Resumen

Base and bounds

El registro base te fija donde empieza el bloque de memoria y el otro donde termina.
Mientras mas grande son los registros mas memoria puedo direccionar.

Segmentation

Cual es la mejor forma de tener mas memoria usando los registros base and bounds?

Segment table:

Base	Bound	Access
		Read
		R/W
		R/W

Le sumo a la VA el base y me fijo si paso el bound.
Cada segmento del programa (*.data*, *heap*, *stack*, *code*) tiene su propio valor de base and bounds.
Te permite tener:

- Una tabla mas grande
- Mas memoria
- Usar el mismo hardware

Todas estas operaciones las hace el MMU. Cada VA se divide en:

Segment	Offset

Donde el segmento indica que tabla es y el offset su ubicacion.

Paginacion

La paginacion implica mayores niveles de indireccion.
La misma direccion de VA:

Segment	Offset

Se divide en:

Segment (10 bits)	Segment (10 bits)	Offset

El primer segmento indica la primer tabla, y el segundo la segunda pagina que esta en la tabla que la primera referencia.

Hacia una eficiente Address Translation

> Esto es todo un tema, porque, cuando tengo que acceder a memoria?
> -Mariano

> Siempre.
> -Martin Picco, alumno ejemplar.

Si tenes:

```
for(i=0; i<=1000; i++){
    X[i] = V[i]+W[i]
}
```

Hay dos principios, el de localidad y el de temporalidad.
El de temporalidad es que tenes que acceder al indice i, y la localidad es porque todo esta continuo en el mismo bloque de memoria, probablemente en la misma pagina.
Para que voy a hacer la traduccion en cada uno de esos casos?
Para esto se usa la cache:

> Si buscan la traduccion en ingles de cache, que a mi siempre la palabra en espaniol me trajo *escondrijo*, la palabra significa "escondido"
> -Mariano

La cache es limitada, no puedo tener toooda la memoria dentro del procesador, pero puedo agarrar un cachito de memoria y meterlo en el procesador.

Hay distintos niveles de cache, L1, L2..., cuando mas chico es el numero mas cercano al procesador esta.
La cache de la que hablamos, la *tlb*, no es ninguna de esas, es todavia mas pequenia y guarda lo que llamamos *lookaside buffer*.
La *tlb* es una tabla de pocas entradas que contiene traducciones de MMU ya hechas.
Va a guardar:

Virtual page number	Physical page number	Metadata

Guarda las direcciones del page directory ya transformadas (sin offset), por lo que ya no debe pasar por la page directory. Cuando va a buscar memoria se va a fijar si esta en tlb, si esta en la tlb se dice que es un *hit* y no tiene nada que traducir, solo suma el offset.
Tengo que saber cuales de la tlb estan activas o inactivas para saber si la puedo seguir usando en la parte de metadatos. Segun algunas politicas se van a invalidar esas entradas.
Como 128 entradas en la tlb no es suficiente hay dos niveles de tlb, si no esta en la primera tabla, se busca en la segunda.

> El problema de todo esto, es, que haces con los cambios de contexto? De proceso? Tenes que invalidar.
> -Mariano

> Yo no soy tan buena como para salir en tus apuntes...
> -Olivia Fernandez, alumna no ejemplar.

> Si queres simular el clima del mundo por 10 anios en una computadora te toma 3 meses en una computadora Quad-Core, porque tenes que multiplicar matrices de 4 dimensiones.
> -Mariano Mendez

Narrador: Ahre que ya no se entendia de que hablaban

> En el momento en que te interesa la eficiencia del programa deja de ser transparente el sistema operativo.
> -Kevin Spasiuk, otro alumno ejemplar

Scheduling

Que sabemos:

- El esquema de los procesos
- Como se maneja la memoria
- Como se crean los procesos
- Que es el kernel
- Que es un usuario

Y ahora tenemos que ver una cosa, que es, como sabemos que proceso se va a ejecutar?
El unico que decide eso es el kernel, y cuando lo hace? Cuando se despierta.
Hay tres maneras de pasar de un modo usuario a modo kernel:

- Por interrupcion
- Por que se termino el tiempo
- Syscall

> Syscall no vendria a ser tambien una interrupcion?
> -El pibe de Mulesoft que no me acuerdo el nombre

> Si, pero es una interrupcion mas copada.
> -Kevin Spasiuk

El Scheduling existe desde que existe el *time sharing*.

