Capítulo 5: Administración de memoria

Contenido:

- Implementación de espacios de direcciones virtuales. Se necesita para lograr protección entre procesos. Mecanismos:
  - *Segmentación*: El primer computador con Unix del DCC y la facultad operaba con segmentación. No se verá porque está obsoleto.
  - *Paginamiento*: Todos los computadores de uso general implementan hoy en día este mecanismo, desde servidores hasta smartphones.
    - Solo se excluyen microprocesadores económicos para sistemas embebidos.
    - El primer PC y el primer Macintosh no implementaban espacios de direcciones virtuales.
- Paginamiento en demanda: memoria virtual
  - Estrategia del *reloj*.
  - Estrategia del *working set*.

Paginamiento

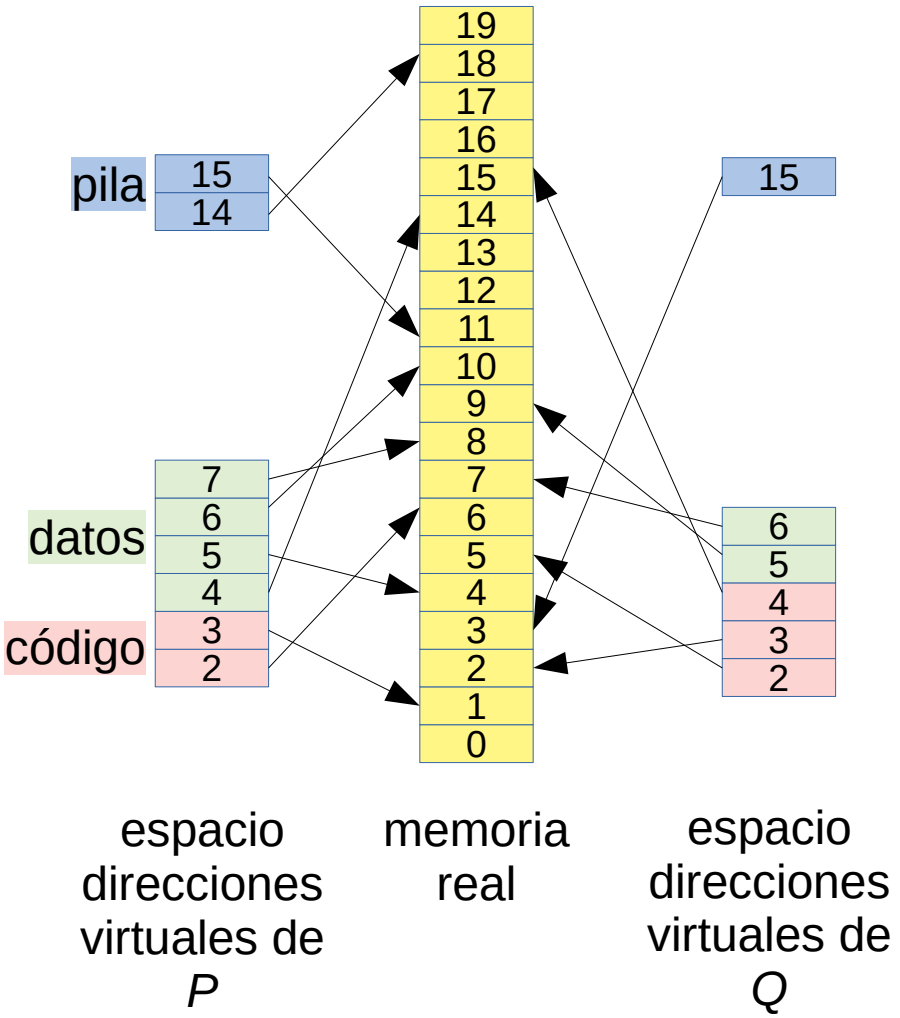
- El espacio de direcciones virtuales y la memoria real del computador (memoria física) se descompone en páginas
- Página:
  - Su tamaño está fijado por la arquitectura del procesador: 4 KB en x86
  - Es una potencia de 2: 4 KB = 2 ^ 12 B
  - Páginas se enumeran como 0, 1, 2, 3, etc.
  - Sea *S* = tamaño de página = 2 ^ *k*
  - Comienza en una dirección que es múltiplo del tamaño de la página: página *p* comienza en dirección *p* * *S* = *p* << *k*
  - Sea *d* = dirección de memoria
  - El tamaño de la página es una potencia de 2 para que sea fácil obtener el número de página a partir de una dirección *d* con *shift*:

    p = *d* / *S* = *d* >> *k*

  - *p* se obtiene ignorando los *k* bits menos significativos de la dirección
- Páginas virtuales: las páginas de un proceso
- Páginas reales: las páginas de la memoria real

