Process virtual memory

Sistemi Operativi A.A. 2020-2021

Marco Tessarotto

Organizzazione della memoria di un processo (memory layout of a process)

- La memoria allocata a ciascun processo è organizzata in sezioni, generalmente denominate **segmenti**.
- **Text segment**: contiene le istruzioni in linguaggio macchina del programma. È di sola lettura per evitare modifiche accidentali delle istruzioni. È condiviso tra processo padre e processi figli (una unica copia, si risparmia memoria)
- **Initialized data segment**: contiene le variabili globali e static che sono inizializzate esplicitamente (i valori di inizializzazione si trovano nel file di programma e sono copiate in memoria all'avvio dello stesso)

Organizzazione della memoria di un processo (memory layout of a process)

- **Uninitialized data segment (bss segment)**: contiene le variabili globali e static che non sono inizializzate esplicitamente; all'avvio del programma, sono inizializzate a zero (non occupano spazio nel file di programma)
- **Stack segment** : la «pila» (stack) organizza le chiamate di funzioni nel corso dell'esecuzione del programma; le variabili locali, i parametri di funzione, il valore di ritorno delle funzioni «vivono» nella stack
- **Heap segment**: la heap («cumulo») è un'area da cui la memoria (per le variabili) può essere allocata dinamicamente a tempo di esecuzione (calloc, malloc, realloc). L'estremità superiore dell'heap è chiamata interruzione del programma («program break»)

esempio

size /bin/ls

text	**data**	**bss**	dec	hex	filename
124042	4728	4832	133602	209e2	/bin/ls

Gestione della memoria virtuale

- Come la maggior parte dei kernel moderni, Linux utilizza una tecnica nota come gestione della memoria virtuale (Virtual memory management) [Kerrisk pag. 118]
- Lo scopo di questa tecnica è quello di fare un uso efficiente sia della CPU che della RAM (memoria fisica) sfruttando una proprietà tipica della maggior parte dei programmi: il **principio di località**
- **Località spaziale**: è la tendenza di un programma a fare riferimento a **indirizzi di memoria vicini** a quelli acceduti di recente (es. array)
- **Località temporale**: è la tendenza di un programma ad accedere **nel prossimo futuro agli stessi indirizzi di memoria** a cui ha avuto accesso nel recente passato (es. cicli).

