memoria virtuale

capitolo 9 del libro (VII ed.), 12.6

Cristina Baroglio

Introduzione

- Nella descrizione delle tecniche di gestione della RAM abbiamo glissato sul problema del caricamento parziale dei processi, lasciando intendere (non troppo esplicitamente) che i processi fossero sempre caricati interamente
- è utile rilasciare questo vincolo?

migliore gestione della RAM?

miglioramento della vita del programmatore?

miglioramento della vita dell'utente dei programmi?

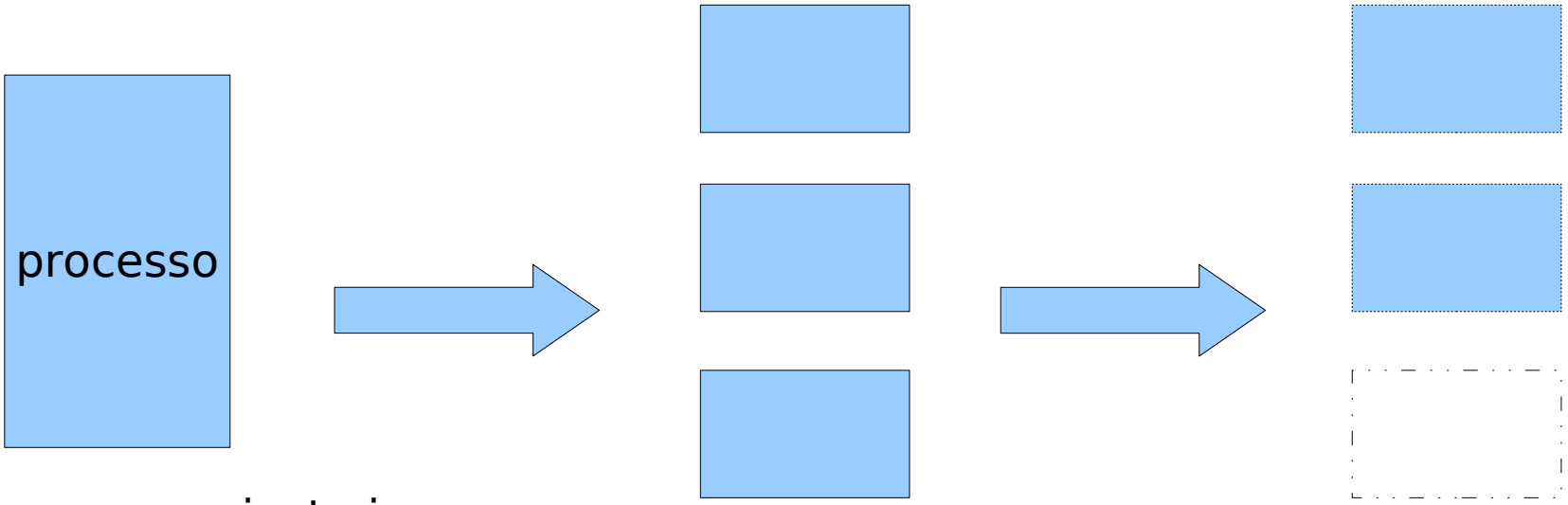
Introduzione

- Nella descrizione delle tecniche di gestione della RAM abbiamo glissato sul problema del caricamento parziale dei processi, lasciando intendere (non troppo esplicitamente) che i processi fossero sempre caricati interamente
- è utile rilasciare questo vincolo?

Introduzione

- Nella descrizione delle tecniche di gestione della RAM abbiamo glissato sul problema del caricamento parziale dei processi, lasciando intendere (non troppo esplicitamente) che i processi fossero sempre caricati interamente
- è utile rilasciare questo vincolo?
- **Sì**:
  - Consente di creare programmi che (da soli o complessivamente) occupano più spazio di quanto disponibile in RAM
  - Si liberano i programmatori dai “vincoli di memoria”: posso portare un grosso programma su un computer con meno memoria e vederlo comunque funzionare
  - Si migliora l'uso della RAM: (1) molte procedure vengono usate in circostanze rare, perché caricarle sempre? (2) spesso array e matrici sono sovradimensionati, perché non aggiungere spazio solo se serve davvero?
  - Riducendo la RAM necessaria a ciascun processo, posso eseguire molti più programmi contemporaneamente! Maggiore multiprogrammazione
  - meno tempo per fare swap
  - meno I/O per caricare i processi