Espacio de direcciones virtuales

- La página virtual *pv* de un proceso reside en alguna página real *pr*
- En general *pv* ≠ *pr*
- Ejemplo:



- Se observa que la página virtual 6 de P reside en la página real 10
- Tabla de páginas para proceso P:

V: validez
W: escritura

	pág. real	VW *Atributos*
15	11	1 1
14	18	1 1
13		0
12		0
11		0
10		0
9		0
8		0
7	8	1 1
6	10	1 1
5	4	1 1
4	14	1 1
3	1	1 0
2	6	1 0
1		0
0		0

Tabla de páginas

- Cada proceso tiene su propia tabla de páginas
- Almacena la ubicación de cada página virtual en la memoria real
- Se subindica por el número de página virtual
- Se obtiene el número de la página real en donde reside
- Los atributos indican qué se puede hacer con esa página:
  - V: bit de validez

    Si el proceso accede a una página con bit V en 0 se gatilla una interrupción llamada *page fault*
  - W: bit de escritura

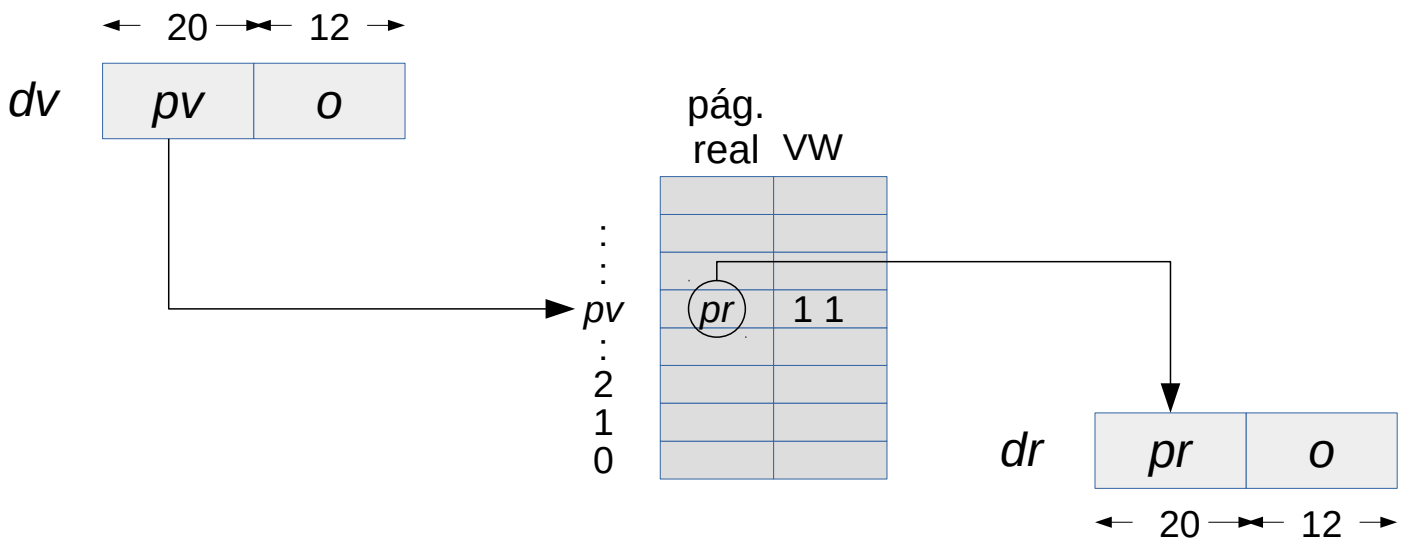
    Si el proceso escribe en una página con bit W en 0 también se gatilla un *page fault*
  - Otros atributos que veremos luego

Traducción de direcciones virtuales a direcciones reales

Cada vez que el proceso accede a una dirección virtual *dv* la *memory management unit* (MMU) del procesador traduce *dv* a una dirección real *dr* de esta manera:

dv	*dirección virtual*
pr = tabla_paginas[*dv*>>*k*].*pr*	*nro. página real*
o = *dv* & ((1<<*k*)-1)	*desplazamiento*
dr = (*pr*<<*k*) + *o*	*dirección real*

Ejemplo: caso x86 con direcciones de 32 bits, páginas de 4 KB (k=12)



