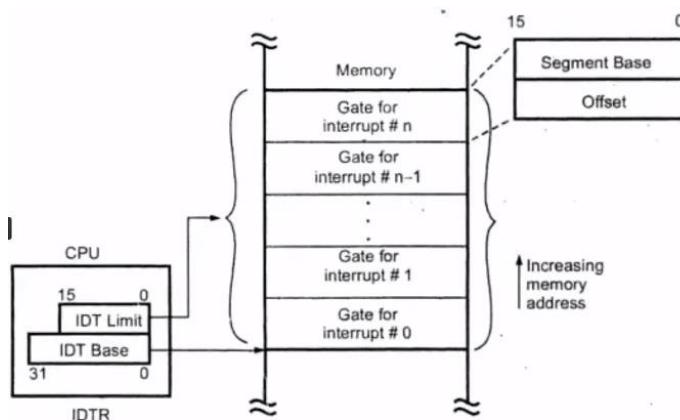


Fig. 1.10 Interrupt descriptor table

desplazamiento, en modo real.

Cuando llega una interrupción de acuerdo al nuevo número de interrupción que se asalta a la columna del medio y esto va a ejecutar la rutina que está en la base de la derecha con el offset.

Para modo real el segmento base se desplaza cuatro lugares y se suma con el offset y me da una rutina en el mega.

1. PF Prefetch
2. D1 Instruction Decode
3. D2 Address Generate
4. EX Execute, Cache and ALU Access
5. WB Writeback

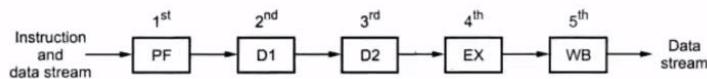


Fig. 1.13 Stages in U and V instruction pipelines

generación de la dirección, la ejecución y la escritura de los registros, esas son las cinco etapas.

Lo separa en pipeline porque básicamente yo podría en una etapa hacer una pre búsqueda de una instrucción mientras que otra puede estar haciendo la decodificación de otra instrucción mientras que otras pues estar haciendo la ejecución u otra puede estar haciendo la escritura del resultado.

Podría pasar lo que vemos en este caso, A y B están adyacentes uno al lado del otro, pero se están compartiendo un pedazo segmento con B y con D, entonces los datos o la información que guardan ahí se podría pisar, entonces no tengo desde el punto de vista hardware, nada que me dé protección contra algún lío que haga el programador.

Otra cosa que tienen modo real a diferencia del 8088 es que tiene interrupciones vectorizadas, entonces tiene una tabla de interrupciones, en el 8088 arranca en la posición 0, en PENTIUM en modo real tengo un par de registros que me permiten ponerlo en cualquier parte de la memoria, no necesariamente va a estar a la posición cero, lo hago con un registro que se llama IDT INTERRUPT DESCRIPTOR TABLE BASE, después tengo un límite que me limita la cantidad de vectores de interrupción que tengo, en INTEL cada vector de interrupción tiene 4 bytes y básicamente son dos bytes para el segmento y dos bytes para el

El pipeline el objetivo del pipeline es que si puedo separar en distintas zonas del procesador y pueden trabajar de manera independiente, entonces en el mismo momento podría hacer varias cosas entonces lo que PENTIUM hizo fue para cada una de las de las trayectorias de datos la U y la V tiene cinco etapas, que es la pre-búsqueda, la decodificación, la

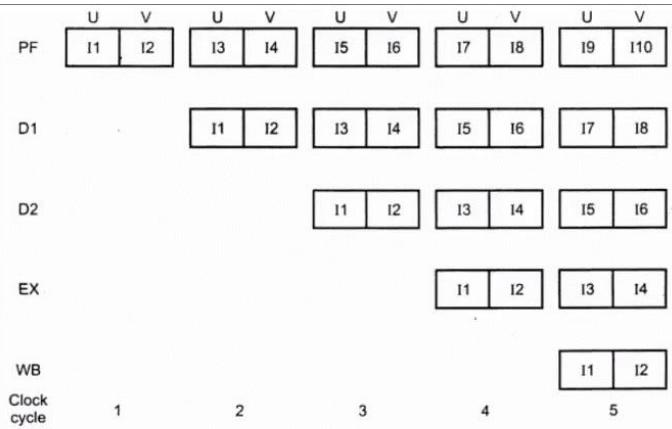
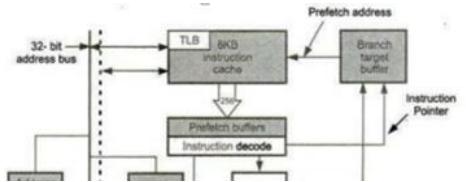


Fig. 1.14 Pipelined instruction execution

obviamente en el ciclo 3 hago la segunda parte de la decodificación que es la generación de la dirección de algún operando, lo hago para la instrucción I1 e I2 mientras que como ya hice la pre búsqueda de la instrucción I3 e I4 en el tercer ciclo ya la puedo decodificar y mientras tanto puedo hacer la pre búsqueda de la instrucción I5 e I6, si hacemos lo mismo con la ejecución y con la escritura, fíjense que a partir del 5to ciclo **en teoría puedo ejecutar dos instrucciones por ciclo**.

Hay algunas salvedades como la dije antes, esta ALU no se bancaba por ejemplo si las dos instrucciones I1, I2 e I3, I4 en ejecución fuesen de punto flotante complejas, hay un par de instrucciones de punto flotante que si se podrían hacer en la ALU.

Cuando puedo ejecutar más de una instrucción a la vez se llama arquitectura súper escalar y bueno Pentium al tener estas cinco etapas podría por ciclos de reloj ejecutar en teoría una instrucción y hay otro tema también, qué pasa si cuando se ejecuta la instrucción tiene un salto condicional, entonces invalidaría las siguientes instrucciones porque el flujo del programa se va por otra parte.



Entonces si al hacer la decodificación me encuentro con que es un condicional, hay un bloque **BRANCH TARGET BUFFER** para saltos, entonces la idea es que cuando decodifico y sé que puede haber un salto lo que hace el bloque es intentar hacer la búsqueda de la posición a la que podría saltar si se cumple ese salto condicional.

Es por eso que lo almacena primero en la pre búsqueda, entonces tendría que por cada ALU dos búsquedas de instrucciones, la que sigue o la que sigue si salta a otra posición de memoria, entonces de alguna manera con esto lo que intento hacer es acelerar o tratar de que no pierda todo el procesamiento cuando me di cuenta que hay un salto condicional, si lo ejecuto y resulta que va a otro lado, ya tener la búsqueda de antemano en mí procesador, me ahorro la búsqueda en memoria.

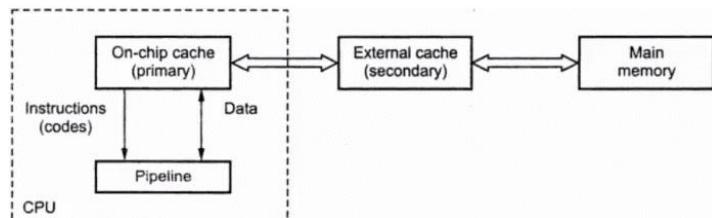


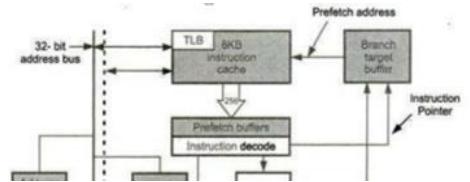
Fig. 1.16 Two-level cache system in a microcomputer

memoria caché, básicamente la memoria caché son más rápidas que las memorias RAM, porque son memorias asociativas, lo que pasa que son más caras, ocupan más lugar, entonces en una pastilla si pudiera meter memoria asociativa sería mucho más rápido pero sería más caro y debería ser la pastilla más grande y cosas por el estilo, entonces normalmente lo que hago es poner una memoria chiquita donde almaceno por ejemplo las últimas instrucciones que usé, en realidad cuando accedo a la memoria principal se excede en bloques de 32 Bytes, entonces buscó 30 Bytes de instrucciones y los almacenó en la caché, lo más probable que dar un salto condicional la próxima instrucción ya esté en la caché, que está almacenada y no me haga falta ir a la memoria principal. Es mucho más rápida y además está dentro de la pastilla del procesador, gano tiempo.

En esta imagen está puesto como si no hubieran saltos ni nada por el estilo, en el eje horizontal tenemos ciclos de reloj donde puedo hacer cada una de las etapas con un ciclo de reloj, entonces en el primer ciclo del reloj tengo la pre búsqueda tanto para la trayectoria de datos U como para la V, entonces podría poner una instrucción en el U y otra en la V, tengo la instrucción I1 e I2, de la pre búsqueda nada más, mientras que en el ciclo dos esas instrucciones como ya la busqué las puedo mandar al decodificador, que es lo que pasa en ciclo dos, se está decodificando la I1 y la I2, mientras que esto sucede podría hacer la pre búsqueda de la instrucción I3 y la I4, obviamente en el ciclo 3 hago la segunda parte de la decodificación que es la generación de la dirección de algún operando, lo hago para la instrucción I1 e I2 mientras que como ya hice la pre búsqueda de la instrucción I3 e I4 en el tercer ciclo ya la puedo decodificar y mientras tanto puedo hacer la pre búsqueda de la instrucción I5 e I6, si hacemos lo mismo con la ejecución y con la escritura, fíjense que a partir del 5to ciclo **en teoría puedo ejecutar dos instrucciones por ciclo**.

Hay algunas salvedades como la dije antes, esta ALU no se bancaba por ejemplo si las dos instrucciones I1, I2 e I3, I4 en ejecución fuesen de punto flotante complejas, hay un par de instrucciones de punto flotante que si se podrían hacer en la ALU.

Cuando puedo ejecutar más de una instrucción a la vez se llama arquitectura súper escalar y bueno Pentium al tener estas cinco etapas podría por ciclos de reloj ejecutar en teoría una instrucción y hay otro tema también, qué pasa si cuando se ejecuta la instrucción tiene un salto condicional, entonces invalidaría las siguientes instrucciones porque el flujo del programa se va por otra parte.



Entonces si al hacer la decodificación me encuentro con que es un condicional, hay un bloque **BRANCH TARGET BUFFER** para saltos, entonces la idea es que cuando decodifico y sé que puede haber un salto lo que hace el bloque es intentar hacer la búsqueda de la posición a la que podría saltar si se cumple ese salto condicional.

Es por eso que lo almacena primero en la pre búsqueda, entonces tendría que por cada ALU dos búsquedas de instrucciones, la que sigue o la que sigue si salta a otra posición de memoria, entonces de alguna manera con esto lo que intento hacer es acelerar o tratar de que no pierda todo el procesamiento cuando me di cuenta que hay un salto condicional, si lo ejecuto y resulta que va a otro lado, ya tener la búsqueda de antemano en mí procesador, me ahorro la búsqueda en memoria.

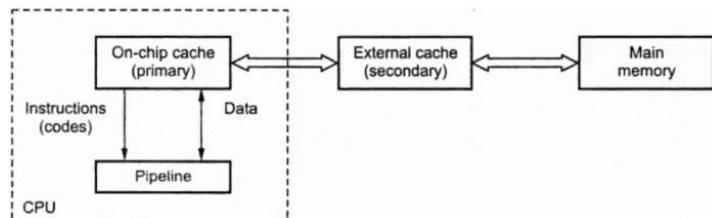
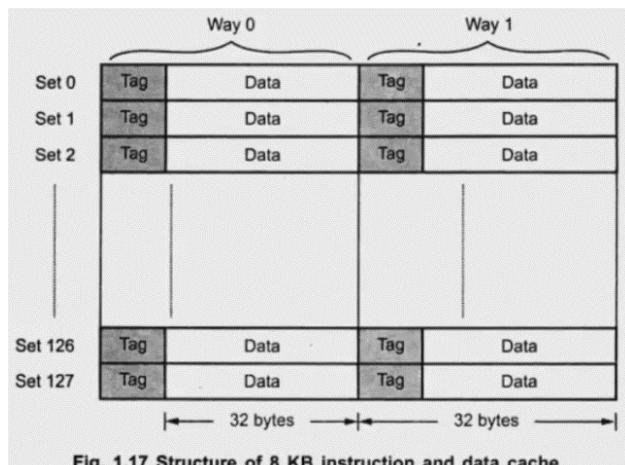


Fig. 1.16 Two-level cache system in a microcomputer

memoria caché, básicamente la memoria caché son más rápidas que las memorias RAM, porque son memorias asociativas, lo que pasa que son más caras, ocupan más lugar, entonces en una pastilla si pudiera meter memoria asociativa sería mucho más rápido pero sería más caro y debería ser la pastilla más grande y cosas por el estilo, entonces normalmente lo que hago es poner una memoria chiquita donde almaceno por ejemplo las últimas instrucciones que usé, en realidad cuando accedo a la memoria principal se excede en bloques de 32 Bytes, entonces buscó 30 Bytes de instrucciones y los almacenó en la caché, lo más probable que dar un salto condicional la próxima instrucción ya esté en la caché, que está almacenada y no me haga falta ir a la memoria principal. Es mucho más rápida y además está dentro de la pastilla del procesador, gano tiempo.

En el caso de Intel puedo tener una varios niveles de caché dentro del procesador, se le llama primaria, que es la que vimos en las filminas, podría tenerla fuera de la pastilla, eso es una cache secundaria.



La estructura interna tiene dos canales dado que hay dos trayectorias de datos, U y V, lo que hace cada vez que lee un dato es almacenar, en este caso de 8k, que hoy en día la nueva a ser mucho más este, este almacena 32 Bytes, o sea, lee 32 byte de la mara principal y lo escribe tanto para él canal 1 como para el canal 2, y este en este caso tiene 128 registros o entradas, en el caso que no se encuentra acá lo busca con la etiqueta y va a la memoria principal para acceder al dato, pero no se accede a ese byte solo, si no que accedo en este caso a ese Byte más 32 más y antes de mandarlo a la unidad de pre búsqueda directamente los copio acá a esos 32, porque probablemente la siguiente instrucción esté en esos 32 Bytes.

## SEGMENTACION Y PAGINACION

### Segmentación y paginación Arquitectura IA32

Técnicas Digitales III  
UTN – FRM

Colaboración: grid T ICS

En esta presentación vamos a ver un ejemplo de segmentación y paginación para arquitectura IA-32.

#### Visión general de la gestión de memoria

La segmentación brinda un mecanismo para aislar módulos de código, datos y pila para que múltiples programas puedan ejecutarse en el mismo procesador sin interferirse.

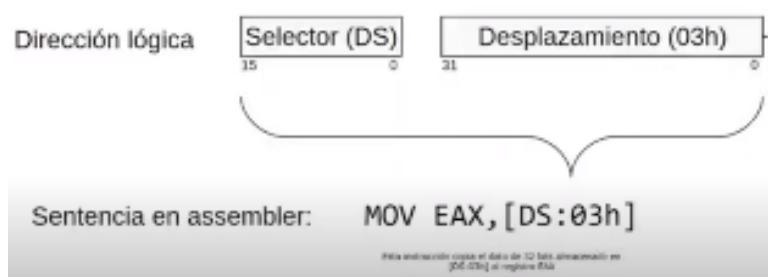
La paginación permite implementar un sistema de memoria virtual en el cual el entorno de ejecución del programa se mapea en la memoria física a demanda. Este mecanismo también permite aislación entre múltiples programas.

El mecanismo de direccionamiento en IA-32 implica dos procesos, la segmentación y la paginación, mientras que la primera transformación es obligatoria y siempre va a existir, la paginación es opcional y puede deshabilitarse.

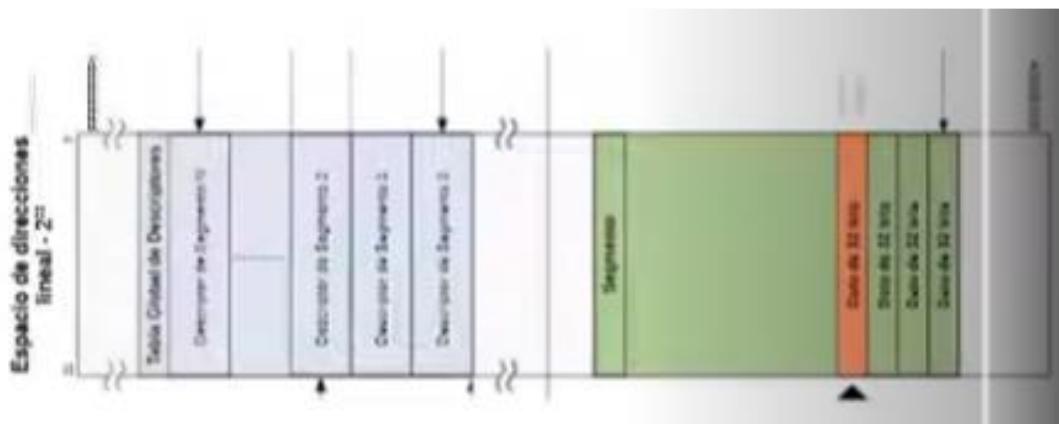
## MOV EAX, [DS:03h]

Esta instrucción copia el dato de 32 bits almacenado en [DS:03h] al registro EAX

El ejemplo lo vamos a ver a través de una sentencia en Assembler específicamente la instrucción MOV, esta instrucción lo que hace es copiar desde una posición de memoria un dato al registro interno del CPU EAX.



La dirección lógica presente en esta instrucción está conformada por dos partes por un lado un selector DATA SEGMENT en este caso, y un desplazamiento.



Observamos aquí el espacio lineal de direcciones que es el espacio de direcciones que tiene a su disposición el procesador.

## Espacio de direcciones lineal - 2<sup>32</sup>

Está compuesto por dos a la 32 posiciones es decir 4 Gb.

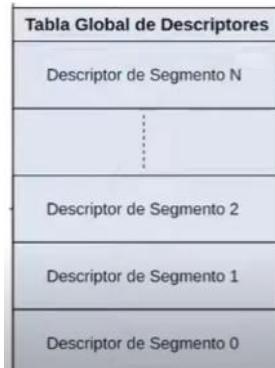
El espacio de direcciones  
lineal contiene todos los  
segmentos y tablas  
definidos para un sistema

En este espacio direcciones va a contener todos los segmentos de datos, de código, de pila y todas las tablas globales y locales definidas para un sistema, todo va a estar almacenado dentro de este espacio lineal de direcciones.

Es una abstracción que  
presenta al procesador una  
memoria única y continua

Sin embargo este espacio de direcciones es una abstracción, no está implementado físicamente, es la memoria como la ve el procesador y la ve como una memoria única y continua a pesar de que en la realidad podría no ser así.

El espacio lineal de direcciones comienza en la posición cero 00000000h y se extiende hasta la posición 2<sup>32</sup> menos 1 FFFFFFFFh.

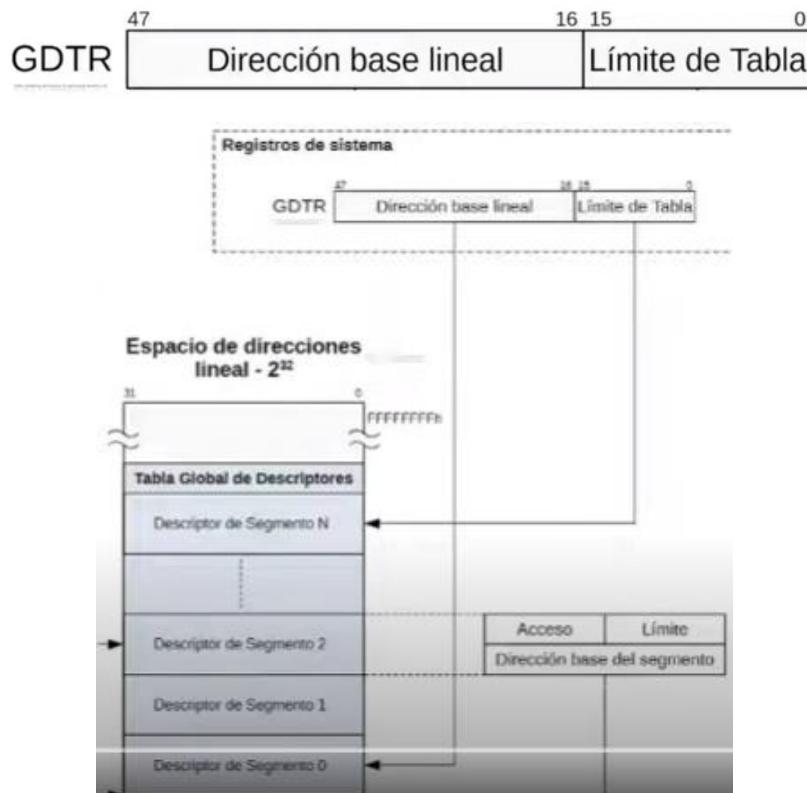


Dentro del espacio lineal de direcciones, para este caso particular, vamos a tener almacenada la tabla global de descriptores, esta tabla almacena “N” descriptores de segmentos desde 0 hasta “N”, el primer segmento, el cero, no es utilizado por las aplicaciones.

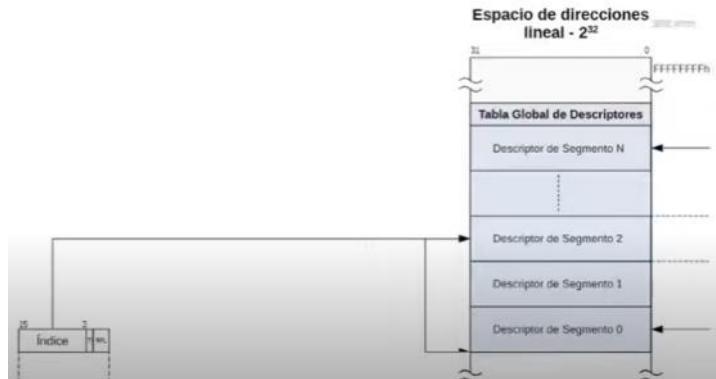
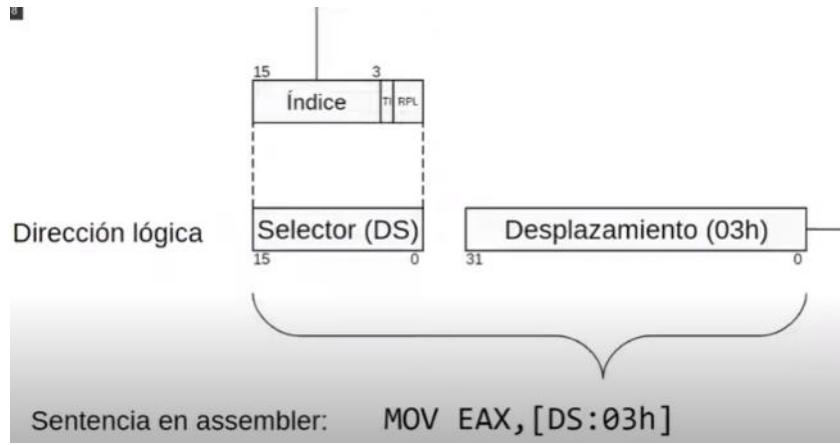
# GDTR

Este registro almacena la dirección lineal y el tamaño de la Tabla Global de Descriptores

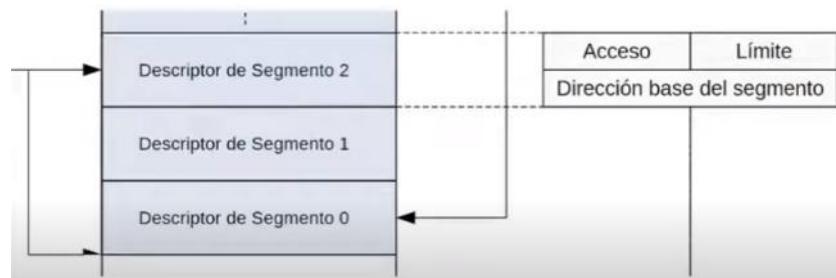
La ubicación de la tabla global de descriptores está determinada por el registro GDTR.



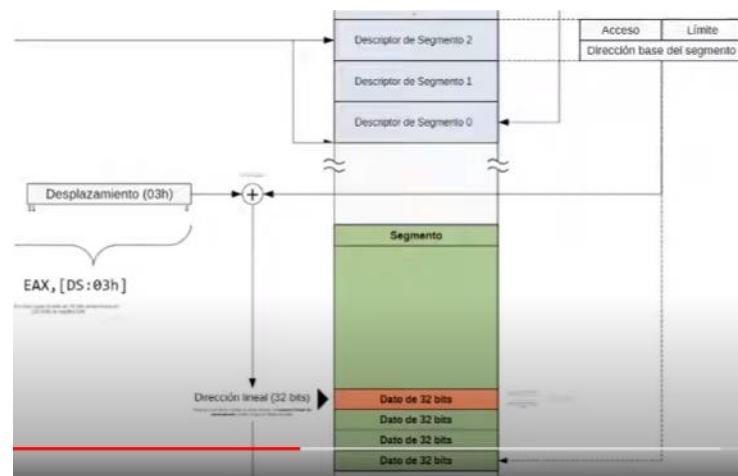
Este registro me indica la dirección base lineal, que es donde comienza la tabla global de descriptores y el límite de la tabla.



Volvemos a la dirección lógica y vamos a analizar el selector. El selector tiene un campo llamado índice que me va a indicar dentro de la tabla global de descriptores cuál es el descriptor de segmento que tengo que utilizar para el direccionamiento.

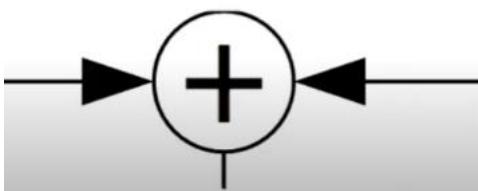


Este descriptor de segmento contiene información sobre el segmento al que yo quiero acceder.



En particular la dirección base del segmento que me indica dentro de la memoria lineal donde comienza este segmento.

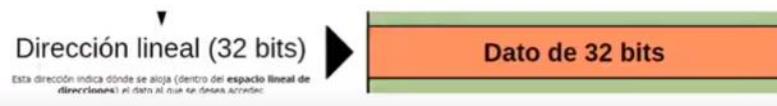
Sumando la dirección lineal base del segmento con el desplazamiento se obtiene la **dirección lineal**



Sumando la dirección lineal base del segmento con el desplazamiento se obtiene la **dirección lineal**

## Dirección lineal (32 bits)

Esta dirección indica dónde se aloja (dentro del **espacio lineal de direcciones**) el dato al que se desea acceder.



Si tomo esta dirección base y la sumo al desplazamiento que se encuentra en la instrucción en Assembler voy a obtener la dirección lineal donde se almacena el dato al que quiero acceder.

Si solo se estuviera usando segmentación,  
la dirección lineal obtenida se usaría  
directamente para direccionar la memoria  
física.

Vale aclarar que para que el sistema  
funcione apropiadamente en este caso,  
para cada dirección lineal utilizada tiene  
que existir una dirección de memoria física  
disponible.

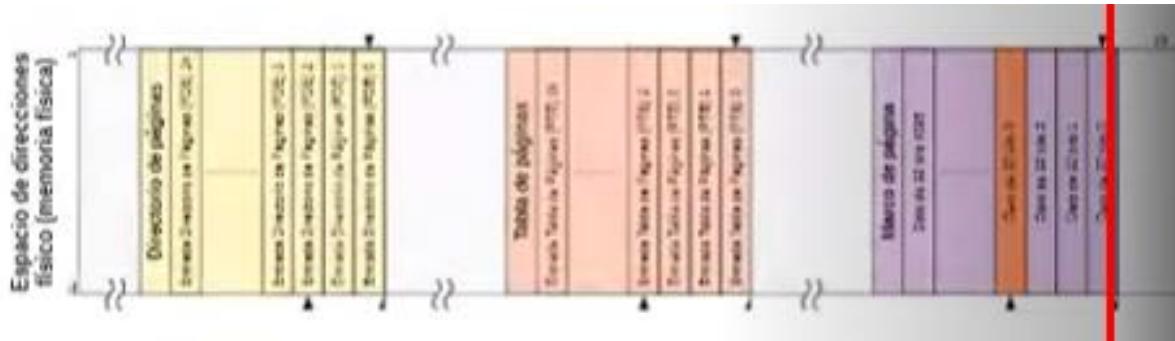
Si solo estuviéramos usando la transformación de segmentación, esta dirección lineal se transforma directamente en una dirección física para direccionar directamente la memoria RAM. No está de más aclarar que para qué el sistema funcione de esta manera la memoria RAM en esa dirección tiene que existir porque si no el sistema fallaría.

En el caso de utilizar segmentación y  
paginación, la dirección lineal sufre una  
nueva transformación

En el caso de que estemos utilizando además paginación la dirección lineal que hemos obtenido va a sufrir una nueva transformación.



La dirección lineal se va a separar en tres campos, directorio, tabla y desplazamiento.



Lo que vemos aquí es el espacio de direcciones físicas y la memoria física efectivamente implementada en el sistema.

Directorio de páginas
Entrada Directorio de Páginas (PDE) 1k
Entrada Directorio de Páginas (PDE) 3
Entrada Directorio de Páginas (PDE) 2
Entrada Directorio de Páginas (PDE) 1
Entrada Directorio de Páginas (PDE) 0

Dentro de este espacio de direcciones vamos a almacenar un directorio de páginas.

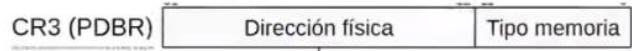
Tabla de páginas
Entrada Tabla de Páginas (PTE) 1k
Entrada Tabla de Páginas (PTE) 3
Entrada Tabla de Páginas (PTE) 2
Entrada Tabla de Páginas (PTE) 1
Entrada Tabla de Páginas (PTE) 0

Una o más tablas de páginas

Marco de página
Dato de 32 bits 4095
Dato de 32 bits 3
Dato de 32 bits 2
Dato de 32 bits 1
Dato de 32 bits 0

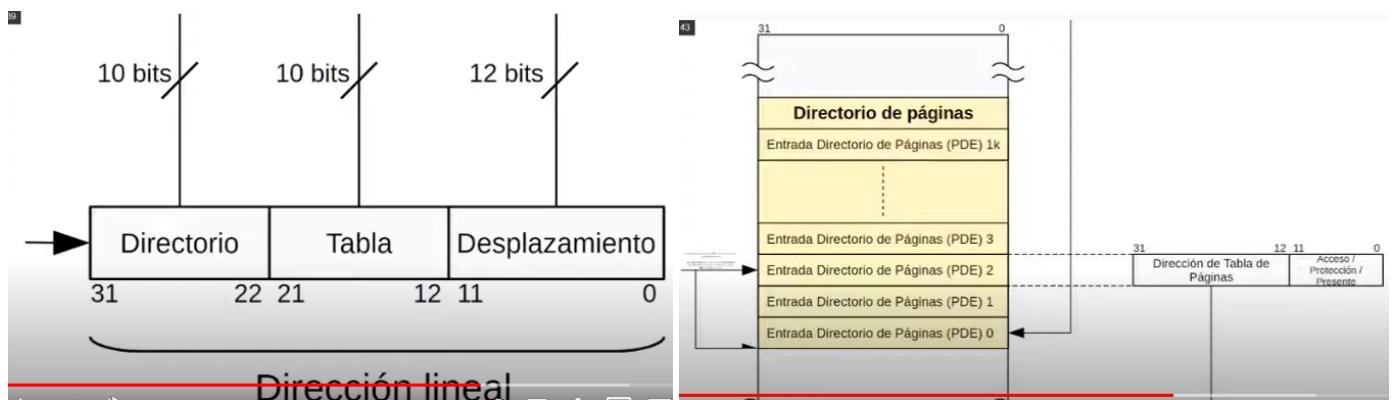
Y varios marcos de página.

# CR3 (PDBR)



Este registro almacena la dirección base física del directorio de páginas

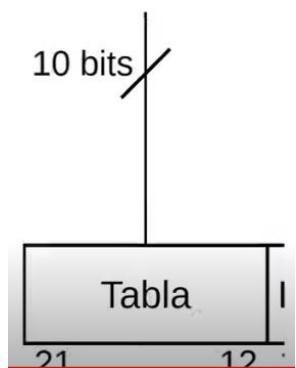
El CR3 (PDBR) me indica la dirección física del directorio de páginas



Si tomo esta dirección física y la combinó con el campo directorio de la dirección lineal voy a obtener una entrada del directorio de páginas.



A su vez esta entrada al directorio de páginas va a contener la dirección física de una tabla de páginas.



Si convino esta dirección física de la tabla de páginas con el campo tabla de la dirección lineal voy a obtener una entrada de la tabla de páginas y a su vez me brinda una dirección de marco de página una dirección física de marco de página.