Percorso



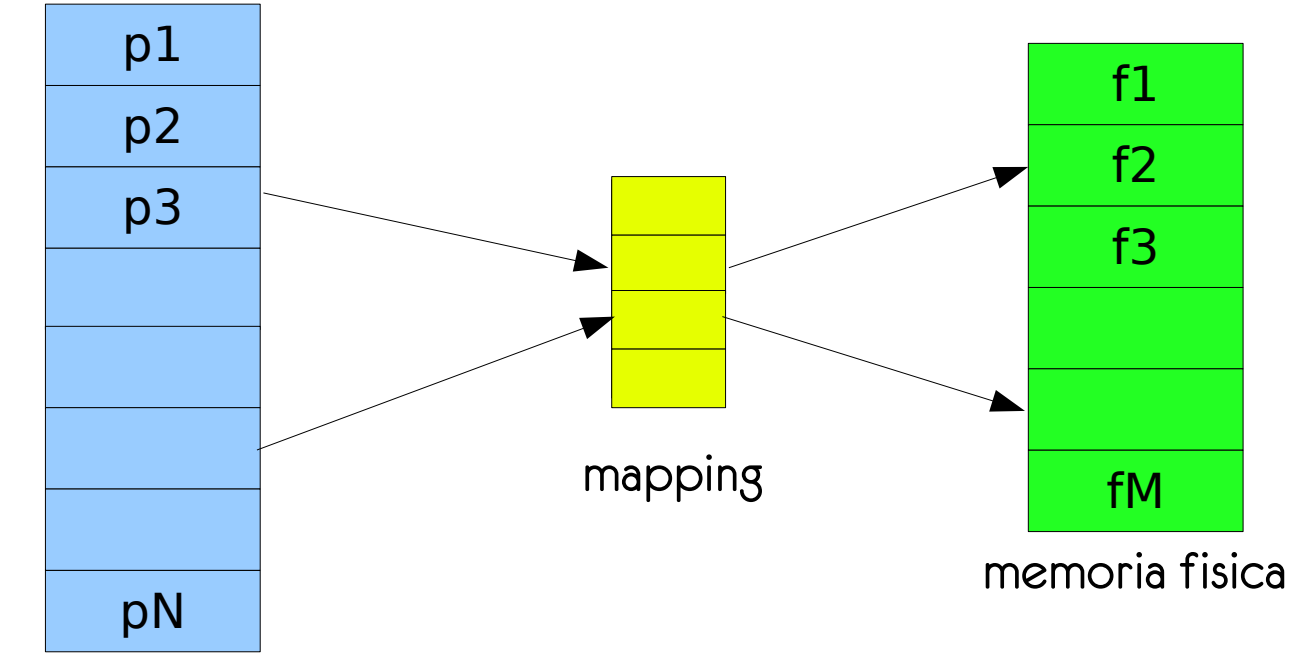
Processo caricato in RAM in modo **contiguo e totale**

Si rilascia il vincolo di contiguità:
- paginazione
- segmentazione

Si rilascia il vincolo di totalità:
- memoria virtuale

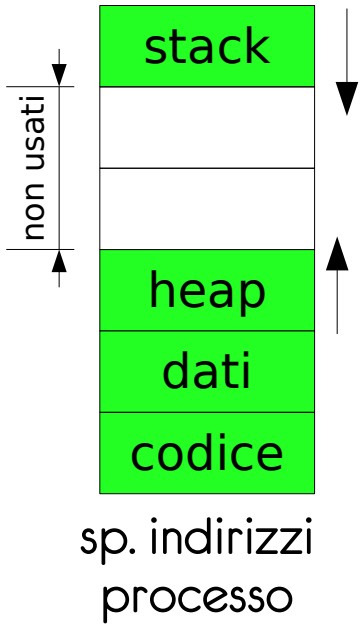
Memoria virtuale

- Il concetto di “memoria virtuale” nasce dalla separazione della memoria logica (la memoria come percepita dall'utente) dalla memoria fisica, a disposizione



si hanno i vantaggi già citati parlando della paginazione, fra le altre cose la condivisione di pagine (librerie/memoria condivisa)

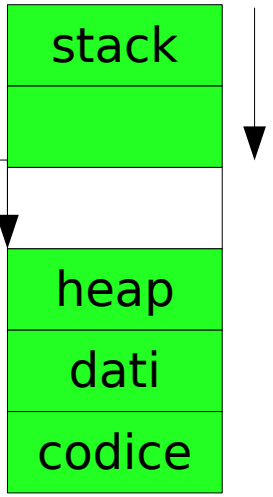
Spazio degli indirizzi di un processo



Lo spazio degli indirizzi di un processo è predefinito e, in genere, è molto più grande dello spazio realmente occupato dal processo

Con l'esecuzione stack e heap cresceranno l'uno verso l'altro. Per contenere i nuovi dati potranno essere allocati nuovi frame.

