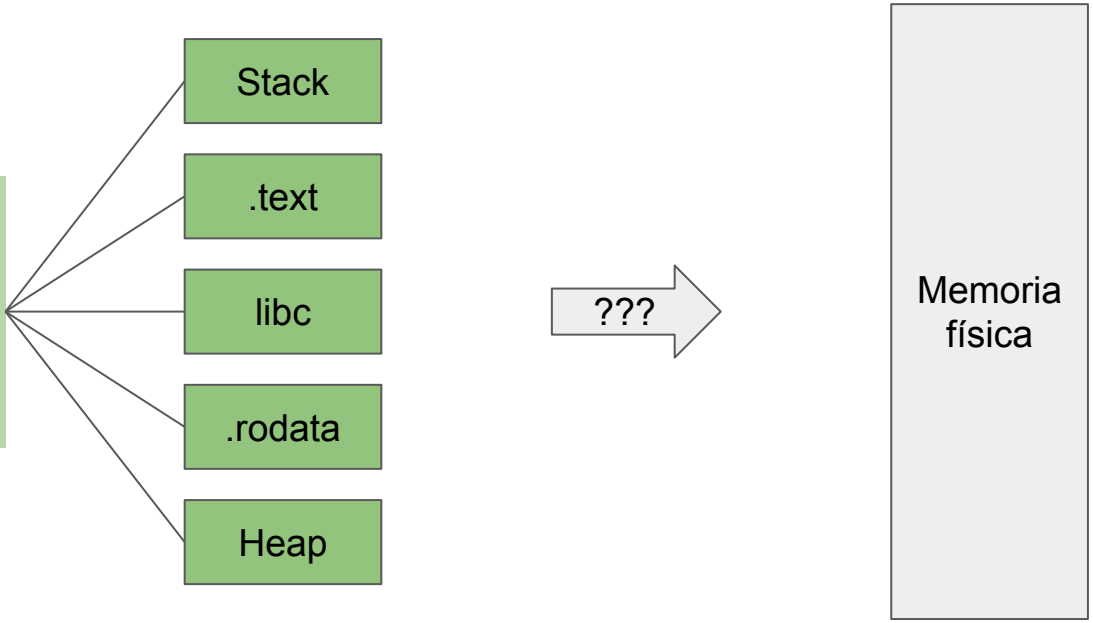
Sistemas Operativos 72.11

Memoria

Reflexión

```c
#include <stdio.h>

int main() {
    printf("Hello, World!\n");
    return 0;
}
```



Desde que comenzamos a programar no nos hemos tenido que preocupar por determinar dónde almacenar toda esta información y simplemente lo damos por hecho.

Uno no sabe lo que tiene hasta que lo pierde…

…vamos a perderlo (por un rato)

Vamos desde el comienzo

¿Cuáles son las principales **responsabilidades** de un sistema operativo (**SO**)?

- Proveer un conjunto abstracto y limpio de los **recursos**
- **Administrar** estos recursos

¿En qué **mecanismos** se basa una **SO** para lograr esto?

- **Abstracción**
- **Interposición**

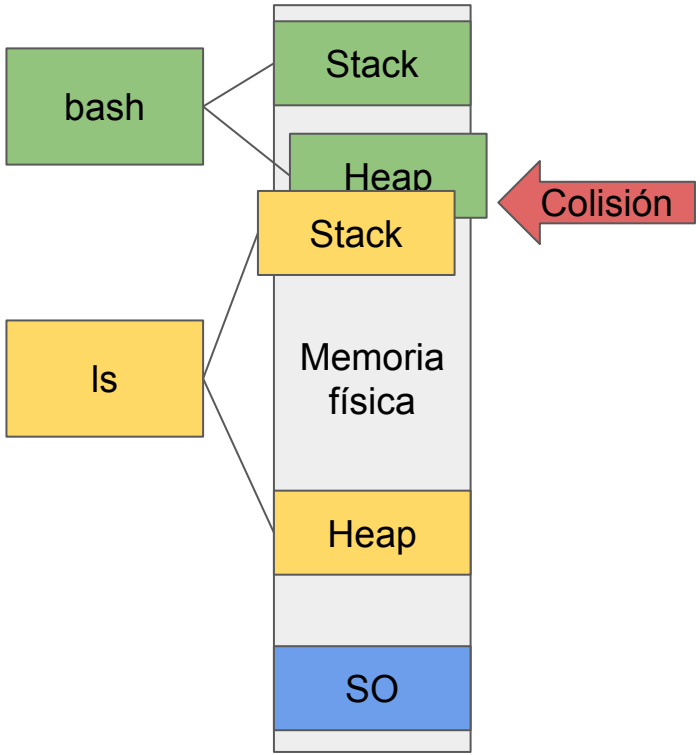
Instanciemos estos mecanismos en su versión más simple:

- Abstracción: **Sin abstracción**
- Interposición: **Sin interposición**

Cada proceso accede directo a memoria física

Problemas

Cada proceso accede directo a memoria física



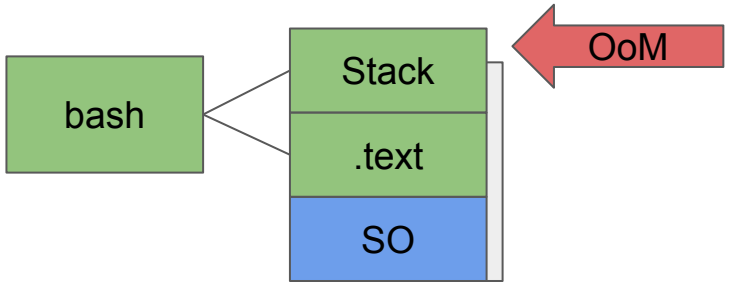
Es muy difícil tener múltiples procesos ejecutando simultáneamente

- Cada proceso decide unilateralmente qué parte ocupar
- Procesos con bugs o malintencionados
- ¿Quién protege al **SO**?

Traslademos esta situación a la asignación de espacios en el ITBA

Problemas

Cada proceso accede directo a memoria física



Es muy difícil ejecutar programas que necesitan más memoria que la tenemos disponible

Este problema no es nuevo… ni obsoleto…

En los 60s, ciertos programas para simulaciones eran divididos en **overlays**

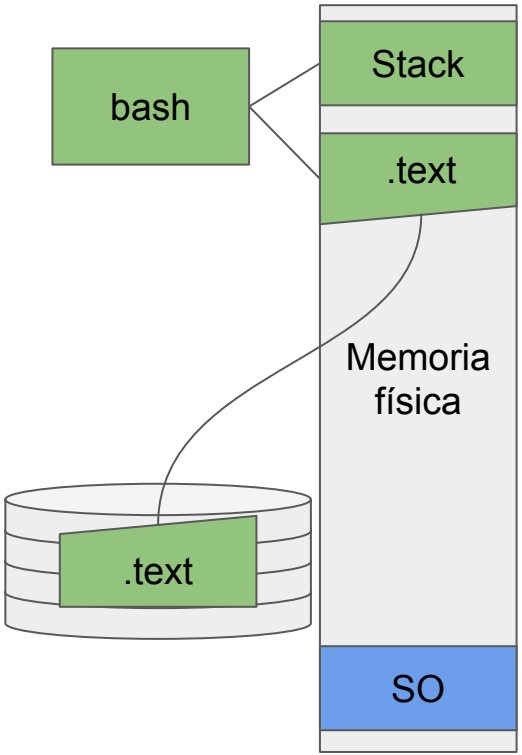
En los **90s** algunos juegos se dividían en múltiples **disquetes**

Actualmente algo parecido ocurre con los DLC

¿Quién se ocupa de particionar y decidir cuándo cargar cada parte?

Problemas

Cada proceso accede directo a memoria física

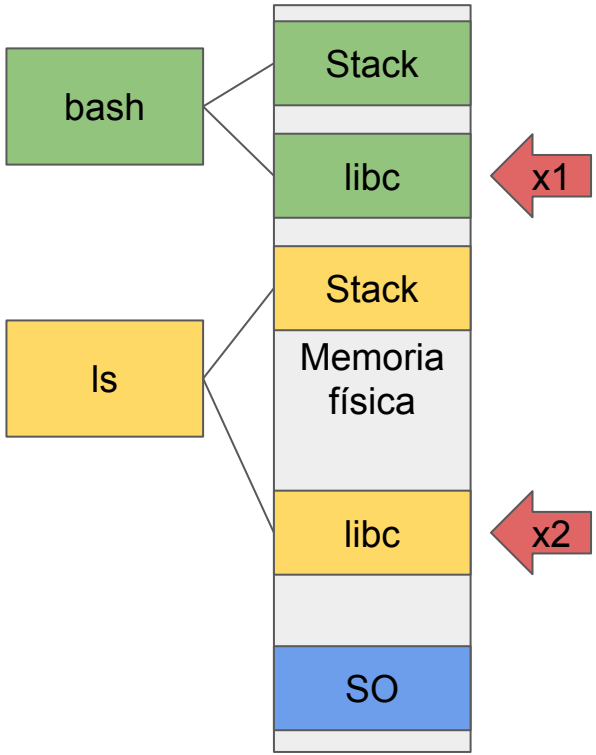


Es muy difícil ejecutar programas que no están completamente cargados en memoria

Traslademos esta situación a la lectura de una enciclopedia con múltiples tomos

Problemas

Cada proceso accede directo a memoria física



Es muy difícil que múltiples procesos que comparten una librería se pongan de acuerdo

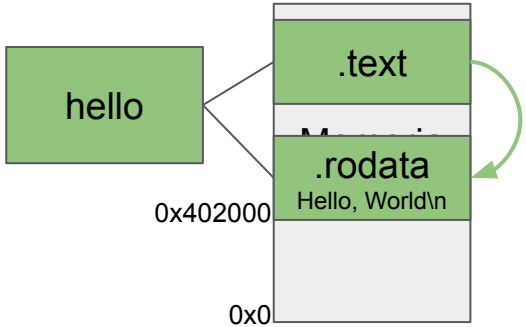
- Cada proceso tiene su copia privada

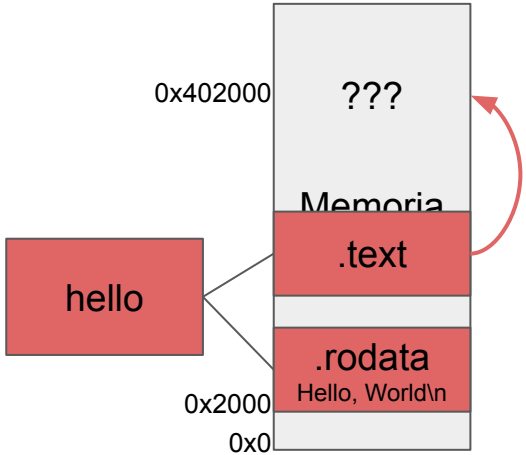
Traslademos esta situación a los anuncios del campus

¿Cuánto ocupa libc?

