Binding e modelli d’esecuzione

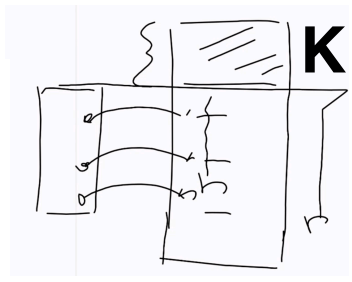
mercoledì 10 maggio 2023 12:00

Binding e modelli d’esecuzione

L'esecuzione seriale prevede che, quando io ho la RAM, all'interno di essa ci può essere solo un'applicazione, e quest'applicazione tipicamente può essere caricata a partire esattamente a partire dall'indirizzo 0 della RAM, e quando viene ad essere caricata in quel punto, nessun'altra applicazione può essere caricata all'interno della memoria di lavoro della mia macchina.
Questo era uno scenario in cui il binding a tempo di compilazione andava benissimo. Avevamo la possibilità di sapere che l'address space dell'applicazione veniva caricato in RAM a partire dall'indirizzo zero e sapevo esattamente qual'era la posizione di OGNI elemento all'interno dell'address space, se guardavo la corrispettiva posizione all'interno della RAM.
Il binding a tempo di compilazione va bene anche per sistemi batch monoprogrammati. Essi funzionano che un'applicazione per volta può essere attiva e la RAM è usata in due zona, la zona per il monitor sopra, e sotto l'applicazione effettiva caricata in ram; Quindi sappiamo che ogni applicazione che noi vogliamo mandare in esecuzione sarà caricata esattamente all'interno della zona sotto della RAM.
Quindi se noi consideriamo un address space che dobbiamo caricare nella RAM, per ognuno degli elementi che sono all'interno di questo Address Space sappiamo qual è la corrispettiva posizione in RAM.

Binding a tempi di compilazione/caricamento

- ✓ adatto a contesti di esecuzione seriale
- ✓ adatto a sistemi batch monoprogrammati
- ✓ in entrambi i casi è nota la partizione di memoria fisica riservata per il codice dell’applicazione
- ✓ la compilazione e/o il caricamento genera un eseguibile “consapevole” di risiedere in quella data regione di memoria fisica



Questa cosa qui funziona anche con il binding a tempo di caricamento, carichiamo sempre a partire da 0, oppure carichiamo nella posizione ultimativa del nostro monitor. Anche se le informazioni nell'AB sono rilocate in modo tale che un'istruzione riferisce un'altra istruzione utilizzando un OFFSET interno all'address space, quando andiamo in RAM basta applicare l'offset K per il monitor, per poter accedere alle stesse informazioni, e quindi per determinare qual è la posizione EFFETTIVA della nostra informazione che stiamo riferendo all'interno della memoria fisica.
Se noi analizziamo il binding a tempo d'esecuzione qui andiamo a lavorare in maniera adeguata con sistemi e modelli d'esecuzione più complessi: sistemi batch multiprogrammati e time sharing.
Sono scenari in cui noi abbiamo che all'interno di un sistema operativo noi possiamo avere più applicazioni, quindi l'applicazione A,B e C, tutte e tre attive allo stesso istante di tempo. Quindi ovviamente tutte quante caricate all'interno della RAM. Chiaramente NON possiamo caricare B dove è presente A o C, e la stessa cosa vale per le altre due applicazioni.
Quando lanciamo C dobbiamo verificare qual è la zona della RAM che abbiamo a disposizione ed eventualmente far sì che, tutto ciò che C all'interno del suo address space sta riferendo da un elemento verso un altro elemento, poi si dovrà determinare il reale posizionamento degli indirizzi all'interno della RAM.
Il binding a tempo d'esecuzione serve, perché quando noi abbiamo altri processi sappiamo che è possibile che uno di questi processi, per esempio C, lo possiamo swappare fuori dalla memoria per liberare la RAM a qualche altro processo, e poi lo possiamo swappare IN nella stessa zona dov'era prima oppure in un'altra: quando noi lavoriamo secondo uno schema batch multiprogrammato in cui eventualmente possiamo anche eliminare dalla RAM alcune applicazioni perché magari sono in stato di blocco, per liberare RAM A VANTAGGIO di altre, riportare in RAM queste applicazioni nel momento in cui queste vengono ad essere sbloccate (La stessa cosa avviene in sistemi time-sharing), ovviamente abbiamo la possibilità di avere il binding a tempo d'esecuzione.

Binding a tempo di esecuzione

- ✓ adatto a sistemi batch multiprogrammati
- ✓ adatto a sistemi time-sharing
- ✓ ogni processo potrà essere caricato e/o spostato dinamicamente in zone di memoria fisica differenti in modo trasparente alla struttura del codice

Quindi ogni processo può essere caricato in un punto qualsiasi della memoria e spostato anche dinamicamente in zone differenti della memoria. Questo per effetto di una "SWAP OUT" e poi successivamente di una "SWAP IN". Quindi quell'applicazione la abbiamo in un'altra zona della memoria a valle dell'esecuzione di questo "Swap In".