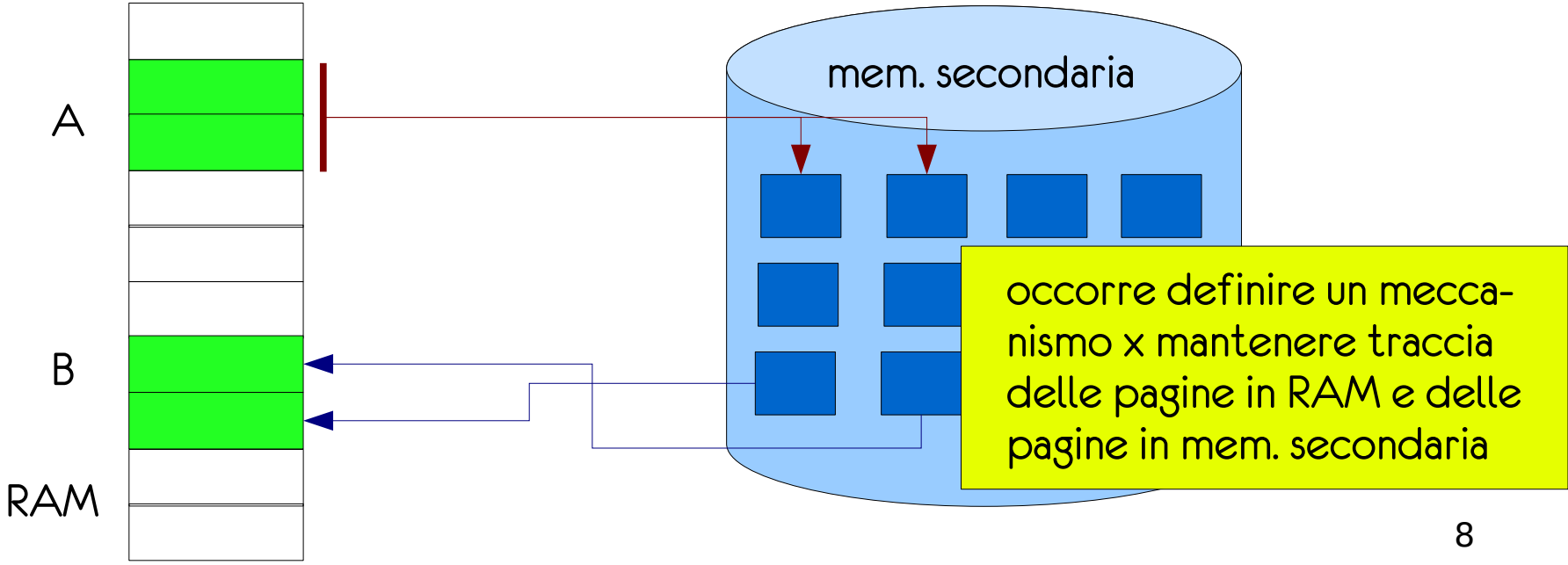
Questo non cambia lo spazio degli indirizzi virtuali del processo, semplicemente una parte di indirizzi prima non corrispondenti ad alcun indirizzo assoluto saranno ora mappati su parole di memoria effettive



