SO Curs 6

Memoria virtuala

Cuprins

Memoria virtuala
Page fault
Demand paging
Copy - on - Write
Mapareea fisierelor
Inlocuirea paginilor

Suport de curs

OSCE
- Capitolul 6 – Virtual Memory
- Capitolul 8 – Memory Management
- Sectiunile 4.4, 4.5

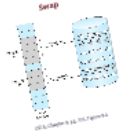
Cuvinte cheie

memorie virtuala	memory-mapped files
spatiu de adrese	memorie partajata
swap	inlocuirea de pagina
demand paging	MRU
page fault	FIFO, second chance
copy-on-write	thrashing

Concepte de memorie virtuala

Page fault
Swap
Demand paging

Swap



Swap (2)

Page fault (2)

Demand paging

Demand paging (2)

Page fault

Alocarea memoriei virtuale

Rezervare si commit
Demand paging
Pagini rezidente
Page locking

Demand paging

Pagini rezidente

Rezervare si commit

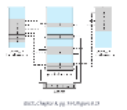
Page locking

Mecanisme de memorie virtuala

Copy-on-Write
Mapareea fisierelor

Copy-on-Write (2)

Mapareea fisierelor



Copy-on-Write

Mapareea fisierelor (2)

Stiva

Pagini

Abstractizari ale SO

Spatiul virtual de adrese

Memoria virtuala

De ce memorie virtuala?
Pagini
Abstractizari ale SO
Spatiul virtual de adrese
Avantaje si dezavantaje memorie virtuala

De ce memorie virtuală?

Avantaje si dezavantaje memorie virtuala

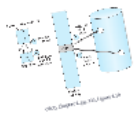
Algoritmi de inlocuire a paginilor

Algoritmi de inlocuire a paginilor (2)

Inlocuirea paginilor

Inlocuirea paginilor
Algoritmi de inlocuire a paginilor
Anomalia lui Belady
Thrashing

Inlocuirea paginilor



Anomalia lui Belady

Thrashing

SO Curs 6

Memoria virtuala

Cuprins

Memoria virtuala
Page fault
Demand paging
Copy - on - Write
Mapareea fisierelor
Inlocuirea paginilor

Suport de curs

OSCE
- Capitolul 6 – Virtual Memory
- Capitolul 8 – Memory Management
- Sectiunile 4.4, 4.5

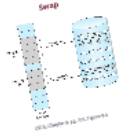
Cuvinte cheie

memorie virtuala	memory-mapped files
spatiu de adrese	memorie partajata
swap	inlocuirea de pagina
demand paging	MRU
page fault	FIFO, second chance
copy-on-write	thrashing

Concepte de memorie virtuala

Page fault
Swap
Demand paging

Swap



Swap (2)

Page fault (2)

Demand paging

Demand paging (2)

Page fault

Alocarea memoriei virtuale

Rezervare si commit
Demand paging
Pagini rezidente
Page locking

Demand paging

Pagini rezidente

Rezervare si commit

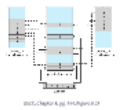
Page locking

Mecanisme de memorie virtuala

Copy-on-Write
Mapareea fisierelor

Copy-on-Write (2)

Mapareea fisierelor



Copy-on-Write

Mapareea fisierelor (2)

Stiva

Pagini

Abstractizari ale SO

Spatiul virtual de adrese

Memoria virtuala

De ce memorie virtuala?
Pagini
Abstractizari ale SO
Spatiul virtual de adrese
Avantaje si dezavantaje memorie virtuala

De ce memorie virtuală?

Avantaje si dezavantaje memorie virtuala

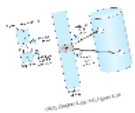
Algoritmi de inlocuire a paginilor

Algoritmi de inlocuire a paginilor (2)

Inlocuirea paginilor

Inlocuirea paginilor
Algoritmi de inlocuire a paginilor
Anomalia lui Belady
Thrashing

Inlocuirea paginilor



Anomalia lui Belady

Thrashing

Suport de curs

OSCE

- Capitolul 8 – Virtual Memory

MOS

- Capitolul 4 – Memory Management
  - Sectiunile 4.4, 4.5

Cuprins

Memoria virtuala
Page fault
Demand paging
Copy-on-Write
Maparea fisierelor
Inlocuirea paginilor

Memoria virtuala

De ce memorie virtuala?
Pagini
Abstractizari ale SO
Spatiul virtual de adrese
Avantaje si dezavantaje memorie virtuala

De ce memorie virtuală?

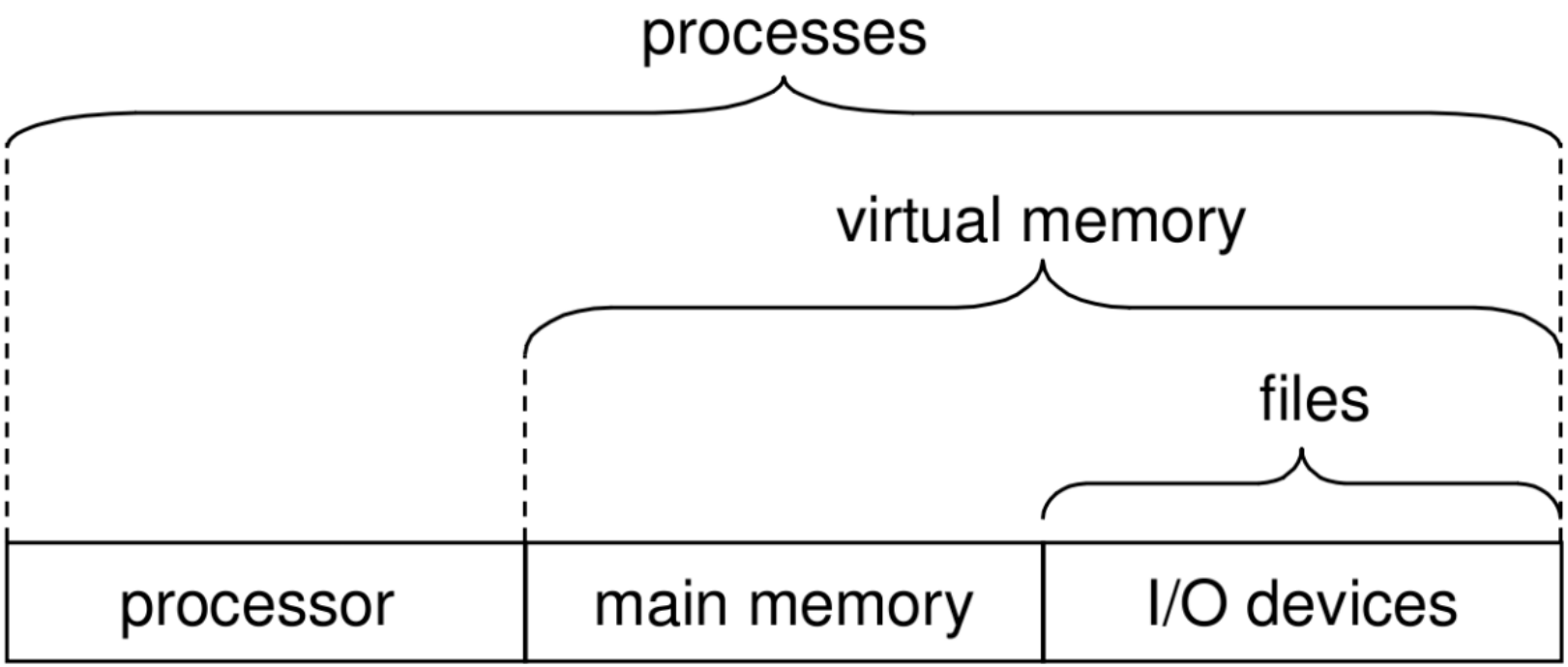
dificil să plasezi un proces de la adresa 0

decuplare de spatiul fizic de memorie
- un proces are acces la propriul spatiu virtual

memorie partajata
copy-on-write
maparea fisierelor

poti aloca doar memorie virtuala (demand paging)

