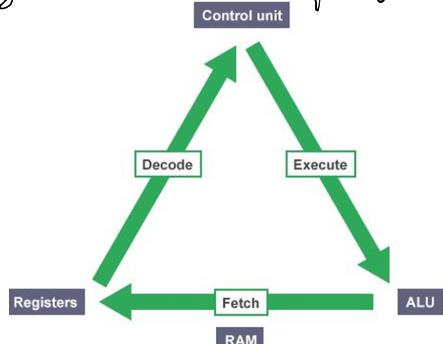
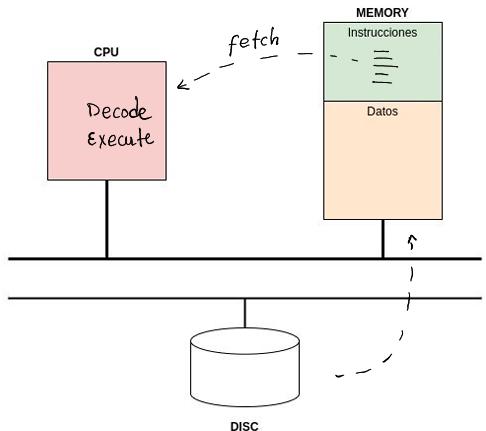


funcionamiento del sistema operativo

Ejecución de un programa



1. fetch: obtiene la instrucción de MEM
2. Decode: Averigua el tipo de instrucción
3. Exec: Realiza la operación indicada
4. luego salta a la proxima instrucción



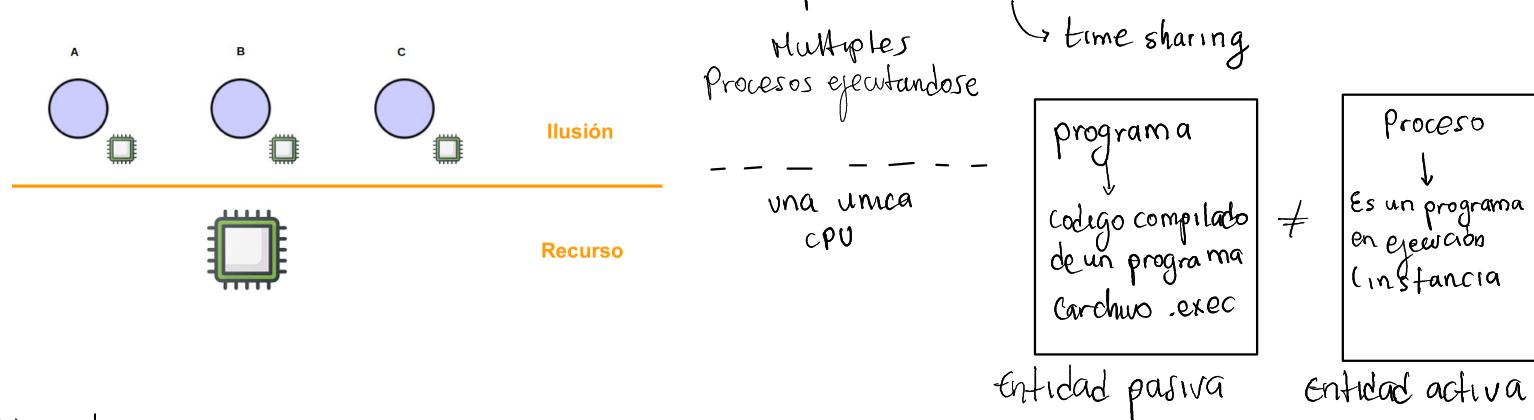
Entonces el sistema operativo, administra justa y eficientemente los recursos, controla la ejecución de los programas (accesos a API System). Brinda la abstracción de los recursos para facilitar el uso

Virtualización de recursos: El S.O transforma un recurso físico en una forma virtual de si mismo.



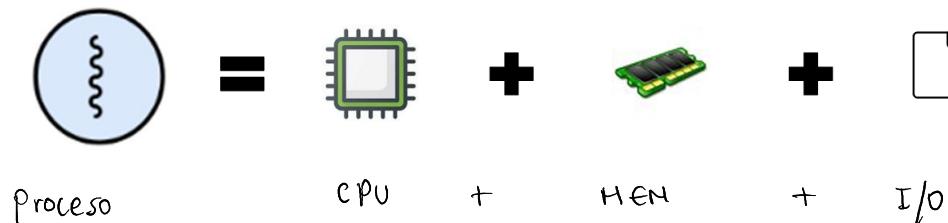
llamadas al sistema permiten que el usuario pueda decirle al sistema operativo que hacer:
API system.

Virtualización de CPU: Convertir una CPU en "infinitas CPUs virtuales".

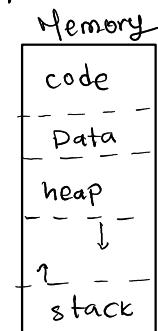


Ahora los procesos

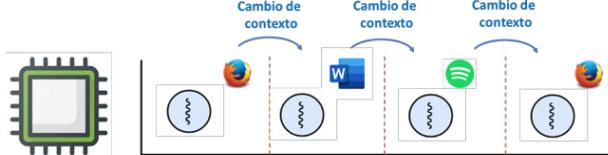
$$\text{Proceso} = \text{CPU} + \text{Memory} + \text{I/O Info}$$



Conformación de un proceso (Memoria)



Multitarea { una sola CPU puede cometer varias tareas a la vez }

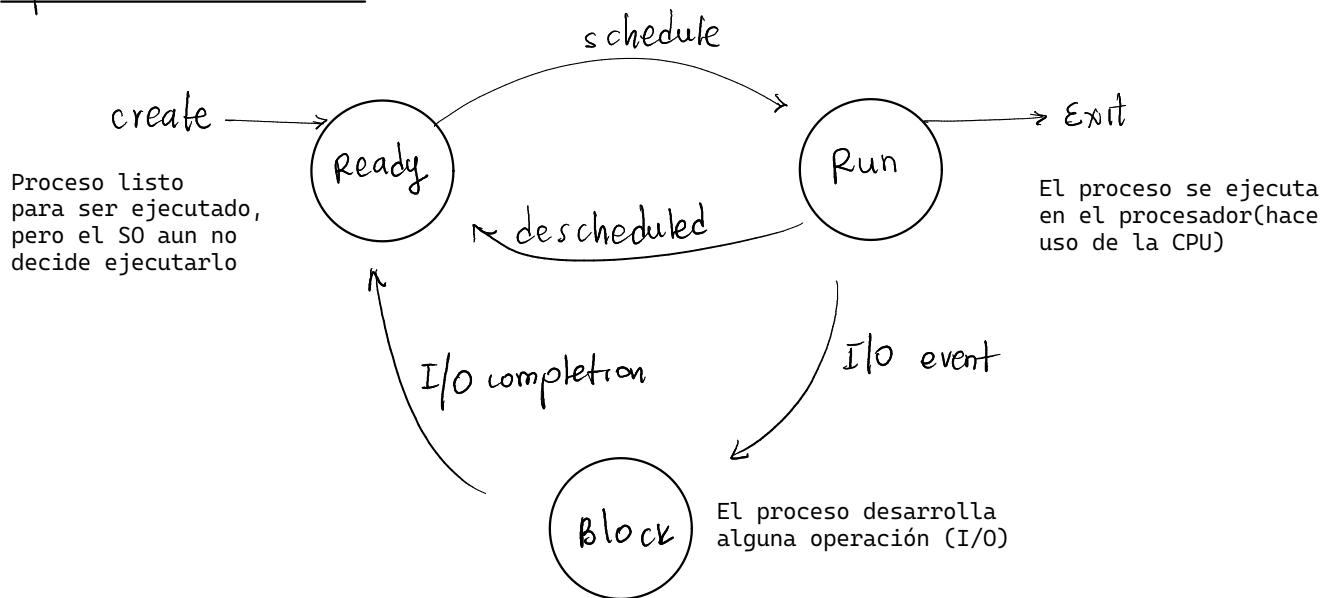


Usamos time sharing [cada proceso está un lapso de tiempo en la CPU]

Elementos importantes en S.O.