Di qui in avanti pensiamo al processo come a un insieme di pagine

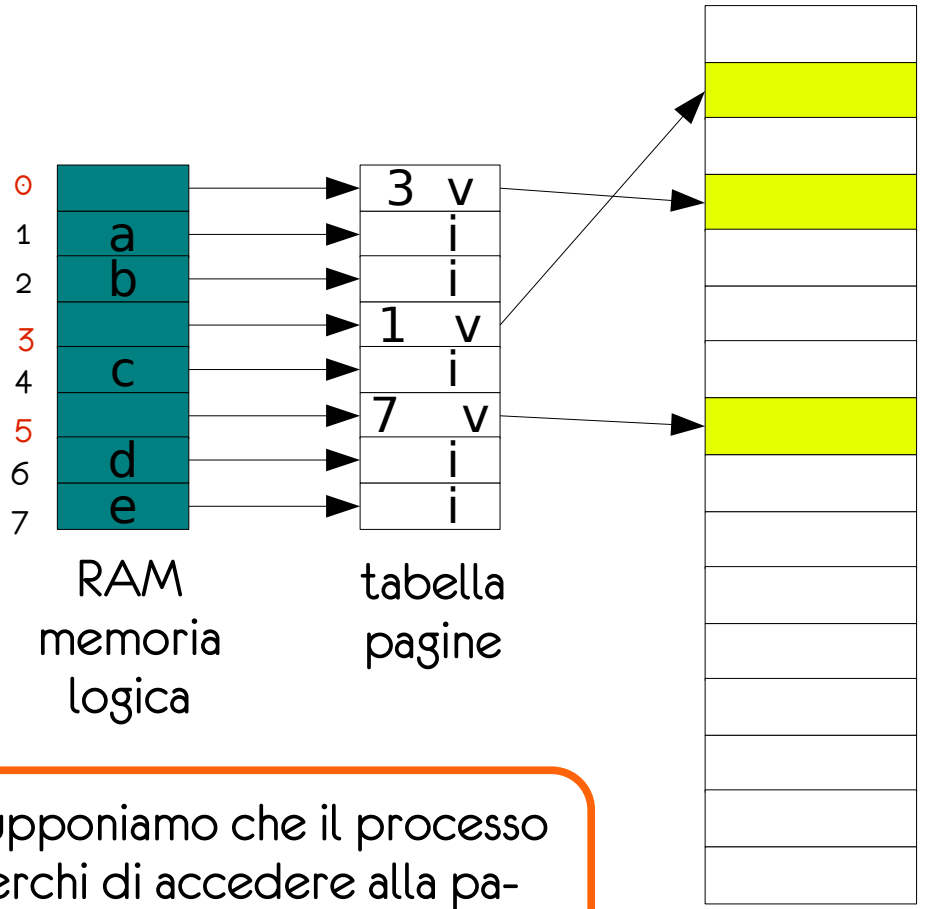
Demand paging - lazy swapping

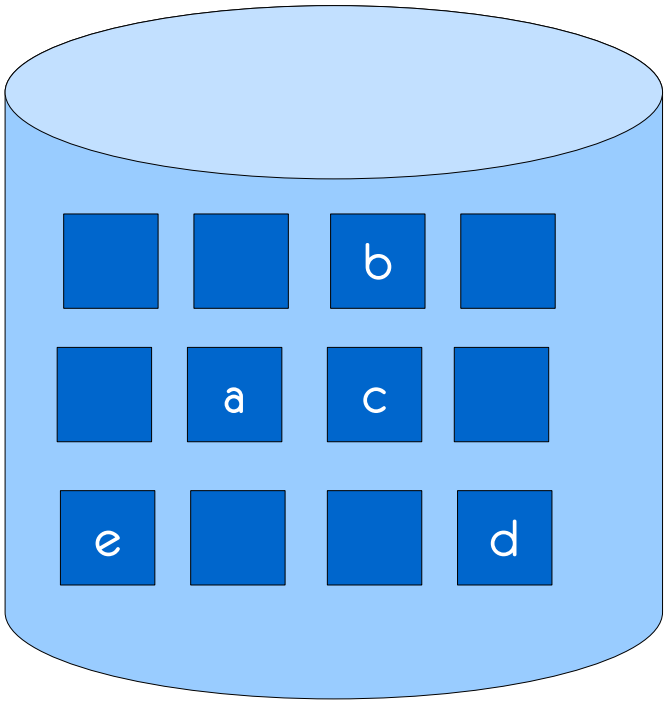
- Ricorda lo swapping ma riguarda porzioni di processo
- I processi vengono caricati in modo parziale:
  quando una pagina diventa utile (perché contiene una parte di codice da eseguire o dei dati da elaborare) la si carica avvicendandola con una pagina precedentemente in RAM
- Questo meccanismo è noto come **lazy swapping**
- La parte del SO che gestisce il caricamento delle pagine è detto **pager**



Bit di validità

nella tabella delle pagine di un processo si interpreta il bit di validità in questo modo: se è falso la pagina o appartiene a un altro spazio degli indirizzi oppure appartiene al processo ma al momento risiede in memoria secondaria





le pagine invalide sono in memoria secondaria

Supponiamo che il processo cerchi di accedere alla pagina "a" ...

Page fault

- Quando un processo cerca di accedere a una pagina che non valida si genera un interrupt (page fault exception)
- Il sistema accede a una tabella conservata nel PCB del processo per verificare se si tratta di una pagina del processo, ancora in memoria secondaria (altrimenti si tratta di una pagina invalida):
- **Se sì**:
  - individua un frame libero
  - è richiesta un'operazione di copiatura da disco della pagina desiderata
  - a lettura completata si aggiorna la tabella delle pagine
  - si riavvia l'operazione interrotta dall'interrupt e il processo riprende come se nulla fosse
- **Se no** si termina il processo

Page fault: esempio 1

- L'interruzione per page fault può occorrere in momenti diversi dell'esecuzione di un'istruzione
- Consideriamo per es. l'istruzione ADD A B (somma A e B memorizzando il risultato in qualche locazione C non menzionata nell'istruzione):
  - <1> page fault all'atto del caricamento dell'istruzione
  - <2> page fault quando si cerca di accedere a un operando
  - <3> page fault quando si deve salvare il risultato in C
- Cosa significa “riavviare l'istruzione” in questi tre casi?
- In generale l'interruzione del ciclo fetch-decode-execute causa il riavvio del ciclo stesso per l'istruzione interrotta

Page fault: esempio 1

- <1> page fault all'atto del caricamento dell'istruzione:
  - l'istruzione da eseguire non è in RAM: viene caricata la pagina contenente l'istruzione, si carica l'istruzione, la si esegue
- <2> page fault quando si cerca di accedere a un operando:
  - l'istruzione è già stata caricata e decodificata, siamo in fase di esecuzione: si carica la pagina mancante, si ricarica l'istruzione, la si esegue
- <3> page fault quando si deve salvare il risultato in C:
  - l'istruzione è già stata parzialmente eseguita: si carica la pagina mancante, si ricarica l'istruzione, la si riesegue (salvando il risultato)