Abstractizari ale SO



Pagini

fara fragmentare externa

cea mai mica unitate de alocare la nivel de SO

pagini virtuale (pages)
pagini fizice (frames)

maparea memoriei: asocierea unei pagini virtuale cu una fizica

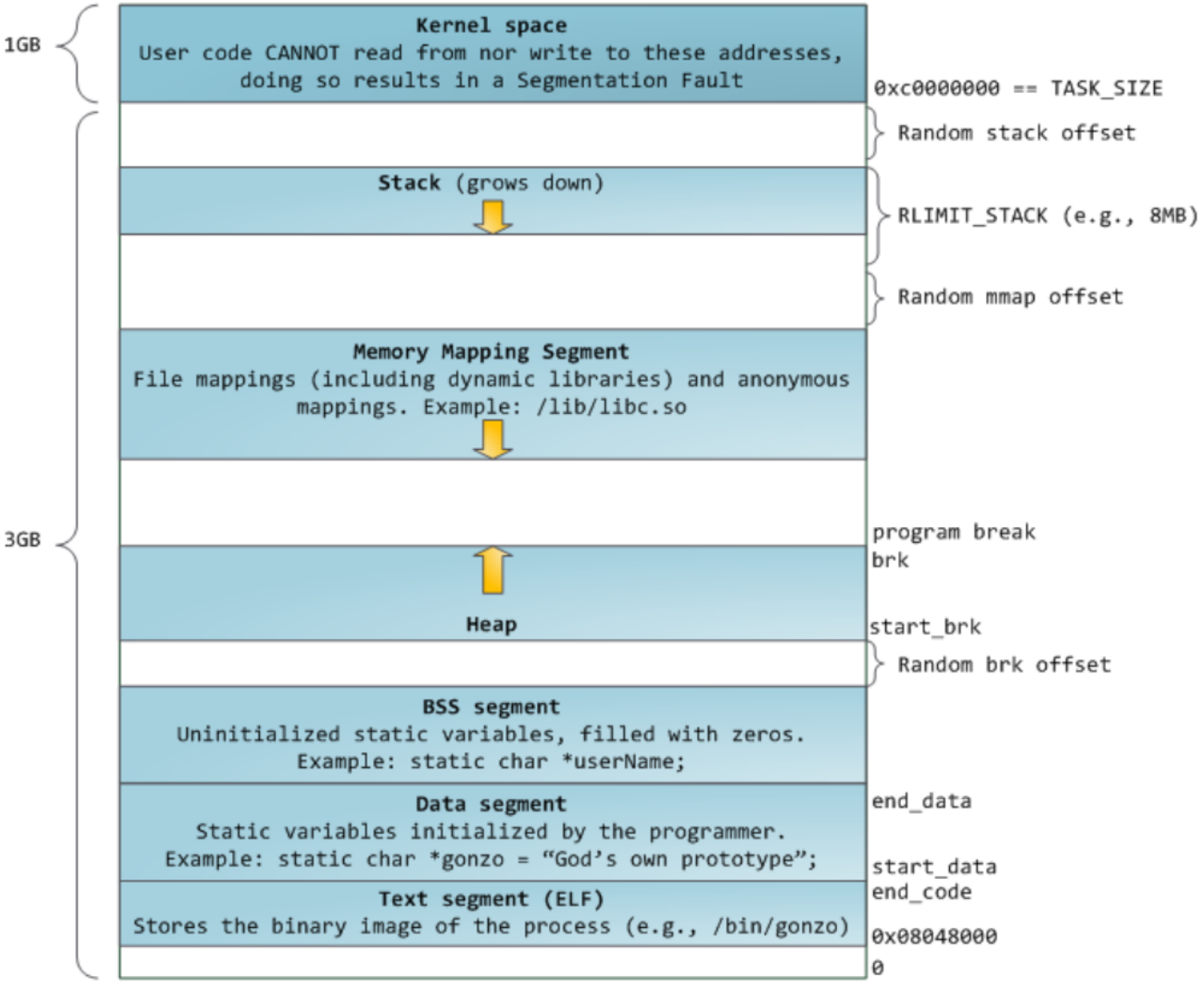
Avantaje si dezavantaje memorie virtuala

programatorul este eliberat de
- plasarea codului la o adresa data
- alocarea de memorie fizica

partajare memorie
folosire mai multa memorie decat are sistemul

acces dublu la memorie (tabela de pagini, TLB)
suport hardware (MMU, TLB)
componenta dedicata in SO (complexitate)

Spatiul virtual de adrese

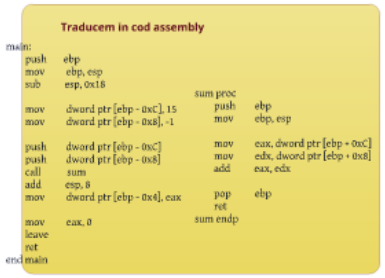


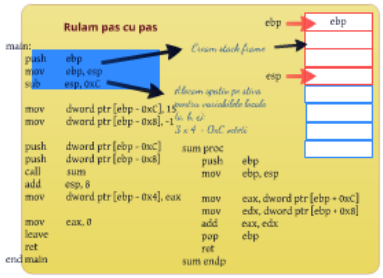
http://duartes.org/gustavo/blog/post/anatomy-of-a-program-in-memory

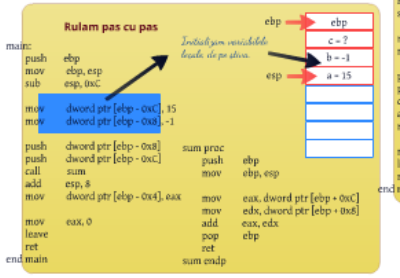
Stiva

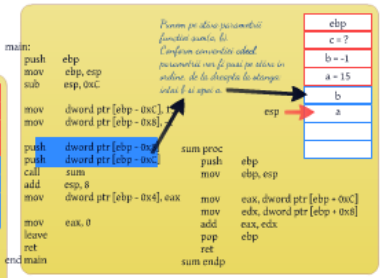
Scriem codul la nivel inalt

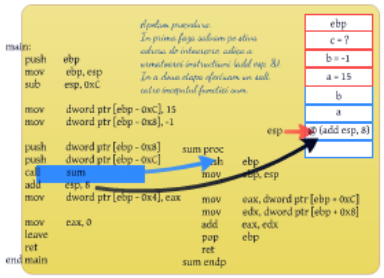
```
int sum(int a, int b) {
    return a + b;
}

int main() {
    int a = 15, b = -1, c;

    c = sum(a, b);
    return 0;
}
```

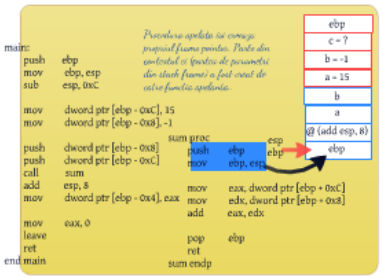


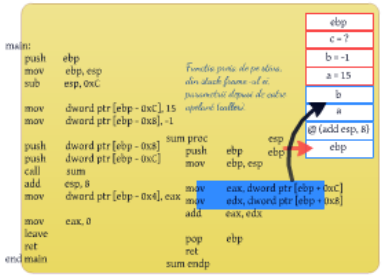


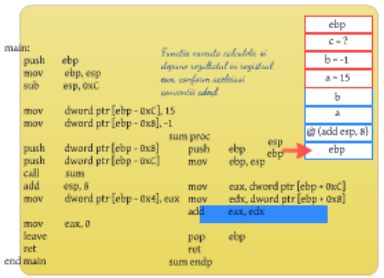


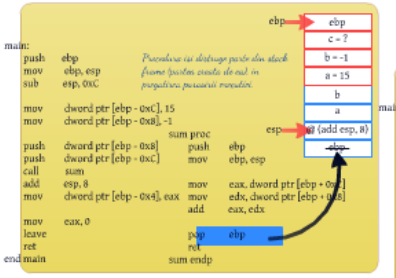


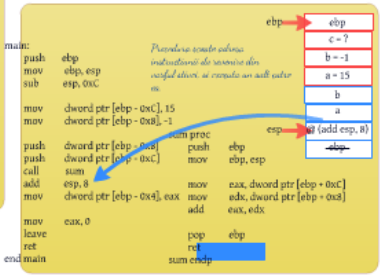


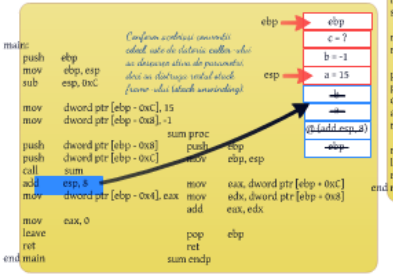


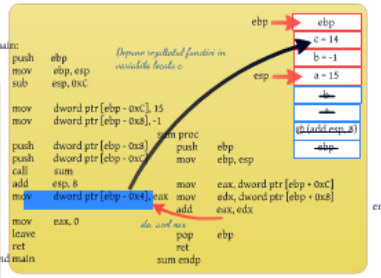


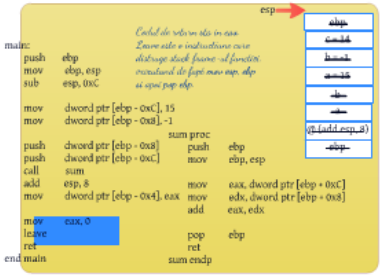












Scriem codul la nivel inalt

```
int sum(int a, int b) {
        return a + b;
}

int main() {
        int a = 15, b = -1, c;

        c = sum(a, b);
        return 0;
}
```

Traducem in cod assembly

```
in:
 push     ebp
 mov      ebp, esp
 sub      esp, 0x18

 mov      dword ptr [ebp - 0xC], 15
 mov      dword ptr [ebp - 0x8], -1

 push     dword ptr [ebp - 0xC]
 push     dword ptr [ebp - 0x8]
 call     sum
 add      esp, 8
 mov      dword ptr [ebp - 0x4], eax

 mov      eax, 0
 leave
 ret
d main
```