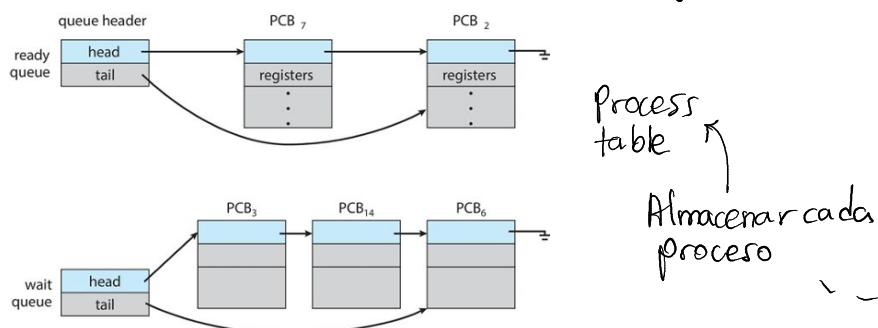
Process management	Abstracción	Proceso
	Mecanismos	create, destroy, wait, status,...
	Políticas	FIFO, RR,...

Modelo de 3 estados:



Elementos claves que requieren estructuras de datos

Process list: (listos, bloqueados o ejecutando)
contextos de registros



Process table

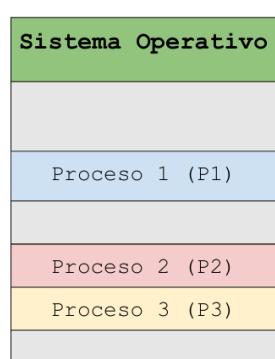
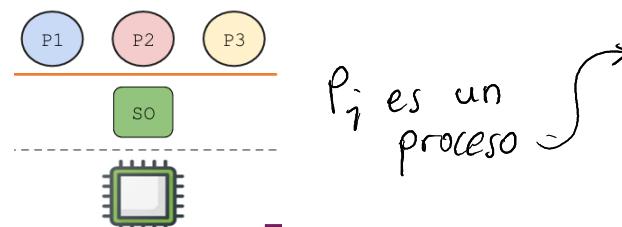
Almacenar cada proceso

mapear la información de cada proceso.

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

Elemento i de la tabla de procesos.

Por tanto sabemos que, la multi programación hace posible la ejecución de varios programas (alojados en memoria) a la vez y compartiendo la CPU

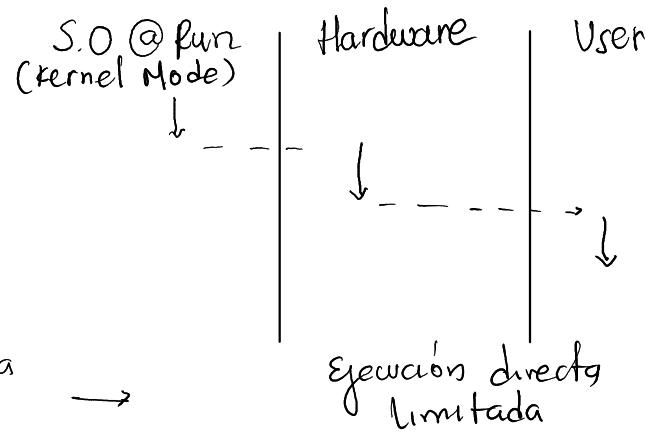
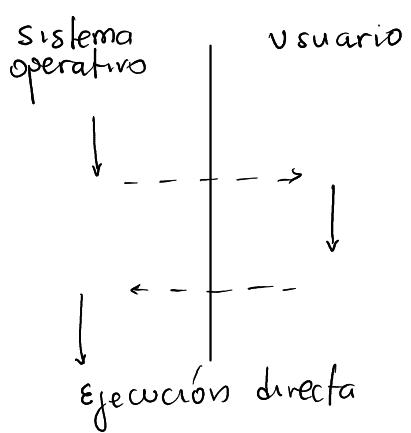


Ejecución directa limitada

LIMITED DIRECT EXECUTION

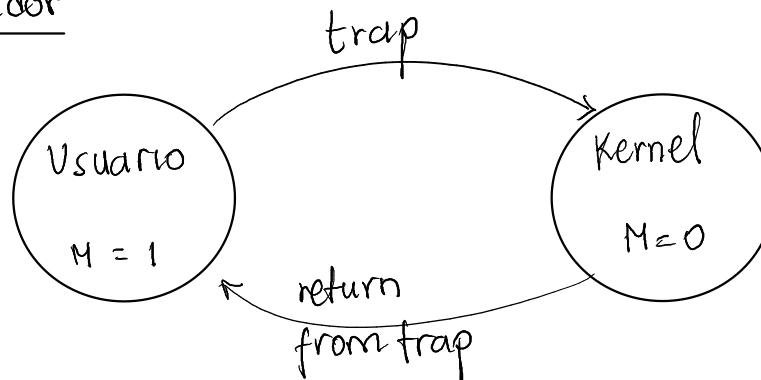
Protección (procesos aislados y protegidos) separando hardware y procesos

Aplicaciones deben ejecutarse directamente en la CPU. El SO valida las instrucciones de las aplicaciones (en run)

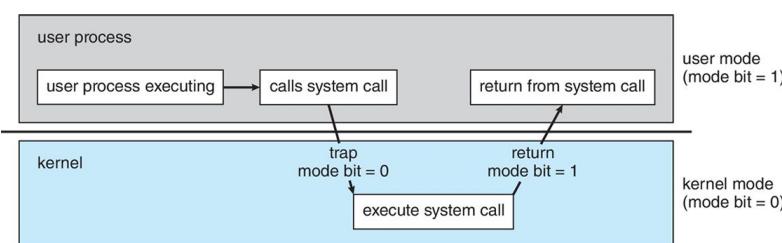


Modos procesador

No acceso privilegiado



Acceso al hardware
instrucciones privilegiadas



Syscalls → interfaz de programación que permite al SO exponer funcionalidades a programas de usuario

Win 32, POSIX, API de Java, ...

Retomando control de la CPU

- propuestas cooperativas: Ceder el control de la CPU mediante llamadas a sistema
- propuestas no cooperativas: El SO toma el control

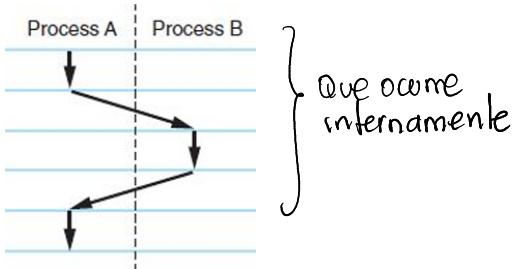
coop (no apropiativa)

- procesos liberan CPU periódicamente
- cambio de contexto (yield)

NO coop (apropiativo)

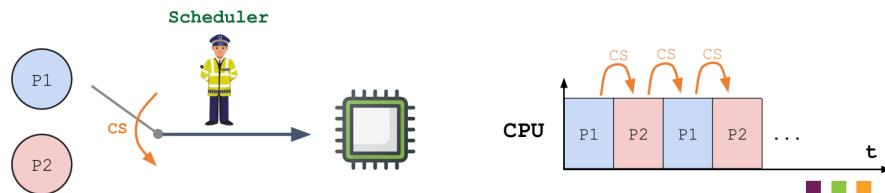
- tomar el control de CPU mediante timer interrupciones.