Gestione della memoria virtuale

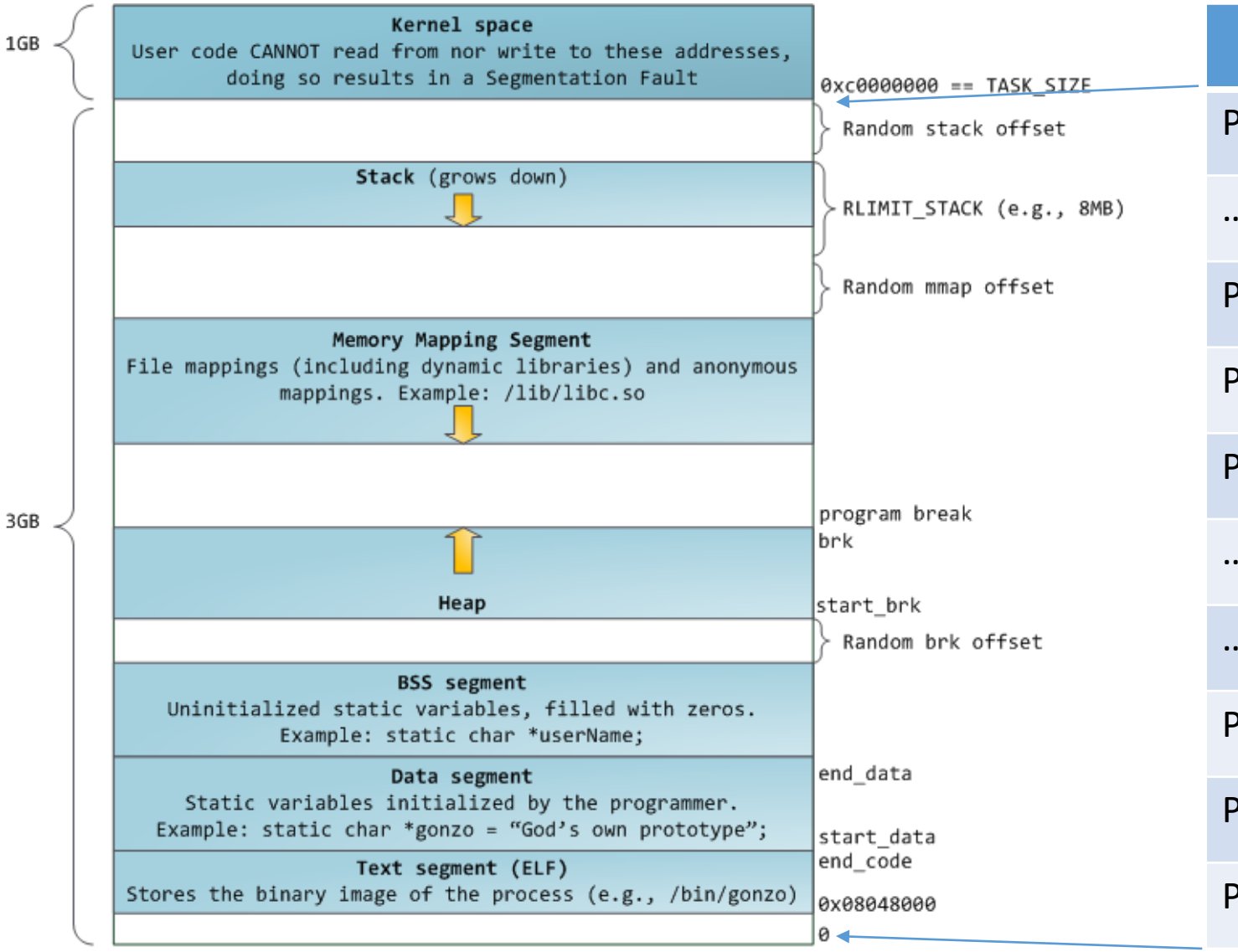
- Fare riferimento al Kerrisk, sezioni 6.1, 6.2, 6.3, 6.4 (a partire da pag. 113)
- Il risultato della «**locality of reference**» (**principio di località**) è che è possibile eseguire un programma mantenendo solo una parte del suo spazio di indirizzi nella RAM. (Kerrisk pag.118)
- Una gestione della memoria virtuale divide la memoria utilizzata da ciascun processo in piccole unità di dimensioni fisse chiamate **pagine** (dimensione 4096 bytes).
- In modo corrispondente, la RAM è divisa in una serie di «**page frame**» (cornici di pagina) della stessa dimensione
- [pagine e cornici di pagine sono «associate» tra di loro dal SO]

Gestione della memoria virtuale

- In qualsiasi momento, solo alcune delle pagine di un programma possono essere residenti in «page frame» della memoria fisica (anche tutte, se c'è memoria a sufficienza); queste pagine formano il cosiddetto «**resident set**» (insieme delle pagine residenti)
- Le copie delle pagine non utilizzate di un programma vengono mantenute nella **swap area** (area di scambio) — un'area riservata dello spazio su disco utilizzata per integrare la RAM del computer — e caricata nella memoria fisica solo se necessario.
- Quando un processo fa riferimento a una pagina che non è attualmente residente nella memoria fisica, si verifica un errore di pagina (**page fault**), a quel punto il kernel sospende l'esecuzione del processo mentre la pagina viene caricata dal disco nella memoria.

Gestione della memoria virtuale

- Memoria virtuale del processo: viene divisa in «**page**» (pagine), page size: 4096 byte
- Memoria fisica: divisa in «**page frame**» (cornici di pagina), page frame size: 4096 byte
- **Resident set**: insieme delle «page» (pagine) residenti in RAM
- **Virtual address space**: l'insieme di tutte le pagine rese disponibili al processo
- **Swap area** (area di scambio): un'area riservata dello spazio su disco utilizzata per integrare la (in aggiunta alla) RAM del computer; sono conservate le pagine non utilizzate dai processi
- Hardware + software (CPU + kernel SO) implementano la gestione della memoria virtuale