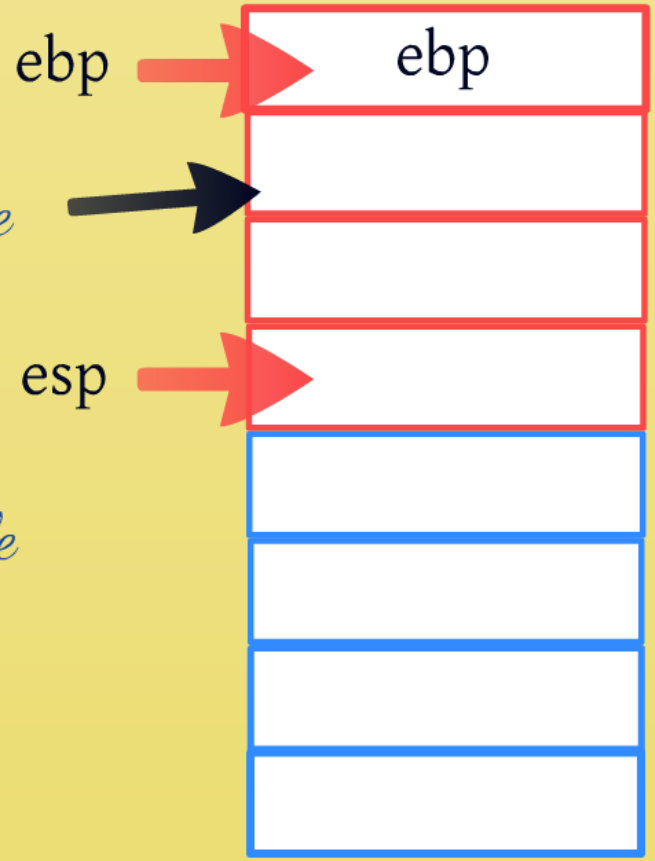
```
sum proc
        push     ebp
        mov      ebp, esp

        mov      eax, dword ptr [ebp + 0xC]
        mov      edx, dword ptr [ebp + 0x8]
        add      eax, edx

        pop      ebp
        ret
sum endp
```

Rulam pas cu pas

```
in:
 push     ebp
 mov      ebp, esp
 sub      esp, 0xC

 mov      dword ptr [ebp - 0xC], 15
 mov      dword ptr [ebp - 0x8], -1

 push     dword ptr [ebp - 0xC]
 push     dword ptr [ebp - 0x8]
 call      sum
 add      esp, 8
 mov      dword ptr [ebp - 0x4], eax

 mov      eax, 0
 leave
 ret
main endp
```

Cream stack frame

Alocam spatiu pe stiva pentru variabilele locale (a, b, c):
3 x 4 = 0xC octeti

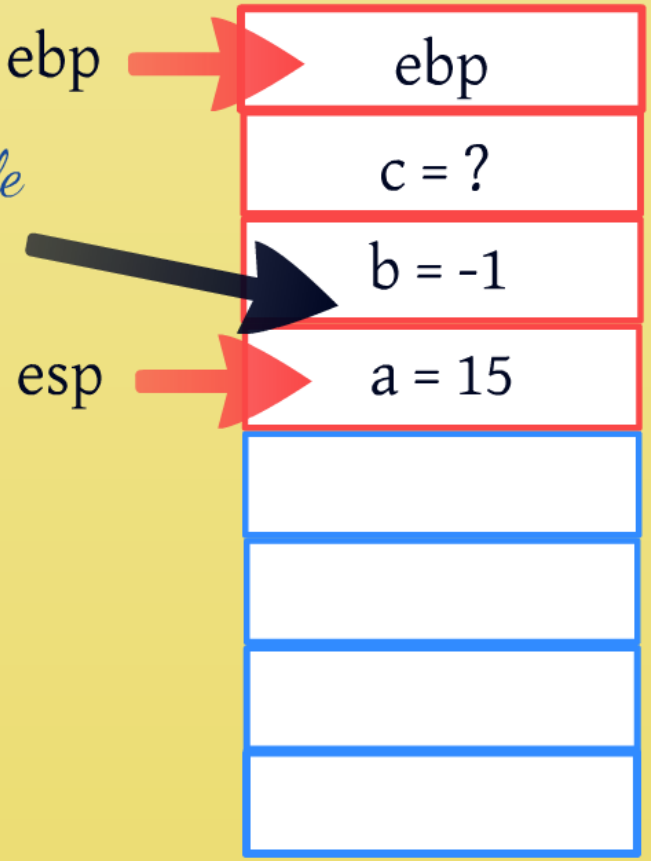



```
sum proc
     push     ebp
     mov      ebp, esp

     mov      eax, dword ptr [ebp + 0xC]
     mov      edx, dword ptr [ebp + 0x8]
     add      eax, edx
     pop      ebp
     ret
sum endp
```

Rulam pas cu pas

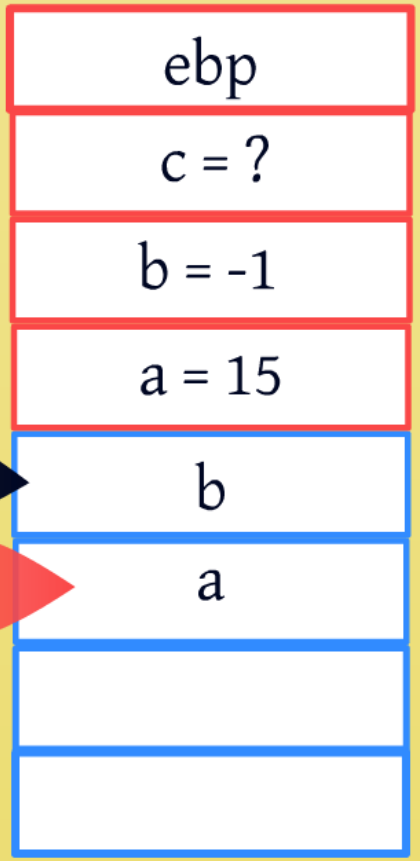
```
in:
 push     ebp
 mov      ebp, esp
 sub      esp, 0xC

 mov      dword ptr [ebp - 0xC], 15
 mov      dword ptr [ebp - 0x8], -1

 push     dword ptr [ebp - 0x8]
 push     dword ptr [ebp - 0xC]
 call     sum
 add      esp, 8
 mov      dword ptr [ebp - 0x4], eax

 mov      eax, 0
 leave
 ret
main endp
```

Initializam variabilele locale, de pe stiva.




```
sum proc
        push     ebp
        mov      ebp, esp

        mov      eax, dword ptr [ebp + 0xC]
        mov      edx, dword ptr [ebp + 0x8]
        add      eax, edx
        pop      ebp
        ret
sum endp
```

Punem pe stiva parametrii functiei sum(a, b).
Conform conventiei cdecl, parametrii vor fi pusi pe stiva in ordine, de la dreapta la stanga: intai b si apoi a.



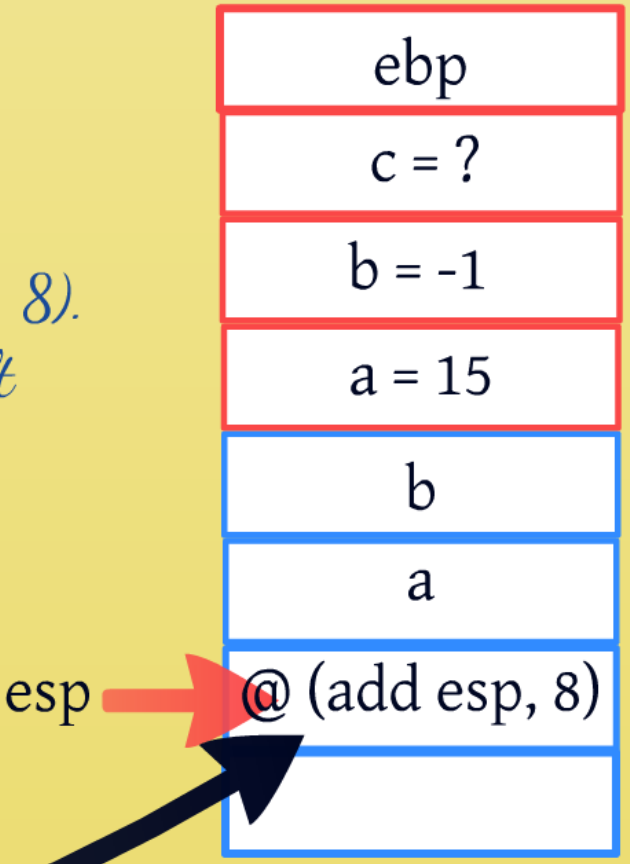

```
in:
 push     ebp
 mov      ebp, esp
 sub      esp, 0xC

 mov      dword ptr [ebp - 0xC], 15
 mov      dword ptr [ebp - 0x8], -1

 push     dword ptr [ebp - 0x8]
 push     dword ptr [ebp - 0xC]
 call     sum
 add      esp, 8
 mov      dword ptr [ebp - 0x4], eax

 mov      eax, 0
 leave
 ret
main endp
```

```
sum proc
     push     ebp
     mov      ebp, esp

     mov      eax, dword ptr [ebp + 0xC]
     mov      edx, dword ptr [ebp + 0x8]
     add      eax, edx
     pop      ebp
     ret
sum endp
```