TLB: Translation Lookaside Buffer

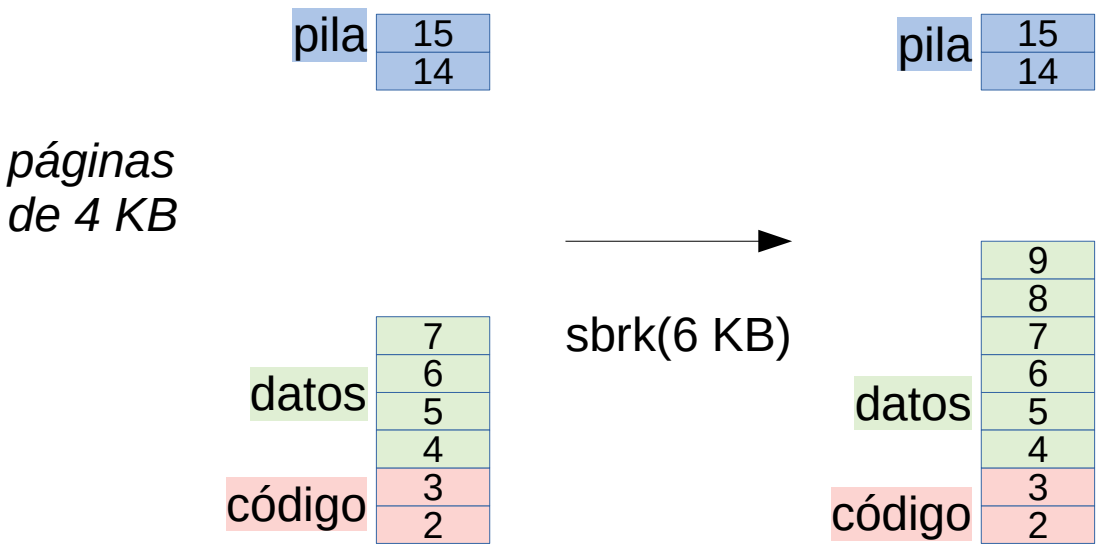
- Ineficiente: cada vez que se realiza un acceso a la memoria hay que realizar un segundo acceso a la tabla de páginas
- 100 % de sobrecosto en accesos a memoria
- Solución: TLB
- La TLB es un cache de traducciones de páginas
- Tamaño: almacena ~ 1024 traducciones hoy en día
- En cada acceso a una página *pv*:
  - Si (*pv*, *pr*) está en la TLB, *pr* es su traducción, sin sobrecosto alguno en tiempo de ejecución
  - En caso contrario: obtener (*pv*, *pr*) de la tabla de páginas en memoria con un sobrecosto de un acceso a memoria adicional, y almacenar (*pv*, *pr*) en la TLB
  - El ~ 99% de los accesos a memoria encuentran la traducción de la página en la TLB
- La búsqueda del dato en el cache L1 se hace usando su dirección virtual al mismo tiempo que se busca la dirección en la TLB
- Si el dato no está en el cache L1 se busca en el cache L2 usando su direción real

Cambio de contexto

- El hardware de cada *core* tiene un registro *TP* con la dirección de la tabla de páginas del proceso en ejecución en ese *core*
- La dirección de la tabla de páginas de un proceso se almacena en su descriptor de procesos y es una *dirección real*
- En un cambio de contexto de un proceso a otro proceso hay que cambiar el registro *TP* por la tabla de páginas del proceso entrante
- También se debe invalidar la TLB
- E invalidar el cache L1 porque se consulta usando direcciones virtuales
- No es necesario invalidar caches L2 y L3 porque se consultan usando las direcciones reales
- Invalidar cache L1 es caro en tiempo de ejecución: varios microsegundos, no tanto invalidar la TLB
- Si el cambio de contexto es a otro thread dentro del mismo proceso, no es necesario invalidar la TLB o el cache L1 porque se continúa en el mismo espacio de direcciones virtuales
- Por eso a los threads se les llama procesos livianos y a los procesos Unix procesos pesados, porque el cambio de contexto de los threads es más rápido