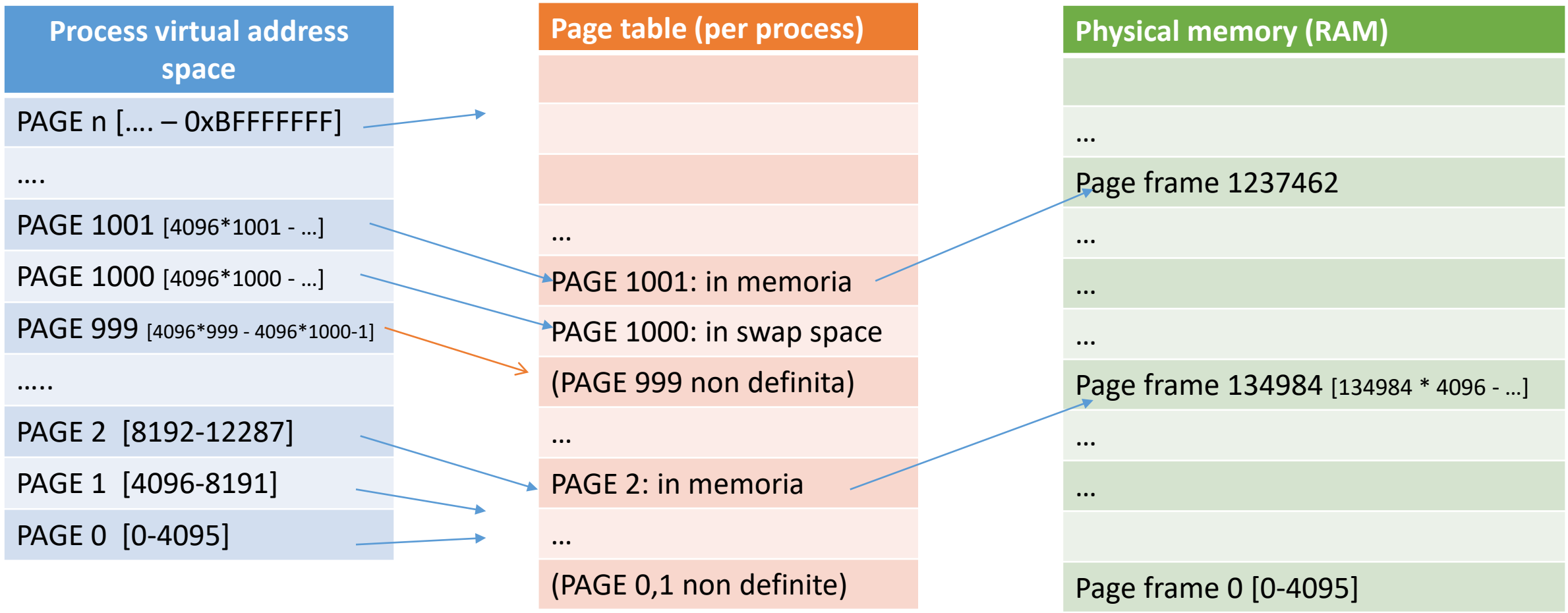


Process virtual address space
PAGE n […. – 0xBFFFFFFF]
…..
PAGE 1001 [4096*1001 - …]
PAGE 1000 [4096*1000 - …]
PAGE 999 [4096*999 - 4096*1000-1]
…..
…..
PAGE 2 [8192-12287]
PAGE 1 [4096-8191]
PAGE 0 [0-4095]

