Notes Software Engineering

Brendon Mendicino

February 28, 2023

Contents

1 Memoria

La memoria si compone si suddivide in:

- registri;
- cache
- memoria primaria;
- memoria secondaria;

Uno principali problemi di accesso alla memoria è quello di assicurarne la sua protezione, ovvero evitare che un programma in memoria riesca ad accedere alle zone di memoria di altri programmi. Una volta fatto il bootstrap del SO, sia esso che i programmi verranno caricati in memoria, per evetire che ogni processo abbia una vista al di fuori del suo scope, si possono usare dei controlli a livello della CPU, esistono dei registri chimati **base** e **limit**, che attraverso dei meccanismi, riescono ad effettuare la protezione della memoria. Esiste anche un altro problema che è quello **relocation**, consiste nel come mappare gli indirizzi delle varie istruzioni di salto e dei vari puntari una volta che il programma viene caricato in memoria, se si utilizzaro degli indirizzi statici (creati solo in fase di compilazione e prendondo come riferimento di inizio del programma l'indirizzo 0), una volta che questo viene caricato in memoria le istrizioni non punterebbero più alle label corrispondenti ma sempre nello stesso punto, dove ad esempio si trova un altro programma. Per questo motivo quando si utilizzano valori di registri vengono sommati al base register, mentre le boundry del programma in memoria vengono salvate nel limit. Binding ... Si fa:

- **in compilazione**: fatto quando molto semplice, ad esempio quando esistono solo due programmi;
- **in fase di load**: viene fatta la rilocazione durante il caricamento in memoria;
- **in esecuzione**: viene fatto il binding degli indirizzi in modo dinamico;

Per risolver questo problema ci si affida all'hardware, che è incaricato di fare la traduzione: l'indirizzo rimane lo stesso (logico) all'interno del processore, prima di arrivare all'address bus viene tradotto in indirizzo fisico, esiste dunque una dicotomia di indirizzo logico e fisico, in questo modo quando si scrive un programma, l'indirizzo parte sempre da 0. Esistono due tipi di indirizzamento che sono di tipo logico, dove l'intervallo degli indirizzi utilizzabili è logico, ed un indirizzamento fisico, dove il range è limitato dalla memoria del sistema. Per effettuare questa traduzione da indirizzo fisico a indirizzo logico e viceversa si utilizza una **MMU** (Memory Management Unit). Il modo più facile per realizzare una MMU, è quello di usare un **relocation register**, ovvero un registro che contiene il valore da aggiungere un indirizzo logico per fare

un indirizzo fisico, il modello più semplice di MMU è fatto da un sommatore ed un comparatore.

Per aumentare le prestazioni ed usare la memoria in modo più efficente si possono usare delle tecniche dinamiche.

- Si parla di **dynamic loading** quando, un programma viene caricato in memoria principale in modo frammentato, utilizzando solo i componenti che effettivamento vengono chiamati;
- Si parla di **dynamic linking** quando i file che contengono le funzioni che devono essere linkate (come le librerie standard) non vengono inserite all'interno dell'eseguibile, ma gli indirizzi vengono risolti in modo dinamico durante l'esecuzione;
- Il **link statico** è quando si crea un eseguibile con tutte le funzioni dentro, di fatto il loader carica tutto quando in memoria.

Il **dynamic loading** vuol dire che una routine non è caricata finché non è necessaria, questo può essere fatto quando il programmatore ne è consapevole, infatti il processo di load non è trasparente:

```
void myPrinf(**args) {
  static int loaded = 0;
  if (!loaded) {
    load("printf");
    loaded = 1;
  }
  printf(args);
}
```

Il **linker-assisted DL** si usa una chiamata fasulla che prima chiama la load, usando una **stub**.

Le **shared libraries** sono in grado di condividere le risorse, infatti se più processi utilizzano la stessa funzione essa viene messa a disposizione e per ogni nuova chiamata non ci sarà bisogno di chiamre una load.

Come si alloca memoria per un programma (immagine)? La più semplice è l'allocazoine contigua, dove si vede la RAM come due partizioni, una per il SO una per i processi (indirizzi più alti), per caricare un processo si parte da un indirizzo di inzio ed un indirizzo di fine, la MMU vista prima funziona solo con i casi di allocazione contigua (basilare). L'**allocazione contigua con paritizione variabile**: quando ci sono più buchi ci sono delle politiche differnti per inserire nuovi programmi:

- **first-fit**: il primo che si trova;
- **best-fit**: il buco con la dimensione più piccola;
- **worst-fit**: il buco can li dimesnione più grande;

La fragmentazione è sparsità dei buchi all'interno della memoria. Si dice **esterna** perché è al di fuori dei processi, si dice **interna** quando è interna al processo, ovvero che ha più memoria di quello che serve. La frammentazione ha bisongo di **compattazione**, il SO sposta i pezzi e poi si riparte.