Problemas

Cada proceso accede directo a memoria física





Es muy difícil que un programa que maneja direcciones físicas absolutas funcione si no está cargado exactamente donde debería

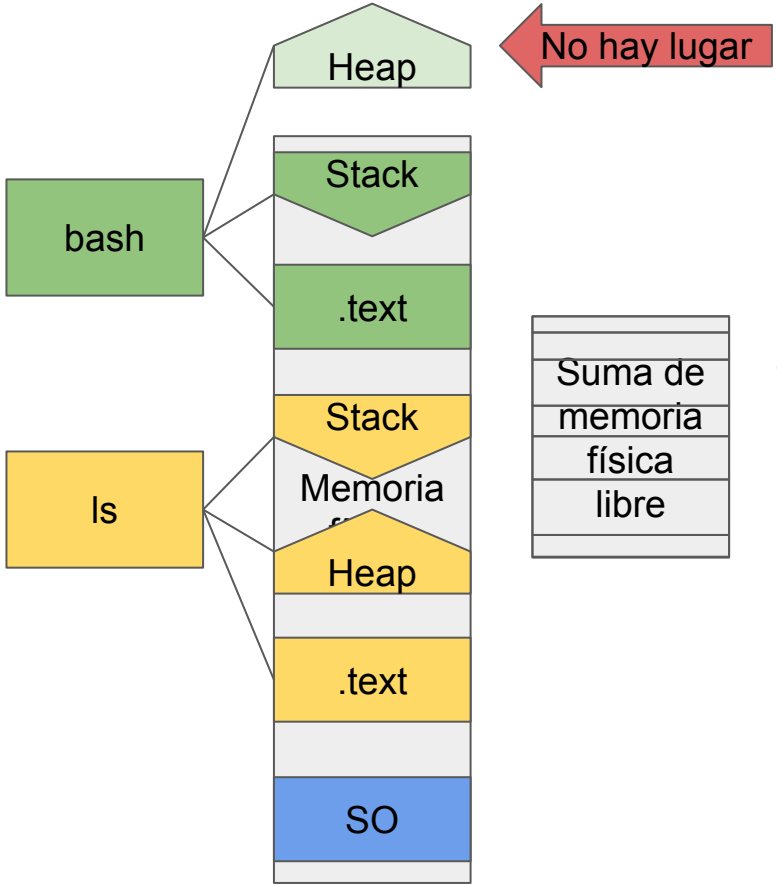
- `movabs $0x402000,%rsi`
- Es necesario conocer a priori la organización de la memoria

Traslademos esta situación a la siguiente remera

ITBA
Instituto Tecnológico de Buenos Aires

Problemas

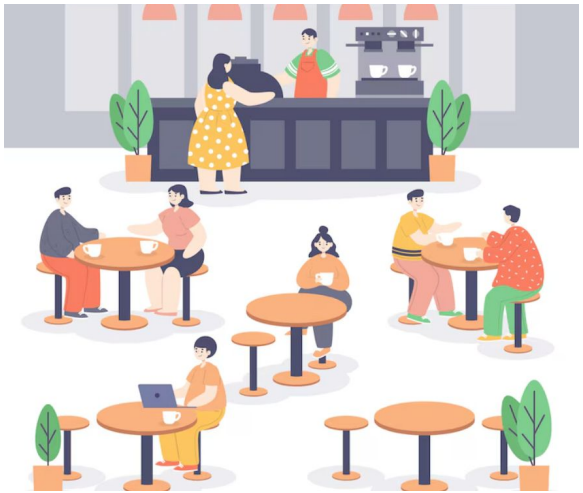
Cada proceso accede directo a memoria física



Es muy difícil impedir que la memoria se **fragmente**. También lo es permitir que los bloques **crezcan** dinámicamente

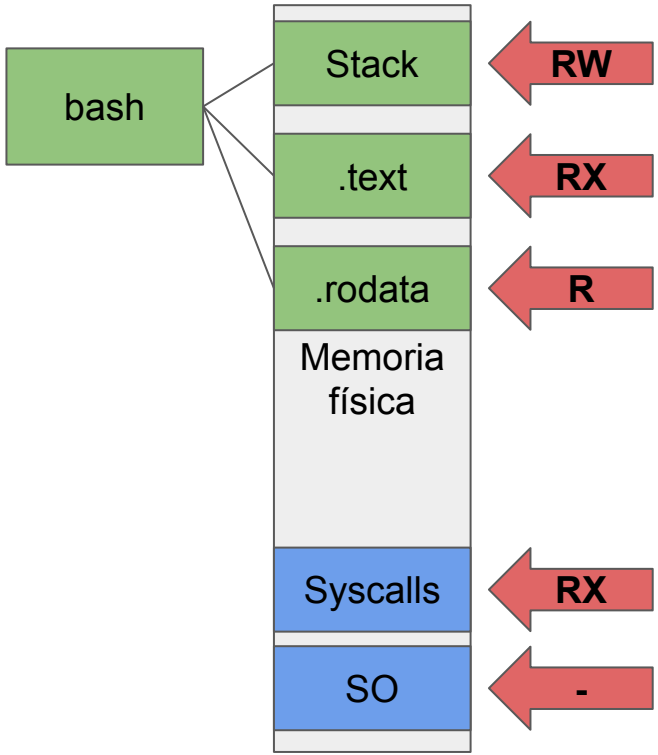
- Existe suficiente memoria libre pero no contigua
- Stack y Heap cambian de tamaño durante la ejecución

Traslademos esta situación a las sillas de un cafe



Problemas

Cada proceso accede directo a memoria física



Es muy difícil establecer permisos para diferentes áreas de la memoria

- Puedo leer **R**, escribir **W** y ejecutar **X** toda la memoria

Traslademos esta situación a los visitantes de un museo

- Piezas en exposición **R--**
- Libro de quejas **RW-**
- Baño **R-X**
- Depósito **-**

Basta de problemas

Ya nos quedó claro que esta instanciación no es muy útil:

- Abstracción: **Sin abstracción**
- Interposición: **Sin interposición**

Probemos con esta:

- Abstracción: **Memoria virtual**
- Interposición: **Completa (con ayuda del hardware)**

Idea central de **memoria virtual:**
Separar el **espacio de direcciones virtuales** (**EDV**) del **físico** (**EDF**)

Espacio de direcciones

Para entender **memoria virtual** necesitamos incorporar el concepto de **espacio de direcciones**. Para esto trasladamos la idea al sistema de telefonía:

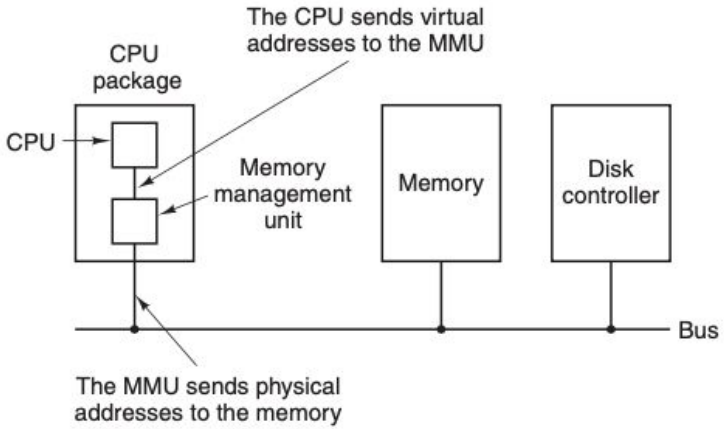
- Un usuario marca un número de teléfono
- La central telefónica, de forma transparente, puede
  - Conectarlo con un teléfono fijo / celular (incluso si se muda / mueve)
  - Hacerlo esperar si la línea está ocupada
  - Abortar la llamada si el número no corresponde a un abonado
  - Abortar la llamada si no hay saldo suficiente
  - Abortar la llamada si existe una restricción judicial que le impide comunicarse con ese número
  - Si marca 911, conectarlo con diferentes servicios de emergencia dependiendo de su ubicación geográfica
  - Incluso puede hacer que diferentes números correspondan al mismo teléfono

El conjunto de números que el usuario puede marcar es su **espacio de direcciones**.

- Único por usuario
- Se distingue del **espacio real / físico** de teléfonos

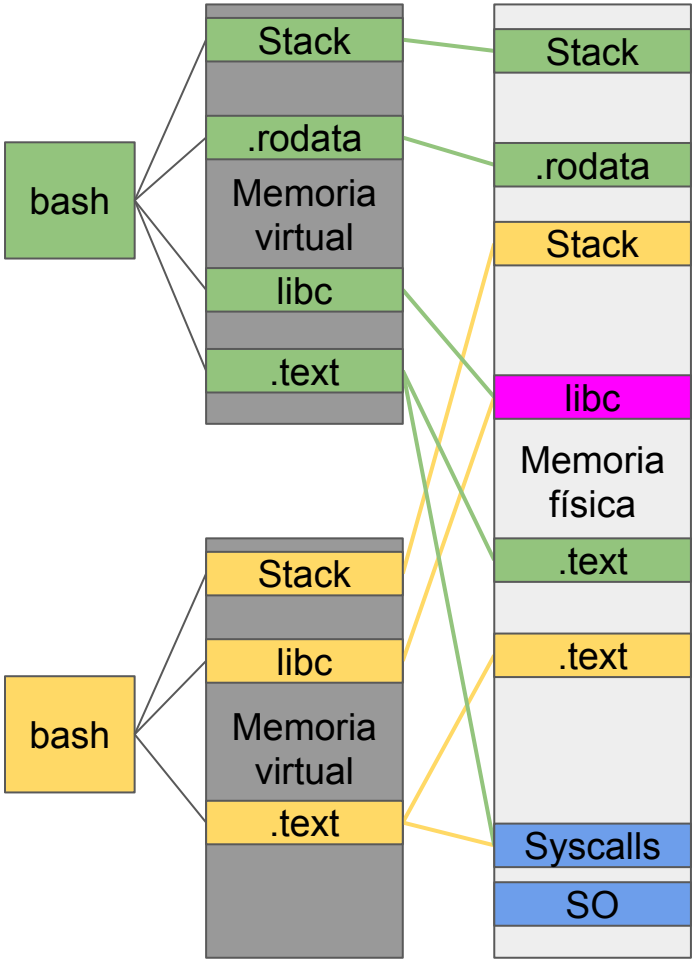
¿Cómo implementamos memoria virtual?

