# Resumen

## Primeros sistemas operativos

Los primeros SO no hacian mas que poner las tasks en memoria y todo lo que se ejecutaba estaba ahi.
Corrian un unico programa.
Generalmente el operador cargaba una tarjeta perforada

| **SISTEMA OPERATIVO** |
| --- |
| TASK |
| --- |

##Sistemas operativos de los 70
Se necesitaban sistemas que puedan ejecutar multiples jobs, como UNIX.

| **SISTEMA OPERATIVO** |
| --- |
| TASK 1 |
| --- |
| TASK 2 |
| --- |
| TASK 3 |
| --- |

El problema es que la memoria fisica es una sola.
Como hacer para que distintas tareas puedan usar la misma memoria y que ellos no requiera que tareas esperen por una posicion determinada?
Se crea el concepto de *virtualizacion de memoria*.
Ademas tengo un solo procesador para varias tareas, son necesarias las politicas de *scheduling*, se usan politicas para time sharing, lo que requiere tener cargado los procesos en memoria.

Todas estas cuestiones con el tiempo pasarian a ser responsabilidad del kernel en combinacion con herramientas de hardware.

# Virtualizacion de memoria

Cuando uno necesita virtualizar memoria necesita:

- Aislamiento
- Proteccion
  En ejecucion
  En memoria
  Para el aislamiento y proteccion el modo dual es crucial para separar los tipos de instrucciones que puede usar el kernel y el usuario.

*Addres Space*: Es el mecanismo de virtualizacion de memoria, es decir, un mecanismo de abstracion que permite manejar la memoria de forma transparente.

| **Memoria fisica** | **Virtualizacion** |
|---|---|
| SO | .text |
| .text | .data |
| --- | heap |
| --- | stack |
| --- | --- |
| heap | --- |
| --- | --- |
| stack | --- |
| --- | --- |
| data | --- |

El *address space* tiene toda la informacion de la memoria de un proceso.
En general el *stack* crece y el *heap* decrece.
Podemos ver los limites de los procesos con:

```
➜  ~ ulimit -a
-t: cpu time (seconds)              unlimited
-f: file size (blocks)              unlimited
-d: data seg size (kbytes)          unlimited
-s: stack size (kbytes)             8192
-c: core file size (blocks)         0
-m: resident set size (kbytes)      unlimited
-u: processes                       14309
-n: file descriptors                1024
-l: locked-in-memory size (kbytes)  16384
-v: address space (kbytes)          unlimited
-x: file locks                      unlimited
-i: pending signals                 14309
-q: bytes in POSIX msg queues       819200
-e: max nice                        0
-r: max rt priority                 0
-N 15:                              unlimited
```

| **Virtualizacion** |
| --- |
| .text |
| .data |
| heap |
| ↧ |
| ///// |
| ///// |
| ↑ |
| stack |

El heap empieza justo despues el .text y .data

El sistema operativo necesariamente tiene que virtualizar memoria, su principal caracteristica debe ser la transparencia, el programa no debe saber que esta siendo ejecutado con memoria virtualizada.
Esta transparencia tiene un costo para el sistema operativo, que debe hacerlo en el menor tiempo y ocupando la menor cantidad de recursos.
Las metas son:

- Transparencia

- Proteccion
- Eficiencia

> "Nadie protege al kernel de si mismo, se supone que el kernel sabe lo que hace."
> *-Mariano Mendez*

Para manejar la memoria habitualmente en linux se usa un "API", que no es una *syscall*.

> "Claro"
> *-Adeodato Simo Piqueres*

Un programa en C tiene 3 tipos de memoria:

- Stack
- Heap
- Extern

La unica por la cual debemos preocuparnos como programadores es el heap, en donde nos manejamos con *malloc* y *free*. Son dos library calls que dependen una de la otra.

La funcion *malloc* recibe la cantidad de bytes necesaria en el heap y en caso de falla devuelve *NULL*.
La funcion *free* por otro lado libera memoria dinamica que fue previamente alocada con *malloc*, *calloc* o *realloc*. Esta funcion no avisa si algo salio mal, tiene comportamiento indefinido y solo recibe la direccion de memoria.

> "La memoria se crea y se destruye, nunca se transforma."
> *-Mariano Mendez*

En este proceso se realizan las *syscalls* o library calls dependiendo el SO:

- sbrk
- brk

## Que es *brk* (break)?

Cambia la direccion final del heap a la indicada.