Cambios de contexto



- Almacenar el estado del proceso (en el PCB), es decir, debemos guardar valores de CPU en el kernel stack
 - Registros de propósito general
 - PC
 - Kernel stack pointer
- Restaurar valores del proceso
- cambiar el kernel stack del nuevo proceso

políticas de scheduling

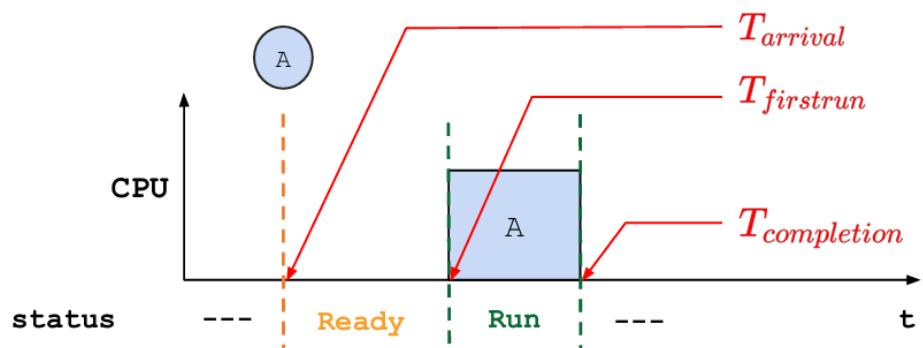


↓
¿que proceso se debe elegir para que use la CPU?
↓
dentro de los procesos que tengo en ejecución (workload)

Métricas de planificación

- Turn around time
- fairness
- Response time

Tres, Tat
↓ tiempos bajos
se quiere esto



turn around time: tiempo comprendido entre la llegada y planificación de un proceso

$$T_{turnaround} = T_{completion} - T_{arrival}$$

Response time: tiempo medido desde que el trabajo llega hasta que es programado por primera vez

$$T_{response} = T_{firstrun} - T_{arrival}$$

Conociendo que,

$T_{arrival} (T_a)$ = tiempo desde el create hasta ready

$T_{firstrun} (T_{fr})$ = tiempo desde create hasta que se ejecuta

$T_{completion} (T_c)$ = tiempo hasta que deje de correr
(se vaya a exit)

FIFO - FCFS : la política FIFO maneja los procesos de acuerdo al tiempo de llegada
(El primer proceso que llega, es el primero en ser atendido)

No apropiativo

- Excelente tiempo de respuesta
- Da cabida al efecto convoy

SJF

Shortest Job first: Esta política de planificación se hace de acuerdo al tiempo de ejecución.

→ El planificador ejecuta primero los procesos más cortos

No apropiativo

- Buen comportamiento en turn around time
- Hay problemas de inanición cuando hay muchos trabajos ligeros y un proceso demorado.

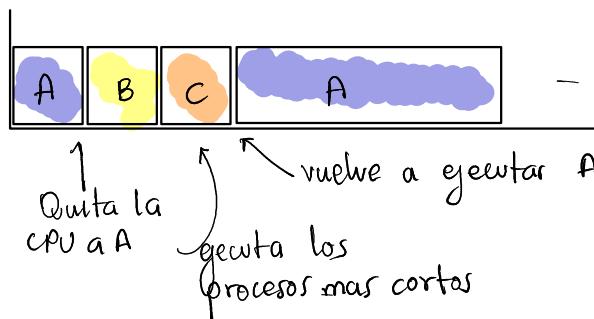
STCF

Shortest time to completion first: Versión apropiativa del SJF. Es una política parecida a la SJF, pero tenemos la posibilidad de detener un proceso para entregar la CPU a otro proceso que necesite menor tiempo en ejecutarse.

Apropiativo

s.o forma la cpu.

- Puede tener bajo desempeño en T response
- Pero Turn around time es bueno



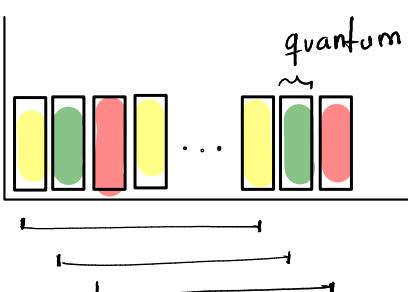
RR

Round Robin: planificar de acuerdo a porciones de tiempo (time slicing)

quantum: duración de esa porción

Luego ejecutar el siguiente trabajo en la cola de procesos listos.

- RR es justo (fairness)
- Bajo desempeño en Taround time



$q \downarrow$

- Mejor tiempo de respuesta
- Context switch es costoso

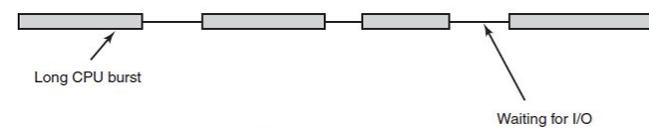
$q \uparrow$

- Disminuye el costo de c.s
- Disminuye el tiempo de respuesta

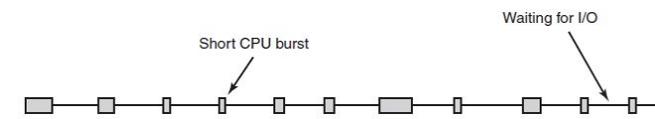
importante que cuando haya operaciones I/O se haga uso de la CPU con otros procesos

Tipos de procesos

Procesos tipo batch: Alto consumo de CPU (hay pausas cortas) (predomina el turnaround time)



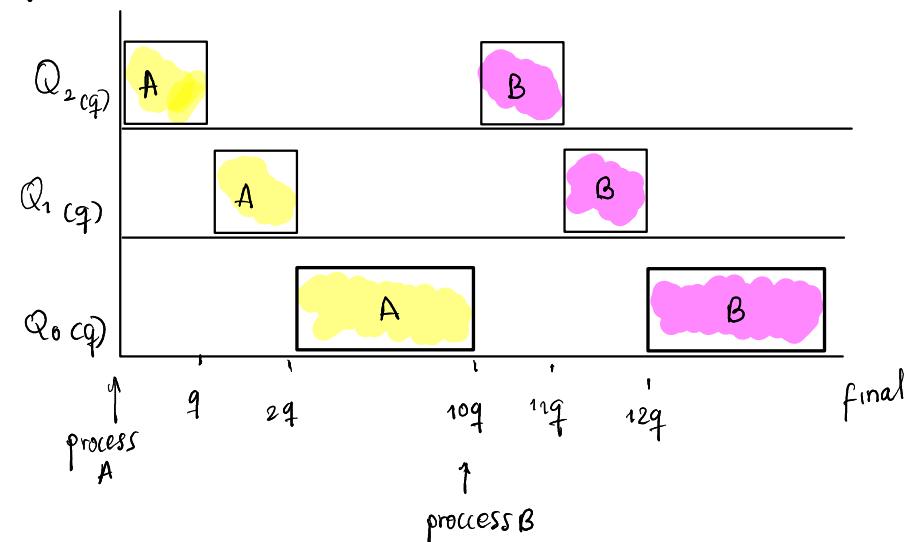
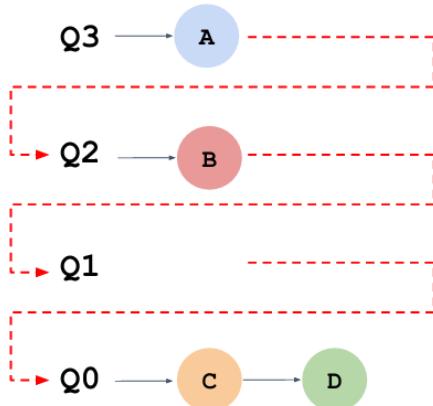
Procesos interactivos: Bajo consumo de CPU (largas pausas) (predomina el response time)



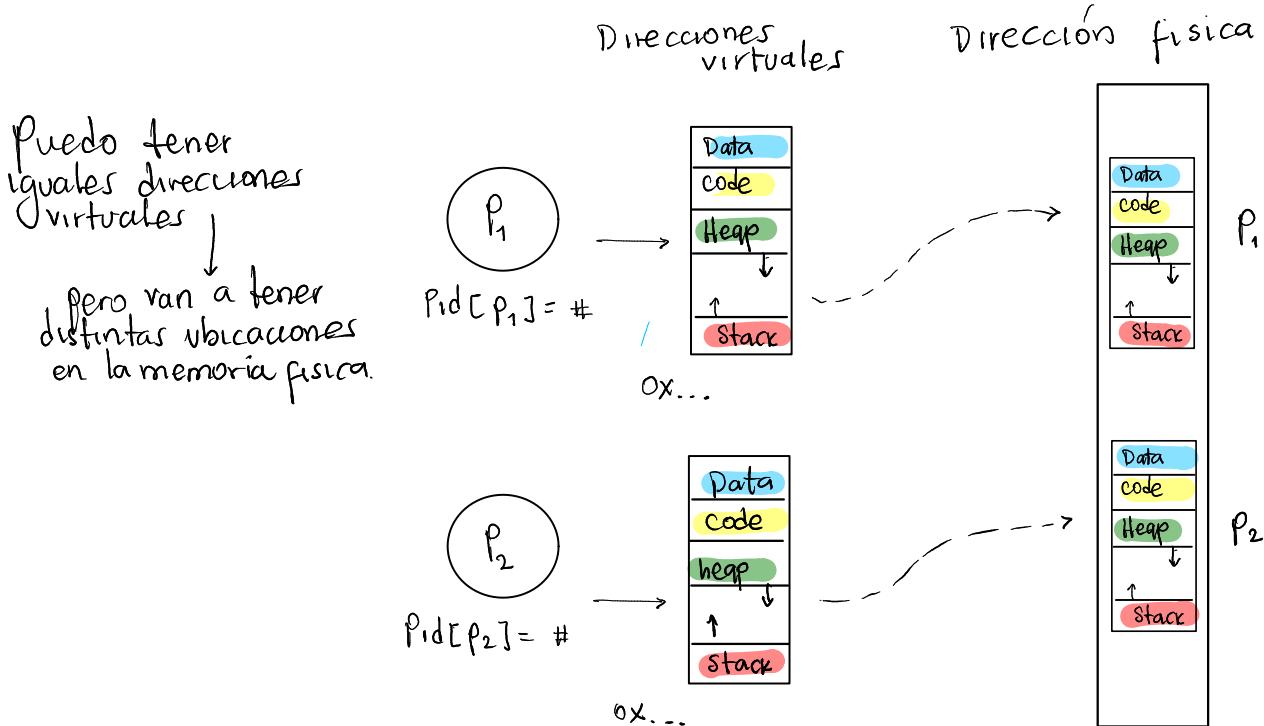
Planificador de propósito general (MLFQ)