- El **EDV** se le presenta a cada proceso como un bloque **contiguo** y **exclusivo** de memoria.
- El **EDF** no es visible por los procesos
- El **SO** centraliza la administración el **EDF** y **mapea** el **EDV** de cada proceso al **EDF**.
- Este mapeo se puede interpretar como una **función matemática** que además verifica permisos (**RWX**) entre otros flags. Algunas **direcciones virtuales** (**DV**) pueden no estar mapeadas a ninguna **dirección física** (**DF**), tal como ocurre con las funciones.
- **Cada** acceso a memoria es supervisado por la **MMU** y ante cualquier problema se notifica al **SO**



Tanenbaum, Modern Operating Systems 4e, 2015 Prentice-Hall

¿Cómo se resuelven los problemas?



- El **SO** evita colisiones
- No se mapean áreas de memorias que no deben ser accedidas
- Una librería en el **EDF** puede estar mapeada en múltiples **EDV**
- El **SO** puede restringir los permisos de cada área de memoria mapeada

shared libraries

Julia Evans
@b0rk

Most programs on Linux use a bunch of C libraries

some popular libraries:

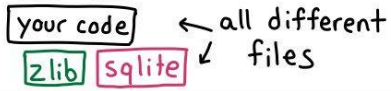
- openssl (for SSL!)
- sqlite (embedded db!)
- libpcre (regular expressions!)
- zlib (gzip!)
- libstdc++ (C++ standard library!)

There are 2 ways to use any library

① Link it into your binary

big binary with lots of things!

② Use separate shared libraries

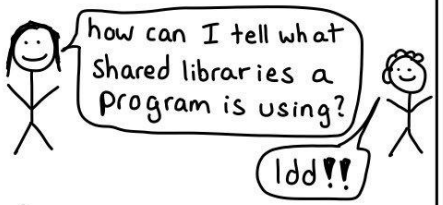


Programs like this:

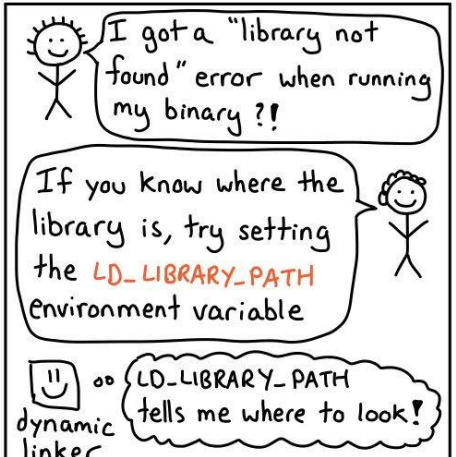
are called "statically linked"

and programs like this:

are called "dynamically linked"




```
$ ldd /usr/bin/curl
    libz.so.1 => /lib/x86-64..
    libresolv.so.2 => ...
    libc.so.6 => ...
    +34 more
```



Where the dynamic linker looks

① DT_RPATH in your executable

② LD_LIBRARY_PATH

③ DT_RUNPATH in executable

④ /etc/ld.so.cache
(run ldconfig -p to see contents)

⑤ /lib , /usr/lib

Julia Evans
@b0rk

mmap

drawings.jvns.ca

what's mmap for?

I want to work with a VERY LARGE FILE but it won't fit in memory

you could try mmap!

(mmap = "memory map")

load files lazily with mmap

When you mmap a file, it gets mapped into your program's memory

2TB file ← 2TB of virtual memory

but nothing is ACTUALLY read into RAM until you try to access the memory (how it works: page faults!)

how to mmap in Python

```
import mmap
f = open("HUGE.txt")
mm = mmap.mmap(f.fileno(), 0)
```

this won't read the file from disk! Finishes ~instantly.

```
print(mm[-1000:])
```

this will read only the last 1000 bytes!

sharing big files with mmap

we all want to read the same file!

no problem! mmap

Even if 10 processes mmap a file, it will only be read into memory ♥once♥

dynamic linking uses mmap

program: I need to use libc.so.6 (C standard library)

you too eh? no problem I always mmap, so that file is probably loaded into memory already

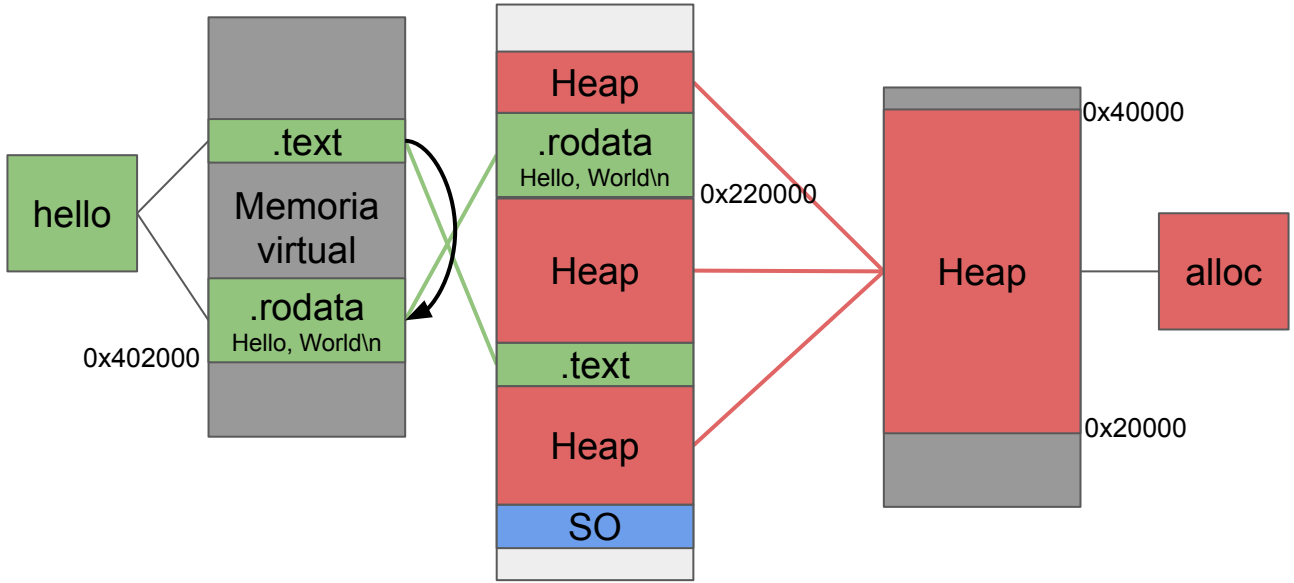
ld (dynamic linker)

anonymous memory maps

- not from a file (memory set to 0 by default)
- with MAP_SHARED, you can use them to share memory with a subprocess!

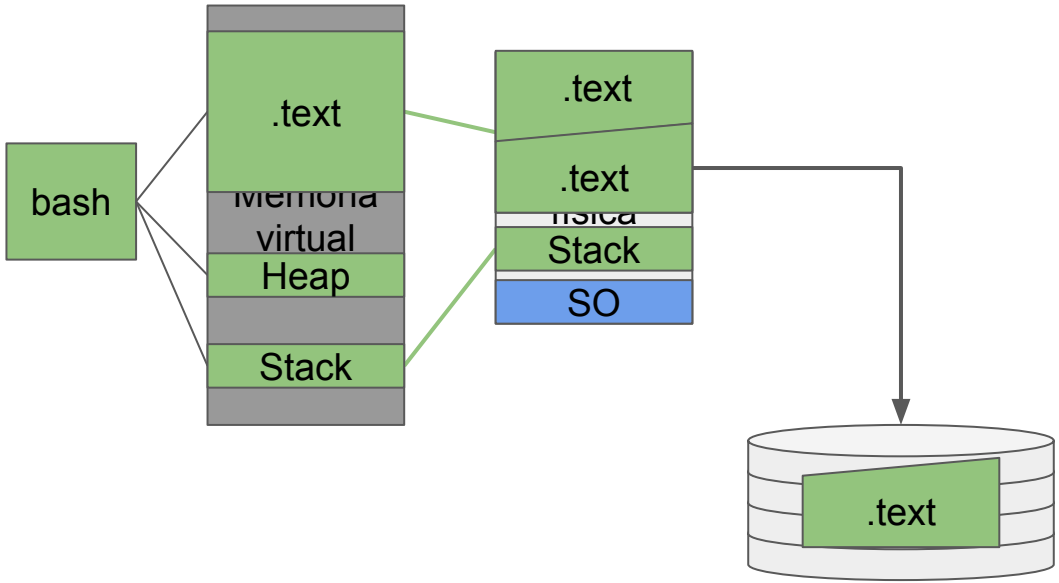
¿Cómo se resuelven los problemas?

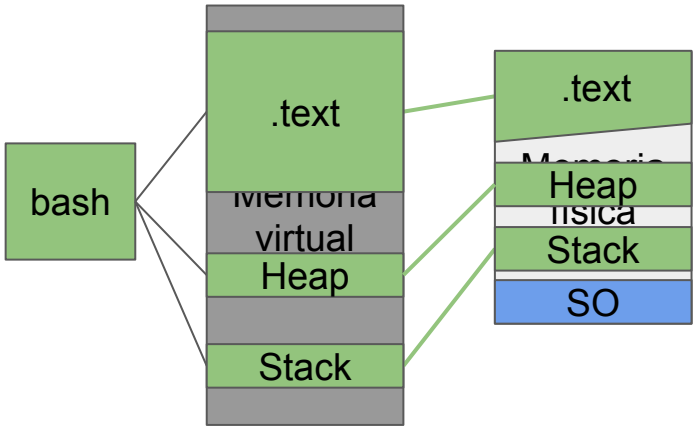
- El **SO** puede ubicar a cada proceso en cualquier posición del **EDF**, respetando la posición esperada en el **EDV**
- Áreas fragmentadas del **EDF** pueden ser mapeadas desde un área continua en el **EDV** al igual que áreas que crecen dinámicamente



¿Cómo se resuelven los problemas?