## Que es *sbrk*?

Cumple la misma funcion que *brk* pero en lugar de recibir una posicion de memoria recibe la cantidad de bits que debe trasladarse.

# Adress Translation

Una de las cosas que implica el proceso de virtualizacion es el *address translation*. Es el proceso que se utiliza para traducir una direccion virtual a una direccion fisica. Esto se hace por hardware, debe mantener el registro de la memoria en uso, que memoria esta libre y como esta siendo manejada la memoria.

> "En realidad los datos de la direccion virtual no estan en la direccion fisica"
> *-Mariano Mendez*

# Metodos de virtualizacion de memoria

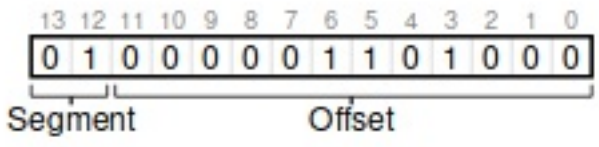
Existen varios metodos de traduccion de memoria virtual a memoria fisica y basicamente el mas viejo es el llamado *base* and *bounds*.

## Base and bounds

Tenes un offset y un limite para tu memoria, entonces asignas un registro a cada uno de ellos.
La posicion del segmento + el offset te da la direccion real, si esta sobrepasa el limite tenes un error de memoria. Esta traduccion se hace por hardware.
El offset en el registro *base* y el limit en el registro *bounds* pueden ser cambiados.
En x86 este metodo solo puede manejar 1MB de memoria para cada programa. Se llama *real mode*, cualquier arquitectura x86 arranca al principio en este modo y es tarea del sistema operativo salir del mismo.

## Segmentation

Es una tabla de base and bounds donde todos los sectores de memoria en C se dividen en partes de memoria. La direccion virtual se divide en partes que indican a que segmento pertenece y su ubicacion en el mismo.




La tabla de segmentos se encuentra en memoria.

**Narrador en off:** ***La clase de mariano fue irrumpida por la conexion de la notebook de Dato, en este momento se aprovecha a preguntar cosas a mendez o hablar de trivialidades.***

## Cuando es el parcial?

> "La fecha de parcial es en la semana 13 de la cursada, porque hay un solo parcial... y en realidad, que te voy a tomar? Te voy a tomar todo"
> *-Mariano Mendez*

> "Si, el 16 de Noviembre"
> *-Adeodato Simo Piqueres*

**Narrador en off:** ***Dato irrumpe el momento distendido para explicar paginacion.***

# Paginacion

El kernel se inventa los address spaces y le da uno a cada proceso, es un espacio de direcciones virtual. Cada proceso puede tener la misma direccion virtual pero se traduce de forma distinta. El kernel debe encontrar regiones en donde cada segmento queda en forma contigua.
Lo que quiere el sistema operativo en realidad es que segmentos contiguos no requieran ser contiguos en memoria fisica. Esto tiene por objeto simplificar la vida del SO y evitar la fragmentacion de la memoria.

> "Bueno, a partir de ahora la memoria fisica la voy a considerar que esta dividida en cachitos pequenios a los que voy a llamar paginas, paginas fisicas"
> *-Adeodato Simo Piqueres*

El sistema operativo ahora da paginas para cada segmento, por lo que la memoria fisica ya no tiene que ser contigua. Todos los address spaces van a estar ahora divididos en paginas. El kernel sabe en donde se aloja cada pagina de cada proceso.