NB: una stessa istruzione potrebbe causare diversi page fault

Paging on demand

- Anche detta “paginazione a richiesta” (o demand paging), è un meccanismo per cui una pagina viene caricata in RAM SSE è stata richiesta
- (ipotesi estrema) Un processo viene avviato *senza caricare* alcuna sua pagina
- Il caricamento della prima istruzione causa un page fault e quindi il caricamento della prima pagina, ecc.
- In generale, si tratta di una tecnica costosa? Ogni istruzione può causare diversi page fault, il tempo di esecuzione potrebbe moltiplicarsi a dismisura ...
- È una tecnica che richiede particolari supporti HW?

Paging on demand

- Le uniche strutture HW di supporto richieste sono quelle già viste per realizzare la paginazione e lo swapping (cap. 8)
- Riguardo l'esecuzione:
  - in generale vale il principio di località dei riferimenti (riprenderemo il concetto parlando di paginazione degenere)
  - proviamo però a fare un calcolo x valutare il peso della paginazione su richiesta sul tempo di accesso effettivo alle pagine:

$$\text{tempo di accesso effettivo} = (1-p) * ma + p * tgpf$$

    - dove *p* = probabilità che si verifichi un page fault, $p \in [0, 1]$
    - *ma* = tempo medio di accesso alla RAM $\in$ [10, 200] nsec
    - *tgpf* = tempo di gestione di un page fault, comprende il tempo di lettura e caricamento della pagina da memoria secondaria $\in$ 8msec

Gestione del page fault

- si genera un'interruzione
- salvataggio dei registri e dello stato del processo
- verifica dell'interruzione: in questo caso si determina che si tratta di page fault
- controllo della correttezza del riferimento (pagina invalida perché?) e individuazione della locazione occupata dalla pagina mancante su disco
- lettura e copiatura della pagina da memoria secondaria in RAM (operazione di I/O)
- durante l'attesa per il completamento di questa operazione, allocazione della CPU a un altro processo
- interrupt che segnala il completamento dell'operazione di caricamento della pagina
- aggiornamento della tabella delle pagine
- quando lo scheduling riavvia il processo sospeso, ripristino dello stato
- riesecuzione dell'istruzione interrotta

Gestione del page fault

- si ge
- salva
- verifi na che si tratta di page fault
- contr nvalida perché?) e individuazione della locazione occupata dalla pagina mancante su disco
- lettura e copiatura della pagina da memoria secondaria in RAM (operazione di I/O)
- durante l'at della CPU a un altro pro
- interrupt ch della pagina
- aggiorname
- quando lo scheduling riavvia il processo sospeso, ripristino dello stato
- riese ta

Ordini di Grandezza

1 secondo	10^{0}	unità di misura
1 millisecondo	10^{-3}	
1 microsecondo	10^{-6}	
1 nanosecondo	10^{-9}	

più in breve possiamo riassumere queste operazioni in

<1> servizio di interruzione per page fault
<2> lettura della pagina
<3> riavvio del processo

richiede circa 8 millisecondi

dominante

richiedono da 1 a 100 microsecondi

Un po' di numeri ...

tempo di accesso effettivo = *(1-p) * ma + p * tgpf*

ma = 200 nsec

tgpf = 8 msec

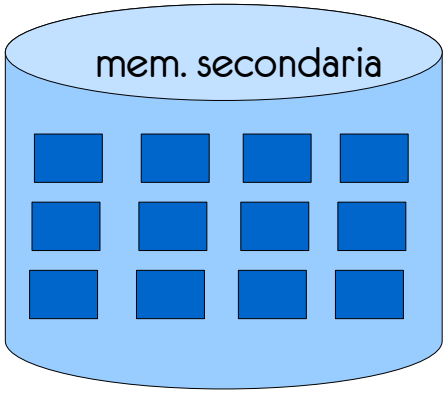
t. a. e. = *(1-p) * 200 + p * 8.000.000*
= *200 – p*200 + p * 8.000.000 =*
= *200 + p * 7.999.800*

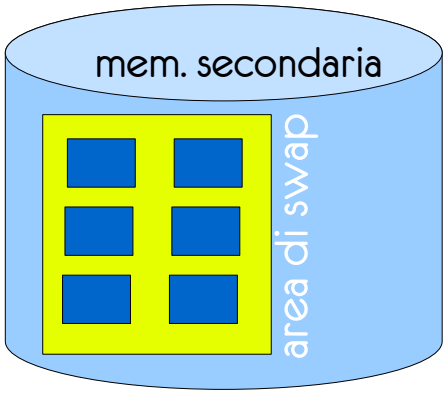
supponiamo di avere un page fault ogni 1000 accessi: p = 0.001

t. a. e. = *200 + 0.001 * 7.999.800 = 200 + 7.999,8 ≈ 8.000 nsec*
(8 microsec)