La dirección física del dato se conforma de la siguiente manera:

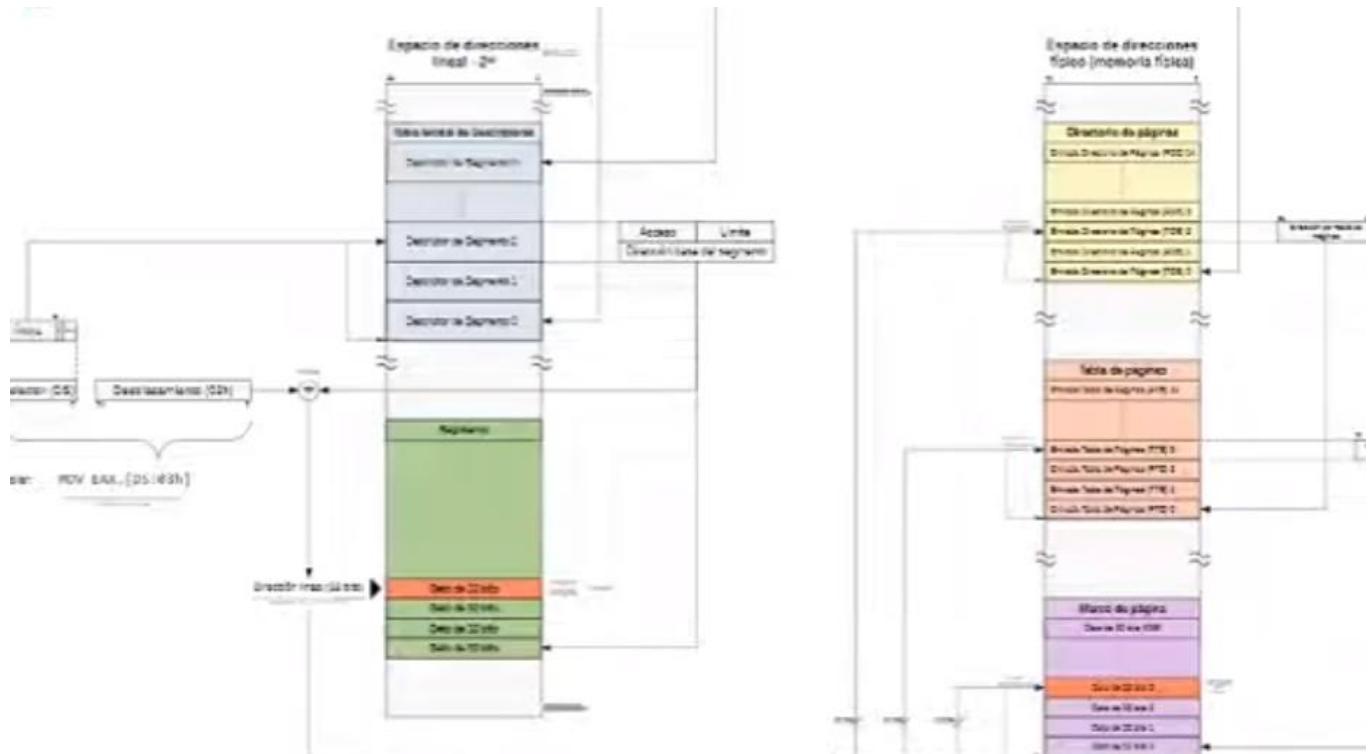
Bits **31 al 12**: Dirección de marco de página (PTE)  
Bits **11 al 0**: Desplazamiento (en dirección lineal)

Finalmente si combino esta dirección física de marco de página con el desplazamiento presente en la dirección lineal voy a obtener la dirección física del dato al que quería acceder en la instrucción de Assembler.

En el caso que el marco de página no se encuentre en memoria, se generará una excepción y una rutina de servicio la cargará desde el almacenamiento secundario.

Finalmente se almacena el dato en el registro EAX

Puede existir la posibilidad de que el marco de página al que yo deseo acceder no se encuentre en la memoria RAM, en este caso se ha producido una excepción una rutina de servicio a leer ese marco de página desde un almacenamiento secundario y lo va a cargar en la memoria RAM, posteriormente el acceso continúa de manera normal se lee el dato y para nuestro ejemplo el dato leído se va a almacenar en el registro.



Los invito a que tomen esta presentación se puede navegar libremente y recorran nuevamente los dos procesos de transformación realizan todas las consultas que consideren necesarias les recomiendo que pongan especial atención en diferenciar que hay dos espacios de direcciones dentro de este proceso, un espacio dirección es lineal que es la memoria como la ve el procesador y hay un espacio direcciones físico que es la memoria RAM que yo efectivamente tengo disponible en el sistema.

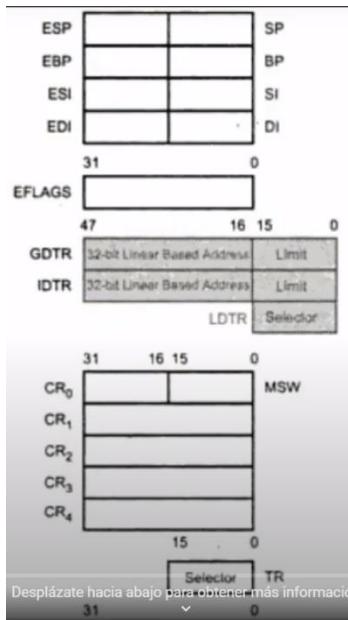
## MODO PROTEGIDO – SEGMENTACION Y PAGINACION

*La semana pasada habíamos visto básicamente una estructura de la IA 32, que cosas tenía dentro.*

*la clase pasada conté que había un módulo que era de ejecución predictiva en función de saltos condicionales y por otro lado hay una parte que todavía no hemos visto que es de protección, entonces eso lo que hacía era intentar saltar a una posición de memoria que no correspondía, o sea que no tenía atributos o permisos para acceder, que pueden ser de otros procesos, de otros usuarios o del KERNEL, entonces mientras estaba decidiendo si saltaba a esa posición o no iba a buscar a la memoria, la dejaba en la caché y cuando llegaba a decodificar y ejecutar esa instrucción que hacia el salto condicional si iba a esa dirección que no estaba permitida lo que hacía el programa es dar una excepción y termina.*

*pero lo malo es que ya en la pre búsqueda había cargado esa posición en la caché, entonces si hacía un programa muy sencillo que hace un DAM de la caché con eso podía obtener los valores que tenía, o sea nunca pude ejecutar la instrucción que yo no podía hacer, todavía no hemos visto esa parte pero se llama protección, pero al haber ejecutado instrucciones por adelantado o haber hecho en realidad búsquedas por adelantado lo guardara temporalmente ahí, entonces por más que termine el programa, podía acceder esos datos después desde cualquier otro programa leyendo esa caché. Básicamente de esos se valían estas vulnerabilidades, y el tema es que casi todos los procesadores para que sean rápido tratan de ir adelantando lo más que puedan las ejecuciones, entonces por eso prácticamente todos los procesadores tenían ese problema. Tenía un poco que ver con lo que hablamos de arquitectura, de la caché y de todo eso, por eso digamos venía el caso.*

Hemos visto que básicamente cuando arranca la arquitectura IA 32 lo arrancaba casi como un 8088, una máquina con un direccionamiento de 1MB, incluso ni siquiera de manera continua sino que era de a pedacitos de 4K.



Si no me equivoco hay un BIT (en los registros internos del procesador) que está en el registro CRO, el primer bit o alguno de ellos no me acuerdo cuál, si lo pongo en 1 paso a un modo de funcionamiento que se llama modo protegido, donde tengo todas las funcionalidades, puedo direccionar 4GB de RAM, y tengo visibilidad de algunos registros por ejemplo los que están en gris, que ahora puedo acceder en este modo, hay uno que se llama GDTR, otro IDTR y otro LDTR.

**GDTR: THE GLOBAL DESCRIPTOR TABLE REGISTER**

**IDTR: INTERRUPT DESCRIPTOR TABLE.**

**LDTR: LOCAL DESCRIPTOR TABLE.**

Otra de las características interesantes que tienen en modo protegido que aún no vamos a ver, es que además de poder direccionar 4GB de RAM, tengo algo que se llama protección o aislación entre tareas, significa que si tengo más de un programa corriendo es imposible que uno escriba o lea información del otro, tengo una funcionalidad que se llama protección y eso directamente por hardware si intento acceder a una posición de memoria que no es mía da una excepción hardware y se termina la aplicación, no puede acceder.

Otra cosa interesante es que tiene cuatro modos de funcionamiento, cuatro niveles de privilegio, donde hay un nivel que tiene acceso a todo el set de instrucciones, me refiero instrucciones de entrada salida sobre todo, mientras que el último nivel es el que no tiene acceso a todo eso, entonces sería una especie de set de instrucciones recortados para evitar que haga lío o que pueda hacer cosa entrada/salida y rompa alguna aplicación.

Entonces lo que pasa o lo que termina pasando normalmente es que los sistemas operativos trabajan en ese modo que se llama privilegiado o modo KERNEL donde tengo acceso a todo el set de instrucciones.

Cuando yo usuario escribo un programa termino trabajando en el modo menos privilegiado entonces no puedo hacer líos por más que quiera, además que tengo evitado escribir o leer posiciones de memoria que son de otro programa, tampoco puedo hacer entradas-salidas o instrucciones que modifiquen por ejemplo los registros de segmentos que eran el CODE SEGMENT, DATA SEGMENT, STACK SEGMENT, en total eran 6. En modo protegido siguen estando estos 6 pero si trabajo en el nivel menos privilegiado o modo usuario, no tengo posibilidad de modificarlo, entonces si no tengo posibilidad de modificarlo no voy a poder hacer lío ni escribir en el lugar de otro programa. Estas básicamente son las ventajas de trabajar en modo protegido.

Lo que hoy quiero que trataremos de entender es cómo se direcciona la memoria, antes era fácil, supongan que voy a direccionar una a posición de memoria para buscar instrucción, tengo el CODE SEGMENT que era de 16 bits, lo desplazaba 4 lugares, y después le sumaba el offset, que el offset en este caso era el INSTRUCCIÓN POINT, el contador del programa, eso se hacía en modo real y era la dirección física. Ese era el MEGA que podía direccionar pero obviamente una vez que yo fijaba el CODE SEGMENT tenía nada más que 64K.

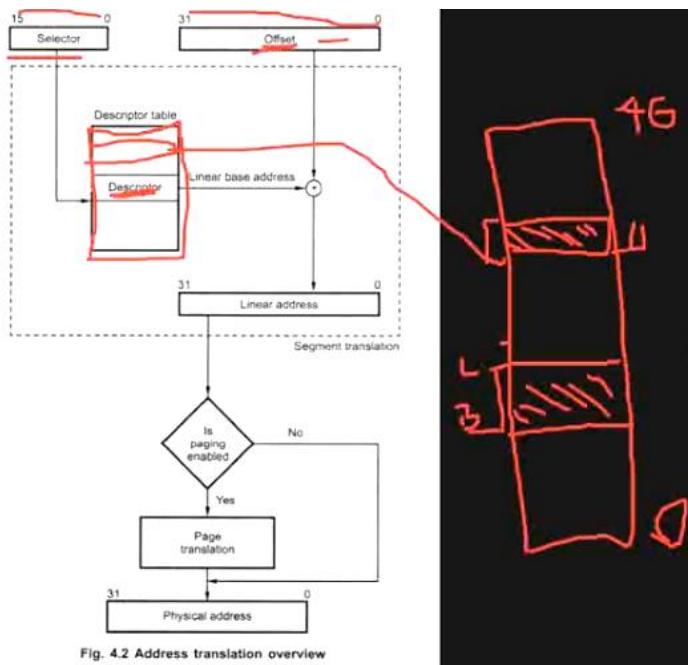
En modo protegido puedo acceder a 4GB, los registros de segmentos siguen siendo de 16 bits, ahora la diferencia es que el desplazamiento o sea el offset son de 32, por ejemplo el con el STACK POINTER ahora se llama EXTENDED STACK POINTER, y en vez de tener 16 tiene 32Bytes como se ve, lo mismo para el contador del programa que creo que no está acá, lo mismo que para el resto de los registros, excepto los de segmentos que siguen con 16 los otros no.

En vez de tomar en el segmento y desplazarlo 4B, ahora tengo un segmento que sigue teniendo 16 bits pero un offset de 32.

En vez de desplazarse, tiene un selector, una tabla que se llama tabla de descriptores, que se encuentra ubicada en la memoria principal en la RAM, es una tabla donde tengo información, se encuentra agrupada de a varios Bytes que se llaman descriptores, básicamente lo que describen son partes, segmentos de memoria, es decir, en RAM tengo "N" descriptores, contienen la dirección base, algunos tienen un offset y también tienen algo que tiene que ver con los niveles de privilegio que veremos luego al ver protección.

Cuando en mi CODE SEGMENT pongo un valor, apunto dentro de esa tabla que está en RAM, y me mapea la dirección inicial, un límite de un segmento de código, y algunos bits más de privilegio.

Por cada segmento que tenga la memoria principal voy a tener que tener un descriptor. Con esto ya tengo información de donde arranca el segmento, y lo que hace es agregarle el offset, por hardware tengo una protección si me salgo de los límites de la memoria asignada.



Cuando hago PUSH va subiendo y cuando hago POP voy bajando, ese que va incrementándose que sube y baja es el offset, que sería el STACK POINTER.

En el caso que yo tenga esta área, donde tengo mi programa mi código, el selector va a apuntar un descriptor que va a ser de mi segmento de código, va a tener información de donde arranca, dirección base, y cuál es el límite que va a tener, con el contador de programa sabré que instrucción voy a ejecutar después, suponiendo un programa secuencial irá incrementándose, se irá leyendo a cada una de estas instrucciones.

Si alguna instrucción tiene que poner o sacar algo en la pila lo que va a hacer es en el selector va a ir a parar al STACKS SEGMENT que seguramente va a apuntar a otro descriptor y ese me va a definir o describir un área de memoria que está en otro lado que tiene una base y un límite y el que va a ser el offset en ese caso va a ser el puntero de pila STACK POINTER.

De esta manera es una especie de direccionamiento indirecto a través de una tabla que yo previamente cargó en RAM, de alguna manera tengo mapeado todo mi espacio de dirección que son los 4 GB, donde tengo mis datos, mi código, mi pila y hay un par de segmentos más extras que se pueden usar para otra cosas.

Me genera una dirección de 32 bits que normalmente se llama lineal, arriba donde está el procesador se llama direccionamiento virtual, o la dirección lógica o virtual, que es lo que el procesador cree que direcciona, desde el punto de vista el procesador él se cree que tiene por ejemplo un área de memoria única para su programa, un espacio de direcciones distintas para su pila, otro para los datos, por en definitiva van a para todos a una única memoria unidimensional, entonces arriba tengo lo que el procesador cree que ve, en el medio lo acomodo todo una sola dirección lineal, de 0 a 4 GB.

Esto es obligatorio de alguna manera en Intel en el modo protegido, hago si o si traslación de direcciones usando segmentación que se llama esta técnica de traslación, traslado lo que el procesador ve que son "N" espacios distintos, uno para cada segmento. Por ejemplo, en la posición 100 del segmento de código va a parar a otro lugar que la posición 100 de los datos, que la posición 100 de la pila, como les digo el procesador creería o podría creer que tiene como "N" BUS distintos, un BUS para el programa, otro para los datos, otro para la pila otro para un segmento extra y cosas por el estilo.

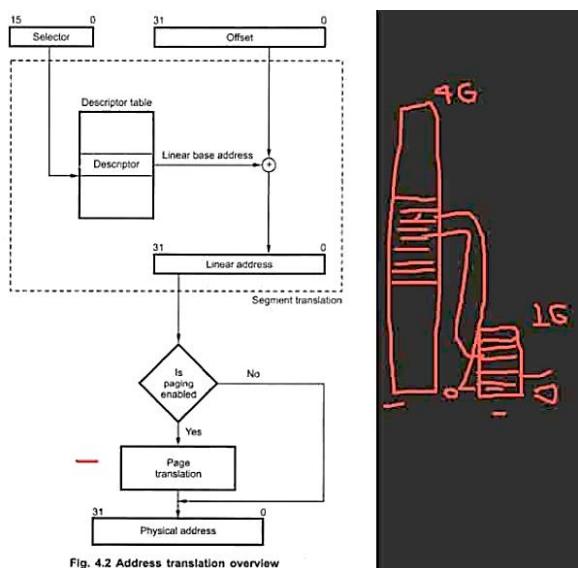
Una vez que al hacer la transacción se pone todo en la lineal queda todo en un solo espacio direcciones.

En resumen habíamos quedado que como primera instancia cuando está en modo protegido sí o sí ocurre una traslación de direcciones a partir de un direccionamiento lógico virtual que está en el procesador donde cree que tiene espacio de direcciones y se acomodan todos en una única memoria lineal de 32 bits, de ahí esta traslación que es **segmentación**, se obtiene una dirección lineal de 32 bits que es la segmentación.

Eso quedaría ahí y se terminaría el tema de direccionamiento de la memoria en modo protegido a no ser que yo también pueda hacer una segunda traslación y eso se llama **paginación**, si tengo algunos bits de esos registros de control CR habilitados voy a tener que hacer otra traslación, caso contrario estos 32 bits son los que van directamente al Bus de direcciones, esta sería la memoria física, en definitiva la arquitectura de Intel tiene una memoria virtual que ve el procesador, una lineal una vez que se hizo la segmentación y una física si es que hay paginación, que es la segunda traslación, en caso que no haya traslación la dirección lineal equivale a la física.

¿Porque una segunda traslación? Suponiendo que tengo un programa que ocupa en memoria un montón de espacio de memoria RAM, Office por ejemplo, para ejecutar un programa tiene que estar en la memoria principal, y si tenemos 1GB de RAM y queremos ejecutar Office no nos va a quedar lugar para casi nada más, entonces a alguien se le ocurre decir, bueno que pasa así al programa no lo ponemos entero en la memoria, lo vamos cortando de a pedacitos y a medida que intente acceder una posición de memoria que no está, ahí si lo ponga en la memoria principal, y si hay un pedazo de mi programa que está en la memoria principal y hace mucho que no se usa lo podría sacar, eso me permitiría no ocupar tanta memoria principal. Muy conceptualmente de eso se trata la paginación.

La primera traslación de segmentación también es una técnica de memoria virtual y la de paginación también, o sea estamos hablando de memoria virtual, significa por ejemplo en este caso yo le hago creer al procesador que tiene "N" segmentos y con un largo determinado, sin embargo van a parar a una única memoria lineal donde están organizados de esta manera, varios segmentos y no se solapan, el procesador va a creer que nunca se solapan porque tiene un selector y un offset para cada uno de esos segmentos, sin embargo en la memoria principal se podrían solapar, el cree o le hago creer que su memoria virtual son "N" espacios distintos, él podría creer que tiene un espacio de direcciones de 0 a un valor determinado que es para él el código, tiene otro espacio de direcciones que será más grande para los datos, otro chiquito para la pila y algún otro más y eso es lo que se cree el procesador.



Suponiendo que tengo una memoria lineal (4GB) que estoy viendo y la memoria física que tengo (1GB en este caso), la memoria física la separó en pedacitos iguales de Intel, para Intel puntualmente de 4k y se llaman **marco de página** es donde se van a alojar las páginas, por otro lado la memoria lineal que tengo, suponiendo que tengo un segmento de código también los separo en pedacitos de 4k y se van a llamar **páginas**, entonces cada vez que yo necesite ejecutar una instrucción que esté dentro de esa página, la voy a elegir y la voy a guardar en un marco de página libre, como la página de marco tienen el mismo tamaño 4k, muevo los 4K de una a la otra, a la memoria física y listo, si en algún momento necesito ejecutar otra parte del código si no está cargado lo voy a cargar, o sea una página la mete en un marco libre y si no tiene que hacer lugar y conseguir uno libre.

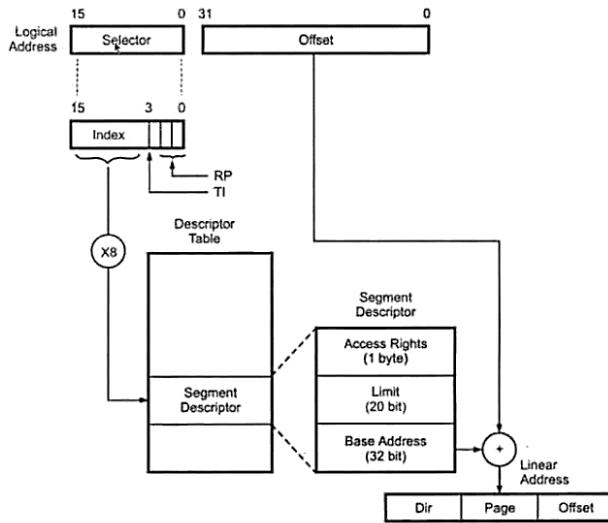


Fig. 4.3 Segment translation mechanism

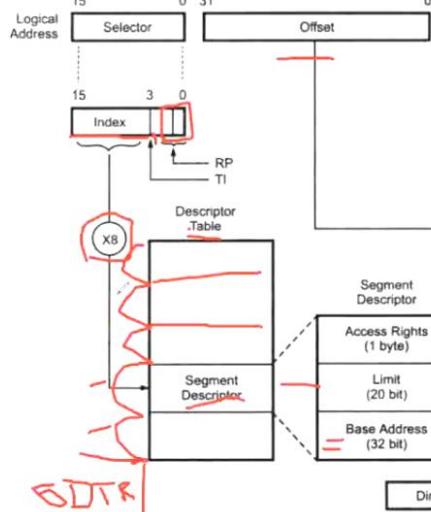
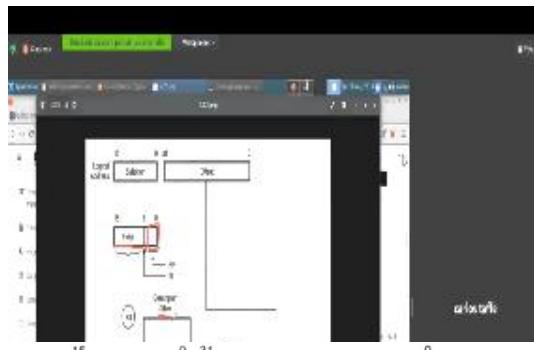
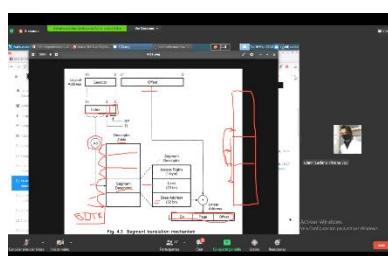


Fig. 4.3 Segment translation mechanism

Cuando hago la segmentación o la primera traslación, que esa sí o sí es obligatoria en modo protegido, de los 16 bits tengo un par de bits que tienen que ver con privilegios que hoy no lo vamos a tocar, tiene un bit que me dice que tabla de descriptores es y después el resto me sirven para mover el offset dentro de mi tabla de descriptores.

Tengo una única tabla de escritores globales en el sistema cuando arranco que está puesta en una posición de memoria determinada y acá tengo apuntando a mí GDTR, un registro que tiene 32 bits que me dice de toda la RAM a partir de dónde está mi tabla de descriptores, entonces yo con el selector como hicimos recién selecciona uno de los "n" registros que hay acá dentro de descriptores, tengo N registros de descriptores acá adentro. Dice X8 porque la tabla de descriptores utiliza 8 Bytes, son 3 bits así que tiene un ancho de 8 Bytes, son 64 bits en total de información, y esa tabla tiene tres cosas importantes, primero una dirección base que habíamos quedado que eso me dice de la dirección lineal en qué parte arranca, me da un límite que esto me dice cuánto podría ser de grande y tiene un par de

bits que son de permisos que los veremos cuando veamos el tema de protección, entonces lo que hace después como dijimos recién le suma el offset, esto es muy similar a lo que vimos recién nada más que aclara que acá se va saltando en esa tabla de descriptores global como de a 8 Bytes, porque el ancho de cada descriptor tiene 8 Bytes, entonces si direccionalo en el índice el primer descriptor que sería el 0 apunta a 8 Bytes, si le digo que es el 1 apunta del byte 8 al 15 que son los segundos 8 Bytes y así sucesivamente.



Cuando se obtiene la dirección lineal obviamente se hace un control de hardware que el offset no pase el límite, si pasa el límite es una excepción y estoy yéndome por arriba o por abajo de mi segmento, no podría leer ni escribir en esas posiciones de memoria.

Lo interesante de este gráfico es que TI me dice si es una tabla global o local, como les decía hay una sola tabla global en el sistema, pero pueden haber una o muchas tablas de descriptores locales, entonces TI me va a decir cómo acceder, si es global o el local, si es global es fácil porque accedo a partir del registro GDTR, si es local lo vamos a ver más adelante.

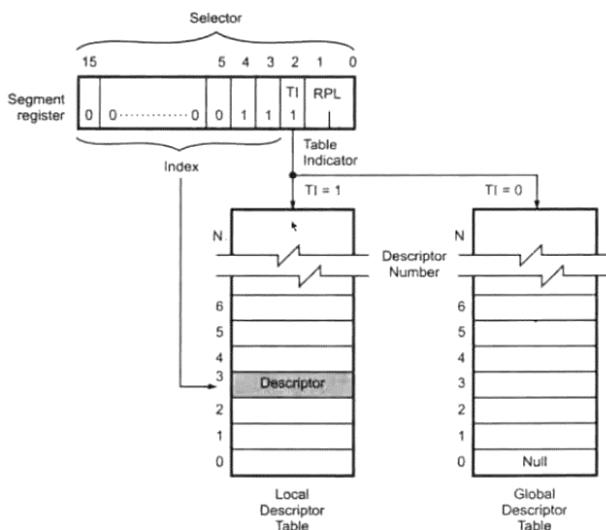


Fig. 4.4 Selector and descriptor tables

SEGMENT BASE 15 .... 0										SEGMENT LIMIT 15 .... 0										0	Bytes
BASE 31 .... 24										P	DPL 7	S 6	TYPE 5	A 4	0 3	1 2	1 1	0 0	BASE 23 .... 16	+ 4	4
Access Rights Bytes																					8
BASE	Base Address of the segment																				
LIMIT	The length of the segment																				
P	Present Bit : 1 = Present 0 = Not present																				
DPL	Descriptor privilege Level 0 - 3																				
S	Segment Descriptor : 0 = System Descriptor 1 = Code or Data Segment Descriptor																				
TYPE	Type of segment																				
A	Accessed Bit																				
G	Granularity Bit : 1 = Segment length is page granular 0 = Segment length is byte granular																				
D	Default Operation Size (recognised in code segment descriptors only)																				
0	1 = 32-bit segment 0 = 16-bit segment																				
AVL	Bit must be zero (0) for compatibility with future processors																				
Note :	In a maximum - size segment (i.e. a segment with G = 1 and segment limit 19 .... 0 = FFFFFH), the lowest 12 bits of the segment base should be zero. (i.e. segment base 11 .... 000 = 000H).																				

Fig. 4.5 General Segment Descriptor Format

manera distinta, algunos que crezcan hacia arriba digamos en la dirección y otro que crezca hacia abajo.

Este es el formato genérico de un descriptor de segmento, hay distintos segmentos, para datos para pila, para el sistema operativo entonces por ahí el formato interno cambia, pero básicamente las tres cosas importantes que tiene son la base, el límite y los permisos, estos dos bits son porque en este caso de Intel tengo cuatro niveles de privilegio.

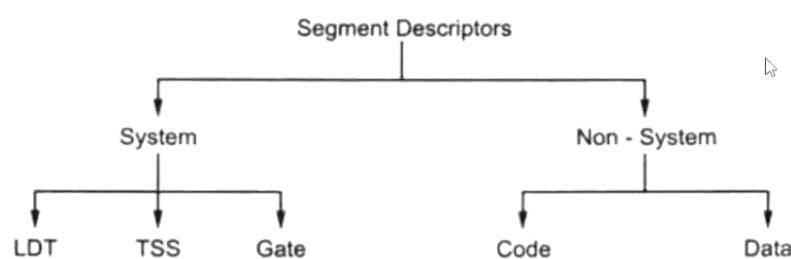


Fig. 4.6 Types of segment descriptors

Este es un ejemplo de un descriptor, esos 8 bytes como están separados. Por ejemplo la dirección base que es una dirección de 32 bits, hay un poco del 0 al 15, del 16 al 23 y del 24 al 32, todo esto me da la dirección línea base donde arranca el segmento, después el límite que eran 20 bits, así que tengo 16 más 4 da 20. DPL que son niveles de privilegio que eran 2 bits, después tengo TYPES que me indica que tipo de descriptores son, vamos a ver que son varios, y después tengo G que es granularidad, el límite puede ser en bytes o en segmentos, y alguno más que me dice el orden en el que crece, si asciende o descende el segmento, vamos a ver que se podrían acomodar de

Acá muestra a todos las distintas posibilidades de descriptores que tengo, no hemos visto todavía casi ninguno, pero dentro de la tabla de descriptores globales podría tener un descriptor del área de código, otro de datos, son distintos luego una entrada en la tabla global de un descriptor local, una entrada en las tablas de descriptores globales de segmentos de estado de tareas, y otras que se llaman puertas, o sea

puedo tener varias formas y varios formatos de descriptores, pero defendiera todos describen área particulares para distintos tipos de cosas, no vamos a profundizar en la diferencia, no tiene sentido saber cómo están internamente.

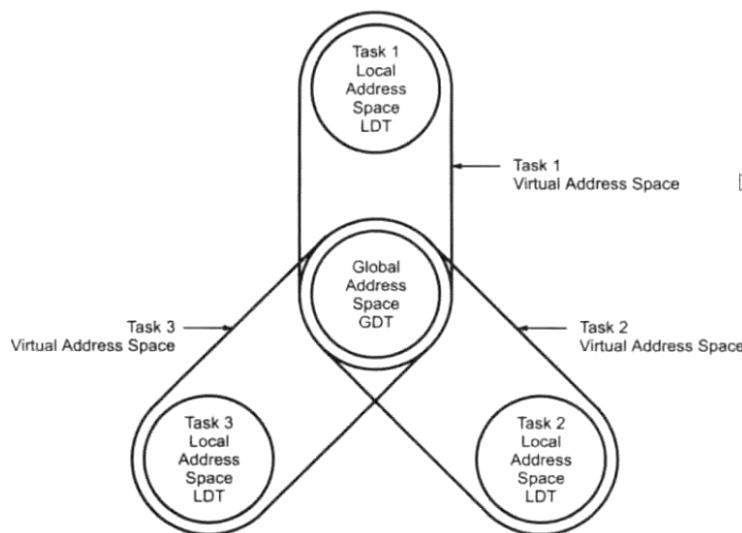


Fig. 4.12 Memory area shared by different tasks

Esto es lógicamente lo que comentábamos de las tablas locales y globales, cada tarea, cada proceso o cada programa puede tener o va a tener siempre acceso a una tabla de escritores globales que es única para el sistema, pero una tarea podría tener también una tabla de descriptor local, es local para esa tarea, entonces podría encontrarse la tarea 3 que tiene una tabla descriptores locales más la global, la 1 tiene la global más otra tabla local y la 2 también. La tarea podría tener su tabla de descriptores locales.

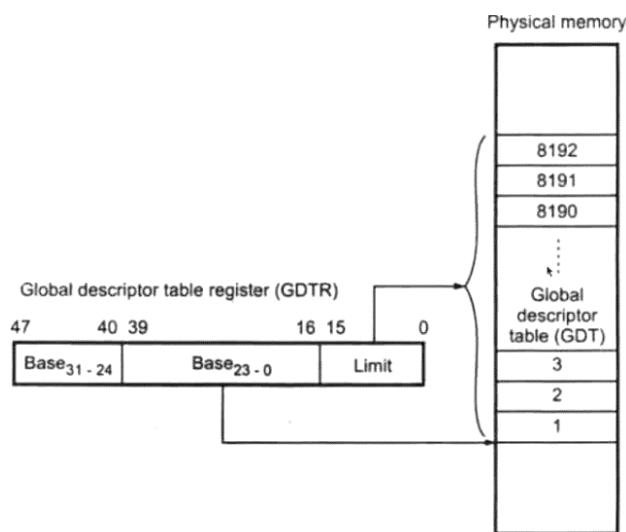


Fig. 4.13 GDTR and GDT

La pregunta sería en que posición de memoria está la tabla de descriptores globales que es única, tengo un registro que se llama GDTR que tiene 32 bits, apunta a la base, a partir de ahí está el primer descriptor de segmento, tiene también 15 bits que me dicen el límite, significa que puedo tener hasta solo 8 k de descriptores del segmento.

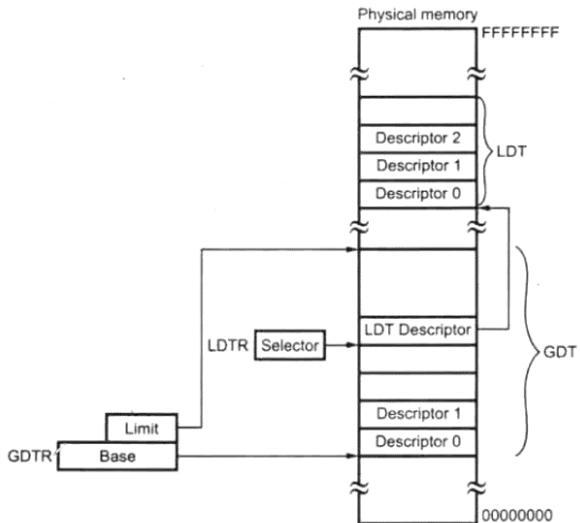
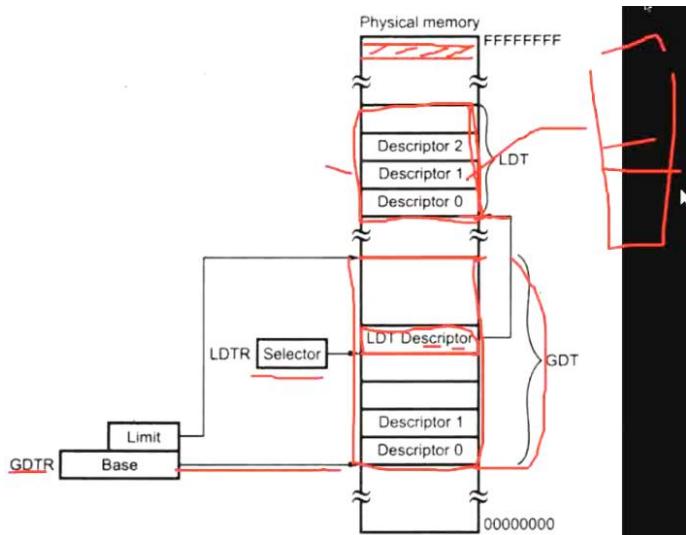


Fig. 4.15 Global and local descriptor tables

Bien y en el caso de que haya seleccionado, en el selector en vez de la tabla descripciones globales, si estuviera un 1 en ese bit, tengo que acceder a una tabla descriptores locales, el procedimiento es bastante distinto, ¿Cómo la encuentro?, como siempre el GDTR señala la base de mi tabla de descriptores globales, cuando tengo un beat 1 en el selector entonces lo que hago es en vez que con el offset del selector o el índice del selector los 13 bits más altos del selector en vez de buscar el descriptor lo que hago es usar el otro registro que se llama LDTR que también es un registro físico pero que tiene 16 bits lo uso como offset y lo que encuentro es un descriptor de un área de memoria, de un segmento, pero ya no es un segmento de datos, de código ni nada de eso, sino acá tengo información de la dirección base y el offset de una tabla de descriptor local, la base y el límite delimitan 4 descriptores.