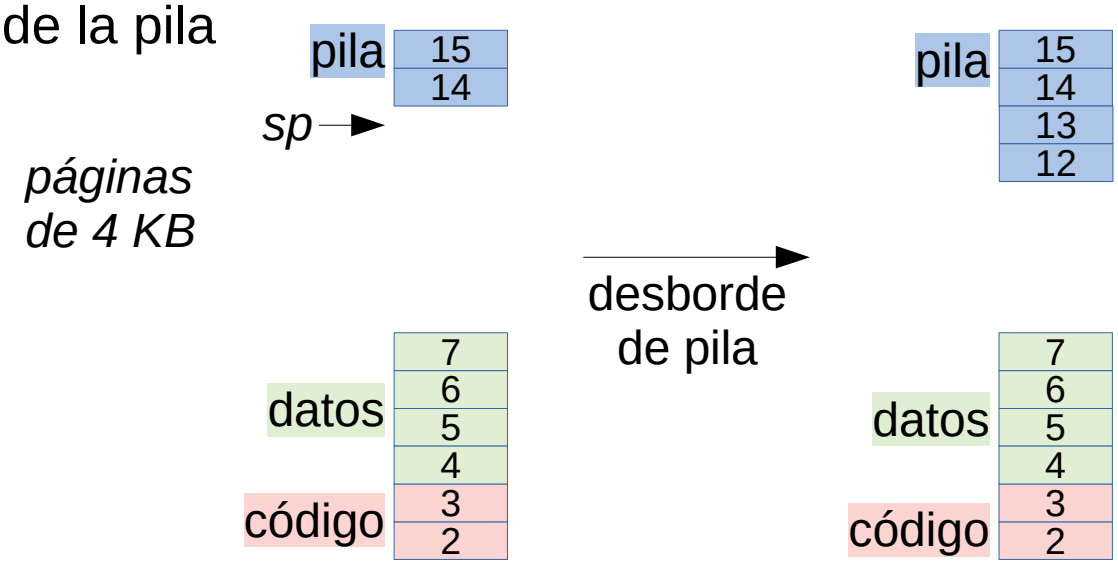
El potencial del paginamiento - I

- Espacios de direcciones virtuales
- Protección entre procesos
  - un proceso no puede leer o modificar la memoria de otro proceso
- Extensión explícita de los datos
  - cuando se le acaba la memoria a *malloc*, solicita más memoria con la llamada a sistema *sbrk*

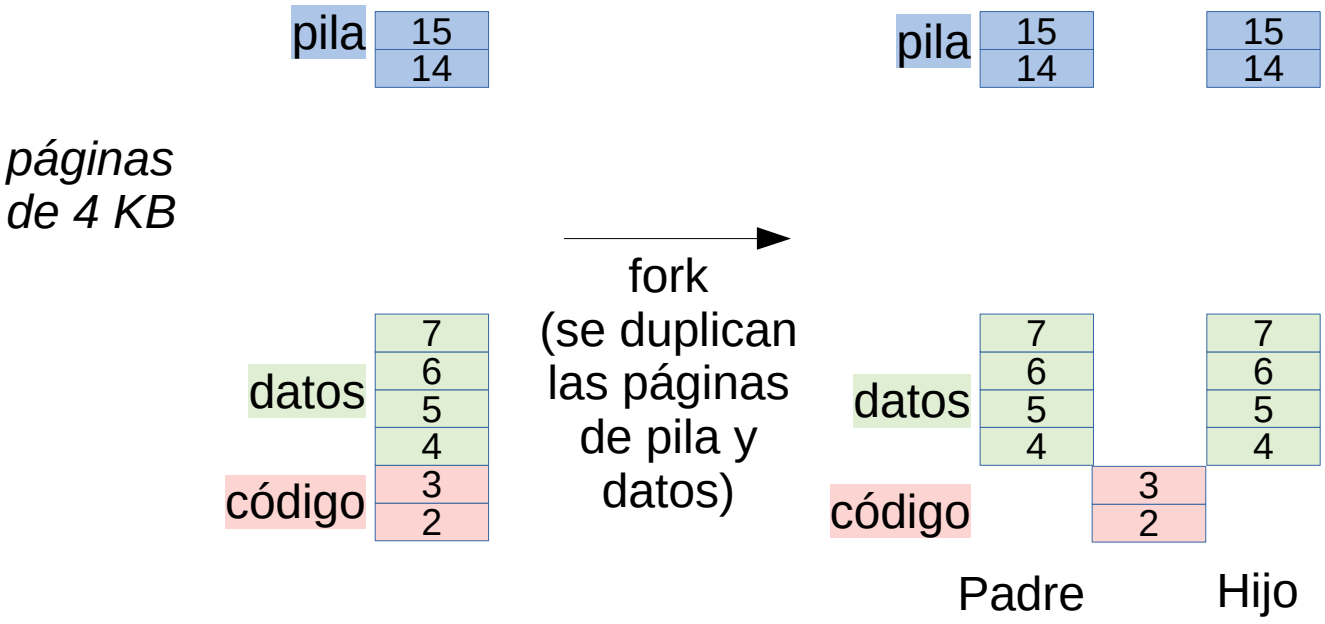


El potencial del paginamiento - II

- Extensión implícita de la pila en caso de desborde
  - cuando ocurre un *page fault* con una dirección en el rango de la pila (mayor que *sp*), el núcleo extiende automáticamente el tamaño de la pila