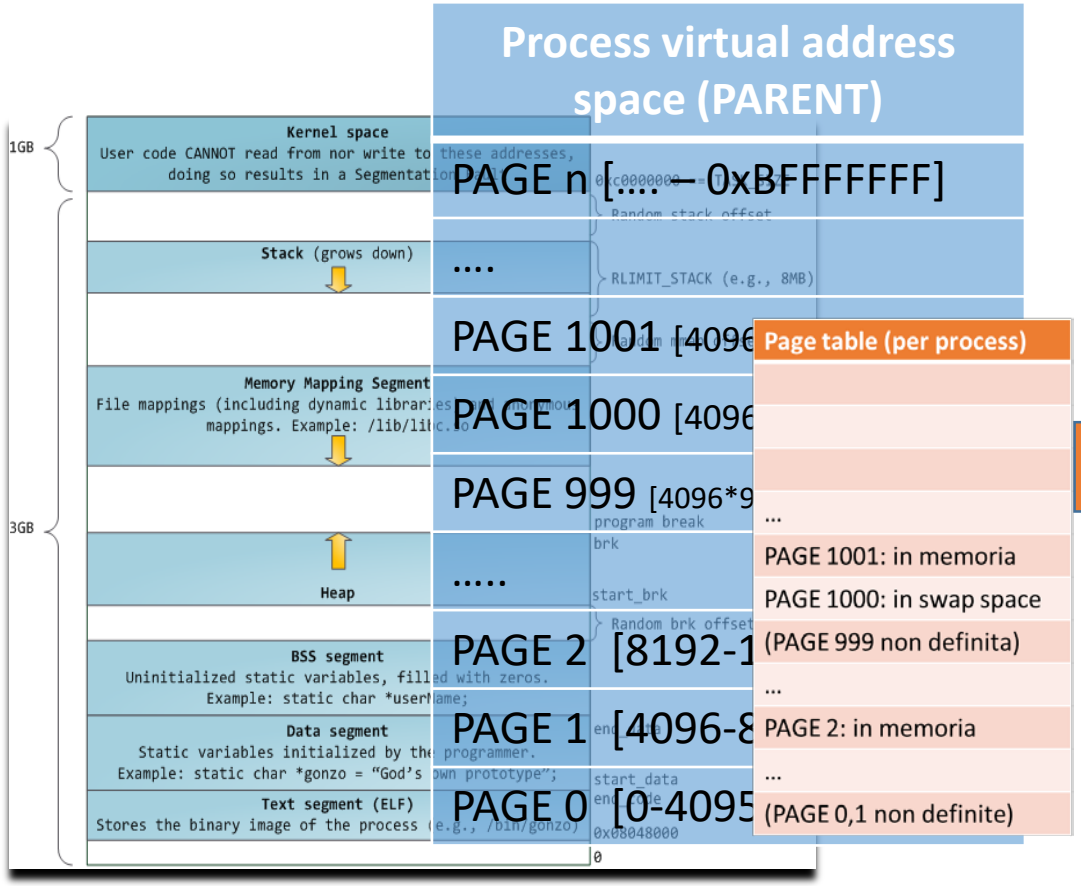
Esempio (regioni di memoria di un processo)