Multi-level feedback queue:

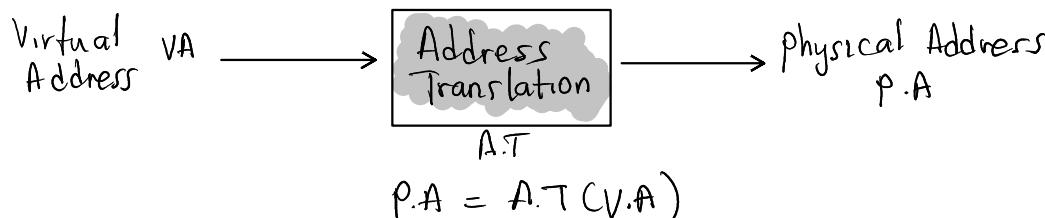
1. Si prioridad (A) > prioridad (B) A se ejecuta highest process I/O Bound (tiempo real)
2. Si prioridad (A) = prioridad (B) RR para A y B procesos sistema (procesos interactivos)
3. Cuando un trabajo llega al sistema es ubicado en la cola con prioridad mayor. procesos interactivos (batch process)
4. Reducir prioridad si se acaba el quantum lowest process CPU Bound
4. Mantener prioridad si no se usa durante todo el quantum
5. Despues de un tiempo s , mover todos los trabajos al mayor nivel de prioridad



Memoria Virtual: Cuando se crea un proceso, se crea un espacio de direcciones propio, cada proceso tiene una dirección propia.



Mecanismo para mapear traducciones de memoria virtuales → A la memoria física



Physical Memory	
0KB	Operating System (code, data, etc.)
64KB	Free
128KB	Process C (code, data, etc.)
192KB	Process B (code, data, etc.)
256KB	Free
320KB	Process A (code, data, etc.)
384KB	Free
448KB	
512KB	Free

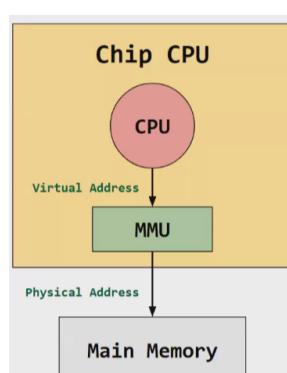
Espacio de direcciones: Abstracción de la memoria física creada por el sistema operativo

Program Code	<ul style="list-style-type: none"> Instrucciones Variables globales
Heap	<ul style="list-style-type: none"> Memoria asignada dinámicamente <ul style="list-style-type: none"> malloc en C new en Java o C++
Stack	<ul style="list-style-type: none"> Memoria automática Direcciones y valores de retorno Variables locales y argumentos pasados a rutinas

segmentos del espacio de direcciones

la virtualización de memoria requiere soporte en hardware:

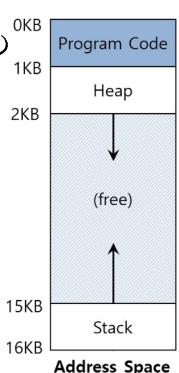
- Modos CPU
- Interrupciones
- Registros
- Segment tables
- TLB
- Page Tables



• Program code
• Data
• Heap
• Stack

} datos e instrucciones de un proceso en ejecución

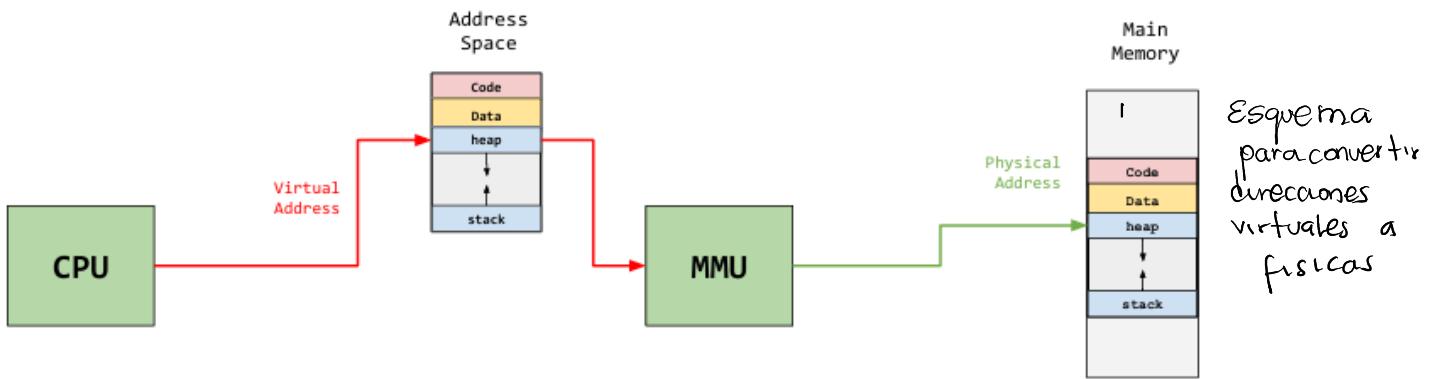
Mapa de memoria



Traducción de direcciones

↓
con ayuda del hardware

Traducción de direcciones



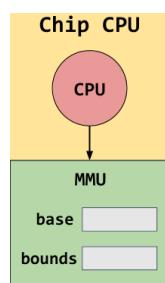
Mediante los mecanismos

1. Dynamic Realloc (Base and Bound)
2. Segmentation
3. pagination

Dynamic Reallocation

Hardware-based reallocation ó base & bound reallocation

- Base: Almacena la dirección física más pequeña
- Bounds: Almacena el tamaño del segmento de memoria virtual asociada al proceso



MMU: Memory management unit

la traducción de direcciones:

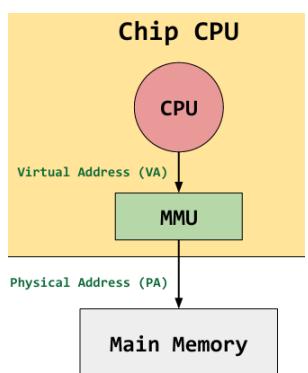
- Cuando el proceso es creado, el S.O elige donde cargar la imagen de memoria:

* fija un valor para el registro base

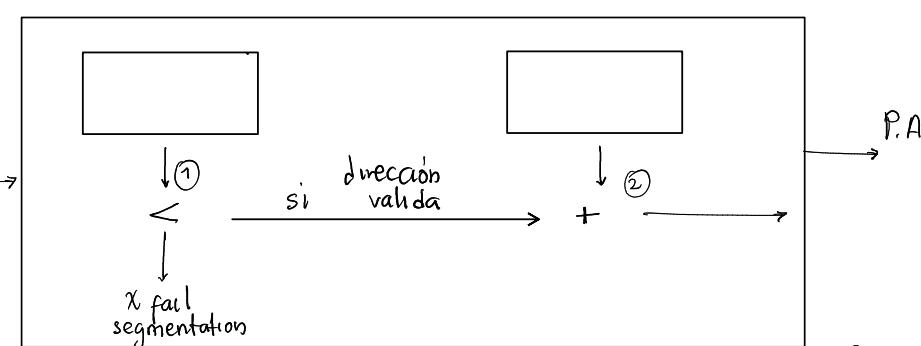
$$PA = VA + \text{base}$$

* las direcciones virtuales no deben ser mayores al valor en el registro "Bounds"

$$0 \leq VA \leq \text{bounds}$$



MMU



① chequear los límites
Entra una dirección virtual y comparamos si $0 \leq VA \leq \text{Bounds}$

② Sumar:
 $VA + \text{Base}$ [mapeo]

Bound: tamaño del espacio de direcciones

Problematika del Dynamic Realloc (Base and Bound)

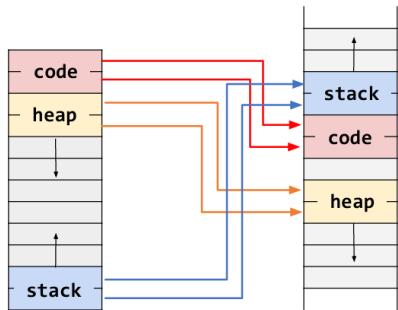
- fragmentación interna

Soluciona la segmentación [aca el problema es fragmentación externa]

Paginación: Virtualizar la memoria con páginas

Paginar: Dividir espacio de direcciones en fragmentos más pequeños

→ Mapear esos fragmentos pequeños



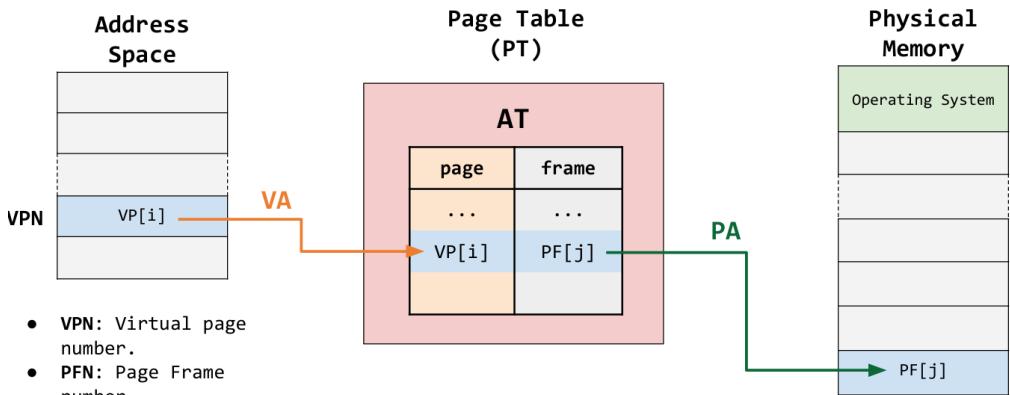
Page: Bloque de memoria virtual (Address Space) de tamaño fijo

páginas: N

frame: Es un bloque de memoria física de tamaño fijo el cual se encuentra asociado una page

frames: N

Page Tables: permite la traducción de VA a PA mapeando pages a frames



$$(\# \text{ pages}) \neq (\# \text{ frames}), \quad \text{size(page)} = \text{size(frame)}$$

VPN (Virtual page Number)
Índice asociado a una página en la dirección virtual.

PFN (page frame number)
Frame dentro del que se encuentra la dirección física

offset: desplazamiento dentro de la pag o frame

$$\star \text{num_frames} = \frac{\text{size(AS)}}{\text{size(frame)}}$$

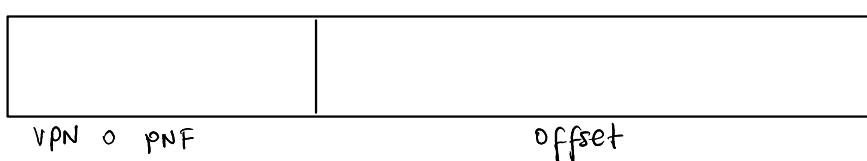
$$\star \text{num_pages} = \frac{\text{size(AS)}}{\text{size(page)}}$$

$$\star \text{Bits necesarios de la dirección virtual: } M = \log_2 (\text{size(AS)})$$

$$\star \text{Para los bits necesarios del offset: } N = n(\text{PFN}) + n(\text{offset})$$

$$\star \text{VA - offset} = m - n.$$

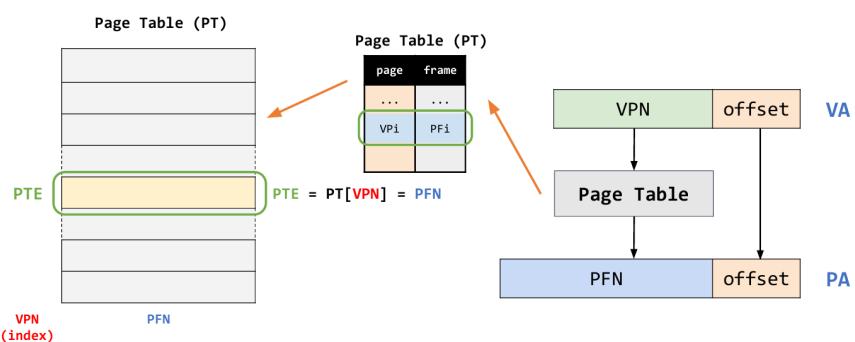
Address Space



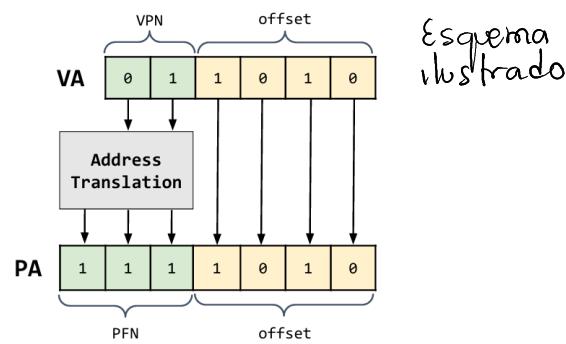
arreglo

Array que traduce direcciones virtuales a físicas.

PTE
↓
Entrada en una Page Table



PTE (Page Table Entry): Hay una por cada página en el AS.

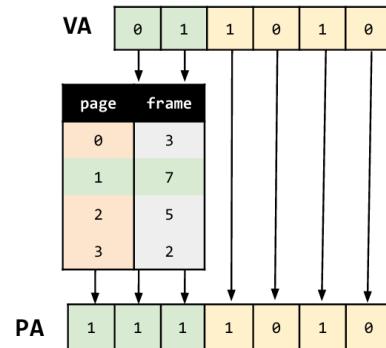


$$P.A = P.F.N \times \text{size}(\text{page}) + \text{offset}$$

Estructura de una PTE

Estructura de una PTE (Page Table Entry)

- **Bit de validez (V):** Indica si la traducción es válida.
- **Bits de protección (Prot):** read - write - execute.
- **Bit de presencia (P):** Indica si la página está en la mem. física.
- **Bit sucio (M):** Indica si la página ha sido modificada.
- **Bit de referencia (R):** Indica si la página ha sido accedida.

