Architettura degli Elaboratori, 2007-08

Trattazione semplificata del Metodo a Capability per l'allocazione dinamica di oggetti condivisi

Nel Cap. VIII, sez. 3.2 è introdotto il problema della *condivisione di oggetti mediante riferimenti indiretti* (cioè, dei "puntatori condivisi"). Per risolvere questo problema, nella sez. 3.2.1 sono esposti tre *metodi statici*, e nella sez. 3.2.2 è introdotto un *metodo dinamico*, detto **metodo a capability**. Questo è trattato in maniera approfondita nella sez. 4, partendo da una base teorica dei sistemi di protezione. In queste note è fornita una trattazione semplificata del metodo a capability, in modo da sganciarne lo studio da quello della protezione. Il risultato, in termini di implementazione e modalità di utilizzazione, è identico.

La tecnica che cerchiamo permette di *allocare dinamicamente oggetti condivisi nella memoria virtuale di processi*. Questa tecnica rende più generale ed efficiente, rispetto ai metodi statici, la soluzione al problema della condivisione di oggetti mediante riferimenti indiretti. Esempi di applicazione sono:

- condivisione di tutti i PCB da parte di tutti i processi,
- condivisione, da parte del processo mittente, della variabile targa nell'implementazione di comunicazioni asincrone (e condivisione, da parte del processo destinatario, del messaggio nella comunicazione sincrona alla pari),
- condivisione, da parte di processi-servizi, di canali in cui inviare, ai processi applicativi, informazioni elaborate ("risposte") in funzione di loro precedenti comunicazioni ("richieste"),
- condivisione, da parte del processo gestore della memoria, delle Tabelle di Rilocazione di tutti i processi creati.

In tutti questi casi, e comunque in qualunque caso di interesse, la situazione è la seguente:

> siano due processi A e B che condividono staticamente una struttura dati S. In un certo istante, A possiede un certo oggetto X nella sua memoria virtuale MV_A, mentre B non lo possiede. Il meccanismo che cerchiamo deve far sì che X diventi dinamicamente condiviso da B, cioè che anche B allochi dinamicamente lo stesso oggetto X nella sua memoria virtuale MV_B. B deve quindi essere messo in grado di indirizzare X correttamente mediante propri indirizzi logici, in modo tale che la loro traduzione fornisca come risultato gli indirizzi fisici di X. Successivamente, quando B avrà utilizzato X, potrà deallocarlo da MV_B ritornando allo stato di partenza.

In termini intuitivi, l'idea è la seguente:

> A sa come rilocare i propri indirizzi logici di X, di conseguenza può aiutare B così: "scegli pure tu il tuo indirizzo logico di X, io ti dico come tradurlo".

Il meccanismo è schematizzato nelle Fig. 1- 4, supponendo che l'oggetto condiviso X sia ampio (al più) *una pagina*:

1. in Fig. 1 sono schematizzate le memorie virtuali di A e B prima del passaggio dinamico, a livello di dettaglio di pagine logiche. Sia *ipl-AX* l'identificatore di pagina logica di X nello spazio di indirizzamento di A. MV_B va letta come non contenente X in nessuna pagina logica o, equivalentemente, "B non conosce X", cioè B non consosce alcun identificatore di pagina logica con cui riferire X. MV_B contiene un certo numero di pagine "libere", cioè non ancora allocate;
2. in Fig. 2 sono schematizzate le Tabelle di Rilocazione di A e B. In particolare, $TABRIL_B$ contiene un certo numero di entrate non utilizzate ("libere"); usiamo una variabile *free* locale a B per indicare la prima posizione "libera" (non utilizzata) di TABRILB. L'entrata della Tabella di Rilocazione relativa alla pagina contenente X è detta convenzionalmente **capability di X**, indicata con *CAP-X*. Nella stessa Fig. 2 è mostrato che il processo A, quando intende metter B in grado di acquisire X, *copia CAP-X in un campo ben definito della struttura condivisa S*;
3. in Fig. 3 è mostrato che B, quando vuole acquisire X in MV_B, legge CAP-X da S (precedenti azioni hanno permesso di essere certi che S contiene il valore aggiornato di CAP-X) e la copia nella posizione *free* di $TABRIL_B$;
4. come mostrato in Fig. 4, *da questo momento X appartiene anche a* MV_B, e *l'identificatore di pagina logica con cui B può riferire X* è *ipl-BX = free*. Un qualunque indirizzo logico di X avrà questo valore nel campo IPL e displacement noto dall'uso che deve essere fatto di X (ad esempio, la base di X ha displacement uguale a zero).