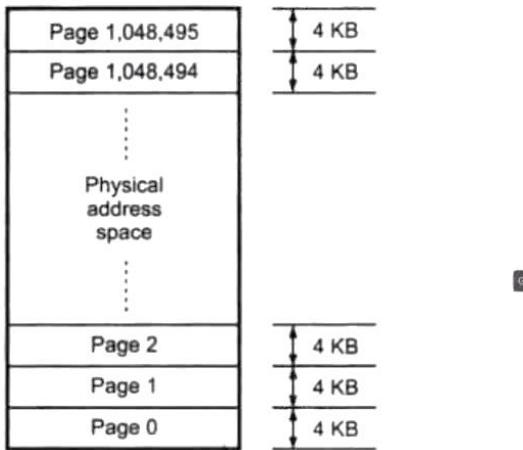


Entonces ya tengo la base y lo que tengo que hacer ahora es con el offset del selector elegir el descriptor que yo quiera, y ahí recién me está mapeando a un área memoria de la memoria principal. Con esto podría ampliar la tabla de descriptores, que es el objetivo, si no me alcanzara los 8K de descriptores.

En el selector los primeros bits se usaban como índice, índice, si el bit me decía que era una tabla global, listo, tengo acá el descriptor de la tabla, la base de la tabla de descriptores globales y ese índice lo usaba y accedía al descriptor en cuestión, si tenía la mala suerte de una tabla local lo que hacía es, en la tabla de descriptores globales usaba el registro selector como índice, una vez que obtenía la descripción de donde estaba mi tabla de descriptores locales lo que hago ahora es el índice lo uso pero en esa tabla y finalmente el descriptor va a apuntar al área de memoria, es como un segundo proceso de traslación.

Hasta acá ha sido todo segmentación, sin tener paginación, y así como en las imágenes anteriores estaría mi memoria física.

Si tengo paginación a esta traslación que ya hemos tenido le tengo que agregar una nueva traslación.



Acá se puede ver cómo está organizada la paginación, donde lo voy separando en bloques de 4K entonces la memoria virtual tiene páginas y la memoria física donde se van a alojar esas páginas tiene marcos de página.

Fig. 4.18 Paged organization of the physical address space

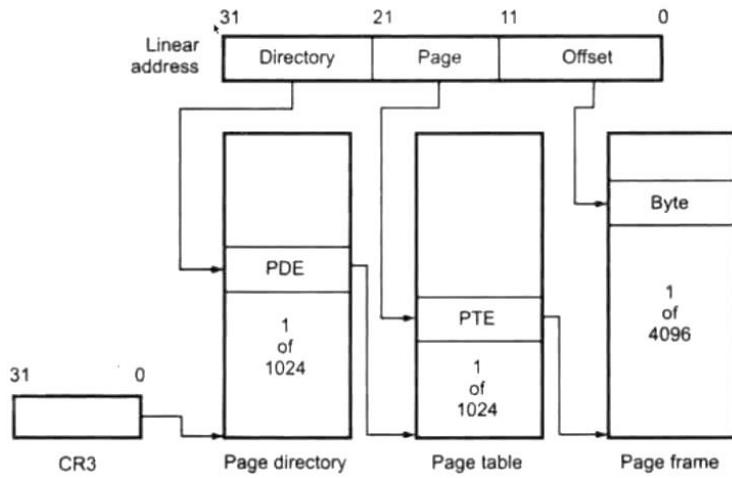


Fig. 4.20 Linear to physical address translation

Esta es la manera en la que funciona esa traslación, tengo la dirección lineal, los 32 bits, o sea esto sería lo que terminaba de dar cuando sumaba a la base de segmento el offset en la segmentación si estos 32 bits lineales, lo que hago es separarlo en tres pedazos a esta dirección de 10, 10 y 12, básicamente con los primeros 10 los uso como offset dentro de una tabla que también está en la memoria principal que se llama directorio de página, y con estos 10 bits puedo tener una entrada de obviamente 1 de 1024 posibles entradas de tabla de directorio, en este caso de la filmina hay una direccionada como ejemplo, y ésta tiene información de donde arranca otra tabla que se llama tabla de página en la memoria principal,

entonces tiene la dirección base de esta tabla de página, esa tabla de página otra vez con los 10 bits de la segunda parte de la dirección lineal la utiliza como offset y obtiene una entrada en la tabla de página que le llama PTE y eso tiene la dirección base de un marco de página que esto sí sería la memoria física, con este offset más la dirección base del marco de página yo me muevo algún byte de la memoria física al que yo quería, esta es la segunda traslación.

¿Dónde encuentro la tabla de directorio de página? la base de eso está apuntada por este registro CR3. A medida que necesitemos más podría tener directorios de páginas, pero como es una entrada de dos niveles, por cada entrada de la tabla del directorio página podría tener 1024 marcos, entonces tendría  $1024 \times 1024$  podría tener hasta 1MB de marcos de tabla de página, y por 4K que tiene cada marco, 1MB por 4K me va a dar los 4GB que tengo en total.

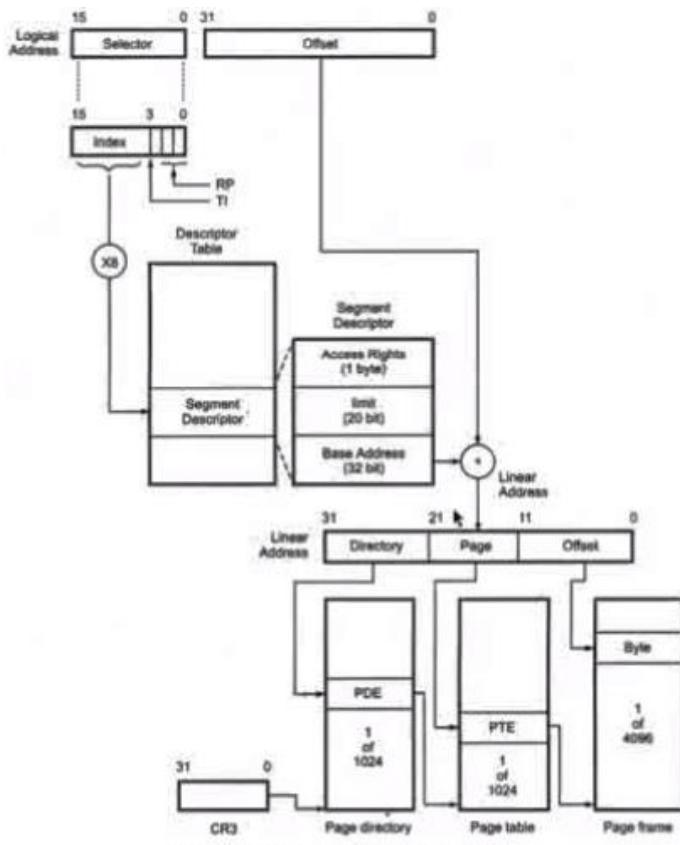


Fig. 4.24 Protected mode address translation

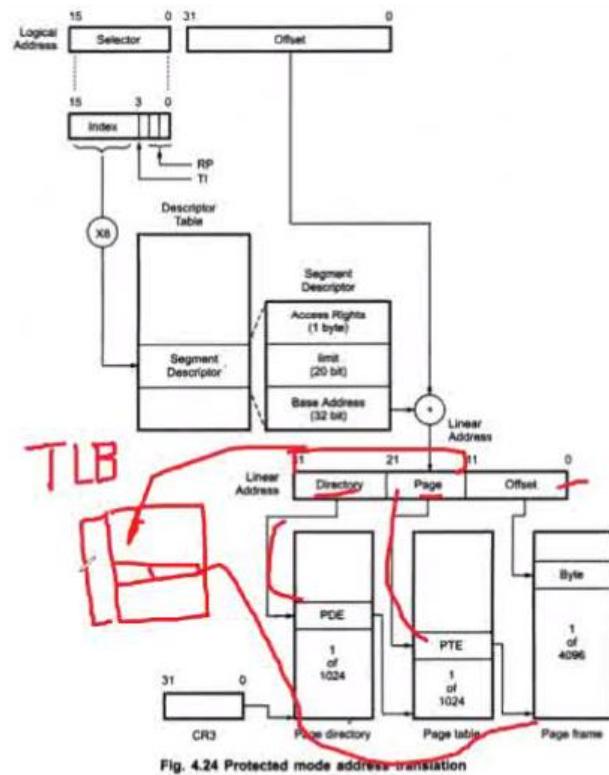
parte oculta de ese selector, o sea, un segmento de código donde puedo almacenar la información del último selector que estaba apuntando, entonces la primera vez si voy a tener que ir a buscar a la memoria principal pero después lo guardo en una parte que se llama oculta de estos registros de segmento, entonces a partir de ahí si yo no cambio el valor del CODE SEGMENT o del DATA SEGMENT, evito este acceso a memoria para obtener esta información, porque de alguna manera estoy guardando en la parte oculta la dirección base y el límite, me ahorro la segmentación.

Esto sería básicamente como es una traslación entera si usara segmentación y luego paginación, si esto sería el resumen. (Estaría faltando el registro GDRT solo que el gráfico no aclara si es para global o local).

Cada par Selector Offset me va a seleccionar un descriptor donde me va a dar información de una dirección lineal, dónde está la base, cuál es el límite y algunas cosas más, entonces ya tengo la base que la sumo con ese offset y me da una dirección lineal, después hago estas dos traslaciones más y finalmente voy a parar a un marco de página 4k de una memoria física y con los bits más bajos, los 12 bits (offset) puedo moverme entre una de las 4K de posiciones.

¿Cuántos accesos a memoria necesito por ejemplo para buscar la siguiente instrucción? Por ejemplo pongo un programa se ejecuta la primera instrucción en la posición 100, ¿cuántos accesos a RAM antes de ejecutar esa instrucción o antes de encontrar ese dato tengo que hacer?, 4 en total, al cuarto yo recién accedo el dato.

Para mejorar esto la paginación lo que hace es tener un cache, y en la segmentación por cada selector tengo una



Con respecto a la traslación que son dos, lo que hace Intel, tiene una memoria caché, pero no es una caché ni de datos ni de códigos, sino es una caché de traslación, esta caché que es una memoria asociativa, le entro con 20 bytes que son los del directorio de página, si encuentra una coincidencia significa que anteriormente ya hice esta doble traslación, o sea porque accedí a esa tabla de directorio página y esa tabla página, y tiene asociado la dirección base del marco de página, entonces una vez que llegue la parte lineal, obviamente la cache tiene capacidad para X entradas, no tiene todas las posibles combinaciones, 4K de entrada algunos modelos de Intel, le accedo con 20 bytes de la dirección y página y si tengo coincidencia obtengo la dirección base, los 20 bytes los 20 más altos del marco de página a dónde está esa dirección lineal, entonces si obtengo eso directamente le sumo el offset y me ahorro de ir a RAM una vez, ir a RAM dos veces.

A esta caché le llama **TLB**, memoria de acceso secuencial más rápido, que puede tener la información que busco o no pero la información no es de un dato ni de una instrucción sino de una traslación de direcciones, se llama **TRANSLATION LOOKASIDE**

BUFFER.

Entonces la mayoría de las veces para acceder a una instrucción o un dato directamente hago un acceso a memoria y todos felices, en el caso que el TLB no tenga acierto, bueno mientras TLB busca secuencialmente intento también acceder a la memoria si no tuve suerte con el TLB accede una vez a la memoria luego accede una segunda vez a la memoria y por ultimo me da la dirección base del marco de página lo guarda en la caché si es que tengo un lugar con los 20 bits más altos me graba esa traslación, entonces me queda guardada en la caché para la siguiente vez que quiere acceder.

Generalmente cuando accedo a un dato o una instrucción lo que voy a hacer después es acceder a la instrucción que sigue o al dato que está contigo entonces desde ahí es que yo tengo siempre muy alto grado de aciertos en esa cache.

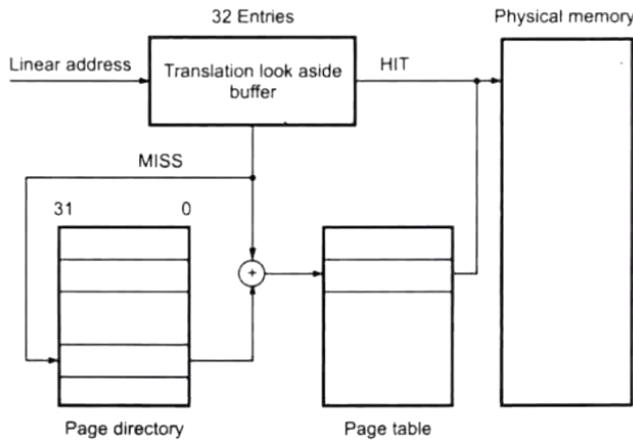


Fig. 4.25 Translation lookaside buffer

("para mí está mal este gráfico") pero básicamente lo que hace es que en paralelo busca el TLB mientras tanto va accediendo a la memoria principal buscando la entrada en la directorio de página, obviamente si tiene acierto descarta la búsqueda si no tienen acierto sigue, una vez que trae la dirección base de ese marco además de agregarla a la entrada del TLB, le suma el offset de los 12 bits más bajo de la dirección lineal y va a apuntar a una posición de memoria.

## Preguntas Cuestionario:

1\_ IA-32 en modo protegido usa: Segmentación y luego paginación.

2\_ Indique la opción verdadera para paginación en IA-32: Los marcos de página y las páginas tienen 4Kbytes.

3\_ Que ventaja trae una TLB: Almacena traslaciones de memoria por paginación.

4\_ Los 4 elementos más importantes de un descriptor son:

- El segmento de código, los permisos y la página.
- El TLB, los permisos y el GDTR.
- El TLB, los permisos y la página.
- Ninguna de las anteriores ES LA CORRECTA.

5\_ Un acierto de TLB ahorra: 2 accesos a memoria RAM.

6\_ Que cantidad de bits tiene el LDTR: 16 bits.

7\_ En IA-32 el CR3: Indica la base del directorio de pagina en uso.

Vamos a ver un poco sobre los mecanismos de protección que tiene Intel. Intel tiene siempre en modo protegido algunos mecanismos de protección tanto para segmentación como para paginación, tiene niveles que es un mecanismo que hay en la parte de segmentación, básicamente de segmentación hay tres mecanismos de protección, ¿y para qué y de qué nos protegemos, para que queremos protección, porque nos queremos proteger, de quién?, por más que haga un programa y haga cualquier lío, no perjudique ni a otro programa de usuario ni obviamente al sistema operativo que también es otro programa, de alguna manera para que queden aislado y no puedan hacer lío entre programas, en el peor de los casos mi programa terminará con una excepción porque quiso hacer algo que no podía y chau.

Resumiendo un poquito **hay tres mecanismos de protección a nivel segmentación y un par de mecanismos de protección a nivel paginación si es que está habilitada si no está habilitada me quedo con los otros tres nada más**, el **chequeo de límites** es muy sencillo digamos, si voy a acceder a una posición de memoria, había un descriptor que me daba una base y un límite que de alguna manera me describía un área de memoria, me mapeado un área de memoria, entonces lo que hace básicamente el hardware este es qué, la base más el offset si pasa el límite directamente da una excepción y termina, es bastante sencillo.

Table 3-1. Code- and Data-Segment Types

Decimal	Type Field				Descriptor Type	Description
	11	10 E	9 W	8 A		
0	0	0	0	0	Data	Read-Only
1	0	0	0	1	Data	Read-Only, accessed
2	0	0	1	0	Data	ReadWrite
3	0	0	1	1	Data	ReadWrite, accessed
4	0	1	0	0	Data	Read-Only, expand-down
5	0	1	0	1	Data	Read-Only, expand-down, accessed
6	0	1	1	0	Data	ReadWrite, expand-down
7	0	1	1	1	Data	ReadWrite, expand-down, accessed
		C	R	A		
8	1	0	0	0	Code	Execute-Only
9	1	0	0	1	Code	Execute-Only, accessed
10	1	0	1	0	Code	Execute/Read
11	1	0	1	1	Code	Execute/Read, accessed
12	1	1	0	0	Code	Execute-Only, conforming
13	1	1	0	1	Code	Execute-Only, conforming, accessed
14	1	1	1	0	Code	Execute/Read-Only, conforming
15	1	1	1	1	Code	Execute/Read-Only, conforming, accessed

La otra protección es referida al **tipo de segmentos**, básicamente tengo segmentos de dos tipos, de código y de datos y cuando el CODE SEGMENT apunte a un segmento en la tabla descriptores hay un campo que se llama TIPO, tanto para los segmentos de datos, el segmento de pila, todos esos sí o sí tengo que tener el bit 11 en 0, si no lo tengo en 0 me va a dar un error porque si está en uno por ejemplo va a creer que es un segmento de datos pero estoy intentando acceder a un segmento de código.

Si tuviéramos un segmento de datos con el bit cero correcto, si va a ser un segmento de pila por ejemplo si o si debería ser lectura-escritura o sea debería tener

el bit de RAID en uno, porque si esta en cero es sólo lectura entonces la pila siempre esta PUSH y POP entonces eso también me daría una excepción, o sea si quiero dimensionar un segmento de pila tiene que estar tanto el bit 11 me dice que tipo de segmento 0, y el bit de escritura en 1 porque tiene que ser lectura-escritura.

Si voy a acceder a un segmento de código si o si el bit 11 debería estar en 1, si esta en 0 me da excepción, hay veces que en el segmento del código tengo, datos, valores y variables, declaradas cuando compilo con un valor estático entonces a veces tengo que leer datos de ahí, entonces para esos casos se usa el bit de READ, si ese es el caso debería estar en 1 sino en 0. Si solo lectura debería estar el R en cero.

Finalmente tiene algo que se llama un bit conformante, el tema de conformante tiene con el otro mecanismo de protección que tiene segmentación que es por niveles de privilegio, de alguna manera poniendo un bit 1 (bit 11), podría exceptuarme de hacer algunas validaciones de niveles de privilegios, aparentemente este bit de conforme está puesto por compatibilidad hacia atrás 8086 que tenía un modo protegido medio distinto que se llamaba virtual, en definitiva esta forma de cambiar de nivel de privilegio con ese bit en 1 no es de lo mejor. Primero valido en segmentación que donde quiera ir corresponda, después si el descriptor que tiene cada segmento que quiero acceder es correcto o al menos de mi tipo, si quiero acceder a un segmento de datos debería tener ese bit en 0 y después si voy a leer o escribir en 0 o 1 la escritura o si es de código también.

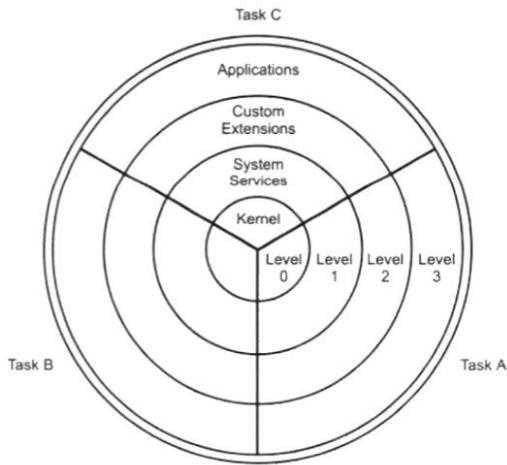
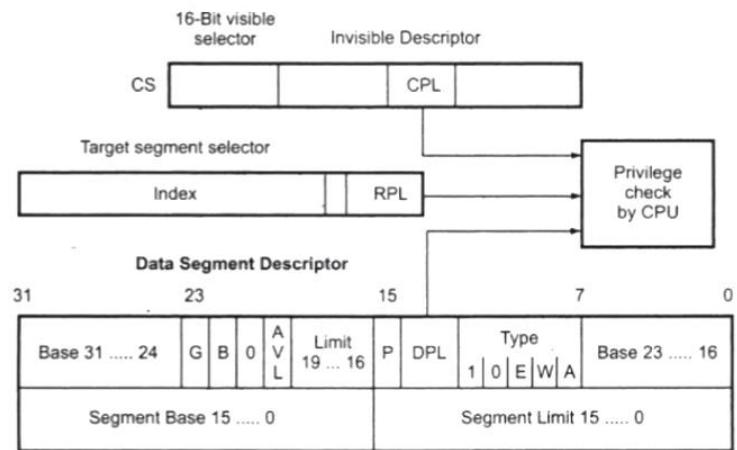


Fig. 4.26 Assignment of privilege levels

nivel 0 donde tengo control de todas las instrucciones y por ende del hardware y un nivel donde tengo muchísimo menos que es en este caso el nivel 3, entonces cuando corro un sistema operativo, el núcleo del sistema operativo trabaja en el nivel 0 y mi aplicación trabaja fuera del nivel 0, mi aplicación básicamente no puede hacer lo que quiere, no puede ejecutar instrucciones que me pueden complicar algunas cosas que pueda romper otro programa.



CPL - Current Privilege Level  
RPL - Requestor's Privilege Level  
DPL - Descriptor privilege level

Fig. 4.27 Privilege check for data access

límite.

Entonces tengo una instrucción determinada en el CODE SEGMENT al que apuntaba, y en la parte oculta tengo la base al desplazamiento pero también tengo el nivel de privilegio, esos dos bits que me dicen el nivel de privilegio en el que estoy ejecutando, que toda esa parte la fue a buscar a la tabla descriptores y lo trae y guarda para que me quede. El el nivel que esté ejecutando actualmente se llama CURRENT PRIVILEGE LEVEL o nivel de privilegio actual y lo tengo en la parte oculta de mi descriptor. Si ahora quiero cambiar y quiero ir a otro segmento de código por ejemplo, lo voy a hacer es poner un valor nuevo en el segmento de código, entonces lo pongo en los 16 bits del selector, de esos 16 bits los últimos 2 bits RPL son el nivel de privilegio que estoy pidiendo para ir a este segmento que tiene este nivel de privilegio, REQUESTOR'S PRIVILEGE LEVEL.

Por otro lado puse los 16 bits en el selector, entonces eso va a ir a la tabla de descriptores con estos 13 bits más altos accedo a la tabla descriptores y voy a obtener un selector determinado, que tiene 8 bytes y tiene información, también tiene dos Bits más que era el nivel de privilegio del descriptor, entonces analizando estos tres valores me dice que voy a hacer, si estoy en un nivel de privilegio menor y quiero acceder a uno que tiene más nivel de privilegio el PRIVILEGE CHECK BY CPU va a dar una excepción y mi programa va a terminar, si está esta lógica para analizarlo me dice que sí, que es factible voy a poder cambiarme y ejecutar ese segmento de código que está en otra parte pero tiene un nivel de privilegio que puedo. **Básicamente entonces toma esos tres elementos para evaluar que son, el valor que tiene actualmente el nivel de privilegio, el valor que estoy pidiendo cuando pongo en el selector el**

Ahora viene el **tercero de los mecanismos** de protección que es quizá el más HEAVY o más complejo, esto tiene cuatro niveles de privilegio, (Intel) son dos bits RP, son los dos bits de privilegios, por lo tanto tengo cuatro niveles y lo interesante es que el nivel 0 es el que tiene más privilegio, acá tengo todo el set de instrucciones del procesador y en el nivel 3 que es el de menos privilegio tengo por ejemplo recordado todo lo que es entra-salida, todo lo que es cargar los registros, por ejemplo el GDTR, la instrucción LOAD GDTR obviamente en nivel 3 no lo puedo hacer porque si no podría modificar la tabla de descriptores globales y explota todo y un montón de restricciones.

En el intermedio tengo dos niveles más que a medida que voy a aumentando el privilegio tienen más funcionalidad, pero la maría de los operativos por más que tenga cuatro niveles lo que hacen es usar un

¿Cómo hago esto? este gráfico lo explica claramente, vamos por partes, supongan que estoy trabajando en un nivel determinado, mi programa se está ejecutando y de repente le pongo por ejemplo el CODE SEGMENT, pongo otro segmento para ir a otro segmento de código, pero a ese otro segmento podré ir siempre y cuando esté en un nivel igual o menor de privilegio donde estoy actualmente.

CS es el registro CODE que tiene 16 bits, internamente tenía una parte que no podemos ver que tenía la base y el límite porque se acuerdan que lo buscaba una sola vez y ya quedaba cargada en la sección no visible de todos los segmentos, de códigos, de datos, de pila, tiene base y tiene un límite.

**valor nuevo del segmento donde quiero ir y finalmente eso apunta a un selector y en el selector está el nivel de privilegio del segmento destino, el descriptor, es en función de eso me va a dejar o no.**

Para complicar más las cosas, los niveles más bajos numéricamente tienen más prioridad o son más privilegiados por lo cual el cero tiene mayor nivel de privilegio que el 1, el 2 y el 3.

Lógica que lleva esa esa cajita PRIVILEGE CHECK BY CPU:

Max (CPL, RPL) ≤ DPL

El mayor entre el nivel de privilegio actual y el requerido debería ser menor o igual que el del destino. Si no se cumple eso chau.

Ejemplos:

Microprocessors and Microcontrollers		4-30	Protected Mode	
No	Privilege Levels			Access
	DPL	CPL	RPL	
1	2	0	1	Valid
2	3	1	2	Valid
3	1	1	0	Valid
4	1	2	0	Invalid
5	2	2	3	Invalid

Table 4.3 Data accesses

Si estoy en el actual 0 (CPL) que es el de mayor nivel de privilegio y pido para ir a un nivel de privilegio 1 (RPL) y el destino tienen nivel de privilegio 2 (DPL) voy a poder porque si veo la ecuación, el actual es 0, el requerido es 1, es menor o igual que 2 qué es nivel de privilegio destino.

Hace un programa Hola.c

```
carlost@maquinola:~/tecnicasIII/2021/cap1$ cd
carlost@maquinola:~$ cd 7tm
bash: cd: 7tm: No such file or directory
carlost@maquinola:~$ cd /tmp/
carlost@maquinola:/tmp$ vim hola.c
```

```
#include <unistd.h>
#include <stdio.h>

int main (){
    printf ("holamundo\n");
    return 0;
}
```

```
carlost@maquinola:/tmp$ vim hola.c
carlost@maquinola:/tmp$ make hola
cc    hola.c -o hola
carlost@maquinola:/tmp$ ./hola
holamundo
carlost@maquinola:/tmp$
```

Ahora la pregunta, estoy en modo usuario, modo 3, ¿cómo hice para usar una instrucción de entrada salida? porque eso está apareciendo en la pantalla. En realidad vamos a ver que sí intentara ejecutar quizás una función del operativo que está en un nivel de privilegio 0, mi programa va a explotar o va a terminar, porque me va a dar una excepción por los cambios a niveles privilegios.

La buena noticia es que si quiero ir directamente de un nivel inferior a uno superior en privilegio, no número, me va a dar una excepción, pero la buena noticia es que hay algunos mecanismos que de alguna manera me permiten indirectamente saltar de nivel 3 a 2, 3 a 1, 3 a 0, para hacer algo muy puntual y después sigo con la ejecución en el nivel donde estaba. Este mecanismo se llama CALL GATES o puertas de llamada.

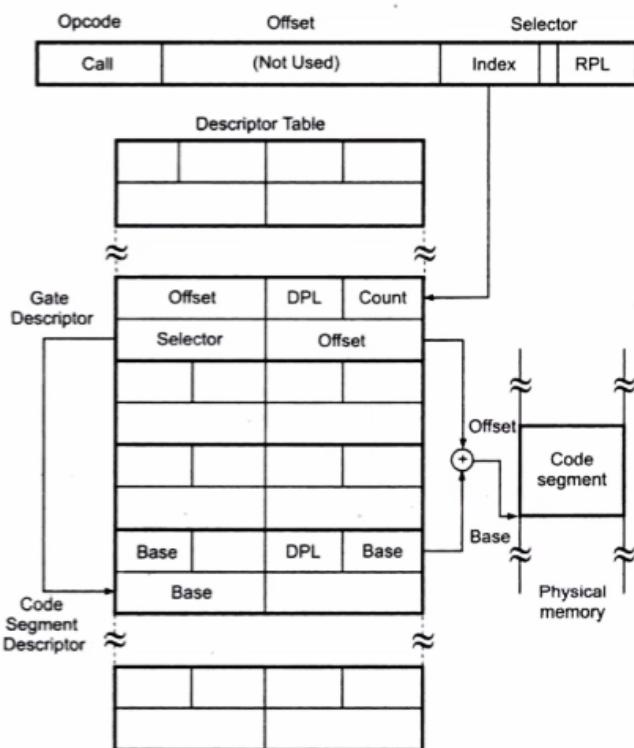


Fig. 4.30 Indirect transfer via call gate

No hace como antes donde usaba el offset y se lo sumaba a este selector, sino que lo que hace es saca el offset del descriptor de puertas, entonces puedo ir a otro segmento el código de otro nivel que es privilegiado por ejemplo, pero no donde yo quiera sino donde me diga el descriptor de puerta, entonces seguramente voy a tener un descriptor de puertas para distintos tipos de función o llamadas a sistema.

La comprobación de nivel, comprueba el requerido con el de la puerta, no con el del segmento donde quiero ir, entonces si la puerta es para rutina de nivel de privilegio 3 por ejemplo del menor, en cualquier programa que está ejecutando podría invocar esa rutina porque no válida el segmento de verdad, al hacer esto, es como si fuera justamente un indexado. Una vez que termine me vuelve a mi nivel de privilegio original.

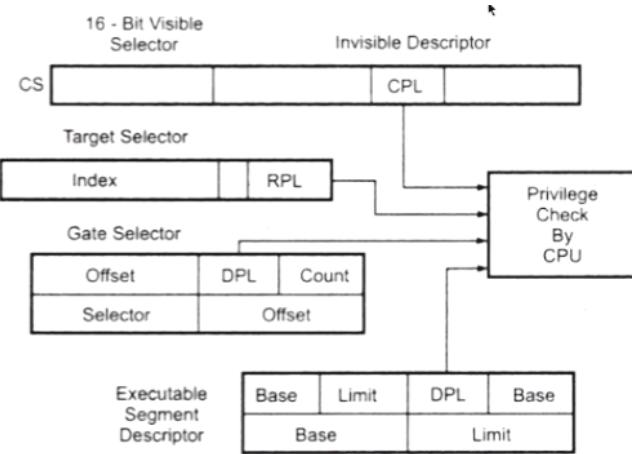


Fig. 4.31 (a) Privilege check via call gate

Acá se puede ver lo que controla con una puerta, El DPL es el del selector de puertas invertido y no el del otro segmento.

Lo que hago es en vez de apuntar en mi selector a un segmento al que no puedo ir, apunto a un descriptor de un CALL GATES, los descriptores CALL GATES están en las tablas de descriptores y tengo descriptores de distintos segmentos y de repente tengo uno que se llama descriptor de puerta GATE DESCRIPTOR, que también ocupa 8 bytes como el resto pero en realidad no me describen un área en memoria, porque cualquiera de los otros como por ejemplo un descriptor de un segmento de código lo que me va a describir es en la memoria lineal un área donde empieza y dónde termina, pero este descriptor no me apunta a un área de memorias sino que tiene otra información, básicamente es como si hiciera un direccionamiento indirecto a la memoria que quiero ir.

Entonces pongo el nuevo selector donde quiero ir, con un nivel de privilegio que quiero, encuentra un descriptor de puerta, y ese descriptor de puerta tiene el selector del segmento a donde realmente quiero ver ir, y este selector me describe un área memoria, un segmento, con una base y tiene un límite,

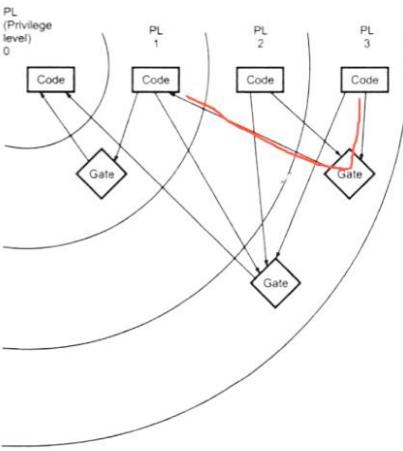


Fig. 4.31 (b) Some valid accesses to higher privilege levels using call gates

En este ejemplo hay tres descriptores de puerta, uno en nivel 1 y dos en el nivel 3, entonces si estoy ejecutando código en nivel 3, podría invocar a través de la puerta código que está en el nivel 1, o podría invocar el nivel más privilegiado de todos que es el nivel 0.

Lo que no podría hacer es desde mi código que esta en nivel 3 ir a la puerta que está en el nivel 1, porque el DPL de esa puerta es 1 y yo estoy en 3.

Pero si podría si estoy en código nivel 1 o 2, ejecutar una puerta que está en nivel 3 y de ahí ir a uno de mayor privilegio.