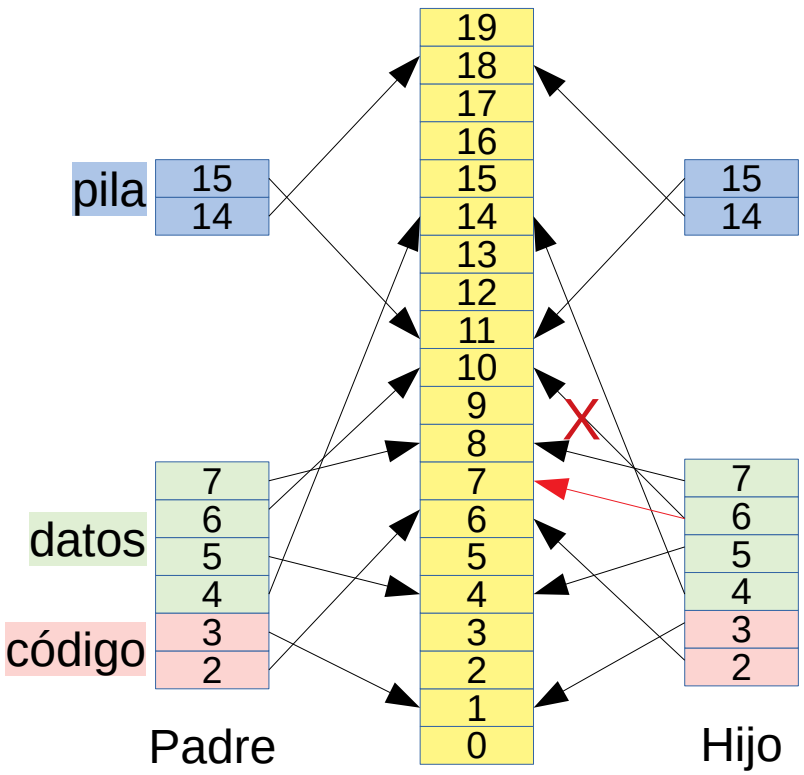


- Implementación *ineficiente* de *fork*



El potencial del paginamiento - III

- Implementación eficiente de *fork*
  - Páginas no se duplican
  - Padre e hijo comparten sus páginas
  - Atributo W se coloca en 0 en todas las páginas
  - Se agrega el atributo *copy on write* (COW): se coloca en 1 en las páginas que tenían W en 1
  - Si se escribe en página 6 se gatilla un page fault y solo entonces el núcleo duplica la página, cambiando el atributo W a 1



- Esta estrategia privilegia el uso más habitual de fork: el shell de comandos lo usa para lanzar un nuevo proceso ejecutando la secuencia *fork/exec*.
- Entre *fork* y exec se modifican muy pocas páginas.
- Cuando el hijo invoca *exec*, se descarta ese espacio de direcciones virtuales
- Es eficiente porque en la mayoría de los casos se duplican pocas páginas
- Pero es ineficiente si el *fork* no va seguido de *exec*, por ejemplo cuando se usa para paralelizar
- Si no hay *exec* después de duplicar ~10 páginas, duplicar todo el proceso

El potencial del paginamiento - IV

- *Swapping*
  - cuando la memoria escasea se llevan procesos completos a disco
  - se graba en disco un copia al byte de las páginas de un proceso
  - se cambia el estado del proceso a *swapped*
  - las páginas liberadas se utilizan para los procesos que quedan en memoria
  - no se puede ejecutar un proceso mientras está en disco
  - Es ingrato para el usuario que está detras de un proceso interactivo porque podría no haber respuesta hasta por un minuto
- Mejor: Paginamiento en demanda
  - Se llevan a disco las páginas no usadas recientemente por algún proceso
  - Se marcan como inválidas pero con un atributo adicional *S* que indica que están grabadas en disco
  - El proceso propietario puede continuar ejecutándose
  - Si el proceso accede a una página en disco, se gatilla un *page fault* y el núcleo carga transparentemente la página en memoria nuevamente