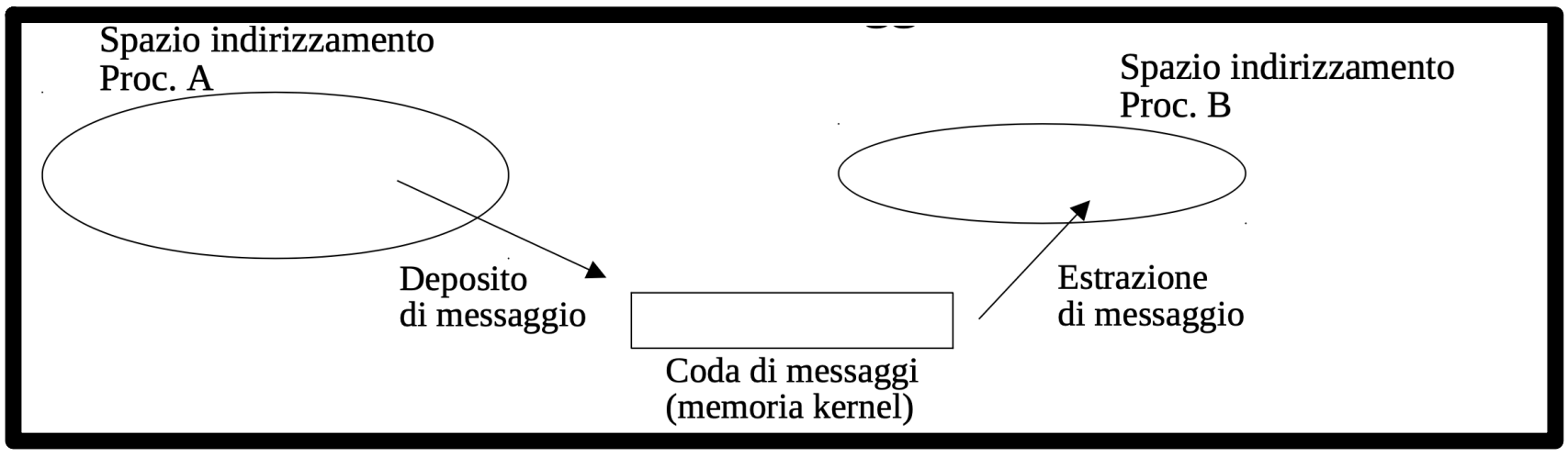
CODE MESSAGGI UNIX

sabato 6 maggio 2023 12:00

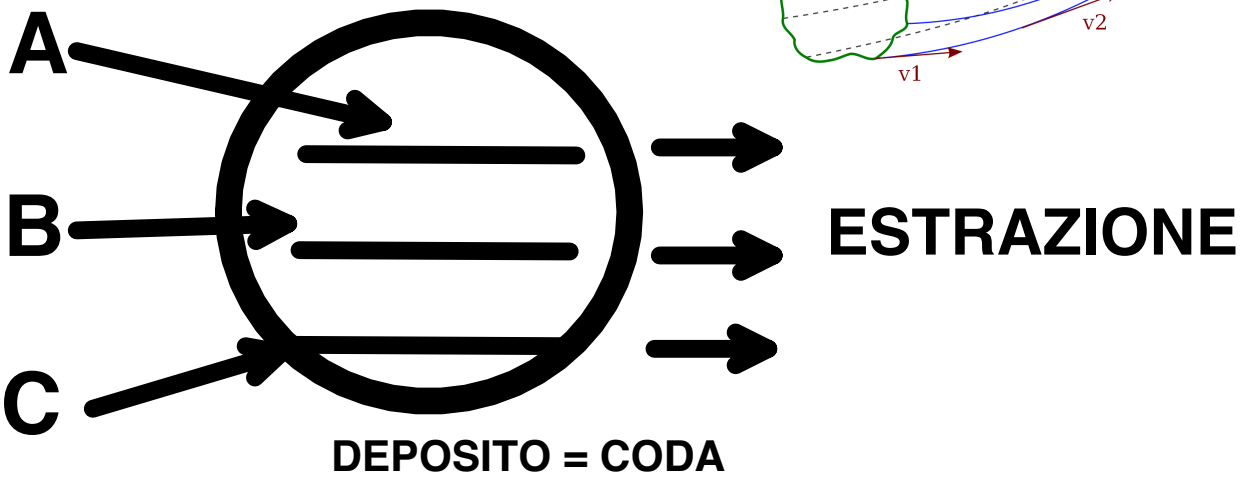
Code di messaggi UNIX

Quando parliamo di message queue su sistema UNIX parliamo di servizi per scambiare messaggi. Una coda di messaggio è una sorta di deposito - quindi è una struttura dati - che viene gestita dal kernel del sistema UNIX e, verso questa struttura dati, chiamando una certa API, possiamo determinare la consegna di un messaggio che noi avevamo originariamente impacchettato all'interno dello spazio di indirizzamento di uno specifico processo.