## PROTECCION A PAGINACION

Tiene dos mecanismos de protección:

- 1- Restricción del dominio direccionable
- 2- Chequeo de tipo

Recordando:

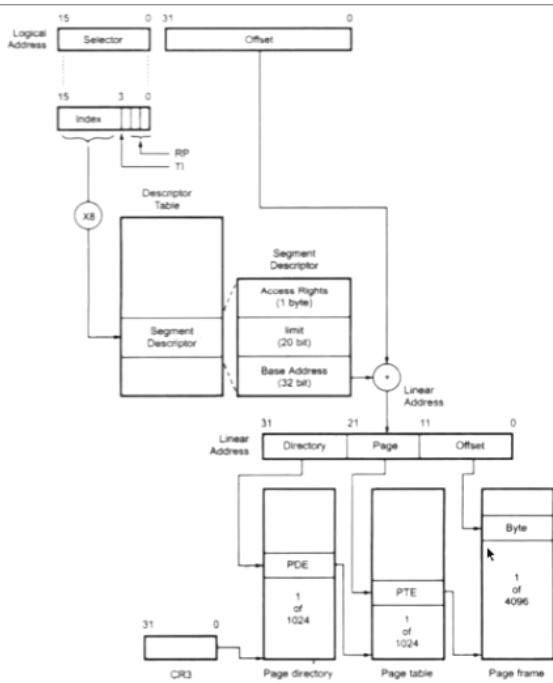


Fig. 4.24 Protected mode address translation

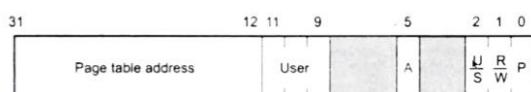


Fig. 4.21 Page directory entry

Tengo la memoria lineal, tengo que llevarla a la memoria física que esta paginada, entonces tenía una entrada al directorio de página y una entrada a la tabla de página, ahí dentro además de tener la base de la tabla de pagina y la base del marco, tiene otras cosas que son justamente estas que vamos a ver.

Contiene también si esa página es de usuario o sistema, otro bit me dice si es lectura, o lectura escritura, tanto en la entrada del directorio de página, como la entrada en la tabla de página.

Si estoy ejecutando una instrucción o un dato de aplicación nivel 3, va a estar en usuario y no podría acceder a segmentos que tengan un nivel de privilegio mayor, me restringe.

En el bit de escritura, si estoy accediendo a un segmento de código de solo lectura me va a dar una protección por tipo y me va a dar un error.

Entonces lo que hace paginación es chequear si es de sistema el descriptor o es de usuario, estos deberían corresponderse con los descriptores que tengo, y lectura o lectura-escritura se corresponde con el tipo de segmento si es de código o es de dato.

## MULTITAREA

Vamos a ver el soporte de hardware que tenemos porque después cuando vemos sistema operativo vamos a verlo desde otro punto de vista.

Básicamente tanto el manual de Intel como Götze plantean que tengo soporte para cambiar los registros internos del procesador rápidamente. ¿Y eso para qué? intercambiar entre tareas.

El concepto de multitarea: tengo un procesador y tengo varios programas, si quiero hacer que trabajen a la vez deberían estar en la memoria principal, en realidad como tengo una sola ALU, una sola CPU no pueden trabajar a la vez, van a tratar de trabajar pseudo en paralelo o casi a la vez. O sea en un momento determinado se va a estar ejecutando un programa u otro pero no pueden los dos a la vez porque se mezclaría.

Lo vamos a ir ejecutando periódicamente, normalmente eso se llama tiempo compartido donde le doy una tarea por un tiempo determinado, en un momento saco de ejecución esa tarea como si le sacara una foto con el estado interno con todos los registros como esta, lo guardo y lo que hago es poner otra tarea ejecutarse y se va a ejecutar por un tiempo y en algún momento lo que hago es otra vez le sacó una foto a esa tarea y guardo su estado y vuelvo a poner la anterior, o una nueva, y así las voy ejecutando periódicamente, voy cambiando, eso se llama cambio de contexto y lo que hago es hacerle la ilusión al usuario que estoy ejecutando varias tareas a la vez.

Por más que mi procesador pueda ejecutar dos instrucciones a la vez se me complica si las instrucciones son de dos programas distintos, en un momento determinado si tengo entrada salida se justifica dejar de ejecutar y comenzar a ejecutar otro, porque la entrada-salida demora mucho comparado con una un acceso a RAM.

Imaginemos que el procesador va a buscar la siguiente instrucción que está en la memoria principal y el acceso está en el orden de los nanosegundos, diferente es si tuviera que buscar un dato que está en el disco rígido por más rápido que sea ese acceso esta en el orden de los milisegundo. Si hacemos un cambio de escala, y decimos que un acceso a RAM demora un día para acceder, ¿cuantos días demoraría una entrada salida rápida? Estamos hablando de tres años mínimo.

Entonces cuando una tarea se está ejecutando y necesita una entrada-salida va a perder un montón de tiempo esperando porque no puedo hacer nada, entonces justo en ese momento me conviene dejar de ejecutar esa tarea y poner ejecutar otra, para aprovechar el tiempo que esa tarea no puede seguir haciendo nada.

Entonces vieron cómo ejecutar más de una tarea a la vez, el problema que tenían era que entre las tareas interactúan y se puedan romper, Intel le agregó una protección, y otro problema es que hay que hacer todo un tema para cambiar de tarea y ejecutar otras, bien lo que hizo Intel fue agregarle soporte hardware para multitareas.

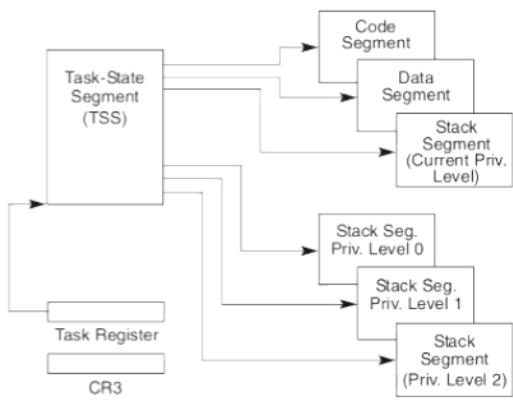


Figure 6-1. Structure of a Task

Cada programa que se está ejecutando Intel le llama tarea, cada rutina de atención-interrupción Intel lo llama tarea, cada rutina del KERNEL también es una tarea, para el de hardware todas son tareas, cualquiera de ellas, no importa si es de usuario, si es una rutina de atención.

¿Cada tarea que cosa tiene? primero algo que se llama TASK-STATE SEGMENT, es un área de memoria, cada tarea tiene una porción pequeña de memoria que tiene un segmento con el estado de la tarea.

¿Qué cosas tiene dentro el TASK-STATE SEGMENT? Esta en la memoria principal, tiene los valores donde guarda las fotos que saqué, qué valor tenía ese momento del contador del programa, qué valor tenía el registro AX, qué valor tenía el CODE SEGMENT, qué valor tiene el DATA SEGMENT, todo eso lo guarda ahí.

Cada tarea tiene un TASK-STATE SEGMENT que es un segmento de memoria que tiene 104 bytes donde se guardan todos los registros internos y algunas cosas más.

¿Cómo encuentro ese segmento de estado de la tarea en la memoria RAM o en la memoria lineal? tengo un registro en el procesador que se llama TASK REGISTER de 16 bits, ese tiene el valor de un descriptor de TASK-STATE SEGMENT, que está en la tabla de descriptores globales que apunta a ese estado.

Si quiero cambiar de tarea lo que tengo que hacer es cargar en el TASK REGISTER el nuevo descriptor de tarea que quiero acceder.

31	Bit Map Offset	0000000000000000	T	0
64		LDT		
60		GS		
5C		FS		
58		DS		
54		SS		
50		CS		
4C		ES		
48		EDI		
44		ESI		
40		EBP		
3C		ESP		
38		EBX		
34		EDX		
30		ECX		
2C		EAX		
28		EFLAGS		
24		EIP		
20		CR3		
18		SS2		
14		EIP2	Editar título	
10		SS1		
0C		EIP1		
8		SS0		
4		EIP0		
0		Back link		

Fig. 5.1 Task state segment

Son los registros EIP, guarda ahí todos los registros internos, guarda la tabla de descriptores locales si tuviera.



Este es un descriptor de estado de segmentos de estados de tarea, tiene como siempre una dirección base y el límite, cuando va a acceder a una nueva tarea valida que el límite sean 104 bytes, todas tienen mismo tamaño. Este descriptor está puesto en la tabla de descriptores globales y es uno por cada tarea.

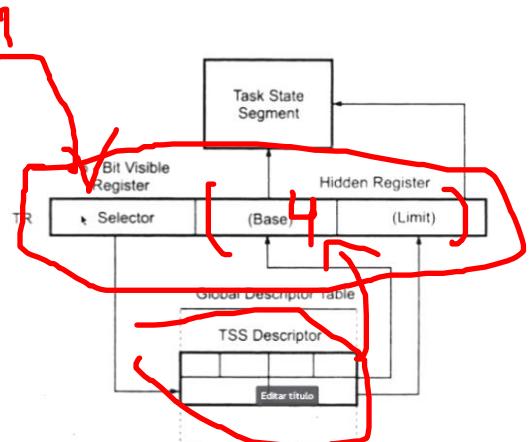


Fig. 5.3 Task register

Es un registro del procesador, TR, tiene 16 bits, básicamente lo que hace es apuntar a la tabla descriptores globales a un descriptor, este descriptor es el que vimos anteriormente TTS, TASK-STATE SEGMENT, y me describe una dirección base y un límite que esta puesta en la memoria principal, tiene el todos los registros que vimos recién almacenados, el registro TR tiene una parte oculta donde una vez que tengo una tarea, la base y el límite lo deja ahí para no tener que ir a buscarlo nuevamente cada vez que quiero acceder al TASK-STATE SEGMENT.

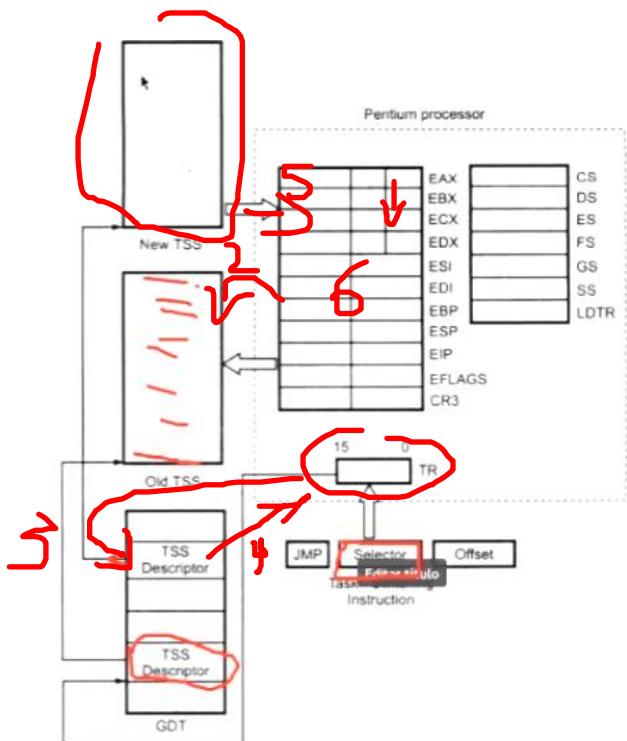


Fig. 5.5 Task switch operation

Les muestro como por ejemplo haría un cambio de tarea, supongan que tengo el registro de tareas TASK REGISTER (TR) en un momento determinado está apuntando a un descriptor de tarea, está ejecutando una tarea, en el momento que quiero cambiar de tarea una de las forma es haciendo un JMP y poniendo un selector de otra tarea, ahora voy a apuntar a un nuevo descriptor (que está más arriba, el TSS DESCRIPTOR),

Antes de cambiar de tarea, arriba a la derecha veo los registros internos del procesador, se van a guardar en la tabla de estado de la tarea vieja, ese es el primer paso que hago antes de cambiar, después apunto a la nueva tarea, que ahora será la que está mas arriba, y ésta está descrita en el otro TSS DESCRIPTOR, los valores de ahí pasan a los registros internos del procesador. Todo esto lo hace Intel directamente, no tengo que programar nada.

Hay cuatro maneras de hacer este cambio contexto, switch o un intercambio de tareas, dos son usando JMP o CALL (un usuario común no lo puede hacer), hay otra que es indirecta, cuando me llega una interrupción lo que hago es cambio de tarea y otra cuando termina una interrupción o rutina de atención de interrupción, en estas dos no son directas, no es que les ponga mano donde quiero ir sino para complicar más las cosas tengo algo que se llama una puerta de tarea, el mismo concepto de CALL GATE pero con una puerta de tarea.

Otro caso sería cuando la tarea por ejemplo hace una entrada salida, que se la pide el operativo y como va a demorar mucho, entonces al operativo le forzamos a que cambie.

The Pentium processor does task switching in any of four cases :

1. A long jump or call instruction contains a selector which refers to a TSS descriptor. This is the simplest method and can be easily implemented by the operating system kernel at the end of a time slice.
2. The selector in a long jump or call instruction refers to a task gate. In this case the selector for the destination TSS is in the task gate. This indirect method has advantages regarding privilege levels and protection.
3. The interrupt selector refers to a task gate in the interrupt descriptor table. The task gate contains the selector for the new TSS. If the access passes all the privilege level tests, the selector and descriptor for the interrupt task will be loaded into the task register. The nested task (NT) bit in the EFLAGS register will be set.
4. An IRET instruction is executed with the NT bit in the EFLAGS register set. The IRET instruction uses the back link selector in the TSS to return execution to the interrupted task.

En los dos primeros es donde el operativo decide por ejemplo una tarea va a hacer entrada salida entonces usando un CALL o un JMP decide cambiar.

En el 3ro cuando hay una interrupción, hay que ejecutar una nueva tarea que es la rutina que atiende esa interrupción, normalmente solo hago a través de un TASK GATE.

En el 4to cuando ejecuto una instrucción IRET, que es retorno de una rutina, lo que tiene que hacer es volver

a ejecutar la rutina anterior que estaba ejecutando antes que llegara la interrupción, y la pregunta es ¿cómo se cuál era la que estaba ejecutando antes? cada tarea tiene 16 bits donde dice BACK LINK, solo lo uso cuando tengo un IRET (solo para este caso), cuando retorno de una rutina de atención de interrupción porque debería seguir ejecutando lo que estaba ajustando antes de que llegara la interrupción.

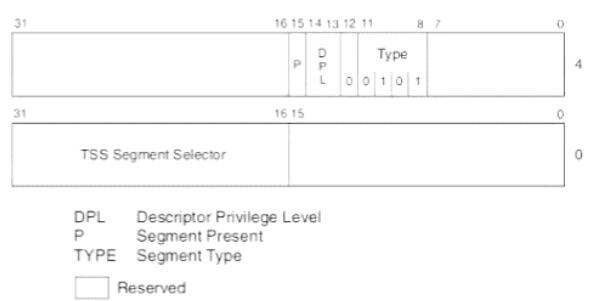


Figure 6-5. Task-Gate Descriptor

Este es un TASK-GATE, estos son los ocho bytes que estarían en la tabla de descriptores globales y básicamente lo que tiene es 16 bits donde apuntan a un selector de TSS, entonces el direccionamiento indirecto.

Tengo la puerta, esa apunta al descriptor de la tarea donde quiero ir y eso apunta al nuevo TASK STATE SEGMENT, esto lo puedo hacer desde una tarea o desde las interrupciones, entonces en el vector de atención de interrupción, en cada una de las interrupciones pongo un TASK-GATE con este mecanismo. Ese TASK GATE apunta a un descriptor de la tabla de descriptores globales que tiene un descriptor TSS, y ese apunta al nuevo TSS que se va a intercambiar por el que estoy usando ahora.

**Lo que toma en el examen:**

**Básicamente lo que he explicado que lo entiendan, la estructura básica que habla un poco los bloques, modo protegido ¿qué es?, y el direccionamiento en modo protegido que es segmentación y paginación, las protecciones, obviamente la más importante por niveles que vimos, y el soporte de conmutación tarea.**

## SISTEMAS OPERATIVOS

### Introducción

- Temario
  - Historia
  - Componentes de un OS
  - Tipos de OS
  - Estructura de un OS

# Historia

## 1945-1955 Primera Generación (Valvulares)

- Las primeras computadoras no incluían Sistemas Operativos
- Las aplicaciones de usuario se hacían completamente en ISA.
- Gralmente. aplicaciones de cálculo numérico.
- Se ingresaba el programa a ejecutar , y se esperaba una respuesta.
- Problemas ??
  - Tiempos ociosos muy grandes

para cálculo numérico, para cálculo de balística o cálculo para intentar descifrar justamente los mensajes que mandaban los alemanes con el código enigma.

Estas computadoras tenían muchos problemas, pero básicamente ponían un código para ejecutarse y esperar a que se termine y diera una respuesta.

Entre sus problemas estaba el consumo, la disipación de calor, tamaño, la confiabilidad más allá de la demora, tenían mil válvulas y el tiempo de vida medio de las válvulas es poco, entonces la mitad de las veces no terminaba ejecutar la tarea porque se rompió una válvula y había que empezar de cero. El modo de cargar los programas, las primeras eran teclas o llaves, era todo mecánico.

Tenían tiempo ocioso muy grande que era desde que me ponía a cargar el programa hasta que podía cargar, básicamente ese fue uno de los mayores problemas y la poca confiabilidad.

# Historia

## 1955-1965 Segunda Generación (Transist, procesos batch)

- Ejecutaba un job a la vez.
- Sistemas de procesamiento batch de flujo único.
- Los Programas y datos se almacenaban consecutivamente en la cinta o tarjetas perforadas.
- Control de las entradas/salidas y la memoria a cargo del programador.
- Programa entero en memoria.
- Problemas ??
  - Pérdida de tiempo entre un job y otro

En la segunda generación ya tengo un poco más de confiabilidad, porque en los cincuenta comenzaron a desarrollarse los semiconductores entonces ya no era tanto el tamaño, disminuyendo el consumo, se fue aumentando la confiabilidad, ejecutaba una tarea por vez, ponía un programa y lo ejecutaba y una vez que terminaba ponía otro programa y así.

Lo que hicieron para acelerar un poco los tiempos muertos, para acelerar el ingreso de los programas, usaron los mecanismos de tarjetas perforadas, y de esa manera era mucho más rápido que andar con una clavija o con unos botoncitos.

Mientras se estaba ejecutando un programa, podía tener una computadora secundaria que es la que leía las tarjetas perforadas y lo pasara a una unidad cinta por ejemplo que es más rápida, y la salida de esos programas eran en impresoras de tipo matriciales, entonces otra alternativa era que la salida en vez de ser una impresión que era lenta, se copiara el resultado en una cinta, y esa cinta se pasaba a la otra máquina secundaria que leía la cinta e imprimía. Entonces esto me permitía aumentar el tiempo de uso de la CPU.

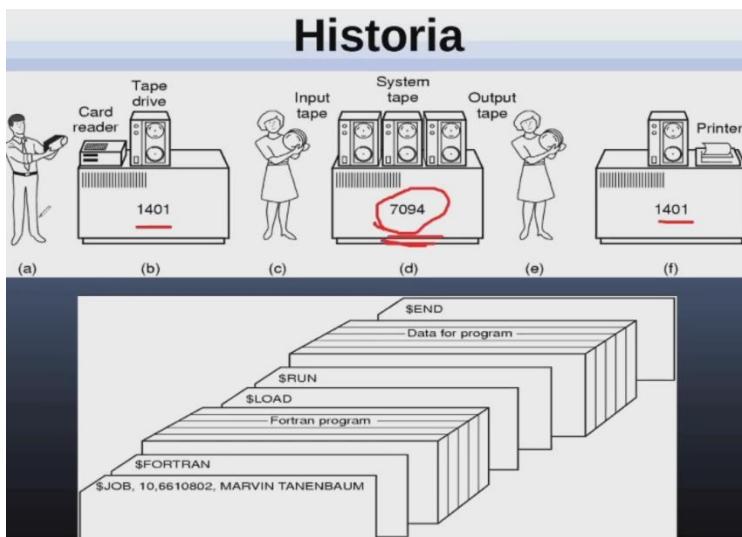
Ahí se empezó a separar las funciones, el que desarrollaba el hardware, otro hacía los programas, otro operaba ponía la cinta, movía la cinta de un lugar a otro etcétera, a estos se le llamaba operadores.

Tenía el problema de que podría ejecutar un programa a la vez. Perdía tiempo entre una tarea y otra entonces tuvieron la brillante idea de en una cinta no poner una sola tarea sino poner una serie de tareas que se ejecutaran

Si bien la computadora arrancaron en los 1800 y pico más o menos, en realidad el primero gran avance fue entre el 45 y el 55, acá estoy hablando de computadoras, no de sistemas operativos, porque obviamente las primeras computadoras no tenía sistema operativo.

Comenzaron con semiconductores, algunos con contactores, se usaban básicamente

una tras otra y ahí escribieron un mini programa que se llamaba sistema de operador que terminó siendo el sistema operativo, iba leyendo una a una las tareas de la cinta y las iba ejecutando, entonces no se perdía tiempo entre ir intercambiando cintas, sino que en una cinta ponían todas las tareas que tenía que hacer. A esto se le llama procesamiento BATCH.



Se ve una computadora cara (7094) que hace todo el procesamiento y el objetivo es tratar de que esta computadora esté ociosa el menor tiempo posible, entonces para eso tenía computadoras mucho menos potentes y mucho más barata, donde el señor operador o la señorita operadora cargaban las tarjetas perforadas que eran los programas, y una vez que tenía todo guardado en la cinta recién ahí metía esta cinta, no llevaba una tarea sino llevaba todo este conjunto de tareas.

## Historia

1965-1980 Tercera Generación (Circuitos Integrados)

- IBM presentó la familia de computadoras System/360
  - Cálculos & comercial → software compatible
  - Multiprogramación
  - Problemas ?
    - Sistema operativo complejo e ineficiente
  - Aparición de minicomputadoras (p.ej. DEC PDP-7)

Hasta ahora el sistema operativo era básicamente una rutina que lo que hacía era ir leyendo de una cinta tarea por tarea y las lanzaba a ejecutar para tratar de evitar tener la CPU ociosa.

La 3ra generación de computadores ya tienen circuitos integrados, entonces eran bastante más potentes, bastante menos consumidoras, bastante más confiables. Hasta esa época habían como dos líneas de desarrollo, una para computadoras que se dedicaban a hacer cálculos de tipo científico desde ingeniería, y otras que se dedicaban al tema comercial, entonces como que unas estaban más preparadas a procesamiento de datos y otras más preparadas a entrada-salida.

IBM era uno de los grandes jugadores de esa época y tuvo una brillante idea, crear una sola familia de hardware que soporte de los dos tipos target de la computación, porque hasta ese momento todos los fabricantes tenían una división que se encarga para una cosa y otra división para otra, con hardware totalmente distinto e incompatibles. Puso un solo hardware y entonces como los sistemas operativos básicos que tenían para cálculo y los comerciales también eran distintos, lo que tratan de hacer es ponerlo todo en uno solo. Otra cosa que también hizo fue usar el mismo software para la máquina con menos potencia hasta la máquina con mayor potencial, de alguna manera estandarizar el uso de un mismo sistema operativo para todo.

En esa época ya tenían multiprogramación, vieron la posibilidad de cargar más de una tarea de memoria sin que se superpongan y hagan lío entre ellos, pero con soporte de hardware.

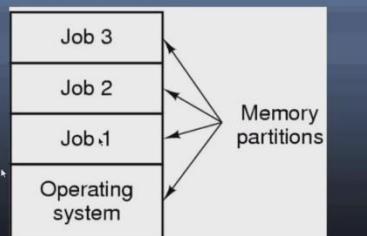
Imaginen los programadores, hace un programa, el problema era el tiempo entre que el programador quería correr su aplicación y que efectivamente tener un resultado.

si bien hicieron un software que era compatible con hardware para cálculo y comercial, para poca potencia y mucha potencia, ese sistema operativo era bastante complejo e ineficiente, porque tenía muchas funcionalidades para las computadoras de gama alta de cualquier estos dos sabores de estos dos tipos de cálculo y comercial, pero si quería correr el mismo software en una computadora en la gama más baja era bastante complejo, porque era muy grande el software, entonces tuvieron esos problemas pero siguieron trabajando en ese sentido, en consolidar un solo software.

Aparecieron las mini computadoras, en ese momento nadie les dio mucha atención porque la potencia cálculo la verdad es que era muy chica, y el costo.

## Historia de Sistemas Operativos

### Multiprogramming



Con respecto a multiprogramación, la idea era que ponga en la misma memoria RAM varias tareas cuando una se está ejecutando y necesita entrada-salida puede automáticamente con ayuda de este software ejecutar otra de las tareas, con ayuda del software y obviamente algunas ayudas de hardware como habíamos visto en el capítulo anterior.

## Historia de Sistemas Operativos

### Multiprogramming

protección entre particiones .... mejor por hardware

Spool (Simultaneous Peripheral operation on line) no mas cintas ....

Problema ??

- Tiempo perdido del programador

multiprogramming + terminales = time sharing (primer OS CTSS)

Generar respuesta dentro de un período de tiempo limitado.

MIT, GE y Bell usaron el sistema CTSS para desarrollar su propio sucesor, Multics. (MULTI Information Computing Service)

Luego del fracaso comercial de Multics, en los Labs Bell Thomson y Richie escriben UNIX PDP-7

Protocolo estándar de comunicaciones TCP/IP ambientes militares

Tengo que tener una protección entre las particiones, si tienen soporte de hardware lo hace mejor. Lo que le agregó al hardware esto era Operación simultánea en los periféricos, entonces ya no necesitaba tanta cinta para entrada ni para salida sino directamente la computadora permitía leer de tarjeta perforada o escribir en una impresora sin que el programa tenga que esperar que termine, a esto se le llama SPOOL.

Si bien la multiprogramación mejoró, el problema que seguía teniendo es que los programadores tenían un DELAY bastante grande entre que querían compilar por ejemplo su programa y tener el resultado.

Lo que se planteó para solucionar esto, fue decir qué pasa si hago que la respuesta al programado sea rápida, eso se llama TIME SHARING o tiempo compartido, que básicamente es multiprogramación, puedo tener más de un programa, pero tengo terminales o dispositivos periféricos donde una persona puede estar interactuando o trabajando, entonces como no todos los programadores van a estar a la vez escribiendo cosas, código o compilando, el sistema operativo lo que va haciendo es mientras uno de los programadores está pensando o se fue hacer un café o lo que fuera, va asignando recursos a otros programas pero la clave es que ahora a diferencia de antes que tenía que esperar que me diera un resultado en una impresora o lo que sea, usaron dispositivos, básicamente un teclado y una pantalla que están conectados generalmente por un puerto serie a una computadora central, entonces le asignaba un espacio de tiempo y era como si trabajara solo en la computadora sin nada más que lo fuera conmutando en el tiempo, básicamente era multiprogramación pero con períodos cortos de tiempo entonces tenía como una respuesta instantánea para el programador.

A partir de eso el MIT había generado un sistema este que se llamó CTSS que fue el primero, después se juntó con un par de empresas para hacer uno mejor, se juntó con laboratorio BELL y con la gente de GE que en ese momento se dedicaba a las computadoras. Entre los tres decidieron crear un sistema operativo que le llaman MULTICS y la idea era vender servicios, no vender computadoras, ellos iban a tener una computadora muy grande y el que quisiera servicio de procesamiento le ponía una terminal en su oficina y le cobraban por el tiempo de uso. (Como si fuera una nube). Tuvieron un montón de ideas interesantes que después fueron aprovechadas por otros sistemas operativos.

Algunos de los que participaron, un par de personas un tal Thomson y Ritchie, trabajaban para BELL, que era uno de los que había participado, tenían mini computadora PDP-7 con mucha menor potencia que la supercomputadora que habían armado, el hardware no alcanzaba para MULTICS, entonces lo que hicieron fue escribir un sistema operativo con alguna de las características que tenía MULTICS solo que para uso personal y le pusieron y UNIX en ese momento.

Crearon una abstracción, un lenguaje para programar el operativo en ese lenguaje y no tiene que hacerlo en lenguaje máquina, entonces crearon un lenguaje que se llama "C". Para poder portar UNIX a otra plataforma.

En los 70 y picos también en el MIT hicieron algunas aplicaciones de comunicaciones que se llamaban TCP/IP en ese momento, era para militares.

## Historia

1980-Actualidad Cuarta Generación (PC computers , LSI)

- 8080 nace en 1975
- IBM PC 1980
- CP/M
- Microsoft
- Apple Lisa → Apple Macintosh
- Windows
- Network Operating Systems
- Modelo de Computación Cliente/servidor
- Aparece el software fuente abierto y libre
- Se funda Open Source Initiative (OSI) para beneficios futuros de programación open-source

de la 34 hasta la carita feliz no va nada

El 8080 INTEL lo hace en el año 75, algunos fabricantes ya hacían computadoras, computadoras con hardware pequeño. IBM lanza la IBM PC, no tenía operativo, no tenía nada para eso, lo que hace es usar uno que se llama CP/M de un agente DIGITAL RESEARCH, lo había hecho para el procesador 8080, para distinto hardware, cuando IBM saca su computadora buscan un operativo y la gente de DIGITAL RESEARCH no le dio ni bolilla, IBM buscó y encontró un tal Gates que tenía un compilador en BASIC entonces él se consiguió una copia de DIGITAL RESEARCH y lo pago barato.

Apple saca interfaz gráfica, Windows le copia. Si queremos podemos ahondar en este tema que hay por todos lados.

## Sistema Operativo

¿Entonces, qué es un Sistema Operativo ?

Existen dos enfoques

- Máquina extendida
  - Capa de abstracción
  - Uso de Syscalls por ejemplo
- Manejador de Recursos
  - Procesos (que es un proceso?)
  - Memoria
  - I/O
  - Sistemas de Archivos
  - Multiplexación (Compartir recursos en tiempo y espacio)

Lo que el sistema operativo trata de darme es un interfaz genérica y única, para lo que quiera hacer, de más alto nivel y más homogénea para los dispositivos de bajo nivel, eso por un lado.

Este es uno de los enfoques y se llama máquina extendida, me da una abstracción de todo el hardware feo. ¿Cómo accedo al operativo? usando llamada a sistema, hago una llamada a sistema para leer o escribir, no tengo que saber qué tipo de dispositivo es.

El otro enfoque es lo que se llama un manejador de recursos donde la computadora tiene un montón de recursos, llámese procesos que son programas que se están ejecutando

básicamente, tiene recursos como memoria, como dispositivo de entrada-salida, como un sistema de archivos, y lo que tengo que hacer es compartirlos, hacer una multiplicación entre los distintos usuarios, puedo hacer multiplexación de tiempo, un ejemplo eso sería tener varios programas para ejecutar entonces lo que hago es asignar la única CPU que tengo en el tiempo.

En espacio tengo toda una memoria pero resulta que tengo que poner varios programas entonces tengo que ir asignándole partes, entonces esa parte se la doy todo el tiempo hasta que termine mi programa, pero me tengo que encargar de qué los otros no lo ocupen.

# Sistema Operativo

Pero ....¿Que es un Sistema Operativo ?

No hay definición completamente adecuada:

- El objetivo de las computadoras es resolver problemas del usuario (son muy variados)
- Todos los programas usan operaciones comunes (ej:E/S)
- El software que hace operaciones de control de Hw.
- Aquel programa que brinda un entorno para que se ejecuten otros programas, generalmente se ejecuta permanentemente (kernel)

En definitiva no va a encontrar una definición completa, hay varias que puse.

Es un programa que le da a otros programas un entorno para que les sea más fácil trabajar.

Su función es hacerle la vida más fácil al programador con abstracciones o con una máquina extendida se le dice o se le llama, sería un SET de instrucciones de más alto nivel por eso le llama máquina extendida, y la otra es multiplexar los recursos que tengo en todas las tareas que quiero hacer.

## Gestiones del Sistema Operativo

Gestión de Procesos:

Crear/terminar/suspender/reanudar procesos

Gestión de Memoria:

Controlar que partes de memoria corresponden a cada proceso

Asignar/liberar espacio de memoria a los procesos.

Gestión de Almacenamiento:

Archivo y directorio (almacenamiento lógico)

Creación/borrado archivos y directorios

Asignación de archivos/directorios a medios secundarios

Protección y Seguridad:

Evitar sobreescrituras y monopolización de CPU

Esquema de usuarios y permisos de acceso

¿Básicamente qué es lo que hace el sistema operativo, que controlo, de qué se encarga de manejar? lo que se llama gestión de procesos, tomemos procesos como programas en ejecución, entonces debería poder crear, terminar, suspender, reanudar procesos, un sistema operativo debería encargarse de eso. suspender y reanudar es porque si tengo una sola CPU podría intercambiarlos como hemos visto en multitarea, debería poder gestionar la memoria, saber de toda la memoria cómo está distribuida entre distintos procesos, programas, asignarle memoria para procesos nuevos, liberar si un proceso terminó para poder utilizarla de nuevo. También tiene que encargarse del almacenamiento, los dispositivos almacenamiento son

por bloques, los discos por ejemplo, entonces debería saber en qué bloque leer o escribir cosas y cuántos bloques, el tema qué va a cambiar la geometría dependiendo del tipo de dispositivo entonces si lo hago por bloques la verdad es un lío, entonces lo que hace el operativo es hacer una abstracción que se llama sistema archivo y archivos, entonces puedo crear archivos, leer archivo, borrarlo, crear directorios, borrar directorio.

Además algunos temas como protecciones de seguridad, evitar que una tarea no monopolice la CPU, me refiero a que el resto no pueda ejecutarse hasta que no termine esa, evitar que una tarea interactúe con otras, que escriba cosas que rompa otra tarea, de estas cosas también se debería encargar el sistema operativo.

También algunos esquemas de usuario y permisos de acceso.

## Tipos de Sistemas Operativos

Sistemas de propósito general: Sistemas para pc.

Sistemas de tiempo real: Tareas muy específicas, requisitos rígidos de respuesta en tiempo.

Sistemas Multimedia: Posibilidad de trabajar video/sonido. Algunas restricciones de tiempo.

Sistemas de mano (PDA): Procesamiento lento para ahorro de energía. Pocos recursos HW.

Sistemas Centralizados: Todo procesamiento en servidor, terminales "bobas".

Sistemas Cliente Servidor: La presentación se hace en el cliente.

Sistemas entre Iguales (Peer to Peer): Cliente/servidor simultáneamente. Previa Registración.

Sistemas Basados en WEB: Centralizado ? Uso de alta disponibilidad y equilibrio de carga.