- https://github.com/marcotessarotto/exOpSys/tree/master/054showmem

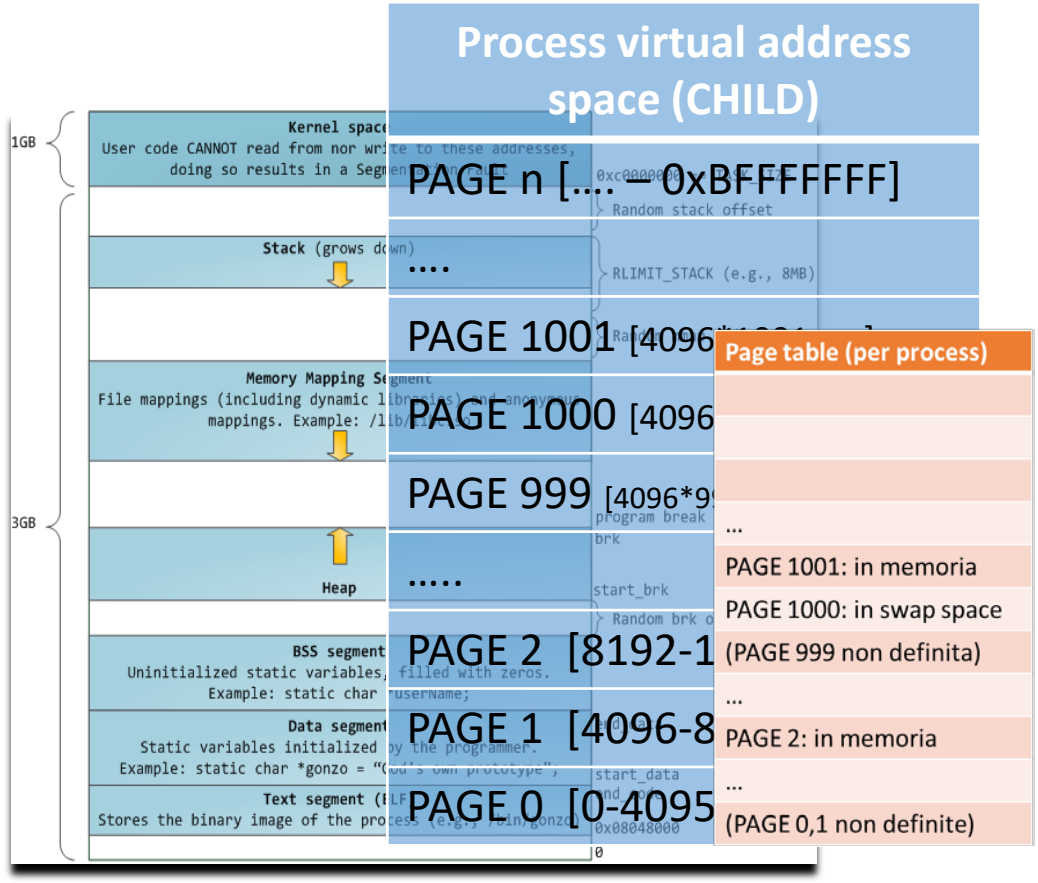
```
address (start-end)       perms offset  dev   inode   pathname
55e68b6b1000-55e68b6b2000 r-xp 00001000 00:19 8423916 054showmem
…
55e68b6b4000-55e68b6b5000 rw-p 00003000 00:19 8423916 054showmem
…
55e68cfcf000-55e68cff0000 rw-p 00000000 00:00 0 [heap]
…
7ffe65028000-7ffe6504b000 rw-p 00000000 00:00 0 [stack]
…
p : private (copy on write)
fork(): viene creata una copia alla prima scrittura da parte del nuovo processo
ma se i permessi sono di sola lettura => non viene fatta la copia ovvero una unica copia per i processi padre-figli
```



Indirizzo di memoria nel «virtual address space» del processo (shift right 12 bit) -> PAGE number
Lookup in process page table:
- page non definita -> SIGSEGV signal
- page ok e presente in memoria (parte del «resident set») -> physical memory (page frame)
- page ok e in swap space -> kernel: trovare e/o fare spazio in physical mem. e caricare da swap space la pagina



fork()



processo padre

processo figlio

system call fork() crea una COPIA identica della memoria virtuale del processo
QUINDI dopo fork gli indirizzi della memoria virtuale dei due processi sono IDENTICI ma DISTINTI
Il meccanismo copy on write serve a ritardare (se e quando serve) l'operazione di duplicazione delle page frame (memoria fisica)

processo padre

Process virtual address space
PAGE n [.... – 0xBFFFFFFF]
....
PAGE 1001 [4096*1001 - ...]
PAGE 1000 [4096*1000 - ...]
PAGE 999 [4096*999 - 4096*1000-1]
.....
PAGE 2 [8192-12287]
PAGE 1 [4096-8191]
PAGE 0 [0-4095]

Page table (per process)
...
PAGE 1001: in memoria
PAGE 1000: in swap space
(PAGE 999 non definita)
...
PAGE 2: in memoria
...
(PAGE 0,1 non definite)

dopo fork(), contenuto di page frame 3123498 == contenuto di page frame 1237462 etc etc