Area di swap

- Abbiamo detto che le pagine di un processo che non sono in RAM sono contenute in memoria secondaria (backing store)
- In particolare, tali pagine sono conservate in una porzione speciale della memoria secondaria, detta area di swap





- L'area di swap è vista come un'estensione della RAM, anche se i tempi di accesso sono molto maggiori

Area di swap

- La gestione dell'area di swap varia molto da SO a SO sia per quel che riguarda la sua implementazione sia per quel che riguarda il suo dimensionamento sia per quel che riguarda i suoi contenuti
- dimensionamento
  - Linux: suggerisce di allocare il doppio della quantità di RAM a disposizione
    - es. se ho ½ GB di RAM -> riservo 1GB di area di swap
  - Solaris: suggerisce di allocare una quantità pari alla differenza fra la dimensione dello spazio degli indirizzi logici e la dimensione della RAM
    - es. 2^{32}, sp. ind. indirizzi logici, 2^{10} spazio di RAM, $2^{32} - 2^{10}$ dimensione dell'area di swap
- **Consiglio generale**: *melius abundare quam deficere*, se l'area di swap si esaurisce il SO dovrà terminare qualche processo per liberare memoria

Area di swap

- Implementazione
  - come file: l'area di swap è un file speciale del file system
    - pro
      - si evita il problema del dimensionamento, un file cresce/ diminuisce a seconda delle necessità
    - contro
      - la gestione del file system introduce delle sovrastrutture che rallentano ulteriormente l'accesso
  - come partizione a sè, non formattata
    - si usa un gestore speciale che manipola direttamente pagine e adotta algoritmi ottimizzati per ridurre i tempi di accesso
      - pro: maggiore velocità
      - contro: per ridimensionarla occorre ripartizionare il disco

Area di swap

- Contenuti
- SO di qualche anno fa mantenevano nell'area di swap pagine di codice e pagine di dati ...
- ... però dato che le pagine di codice non sono modificate dall'esecuzione dei programmi e i tempi di accesso all'area di swap non sono poi di molto inferiori a quelli di accesso al resto del disco ...
- ... oggi si preferisce mantenere nell'area di swap per lo più pagine di dati (stack/heap dei processi) prelevando le pagine di codice direttamente dal file system
- Esempi di SO che adottano questa tecnica: Solaris e BSD

processi padri e figli

capitolo 9 del libro (VII ed.)

Processi padri e figli

processo P

esegue fork

processo F: figlio

pag. codice 1

...

pag. codice N

pag. dati 1

...

pag. dati M

(1) in RAM

pag. codice 1

...

pag. codice N

pag. dati 1

...