- De ser necesaria más memoria que la instalada, el **SO** puede hacer espacio moviendo al disco parte de la memoria que no se esté usando
- De la misma manera se puede ilustrar que no es necesario que todo el código esté en memoria simultáneamente para poder ejecutar

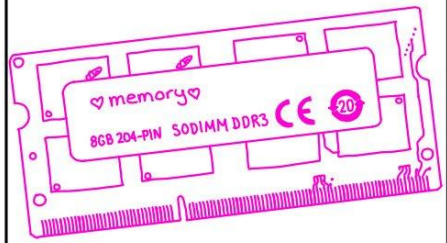




virtual memory

JULIA EVANS
@b0rk

your computer has physical memory



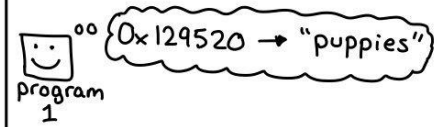
physical memory has addresses

0 - 8GB

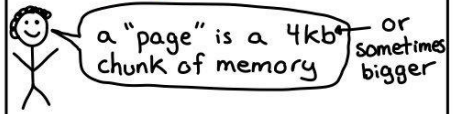
but when your program references an address like 0x5c69a2a2

that's not a physical memory address!
It's a virtual address

every program has its own virtual address space

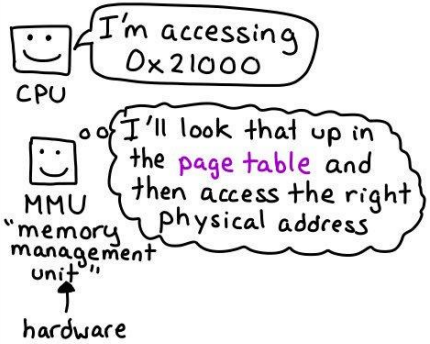


Linux keeps a mapping from virtual memory pages to physical memory pages called the "page table"



PID	virtual addr	physical addr
1971	0x20000	0x192000
2310	0x20000	0x228000
2310	0x21000	0x9788000

when your program accesses a virtual address



every time you switch which process is running, Linux needs to switch the page table

Paginación

Hasta ahora no hemos determinado ningún nivel de **granularidad** particular y solo hemos hablado de **áreas** de memoria.

Conceptualmente, podría ser a nivel de **bytes** o a nivel de **palabra**, es decir mapear cada byte o palabra por separado y cada una con sus permisos específicos. Este enfoque no **escala** además de ser **innecesario.**

Paginación consiste en agrupar bytes contiguos en bloques del mismo tamaño, por ejemplo 4KB, tanto en el **EDV** como en el **EDF**.

En el **EDV** estos bloques se denominan **páginas virtuales** y en el **EDF marcos de página** (**virtual page VP** / **page frame PF**)

Tabla de páginas

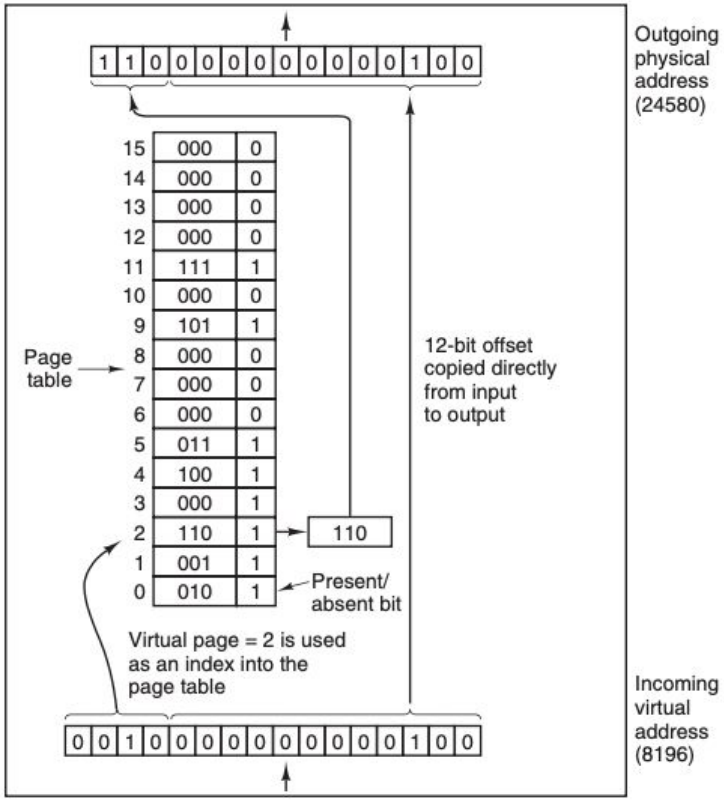


Figure 3-10. The internal operation of the MMU with 16 4-KB pages.

La **función matemática** que mapea **DV** a **DF** se define en la **tabla de páginas** y posee una entrada por cada **VP**. Es decir que mapea **VP** a **PF** en lugar de direcciones aisladas

La **DV** se divide en 2 partes

- Bits más significativos: número de **VP** (índice en la tabla)
- Bits menos significativos: **offset** dentro de la página

Ejemplo

Para una dirección de **16 bits** y páginas de **4KB**, los **4 bits** más significativos indican la **VP** y los **12 bits** restantes el **offset** dentro de la **VP**.

Tabla de páginas

Hagamos zoom en una **entrada** de la tabla de páginas

- **PF number**: ¡fundamental!
- **Present/absent**: ¿Está el **PF** presente en memoria física?
- **Protection**: ¿Qué acceso se permite? **RWX**
- **Modified**: ¿Fue modificada la página (**W**)?
- **Referenced**: ¿Fue referenciada la página (**R**/**W**)?

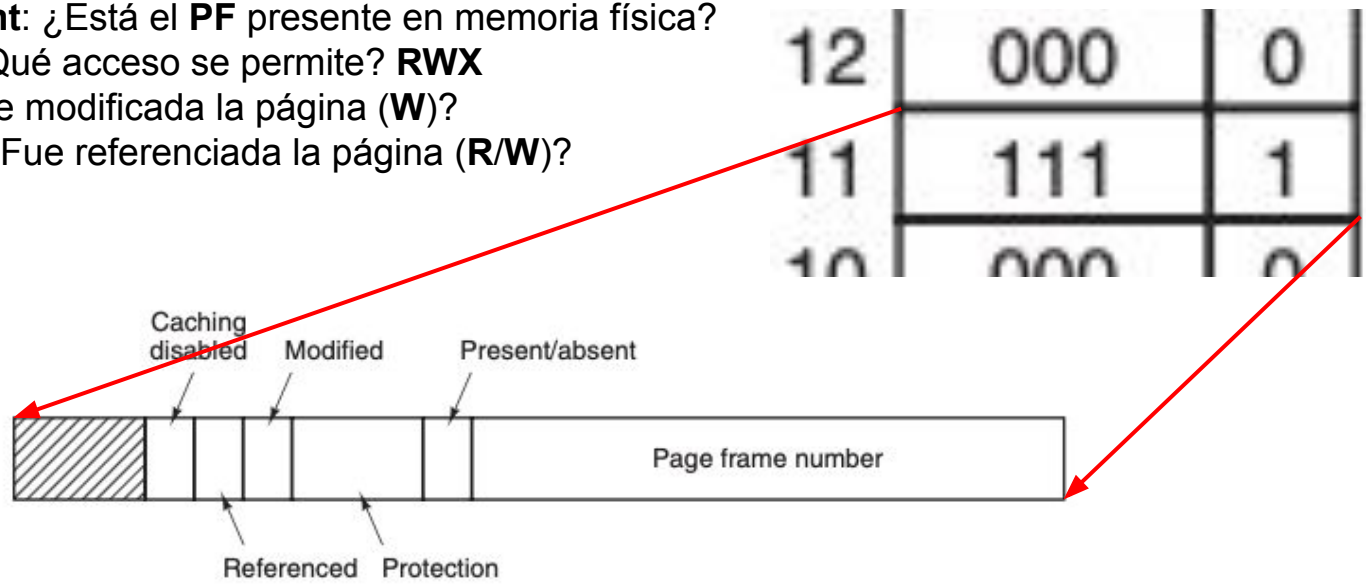


Figure 3-11. A typical page table entry.

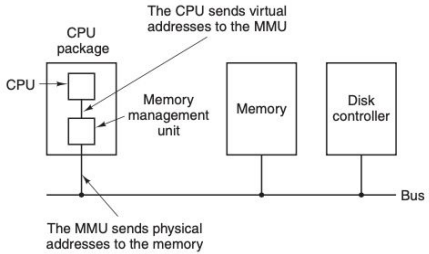
Page fault

¿Qué pasa si estos chequeos fallan?

- **PF number**: ¡fundamental!
- **Present/absent**: ¿Está el **PF** presente en memoria física?
- **Protection**: ¿Qué acceso se permite? **RWX**
- **Modified**: ¿Fue modificada la página (**W**)?
- **Referenced**: ¿Fue referenciada la página (**R**/**W**)?

La **MMU** lanza una **excepción antes** de que se acceda a la memoria y le pasa el control al **SO**

- Si el **PF** no está **presente**, se trae del disco (**hard**) o se mapea si ya está en memoria (**soft**)
- Si se violan los permisos, se **mata** al proceso. A menos que sea un caso de **Copy-on-Write**
- Si se accede a una **VP** que no está mapeada directamente, se **mata** al proceso. A menos que hablemos de **asignación lazy**



Si, siempre hay un **a menos que…**

page faults

Julia Evans
@b0rk

every Linux process has a page table

★ page table ★

virtual memory address	physical memory address
0x19723000	0x1422000
0x19724000	0x1423000
0x1524000	not in memory
0x1844000	0x4a000 read only

some pages are marked as either

★ read only

★ not resident in memory

when you try to access a page that's marked "not in memory", that triggers a ! page fault !

what happens during a page fault?

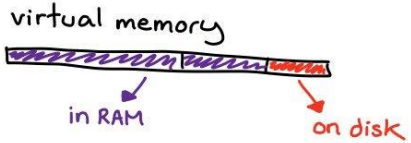
→ the MMU sends an interrupt

→ your program stops running

→ Linux kernel code to handle the page fault runs

Linux: I'll fix the problem and let your program keep running

"not in memory" usually means the data is on disk!



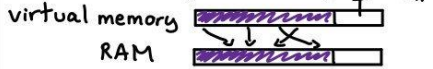
Having some virtual memory that is actually on disk is how swap and mmap work

how swap works

① run out of RAM

② Linux saves some RAM data to disk

③ mark those pages as "not resident in memory" in the page table



④ When a program tries to access the memory there's a ! page fault !

⑤ Linux: time to move some data back to RAM!