Quindi c'è una System Call e noi passiamo questo messaggio dallo spazio di indirizzamento del processo A verso la coda di messaggi. E ovviamente abbiamo anche la possibilità di estrarre i messaggi che sono presenti all'interno di questa struttura per ottenerne il contenuto all'interno dello spazio di indirizzamento del processo B. Chiaramente questo scambio di messaggi può avvenire all'interno della stessa applicazione. Quindi noi possiamo scambiare messaggi tra thread diversi della stessa applicazione.
Un thread impacchetta un messaggio da qualche parte, lo inserisce nella coda dei messaggi e questo viene estratto da un altro thread.
I messaggi, se noi andiamo ad usare ricezioni bloccanti, che dicono che il thread quando chiama il servizio di ricezione deve entrare in blocco - quindi deve essere eliminato dalla CPU e non tornare ready fin tanto che qualcosa non avvenga all'interno del sistema - possono essere utilizzati come oggetti per SINCRONIZZARE le attività dei vari thread.
Un thread, fino a che un altro thread non consegna uno specifico messaggio, rimane bloccato in attesa che questo messaggio venga consegnato.

Abbiamo una "SEND SINCRONA", che implica dire che quando noi spediamo un messaggio è sicuro che possiamo ri-utilizzare un buffer di memoria usato nello spazio di indirizzamento di A, senza danneggiare il contenuto del messaggio che abbiamo spedito.
Questo ovviamente perché viene effettuata una copia del messaggio all'interno del deposito, quindi nella coda di messaggi che stiamo utilizzando. Infatti abbiamo un indirizzamento indiretto.
La send è sincrona, ma può essere sia bloccante che NON, quando spediamo un messaggio abbiamo la necessità di eseguire una copia delle informazioni che sono presenti all'interno dell'address space all'interno della coda, CHE ATTENZIONE POTREBBE ESSERE SATURA!
In questo caso noi eventualmente possiamo andare in blocco su una spedizione, semplicemente perché la struttura dati non può ORA accogliere il nostro messaggio. Quindi noi veniamo messi in stato di wait, in attesa che lo stato di questo deposito sia modificato. Qualcuno magari estragga messaggi presenti e quindi ci sia la possibilità di andare ad utilizzare altro storage all'interno di questa struttura dati per ospitare nuovi messaggi.
Possiamo chiamare la send però in maniera non bloccante, ciò implica dire che noi non vogliamo rimanere bloccati neanche nel caso in cui non ci sia spazio sufficiente per accogliere il nostro messaggio. Il che implica dire che quando chiameremo una send sincrona non bloccante chiaramente possiamo ottenere un errore e l'errore ci può indicare appunto che la spedizione non è avvenuta, ma il thread non è comunque andato in blocco esattamente a causa dell'impossibilità da parte della coda di acquisire questo messaggio, che stiamo cercando di spedire.
Le receive() sono sincrone anche loro, il che implica dire che quando noi chiamiamo una receive per estrarre un messaggio dalla coda, siamo sicuri che - se questo messaggio era presente - ci è stato consegnato all'interno dell'area tampone dello spazio di indirizzamento di B.
Quindi abbiamo una SINCRONIA delle attività che vengono ad essere eseguite quando riceviamo il messaggio, rispetto alle attività del thread chiamante di questa applicazione di ricezione.
Ma anche la receive può essere bloccante o NON.
Bloccante implica dire che, se il messaggio non c'è, noi vogliamo rimanere fermi in attesa che il messaggio arrivi.
Quando chiamiamo questo servizio di receive il thread viene messo in stato di blocco.
Ovviamente possiamo avere la versione in cui chiamiamo questo servizio di receive indicando che, se il messaggio c'è noi lo vogliamo ricevere, quindi quando riprendiamo il controllo il messaggio deve essere presente all'interno del buffer dello spazio di indirizzamento di B, ma se il messaggio non c'è, NON VOGLIAMO comunque rimanere in attesa di messaggi.
Chiaramente l'API tramite il valore di ritorno, ci andrà ad indicare se ci è stato consegnato o non in modalità non bloccante.
Il buffering è a livello della memoria kernel, quindi è una struttura dati del kernel a capacità LIMITATA. Il sistema operativo gestisce il campo TIPO del messaggio, generando un MULTIPLEXING.
Multiplexing indica un oggetto con tante VIE, e questo implica dire che noi quando lavoriamo con un deposito di messaggio all'interno di un sistema operativo UNIX è la seguente cosa: Sul deposito possiamo immettere messaggi con un certo TIPO A, B o C, ed è come se avessimo 3 flussi differenti.
Il flusso di messaggi targati con tipo A, il flusso di messaggi targati con tipo B e il flusso di messaggi targati con tipo C.



Possiamo avere dei flussi indipendenti l'uno dall'altro, per andare poi eventualmente ad estrarre questi messaggi in base al flusso. Questo si chiama MULTIPLEXING.

Il contenuto di un messaggio ha una taglia tipicamente variabile, ma limitata superiormente. Il limite superiore è in funzione di come è configurato questo oggetto di coda che viene ad essere gestita dal sistema operativo.