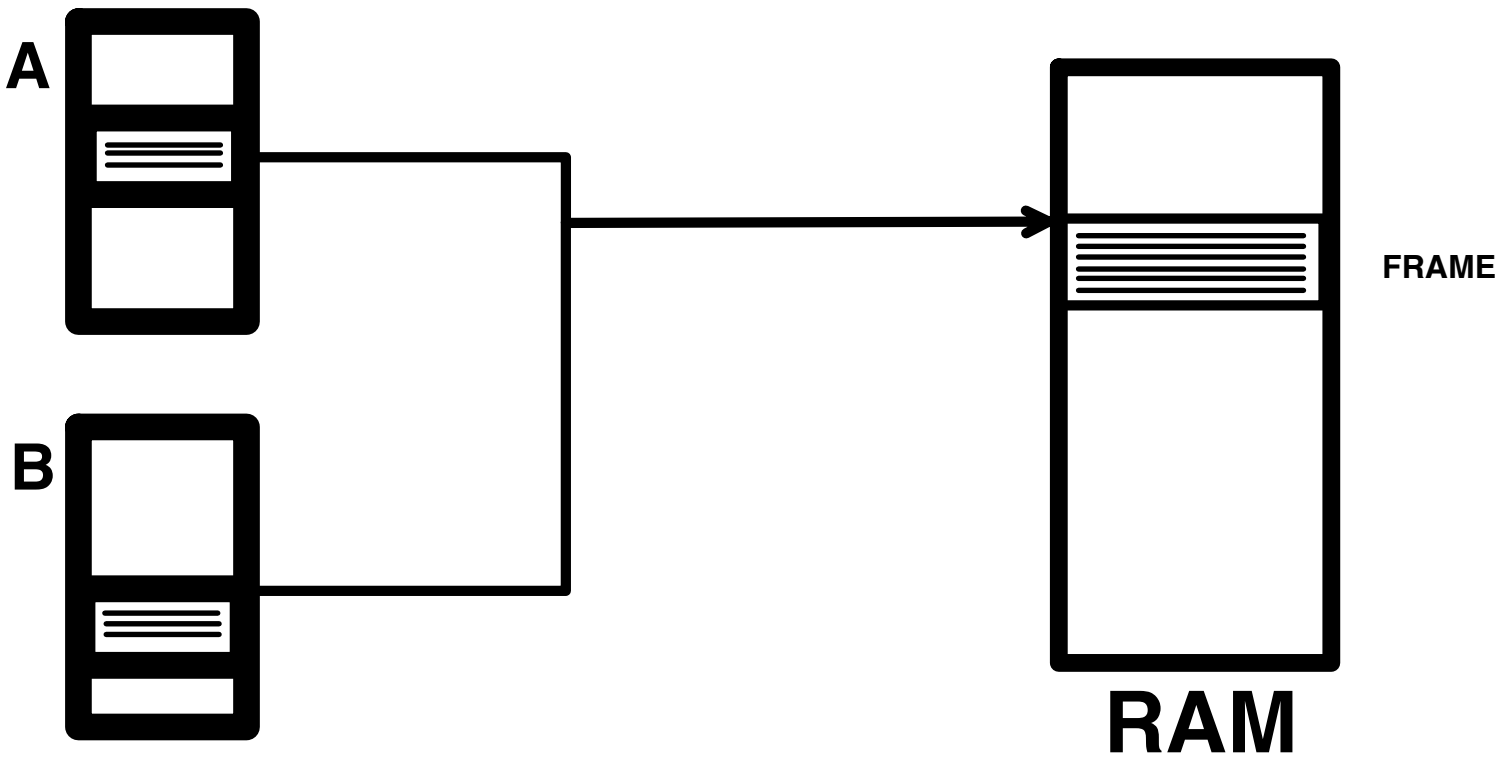
Condivisione

lunedì 15 maggio 2023 16:03

Condivisione

Si parla di condivisione della memoria fisica. Cosa significa condividere la memoria principale?