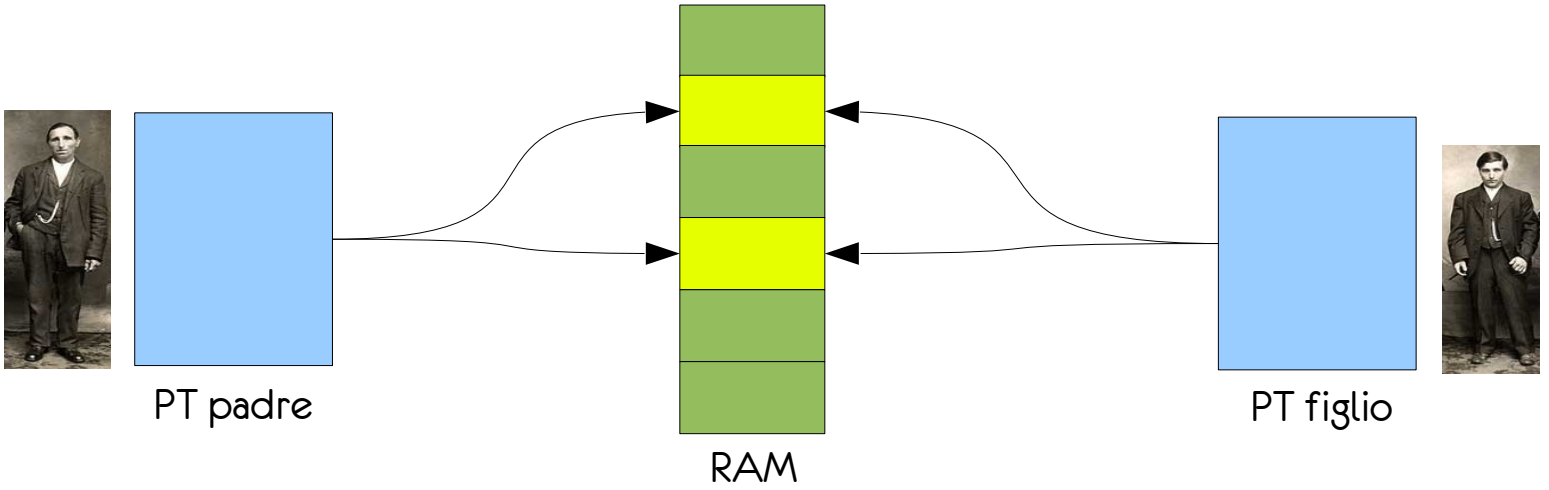
pag. dati M

esegue lo stesso codice del padre

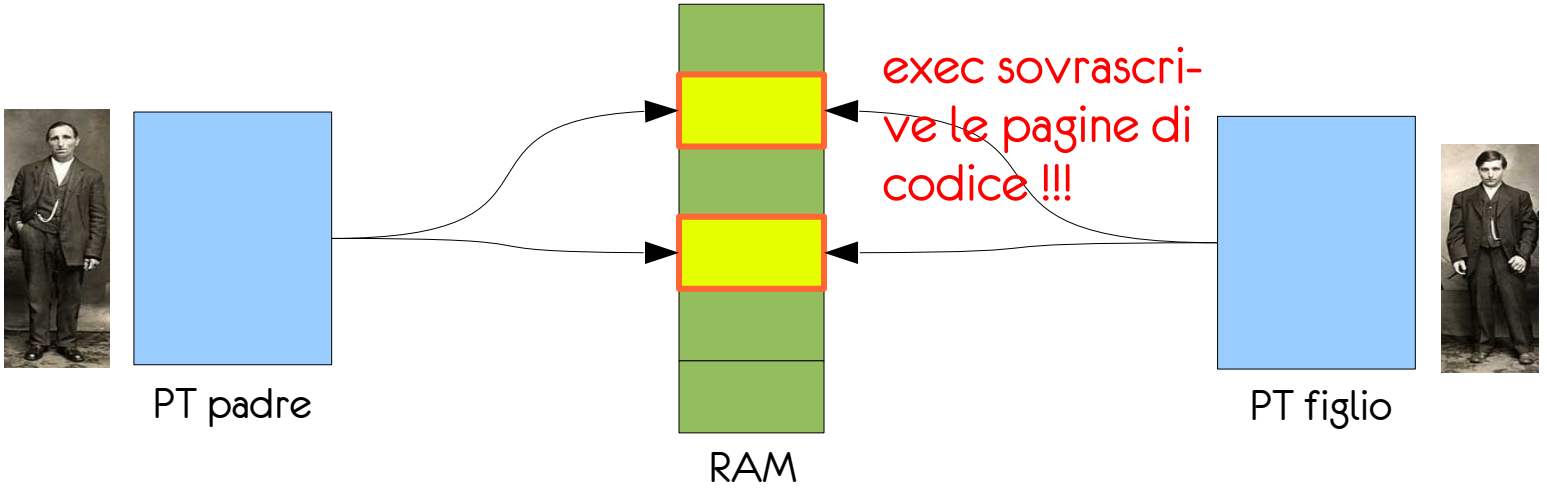
ha una copia delle vrb. del padre e dello stack

(2) ha però la possibilità (sfruttata molto di frequente) di cambiare il codice da eseguire tramite la system call "exec": l'effetto è una sovrascrittura delle pagine di codice del processo