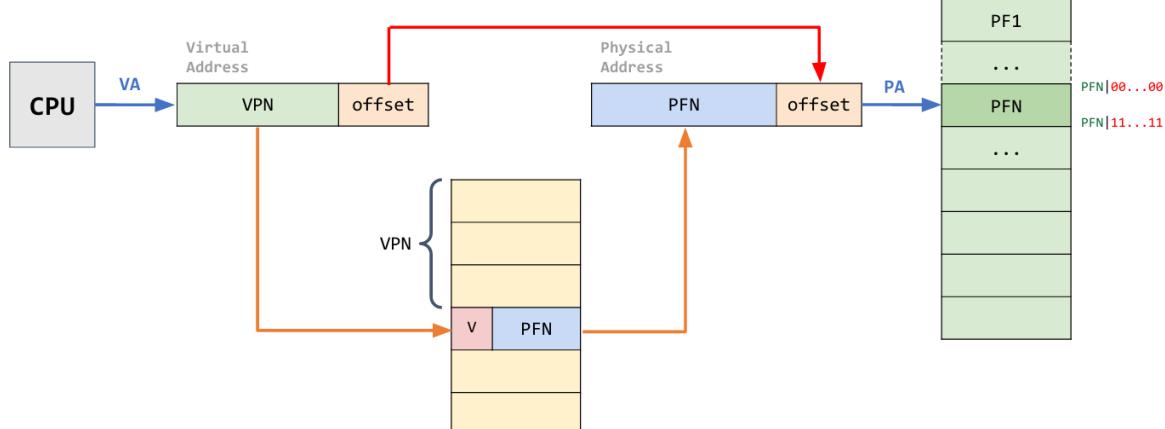


$$\# \text{ PTE's} = \# \text{ páginas}$$

$$\text{size}(PTE) = \text{size}(bits) + \text{size}(PFN)$$

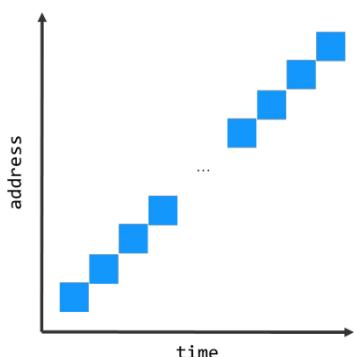
$$\text{size}(PT) = \text{num_pages} * \text{size}(PTE)$$

Hardware para la traducción.

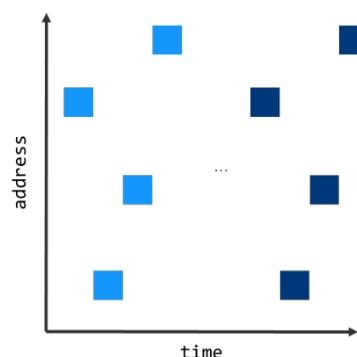


Traducción de Direcciones

Localidad espacial
Acceso secuencial



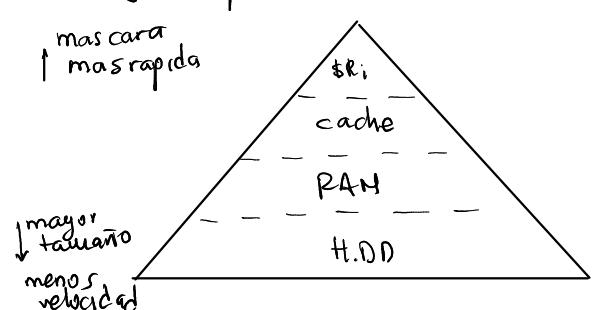
Localidad Temporal
Acceso aleatorio repetido



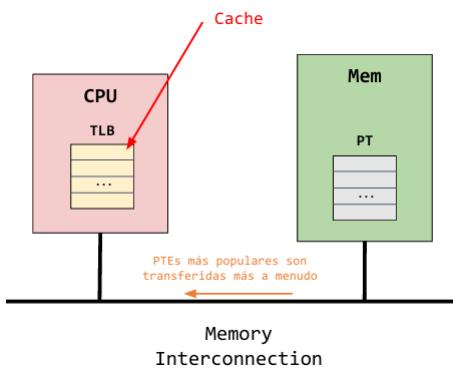
→ Punto de partida
(aprovechar dichas localidades)

Dado que para encontrar el PTE requerido necesitamos conocer la dirección de memoria donde se encuentra la tabla de pag.

↓ Jerarquía de memoria



TLB (Translation Lookaside Buffer)



Memoria cache en CPU

- Mantiene las direcciones más populares en la tabla de página.
- (Buscando acelerar el proceso de traducción de direcciones)

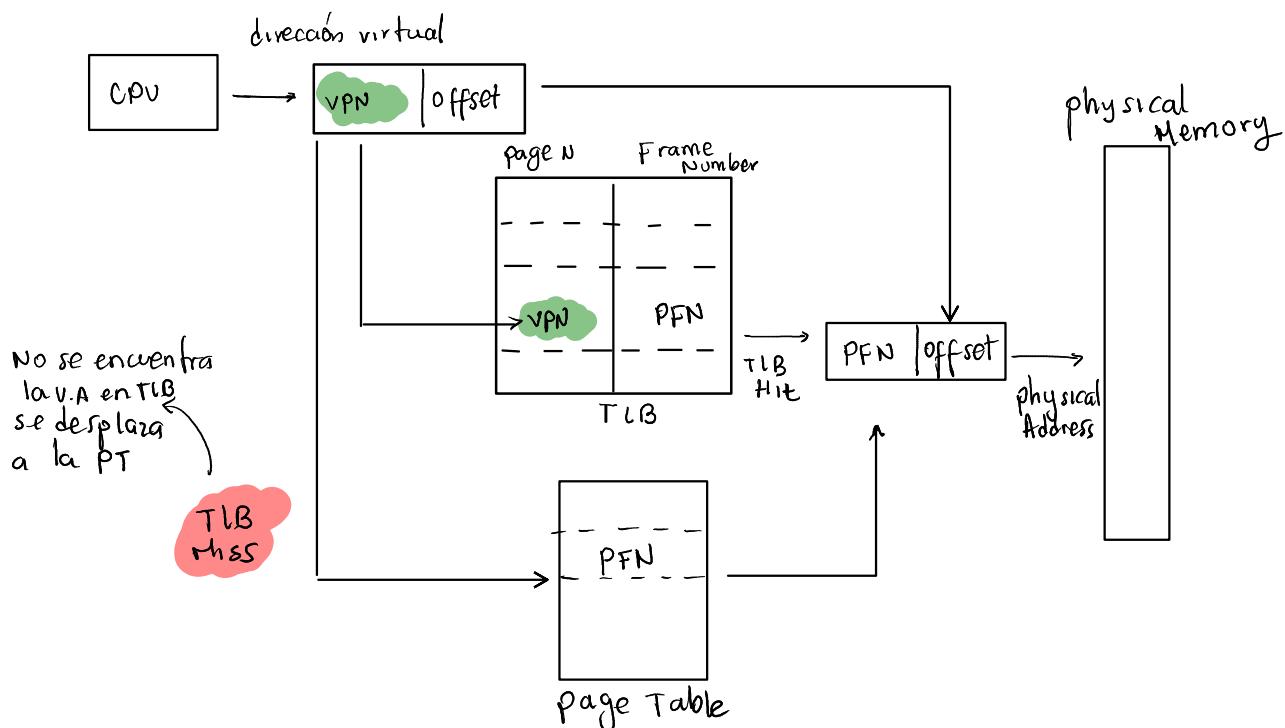
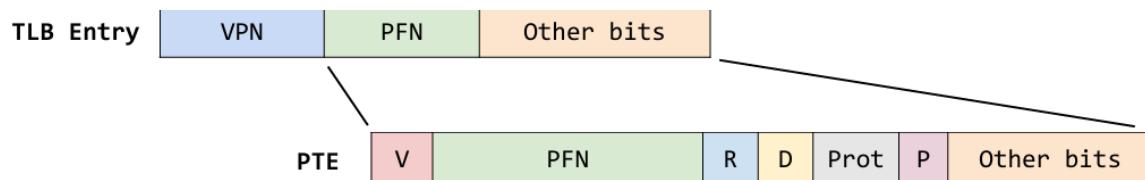
Hit → Encontrar en la cache
Miss → Desplazarse a la memoria física.

Contenido de una entrada TLB



Entrada TLB

- V (Valid)**: bit que dice si un PTE puede o no ser usada. Se evalúa cada vez que una dirección virtual es usada.
- D (Dirt)**: Indica si la salida ha sido modificada (Una operación write ocurre sobre esta).
- R (Reference)**: Indica si la página ha sido accedida. Es llevada a 1 (set) cuando la página ha sido leída o escrita. Es útil ya que:
 - Rastrea el acceso a la página.
 - Determina la popularidad de la página.
- P (Protection)**: Bits que controlan las operaciones que son permitidas (R, W, X, User/Kernel, etc).
- P (Present)**: Indica si la página se encuentra en memoria física o en el disco.
- ASID (Address Space Identifier)**: Lo veremos en breve.