Narrador: Mariano le pregunto a Uriel Kelman en que edita sus apuntes, y nos recomendo usar MKDocs, que es lo que el esta usando para pasar los apuntes.

El sistema operativo debe estimar cuando va a terminar el proceso o que tan prioritario es el proceso a ejecutar para poder saber cuando tiempo darle. Esto lo hace automaticamente el *time scheduler*

> Es pura magia.
> -Mariano Mendez

Necesito usar una politica para poder definir si un proceso debe o no ejecutarse, y para usar una politica necesitamos parametros mesurables.
Una de las cosas que por tradicion se suele decir es que cuando uno habla de scheduling se suele hablar de *job* en lugar de proceso.
El *workload* es cuanto trabajo tiene el proceso. Las tareas con poco *workload* suelen ser interacciones con el usuario.

Para empezar a calcular el tiempo del procesador vamos ser *naive*, vamos a tener supuestos, que todos son mentira.

Los supuestos sobre los *jobs* en ejecucion son:

1. Cada proceso se ejecuta la misma cantidad de tiempo.
2. Todos los *jobs* llegan al mismo tiempo para ser ejecutados.
3. Una vez que empieza un job sigue hasta completarse.
4. Todos los *jobs* usan unicamente CPU.
5. El tiempo de ejecucion (*run-time*) es conocido.

En base a estas mentiras vamos a plantear unas metricas para planificar.
Vamos a utilizar una unica metrica, el *turnaround time*. Esto es el tiempo que tarda en terminar una tarea restado por el tiempo que tarda en llegar a ser ejecutada. Debido al supuesto 2 el tiempo de *arrival* o que tarda en ser ejecutada es 0.

Politicas para sistemas Mono-Procesador.

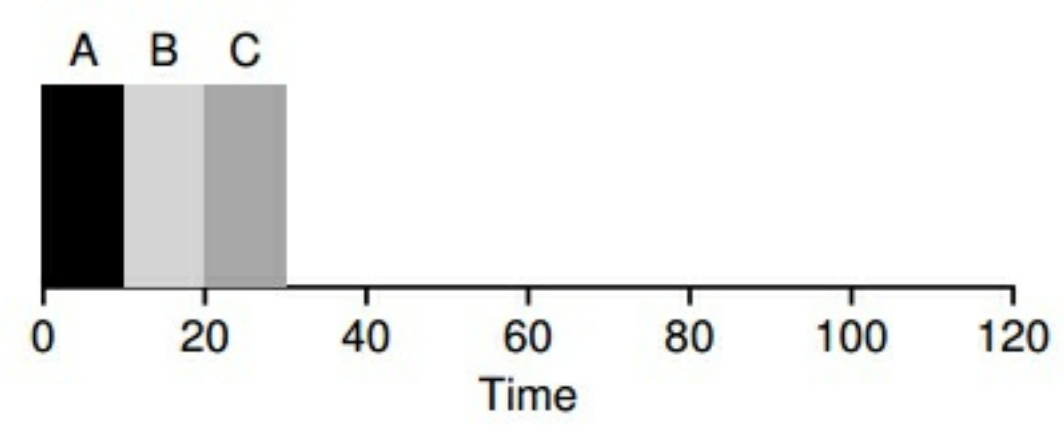
FIFO

La politica de planificacion mas facil que hay es la First In, First Out.