Incrementando il valore di *free*, B si predispone ad allocare dinamicamente ulteriori pagine in MV_B.

La deallocazione di X da MV_B comporta semplicemente il decremento di *free* e, per sicurezza mettere il bit di presenza a zero ed annullare tutti i diritti nell'entrata di $TABRIL_B$.

Nel Cap. VIII, sez. 4.6, è mostrata l'implementazione in D-RISC delle operazioni su capability.

Se più pagine sono allocate dinamicamente in tempi diversi, la deallocazione comporta una gestione opportuna dello stato di allocazione delle pagine. In molti casi di interesse, gli oggetti vengono allocati e deallocati dinamicamente secondo semplici schemi del tipo:

alloca (X1), dealloca (X1), alloca (X2), dealloca (X2), …

alloca (X1), alloca (X2), …, dealloca (X2), dealloca (X1)

Il meccanismo si generalizza al caso di *oggetti memorizzati su più di una pagina*, implementando CAP-X come più entrate della Tabella di Rilocazione corrispondenti a X.

Inoltre, all'atto del passaggio della capability (copia di CAP-X da $TABRIL_A$ in S), il processo A può *modificare i diritti di protezione* su X in modo che rispondano alle operazioni che su X dovrà eseguire B (in generale, diverse da quelle che vi esegue A).

Il metodo a capability ha portata del tutto generale, ed è quindi usato anche per allocare dinamicamente strutture *private* nella MV di uno stesso processo, ad esempio per implementare comandi, come *new* o *malloc*, presenti in linguaggi di programmazione.