Nosotros vamos a trabajar en sistemas de propósito general y después vamos a incursionar en sistemas de tiempo real.

La de propósito general es donde corremos la pc y sirve para cualquier cosa.

Tiempo Real lo vamos a ver luego, pero tengo restricciones temporales.

Sistemas multimedia son de tiempo real pero más laxo, tiene restricciones de tiempo pero si no la cumple no pasa nada.

PDA no existen más, las PALM, debería ser

reemplazado por sistemas de teléfono inteligente, que en definitiva son sistemas de propósito general, pero están pensados un poco para ahorrar energía.

Sistemas centralizados son donde todo el procesamiento se hace en una sola computadora, el resto son clientes que no hacen ningún procesamiento.

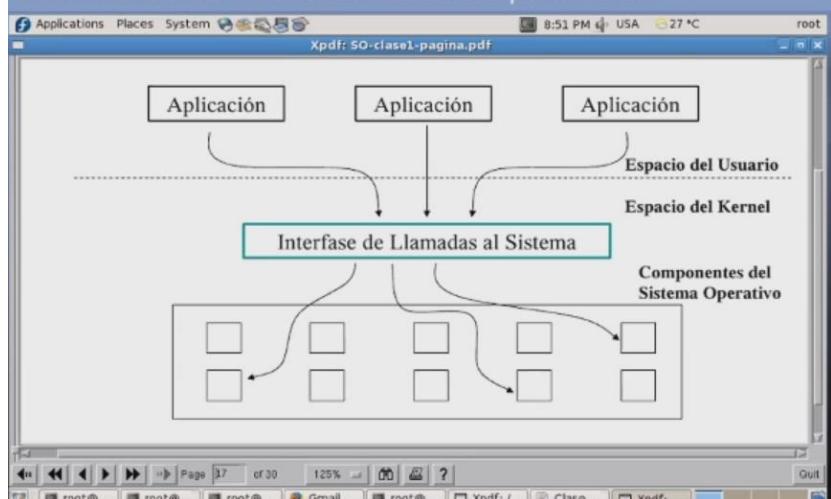
Sistema Cliente Servidores donde parte de la tarea la hace una máquina y la otra la hace otra máquina, normalmente son de red, la parte del servidor es la que tiene información y el cliente el que solicita información y hace la presentación al usuario.

PEER TO PEER pero podemos cumplir los dos roles en distintos momentos, o sea podemos tener un sistema archivos y ofrecer archivos o documentos y también podemos ir a copiar desde otra máquina y traerlos a la máquina local. Sería cliente servidor simultaneo a la vez.

Sistema Basado en WEB que en realidad son centralizados y no, la información está centralizada pero el procesamiento está distribuido, o sea toma un poco las partes buenas de los sistemas centralizados y los sistemas clientes servidor.

## Estructura de un Sistema Operativo

Como usar las funciones del Sistema Operativo ???

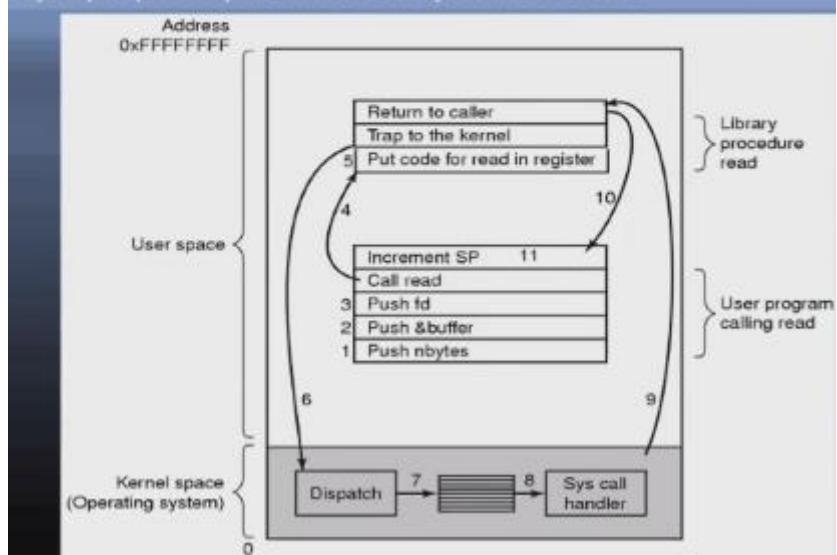


Como haría yo como programador para utilizar las funcionalidades del sistema operativo eso de abstraerme del hardware feo o interfaces feas con el hardware. Como para poner hola mundo a la pantalla sin tener que lidiar con esas interfaces feas.

Hago es una aplicación y lo que hago es hacer una llamada a sistema y el que se tiene que pelear con el hardware feo es el sistema operativo, el sistema operativo tiene componentes, dentro de esos componentes están los Drivers, los manejadores de tal controlador de hardware, y lo único que hago es decirle al operativo que se encargue de hacer eso y el dependiendo del hardware que tenga es lo que va a hacer.

# Estructura de un Sistema Operativo

Ejemplo pasos para hacer un syscall de lectura



Como ejemplo, la llamada a sistema de READ, primero hay una línea que divide el bien y el mal (la parte inferior), modo KERNEL o modo privilegiado, este gráfico tiene muy separado modo usuario y modo KERNEL, el modo usuario donde el nivel de privilegio es 3, y el KERNEL nivel 0. En el 3 tengo todas las instrucciones y no tengo nada entrada-salida, está mi programa que hice en "C" o cualquier otro lenguaje, una vez que se compila o se interpreta se está ejecutando obviamente en lenguaje máquina, y arranco antes de hacer la llamada a sistema que sería CALL READ, lo que hago primero es poner puntualmente READ necesita tres argumentos, entonces lo que se hace primero es ponerse en la pila los tres argumentos que necesita. READ necesita un descriptor de archivo, necesita un BUFFER donde guardar lo que lee de ese descriptor, y

me dice cuántos bytes voy a leer.

Entonces lo que tendría que hacer es pasar esos tres elementos y lo que hace es PUSHAR a la pila entonces los primeros tres pasos son poner y lo hace en orden invertido, porque el primer argumento que lleva READ es el descriptor, después el buffer y después la cantidad de bytes.

```
read manual entry for read in section 2  
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase1$ man 2 read
```

Comando: MAN 2 READ

```
File Edit View Search Terminal Help  
READ(2) Linux Programmer's Manual READ(2)  
NAME  
read - read from a file descriptor  
SYNOPSIS  
#include <unistd.h>  
ssize_t read(int fd, void *buf, size_t count);  
DESCRIPTION  
read() attempts to read up to count bytes from file descriptor fd into the buffer starting at buf.  
On files that support seeking, the read operation commences at the file offset, and the file offset is incremented by the number of bytes read. If the file offset is at or past the end of file, no bytes are read, and read() returns zero.  
If count is zero, read() may detect the errors described below. In the absence of any errors, or if read() does not check for errors, a read() with a count of 0 returns zero and has no other effects.  
According to POSIX.1, if count is greater than SSIZE_MAX, the result is implementation-defined; see NOTES for the upper limit on Linux.  
RETURN VALUE  
On success, the number of bytes read is returned (zero indicates end of file), and the file position is advanced by this number. It is not an error if this number is smaller than the number of bytes requested; this may happen for example because fewer bytes are actually available right now (maybe because we were close to end-of-file, or because we are reading from a pipe, or from a terminal), or because read() was interrupted by a signal. See also NOTES.  
On error, -1 is returned, and errno is set appropriately. In this case, it is left unspecified whether the file position (if any) changes.  
ERRORS  
EAGAIN The file descriptor fd refers to a file other than a socket and has been marked nonblocking (O_NONBLOCK), and the read would block. See open(2) for further details on the O_NONBLOCK flag.  
Manual page read(2) line 1 (press h for help or q to quit)
```

Esto es como invoco esa llamada a sistema desde "C" donde la llamada a sistema se llama READ, le tengo que pasar un descriptor, un puntero a algo a un búfer, y lo cantidad de bytes máximo que quiero leer.

Se puede ejecutar esa lectura, lo que hace es poner en el búfer la cantidad de bytes que leyó y le retorna la cantidad total que bytes que va a ser igual o menor a lo que quise leer.

Entonces volviendo al programa, cuando ejecuto READ paso los argumentos. ejecuto READ pero en realidad no es la llamada a sistema sino lo que hace es hacer un CALL, a un procedimiento de biblioteca y ese es el que hace efectivamente la llamada a sistema, la llamadas sistema es parecida a una función común pero primero pasa de

usuario a modo KERNEL y segundo que no va a cualquier lado, no puedo decirle a dónde, sino que va a un punto específico del KERNEL, una puerta básicamente CALL GATE.

Hacemos en "C" un programa muy sencillo que hace un READ:

```
File Edit View Search Terminal Help
#include <unistd.h>
#include <stdio.h>

int main (){
    char buff[100];
    int leido;
//    scanf("%s", buff);
//    printf("leido del teclado: %s \n", buff);
    leido = read(0, buff, 40);
    write (1,buff, leido);
    return 0;
}
```

Normalmente si quieren leer algo usábamos SCANF y PRINTF, bueno esas son llamadas de alto nivel que en definitiva lo que hacen es invocar al procedimiento de rutina de biblioteca que básicamente lo que hace es REED.

Eso lo estoy obviando y voy a poner directamente REED y WRITE que es para imprimir. REED lleva descriptores de archivo o de canales.

Un descriptor es un número entero que está asociado con un canal.

Cada vez que un proceso arranca o creo un proceso por defecto se crean tres canales o tres descriptores uno de entrada o STDIN que está asociado al teclado, al número cero, al descriptor 0, uno de salida o STDOUT que está asociado a un monitor, hay un canal también asociado al 1 que está asociado a la salida por defecto a la serie estándar y hay otro canal 2 que también está asociado a la pantalla que es el canal de salida de errores.



Entonces lo que estoy haciendo básicamente es decirle que lee desde el canal 0 que es el teclado, que lo guarde en el buffer, hay que pasale un puntero donde almacenar, y digo que lea 40 bytes, obviamente si escribo menos de 40 bytes lo que retorna la llamada a sistema operativo REED es efectivamente lo que leyó, en el caso de haber leído 10 bytes retornara 10 bytes, y después la otra llamada a sistema que hago es WRITE, es parecida, le tengo que pasar un descriptor, el buffer con el contenido que quiero escribir y la cantidad de bytes que voy a escribir.

Al compilar y ejecutar obtenemos la salida:

```
File Edit View Search Terminal Help
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase1$ ./lee
hola
hola
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase1$
```

Lo que puedo hacer es correr "STRACE ./LEE" que me muestra las llamadas a sistema:

Al principio muestra varias porque carga las bibliotecas que usa, lo que hizo fue leer de "0" 40 bytes, en realidad READ retorno 8 porque escribí sólo 8 bytes y después la otra llamada que ejecuto es WRITE en el canal 1 y escribo los 8 bytes.

Vamos a hacer algo más, hacemos un pre compilado, lo que hace es incluir todas las bibliotecas:

```
carlostemaaquinola:/tecnicasIII/2021/cap2/CiaseI$ gcc -E lee.c -o lee.cpp
```

Se genera el archivo lee.cpp y si lo edito:

```
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase1$ gcc -E lee.c -o lee.cpp  
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase1$ vim lee.c
```

```
File Edit View Search Terminal Help

extern int pclose (FILE *_stream);

extern char *ctermid (char *_s) __attribute__ ((__nothrow__ , __leaf__));
# 840 "/usr/include/stdio.h" 3 4
extern void flockfile (FILE *_stream) __attribute__ ((__nothrow__ , __leaf__));

extern int ftrylockfile (FILE *_stream) __attribute__ ((__nothrow__ , __leaf__));

extern void funlockfile (FILE *_stream) __attribute__ ((__nothrow__ , __leaf__));
# 858 "/usr/include/stdio.h" 3 4
extern int __uflow (FILE *);
extern int __overflow (FILE *, int);
# 873 "/usr/include/stdio.h" 3 4
    I
# 3 "lee.c" 2

# 4 "lee.c"
int main (){
char buff[100];
int leido;

leido = read(0, buff , 40);
write (1,buff, leido);
return 0;
}
```

Al final podemos ver el código, arriba muestra todas las definiciones, los ".h", (para salir de VIN: !q)

Ahora el punto siguiente es convertir esto a lenguaje ensamblador:

```
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase1$ gcc -E lee.c -o lee.cpp  
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase1$ vim lee.cpp  
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase1$ gcc -S -x cpp-output lee.cpp -o lee.asm
```

Y esto me genera un programa en ensamblador:

```
File Edit View Search Terminal Help
.file "lee.c"
.text
.globl main
.type main, @function
main:
.LFB0:
.cfi_startproc
.endbr64
pushq %rbp
.cfi_def_cfa_offset 16
.cfi_offset %rbp, -16
movq %rsp, %rbp
.cfi_def_cfa_register 6
addq $-128, %rsp
movq %fs:40, %rax
movq %rax, -8(%rbp)
xorl %eax, %eax
leaq -112(%rbp), %rax
movl $40, %edx
movq %rax, %rsi
movl $0, %edi
call read@PLT
movl %eax, -116(%rbp)
movl -116(%rbp), %eax
movslq %eax, %rdx
leaq -112(%rbp), %rax
movq %rax, %rsi
movl $1, %edi
call write@PLT
movl $0, %eax
movq -8(%rbp), %rcx
xorq %fs:40, %rcx
je .L3
call __stack_chk_fail@PLT
.L3:
.leave
```

```
File Edit View Search Terminal Help
.file "lee.c"
.text
.globl main
.type main, @function
main:
.LFB0:
.cfi_startproc
.endbr64
pushq %rbp
.cfi_def_cfa_offset 16
.cfi_offset %rbp, -16
movq %rsp, %rbp
.cfi_def_cfa_register 6
addq $-128, %rsp
movq %fs:40, %rax
movq %rax, -8(%rbp)
xorl %eax, %eax
leaq -112(%rbp), %rax
movl $40, %edx
movq %rax, %rsi
movl $0, %edi
call read@PLT
movl %eax, -116(%rbp)
movl -116(%rbp), %eax
movslq %eax, %rdx
leaq -112(%rbp), %rax
movq %rax, %rsi
movl $1, %edi
call write@PLT
movl $0, %eax
movq -8(%rbp), %rcx
xorq %fs:40, %rcx
je .L3
call __stack_chk_fail@PLT
.L3:
.leave
```

CALL READ@PLT es la llamada que hace CALL GATE

En esas tres líneas se ve el 40, el buffer, y el 0 que es el descriptor de archivo. Algo que se puede ver que no es como dice en el libro es que no va a la pila.

Todo esto es para mostrar que si bien el usuario hace un PRINTF o SCANF el sistema termina haciendo un CODE GATE, una llamada a sistema.

## Tipos de Sistemas Operativos

### Estructura de un Sistema Operativo

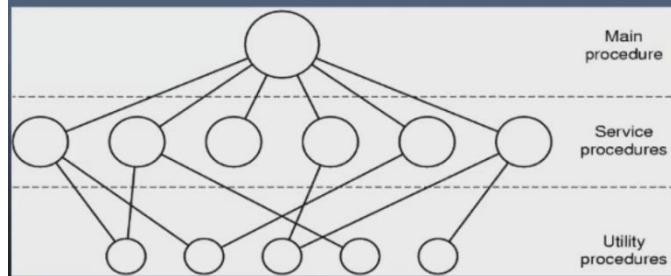
- Monolítico
- En capas
- Microkernel
- Máquinas Virtuales
- Microkernel
- Cliente Servidor
- 

Nos queda es ver cómo están estructurados los sistemas operativos y hay varias formas.

#### MONOLITICO

### Estructura de un Sistema Operativo

- Monolítico



### Estructura de un Sistema Operativo

- Monolítico

- Bien definida las interfaces entre las rutinas
- cualquier rutina puede llamar y ver a cualquier otra.
- Estos componentes son independientes entre si
- Soporte de extensiones :
  - ddl
  - modulos

El monolítico son todas funciones que se compila y es un solo binario que se está ejecutando, igual se podría organizar por servicios, generalmente hay una sola función principal después esa invoca los distintos servicios y por ahí hay también utilitarios o aplicaciones utilitarias que usan algún servicio, lo pro es que desde cualquier servicio podría usar cualquier utilidad del MAIN puede invocar cualquier servicio sin tener ninguna limitación, se ven todos los servicios contra todos los servicios, la contra es que tener bien claras las interfaces porque podría hacer algo mal y eso haría que explote absolutamente todo porque un solo programa, explota y explotó todo.

Linux es monolítico (y los UNIX), la única diferencia es que no carga todo, tiene lo que se llama soportes extensiones que me permite poner lo mínimo y a medida que el operativo necesita algún driver lo que hace es llevarlo a la memoria, es parecido a los DLL de Windows. Hasta el Windows 98 fue monolítico, hoy son híbridos.

La ventaja es muy rápido.

## EN CAPAS

### Estructura de un Sistema Operativo

- En Capas
  - Capas Gerárquicas, cada uno construida sobre la anterior.
  - Cada capa se comunica con la adyacente
  - Menor throughput que los kernels monolíticos.
- Estructura de Sistema Operativo "THE" Dijkstra, Multics

Layer	Function
5	The operator
4	User programs
3	Input/output management
2	Operator-process communication
1	Memory and drum management
0	Processor allocation and multiprogramming

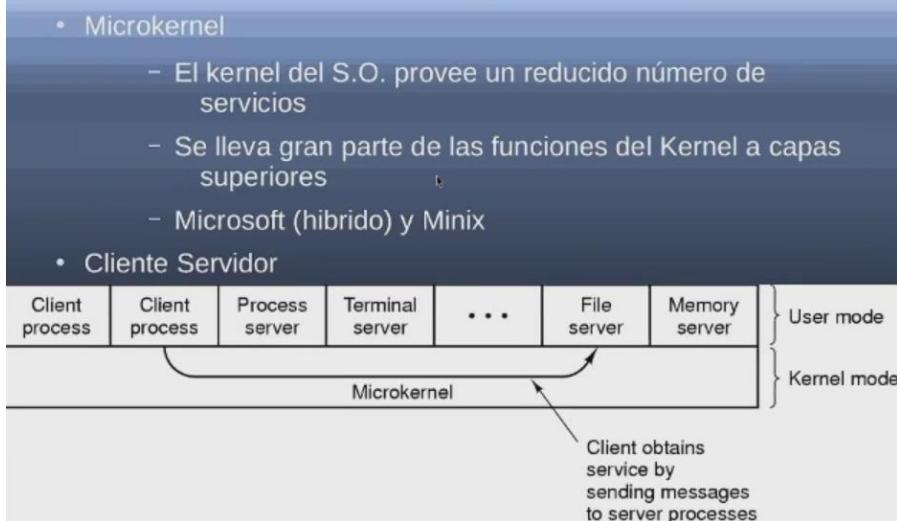
Va armando capas donde cada capa le da un servicio a la anterior, por ejemplo la capa 0 que está debajo de todo lo que hace es básicamente es solo multiplexación de tareas en la CPU, entonces después de la capa 1 no interesa si hay varias tareas sino que de lo único que se encarga de asignar memoria, entonces ya tengo asignación de memoria y SWITCHEO entre las tareas desde la CPU, entonces la capa 2 de lo único que se encarga es de lo que se llama OPERATOR-PROCESS COMMUNICATION solamente eso, porque ya sabe que alguien más se encarga de la memoria y alguien más se encarga de multiplexar, después la otra capa hace de entrada/salida, en la otra son los programas de usuarios y la última es la interacción con el usuario.

Es mucho más lento, porque para hacer algo tengo que ir capa por capa pidiendo los servicios.

## MICROKERNEL

### Estructura de un Sistema Operativo

- Microkernel
  - El kernel del S.O. provee un reducido número de servicios
  - Se lleva gran parte de las funciones del Kernel a capas superiores
  - Microsoft (hibrido) y Minix
- Cliente Servidor



Hay otro que se llama MICRKERNEL que ese si usa MINIX y básicamente lo que hace es dejar solo las funcionalidades principales en modo KERNEL, el resto de las cosas como por ejemplo manejo de sistemas archivo, manejo de memoria, manejo de procreación de procesos, todo ese tipo de cosas las hace un programa en modo usuario, ¿Por qué? tratando de tener más confiabilidad, porque si estuviera todo en KERNEL y hay un BAG no sé en un módulo si estuviese en KERNEL explota todo, en cambio sí está en modo usuario se rompe ese programa pero el

resto sigue andando. Quizá hay un módulo que se encarga periódicamente de ver que todos los módulos del sistema operativo en modo usuario estén funcionando, y si no está crea un proceso nuevo que se encarga de hacer funcionar lo que dejó de andar.

Tiene menos seguridad.

Windows tiene algo así pero algunas cosas en KERNEL y otras en modo usuario, se llama híbrido cuando es así.

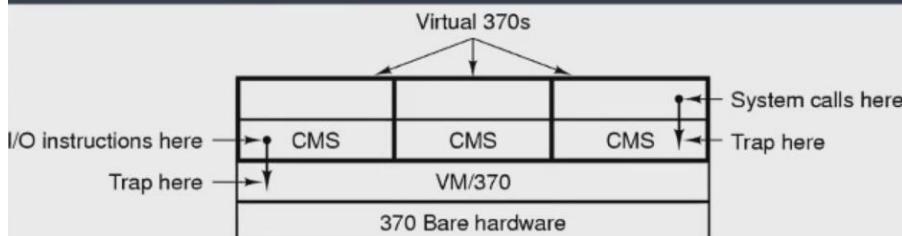
MINIX es totalmente MICROKERNEL.

Hay otro tipo muy parecido al MICROKERNEL que se llama CLIENTE SERVIDOR, funciona de la misma manera, la única diferencia es que los distintos procesos a modo usuario están en distintas máquinas, en distintos clientes o servidores, y sería una especie de sistema operativo distribuido. [profundizar con el tanenbaum](#)

## **MAQUINAS VIRTUALES**

### **Estructura de un Sistema Operativo**

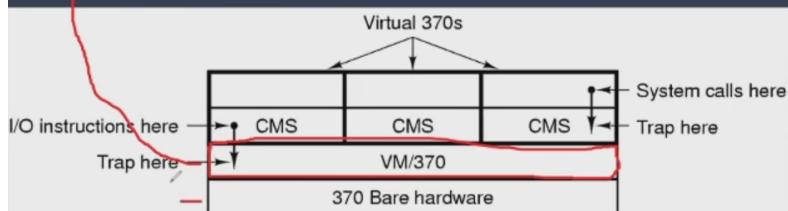
- Máquinas Virtuales
  - Un sistema de time sharing provee:
    - multiprogramación
    - máquina extendida
  - La escencia es separar completamente esas dos funciones.
- Estructura de VM/370 con CMS



Se basa en el concepto que hablamos al principio donde puedo multiplexar el hardware y hago multiprogramación y por otro lado tengo una máquina extendida donde tengo una abstracción de las interfaces feas, lo que vieron la gente de IBM en ese momento, estamos hablando de los 70', es que si separo estas dos funciones totalmente podría hacer lo siguiente: un sistema operativo que solo se encargue de la multiprogramación, entonces tengo el hardware y este sistema operativo básicamente se encarga de poder correr varios programas simultáneamente.

### **Estructura de un Sistema Operativo**

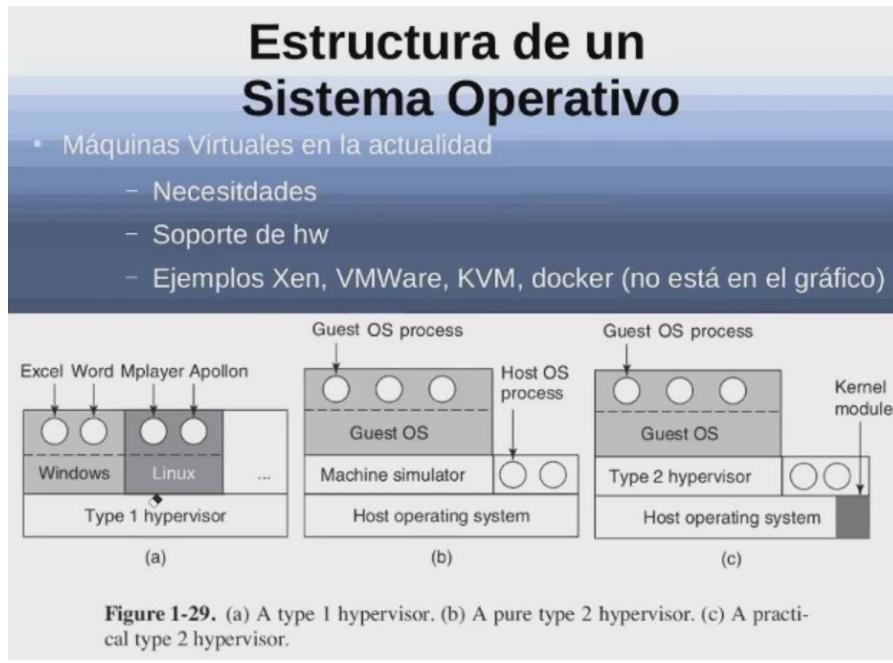
- Máquinas Virtuales
  - Un sistema de time sharing provee:
    - multiprogramación
    - máquina extendida
  - La escencia es separar completamente esas dos funciones.
- Estructura de VM/370 con CMS



Entonces de alguna manera, aislados, separa los distintos programas, tengo tres programas y se ejecutan y esto como no da una máquina extendida básicamente sólo hace multiplexación, entonces estos programas que pongo arriba se creen que están interactuando con el hardware porque no tengo ninguna abstracción, o sea, no es que ponga un interfaz como READ o WRITE, no tengo ninguna abstracción, o sea, si quiero manejar con un programa que está acá arriba un periférico tengo que escribir el programa, porque lo único que me hace este operativo es multiplexación de los distintos procesos que tengo acá arriba.

¿Entonces cuál es el cual el truco? bueno este operativo hacer multiplexación y después debe poner distintos procesos arriba, lo que hago es instalar distintos sistemas operativos, estos sistemas operativos, como no tengo una interfaz que me dé una máquina extendida, se creen que están hablando con el hardware pero en realidad están hablando con esta capa, y esta capa es la que se encarga de interactuar con el hardware.

## MAQUINAS VIRTUALES EN LA ACTUALIDAD



¿Para que tener máquinas virtuales? básicamente para ahorrar hardware, si tenemos un solo hardware y necesitas tener varios servicios corriendo en tu empresa, por ahí en un mismo sistema operativo configurar varios servicios es un lío, me refiero por ahí modificas un servicio y sin querer influye en el otro o se rompe el otro, entonces lo mejor es configurar un servicio en cada sistema operativo distinto. Si puedo tener un solo hardware y virtualizar ahí, y tener tres sistemas operativos distintos y en cada uno configuro una cosa y listo, no interactúan entre sí.

Hay varios sabores de esto, hay uno que no figura acá pero es el que actualmente se está usando más.

El hipervisor como decía IBM, la ventaja es que se le dio soporte hardware a los procesadores de Intel para lograr esa virtualización. La otra alternativa, es un paso intermedio que no se usa prácticamente, es tener un sistema operativo, porque instale un hipervisor que no me da ninguna máquina extendida, entonces arriba instalo Windows, Linux o lo que fuera, un sistema operativo de cero como si estuviera en el hardware. La otra alternativa es poner un sistema operativo con algunos módulos de KERNEL que manejan el tema de multiplexación de distintos clientes entonces el hipervisor es el que me hace la gestión de los distintos clientes, este caso es por ejemplo virtual box, se necesita un sistema operativo que pueda trabajar con ese hipervisor, de hecho si vos lo instalas en Linux y por defecto no lo tenía te instala un KERNEL nuevo que tenga módulos que interactúan con ese hipervisor, y el que se encarga de gestionar los clientes es el hipervisor virtual box.

## Virtualización DOCKER

En esta virtualización no es a nivel hardware a nivel sistema operativo si no que es a nivel aplicación, tengo una máquina y lo que hago es a las aplicaciones directamente las separo entre sí y no tienen acceso a nada, es una manera de empaquetar aplicaciones con bibliotecas todas juntas, ¿qué ventajas tiene eso? en este caso supongan que tengo un sistema operativo que tiene mi aplicación, están las bibliotecas que utiliza, están las llamadas al

sistema del sistema operativo GUEST y después de eso están las llamadas sistema del sistema operativo HOST o sea tengo un montón de capas por las que tiene que pasar mi aplicación antes de ejecutarse en el hardware, si virtualizo mi aplicación (dentro de la misma aplicación entiendo) tiene mi programa, las bibliotecas, no usa KERNEL, la clave es que no tiene un KERNEL, o sea esto no es un sistema operativo, es solo mi aplicación con las bibliotecas que usa, lo que hace es usar el sistema operativo del HOST que usa el hardware, ¿qué me ahorro? Un montón de capas que tiene de más, ¿cuál es la única desventaja? que obviamente puedo virtualizar un montón de máquinas pero todas para el mismo sistema operativo que tengo instalado, por ejemplo Linux.

Lo bueno es que hay millones de este tipo de máquinas pre armadas con todas las bibliotecas y las aplicaciones que ustedes quieran, esto así se llama contenedor. Hoy en día se está usando muchísimo más.

## PREGUNTAS:

**1\_ Indique que arquitecturas son usadas de sistemas operativos**

Máquina Virtual

**2\_ Indique el orden correcto de pasos en una llamada a sistema**

PUSH, biblioteca, procedimientos, TRAP, HANDLER, retorno, PULL

**3\_Cual de las siguientes características de sistemas operativos se “redescubrieron” recientemente:**

Computación como un servicio

**4\_Cual de las siguientes características fundamentales debe ofrecer un sistema operativo de propósito general.**

Gestión de memoria

## FUNCIONES ESPECIALES DE UN SISTEMA OPERATIVO

### **FUNCIONES PRINCIPALES DE UN SISTEMA OPERATIVO**

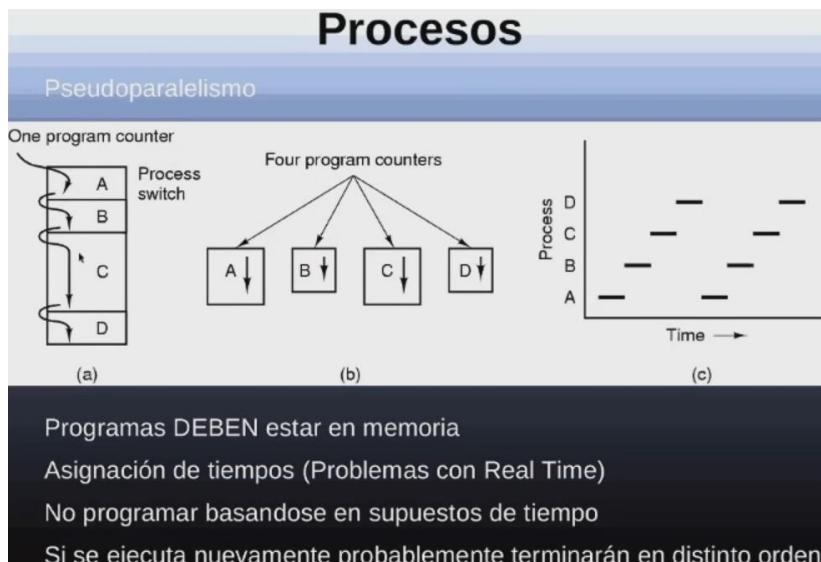
- PROCESOS:
  - Necesidad y Definición
  - Creación/Terminación de procesos
  - Estados de procesos / Cambios
  - Jerarquía de procesos
  - Implementación de procesos en Linux
- SEÑALES
  - Distintos tipos de Señales
  - Manejador
  - Implementación de señales en Linux

## Procesos

- Velocidad Procesador
  - Velocidad Entrada Salida- Pseudoparalelismo
  - Un solo procesador
  - Una instrucción por vez
  - ¿Cómo lo logró?

Vamos a ver procesos, la necesidad, que es, como creo y termino procesos, estados, algo de jerarquía y algunos ejemplos. Señales.

¿Porque quiero tener más de un proceso?, más de un programa ejecutándose, bueno básicamente porque la velocidad del procesador comparada con la entrada-salida es bastante más rápida, entonces si un programa lo podemos ejecutar y está único en la memoria, si lo mando a hacer entrada-salida el procesador se aburre de no hacer nada. Entonces una alternativa de eso es hacer uso de paralelismo, pero si tenemos un solo procesador la pregunta es cómo ejecutamos una instrucción, ¿cómo hacemos paralelismo?, no se puede hacer paralelismo con un solo núcleo, con un solo procesador, lo que se hace es un pseudoparalelismo.



Tengo en la memoria principal varios programas que están ejecutándose, que están en la memoria, tenemos un gráfico en función del tiempo y los programas que ejecuto, entonces en el primer instante empiezo a ejecutar instrucciones del programa A, entonces es como si nosotros viéramos que se viene ejecutando esa parte del código, después, en algún momento se empieza a ejecutar código del programa B, entonces el contador programa de repente va a saltar y va a empezar a ejecutar código del B, va a ejecutar un rato y después va a ejecutar algo del C, después de algo del D, y periódicamente en algún momento va a volver al A y va a continuar.

En definitiva el efecto que me hace esto es que veo como si los cuatro programas que se están ejecutando, se estuvieran ejecutando a la vez, y eso es lo que llamamos pseudoparalelismo, si tengo un solo procesador voy como repartiendo, multiplexando en el tiempo la ejecución del código de cada uno de ellos.

Primero, los programa los debo tener en la memoria principal para que se ejecuten, si no se pueden ejecutar, otro tema es que si vamos a hacer cosas de tiempo real, este tipo de mecanismo no es de lo de lo más fiable, porque no puedo hacer programas basándome en algunos supuestos temporales, porque en definitiva el grafico del tiempo está muy bonito, le da un poquito de tiempo a "A", después a "B", después a "C", para después volver a "A", pero esto no siempre va a ser así, entonces el tema es que si corro estos cuatro programas en algún momento pueden terminar de distinta manera, tal vez termine el "C" primero que el "A", no puedo de antemano saber cómo van a terminar, eso es algo muy importante.