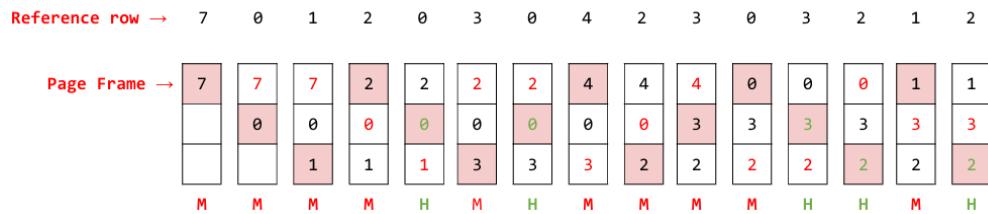
- políticas de reemplazo**: Queremos reemplazar una entrada antigua en la TLB (cache) por una nueva. Buscando reducir la tasa de Miss

LRU

Random

LRU (Last Recently Used)

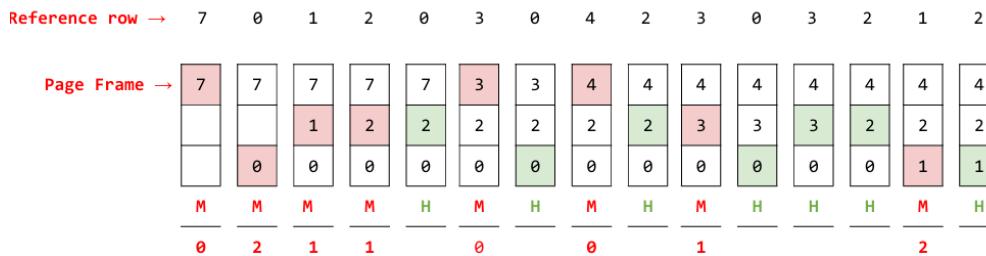
7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2



Reemplaza la entrada de la TLB que no haya sido usada recientemente (la que lleva más tiempo sin ser accedida)

Random

7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2



Realiza un reemplazo seleccionando una entrada al azar de la TLB.

¿Cómo ir más allá de la memoria física?

¿Cómo puede el sistema operativo usar un dispositivo más lento para proporcionar de forma transparente la ilusión de un gran espacio de direcciones virtuales?

Swapping

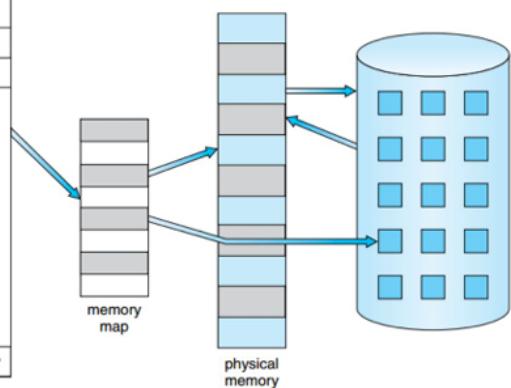
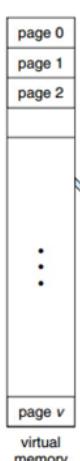
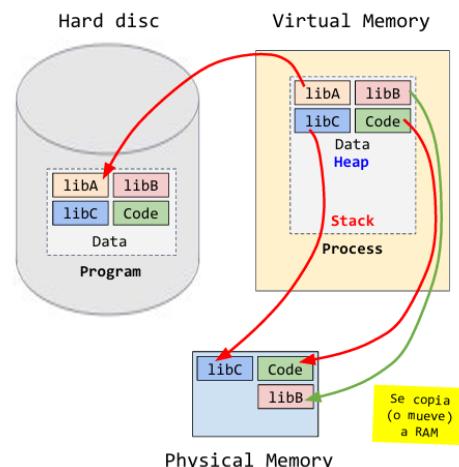
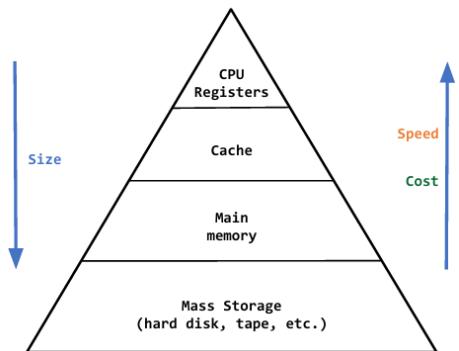
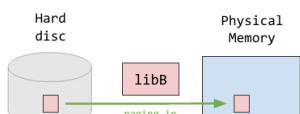
1. Ejecución del programa (reserva memoria virtual)

2. Se crean las referencias



3. se accede al Disco en caso de necesitar datos de allí

4. Se toma esa información y se mueve a la memoria principal.

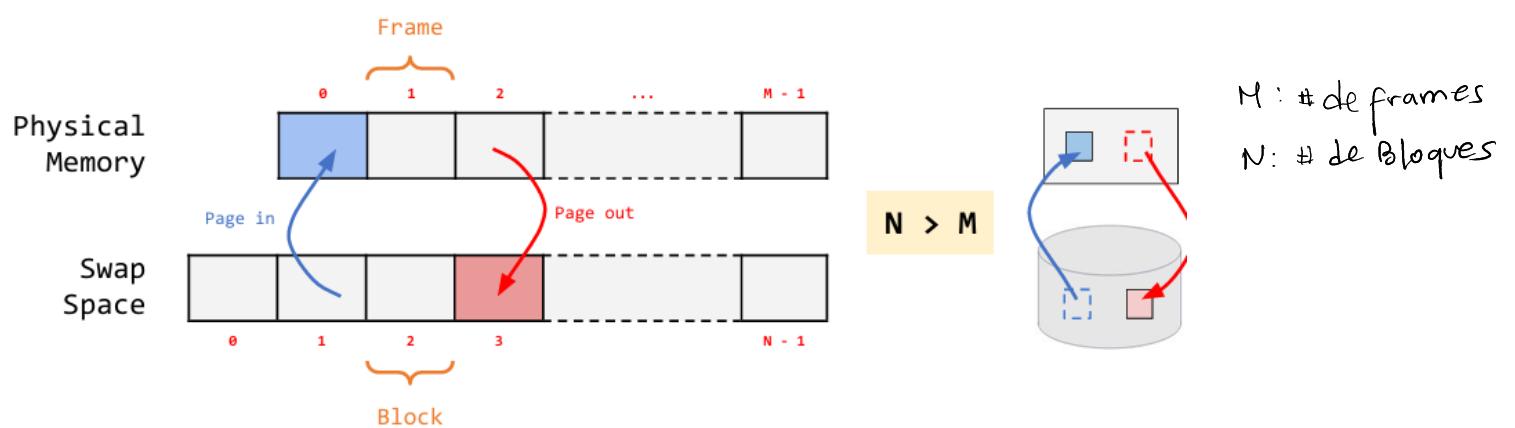


El espacio swap es una región del disco duro empleada para almacenar páginas en memoria

S.O divide el espacio swap en páginas:

Determina el número de páginas que el sistema puede utilizar

swap out → Page out
swap in → Page in.



Block: Espacio de memoria (página) en espacio Swap

Frame: Espacio de memoria (página) en memoria física.