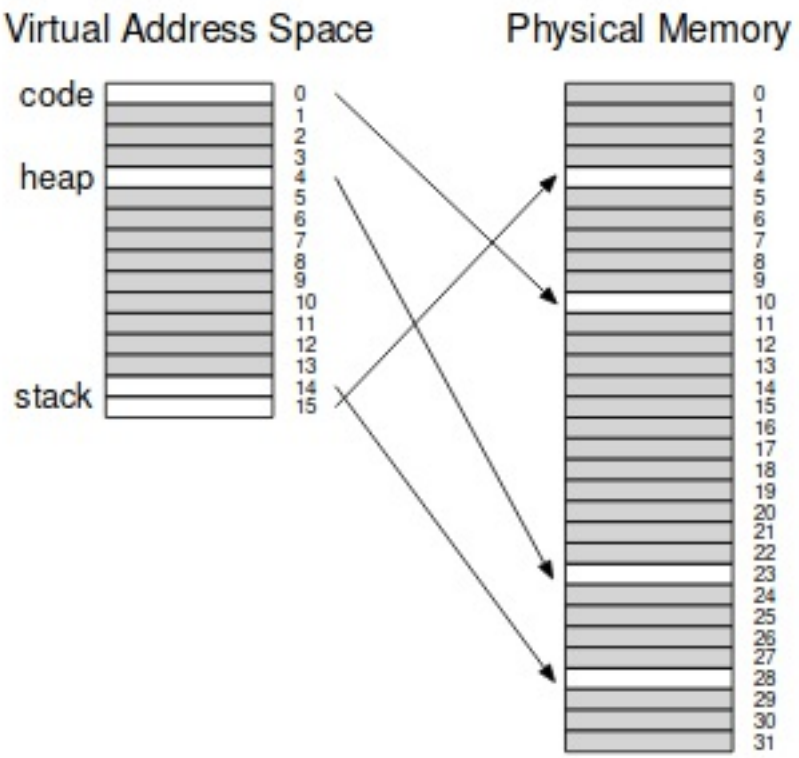



Figure 20.1: **A 16KB Address Space With 1KB Pages**

> Lo que nos va a llevar el resto de la clase es ver como diablos se implementa esa flechita. La MMU es esa flechita.
> *-Adeodato Simo Piqueres*

Al igual que la memoria fisica un address space es un arreglo de bytes conceptualmente. Si nosotros queremos numerar esos bytes en una maquina de 32 bits aunque hubiera mucha mas memoria en numero mas grande que yo puedo escribir es $2^{32}$, ese es el ultimo byte que puedo referenciar.
Por ej: Cuantas paginas de 4K hay en 4GB? Un millon.

> Si tengo el mismo programa corriendo dos veces el segmente que es igual en ambos procesos puede apuntar a la misma paginas
> *-Adeodato Simo Piqueres*

El sistema operativo al inicializarse es crear una lista de paginas libres para sacar de ahi paginas cuando se necesite memoria.
Dada una direccion de memoria virtual, como se que numero de pagina le corresponde? En terminos de bits hay una parte que corresponde al page number (bits del 12 al 31) y otra a su desplazamiento en el mismo (del 12 al 0).

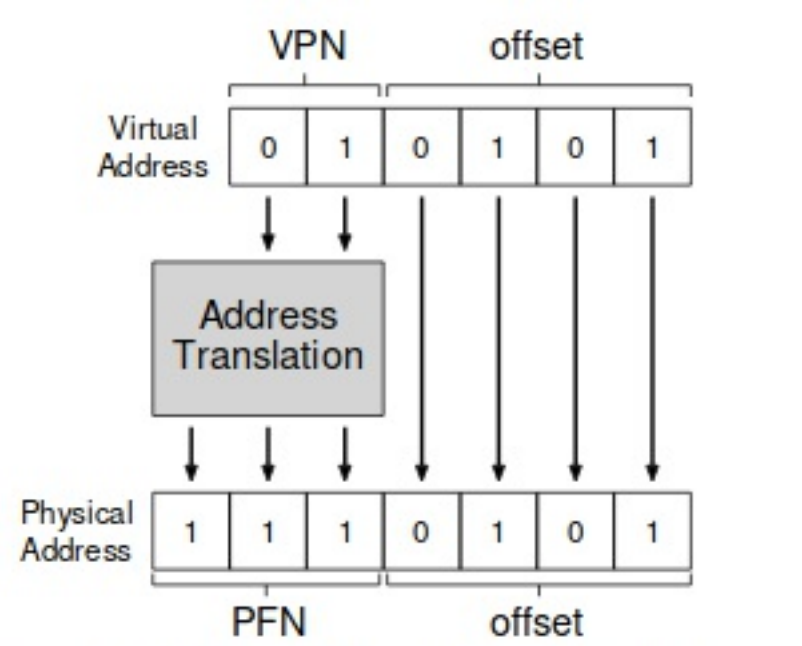


Figure 18.3: The Address Translation Process

Esta traduccion podria realizarse con una tabla de hash que dado un numero de hash te da una fisica. Pero, donde podria dejar el kernel escrito esto para que la MMU puedan consultarlo de forma rapida por hardware? Es poco viable.
Les propongo como alternativa una tabla con $2^{20}$ entradas en memoria que para cada numero de pagina indica cual es la posicion en memoria de esa pagina.