**Supponiamo di avere un address space A di un'applicazione attiva, e un address space B di un'applicazione attiva.
Non stiamo indicando la presenza di segmenti dentro i rispettivi address space, non importa, abbiamo rappresentato l'address space con la rappresentazione lineare, tanto comunque questo oggetto lineare è una sequenza di pagine logiche che sono collocate in memoria fisica.
Supponiamo di avere una pagina logica all'interno dell'address space A e una pagina logica all'interno dell'address space B.
La PageTable di A e la PageTable di B possono indicarci che, per esempio, in A la collocazione in RAM di quella pagina logica è esattamente un dato frame.
La PageTable di B ci può dire che la collocazione in memoria fisica di quella pagina logica è anche lei quel frame?**



**Se questa situazione è vera noi stiamo dicendo che, di questi due address space abbiamo che la pagina logica del processo A e la pagina logica del processo B, condividono lo stesso identico frame per essere rappresentate in memoria fisica.
Questa cosa si fa semplicemente strutturando le tabelle delle pagine in maniera tale che ci rimandino esattamente sullo stesso frame.
Non è un problema. Non ci sono particolari problemi in questo quando passiamo attraverso la paginazione.
La paginazione, per ogni pagina logica ci dice qual è il corrispettivo frame fisico di dove è presente; E se abbiamo due tabelle delle pagine, queste due tabelle, in una certa pagina logica, possono portarla ad essere rappresentabile all'interno dello stesso frame.
Questa si chiama CONDIVISIONE.
Essa è particolarmente interessante quando noi abbiamo un processo A (supponendo che sia un editor di file) che ha l'address space composto da 3 pagine:**

Ed 1
Ed 2
Dati 1

1
3
4

Pagina zeresima, prima e seconda. Le prime due pagine mantengono il codice dell'editor, ossia lì abbiamo il software che deve essere semplicemente letto e mandato in esecuzione.
L'address space è mappato nel frame numero 1 la pagina zeresima, nel frame numero 3 la pagina prima e nel frame numero 4 la pagina seconda.

Supponiamo di voler lanciare un'altra istanza di questa stessa applicazione, quindi un'altra istanza di questo stesso editor.

Ed 1
Ed 2
Dati 2

1
3
6

In quest'altro address space avremo lo stesso codice da mandare in esercizio che occupa le stesse pagine, e poi i dati per quest'altra istanza di applicazione.

Memoria principale

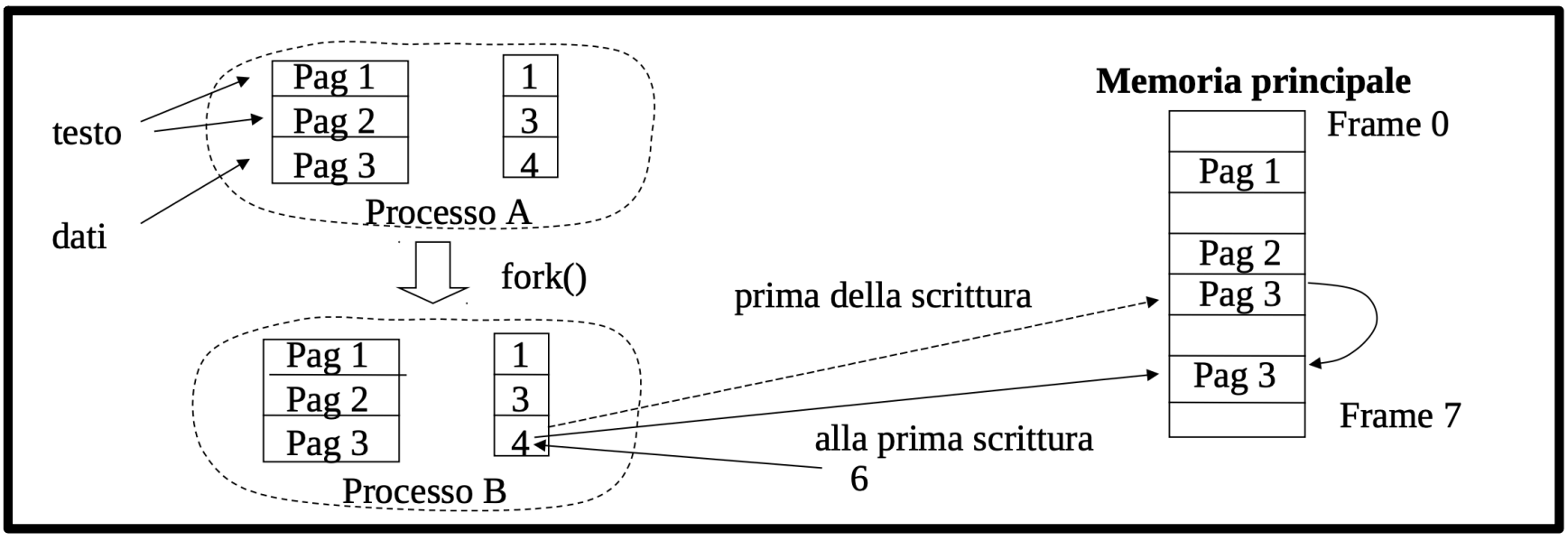
	Frame 0
Ed 1	
Ed 2	
Dati 1	
Dati2	
	Frame 7