⑥ if this happens a lot your program gets VERY SLOW

I'm always waiting for data to be moved in & out of RAM

No todo es color de rosa

- ¿Cómo administramos la memoria libre?
- La traducción implica un costo. **Cada** acceso a memoria debe traducirse.
- Si el **EDV** es grande, la **tabla de páginas** será grande.
- Antes dijiste *“el **SO** puede hacer espacio moviendo al disco parte de la memoria que no se esté usando”*. ¿Con qué criterio se determina eso?

Administrar la memoria libre

2 enfoques:

- **Bitmap**: Se divide la memoria en bloques y cada bit representa su estado -> ¿El tamaño importa?
- **Free list**: Cada proceso y espacio libre tiene un nodo con inicio, longitud y el puntero al siguiente

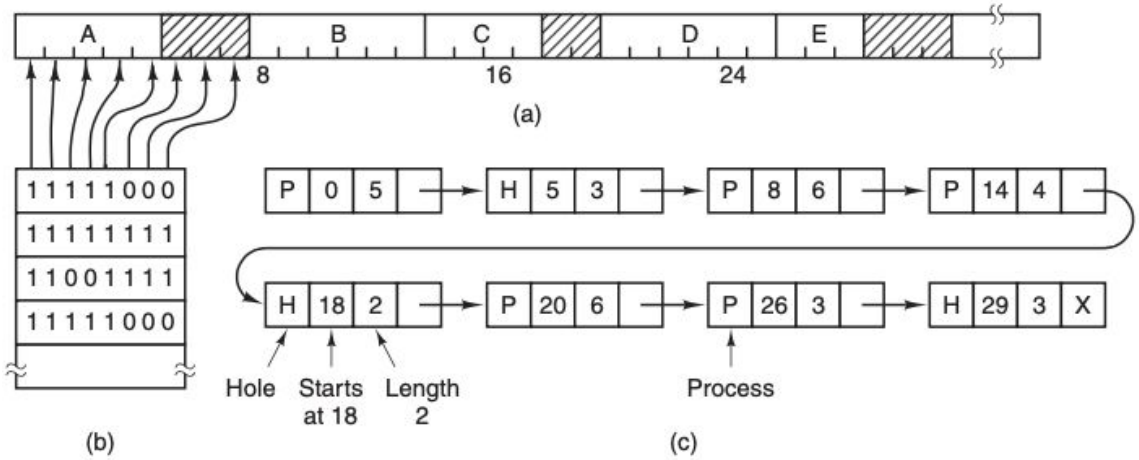


Figure 3-6. (a) A part of memory with five processes and three holes. The tick marks show the memory allocation units. The shaded regions (0 in the bitmap) are free. (b) The corresponding bitmap. (c) The same information as a list.

Administrar la memoria libre

Free list

- Es necesario fusionar bloques libres adyacentes
- Doblemente encadenada facilita este trabajo
- El orden en la lista es relevante
  - Para fusionar bloques libres adyacentes
  - Para buscar bloques libres

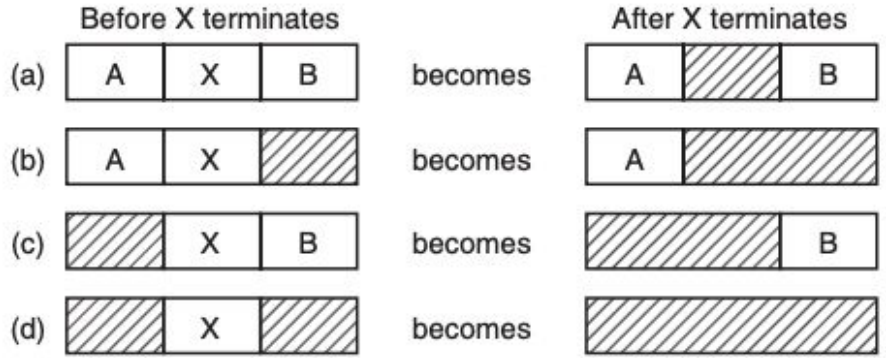


Figure 3-7. Four neighbor combinations for the terminating process, *X*.

Administrar la memoria libre

Buscar un bloque libre, luego se parte en 2, lo necesario y lo que sobra (si sobra)

- **First fit**: Comenzando desde el principio, el primer bloque suficientemente grande
- **Next fit**: Comenzando desde donde quedamos, el primer bloque suficientemente grande
- **Best fit**: Recorre toda la lista y elige el bloque más pequeño, pero suficientemente grande.

¿Velocidad?
¿Fragmentación?

- **Worst fit**: Recorre toda la lista y elige el bloque más grande

Administrar la memoria libre

¡Momento! ¿Por qué se llama free list si la lista contiene espacios libres y ocupados?

- Se pueden separar las listas entre ocupados y libres
- Liberar y asignar un bloque consiste en moverlo de listas
- Puedo ordenar la lista de bloques libres por tamaño (de menor a mayor)
- No necesito una estructura auxiliar, puedo usar la misma memoria libre

- **First fit** y **Best fit**: Se vuelven equivalentes
- **Next fit**: No tiene sentido

¿Y si clasificamos los bloques libres por tamaño frecuentemente solicitado?

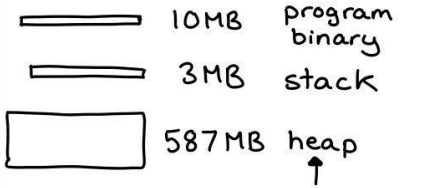
- 4 free lists de bloques de 4KB, 8KB, 12KB y 16KB respectivamente
- **quick fit**: El primer bloque de la lista correspondiente

¿Qué pasa cuando un proceso termina? ¿Se unen bloques libres adyacentes? ¿Qué pasa si no?

memory allocation

JULIA EVANS
@b0rk

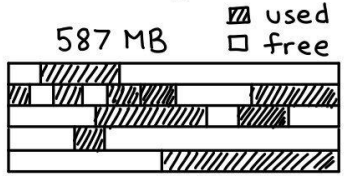
your program has memory



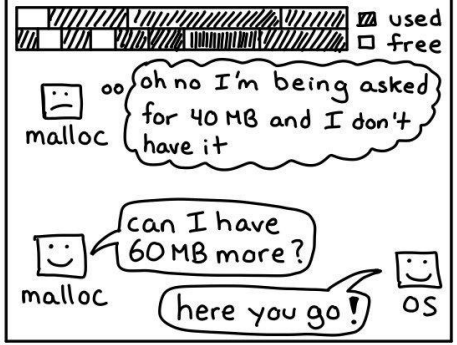
the heap is what your allocator manages

your memory allocator (malloc) is responsible for 2 things.

THING 1: keep track of what memory is used/free



THING 2: Ask the OS for more memory!



your memory allocator's interface

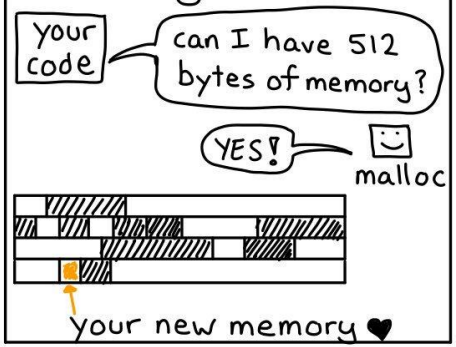
malloc(size_t size)
allocate size bytes of memory & return a pointer to it

free(void* pointer)
mark the memory as unused (and maybe give back to the OS)

realloc(void* pointer, size_t size)
ask for more/less memory for pointer

calloc(size_t members, size_t size)
allocate array + initialize to 0

malloc tries to fill in unused space when you ask for memory



malloc isn't magic

it's just a function!

you can always:

→ use a different malloc library like jemalloc or tcmalloc (easy!)

→ implement your own malloc (harder)

Análisis de costos

2 observaciones

- La traducción de **DV** a **DF** debe ser **rápida**
  - **Cada** acceso a memoria está sujeto a traducción
  - La ejecución de cualquier instrucción requiere acceder a memoria (para traer la instrucción)
  - Algunas instrucciones referencian memoria -> más accesos a memoria
  - Algunas referencias a memoria pueden cruzar las fronteras de una **VP**
- Si el **EDV** es grande, la tabla de páginas será grande
  - Con **DV** de 32 bits y páginas de 4KB, ¿cuántas entradas tendrá la tabla? ¿Cuánto ocupa la tabla? ¿Cuántas tablas necesitamos?

¿Y si agregamos registros al hardware para almacenar toda la tabla y cargamos la tabla en cada context switch?

La tabla de páginas reside en memoria

Translation Lookaside buffer

¿Qué es más cierto?

1. Los procesos hacen muchas referencias a pocas páginas
2. Los procesos hacen pocas referencias a muchas páginas

- Equipar los dispositivos con hardware que mapea **DV** a **DF**: Translation lookaside buffer (**TLB**)
- Usualmente dentro de la **MMU**
- No más de **256** entradas

- ¿A qué se parecen las entradas?

- En este ejemplo un proceso en un loop que usa las páginas 19, 20 y 21, procesa un arreglo en las páginas 129 y 130. La página 140 contiene los índices y el stack está en las páginas 860 y 861

Valid	Virtual page	Modified	Protection	Page frame
1	140	1	RW	31
1	20	0	R X	38
1	130	1	RW	29
1	129	1	RW	62
1	19	0	R X	50
1	21	0	R X	45
1	860	1	RW	14
1	861	1	RW	75

Figure 3-12. A TLB to speed up paging.

Translation Lookaside buffer

Al llegar una **DV** se compara paralelamente con todas las entradas del **TLB** (hardware especial)

- Si la encuentra y no viola la protección:

- Si viola la protección -> page fault
- Si no la encuentra (**miss**) se recorre la tabla de páginas (**memoria**) y se reemplaza con alguna de las entradas de la **TLB**
- ¿Qué hacemos con la entrada de la **TLB** que desalojamos?

Valid	Virtual page	Modified	Protection	Page frame
1	140	1	RW	31
1	20	0	R X	38

EDV grande

El **TLB** nos permite acelerar la traducción de **DV** a **DF** comparado a tener la tabla de páginas únicamente en memoria, pero ¿qué pasa si el **EDV** es muy grande?

2 enfoques:

- Tablas de páginas multinivel
- Tablas de páginas invertidas

Tabla de páginas multinivel

¿Cuánto ocuparía la tabla entera en memoria (sin multinivel)?

- Cada entrada de la tabla 32 bits
- **EDV** de 32 bits
- Páginas de 4KB

La idea de tablas multinivel es evitar tener toda la tabla en memoria, solo tenemos lo necesario:

Ejemplo:

- Un proceso necesita 12MB
  - 4MB .stack (top)
  - VACÍO (mucho)
  - 4MB .data (bottom)
  - 4MB .text (bottom)
- ¿Cómo se resuelve la dirección 0x00403004?