Apelam procedura.
In prima faza salvam pe stiva adresa de intoarcere, adica a urmatoarei instructiuni (add esp, 8).
In a doua etapa efectuam un salt catre inceputul functiei sum.

```
in:
 push     ebp
 mov      ebp, esp
 sub      esp, 0xC

 mov      dword ptr [ebp - 0xC], 15
 mov      dword ptr [ebp - 0x8], -1

 push     dword ptr [ebp - 0x8]
 push     dword ptr [ebp - 0xC]
 call     sum
 add      esp, 8
 mov      dword ptr [ebp - 0x4], eax

 mov      eax, 0
 leave
 ret
main endp
```

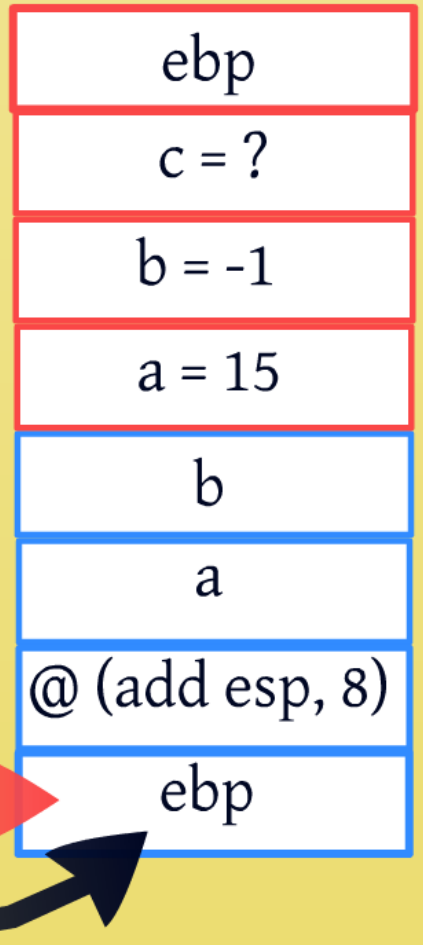
```
sum proc
 push     ebp
 mov      ebp, esp

 mov      eax, dword ptr [ebp + 0xC]
 mov      edx, dword ptr [ebp + 0x8]
 add      eax, edx
 pop      ebp
 ret
sum endp
```



Procedura apelata isi creeaza propriul frame pointer. Parte din contextul ei (partea de parametri din stack frame) a fost creat de catre functia apelanta..

```
in:
 push     ebp
 mov      ebp, esp
 sub      esp, 0xC

 mov      dword ptr [ebp - 0xC], 15
 mov      dword ptr [ebp - 0x8], -1

 push     dword ptr [ebp - 0x8]
 push     dword ptr [ebp - 0xC]
 call     sum
 add      esp, 8
 mov      dword ptr [ebp - 0x4], eax

 mov      eax, 0
 leave
 ret
main endp
```

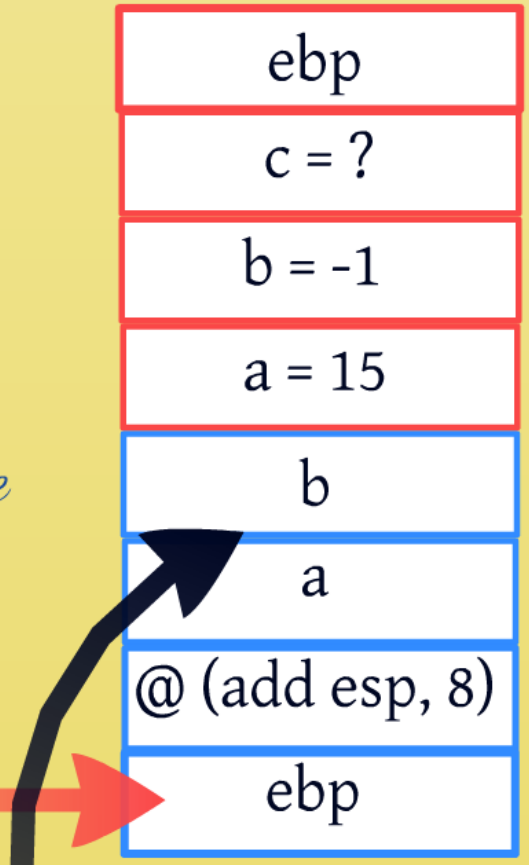
```
sum proc
     push     ebp
     mov      ebp, esp

     mov      eax, dword ptr [ebp + 0xC]
     mov      edx, dword ptr [ebp + 0x8]
     add      eax, edx

     pop      ebp
     ret
sum endp
```



```
in:
 push     ebp
 mov      ebp, esp
 sub      esp, 0xC

 mov      dword ptr [ebp - 0xC], 15
 mov      dword ptr [ebp - 0x8], -1

 push     dword ptr [ebp - 0x8]
 push     dword ptr [ebp - 0xC]
 call     sum
 add      esp, 8
 mov      dword ptr [ebp - 0x4], eax

 mov      eax, 0
 leave
 ret
main endp
```

Functia preia de pe stiva, din stack frame-ul ei, parametrii depusi de catre apelant (caller).

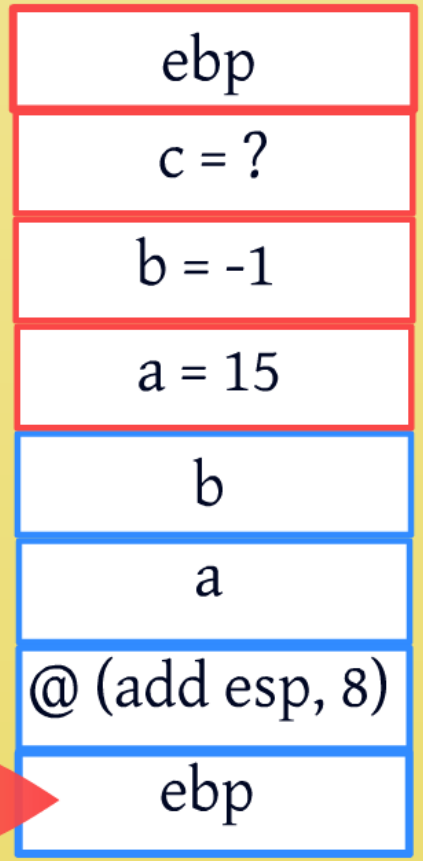
```
sum proc                    esp
        push    ebp         ebp
        mov     ebp, esp

        mov     eax, dword ptr [ebp + 0xC]
        mov     edx, dword ptr [ebp + 0x8]
        add     eax, edx

        pop     ebp
        ret
sum endp
```



```
in:
 push    ebp
 mov     ebp, esp
 sub     esp, 0xC

 mov     dword ptr [ebp - 0xC], 15
 mov     dword ptr [ebp - 0x8], -1

 push    dword ptr [ebp - 0x8]
 push    dword ptr [ebp - 0xC]
 call    sum
 add     esp, 8
 mov     dword ptr [ebp - 0x4], eax

 mov     eax, 0
 leave
 ret
main endp
```

Functia executa calculele, si depune rezultatul in registrul eax, conform aceleiasi conventii cdecl.

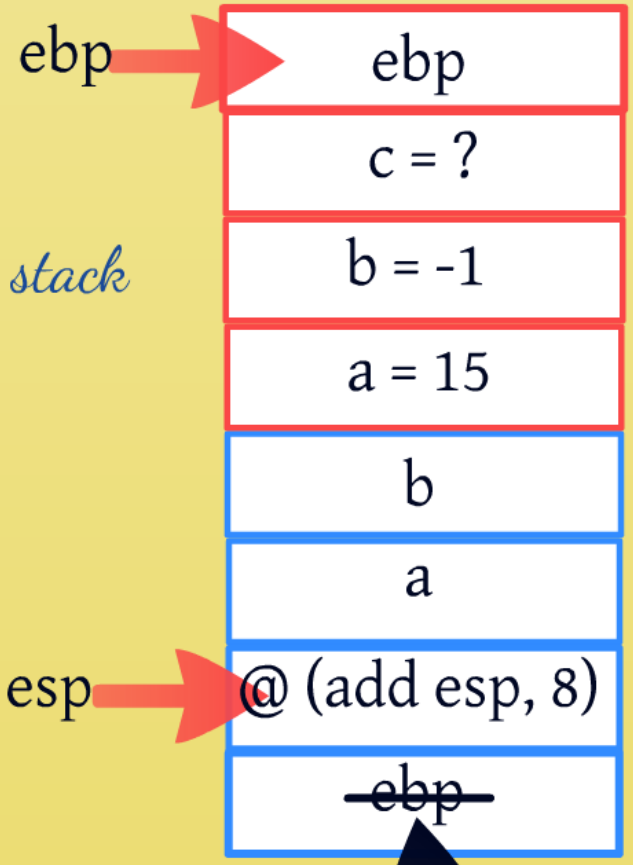
```
sum proc
        push    ebp
        mov     ebp, esp

        mov     eax, dword ptr [ebp + 0xC]
        mov     edx, dword ptr [ebp + 0x8]
        add     eax, edx

        pop     ebp
        ret
sum endp
```



```
in:
 push    ebp
 mov     ebp, esp
 sub     esp, 0xC

 mov     dword ptr [ebp - 0xC], 15
 mov     dword ptr [ebp - 0x8], -1

 push    dword ptr [ebp - 0x8]
 push    dword ptr [ebp - 0xC]
 call    sum
 add     esp, 8
 mov     dword ptr [ebp - 0x4], eax

 mov     eax, 0
 leave
 ret
d(in)in
```

Procedura isi distruge parte din stack frame (partea creata de ea), in pregatirea parasirii executiei.