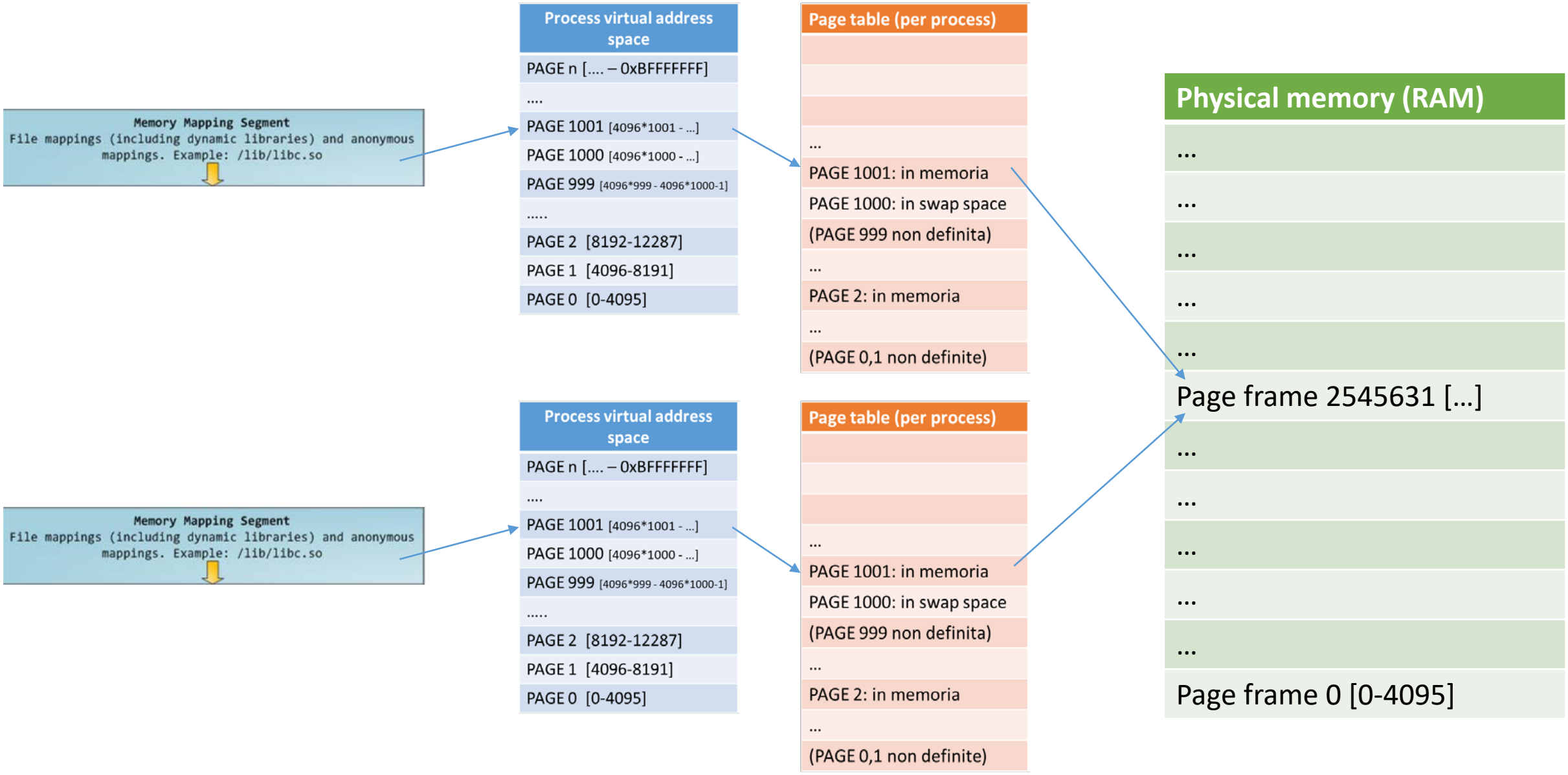
(ipotesi che sia stata fatta la copia dei page frame)

processo figlio

Process virtual address space
PAGE n [.... – 0xBFFFFFFF]
....
PAGE 1001 [4096*1001 - ...]
PAGE 1000 [4096*1000 - ...]
PAGE 999 [4096*999 - 4096*1000-1]
.....
PAGE 2 [8192-12287]
PAGE 1 [4096-8191]
PAGE 0 [0-4095]

Page table (per process)
...
PAGE 1001: in memoria
PAGE 1000: in swap space
(PAGE 999 non definita)
...
PAGE 2: in memoria
...
(PAGE 0,1 non definite)

Physical memory (RAM)
...
...
Page frame 3123498 [...]
...
...
Page frame 1237462 [...]
...
...
Page frame 134984 [134984 * 4096 - ...]
...
Page frame 99454 [...]
...
...
...
Page frame 0 [0-4095]



memory map (mappa di memoria) condivisa tra due processi (dopo fork())

Ipotesi: il memory map segment è contenuto tutto in una singola page (PAGE 1001)