> Esto es facil de implementar funciona perfecto para las suposiciones falsas que hicimos.
> -Mariano

Si damos un ejemplo con procesos A, B y C en donde cada tarea dura 10 segundos, entonces el *turnaround time* es:

```
(10+20+30)/3 = 20
```



Ahora sin la suposicion 1 A dura 100 segundos. Entonces el *turnaround time*:

```
(100+110+120)/3 = 110
```

![fifo2.jpg]

> Ya si sacamos un supuesto esto es una cagada, no nos sirve. Perdonen mi frances.
> -Mariano

Este desastre se llama convoy effect.

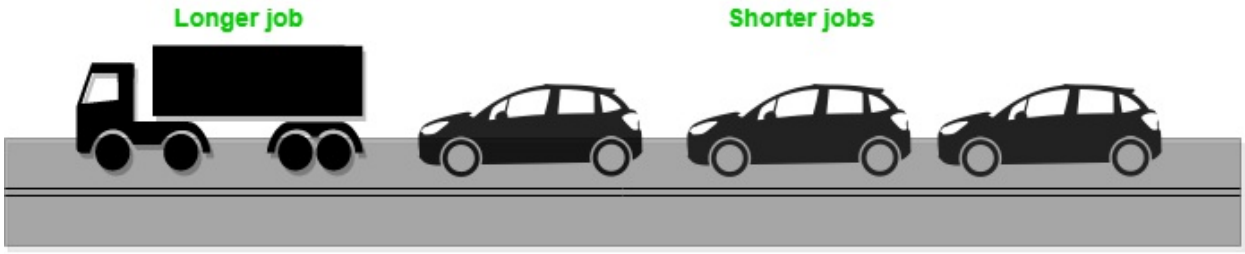


Figure - The Convey Effect, Visualized

Shortest Job First (SJF)

Es lo mismo que FIFO pero la que es menor es la que se ejecuta antes. En el mismo ejemplo:

```
(10+20+120)/3 = 50
```

Ahora vamos a relajar la suposicion 2, no llegan todos al mismo tiempo.
Llega primero A, despues B, despues C.

```
(100+110-10+120-10)/3=103.33
```

Shortest Time-to-completion

Vamos a ejecutar el mas corto, pero el mas corto en ser terminado.
Para realizar esto se relaja la suposicion 3.

```
(120-0+20-10+30-10)/3=50
```

Una nueva metrica: tiempo de respuesta

El tiempo de respuesta surge con el advenimiento del *time sharing* ya que los usaurios pretenden una interaccion rapida con la terminal.

Esto es el tiempo que tarda en ejecutarse por primera vez menos el tiempo que llevo.

Para el ultimo ejemplo el tiempo de respuesta es 0, 10 y 20 respectiamente para A, B y C.
El tiempo de respuesta entonces es 3.33 segundos. Como escribir un planificador que tome en cuenta esto?

Round robin

Yo tengo un tiempo n y m procesos. Voy a hacer n/m y eso va a ser el *time slice* para cada procesos. Cada proceso va a tener un tiempo de corrida que va a ser igual para cada proceso.
Los procesos A, B y C llegan a ejecutarse en el mismo instante y tardan 5 segundos, si se utiliza SJF el tiempo de respuesta promedio es:

```
(0+5+10)/3 = 5
```

El tiempo de respuesta promedio para con RR es:

```
(0+1+2)/3 = 1
```

Lo importante es elegir un buen *time slice*.
Por ejemplo, mi *time slice* es 10ms, ahora, mi *context switch* tarda 1ms en hacerse, osea, hay un *context switch* el 10% del tiempo. Si en vez de agarrar un *time slice* de 100ms sigo tardando 1ms de *context switch*, lo que es solo un 1% del tiempo pero la respuesta promedio baja.

La planificacion en la vida real

En la vida real se usa algo... que vamos a ver la clase que viene llamada Multi-Level Feedback Queue, que va aprendiendo de los procesos.

Narrador: Continuara…