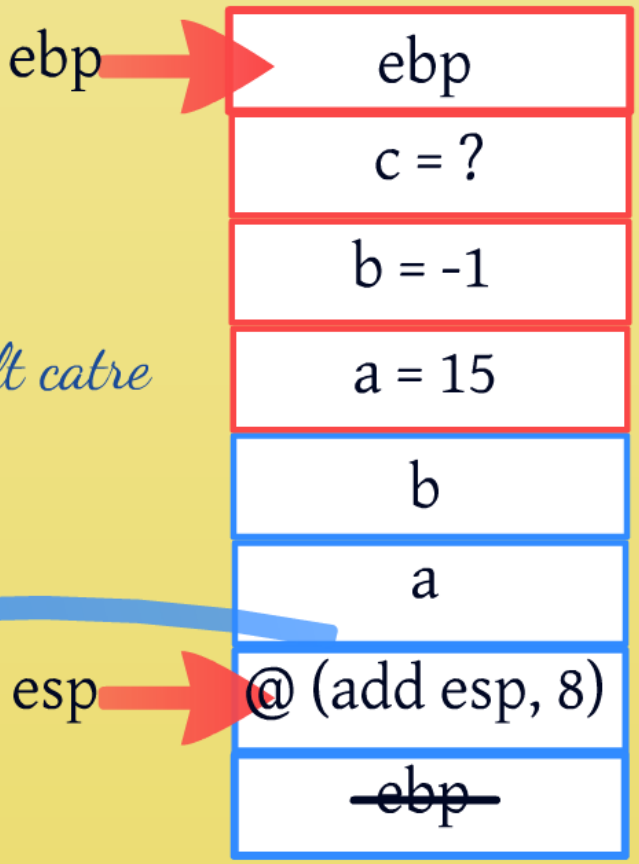
```
sum proc
        push    ebp
        mov     ebp, esp

        mov     eax, dword ptr [ebp + 0xC]
        mov     edx, dword ptr [ebp + 0x8]
        add     eax, edx

        pop     ebp
        ret
sum endp
```



```
in:
 push    ebp
 mov     ebp, esp
 sub     esp, 0xC

 mov     dword ptr [ebp - 0xC], 15
 mov     dword ptr [ebp - 0x8], -1

 push    dword ptr [ebp - 0x8]
 push    dword ptr [ebp - 0xC]
 call    sum
 add     esp, 8
 mov     dword ptr [ebp - 0x4], eax

 mov     eax, 0
 leave
 ret
main endp
```

Procedura scoate adresa instructiunii de revenire din varful stivei, si executa un salt catre ea.

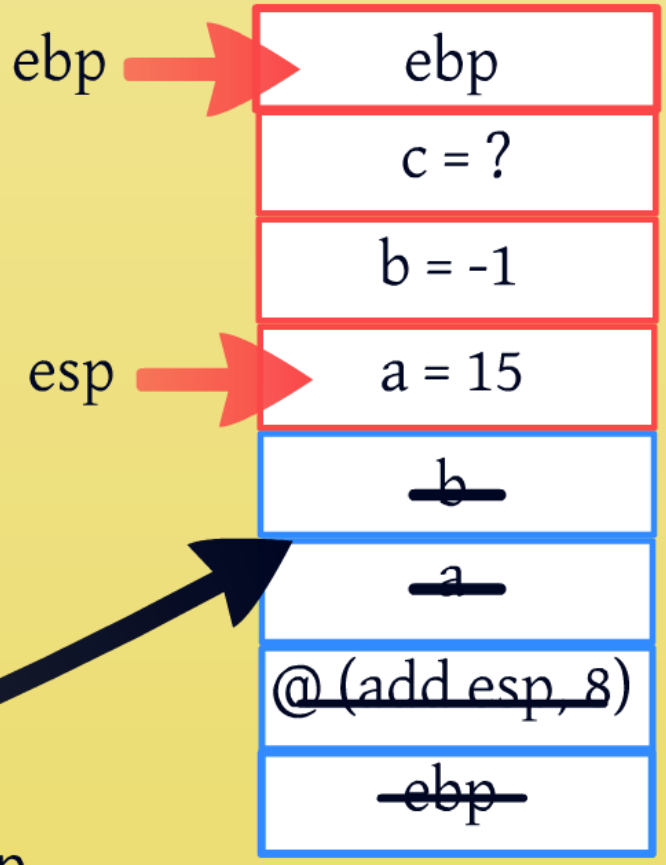
```
sum proc
 push    ebp
 mov     ebp, esp

 mov     eax, dword ptr [ebp + 0xC]
 mov     edx, dword ptr [ebp + 0x8]
 add     eax, edx

 pop     ebp
 ret
sum endp
```



```
in:
 push     ebp
 mov      ebp, esp
 sub      esp, 0xC

 mov      dword ptr [ebp - 0xC], 15
 mov      dword ptr [ebp - 0x8], -1

 push     dword ptr [ebp - 0x8]
 push     dword ptr [ebp - 0xC]
 call     sum
 add      esp, 8
 mov      dword ptr [ebp - 0x4], eax

 mov      eax, 0
 leave
 ret
main endp
```

```
sum proc
    push    ebp
    mov     ebp, esp

    mov     eax, dword ptr [ebp + 0xC]
    mov     edx, dword ptr [ebp + 0x8]
    add     eax, edx

    pop     ebp
    ret
sum endp
```

*Conform aceleiasi conventii cdecl, este de datoria caller-ului sa descarce stiva de parametri, deci sa distruga restul stack frame-ului (**stack unwinding**).*

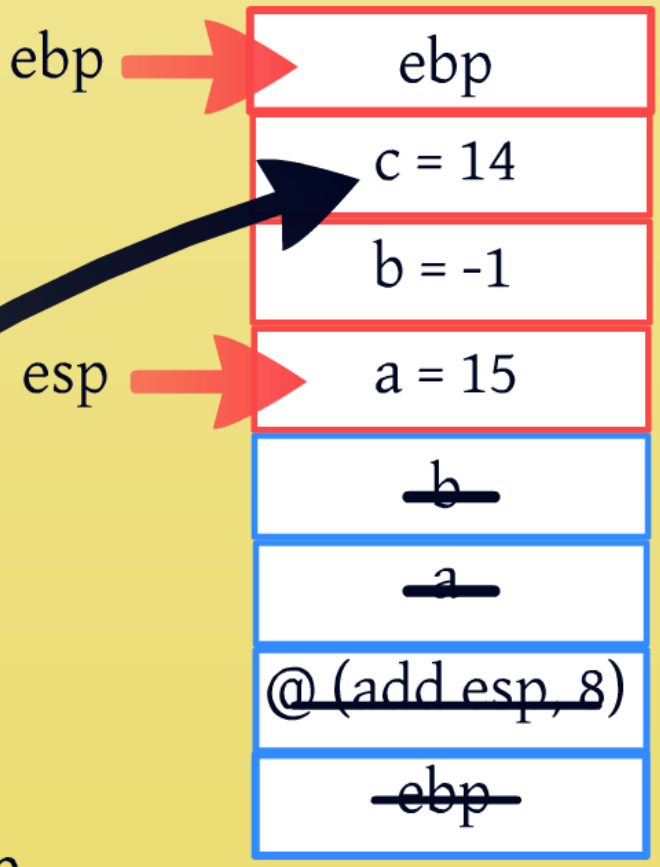


```
in:
 push     ebp
 mov      ebp, esp
 sub      esp, 0xC

 mov      dword ptr [ebp - 0xC], 15
 mov      dword ptr [ebp - 0x8], -1

 push     dword ptr [ebp - 0x8]
 push     dword ptr [ebp - 0xC]
 call     sum
 add      esp, 8
 mov      dword ptr [ebp - 0x4], eax

 mov      eax, 0
 leave
 ret
main endp
```

Depune rezultatul functiei in variabila locala c.

```
sum proc
 push     ebp
 mov      ebp, esp

 mov      eax, dword ptr [ebp + 0xC]
 mov      edx, dword ptr [ebp + 0x8]
 add      eax, edx

 pop      ebp
 ret
sum endp
```