> Una vez que el kernel te ha asignado una flecha, y por lo tanto una memoria fisica a una pagina virtual, no hay razon para cambiarlo.
> *-Dato*[1]

[1] *A partir de aqui nos referiremos a "Adeodato Simo Piqueres" calidamente como "Dato" sin aviso previo porque es muy largo de escribir y nos cae muy bien*

**Narrador en off: *En este momento se producen muchas dudas y confusiones, tanto que un alumno segfaulteo.***

Ese arreglo es una guasada de memoria, son un millon de posiciones, 4kb por posicion, cada tabla para cada proceso ocupa 4MB, si tengo 100 procesos tengo 400MB.

> Nadie compra un sistema operativo que haga eso.
> *-Dato*

Lo que se hace es introducir multiples niveles de paginacionacion. En lugar de tener una tabla de 4MB vamos a tener dos niveles de tablas.

> Voy a usar la mitad de los bits de la izquierda para la primera tabla y la mitad de los de la derecha para la segunda.
> *-Dato*

Estos son 10 bits para cada direccion en cada tabla, por proceso voy a tener una tabla mas chica que la de antes que la voy a llamar *page directory*. Esta tabla tiene $2^{10}$ bits, osea, 1024 entradas.
Cada entrada apunta a la subtabla de 1024 entradas donde voy a usar los siguientes 10 bits.

> Cada tabla ocupa 4KB, oh, que casualidad, ocupa una pagina
> *-Dato*

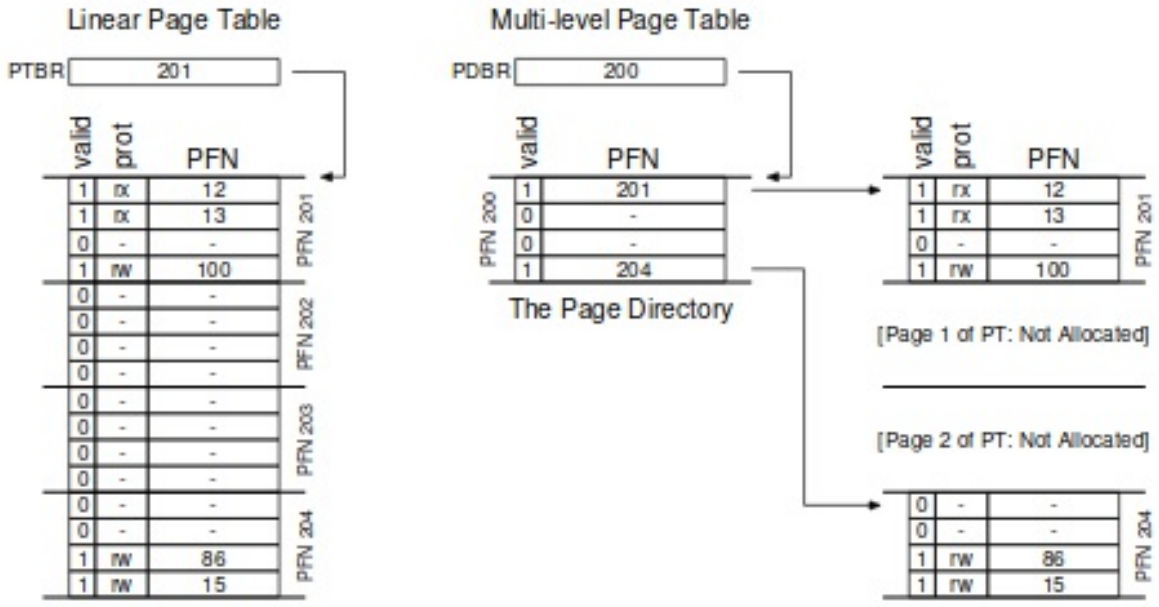



Figure 20.3: **Linear (Left) And Multi-Level (Right) Page Tables**

Hay un registro *cr3* para cada proceso en donde esta la direccion de la primer tabla.
La entrada 1 del *page directory* cubre de la pagina 0 a la 1023, y asi sucesivamente.
Que ganamos con esto? Cada tabla ocupa 4KB, ademas la mayoria de esas subtablas no van a existir.

**Narrador en off:** ***La clase esta terminando, Dato esta comentando que hay que hacer el TP1 y que es en grupo.***