## Procesos cont.

¿Qué es un proceso?

Programa (entidad pasiva) en ejecución

+ valor del contador de programa

+ valor de los registros

+ pila del proceso (parámetros de funciones, direcciones de retorno, etc.)

**PROCESO (entidad activa)**

- Analogía

Entonces ¿qué es un proceso? Programa y procesos son similares pero no es lo mismo, un programa es una receta de pasos, tengo el programa que arranca con un MAIN y una serie de pasos que se van a ejecutar, entonces no sólo alcanza con decir que el proceso es un programa en ejecución, tengo que decir además por dónde va en algún momento, tengo que agregarle o sumarle el contador de programa, tengo que sumarle el estado de todos los registros internos, tengo que saber la pila de ese proceso, qué cosas tiene y alguna otra cosa más como los descriptores abiertos, etcétera, y eso en definitiva me da lo que es un proceso que es una entidad activa, el proceso es lo que se está ejecutando pero con todo su contexto, es como sacar una foto de ese programa en ejecución y supiera el valor de los todos los registros, de todos sus estados, etcétera, en un momento determinado.

¿Porque quiero tener todos esos datos? porque justamente cuando se está ejecutando el programa puedo de alguna manera pausarlo y continuarlo después, pero tengo que continuarlo después tal cual estaba, eso no lo hago yo programador sino el sistema operativo, lo pausa cuando él crea que es conveniente. Entonces el proceso es una abstracción de la primera abstracción que creamos y eso nos permite nos permite hablar de que tengo cuatro programas pero en definitiva son cuatro procesos que se están ejecutando, porque cuando conmuto entre uno y otro lo que hago es salvar el contexto. Lo que habíamos visto en el TASK STATE SEGMENT, básicamente eso, lo guarda el hardware más el operativo que guarda alguna otra cosa más.

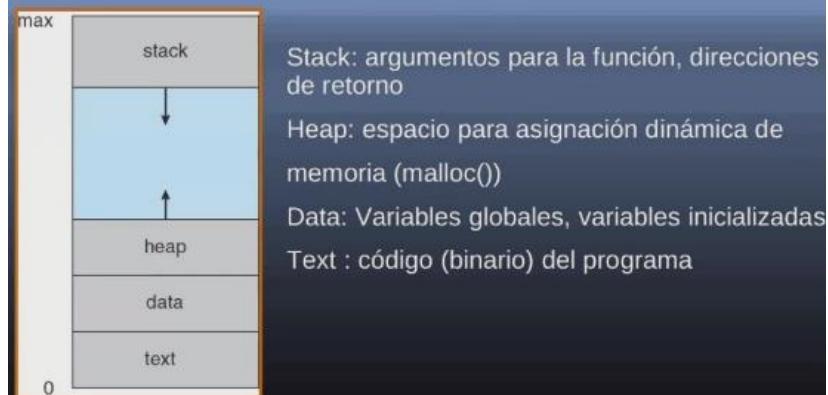
ANALOGIA:

Suponiendo que estamos en una pizzería y el cocinero tiene la receta para hacer una pizza donde tiene todos los pasos y todos los ingredientes, y se pone a ejecutar esa receta, se pone a hacer un proceso de elaborar una pizza y obviamente la CPU sería el cocinero. La mesa 3 pidió una mozzarella, entonces él la empieza a hacer, toma los ingredientes, le pone sal, la harina, mezcla y en ese momento viene el mozo y le dice que tiene al dueño de la pizzería y está en la mesa 4 pero me pide una serrana con jamón y rúcula, ¿qué hace el cocinero? ¿Qué hace?... *es el jefe*, deja lo que está haciendo y se pone a hacer otra pizza, busca la receta de la otra pizza y empieza a seguir los pasos, una vez que termina, cuando vuelve, recuerda que estaba haciendo la mozzarella para la mesa 3, el tema es que si no se acuerda si le había echado sal o no, ¡está en el horno! Porque le va a salir horrible, entonces ¿qué debería haber hecho antes de empezar a hacer la pizza para su jefe? haber salvado, guardar todo.

Entonces un cocinero que es la CPU ejecutando una receta es lo que se llama un proceso.

## Procesos cont.

Ubicación del proceso en memoria:



Tengo un proceso de un programa en ejecución, cuando empieza a ejecutarse obviamente tengo que asignarle el CODE SEGMENT, el DATA SEGMENT, STACK SEGMENT, todos los SEGMENT, y en el en el espacio virtual del proceso (memoria que se le asignó al proceso) el CODE SEGMENT se llama área de texto, segmento de texto, donde está el binario, el programa en lenguaje máquina que se va a ejecutar, tengo el segmento de datos donde tengo las variables que se guardan en el proceso, las que definí estáticamente, las que no inicialice, y después tengo dos los segmentos que van a ir variando durante la ejecución, el segmento de pila y el HEAP.

¿Que guarda la pila, porque varía? lo que va a guardar en esta pila, no es lo mismo que se guarda en el TASK STATE SEGMENT (TTS), cuando un proceso se deja ejecutar el operativo les saca una foto y guarda todo en TSS, no en la pila, la pila la manejo en mi programa haciendo PUSH y POP, guardo cosas y saco cosas pero no lo puedo hacer cuando me dejen de ejecutar, porque no tengo noción de cuándo me sacan la CPU. En esa pila guarda el contador del programa para saber desde donde invoco una subrutina, una función, y saber dónde debe retornar.

Podría tener rutinas anidadas entonces podría crecer y bastante, por eso se le deja una área para la pila.

## Creación de Procesos

¿Para qué creo procesos?

- Inicialización del Sistema
  - Init o systemd
  - Servicios
    - Background, foregraund (bg,fg, &, at)
- Proceso hace llamada a sistema "fork()"
  - Crea un nuevo proceso ... copia...
- Desde Interprete de comandos (o doble "click")
  - Para sistemas interactivos
- Nuevos trabajos batch
  - Para sistemas de procesamiento por lotes

Cuando arranca el sistema tengo varios procesos que se crean que realmente lo van a conocer como servicios, como DAEMON, mal traducido demonios, en realidad son algunos tipos de monitores y lo que hacen es hacer una tarea, al prender mi máquina y por más que no lance ningún programa, podría ver que hay un programa que se pone a ver si me llegan correos, un programa que está periódicamente chequeando si hay algún virus, y un montón de programas que no los veo, no tengo interacción, pero arrancan en el inicio del sistema cuando arranco mi máquina.

La característica es que están en BACKGROUND, mi descriptor de salida que está asociado a la pantalla y los de entrada que están asociados al teclado por ejemplo, están desconectados, cuando están desconectados, mi proceso se ejecuta solo pero no tengo forma de ver, la única manera de verlo con el comando top.

```

File Edit View Search Terminal Tabs Help
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$ top
top - 21:26:38 up 10 days, 12:00, 0 users, load average: 2.35, 2.39, 2.41
Tasks: 290 total, 1 running, 289 sleeping, 0 stopped, 0 zombie
%Cpu(s): 16.5 us, 10.9 sy, 0.0 ni, 71.6 id, 0.0 wa, 0.0 hi, 1.0 si, 0.0 st
MiB Mem : 15743.8 total, 299.0 free, 9713.2 used, 5731.6 buff/cache
MiB Swap: 17408.0 total, 17375.5 free, 32.5 used, 5565.4 avail Mem

      PID USER      PR  NI   VIRT   RES   SHR S %CPU %MEM    TIME+ COMMAND
384097 carlost  20   0 6424528  1.1g 137480 S 105.6  7.1 247:30.49 zoom
382772 root     20   0 811036 171664 121108 S 54.6  1.1 105:06.96 Xorg
387175 carlost  20   0 5948072  2.7g 166032 S 27.8 17.7 67:35.59 Web Content
246012 carlost  9 -11 1215552 19940 15484 S 15.2  0.1 56:13.23 pulseaudio
402226 carlost  20   0 3221032 495740 152024 S 7.9  3.1 11:30.19 Web Content
383341 carlost  20   0 3137604 493196 182608 S 5.6  3.1 46:54.62 Web Content
383102 carlost  20   0 4141948  1.1g 228176 S 5.3  7.0 95:35.60 Firefox
402744 carlost  20   0 3619352 843680 193620 S 3.0  5.2 20:06.02 Web Content
383578 carlost  20   0 3763420 902956 180416 S 1.7  5.6 29:14.07 Web Content
392749 carlost  20   0 3772692 832444 191428 S 0.7  5.2 11:39.01 Web Content
401688 carlost  20   0 3021688 311260 168056 S 0.7  1.9 3:32.14 Web Content
412987 carlost  20   0 9516  4164 3364 S 0.7  0.0 0:43.96 top
382848 carlost  20   0 8872  4260 2984 S 0.3  0.0 0:14.33 dbus-daemon
383471 carlost  20   0 3227312 491296 168932 S 0.3  3.0 13:58.63 Web Content
383646 carlost  20   0 553400 56444 39548 S 0.3  0.4 1:34.75 gnome-terminal-
414211 root     20   0 0     0     0 I 0.3  0.0 0:03.25 kworker/7:1-events
420227 root     20   0 0     0     0 I 0.3  0.0 0:05.75 kworker/6:0-events
422091 carlost  20   0 9384  4216 3416 R 0.3  0.0 0:00.04 top
  1 root     20   0 170436 10972 7004 S 0.0  0.1 0:15.81 systemd
  2 root     20   0 0     0     0 S 0.0  0.0 0:00.29 kthreadd
  3 root     0 -20 0     0     0 I 0.0  0.0 0:00.00 rcu_gp
  4 root     0 -20 0     0     0 I 0.0  0.0 0:00.00 rcu_par_gp
  8 root     0 -20 0     0     0 I 0.0  0.0 0:00.00 mm_percpu_wq
  9 root     20   0 0     0     0 S 0.5  0.0 0.0 0:14.84 ksoftirqd/0
 10 root     20   0 0     0     0 I 0.0  0.0 2:46.28 rcu_sched
 11 root     rt   0 0     0     0 S 0.5  0.0 0.0 0:01.64 migration/0
 12 root     .51  0 0     0     0 S 0.5  0.0 0.0 0:00.00 idle_inject/0
 14 root     20   0 0     0     0 S 0.5  0.0 0.0 0:00.00 cpuhp/0
 15 root     20   0 0     0     0 S 0.5  0.0 0.0 0:00.04 cpuhp/1
 16 root     .51  0 0     0     0 S 0.5  0.0 0.0 0:00.00 idle_inject/1
 17 root     rt   0 0     0     0 S 0.5  0.0 0.0 0:01.72 migration/1
 18 root     20   0 0     0     0 S 0.5  0.0 0.0 0:16.24 ksoftirqd/1
 21 root     20   0 0     0     0 S 0.5  0.0 0.0 0:00.03 cpuhp/2

```

Esto me da el top ten de los procesos que están ejecutándose. Se ve el Firefox dado que se está utilizando, el SYSTEMD (padre de todos), el que maneja el sonido, muchos procesos que se están ejecutando. Lo desconozco porque no tienen asociada la entrada ni la salida. Procesos en segundo plano.

Estos procesos están para hacer cosas automatizadas, desatendidas de mi parte.

Dentro de mi programa lo que hago es crear nuevos procesos (**FORK()**), lo que hace es una copia de mi programa.

¿Para que en mi programa tendría que crear otro programa? Y esto lo hago para realizar una tarea específica. Supongamos que tengo un programa monolítico que realiza una tarea, tiene un montón de partes y va realizando la tarea 1, la tarea 2, etc. Si una de las tareas se demora mucho porque está haciendo entrada salida, mi proceso no podrá seguir hasta que no le llegue esa entrada salida.

La idea es hacer que las otras tareas sean como un programa aparte, entonces mientras la otra está bloqueada esperando entrada salida puedo seguir ejecutando las otras. De esta manera voy ganando en velocidad.

Cuando tengo tareas que las puedo dividir en partes, que puedo implementarlas en distintos programas que trabajan de manera colaborativa me sirve porque se ejecuta mucho más rápido.

Cuando quiero hacer algo nuevo, como por ejemplo cuando le doy doble CLICK a algo o ejecuto un comando, como recién cuando enviamos el comando TOP, se ejecuta ese proceso y nos muestra.

La otra alternativa es para sistemas de procesamiento por lotes. Crear un nuevo conjunto de tareas a ejecutarse.

```
File Edit View Search Terminal Tabs Help  
carlost@maquinola: ~/  
  
#include <stdio.h>  
#include <stdlib.h>  
#include <unistd.h>  
  
int main()  
{  
    printf("soy el padre\n");  
    fork();  
    printf("cree el hijo\n");  
    return 0;  
}
```

Tengo un programa donde mi punto de entrada al programa es MAIN(), lo que hace es mostrar un cartel diciendo "SOY EL PADRE" luego el FORK crea un hijo, y luego muestra "CREE EL HIJO".

```
File Edit View Search Terminal Tabs Help  
carlost@maquinola: ~/tecnicasIII/2021/cap2/Clase2  
carlost@maquinola: ~/tecnicasIII/2021/cap2/Clase2$ vim fork1.c  
carlost@maquinola: ~/tecnicasIII/2021/cap2/Clase2$ make fork1  
make: 'fork1' is up to date.  
carlost@maquinola: ~/tecnicasIII/2021/cap2/Clase2$ ./fork1  
soy el padre  
cree el hijo  
cree el hijo  
carlost@maquinola: ~/tecnicasIII/2021/cap2/Clase2$
```

Tengo un programa donde en el área de código está el código escrito, comienza a ejecutarse hasta que muestra el primer cartel y luego hace la llamada a sistema FORK y el operativo se da cuenta que tiene que crear un hijo, FORK crea ese hijo, el sistema operativo le asigna espacio, y lo que hace es copiar toda el área de texto, toda el área de datos, toda el área de la pila, copia todo absolutamente igual incluyendo también los estados. También se copia el contador de programa, entonces arranca la vida ese proceso en la siguiente vida.

PID, PROCESS IDENTIFICATION, es un numero entero y único que se va incrementando, cada proceso en ejecución tiene un ID distinto. Enteros positivos mayores que 1.

Se puede hacer una distinción entre padre e hijo, la llamada a sistema la invoca el padre que es el único proceso que existe en ese momento, cuando termina esa llamada a sistema, lo que retorna es el PID del proceso que creó, del hijo que creó, al hijo le retorna 0, en el hijo como no tiene hijo no le retorna el PID de un hijo,

```
File Edit View Search Terminal Tabs Help  
carlost@maquinola: ~/tecnica  
  
#include <stdio.h>  
#include <stdlib.h>  
#include <unistd.h>  
  
int main()  
{  
    int pid;  
    printf("soy el padre\n");  
    pid = fork();  
    //hijo  
    if (pid == 0){  
        printf("soy el hijo\n");  
        return 0;  
    }  
    printf("cree el hijo\n");  
    return 0;  
}  
  
carlost@maquinola: ~/tecnicasIII/2021/cap2/Clase2$ make fork1  
cc      fork1.c   -o fork1  
carlost@maquinola: ~/tecnicasIII/2021/cap2/Clase2$ ./fork1  
soy el padre  
cree el hijo  
soy el hijo
```

```

File Edit View Search Terminal Tabs Help
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main()
{
    int pid;
    printf("soy el padre\n");
    pid = fork();
    //hijo
    if (pid == 0){ // hijo
        printf("soy el hijo\n");
        // return 0;
    }
    printf("cree el hijo\n");
    return 0;
}

```

```

File Edit View Search Terminal Tabs Help
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$ ./fork1
soy el padre
cree el hijo
soy el hijo
cree el hijo
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$ 

```

Si comento el RETURN voy a ver que el hijo aparece dos veces, uno lo estaría haciendo el proceso padre y el otro el hijo. Con el RETURN 0 me aseguro que lo que sigue no lo haga el hijo.

Como no tenemos ni idea que va a hacer el planificador, no podremos saber que se ejecuta primero, si el padre o el hijo. Una vez que hago FORK hay dos procesos y tienen vida propia, hay una parte del operativo que se encarga de ver quien se ejecuta primero, se llama planificador.

El EXIT STATUS es un valor en el PCB, en la tabla del proceso, que me dice una vez que termine el proceso, como termine, y el padre lo puede consultar, si pongo ECHO \$? el intérprete de comando, en esa variable almacena como termine el proceso que ejecute.

```

carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$ echo $?
0
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$ 

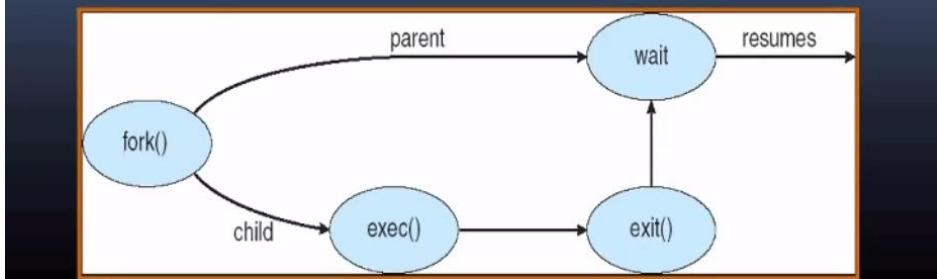
```

Me dice que termine en 0. Por convención cuando termina bien un proceso termina con 0, cuando termina mal por algún error termina con otro valor.

## Creación de Procesos con fork()

Alternativas Padre:

- Ejecuta concurrentemente con hijo
- Espera que finalize el hijo
- Alternativas Hijo:
  - Usa mismo programa y datos que heredó padre
  - Carga un nuevo programa



Acá podemos ver un par de alternativas que se pueden presentar.

Viene un solo proceso que se está ejecutando y de repente hace un FORK, se sigue ejecutando el proceso padre y comienza a ejecutarse un proceso que es el hijo. El padre tiene dos alternativas, una es esperar que el hijo termine o seguir.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main()
{
    int pid;
    printf("soy el padre\n");
    pid = fork();
    //hijo
    if (pid == 0){
        sleep(1);
        printf("soy el hijo\n");
        return 0;
    }
    printf("cree el hijo\n");
    return 0;
}
```

```
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$ ./fork1
soy el padre
cree el hijo
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$ soy el hijo
■
```

Demoro al hijo con 1seg, entonces podemos ver que termina el padre, y el padre del padre que es el intérprete de comando ve que ya termino y me tira el PROMPT, pero después al rato al rato termina el hijo y escribe esa línea, este sería el caso donde el padre no lo espera al hijo. Al no esperarlo no puede leer el EXIT STATUS de como terminó el hijo.

La otra alternativa es que el padre crea al hijo y luego el proceso se bloquea hasta que el hijo termine, y WAIT es la llamada a sistema que utilizo para esperarlo.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

int main()
{
    int pid;
    printf("soy el padre\n");
    pid = fork();
    //hijo
    if (pid == 0){
        sleep(1);
        printf("soy el hijo\n");
        return 0;
    }
    wait(NULL);
    printf("cree el hijo\n");
    return 0;
}
```

```
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$ ./fork1
soy el padre
soy el hijo
cree el hijo
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$ ■
```

Entonces acá le digo que espere que termine el hijo, y luego que siga. El NULL se lo pongo porque en este caso no me interesa saber cómo termina el hijo.

No siempre es necesario esperar al hijo, por ejemplo si tengo un proceso que es un servidor web, el cual espera que se conecte un cliente que le pide un archivo, lo busca y se lo entrega y luego se queda esperando que se conecte otro cliente. Normalmente para tener mayor respuesta lo que hago es, cada vez que se conecta un cliente creo un hijo que se encargue de responderle a ese cliente, entonces si tengo varios clientes que se conectan creo "N" hijos y cada uno le responde a su cliente. Pero el proceso padre no necesariamente tiene que esperar a que terminen todos, porque tal vez uno pide descargar una imagen ISO y puede demorar horas, entonces el padre no puede esperar dos horas, porque si viene un cliente nuevo no va a poder atenderlo. Estos serían los casos concurrentes donde el proceso padre no debe esperar al hijo.

Con respecto al hijo puede usar el mismo programa y los datos que heredo del padre, y a partir de ahí con un IF o con un SWITCH ejecutar algo de código distinto.

```

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

int main()
{
    int pid;
    printf("soy el padre pid = %d y mi padre es %d \n", getpid(), getppid());
    pid = fork();
    //hijo
    if (pid == 0){
        sleep(1);
        printf("soy el hijo\n");
        return 0;
    }
    wait(NULL);
    printf("cree el hijo\n");

    return 0;
}

```

```

carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$ ./fork1
soy el padre pid = 424236 y mi padre es 388229
soy el hijo
cree el hijo
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$ ./fork1
soy el padre pid = 424243 y mi padre es 388229
soy el hijo
cree el hijo
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$ ps
  PID TTY      TIME CMD
 388229 pts/4    00:00:00 bash
 424261 pts/4    00:00:00 ps
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$ 

```

GETPID me da el del hijo.

GETPPID me dice quién es el padre.

El PID del padre cambia cada vez que lo ejecuto, es como la misma receta de pizza pero para mesas diferentes, mientras que el padre del hijo se puede ver que es el mismo, es el intérprete de comando, el BASH. Tener en cuenta que cuando se reinicia la consola también puede cambiar.

Muchas veces pasa que si tengo que hacer cosas muy distintas tengo que meter en el mismo código del padre también el código del hijo, y si son varios hijos queda un montón de código, hay que recordar que al hacer FORK se duplica todo el binario, el segmento de texto, entonces tendría mucha memoria ocupada con código que no lo uso siempre, porque el padre usa una parte de código, el hijo usa otra. Una alternativa a esto, es que una vez que está corriendo mi proceso, el cual tiene un segmento de código, segmento de datos, la pila, si ejecuto EXEC y le paso como argumentos que binarios quiero, lo que hace es ir y en la parte de código pone el nuevo binario, lo reemplaza, desaparece lo que estaba escribiendo y aparece todo nuevo y el contador de programa se pone en el punto de entrada de ese programa. Entonces con EXEC en el cuerpo del proceso hijo pone otro binario, que no estaba ahí, para hacer algo completamente distinto.

# Terminación de Procesos

## Tipos de terminación

### - Terminación Normal

- Tareas que realiza el SO → exit()
- echo \$?

### - Terminación con Error Voluntaria

- exit(!=0) echo \$?
- standard error

### - Terminación con Error Involuntaria

- No hay mas memoria
- Violación de Segmento
- División por cero

### - Recepción de una señal (Que es una señal?)

- Comando kill, llamada a sistema “kill()”
- Handler de señales, llamada a sistema “signal()”

Obviamente los programas no son eternos, tienen que terminar, una vez que hacen lo que necesitamos que hagan tiene una terminación normal, cuando ejecuto una tarea le pongo EXIT o RETURN 0 y luego si desde el sistema operativo hago un ECHO \$? me dice como retorno.

```
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$ ls
Clase2.odp  fatal  fork1  fork2  fork3  fork4  fork5  serial0.c  serial2.c  serial4.c
Clase2.pdf  fatal.c  fork1.c  fork2.c  fork3.c  fork4.c  fork5.c  serial1.c  serial3.c  serial5.c
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$ echo $?
0
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$ ls pepe
ls: cannot access 'pepe': No such file or directory
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$ echo $?
2
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$
```

Si hago un LS me muestra mi directorio, y luego si consulto me muestra 0 porque ese proceso se ejecutó con éxito.

Pero si quiero agregar un error voluntario podría buscar el archivo PEPE, el cual no existe, y me muestra un error, luego al verificar con ECHO \$? no me da 0, me da 2.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main()
{
    printf("Soy un proceso que terminara involuntariamente \n");
    printf ("operacion %d\n", 100/0);
    return 0;
}
```

```
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$ ./fatal
Soy un proceso que terminara involuntariamente
Floating point exception (core dumped)
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$ echo $?
136
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$
```

Fatal es un programa que intenta dividir por 0, para que de un error involuntario, se puede ver que da el error y al verificar con ECHO \$? da 136.

Otra forma de terminar los procesos es cuando reciben una señal.

Una señal es una interrupción por software, no es un hardware, aunque funciona muy parecido. Entonces a un proceso le llega una señal que es un evento de software, lo que hace es ejecutar una rutina de atención a esa señal, por defecto cada señal tiene un comportamiento. Podría modificarlo para que haga otra cosa.

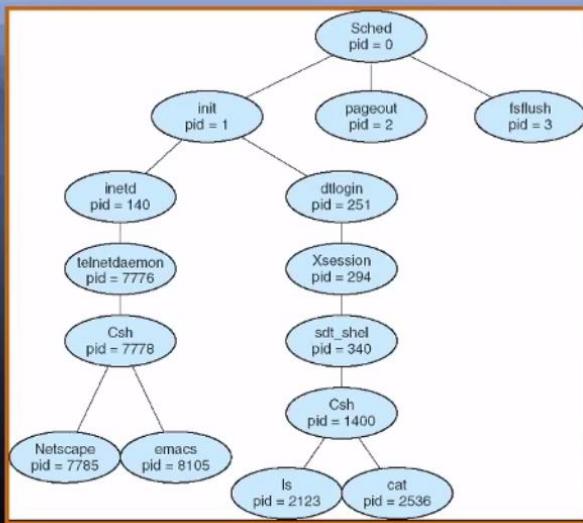
Por ejemplo en caso de llegar la 9 SIGKILL termina el proceso, entonces podría hacer un programa que no haga nada, solo tener una demora de 100 segundos, mientras eso está en ejecución pudo hacer un KILL -9 PID, PID del proceso y termina. Y esto puede ser necesario porque puede pasar que mi proceso esté haciendo algo mal o este en un bucle del que no salga nunca entonces debería tener una forma externa de detenerlo.

```
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$ kill -l
 1) SIGHUP   2) SIGINT    3) SIGQUIT   4) SIGILL    5) SIGTRAP
 6) SIGABRT  7) SIGBUS    8) SIGFPE    9) SIGKILL   10) SIGUSR1
11) SIGSEGV  12) SIGUSR2   13) SIGPIPE   14) SIGALRM   15) SIGTERM
16) SIGSTKFLT 17) SIGCHLD   18) SIGCONT   19) SIGSTOP   20) SIGTSTP
21) SIGTTIN  22) SIGTTOU   23) SIGURG    24) SIGXCPU   25) SIGXFSZ
26) SIGVTALRM 27) SIGPROF   28) SIGWINCH  29) SIGIO     30) SIGPWR
31) SIGSYS   34) SIGRTMIN  35) SIGRTMIN+1 36) SIGRTMIN+2 37) SIGRTMIN+3
38) SIGRTMIN+4 39) SIGRTMIN+5 40) SIGRTMIN+6 41) SIGRTMIN+7 42) SIGRTMIN+8
43) SIGRTMIN+9 44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9 56) SIGRTMAX-8 57) SIGRTMAX-7
58) SIGRTMAX-6 59) SIGRTMAX-5 60) SIGRTMAX-4 61) SIGRTMAX-3 62) SIGRTMAX-2
```

## Jerarquía de Procesos

Relación entre procesos:

- Árbol de procesos
  - Comando pstree
  - Proceso Sched
  - Proceso init o systemd
- Llamadas a sistema:
  - “getpid()”
  - “getppid()”

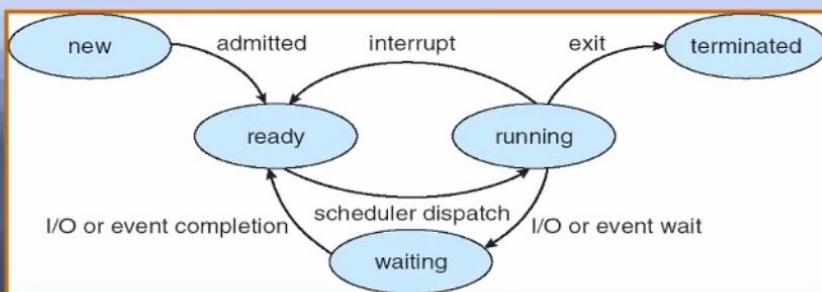


Todos los procesos tienen jerarquía, hay un árbol invertido de procesos, donde cada proceso puede tener uno o más hijos, cada hijo tiene un solo parente.

La parte usuario es de INIT hacia abajo, los otros 3 que se ven son del sistema. El primero de los procesos (INIT) es el planificador, PID=1.

PSTREE me muestra el árbol de dependencias.

## Estado de los Procesos



Diferencia entre los estados:

- ¿Cómo cambian de estado?
  - Scheduler .....que??
  - Interrupciones
  - Usuario puede forzar cambio de estado

Con respecto a los procesos tienen además estados, cuando un proceso arranca es nuevo en una lista, procesos listos para ejecutarse y se quedan ahí esperando, cuando se están ejecutando significa que están usando la CPU efectivamente, cuando un proceso está corriendo y hace entrada salida se queda bloqueado y se queda en WAITING hasta nuevo aviso, cuando llega esa entrada salida vuelve a estar listo, queda en READY para que sea elegido en algún momento por el sistema operativo, el planificador, y lo ponga a ejecutarse.

Suponiendo que tengo un programa que prácticamente no hace entrada salida, que se lo pasa en un bucle de cálculos numéricos entonces va a estar mucho tiempo en RUNNING, entonces para que sea equitativo entre todos los procesos se le asigna una cantidad de tiempo y cuando llega esa interrupción de hardware si ese proceso aún sigue en RUNNING se para a READY, para darle espacio a otro de la lista que están listos a ejecutarse. Una vez que se está ejecutando y llega a un RETURN 0 pasa a terminado.

## Bloque de Control Procesos (PCB)

¿Cómo representa el Sistema Operativo los procesos???

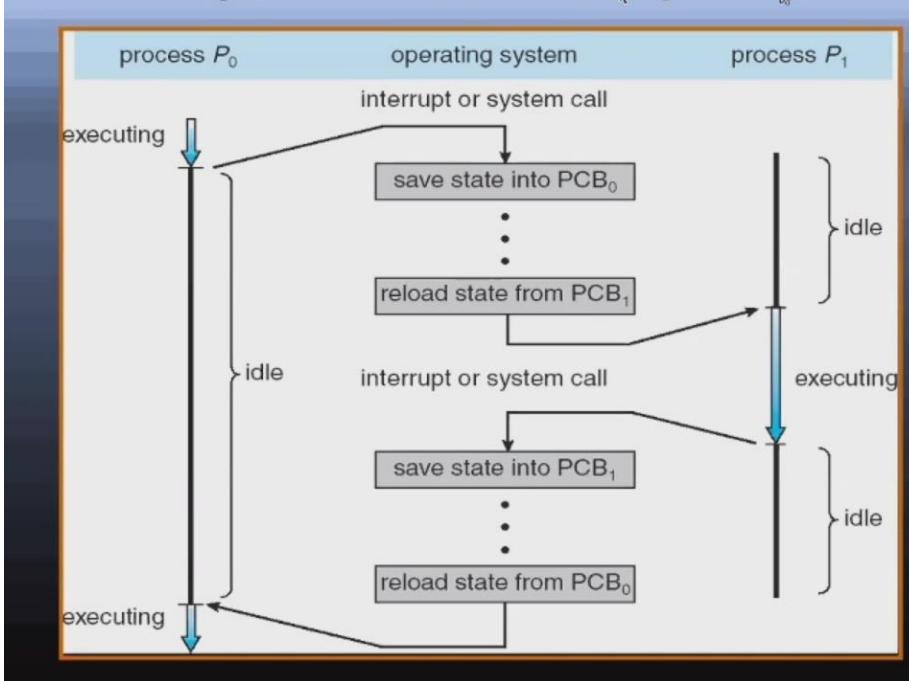
- Número de Proceso (PID)
- Estado
- PC ( contador de programa)
- Registros (generales, puntero de pila)
- Espacio de memoria que ocupa
- Descriptores de archivos abiertos
- Usuario / grupo
- etc.

process state
process number
program counter
registers
memory limits
list of open files
• • •

Como almacena el operativo los procesos. Tiene un PID.

El estado, contador del programa, registros y espacio de memoria ocupada es el TASK STATE SEGMENT TSS, en el espacio de memoria está el CODE SEGMENT, DATA SEGMENT, etc., entonces ahí está toda esa información, mas los registros, mas el contador de programa.

## Commutación entre Procesos (cambio de contexto)

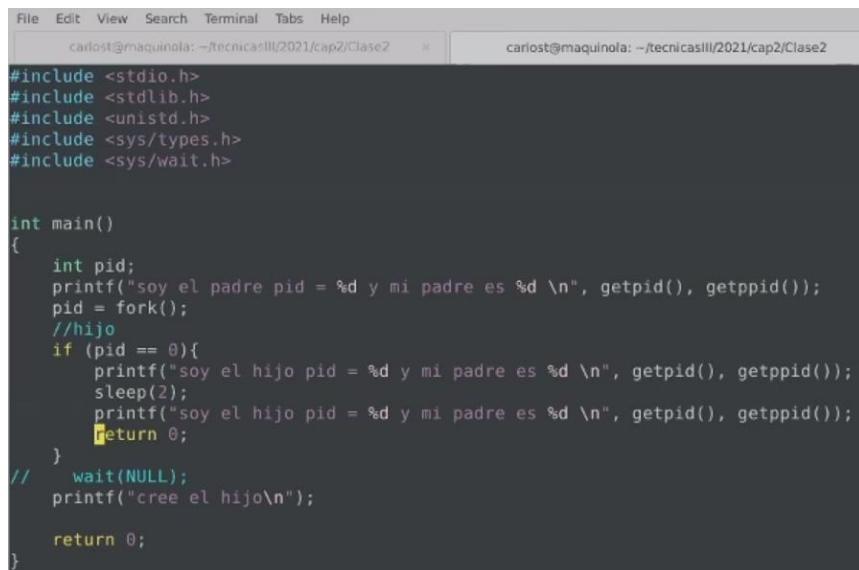


Acá muestra un cambio de contexto. Se está ejecutando un proceso, en algún momento se graba su estado en la entrada, en la tabla de procesos que se llama PCB, esos puntos sucesivos el operativo no sabe que se hace pero nosotros si, luego se recarga la tabla de estados de otro proceso que ya estaba listo para ejecutarse, se sigue

ejecutando, cuando se le acaba el tiempo o cuando hay una llamada a sistema de entrada salida por ejemplo se vuelve a grabar su estado y más adelante se sigue ejecutando el otro. Esta es la conmutación entre dos procesos.

## Procesos huérfanos o procesos Zombis.

Si un proceso crea un hijo, y termina el proceso padre antes de que termine el proceso hijo, en la tabla de procesos para este hijo guardo su PID y guardo el PID del padre, pero cuando ese PID del padre terminó, no puedo poner que su PID era el numero que tenia el padre, queda huérfano, entonces lo que tiene que hacer es asociarlo a algún otro proceso que en teoría sigue ejecutándose. Normalmente lo que pasa es que se asocian al proceso 1.

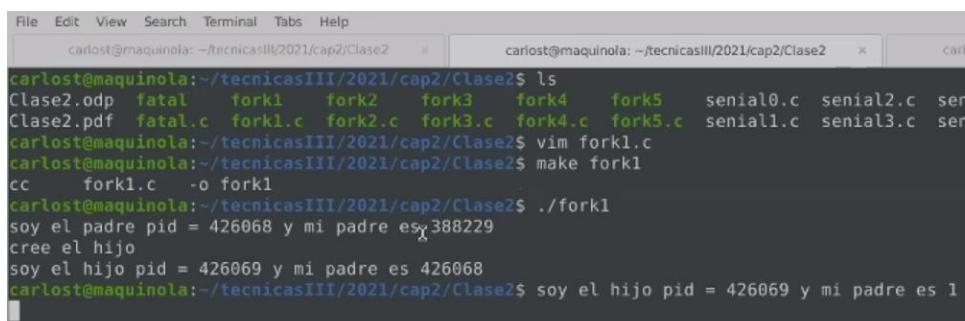


```
File Edit View Search Terminal Tabs Help
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2
```

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

int main()
{
    int pid;
    printf("soy el padre pid = %d y mi padre es %d \n", getpid(), getppid());
    pid = fork();
    //hijo
    if (pid == 0){
        printf("soy el hijo pid = %d y mi padre es %d \n", getpid(), getppid());
        sleep(2);
        printf("soy el hijo pid = %d y mi padre es %d \n", getpid(), getppid());
        return 0;
    }
    // wait(NULL);
    printf("cree el hijo\n");
    return 0;
}
```

Voy a hacer que el padre no lo espere, le voy a dar una demora al hijo.



```
File Edit View Search Terminal Tabs Help
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2
```

```
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$ ls
Clase2.odp fatal fork1 fork2 fork3 fork4 fork5 serial0.c serial2.c ser
Clase2.pdf fatal.c fork1.c fork2.c fork3.c fork4.c fork5.c serial1.c serial3.c ser
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$ vim fork1.c
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$ make fork1
cc fork1.c -o fork1
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$ ./fork1
soy el padre pid = 426068 y mi padre es 388229
cree el hijo
soy el hijo pid = 426069 y mi padre es 426068
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$ soy el hijo pid = 426069 y mi padre es 1
```

El primer PID que muestra es el del padre, el PID del hijo es correlativo al del padre dado que va utilizando PID que no están usados y el padre es el 426068, pero al ratito vuelve a decir lo mismo, donde se puede ver que es el mismo proceso pero el padre ahora es el 1, ahí podemos ver que ha muerto el proceso padre, ahora es el proceso INIT es el que lo adopta de alguna manera.