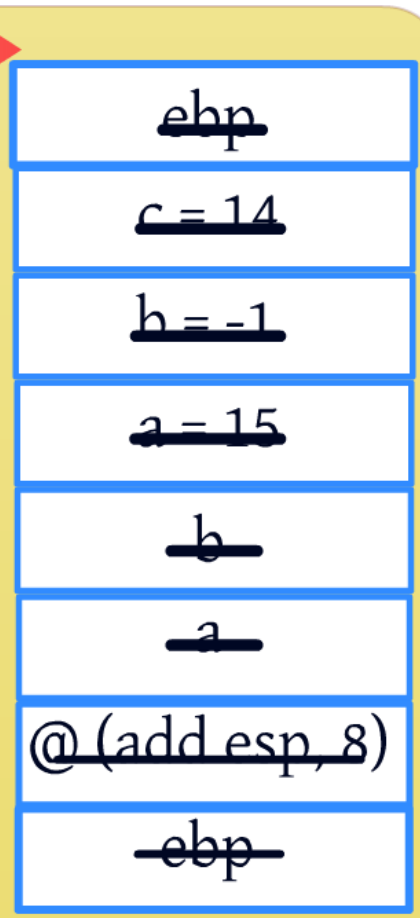
da, acel eax



Codul de return sta in eax.
Leave este o instructiune care distruge stack frame-ul functiei, executand de fapt mov esp, ebp si apoi pop ebp.

```
in:
 push     ebp
 mov      ebp, esp
 sub      esp, 0xC

 mov      dword ptr [ebp - 0xC], 15
 mov      dword ptr [ebp - 0x8], -1

 push     dword ptr [ebp - 0x8]
 push     dword ptr [ebp - 0xC]
 call     sum
 add      esp, 8
 mov      dword ptr [ebp - 0x4], eax

 mov      eax, 0
 leave
 ret
main endp
```

```
sum proc
    push     ebp
    mov      ebp, esp

    mov      eax, dword ptr [ebp + 0xC]
    mov      edx, dword ptr [ebp + 0x8]
    add      eax, edx

    pop      ebp
    ret
sum endp
```



Concepte de memorie virtuala

Page fault

Swap

Demand paging

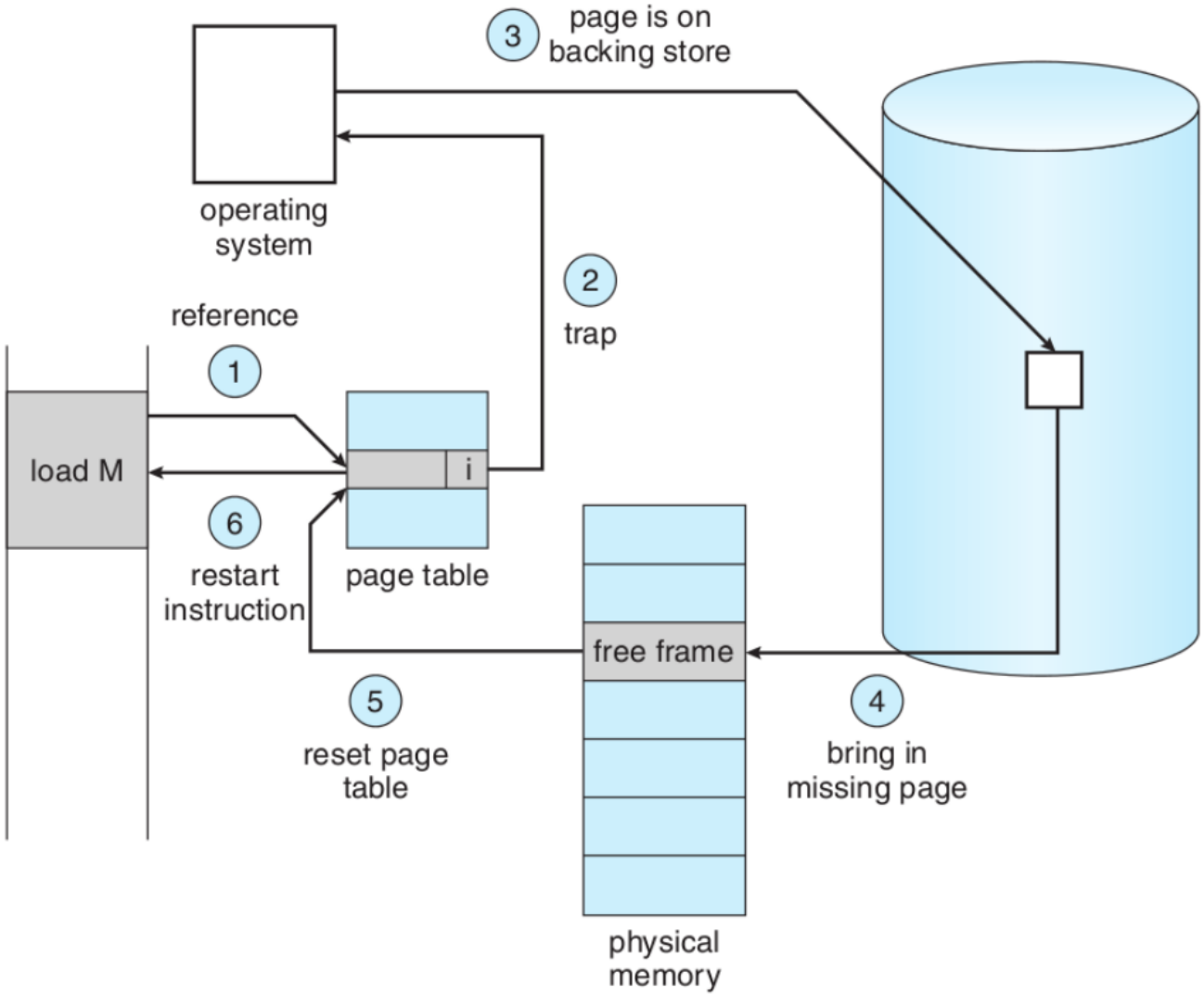
Page fault

acces la o pagina virtuala:
- nemapata (demand paging, swap)
- nevalida (pagina virtuala nealocata)
- drepturi nevalide (scriere pe o pagina read-only, copy-on-write)

se genereaza o exceptie/trap
MMU genereaza exceptia
se ruleaza page fault handler-ul

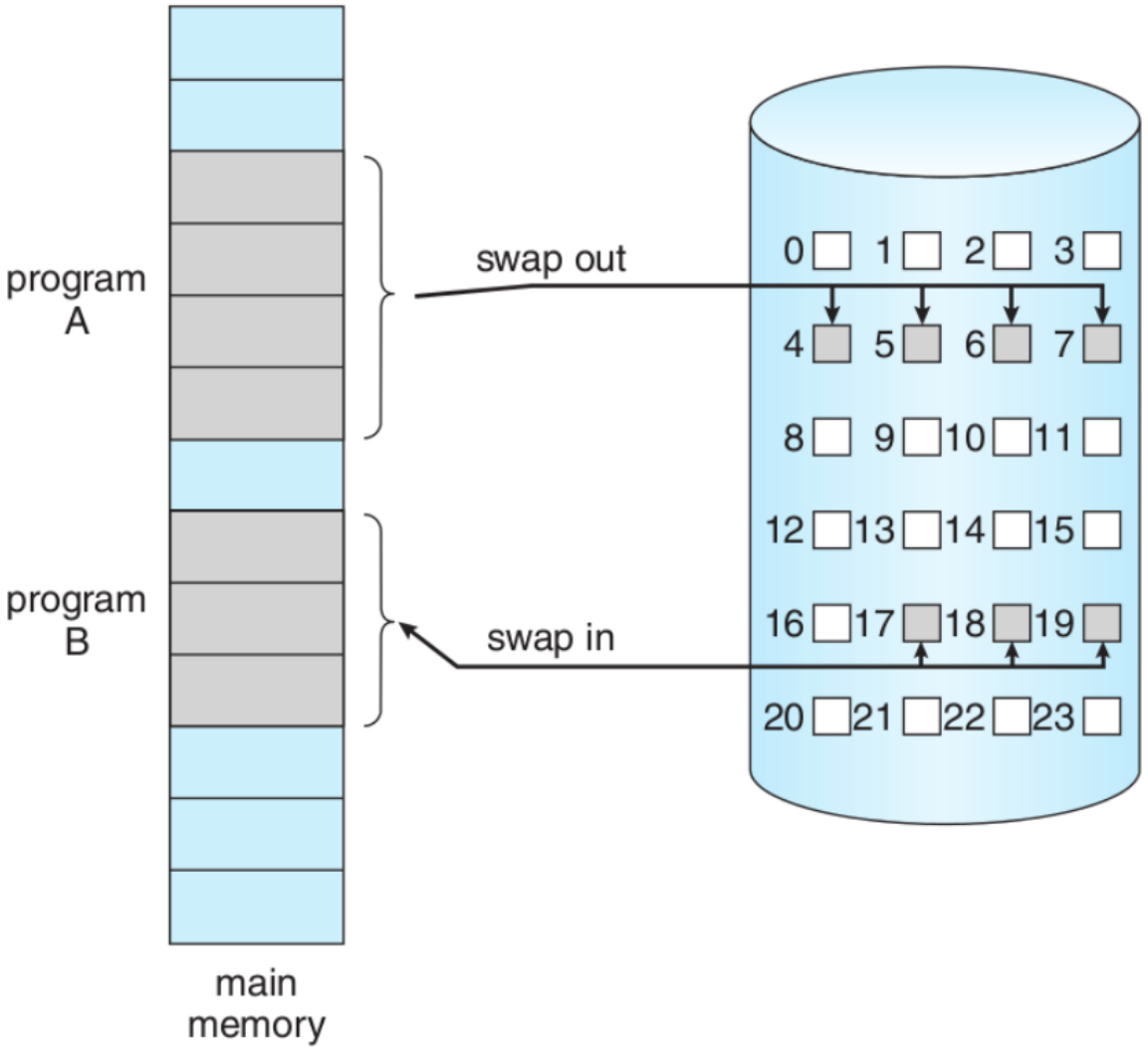
page fault-urile pot sa nu fie erori
- minor page faults: demand paging
- major page faults: swap