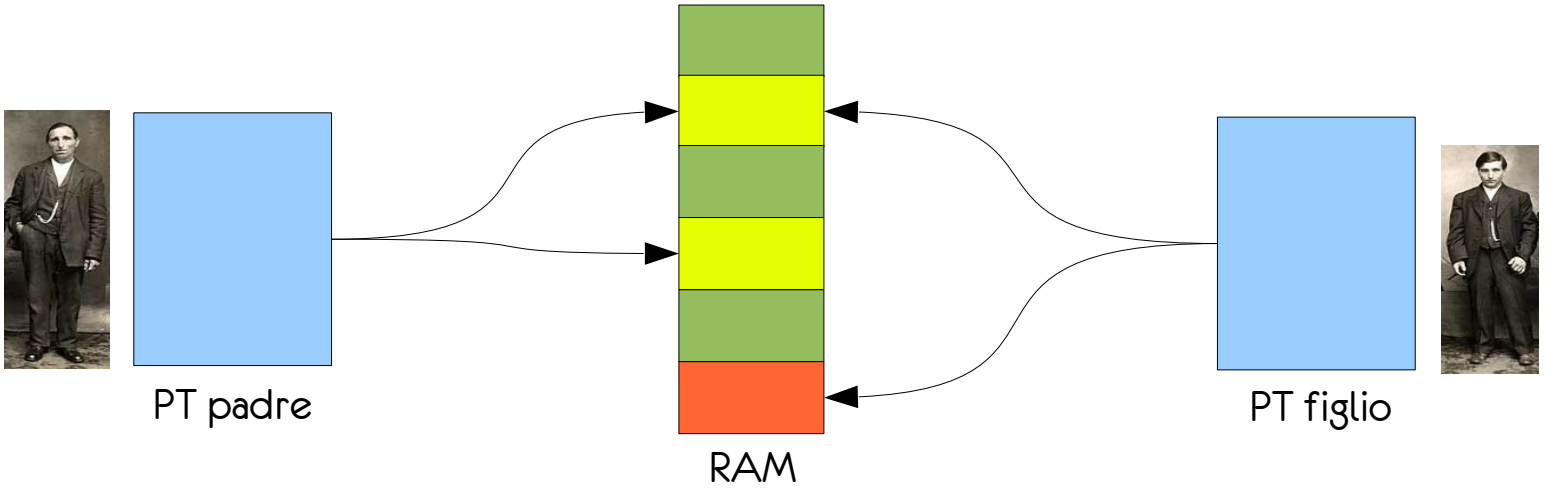
Fork, exec e paginazione



(1) per ovviare al problema della duplicazione delle pagine di codice del processo padre e del processo figlio, è possibile far condividere tali pagine ai 2 processi



Copiatura su scrittura



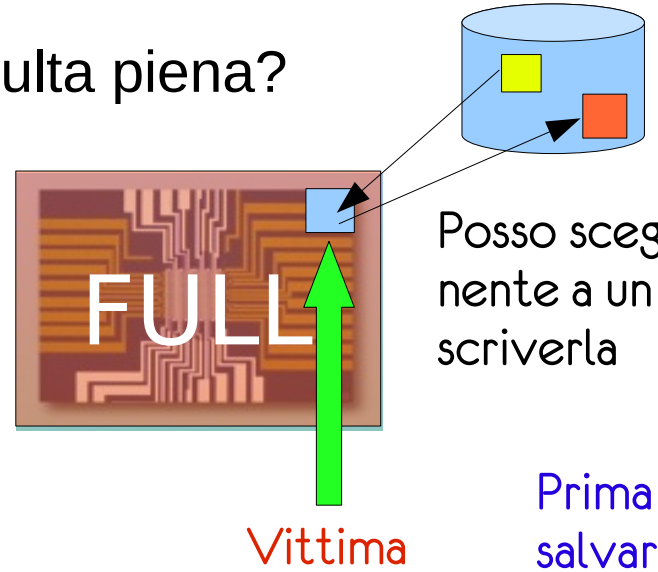
(2) per ovviare al problema della sovrascrittura delle pagine di codice del processo padre si adotta la tecnica di "copiatura su scrittura": quando uno dei due processi esegue una "exec" si allocano delle nuove pagine per il processo chiamante e si carica il nuovo codice in esse

Le ragioni del nome "copiatura su scrittura" derivano dal caso (più generale) in cui un processo deve modificare solo in parte il contenuto di una pagina condivisa. In questo caso la pagina viene prima copiata e poi si consente la modifica della copia.

Es. si possono far condividere ai due processi anche stack e heap, almeno inizialmente. Quando uno dei due cerca di modificare una porzione (es.) di stack si duplica solo la pagina coinvolta

Sostituzione di pagine

- La paginazione come tecnica di realizzazione della memoria virtuale aumenta il livello di multi-programmazione dell'elaboratore
- Abbiamo affrontato il problema del page fault ...
- ... ma non ci siamo ancora posti il problema se, in occorrenza di un page fault, sia sempre possibile individuare un frame libero in cui caricare la pagina richiesta ...
- Cosa fare se la RAM risulta piena?



Posso scegliere una pagina appartenente a un processo in RAM e sovrascriverla

Prima di sovrascrivere occorre salvare la vittima in memoria secondaria

Dirty bit

- Il meccanismo descritto richiede la copiatura di due pagine: la vittima sarà copiata in memoria secondaria, la nuova pagina sarà copiata in RAM
- per ridurre l'inefficienza comportata da questa duplice scrittura, si cerca di limitarla ai casi in cui è effettivamente necessaria:
  - la pagina nuova va necessariamente copiata in RAM
  - ma è sempre necessario copiare la vittima in memoria secondaria?
- È necessario solo se la pagina è stata modificata rispetto a una sua copia già conservata su disco
- Basta mantenere un bit (dirty bit) per ogni pagina: il bit viene settato non appena la pagina è modificata
- if (! dirty bit) → la pagina non va ricopiata

Commenti

- Paginazione:
  - memoria logica e memoria fisica completamente separate
- i processi hanno a disposizione uno spazio degli indirizzi più grande di quello fisico offerto dalla RAM
- realizzazione di un meccanismo dinamico tramite il quale caricare / sostituire pagine a seconda delle esigenze di esecuzione
- introduzione di meccanismi finalizzati ad aumentare l'efficienza, contrastando in parte gli appesantimenti imposti dalle moltiplicate letture/copiature di pagine
- incremento del livello di multiprogrammazione