P0 (Proc 0)			P1 (Proc 1)			P2 (Proc 2)			P3 (Proc 3)		
VPN	PFN	Block	VPN	PFN	Block	VPN	PFN	Block	VPN	PFN	Block
VNP0	PFN0	---	VNP0	---	Block3	VNP0	PFN3	---	VNP0	---	Block5
VNP1	---	Block0	VNP1	---	Block4	VNP1	---	Block6	VNP1	---	Block7
VNP2	---	Block1	VNP2	PFN1	---	---	---	---	---	---	---
...	VPN3	PFN2	---

Physical Memory	PFN 0	PFN 1	PFN 2	PFN 3
	Proc 0 [VPN 0]	Proc 1 [VPN 2]	Proc 1 [VPN 3]	Proc 2 [VPN 0]
Swap Space	Block 0	Block 1	Block 2	Block 3
	Proc 0 [VPN 1]	Proc 0 [VPN 2]	[Free]	Proc 1 [VPN 0]
	Proc 1 [VPN 1]	Proc 1 [VPN 1]	Proc 3 [VPN 0]	Proc 2 [VPN 1]
	Proc 3 [VPN 1]	Proc 3 [VPN 1]		

$P = 1$ (pag en memoria física)
 $P = 0$ (esta en el disco) } para usarlo con sentido $V = 1$

Rol de los bits V (Valid) y P (Present)

TLB Entry:

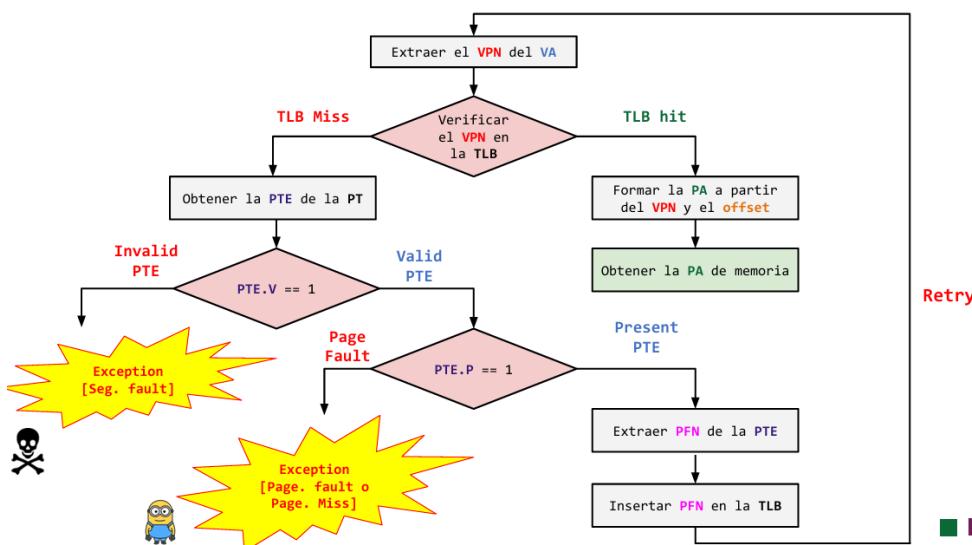
- Valid (V):** Dice si la entrada es una traducción válida o no.

VPN	PFN	Valid	Prot	ASID

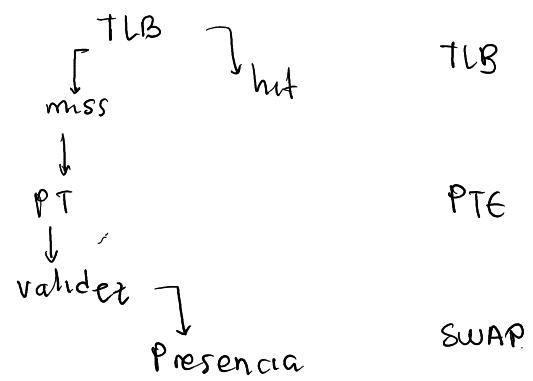
PT Entry:

- Valid (V):**
 - 1: El proceso asignó la página.
 - 0: La página no ha sido asignada por el proceso.
- Present (P):**
 - 1: La página se encuentra en memoria física.
 - 0: La página se encuentra en el disco

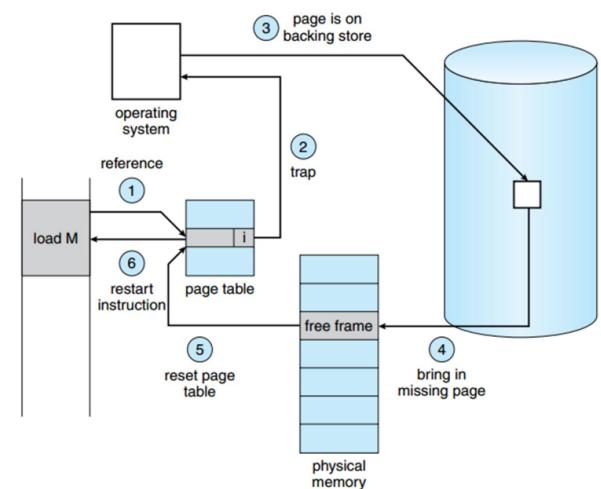
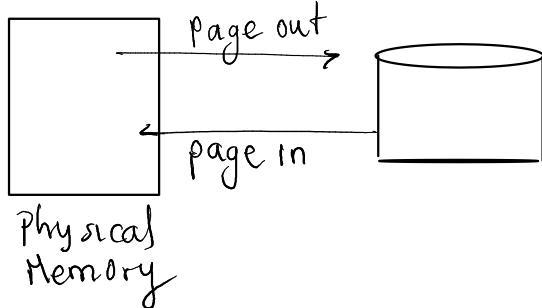
PFN	Valid	Prot	Present



Esquema de la traducción



Swap in: page in
swap out: page out



¿Como decidir cual pagina debo sacar de memoria?

AMAT - Average Memory Access Time)

$$AMAT = P_{HIT} * T_M + P_{MISS} * T_D$$

$$\left. \begin{array}{l} p_{\text{hot}} \rightarrow 1 \\ 0 < p_{\text{miss}} \leq 1 \end{array} \right\}$$

con esto en
mente

$$AMAT = T_M + P_{MISS} * T_D$$

Política de reemplazo óptimo (OPT):

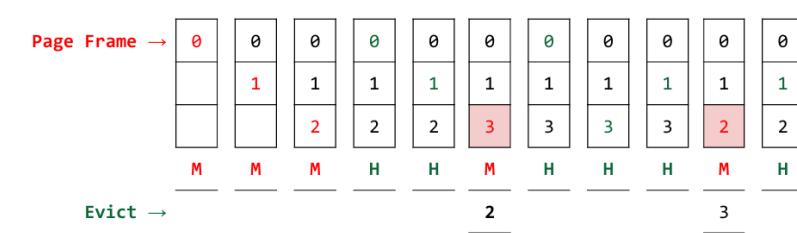
- futuro no conocido
- solución no real

(Asegura el menor numero de misses)

$$P_{\text{Hit}} = \frac{\text{Hits}}{\text{Hits} + \text{Miss}}$$

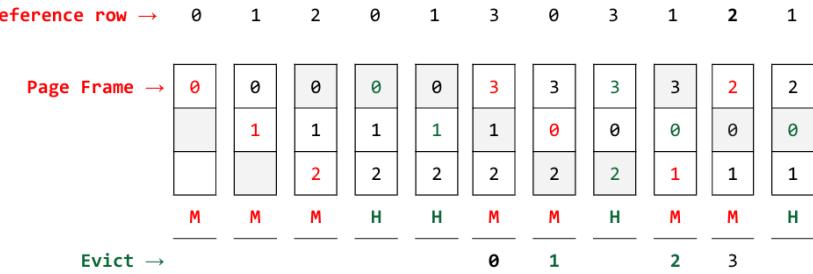
$$P_{\text{Hit}} = 6/11 = 0.545$$

P_{Hit} = 54.5 %



política de reemplazo FIFO: Reemplaza la página que ha entrado primero (las págs entran a una cola cada que llegan a la memoria) Seleccionando la página al final de la cola (la primera que entró)

$$P_{\text{Hit}} = \frac{\text{Hits}}{\text{Hits} + \text{Miss}}$$



política de reemplazo aleatoria: Selecciona de manera aleatoria la página a reemplazar
No tiene en cuenta que tipo de datos se saca de memoria

Reference row →	0	1	2	0	1	3	0	3	1	2	1
Page Frame →	0	0	0	0	3	3	3	1	1	0	0
	M	M	M	H	M	M	H	M	H	H	H

P_Hit = Hits/(Hits + Miss)
P_Hit = 5/11 = 0.4545
P_Hit = 45.45 %

Evict →

0	1	3

política de reemplazo LRU (Last Recently Used)

Información histórica	Descripción	Algoritmo
Frequency	Si una página ha sido accedida muchas veces, no debería ser reemplazada.	LFU
Recency	Una página que haya sido recientemente accedida, es muy probable que sea accedida de nuevo.	LRU

El desempeño depende de la suerte que se tenga al elegir los reemplazos

P_Hit = Hits/(Hits + Miss) Reference row → 0 1 2 0 1 3 0 3 1 2 1

P_Hit = 6/11 = 0.545 Page Frame → 0 0 0 0 0 0 0 0 0 0 0 0

P_Hit = 54.5 % M M M H H M H H M H H

Evict → 2

0