Page fault (2)



OSCE, Chapter 8, pg. 325, Figure 8.6

Swap



OSCE, Chapter 8, pg. 323, Figure 8.4

Swap (2)

in absenta memoriei fizice, se pot evacua pagini pe disc
spatiu de swap

- swap in (aducerea unei pagini de pe disc in RAM)
- swap out (trimiterea unei pagini din RAM pe disc)

mareste spatiul posibil folosit
mult mai incet decat memoria RAM

algoritmi de inlocuire de pagina

Demand paging

Paginare la cerere

paginare = alocarea unei pagini fizice si maparea acesteia

decuplarea alocarii de memorie virtuala de mapare

lazy allocation/paging

la alocare se aloca o intrare in tabela de pagini

- pagina este marcata nevalida
- este alocata pagina fizica (de la zero sau de swap) la nevoie

Demand paging (2)

un proces incarcat in memorie isi incarca doar o parte din date
- restul sunt incarcate la nevoie

o alocare cu mmap aloca doar pagini virtuale
- paginile fizice sunt alocate la acces

alocarea fizica si maparea se fac in page fault handler

Alocarea memoriei virtuale

Rezervare si commit
Demand paging
Pagini rezidente
Page locking

Rezervare si commit

rezervare: alocare memorie virtuala (doar virtuala)

- apelurile malloc (dimensiuni mari), mmap, VirtualAlloc

commit: alocare memorie fizica pentru memoria alocata

- la demand paging

decuplarea celor doua permite alocarea rapida de memorie

apelurile de biblioteca malloc fac mai putine apeluri de sistem brk

Demand paging

alocare la cerere

rezervare in prima faza

commit in page fault handler, la acces

Pagini rezidente

paginile prezente in RAM sunt rezidente
altfel pot fi pe swap sau inca nealocate

RSS = Resident Set Size, spatiul ocupat de un proces in RAM

alocare rezidenta (paginile nu pot fi swappate)

- la nivelul nucleului de operare
- sau prin page locking/pinning

Page locking

page pinning, fixed pages, non-pageable

pagina este marcata rezidenta (neswappabila)

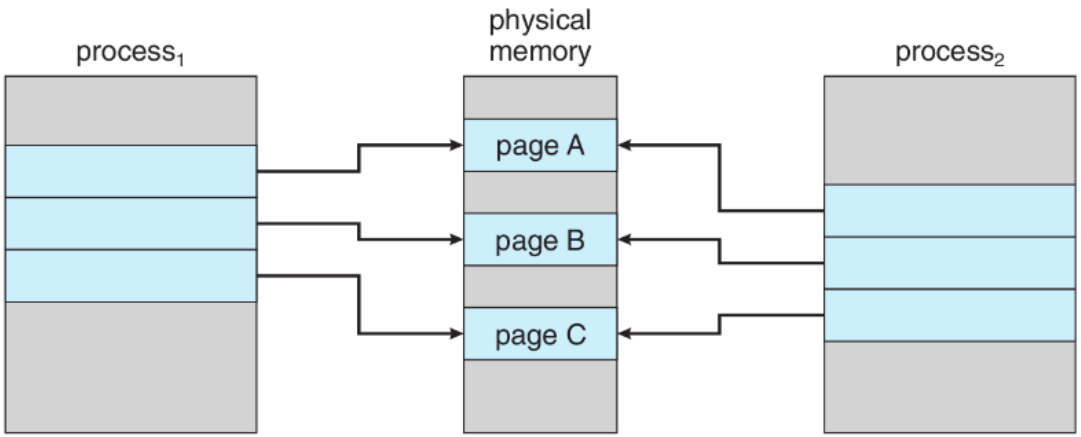
utila pentru comunicarea cu dispozitive I/O

mlock() - Linux
VirtualLock() - Windows

Mecanisme de memorie virtuala

Copy-on-Write
Maparea fisierelor

Copy-on-Write



OSCE, Chapter 8, pg. 330, Figures 8.6 & 8.8

Copy-on-Write (2)

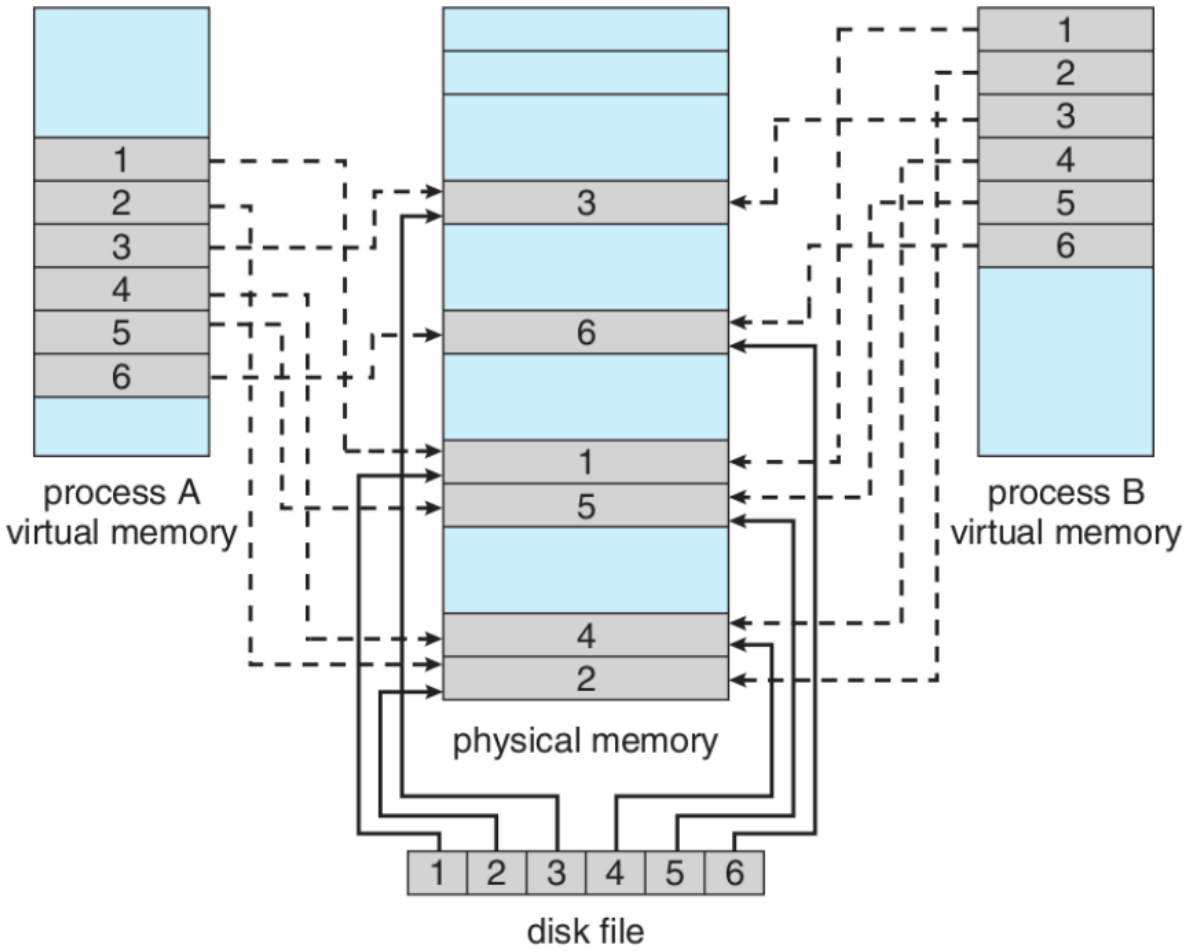
implementare naiva fork(): se copiaza spatiul de adresa

implementare eficienta: se duplica tabela de pagini

se partajeaza paginile intre procesul fiu si procesul parinte
- paginile sunt marcate read-only

in momentul unui acces de tip scriere se genereaza page fault
- se creeaza o copie a paginii configurata READ_WRITE la care va avea acces doar procesul care a generat page fault
- Pagina originala ramane configurata READ_ONLY si va avea acces la ea doar celalalt proces

Maparea fisierelor



OSCE, Chapter 8, pg. 354, Figure 8.23

Maparea fisierelor (2)

un bloc de date al unui fisier este mapat intr-o pagina/set de pagini
lucrul cu fisiere se realizeaza prin operatii de acces la memorie

folosit pentru incarcarea executabilelor si bibliotecilor

scrierea la o adresă de memorie inseamna scrierea in fisier
scrierile nu sunt imediate/sincrone (se folosesc apeluri msync)
memoria este folosită pe post de cache