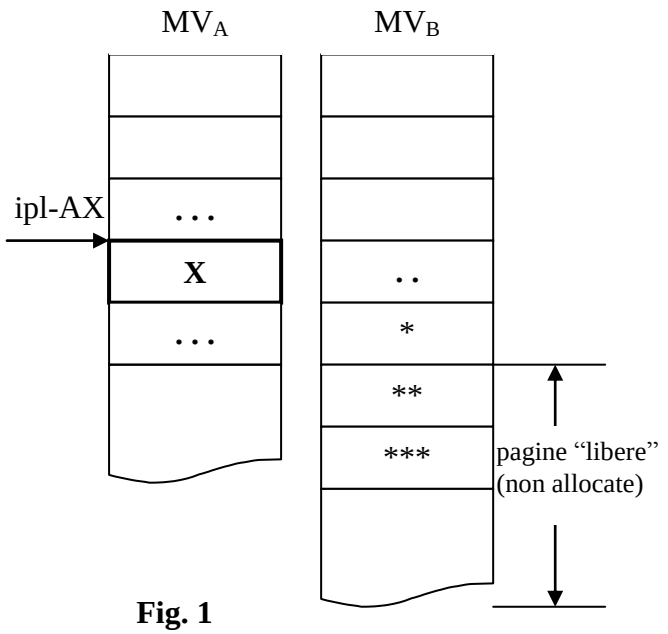

Fig. 1

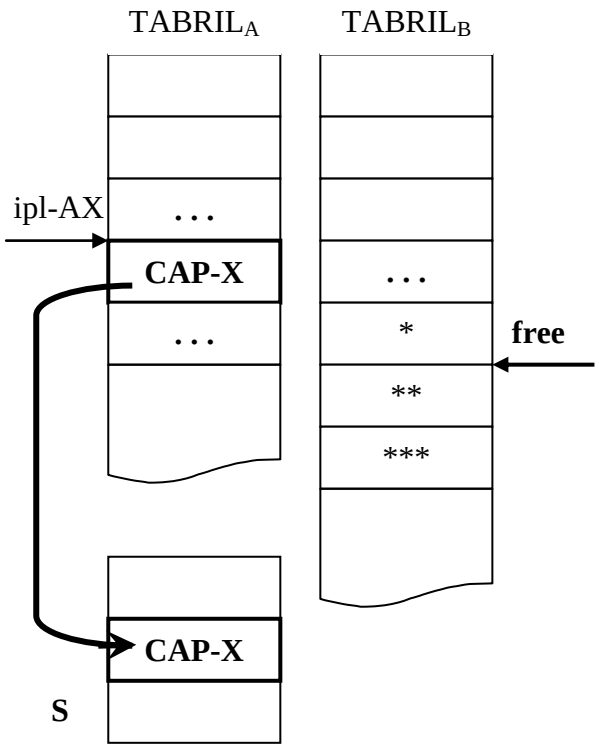

Fig. 2

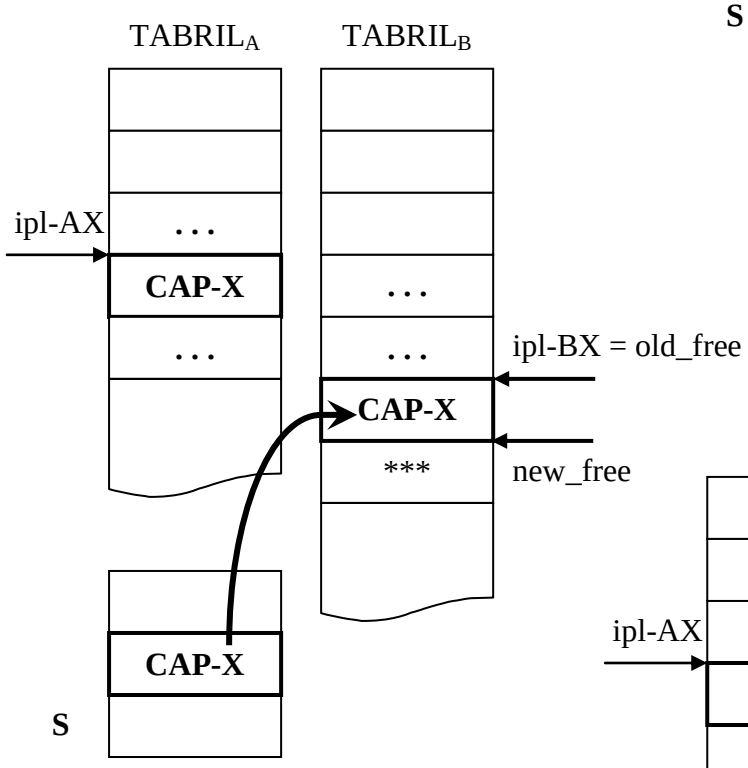

Fig. 3

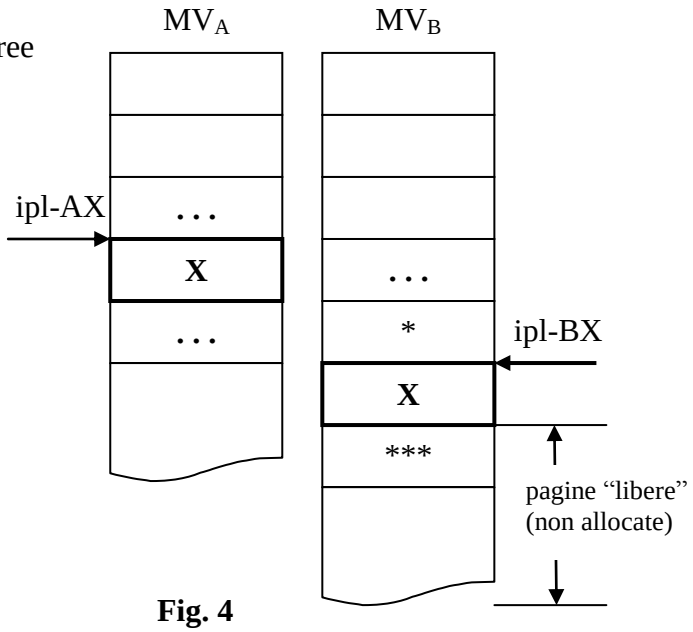

Fig. 4