La pagina ED 1 nella seconda applicazione è mappata anch'essa su un frame dove ospitavamo la ED 1 del processo precedente, e la ED 2 mappata esattamente sullo stesso frame dove ospitavamo la ED 2 del processo precedente.

- ulteriori vantaggi della paginazione e della segmentazione consistono nella possibilità di condividere codice di uso comune (es. text editors)
- ciascun processo possiede dati privati, caricati su frame o partizioni di memoria proprie
- il testo viene invece caricato su pagine o partizioni comuni

- il binding a tempo di esecuzione supporta la condivisione in modo automatico senza costi addizionali

Lo scenario in cui noi abbiamo un'applicazione e poi un'altra, la cui struttura interna è identica a quella dell'applicazione originale, la andiamo ad ottenere quando noi lanciamo una fork().
Quando su UNIX abbiamo un'applicazione attiva con 3 pagine, ad esempio, all'interno dell'address space di cui due di testo e una di dati, e poi su questa applicazione generiamo una fork() che prevede che venga generato un address space clone identico a quello originale, le pagine del processo figlio le possiamo collocare dove erano originariamente le pagine del processo parent? La risposta è SI!



Quando noi effettuiamo una fork() su UNIX, tiriamo fuori una NUOVA PAGE TABLE.
Prima della fork() avevamo una page table per il processo originale, e una volta eseguita abbiamo una nuova page table per il figlio che ci dice che i dati da usare sono gli stessi che avevamo in memoria.
MA GLI STESSI SINO AD UN CERTO PUNTO, perché i due processi sono comunque indipendenti, perché all'interno di questi due address space ci possono essere cose che loro vogliono anche scrivere, questa cosa viene risolta utilizzando all'interno delle PAGE TABLE un BIT DI PROTEZIONE che si chiama "COPY ON WRITE".
Esso va ad indicare, sia nella page table del parent che nella page table del child, che magari se il processo figlio vuole scrivere nella pagina 3 e magari il processo padre vuole continuare solo a leggere dalla pagina 3, originariamente questa pagina 3 era nel frame numero 4 sia per il parent che per il child, ma se il child vuole cominciare a scrivere all'interno di questa pagina NON DEVE andare ad impattare le informazioni visibili al parent.
Quindi appena qualcuno prova a scrivere (quindi il child) il sistema operativo prende il controllo, copia la pagina 3 da un'altra parte (ne fa una copia privata per il processo B) e aggiorna la entry della page table associata alla pag. 3 del child, per puntare ad un altro frame di memoria.
Quindi i due processi condividono il condivisibile, quindi le due pagine dove noi manteniamo il testo, ma sui dati si possono generare copie private all'interno della RAM.
Aldilà del fatto che noi stiamo condividendo queste informazioni all'interno della RAM, comunque i due address space sono logicamente separati, abbiamo comunque due page table differenti e quando si lavoro nel processo B si lavora con la sua page table, stessa cosa per A.

- singole pagine o segmenti possono essere protetti rispetto a specifiche modalità di accesso (lettura/scrittura/esecuzione)
- la tabella delle pagine (o dei segmenti) contiene bit addizionali per determinare il tipo di protezione
- protezione **copy on write**: la pagina o il segmento è condiviso fino alla prima scrittura (supporto efficiente per i meccanismi di duplicazione di processo, es. *fork()* UNIX)

Il primo che scrive si fa la sua copia privata risolve il problema dell'andare a condividere cose che, in qualche modo, non possono essere condivisibili perché qualcuno vuole aggiornarle e qualcun altro vuole continuare a lavorare con le informazioni originali che erano disponibili.