Procesos Zombis son procesos que terminan, se liberan todos los recursos, no existe en la memoria principal, solo queda en la tabla de procesos la entrada a ese proceso PCB, porque su padre tiene que ir a leer el EXIT STATUS dado que el padre aún está vivo, no sabemos en qué momento lo irá a leer, el proceso hijo ya terminó, no está más, solo queda una entrada en la tabla de procesos.

Si nosotros programamos mal, podemos llegar al caso de tener muchos hijos zombis y en algún momento se llene la tabla de procesos y al querer crear un nuevo proceso la tabla llegue a un límite y no se pueda.

```

File Edit View Search Terminal Tabs Help
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2 carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>

int main()
{
    int pid;
    printf("soy el padre pid = %d y mi padre es %d \n", getpid(), getppid());
    pid = fork();
    //hijo
    if (pid == 0){
        printf("soy el hijo pid = %d y mi padre es %d \n", getpid(), getppid());
        return 0;
    }
    // wait(NULL);
    sleep(10);
    printf("cree el hijo\n");
    return 0;
}

```

Hago que el hijo termine antes, y le doy una demora al padre para poder ver el proceso zombi.

```

File Edit View Search Terminal Tabs Help
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2 carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2 car...
top - 22:49:26 up 10 days, 13:23, 0 users, load average: 1.80, 1.98, 2.00
Tasks: 293 total, 1 running, 291 sleeping, 0 stopped, 1 zombie
%Cpu(s): 13.6 us, 10.8 sy, 0.0 ni, 74.7 id, 0.0 wa, 0.0 hi, 0.9 si, 0.0 st
MiB Mem : 15743.8 total, 916.7 free, 9659.6 used, 5167.6 buff/cache
MiB Swap: 17408.0 total, 17373.5 free, 34.5 used, 5643.8 avail Mem

      PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM TIME+ COMMAND
 384097 carlost    20   0  6456152   1.1g 137300 S 105.3  7.4 312:57.28 zoom
 382772 root      20   0  816824 174548 123816 S  54.5  1.1 139:27.15 Xorg
 246012 carlost    9 -11 1215552 19752 15296 S 16.6   0.1 65:38.19 pulseaudio
 402226 carlost    20   0 3229736 451184 151960 S   8.6  2.8 17:15.89 Web Content
 402411 carlost    20   0 3121620 477256 166700 S   5.0  2.0 51:49.72 Web Content

```

Si hago un PS –FEE me lista todos los procesos

```

root      425904    2  0 22:42 ?          00:00:00 [kworker/2:0H-kblockd]
carlost   426034 1444  0 22:46 ?          00:00:00 /usr/lib/x86_64-linux-gnu/tumbler-1/tumblerd
carlost   426049    1  0 22:46 ?          00:00:00 /usr/lib/x86_64-linux-gnu/tumbler-1/tumblerd
root      426099    2  0 22:48 ?          00:00:00 [kworker/u16:0-events_unbound]
root      426129    2  0 22:48 ?          00:00:00 [kworker/2:1H]
carlost   426163 388229  0 22:49 pts/4    00:00:00 ./fork1
carlost   426164 426163  0 22:49 pts/4    00:00:00 [fork1] <defunct>
carlost   426165 424678  0 22:49 pts/0    00:00:00 ps -fea
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$ 

```

Puedo ver que me dice que es un difunto, el hijo terminó y el padre aun no lee el EXIT STATUS. Luego de un tiempo voy a ver que no está más, cuando el padre termina, su padre es el BASH, y si ese lee el EXIT STATUS también lo saca de la tabla de procesos.

## Señales

¿Qué son ?

- Una señal es una notificación asíncrona entregada a un proceso a partir de la ocurrencia de un evento
- Para que se usan ?
  - Aviso del kernel de una excepción de Hardware
    - División por cero, Violación de Segmento, etc.
  - Notificar al proceso de un evento de software
    - Un hijo termina la ejecución, un contador llega a cero, etc.
  - Para controlarlo desde el teclado
    - Suspender el proceso, terminar el proceso.

Son notificaciones asíncronas de software, parecidas a una interrupción de hardware, se usan para avisarle desde el sistema operativo algún evento, por ejemplo cuando divide por cero, le llega al proceso una señal que el comportamiento de la rutina de atención de esas señales que termine, porque es un error grave.

También podría usarlo para mandar estados entre distintos procesos, por ejemplo, si tenemos un proceso que trabaja cooperativamente con otro y tiene que leer un resultado generado por el otro proceso, el primero de los procesos tiene que avisarle cuando tiene el resultado, entonces una alternativa es enviarle una señal cuando este el resultado, entonces en una rutina de atención de señal hago que lea el resultado.

```
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$ ./fork1
soy el padre pid = 426233 y mi padre es 388229
soy el hijo pid = 426234 y mi padre es 426233
^C
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$ echo $?
130
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$
```

Si ejecuto un proceso y presiono CTRL C, que es una interrupción de hardware desde el teclado, pero cuando la atiende el manejador manda una señal al proceso que está ejecutándose. Lo que hace es terminar ese proceso. Si me fijo como terminó, veo que fue con 130.

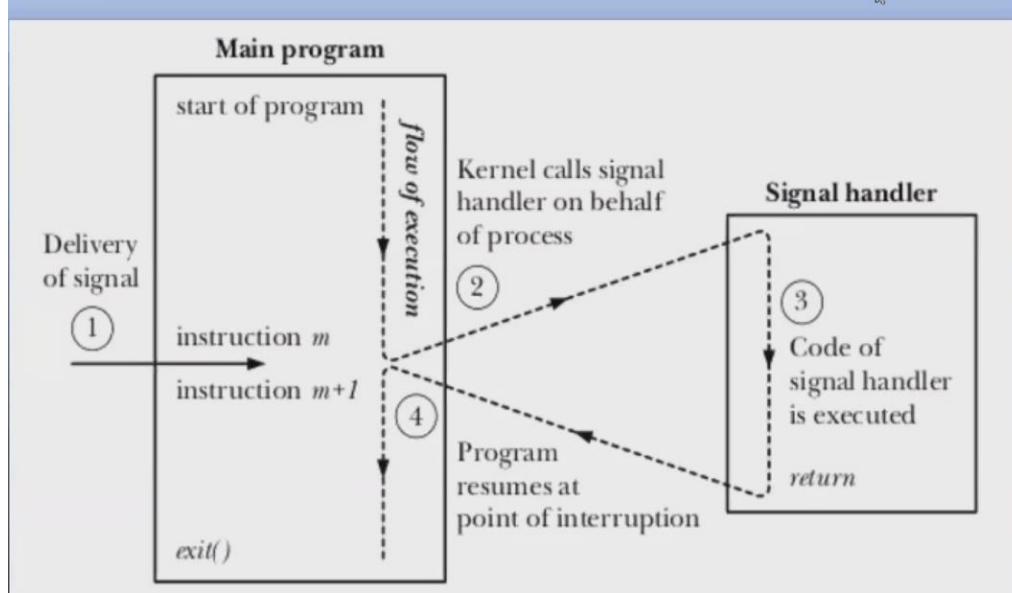
Este CTRL C lo que hace es decirle al KERNEL que le mande una señal de SIGINT al proceso que está en primer plano asociado a ese teclado.

## Manejador de la Señal

- Cuando un proceso recibe una señal este puede:
  - Ignorar la señal.
  - Dejar que se ejecute una acción por defecto relacionada con la señal.
  - Ejecuta un manejador para la señal: programa encargado de “hacer algo” con la señal recibida.

Cuando un proceso recibe una señal podría ignorarla, podría dejar que se ejecute la acción por defecto relacionada con esa señal, o ejecutar un manejador para la señal, para que haga alguna otra cosa.

# Manejador de la Señal



Cuando ejecutamos un manejador para una señal, no sabemos en qué momento del flujo de nuestro programa va a llegar, es por eso que es asíncrono, cuando llega la señal va a ejecutar la rutina del manejador o la rutina de esa señal y una vez que retorna sigue mi programa, es muy parecido a una interrupción, solo que no es de hardware, es de software. Esto es si pongo la opción de poner un manejador.

Si no el manejo por defecto podría ser que si tengo archivos abiertos los cierra, esa podría ser una acción por defecto.

Hay una biblioteca que se llama SIGNAL, que es la que se encarga de hacer algo si yo no quiero que no se haga el DEFAULT.

```
File Edit View Search Terminal Tabs Help
carlost@maquinola: ~/tecnicasIII/2021/cap2/Clase2  carlost@maquinola: ~/tecnicasIII/2021/cap2/Clase2  cā
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <signal.h>

int main (){
    signal(SIGINT, SIG_IGN);
    //luego probar con SIGKILL
    // signal(SIGKILL, SIG_IGN);

    printf("soy el proceso %d\n",getpid());
    printf("estoy trabajando en algo .... para terminar oprima cualquier tecla\n\n");
    int a = getchar();

    printf ("terminando correctamente .... ver exit_status con `echo $?\n`");
    return 0;
}
```

Este programa muestra un proceso y muestra su PID para poder mandarle desde otro proceso la señal y luego se queda esperando entrada salida desde el teclado. El programa no hace nada.

Lo único que hago adelante, apenas arranca, indicarle que si me llega la señal SIG\_ING ignórala.

```
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$ make senial1
cc      senial1.c -o senial1
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$ ./senial1
soy el proceso 426504
estoy trabajando en algo .... para terminar oprima cualquier tecla
```

Voy a enviarle la SIGIN que sería la -2 desde otra terminal

```
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$ kill -2 426504  
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$ kill -2 426504  
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$ kill -2 426504  
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$
```

Y mi proceso la va a ignorar.

```
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$ make seniall
cc      seniall.c   -o seniall
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$ ./seniall
soy el proceso 426504
estoy trabajando en algo .... para terminar oprima cualquier tecla

^C^C^C^C^C
```

Tampoco la puedo terminar con CTRL C porque es la misma, también la ignora.

Ahora si le envío KILL -9 al PID si lo termina, esta es la más violenta.

```
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$ make serial1
cc      serial1.c -o serial1
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$ ./serial1
soy el proceso 426504
estoy trabajando en algo .... para terminar oprima cualquier tecla

^C^C^C^C^C^C^C^CKilled
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$
```

Termina lo que esté haciendo. De esta manera pude ignorar la señal SIGIN.

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <signal.h>

int main (){
    signal(SIGINT, SIG_IGN);
    //luego probar con SIGKILL
    signal(SIGKILL, SIG_IGN);

    printf("soy el proceso %d\n",getpid());
    printf("estoy trabajando en algo .... para terminar oprima cualquier tecla\n\n");

    int a = getchar();

    printf ("terminando correctamente .... ver exit_status con \"echo $?\n\"");
    return 0;
}
```

La señal 9 es SIGKILL y ahora lo habilito en el código para ignorarla también.

```
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$ make seniall  
cc      seniall.c -o seniall  
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$ ./seniall  
soy el proceso 426556  
estoy trabajando en algo .... para terminar oprima cualquier tecla  
^C^C^C^C^C^C
```

Como podemos ver CTRL C no funciona.

```
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$ kill -2 426556  
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$ kill -9 426556  
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$
```

Le envío KILL -9 al PID

```
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$ make senial1
cc      senial1.c -o senial1
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$ ./senial1
soy el proceso 426556
estoy trabajando en algo .... para terminar oprima cualquier tecla
^C^C^C^C^C^C^C^C^Killed
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$
```

Y se puede ver que lo mató. Esta señal no puede ser ignorada. Hay dos señales que no se pueden ignorar.

SIG STOP y SIG KILL.

```
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$ ./senial1
soy el proceso 426600
estoy trabajando en algo .... para terminar oprima cualquier tecla
^Z
[1]+  Stopped                  ./senial1
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$ fg
./senial1
```

SIG STOP es cuando estoy haciendo algo y hago CTRL Z, lo que hace es parar el proceso, queda bloqueado hasta nuevo aviso, lo puedo arrancar nuevamente si le doy FG. Pero si quisiera poner un manejador para CTRL Z o para CTRL C no puedo, me deja ponerlo pero no hace nada.

```
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$ kill -l
 1) SIGHUP      2) SIGINT      3) SIGQUIT      4) SIGILL      5) SIGTRAP
 6) SIGABRT     7) SIGBUS      8) SIGFPE       9) SIGKILL     10) SIGUSR1
11) SIGSEGV     12) SIGUSR2     13) SIGPIPE     14) SIGALRM     15) SIGTERM
16) SIGSTKFLT   17) SIGCHLD     18) SIGCONT     19) SIGSTOP     20) SIGTSTP
21) SIGTTIN     22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO       30) SIGPWR
31) SIGSYS      34) SIGRTMIN    35) SIGRTMIN+1  36) SIGRTMIN+2  37) SIGRTMIN+3
38) SIGRTMIN+4  39) SIGRTMIN+5  40) SIGRTMIN+6  41) SIGRTMIN+7  42) SIGRTMIN+8
43) SIGRTMIN+9  44) SIGRTMIN+10 45) SIGRTMIN+11 46) SIGRTMIN+12 47) SIGRTMIN+13
48) SIGRTMIN+14 49) SIGRTMIN+15 50) SIGRTMAX-14 51) SIGRTMAX-13 52) SIGRTMAX-12
53) SIGRTMAX-11 54) SIGRTMAX-10 55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7
58) SIGRTMAX-6  59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$
```

Para ver todas las que existen ponemos KILL -L.

Ahora vamos a tratar de ejecutar un manejador.

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <signal.h>

void manejador(){
    printf("yo pienso terminar\n");
    //signal(SIGINT,SIG_DFL);

}

int main (){
    signal(SIGINT, manejador);

    printf("soy el proceso %d\n",getpid());

    printf("estoy trabajando en algo .... para terminar oprima cualquier tecla\n\n");

    int a = getchar();

    printf ("terminando correctamente .... ver exit_status con \"echo $?\"\n");

    return 0;
}
```

El programa es básicamente el mismo, muestra el PID y se queda ahí tonteando pero en vez de decirle que la ignore pongo el nombre de una función, de una rutina. Con esa rutina hago que haga algo, en este caso larga un texto que dice “no pienso terminar”.

```
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$ ./senial2
soy el proceso 426885
estoy trabajando en algo .... para terminar oprima cualquier tecla

^Cno pienso terminar
^Cno pienso terminar
^Cno pienso terminar
no pienso terminar
no pienso terminar
no pienso terminar
^Cno pienso terminar
^Cno pienso terminar
^Cno pienso terminar
^Z
[1]+ Stopped ./senial2
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$ ^C
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$ ^C
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$ ^C
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$ fg
./senial2
^Cno pienso terminar
■
```

Lo mismo pasaría si le doy la señal desde otra consola.

```
^Z
[1]+ Stopped ./senial2
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$ kill -9 427024
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$ 
[1]+ Killed ./senial2
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$ 
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$ ./senial2
soy el proceso 427087
estoy trabajando en algo .... para terminar oprima cualquier tecla

^Z
[1]+ Stopped ./senial2
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$ ./senial2
soy el proceso 427088
estoy trabajando en algo .... para terminar oprima cualquier tecla

^Z
[2]+ Stopped ./senial2
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$ ./senial2
soy el proceso 427091
estoy trabajando en algo .... para terminar oprima cualquier tecla

^Z
[3]+ Stopped ./senial2
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$ jobs
[1] Stopped ./senial2
[2]- Stopped ./senial2
[3]+ Stopped ./senial2
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase2$ fg 2
./senial2
no pienso terminar
```

Puedo pausar procesos, volver a ejecutarlos y se van generando procesos que son la misma receta, luego con JOBS puedo ver los procesos que están en segundo plano, y con FG y el número del proceso volver a retomarlo.

## IPC – INTER-PROCESS COMMUNICATION

## ¿Qué es?

- Función básica de un OS.
- Permite enviar información entre procesos (datos o estado)
- Diferentes técnicas

Es una función básica que nos dan los sistemas operativos, me permite enviar información, dato o estado, entre dos procesos distintos. Hemos visto señales que en definitiva termina siendo un mecanismo de IPC muy sencillo que me da el estado de otro proceso o porque parte va, cuando un proceso quería avisarle algo a otro le enviaba una señal y ese otro en función de esa señal podía hacer algo o no. Entonces si quiero que interactúen esos procesos puedo usar semáforos, no van a compartir datos, no es que le mande un resultado de una operación pero podría con señales decirle que lea un archivo dado que guarde el resultado ahí.



Tenemos un espacio KERNEL, espacio usuario y los procesos funcionando en este espacio usuario. Para evitar problemas entre los procesos había creado un espacio de direcciones que era virtual y cada proceso está como enjaulado en su propio espacio, entonces sí quiero que un proceso le envíe algo a otro proceso es imposible porque los datos no están en mi espacio de direcciones, está completamente prohibido, no tengo forma de cruzar esa barrera.

Quiero que comparten información para que sean una tarea cooperativa.

## Necesidades

### Procesos independientes o cooperativos

- Compartir información (db)
- Acelerar los cálculos (más en multicore)
- Modularidad (1 función por proceso)
- Conveniencia (1 usuario, varias tareas al mismo tiempo)

Puedo tener procesos independientes o cooperativos. En los procesos independientes donde no tiene que nada que ver uno con otro entonces en principio no necesitan intercomunicarse dado que no les interesa. Pero podría pasar que tengo una tarea para hacer y la separo en distintos procesos y entre ellos deberían cooperar, de alguna manera deberían interactuar, recibir alguna información uno del otro.

Casos comunes:

**Cuando quiero compartir alguna información**, por ejemplo hay varios procesos que quieren acceder a los datos de una base de datos, son procesos distintos, ni siquiera están relacionados, porque uno puede ser del usuario Juan, el

otro del usuario Pepe, no son ni padres ni hijos ni nada de parientes, y sin embargo quieren acceder a los datos que maneja otro proceso que sería el motor de la base de datos.

**Para acelerar los cálculos**, si tengo una tarea compleja y la puedo separar en tareas más sencillas, si esto lo escribo en un solo programa monolítico y se corre como un solo proceso, cada vez que haga entrada-salida o se bloquee esperando algo ese proceso se va a demorar, si la separo en varias funciones y la implemento como un proceso distinto, si hay un proceso o una función que se bloquea por algo se pueden seguir ejecutando otras, entonces de esta manera podre ir avanzando, y en algún momento van a tener que intercambiar datos. Si tengo procesadores con muchos núcleos mejor aún, si tengo un solo núcleo se ejecutarán en ese, pero si tengo varios se irán ejecutando en paralelo.

**Modularidad**, por ejemplo tengo un proceso que hace una parte de la tarea, otro proceso que hace otra parte, y con algún mecanismo intercambian información, si soy el programador y desarrollo este proceso, lo escribo, solo debo preocuparme de como voy a compartir la información, en cada programa podría mejorar su código, cambiar de lenguaje, sustituir por otro siempre que respete lo que se va a intercambiar, esto me da modularidad y puedo mejorar el código.

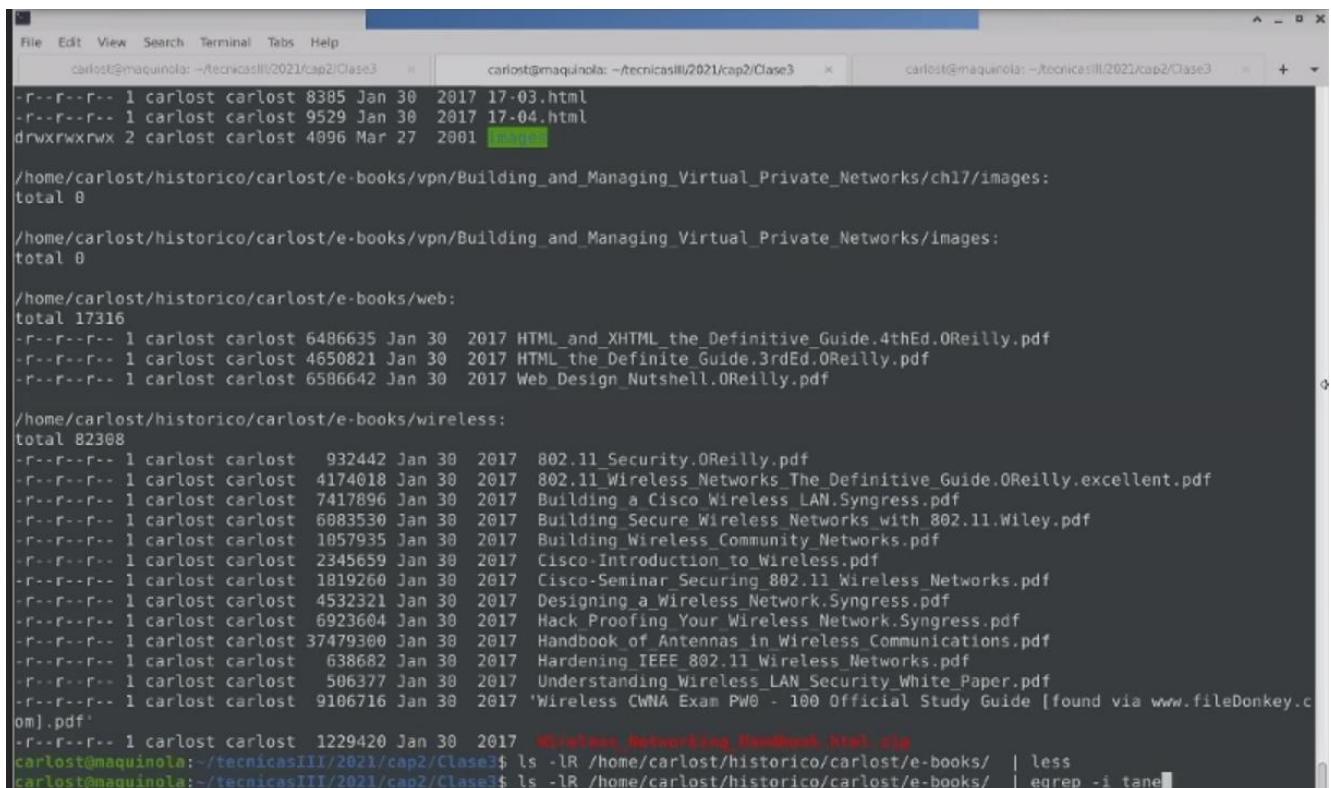
**Conveniencia**, podría pasar que tenga varios procesos por separado para ejecutar y los quiero ejecutar a la vez para que intercambien datos, supongamos que tengo un directorio con un montón de libros y quiero buscar todos los libros que son de un autor y quiero ver de esos cuales son los que más tamaño tienen. Tendría que abrir un navegador, ir directorio por directorio viendo cual es el tamaño de los libros. Podría utilizar varios procesos que se comuniquen entre ellos y me resuelvan ese problema.

LS -LR me da un listado de todo un directorio a partir del lugar donde le digo.



```
carlost@maquinola: ~/tecnicasIII/2021/cap2/Clase3
carlost@maquinola: ~/tecnicasIII/2021/cap2/Clase3
carlost@maquinola: ~/tecnicasIII/2021/cap2/Clase3$ ls -LR /home/carlost/historico/carlost/e-books/
```

El palito hace la comunicación entre procesos (luego de darle ENTER le muestra una lista enorme de libros).



```
File Edit View Search Terminal Tabs Help
carlost@maquinola: ~/tecnicasIII/2021/cap2/Clase3
carlost@maquinola: ~/tecnicasIII/2021/cap2/Clase3
carlost@maquinola: ~/tecnicasIII/2021/cap2/Clase3$ ls -LR /home/carlost/historico/carlost/e-books/
r--r--r-- 1 carlost carlost 8385 Jan 30 2017 17-03.html
r--r--r-- 1 carlost carlost 9529 Jan 30 2017 17-04.html
drwxrwxrwx 2 carlost carlost 4096 Mar 27 2001 vpn
/home/carlost/historico/carlost/e-books/vpn/Building_and_Managing_Virtual_Private_Networks/ch17/images:
total 0
/home/carlost/historico/carlost/e-books/vpn/Building_and_Managing_Virtual_Private_Networks/images:
total 0
/home/carlost/historico/carlost/e-books/web:
total 17316
r--r--r-- 1 carlost carlost 6486635 Jan 30 2017 HTML_and_XHTML_the_Definitive_Guide.4thEd.OReilly.pdf
r--r--r-- 1 carlost carlost 4650821 Jan 30 2017 HTML_the_Definite_Guide.3rdEd.OReilly.pdf
r--r--r-- 1 carlost carlost 6586642 Jan 30 2017 Web_Design_Nutshell.OReilly.pdf
/home/carlost/historico/carlost/e-books/wireless:
total 82308
r--r--r-- 1 carlost carlost 932442 Jan 30 2017 802.11_Security.OReilly.pdf
r--r--r-- 1 carlost carlost 4174018 Jan 30 2017 802.11_Wireless_Networks_The_Definitive_Guide.OReilly.excellent.pdf
r--r--r-- 1 carlost carlost 7417896 Jan 30 2017 Building_a_Cisco_Wireless_LAN.Syngress.pdf
r--r--r-- 1 carlost carlost 6083530 Jan 30 2017 Building_Secure_Wireless_Networks_with_802.11.Wiley.pdf
r--r--r-- 1 carlost carlost 1057935 Jan 30 2017 Building_Wireless_Community_Networks.pdf
r--r--r-- 1 carlost carlost 2345659 Jan 30 2017 Cisco-Introduction_to_Wireless.pdf
r--r--r-- 1 carlost carlost 1819260 Jan 30 2017 Cisco-Seminar_Securing_802.11_Wireless_Networks.pdf
r--r--r-- 1 carlost carlost 4532321 Jan 30 2017 Designing_a_Wireless_Network.Syngress.pdf
r--r--r-- 1 carlost carlost 6923604 Jan 30 2017 Hack_Proofing_Your_Wireless_Network.Syngress.pdf
r--r--r-- 1 carlost carlost 37479300 Jan 30 2017 Handbook_of_Antennas_in_Wireless_Communications.pdf
r--r--r-- 1 carlost carlost 638682 Jan 30 2017 Hardening_IEEE_802.11_Wireless_Networks.pdf
r--r--r-- 1 carlost carlost 506377 Jan 30 2017 Understanding_Wireless_LAN_Security_White_Paper.pdf
r--r--r--r-- 1 carlost carlost 9106716 Jan 30 2017 'Wireless CWNA Exam PW0 - 100 Official Study Guide [found via www.fileDonkey.com].pdf'
r--r--r--r-- 1 carlost carlost 1229420 Jan 30 2017 Wireless_Networking_Handbook.html.xls
carlost@maquinola: ~/tecnicasIII/2021/cap2/Clase3$ ls -LR /home/carlost/historico/carlost/e-books/ | less
carlost@maquinola: ~/tecnicasIII/2021/cap2/Clase3$ ls -LR /home/carlost/historico/carlost/e-books/ | egrep -i tane
```