  PT1 PT2 Offset

  0000 0000 0100 0000 0011 0000 0000 0100
- ¿Cuántas ocupa la tabla en memoria (con multinivel)?

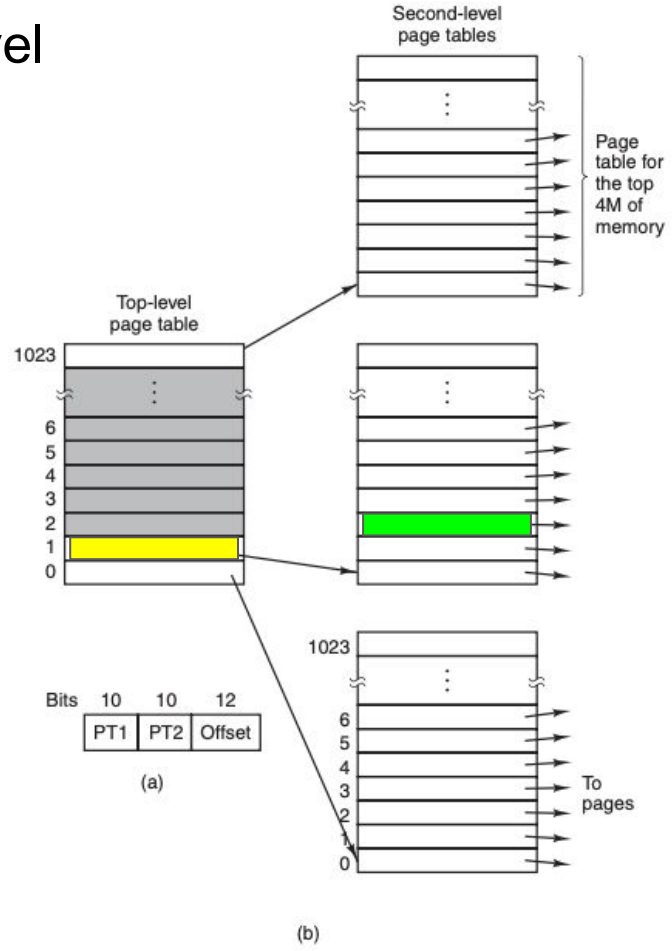


Figure 3-13. (a) A 32-bit address with two page table fields. (b) Two-level page tables.

Tabla de páginas multinivel

- El esquema $2^{10} \times 2^{10} \times 2^{12} = 2^{32}$ visto fue usado por el procesador 80386 de Intel (lanzado en 1985), pero otros esquemas son posibles (**page directory** - **page table**).
- Diez años después, el **Pentium Pro** incorpora un **tercer nivel** (**Page Directory Pointer Table**) y extiende a **64 bits** las entradas de las tablas. En este caso, el esquema era $2^9 \times 2^9 \times 2^2 \times 2^{12} = 2^{32}$. ¿Cuánta memoria puede direccionar?
- Cuando se incorpora el soporte completo a **64 bits**, se agrega un **cuarto nivel**. Sorpresivamente, no se llama *Page Directory Pointer Table Pointer*, sino **Page Map Level 4 (PML4)**. El esquema pasa a ser: $2^9 \times 2^9 \times 2^9 \times 2^9 \times 2^{12} = 2^{48}$. ¿Cuánta memoria puede direccionar?
- A partir de **2017**, Intel incorpora un **quinto nivel** de paginación en algunos procesadores (como los Ice Lake), llamado **Page Map Level 5 (PML5)**. Se utiliza el esquema: $2^9 \times 2^9 \times 2^9 \times 2^9 \times 2^9 \times 2^{12} = 2^{57}$

En todos estos casos, el **EDF** direccionable suele ser **menor** que el **EDV**. Por ejemplo, con **PML5**, las direcciones virtuales pueden alcanzar los 57 bits, pero la mayoría de los procesadores actuales solo permiten **52 bits físicos**, lo que equivale a **4 PiB** de RAM real direccionable.

Tabla de páginas invertidas

- En este esquema hay una entrada por cada PF en lugar de cada VP (virtual page)

Figure 3-14. (a) A 32-bit address with two page table fields. (b) Two-level page tables.

Tabla de páginas invertidas

- En este esquema hay una entrada por cada **PF** en lugar de cada **VP**

Ejemplo:

- Con un **EDV** de 64 bits, páginas de 4KB y 4 GB de memoria física se necesitan solo 2^{20} entradas en la tabla, una por cada **PF**.
- Cada entrada registra qué proceso y **VP** está ubicado en el **PF**.

- Se ahorra mucho espacio cuando el **EDV** es mucho más grande que el **EDF**, pero…
- La traducción de **DV** a **DF** es más compleja: ¿Qué pasa si el proceso **n** referencia la **VP p**? -> **p** ya no sirve como índice en la tabla y es necesario buscar la tabla entera en busca de la entrada (**n, p**) en **cada** referencia a memoria.

- Es común en arquitecturas con 64 bits, incluso con páginas muy grandes la tabla sería enorme. ¿Cuántas páginas de 4MB necesito para cubrir el **EDV** de 64 bits?

Tabla de páginas invertidas

- Podemos usar el **TLB** pero un miss requiere recorrer la tabla completa en memoria (software)
- Para agilizar esta búsqueda se puede tener una tabla de **VP** hasheadas y todas las **VP** con el mismo hash se ubican en una lista
- Al encontrar el par (**VP**, **PF**) se carga en el **TLB**

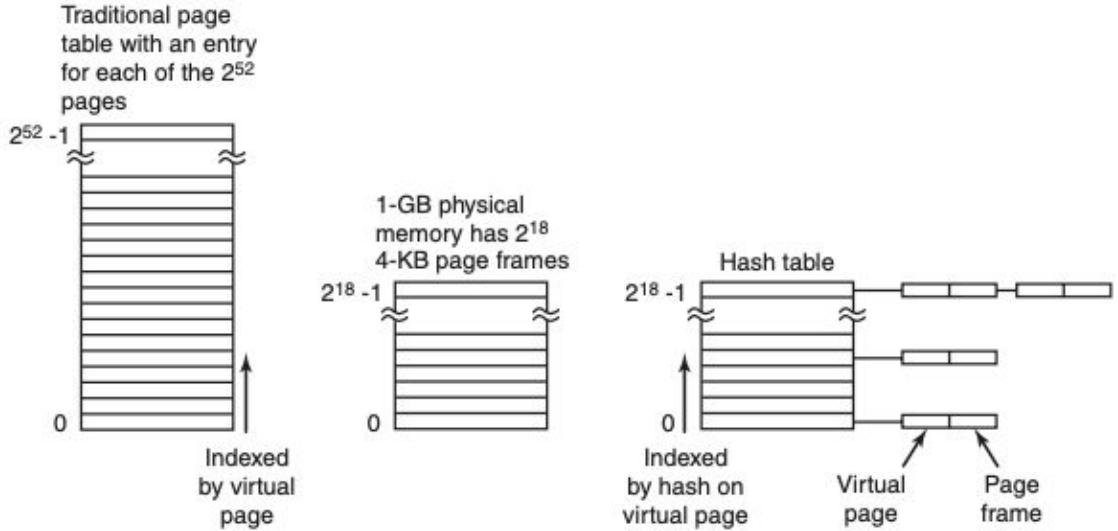


Figure 3-14. Comparison of a traditional page table with an inverted page table.

Algoritmos de reemplazo de páginas

Con paginación podemos darnos el lujo de no tener todo el proceso en memoria para que funcione y cuando hace falta un **PF** que no está presente se trae del disco a la memoria, pero para esto hay que hacer espacio (desalojar un **PF**), la pregunta de rigor es la siguiente:

¿Qué **PF** desalojar?

Algoritmos de reemplazo de páginas

Óptimo

- Fácil de describir
- Imposible de implementar

Clasificar las páginas de acuerdo a la cantidad de instrucciones que se ejecutarán hasta que sean referenciadas

Ejemplo

- 2 páginas serán referenciadas en 1500 y 2000 instrucciones respectivamente
- ¿Cuál conviene desalojar?

¿Para qué nos sirve este algoritmo?
¿Cómo podríamos evaluarlo?

Algoritmos de reemplazo de páginas

Not recently used (NRU)

- Usa los bits **R** y **M** (se deben actualizar en cada referencia ¿quién lo hace?)
- Se puede simular si no se dispone de los bits ¿Cómo?

- Inicialmente ambos bits en 0
- Periódicamente se resetea **R** ¿Se puede resetear **M**? ¿Qué fuente de periodicidad posee el kernel?
- Al momento de elegir una página para desalojar se clasifican en función de **R** y **M**

Clase 0: ¬**R** & ¬**M**
Clase 1: ¬**R** & **M** —> ¿Es posible esta clase?
Clase 2: **R** & ¬**M**
Clase 3: **R** & **M**

¿Cómo podemos usar esta información para elegir una página?
¿Por qué elegimos la clase 1 por sobre la 2 si la primera fue modificada?

- Simple, moderadamente eficiente y rendimiento razonable aunque no óptimo

Algoritmos de reemplazo de páginas

First-in first-out (FIFO)

- Se registra el tiempo en el que cada página se cargó en memoria
- Simplemente desalojamos la página más antigua

Análisis

- Analogía supermercado

Algoritmos de reemplazo de páginas

Second chance (SC)

- Modificación simple de FIFO
- Evita desalojar una página antigua pero muy usada ¿ejemplos?
- Usa el bit **R**

- Periódicamente se resetea **R**
- Si la página candidata (según FIFO) es **R**, se resetea **R** y se envía al comienzo de la cola como si recién llegara
- Si la página candidata (según FIFO) es ¬**R** se desaloja

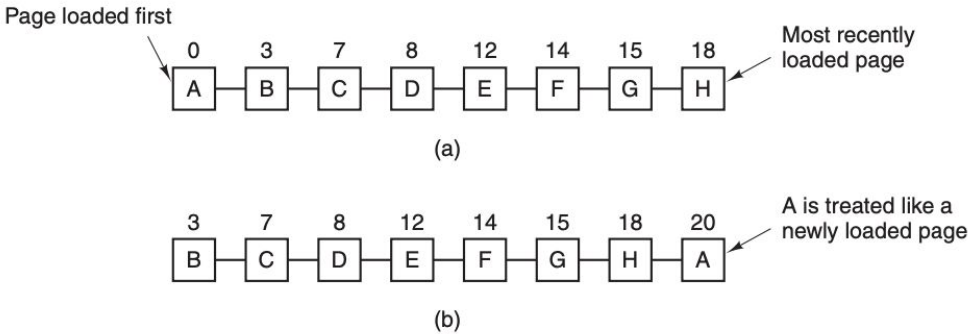


Figure 3-15. Operation of second chance. (a) Pages sorted in FIFO order. (b) Page list if a page fault occurs at time 20 and *A* has its *R* bit set. The numbers above the pages are their load times.