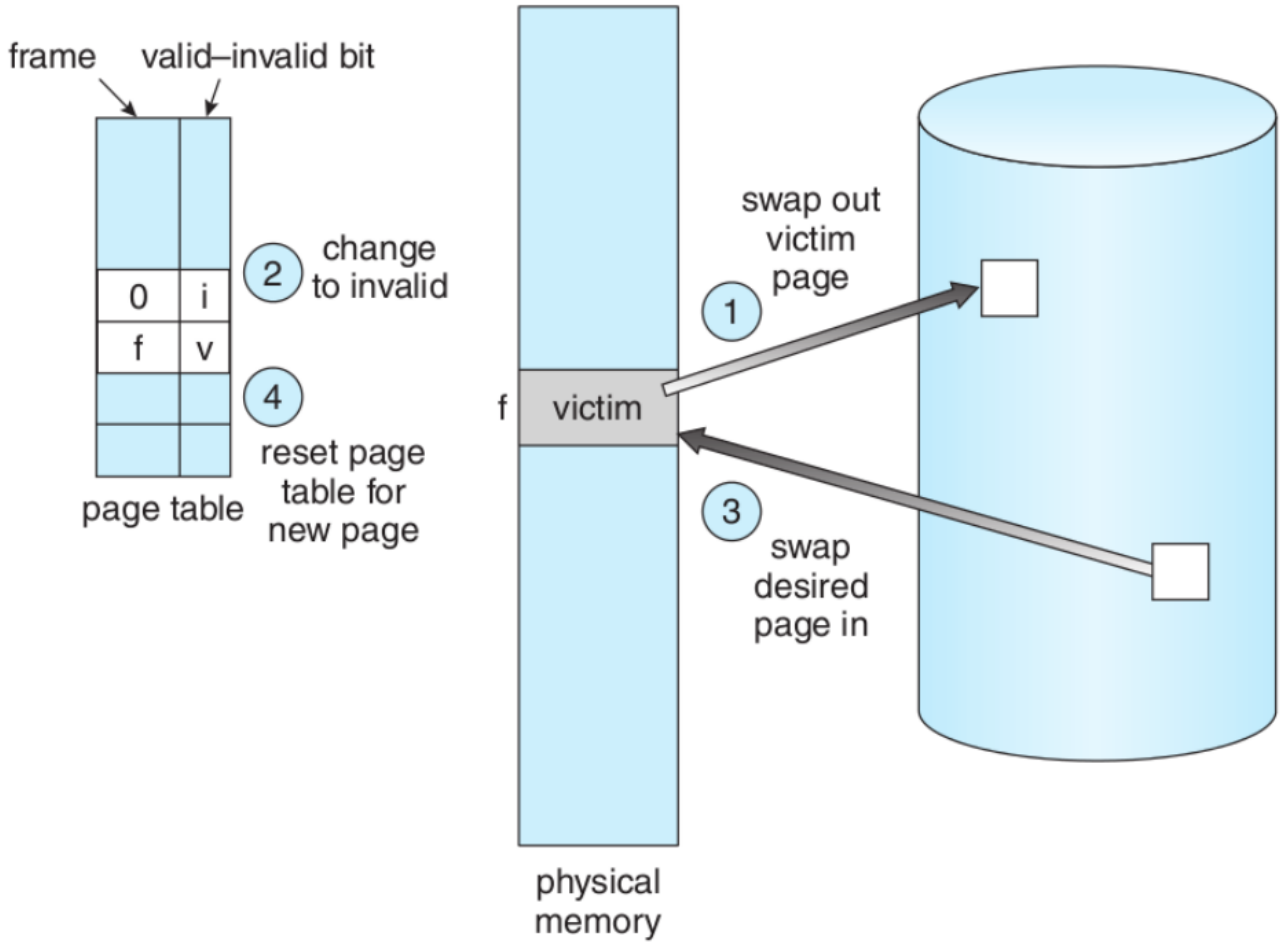
mai putine apeluri de sistem pentru lucrul cu fisiere

permite shared memory

Inlocuirea paginilor

Inlocuirea paginilor
Algoritmi de inlocuire a paginilor
Anomalia lui Belady
Thrashing

Inlocuirea paginilor



OSCE, Chapter 8, pg. 333, Figure 8.10

Algoritmi de inlocuire a paginilor

ce pagina va fi evacuata pe disc

optim: se inlocuieste pagina care va fi referita cel mai tarziu

biti de modified (M) si referenced (R) in pagina

- o pagina cu M a fost scrisa
- o astfel de pagina este "dirty"
- o pagina cu R evacuata, va fi doar invalidata nu si copiata pe disc

NRU: Not Recently Used (se inlocuieste clasa cea mai mica)

- clasa 0: R = 0, M = 0
- clasa 1: R = 0, M = 1
- clasa 2: R = 1, M = 0
- clasar 3: R = 1, M = 1

Algoritmi de inlocuire a paginilor (2)

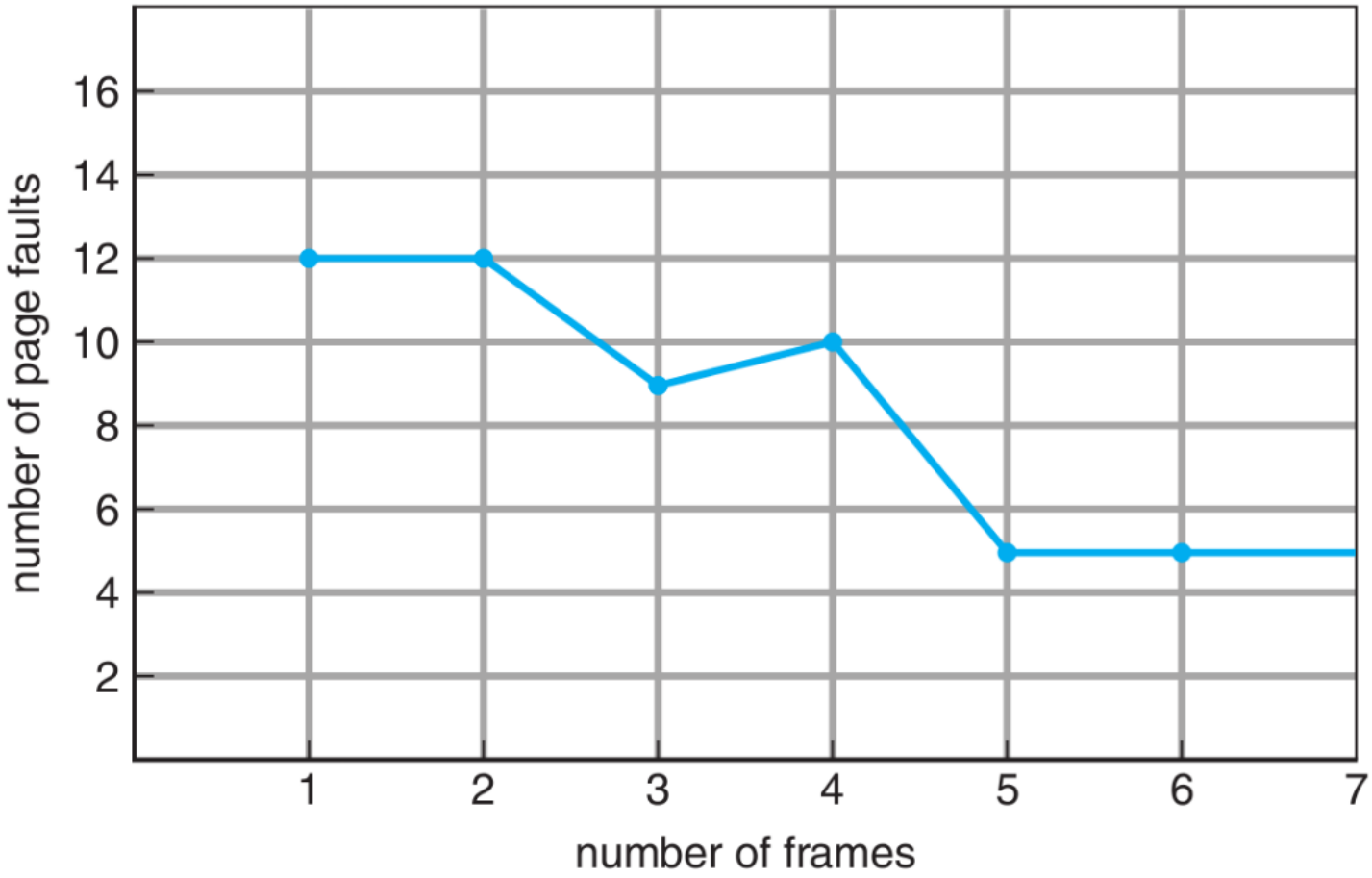
FIFO

- se mentine o lista cu paginile din memorie
- noile pagini sunt adaugate la sfarsitul listei
- se inlocuieste prima pagina din lista (cea mai veche)

Second chance (varianta modificata de FIFO)

- se tine cont de bitul R
- se inspecteaza prima pagina din lista
- daca R=0, pagina este selectata pentru inlocuire
- daca R=1, pagina este mutata la coada listei si R=0

Anomalia lui Belady



OSCE, Chapter 8, pg. 337, Figure 8.13

Thrashing

la incarcare mare a sistemului inlocuiri frecvente

schimbarea contextului duce la noi inlocuiri
- swap in/swap out frecvente

se petrece mult timp in page fault-uri (swap in/out)
- ineficienta in folosirea procesorului

exista si notiunea de cache thrashing

Cuvinte cheie

memorie virtuala
spatiu de adresa
swap
demand paging
page fault
copy-on-write

memory-mapped files
memorie partajata
inlocuire de pagina
NRU
FIFO, second chance
thrashing