LESS hace que los muestre lento, y con EGREP -i TANE lista todo lo que diga TANE, para buscar los de TANENBAUM.

```

File Edit View Search Terminal Tabs Help
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase3$ carlost@maquinola: ~/tecnicasIII/2021/cap2/Clase3$ carlost@maquinola: ~/tecnicasIII/2021/cap2/Clase3$ +
/home/carlost/historico/carlost/e-books/wireless:
total 82308
-r--r--r-- 1 carlost carlost 932442 Jan 30 2017 802.11_Security.OReilly.pdf
-r--r--r-- 1 carlost carlost 4174018 Jan 30 2017 802.11_Wireless_Networks_The_Definitive_Guide.OReilly.excellent.pdf
-r--r--r-- 1 carlost carlost 7417896 Jan 30 2017 Building_a_Cisco_Wireless_LAN.Syngress.pdf
-r--r--r-- 1 carlost carlost 6083530 Jan 30 2017 Building_Secure_Wireless_Networks_with_802.11.Wiley.pdf
-r--r--r-- 1 carlost carlost 1057935 Jan 30 2017 Building_Wireless_Community_Networks.pdf
-r--r--r-- 1 carlost carlost 2345659 Jan 30 2017 Cisco-Introduction_to_Wireless.pdf
-r--r--r-- 1 carlost carlost 1819260 Jan 30 2017 Cisco-Seminar_Securing_802.11_Wireless_Networks.pdf
-r--r--r-- 1 carlost carlost 4532321 Jan 30 2017 Designing_a_Wireless_Network.Syngress.pdf
-r--r--r-- 1 carlost carlost 6923604 Jan 30 2017 Hack_Proofing_Your_Wireless_Network.Syngress.pdf
-r--r--r-- 1 carlost carlost 37479300 Jan 30 2017 Handbook_of_Antennas_in_Wireless_Communications.pdf
-r--r--r-- 1 carlost carlost 638682 Jan 30 2017 Hardening_IEEE_802.11_Wireless_Networks.pdf
-r--r--r-- 1 carlost carlost 506377 Jan 30 2017 Understanding_Wireless_LAN_Security_White_Paper.pdf
-r--r--r-- 1 carlost carlost 9106716 Jan 30 2017 'Wireless CWNA Exam PW0 - 100 Official Study Guide [found via www.fileDonkey.com].pdf'
-r--r--r-- 1 carlost carlost 1229420 Jan 30 2017 Wireless_Networking_Handbook.html.asp
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase3$ ls -lr /home/carlost/historico/carlost/e-books/ | less
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase3$ ls -lr /home/carlost/historico/carlost/e-books/ | egrep -i tane
-rwrxr-xr-x 1 carlost carlost 128523669 Jan 30 2017 Distributed_systems_Tanenbaum.pdf
-rwrxr-xr-x 1 carlost carlost 26313821 Jan 30 2017 Sistemas_Operativos_Distribuidos_TANENBAUM.pdf
-rw-r--r-- 1 carlost carlost 8967998 Jan 30 2017 Computer_Networks_4th_Ed - Andrew_S._Tanenbaum.chm
-rw-r--r-- 1 carlost carlost 8454231 Jan 30 2017 Computer_Networks - A_Tanenbaum - 5th_edition.pdf
-r-xr-xr-x 1 carlost carlost 299235 Jan 30 2017 Computer_Networks_Problem_Solutions_(4th_Ed_2003) - A_Tanen.pdf
-rwrxr-xr-x 1 carlost carlost 14097349 Jan 30 2017 Redes_de_Computadoras-Tanenbaum-4ed.pdf
-rw-rw-r-- 1 carlost carlost 22163785 Sep 21 2020 redes_de_computadoras-Tanenbaum-5ta.pdf
-r-xr-xr-x 1 carlost carlost 216338 Jan 30 2017 Resumen_del_librer Redes_de_computadoras_(Tanenbaum).pdf
-r-xr-xr-x 1 carlost carlost 86914 Jan 30 2017 Tanenbaum_Wireless_Lan.pdf
-r-xr-xr-x 1 carlost carlost 89897 Jan 30 2017 TRANSMISION_DATOS_CAP3_TANENBAUM_CN3-3.pdf
-r-xr-xr-x 1 carlost carlost 6319385 Jan 30 2017 TANENBAUM, Andrew - Sistemas_Operativos. Disen_o_e_Implementacion.pdf
-r-xr-xr-x 1 carlost carlost 40702229 Jan 30 2017 Tanenbaum, Andrew_S. - Operating_Systems_Design_and_Impleme.pdf
-r--r--r-- 1 carlost carlost 34276144 Jan 30 2017 Tanenbaum-MOS-3e.pdf
-rw-r--r-- 1 carlost carlost 239928 Jan 30 2017 Tanenbaum-MOS-3e-Responses.pdf
-rw-rw-r-- 1 carlost carlost 4564416 Apr 13 2020 tanenbaum-sistemas-operativos-modernos-23.pdf
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase3$
```

```

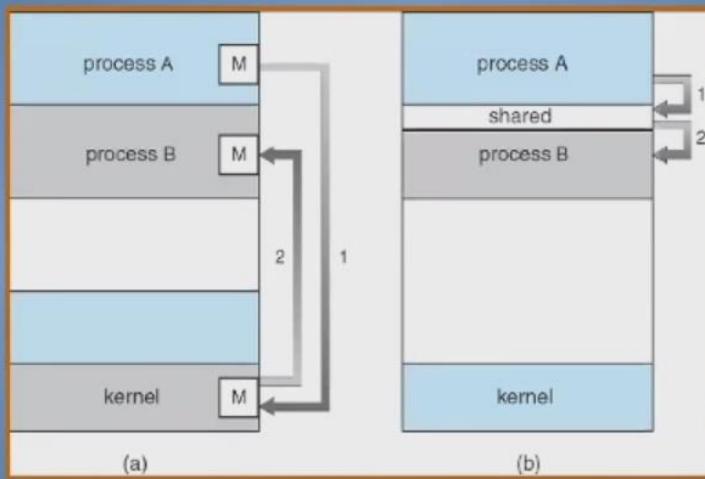
File Edit View Search Terminal Tabs Help
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase3$ carlost@maquinola: ~/tecnicasIII/2021/cap2/Clase3$ carlost@maquinola: ~/tecnicasIII/2021/cap2/Clase3$ +
299235 Jan 30 2017 Computer_Networks_Problem_Solutions_(4th_Ed_2003) - A_Tanen.pdf
4564416 Apr 13 2020 tanenbaum-sistemas-operativos-modernos-23.pdf
6319385 Jan 30 2017 TANENBAUM, Andrew - Sistemas_Operativos. Disen_o_e_Implementacion.pdf
8454231 Jan 30 2017 Computer_Networks - A_Tanenbaum - 5th_edition.pdf
8967998 Jan 30 2017 Computer_Networks_4th_Ed - Andrew_S._Tanenbaum.chm
14097349 Jan 30 2017 Redes_de_Computadoras-Tanenbaum-4ed.pdf
22163785 Sep 21 2020 redes_de_computadoras-tanenbaum-5ta.pdf
26313821 Jan 30 2017 Sistemas_Operativos_Distribuidos_TANENBAUM.pdf
34276144 Jan 30 2017 Tanenbaum-MOS-3e.pdf
40702229 Jan 30 2017 Tanenbaum, Andrew_S. - Operating_Systems_Design_and_Impleme.pdf
128523669 Jan 30 2017 Distributed_systems_Tanenbaum.pdf
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase3$ ls -lr /home/carlost/historico/carlost/e-books/ | egrep -i tane | cut -c 29-28
0 | sort -nr
128523669 Jan 30 2017 Distributed_systems_Tanenbaum.pdf
40702229 Jan 30 2017 Tanenbaum, Andrew_S. - Operating_Systems_Design_and_Impleme.pdf
34276144 Jan 30 2017 Tanenbaum-MOS-3e.pdf
26313821 Jan 30 2017 Sistemas_Operativos_Distribuidos_TANENBAUM.pdf
22163785 Sep 21 2020 redes_de_computadoras-tanenbaum-5ta.pdf
14097349 Jan 30 2017 Redes_de_Computadoras-Tanenbaum-4ed.pdf
8967998 Jan 30 2017 Computer_Networks_4th_Ed - Andrew_S._Tanenbaum.chm
8454231 Jan 30 2017 Computer_Networks - A_Tanenbaum - 5th_edition.pdf
6319385 Jan 30 2017 TANENBAUM, Andrew - Sistemas_Operativos. Disen_o_e_Implementacion.pdf
4564416 Apr 13 2020 tanenbaum-sistemas-operativos-modernos-23.pdf
299235 Jan 30 2017 Computer_Networks_Problem_Solutions_(4th_Ed_2003) - A_Tanen.pdf
239928 Jan 30 2017 Tanenbaum-MOS-3e-Responses.pdf
216338 Jan 30 2017 Resumen_del_librer Redes_de_computadoras_(Tanenbaum).pdf
89897 Jan 30 2017 TRANSMISION_DATOS_CAP3_TANENBAUM_CN3-3.pdf
86914 Jan 30 2017 Tanenbaum_Wireless_Lan.pdf
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase3$ ls -lr /home/carlost/historico/carlost/e-books/ | egrep -i tane | cut -c 29-28
0 | sort -nr | head -n4
128523669 Jan 30 2017 Distributed_systems_Tanenbaum.pdf
40702229 Jan 30 2017 Tanenbaum, Andrew_S. - Operating_Systems_Design_and_Impleme.pdf
34276144 Jan 30 2017 Tanenbaum-MOS-3e.pdf
26313821 Jan 30 2017 Sistemas_Operativos_Distribuidos_TANENBAUM.pdf
I
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase3$
```

También puedo hacer que los ordene por tamaño, y luego que me muestre los 4 más grandes.

Entonces tengo varios procesos que si los hacia uno tras otro a mano me habría demorado mucho, entonces lo que hago es comunicarlos entre ellos, lo que me retorna el primer proceso, lo va enviando a la entrada del proceso siguiente.

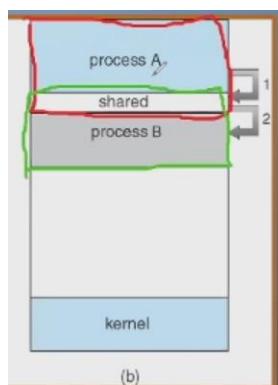
# Clasificación de distintas IPC

- Paso de Mensajes
  - Usando llamados al kernel
  - Pequeñas cantidad de información
  - Fácil de implementar
- Memoria Compartida
  - Comparte espacio de mem.
  - Rápida
  - Sincronizar



Hay distintos tipos de IPC, básicamente se clasifican en dos dependiendo de cómo se comunican. Uno que sería paso mensaje y otro que se llama memoria compartida.

Paso mensaje si o si interactuamos con el KERNEL, el proceso A le quiere pasar un mensaje al proceso B, pero como no puede hacerlo directamente, dado que cada uno tiene su espacio de direcciones y están completamente aislados, entonces le envía un mensaje al KERNEL, y luego el KERNEL se lo envía al proceso en cuestión. Interactúa el KERNEL por lo tanto es más lento, esa copia de datos en el KERNEL generalmente son de menor tamaño entonces por ahí no puedo compartir demasiada información, es bastante fácil de implementar porque tiene algo que se llama sincronización que veremos luego.

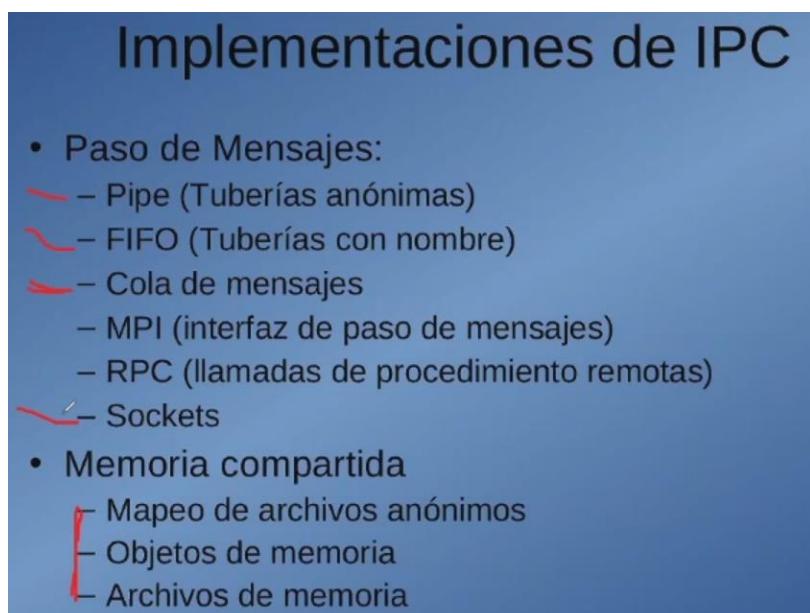


Por otro lado tenemos memoria compartida, El espacio de memoria del espacio A es A y el compartido, y el espacio de memoria del proceso B es B y el compartido. Entonces en el medio tengo un área que es accedida por los dos procesos (se han superpuesto cosa que antes habíamos dicho que no podía hacerse). Tanto el proceso A como el proceso B tienen que hacer una llamada a sistema pidiéndole esto, si el sistema operativo lo acepta estos procesos comparten un área de memoria, entonces el proceso A puede escribir un dato en esa área de memoria y el proceso B lo puede leer, la ventaja es la velocidad, es más rápido, no necesito hacer una llamada a sistema, no hago un cambio de contexto.

El problema aca es que tanto A como B tienen vida propia, entonces va a depender de cuando el planificador ejecuta A y cuando ejecuta B, por mas que diga que A escriba un resultado de una operación y en el programa B le digo que

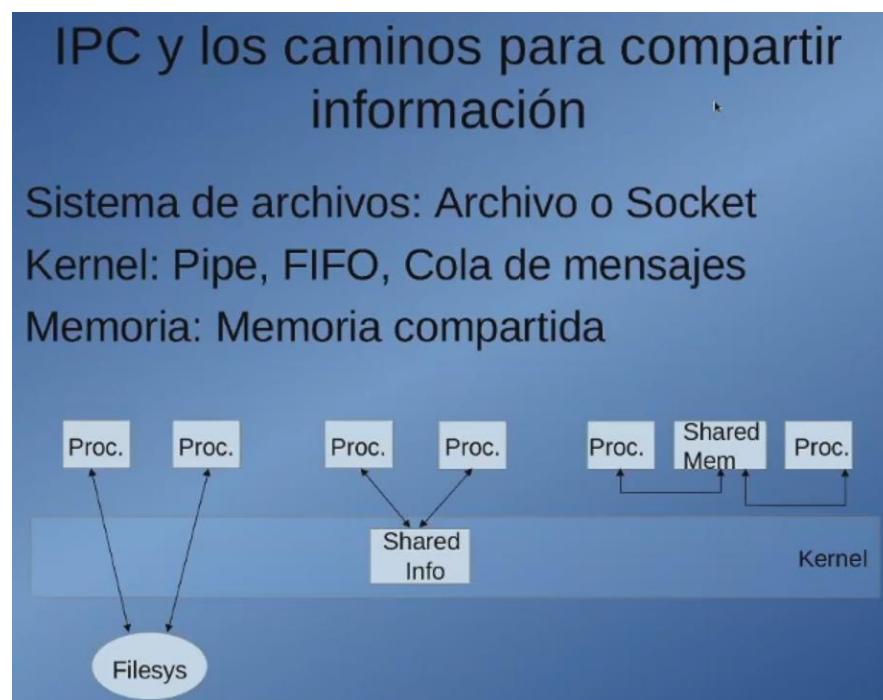
lea esa posición de memoria, no puedo asegurar que B lea después que A haya escrito, esto es sincronización, necesitaría alguna herramienta aparte para poder lograr eso.

El tamaño de esta área compartida es configurado por el programador con el tamaño que quiera.



Vamos a ver para paso de mensajes PIPE o tuberías, FIFO que son funciones muy parecidas a tuberías solo que tienen un nombre, y otras que se llaman cola de mensajes. SOCKETS la vamos a ver cuando veamos redes.

Los mecanismos de memoria compartida no los vamos a ver, cada vez se usan menos



Estos son los caminos, por donde pasa la información para compartirse.

El primer caso no uso un mecanismo de IPC si no que me baso en el sistema de archivos. Un proceso graba un dato, hace una llamada a sistema, está llamada a sistema llama a un driver de entrada-salida, se guarda en el sistema de archivos, luego el otro proceso cuando va a leer tiene que hacer otra llamada a sistema para leer. Esta es la menos indicada, es muy lenta, tiene todo el DELAY del sistema de archivos.

En el medio tenemos lo que habíamos hablado de paso de mensaje, donde la información viaja desde un proceso a un BUFFER en el KERNEL y el otro proceso lee de este BUFFER, aca también hay llamada a sistema para leer y escribir, le pido al KERNEL que se encargue de intercambiar la información.

El tercero utiliza memoria compartida para intercambiar datos.

## Persistencia

Cuanto tiempo permanece en existencia

- Proceso:
  - Pipe
  - Fifo (cuidado no es persistencia de kernel, aunque persiste cuando todos los procesos involucrados terminaron, no los datos de los mismos!)
  - Socket
  - RPC
  - MPI
  - Memoria compartida (mapeo de archivos anónimos )
- Kernel:
  - Cola de mensajes
  - Memoria compartida (objetos de memoria)
- Filesystem
  - Memoria compartida (archivos de memoria)

Cuando hablamos de la persistencia, cuanto tiempo dura la información.

Cuando un IPC decimos que tiene persistencia en proceso, es cuando se termina el proceso si no leí los datos que tenían para mí se pierden, el ejemplo de eso es PIPE, FIFO también, SOCKET lo veremos más adelante.

En KERNEL significa que los datos van a quedar almacenados siempre y cuando no reinicie la máquina y se borre esa información del KERNEL, la persistencia de ese tipo es de cola de mensajes.

De sistema de archivos es de memoria compartida, pero no vamos a ver este mecanismo.

Entonces la persistencia es cuánto tiempo va a durar la información. Solo lo que dure mi proceso, solo el tiempo que dure mi maquina prendida o en el sistema de archivos hasta que no lo borre no pierdo esa información.

## Uso

- Comunicación entre procesos de un mismo PC (los que analizaremos):
  - Pipes
  - Fifo
  - Colas de mensajes
  - Memoria compartida
- Comunicación entre procesos de una misma PC o de distintas PCs en una red :
  - Sockets
  - RPC
  - MPI

Otra clasificación en cuanto al uso, dentro de una misma PC tenemos tuberías, FIFO, cola de mensajes.

Si tengo distintas Pc, tengo SOCKETS RPC y MPI.

# Pipe – Tuberías anónimas

- Permiten comunicar información entre procesos RELACIONADOS por herencia
- Mensajes orientados a "stream" de datos ( no segmentados )
- No permiten compartir información entre procesos no relacionados
- Se pueden abrir configuradas como:
  - Bloqueantes (por defecto)
  - No bloqueantes
- Syscall `pipe(descriptores[2])`

```
int fds[2];
pipe(fds);
```



PIPE es un BUFFER en el KERNEL, donde la restricción o limitación que tiene es que es para procesos relacionados, es decir, un padre, un hijo o un nieto, o dos hijos entre sí, van a poder intercambiar información. Otro proceso que no tiene nada que ver no podría acceder. Este es uno de los mecanismos de IPC más viejos que se crearon en UNIX a principio de los 70.

Está orientado a Flujo de Bytes, es como si fuera un tubo, por un extremo del tubo escribo Bytes, y del otro lado leo Bytes. Es unidireccional, en el extremo en el que escribo no podría leer, y en el extremo que leo no podría escribir.

El primero que entra es el primero en salir.

La llamada a sistema para crear un PIPE es PIPE(FDS); tengo que pasarle un arreglo de dos elementos enteros.

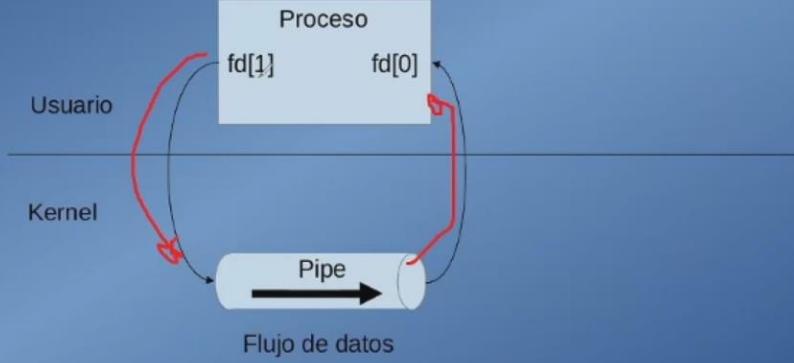
Cuando creo un proceso tengo de movida el descriptor 0 que apunta al canal del teclado, el descriptor 1 que apunta a la pantalla, y el 2 que es el de error.

Cuando creo un PIPE me va a generar dos descriptores más que apuntan a ese BUFFER en el KERNEL, uno va a ser el extremo de escritura y el otro va a ser el extremo de lectura.

Al estar en KERNEL pasan dos cosas, si quiero leer en el descriptor de lectura y no hay ningún dato el proceso se va a bloquear, también puede pasar que si comienzo a escribir y escribir y llegamos al tamaño límite del BUFFER y quiero seguir escribiendo el proceso se va a bloquear hasta que alguien lea en el otro extremo.

# Pipe – Tuberías anónimas

- Ejemplo de uso

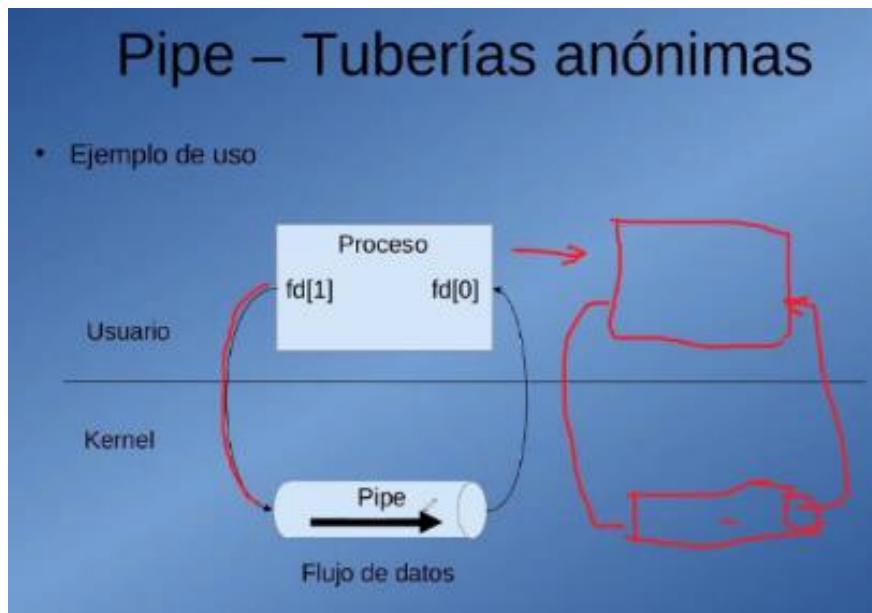


Ejemplo de uso:

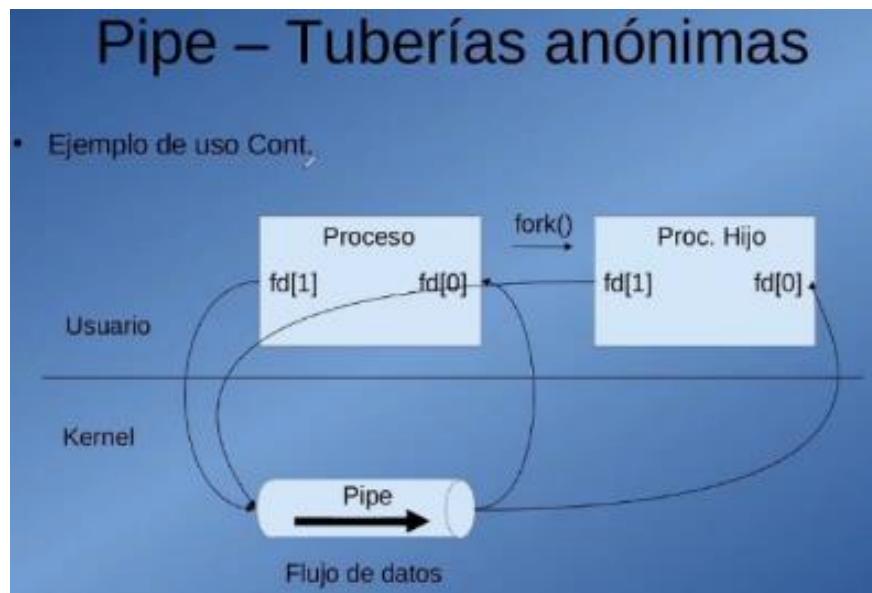
En un proceso invoco a PIPE, me creo estos dos descriptores, me creo el BUFFER en el KERNEL. Si en este proceso yo escribo y luego leo, el extremo de escritura es el 1 y el de lectura es el 0.

Si escribo HOLA MUNDO, al leer obtendré HOLA MUNDO.

VIDEO 2



Entonces sí creo un proceso hijo y luego ejecuto el mismo comando PIPE va a crear otra tubería donde asociara los nuevos extremos, pero lo que escriba en el proceso padre no va a poder ser leído en el proceso hijo, porque cuando lea va a leer de su propia tubería.



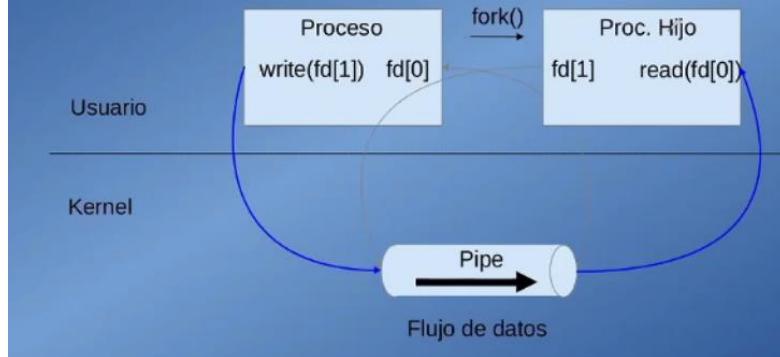
Lo que hay que hacer primero es crear la tubería y luego hacer el FORK(). Cuando hago el FORK() después de asegurarme que ya tengo la tubería, al proceso hijo se le copian todos los descriptores que tiene del padre.

Entonces si del padre le quiero escribir al hijo voy a tener que escribir en el descriptor FD[1], y leer en el hijo en el FD[0]. También podría ser que el hijo escriba en el FD[1] y el padre lea del FD[0].

Lo que no podríamos hacer acá es que el padre le escriba datos al hijo y que el hijo le responda, y esto es porque cuando el padre lee los datos no sabe si los datos que lee son los del hijo o los de él, dado que no podemos asegurar cuando se ejecuta uno o el otro.

# Pipe – Tuberías anónimas

- Ejemplo de uso Cont.



Desde otro proceso que no tenga nada que ver con estos no podría acceder a estos extremos de la tubería porque no estaría referenciado. Solo se comparten datos entre procesos relacionados, Padre, hijo, nieto.

Este es el código C que muestra la creación y uso de un pipe:

```
1 #include <stdio.h>
2 #include <unistd.h>
3 #include <sys/types.h>
4
5 int main(void) {
6     int pfd[2];
7     int nread;
8     char buf[256];
9
10    if (pipe(pfd) < 0) {
11        perror("pipe");
12        return 255;
13    }
14    if (fork() == 0) { /* hijo */
15        nread = read(0, buf, sizeof buf);
16        write(pfd[1], buf, nread);
17        close(pfd[1]);
18        return 0;
19    }
20    /* Padre lee en el lado lectura ... */
21    close(pfd[0]);
22    while((nread = read(pfd[0], buf, sizeof(buf))) > 0) {
23        printf("buf=%s, nread=%d\n", buf, nread);
24        write(1, buf, nread);
25    }
26    wait(NULL);
27 }
```

Este código crea un pipe, lo divide en dos descripciones (pfd[0] para lectura y pfd[1] para escritura), y luego ejecuta fork(). Si es hijo, lee del pipe y lo escribe de vuelta. Si es padre, lee lo que el hijo escribió y lo imprime. Los comentarios resaltados en rojo explican el flujo de control y las acciones de lectura y escritura.

El 2do IF lo ejecuta el hijo y lo de abajo lo ejecuta el padre.

Lo primero que hago es ejecutar la llamada a sistema PIPE, le tengo que pasar una regla de dos enteros (pfd) que está declarado arriba (PDF PIPE FILE REGISTER).

Si en ese IF se logra crear el PIPE devuelve los números de descriptores y si da error retorna un número menor que 0, también lo podría haber hecho más brusco sin necesidad de poner el primer IF.

Luego me encuentro con un FORK(), si es cero ejecuta esa parte del código del IF del hijo, al final le pongo el RETURN(0) entonces el hijo termina ahí sí o sí, el resto del código solo lo ejecuta el padre.

El hijo lee del teclado y lo guarda en el buffer BUF. Hay que recordar que por más que ese buffer tenga 256 el READ va a retornar lo que leyó del teclado que podrían ser 5 o 10 bytes.

Luego escribe en la tubería el PFD[1] es el de escritura , escribe el BUF, NREAD, ese NREAD es la cantidad que lei de teclado.

Luego con CLOSE cierra ese descriptor PFD[1] y termina. Cuando el programa termina no tiene sentido cerrar el descriptor con ese CLOSE, se podría borrar. Esto es porque cuando un proceso termina cierra todos sus descriptores.

El padre como va a leer cierra el extremo PFD[1] dado que no va a escribir.

Entonces con el while se queda en un bucle esperando a que el hijo le escriba, cuando la lectura de 0 significa que no hay nada en el buffer y que además se cerraron todos los extremos de escritura, porque si aun hubiese un extremo de escritura habilitado se queda bloqueado, no sale porque no sabe cuándo en el futuro podrían enviar datos nuevamente. La condición para que el READ termine es que no queden datos en el buffer y que todos los canales de escritura estén cerrados.

Luego a medida que voy leyendo voy escribiendo unos 1 en la pantalla lo que leyó.

Luego espera a que termine el hijo. WAIT(NULL).

```
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase3$ ls
Clase3-2020.odp  Makefile  mkfifo.c  receive.c  send.c  test-pipe2  test-pipe.c
ejemplo-pipe.avi  mkfifo  receive  send  test-pipe  test-pipe2.c
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase3$ vim test-pipe.c

[1]+  Stopped                  vim test-pipe.c
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase3$ fg
vim test-pipe.c
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase3$ make test-pipe
cc      test-pipe.c -lrt -o test-pipe
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase3$ ./test-pipe
[1]+  Stopped                  vim test-pipe.c
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase3$ fg
vim test-pipe.c
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase3$ make test-pipe
cc      test-pipe.c -lrt -o test-pipe
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase3$ ./test-pipe
```

Compilo, ejecuto y veo que se queda esperando. Hay dos procesos. El hijo está bloqueado esperando del teclado STDIN, y el padre está bloqueado esperando en el extremo de lectura del PIPE.



```
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase3$ ls
Clase3-2020.odp  Makefile  mkfifo.c  receive.c  send.c  test-pipe2  test-pipe.c
ejemplo-pipe.avi  mkfifo  receive  send  test-pipe  test-pipe2.c
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase3$ vim test-pipe.c

[1]+  Stopped                  vim test-pipe.c
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase3$ fg
vim test-pipe.c
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase3$ make test-pipe
cc      test-pipe.c -lrt -o test-pipe
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase3$ ./test-pipe
hola
hola
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase3$
```

El primer hola es ingresado por teclado, se desbloquea el hijo, luego se encarga de escribir en el PIPE, y el padre que estaba bloqueado esperando el PIPE se despierta, lee, y cuando el hijo termina se cierra el canal de entrada entonces directamente el proceso padre sale de ese bucle y termina.



```
1 #include <stdio.h>
2 #include <unistd.h>
3 #include <sys/wait.h>
4
5 int main(void) {
6     int pfd[2];
7     int nread;
8     char buf[256];
9
10    pipe(pfd);
11    if (fork()==0) { /* hijo */
12        nread = read(0,buf,sizeof buf);
13        sleep(2);
14        write(pfd[1], buf , nread);
15        return 0;
16    }
17    /* Padre lee en el lado lectura ... */
18 // close(pfd[1]);
19    while((nread=read(pfd[0], buf, sizeof(buf))) > 0) {
20 //        printf("buf=%s, nread=%d\n", buf, nread);
21        write (1, buf, nread);
22    }
23    wait(NULL);
24    return 0;
25 }
```

Con la demora que agregue luego del NREAD del hijo, al escribir hola, demora 2 segundos hasta que lo escribe en el PIPE para que luego el padre se desbloquee.

Pero si comento en el padre la línea CLOSE(PFD[1]) tendría dos entradas en el pipe. Entonces por más que termine el hijo donde cierra todos los descriptores, el padre va a estar leyendo pero como aún hay un proceso conectado a la entrada de la tubería se va a quedar bloqueado porque no sabe a futuro cuando le van a escribir nuevamente. Entonces en este caso se está bloqueando a sí mismo.

**Es por esto que es muy importante cuando uso PIPE CERRAR LOS EXTREMOS QUE NO VOY A USAR.**

```
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase3 carlost@maquinola:~/  
1 #include <stdio.h>  
2 #include <unistd.h>  
3 #include <sys/wait.h>  
4  
5 int main(void) {  
6     int pfd[2];  
7     int nread;  
8     char buf[256];  
9  
10    pipe(pfd);  
11    if (fork() == 0) { /* hijo */  
12        close(pfd[0]);  
13        nread = read(0, buf, sizeof buf);  
14        write(pfd[1], buf, nread);  
15        nread = read(0, buf, sizeof buf);  
16        write(pfd[1], buf, nread);  
17        return 0;  
18    }  
19    /* Padre lee en el lado lectura ... */  
20    close(pfd[1]);  
21    sleep(10);  
22    while((nread = read(pfd[0], buf, sizeof(buf))) > 0) {  
23 //        printf("buf=%s, nread=%d\n", buf, nread);  
24        write(1, buf, nread);  
25    }  
26    wait(NULL);  
27    return 0;  
28 }
```

Ahora repito la parte de escritura del hijo para que escriba dos veces, y el padre va a esperar 10 segundos y va a escribir todo.

```
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase3$ ./test-pipe  
holaaaa  
holaaaaa  
^C  
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase3$ make test-pipe  
make: 'test-pipe' is up to date.  
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase3$ vim test-pipe.c  
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase3$ vim test-pipe.c  
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase3$ make test-pipe  
cc -c test-pipe.c -o test-pipe  
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase3$ ./test-pipe  
holaa  
segunda escr  
holaa  
segunda escr  
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase3$
```

Lo muestra en dos renglones porque luego de escribir el primer hola hay un ENTER.

```

1 #include <stdio.h>
2 #include <unistd.h>
3 #include <sys/wait.h>
4
5 int main(void) {
6     int pfd[2];
7     int nread;
8     char buf[256];
9
10    pipe(pfd);
11    if (fork()==0) { /* hijo */
12        close(pfd[0]);
13        nread = read(0,buf,sizeof buf);
14        write(pfd[1], buf , nread);
15        nread = read(0,buf,sizeof buf);
16        write(pfd[1], buf , nread);
17        return 0;
18    }
19    /* Padre lee en el lado lectura ... */
20    close(pfd[1]);
21    sleep(10);
22    while((nread=read(pfd[0], buf, sizeof(buf))) > 0) {
23 //      printf("buf=%s, nread=%d\n", buf, nread);
24 //      write (1, buf, nread);
25 //      printf ("se leyeron %d bytes\n",nread);
26    }
27    wait(NULL);
28    return 0;
29 }

```

Agrego una línea en el padre con un PRINTF para saber cuántos bytes se leyeron

```

File Edit View Search Terminal Tabs Help
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase3$ ./test-pipe
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase3$ hola
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase3$ que tal
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase3$ ./test-pipe
hola
que tal
hola
que tal
se leyeron 13 bytes
carlost@maquinola:~/tecnicasIII/2021/cap2/Clase3$ 

```

Acá podemos comprobar que el padre hace una sola lectura, en el PIPE se van acumulando y luego se leen todas de una si en el SIZE(BUF) va a leer esa cantidad.

## FIFO: Tuberías con Nombre

### FIFO : Tuberías con nombre

- Permite compartir información entre diferentes procesos no relacionados
- Mensajes orientados a stream de datos ( no segmentados )
- La tubería persiste con un nombre en el filesystem
- Se pueden abrir configuradas como:
  - Bloqueantes
  - No bloqueantes
- Syscall **mkfifo(nombre,permisos)**

Ejemplo: mkfifo nombre , cat nombre , ls -l > nombre  
ls -l > nombre , cat nombre

Este IPC llamado FIFO que es muy parecido a PIPE, cualquier proceso si sabe el nombre o el PAD donde esta ese archivo, puede escribir en ese archivo, independientemente si están relacionados o no están relacionados los procesos.

Acá lo interesante es que el nombre de la tubería persiste en el sistema de archivos, pero los datos no persisten, una vez que el proceso termino se pierden esos datos.

Me va a crear un archivo en el sistema de archivos y después lo puedo utilizar, el cual se abre como un archivo común, leo y escribo ahí. El que lee si no hay nada se bloquea hasta que otro le escriba,

Video 2 26:17