¿Qué pasa si todas las páginas fueron referenciadas?

Algoritmos de reemplazo de páginas

Clock (C)

- Second chance es ineficiente modificando la lista constantemente
- Mantiene las páginas en una lista circular
- La “manecilla” apunta a la página más vieja

- Si la página candidata es **R**, se resetea **R** y se avanza
- Si la página candidata es ¬**R** se desaloja

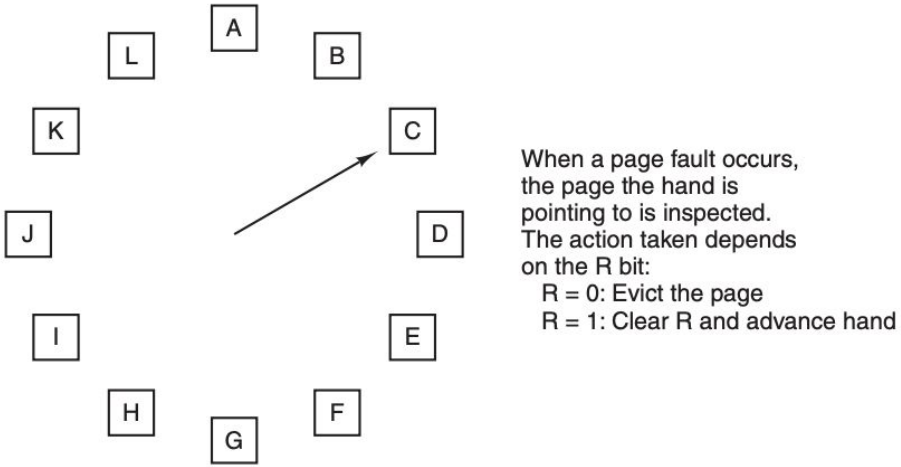


Figure 3-16. The clock page replacement algorithm.

Algoritmos de reemplazo de páginas

Least recently used (LRU)

Observación

- Las páginas muy usadas recientemente suelen seguir siendo muy usadas
- Las páginas escasamente usadas recientemente suelen seguir sin ser muy usadas

- Desalojar a la página que no ha sido usada el mayor tiempo

2 enfoques

- Tener una lista de páginas ordenada por tiempo de acceso. Desalojar a la más antigua
- Tener un registro especial que cuente instrucciones y equipar a cada entrada de la tabla con espacio para este registro. Cuando se referencia una página se actualiza su contador. Desalojar a la que tenga el contador más chico

Análisis

Algoritmos de reemplazo de páginas

Not frequently used (NFU)

- Requerimos de un contador (software) para cada página
- Periódicamente su suma el bit **R** de cada página a su contador

- Se desaloja a la página con el contador más chico

Análisis

Algoritmos de reemplazo de páginas

Simular LRU en software

- Pequeña modificación de NFU
- Los contadores se shiftean a la derecha 1 bit (Regla mnemotécnica: shiftear = empujar)
- Luego se suma el bit **R**, pero a la izquierda

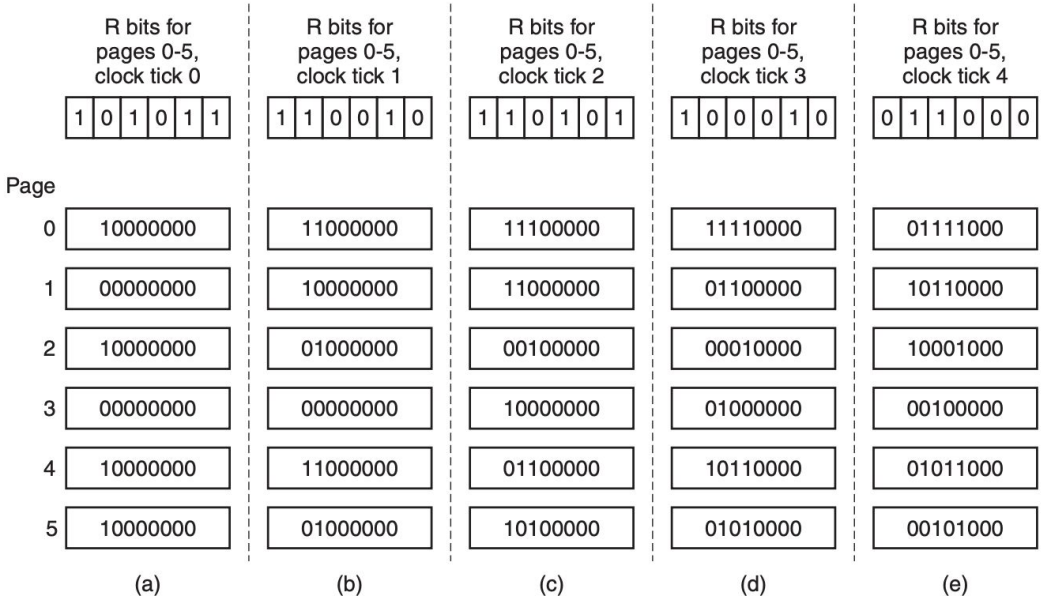


Figure 3-17. The aging algorithm simulates LRU in software. Shown are six pages for five clock ticks. The five clock ticks are represented by (a) to (e).

Se desaloja la página con el contador más chico

¿Qué pinta tendrá el contador de una página no referenciada durante los últimos 4 ticks?

¿Diferencias con LRU?

- Granularidad
- Pasado lejano. Análisis para 8 bits y 20ms

¿Cómo creen que se llama esta técnica?

Algoritmos de reemplazo de páginas

¿Cuántas páginas tiene en memoria un proceso justo antes de comenzar a ejecutar?

Demand paging

Durante cada fase de ejecución, los procesos suelen referenciar una pequeña fracción de sus páginas

Locality of reference

El conjunto de páginas siendo usado actualmente se denomina

Working set

¿Qué pasa si todo el working set está en memoria? ¿Y si no hay memoria suficiente para almacenarlo?

Trashing

En sistemas multiprogramados se suelen bajar a disco las páginas de un proceso para que pueda correr otro proceso. Al volver a ejecutar podemos subirlas todas de vuelta bajo demanda ¿Costo?

Podemos subirlas a todas antes de que comience a ejecutar

Working set model - prepaging

Algoritmos de reemplazo de páginas

Working set (WS)

- En cualquier instante de tiempo **t** existe el conjunto de páginas referenciadas en las últimas **k** referencias a memoria, este conjunto se expresa como **w(k ,t)**
- Esta función es monótona creciente

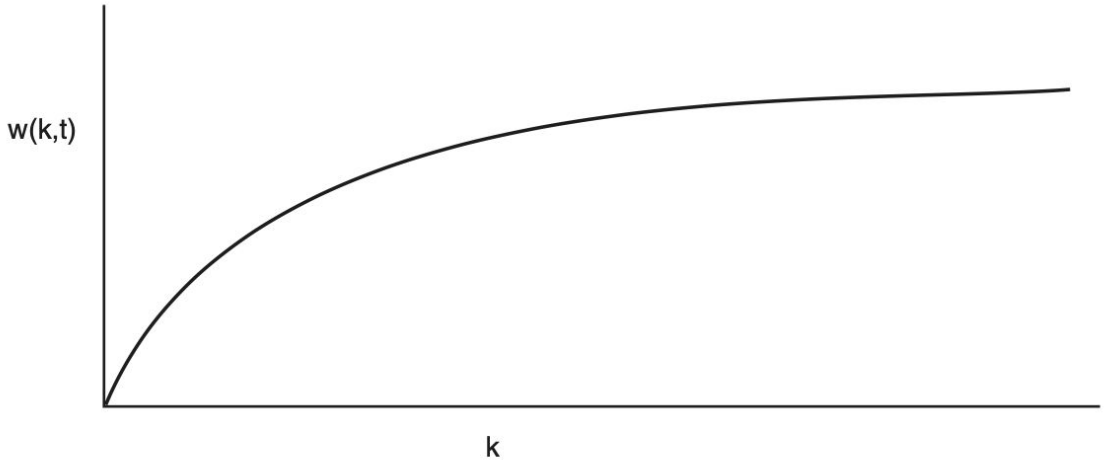


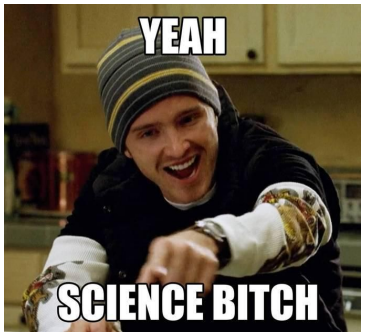
Figure 3-18. The working set is the set of pages used by the k most recent memory references. The function $w(k, t)$ is the size of the working set at time t.

This set, $w(k, t)$, is the working set.

The function $w(k, t)$ is the size of the working set at time t.

Podemos usar cardinalidad

$|w(k,t)|$



Algoritmos de reemplazo de páginas

Working set (WS)

¿Cómo podemos aprovechar el working set para implementar un algoritmo de reemplazo de páginas?

Una vez establecido un k específico, por ejemplo, 10 millones de referencias a memoria el working set queda perfectamente determinado
Luego podemos registrar cada número de página en una lista de tamaño **k**, actualizarla en cada referencia a memoria, eliminar repetidos y demás tareas de mantenimiento ¿Costo?

En lugar de esto se utilizan aproximaciones. Una de ellas consiste en reemplazar las páginas referenciadas en las últimas **k** referencias a memoria por las páginas referenciadas en las últimas **t** unidades de tiempo.

Current virtual time vs real time

Algoritmos de reemplazo de páginas

Working set (WS)

- Periódicamente se resetea **R**
- Se asume que durante **t** unidades de tiempo ocurren muchos ticks

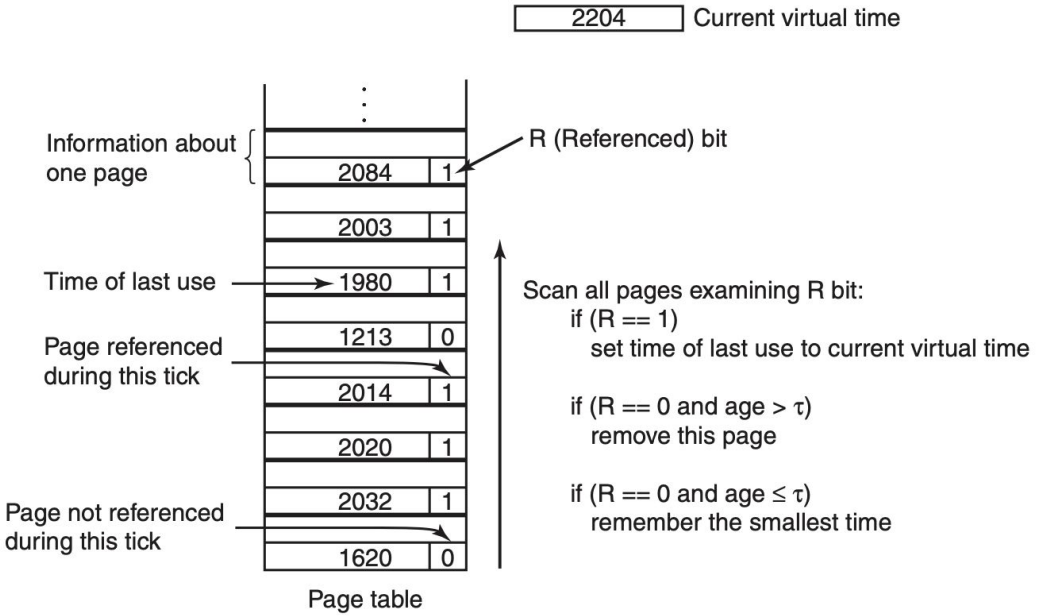


Figure 3-19. The working set algorithm.

Orden:

1. ¬**R** & age > **t** (∉ w(t))
2. ¬**R** & age <= **t** (saco la más vieja)
3. **R** (random, preferentemente ¬**W**)

Algoritmos de reemplazo de páginas

WSClock (WSC)

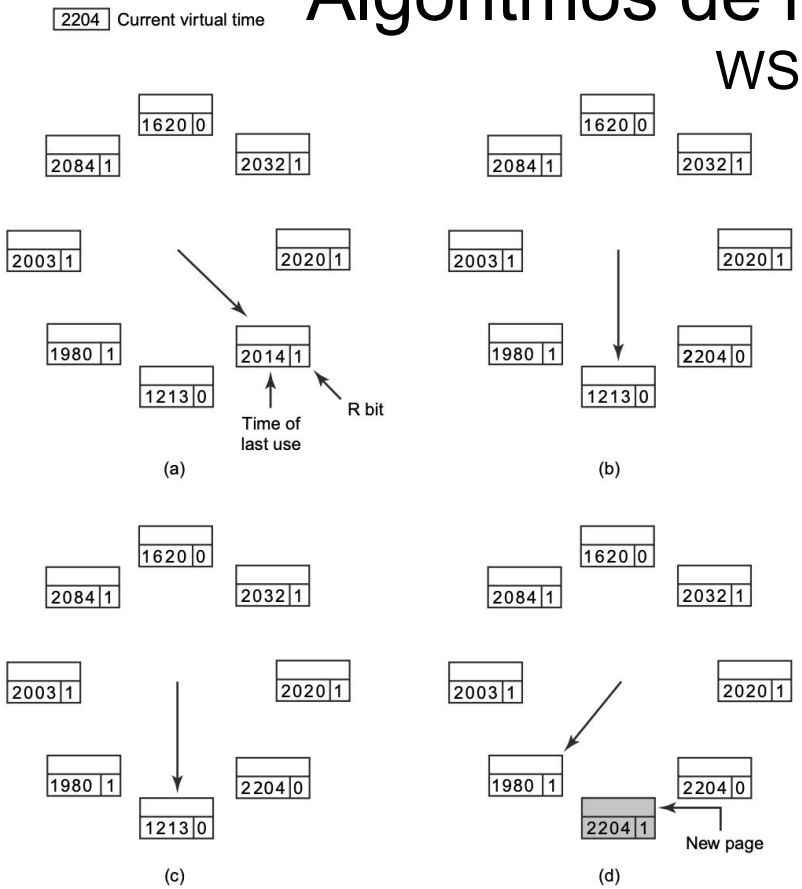


Figure 3-20. Operation of the WSClock algorithm. (a) and (b) give an example of what happens when $R = 1$. (c) and (d) give an example of $R = 0$.

Recorrer toda la tabla en cada page fault hace ineficiente el algoritmo WS.

¿Qué 2 algoritmos combina WSC?

- Si **R** se resetea **R,** se actualiza el tiempo y se avanza.
- Si ¬**R** & age > **t** & ¬**M** se desaloja sin actualizar el disco
- Si ¬**R** & age > **t** & **M** se planifica (con un límite) el guardado a disco y se avanza la manecilla
- Si no se encuentra ninguna hay 2 casos
  - Se planificó alguna escritura y se continúa girando hasta que aparezca ¬**M**.
  - No se planificaron escrituras y se desaloja alguna del working set, de nuevo, la más vieja y preferentemente ¬**M**

Por su simplicidad y eficiencia es muy utilizado en la práctica

ITBA Instituto Tecnológico de Buenos Aires

Algoritmos de reemplazo de páginas

Resumen

Algorithm	Comment
Optimal	Not implementable, but useful as a benchmark
NRU (Not Recently Used)	Very crude approximation of LRU
FIFO (First-In, First-Out)	Might throw out important pages
Second chance	Big improvement over FIFO
Clock	Realistic
LRU (Least Recently Used)	Excellent, but difficult to implement exactly
NFU (Not Frequently Used)	Fairly crude approximation to LRU
Aging	Efficient algorithm that approximates LRU well
Working set	Somewhat expensive to implement
WSClock	Good efficient algorithm

Figure 3-21. Page replacement algorithms discussed in the text.

Ejercicio 1

Para cada una de las siguientes direcciones físicas en base 10, calcule el PF y el offset considerando un tamaño de página de 4 y 8 KB

- 20000
- 32768
- 60000

Ejercicio 2

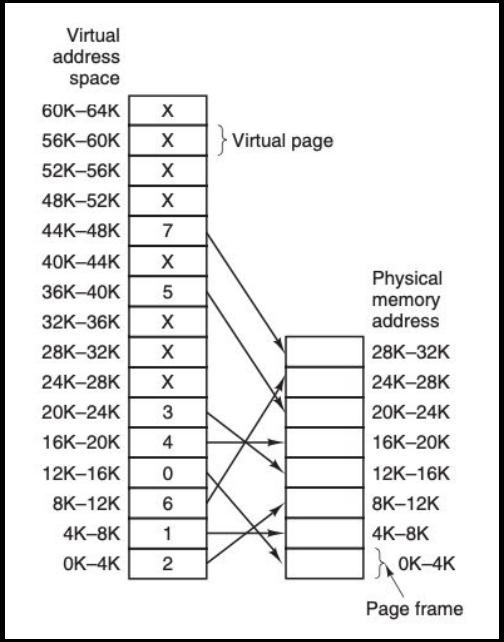
Considere un sistema de swapping en el cual la memoria posee los siguientes espacios libres (en orden): 10, 4, 20, 18, 7, 9, 12 y 15 MB. ¿Qué espacio libre será ocupado por 3 solicitudes consecutivas de 12, 10 y 9 MB para cada una de los siguientes algoritmos?

- First fit
- Best fit
- Worst fit
- Next fit

Ejercicio 3

Usando la tabla de páginas de la imagen, calcule la DF para cada una de las siguientes DVs

- 20
- 4100
- 8300
- 25345



Ejercicio 4

Considere un sistema que tiene un EDV de 48 bits y un EDF de 32 bits

- Si las páginas son de 4KB
  - ¿Cuántas entradas tendrá la tabla de páginas si tiene 1 solo nivel?
  - ¿Cuánto ocupa esta tabla considerando entradas de 32 bits?
- Considerando que este sistema tiene una TLB de 32 entradas y ejecuta un programa cuyas instrucciones caben en 1 página y lee secuencialmente un arreglo de enteros que abarca miles de páginas, ¿Qué tan efectiva será la TLB para este caso?

Ejercicio 5

Usando el algoritmo de reemplazo de páginas FIFO en un sistema con 4 PF y 8 VP, ¿Cuántos page faults ocurrirán para la siguiente secuencia de accesos a páginas: 0172327103 si los 4 PF están inicialmente vacíos?
¿Y para LRU?

Ejercicio 6

Considere el algoritmo de reemplazo de páginas WSClock con **t** = 2 ticks y el siguiente estado del sistema

Page	Time stamp	V	R	M
0	6	1	0	1
1	9	1	1	0
2	9	1	1	1
3	7	1	0	0
4	4	0	0	0

V: Valid
R: Referenced
M: Modified

- Si ocurre una interrupción del timer en el tick = 10, ¿Cómo quedaría la tabla?
- Suponga que ocurre un page fault en lugar de una interrupción del timer, debido a una solicitud de lectura de la página 4, ¿Cómo quedaría la tabla?

Ejercicio 7

Considere un sistema con 4 PF. El tiempo de carga, acceso y los bits **R** y **M** para cada página se presentan en la siguiente tabla (los tiempos están expresados en ticks del timer)

Page	Loaded	Last ref.	R	M
0	126	280	1	0
1	230	265	0	1
2	140	270	0	0
3	110	285	1	1

¿Qué página reemplazará cada uno de los siguientes algoritmos?

- NRU
- FIFO
- LRU
- SC

Ejercicio 8

Considere la siguiente matriz bidimensional

```
int X[64][64]
```

Suponga que un sistema tiene 4 PFs y cada uno tiene 128 palabras (un número entero ocupa una palabra). Los programas que manipulan la matriz X caben exactamente en una página y siempre ocupan la página 0. Los datos se intercambian dentro y fuera de los otros 3 PFs. La matriz X se almacena de manera tal que X[0][1] sigue a X[0][0] en memoria. ¿Cuál de los dos fragmentos de código que se muestran a continuación generará la menor cantidad de fallas de página? Explique y calcule el número total de faltas de página.

```
//Fragmento A
for (int j = 0; j < 64; j++)
  for (int i = 0; i < 64; i++)
    X[i][j] = 0;

//Fragmento B
for (int i = 0; i < 64; i++)
  for (int j = 0; j < 64; j++)
    X[i][j] = 0;
```

Glosario

- Swapping
- Relocation static - dynamic
- Espacio de direcciones
- Fragmentación