

PROGETTO DI SISTEMI OPERATIVI

Ingegneria Informatica

20 aprile 2009

(teoria con soluzione)

(si prega di rispondere descrivendo i passaggi e i risultati intermedi)

1. (8 punti) Sia dato un sistema con memoria fisica di dimensione 128MByte, in cui si utilizza uno schema di gestione a partizioni (contigue) variabili con unità minima di allocazione della memoria di 128 Byte (ovvero lo spazio di memoria viene allocato in multipli di 128 byte).

- Al Sistema Operativo sono allocati in modo permanente i primi 32MByte di memoria.
- La tabella dei processi contiene, per ogni processo attivo, l'indirizzo iniziale (ADDR) e la dimensione (SIZE) della relativa partizione in memoria.
- La memoria viene allocata con strategia Best-Fit
- Le partizioni libere sono gestite mediante una lista linkata, in cui ogni nodo rappresenta una partizione libera; i nodi della lista sono costituiti da due campi: **(dimensioni della partizione, puntatore alla partizione successiva)**, entrambi rappresentati su 4 byte, dimensioni e indirizzi rappresentati in Byte, con valore

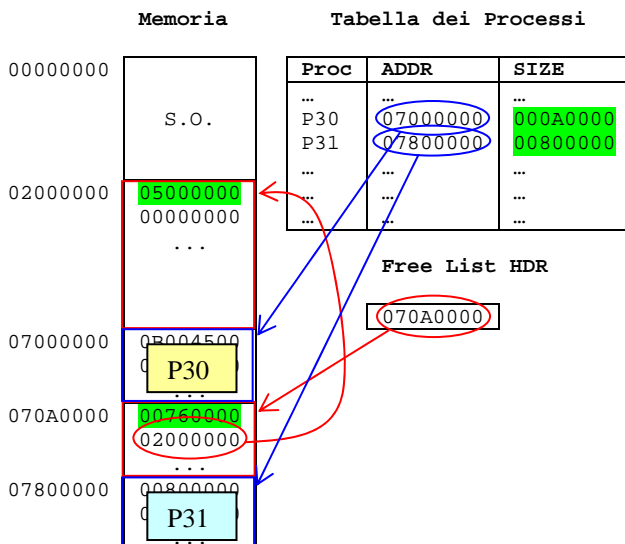
0 usato come puntatore nullo) e sono memorizzati nei primi byte della partizione che rappresentano. La lista è gestita con partizioni ordinate mediante dimensioni crescenti.

Memoria		Tabella dei Processi		
00000000	S.O.	Proc	ADDR	SIZE
	
		P30	07000000	000A0000
		P31	07800000	00800000
	
02000000	05000000 00000000
	
07000000	0B004500 00000000 ...	Free List HDR 070A0000		
070A0000	00760000 02000000 ...			
07800000	00800000 07000000 ...			

Si supponga che ad un dato istante la tabella dei processi e il puntatore alla prima partizione libera contengano le informazioni rappresentate in figura. Si rappresentino le modifiche alle partizioni in memoria, alla tabella dei processi e alla Free List, in seguito all'attivazione di 2 nuovi processi, P32 e P33, che richiedano rispettivamente 75MB e 5MB di memoria, seguita dalla terminazione del processo P30.

SOLUZIONE

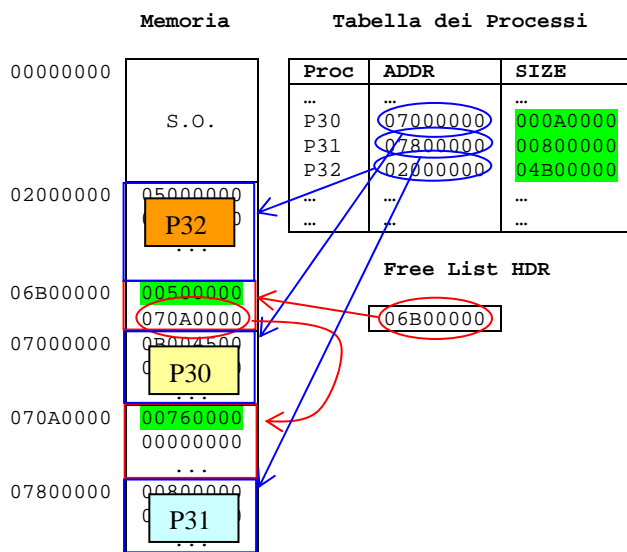
Configurazione iniziale



Ecco la configurazione iniziale che mette in evidenza puntori e dimensione delle partizioni in memoria.

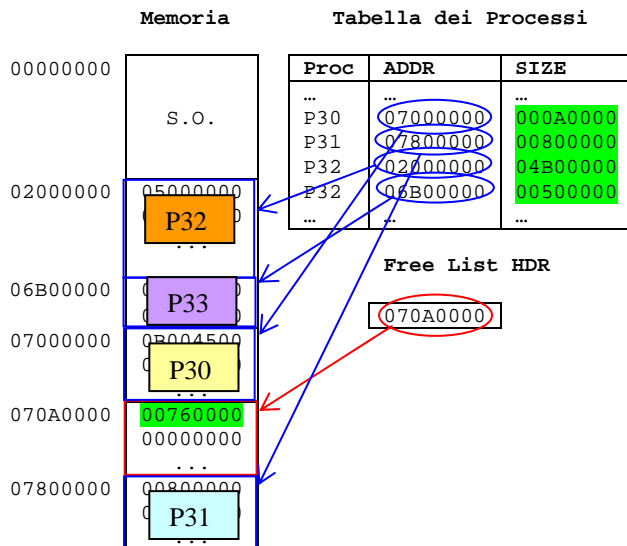
Si noti che le partizioni nella free list contengono, nella loro intestazione, dimensione e puntatore al prossimo elemento in lista. Per le partizioni allocate l'informazione contenuta può essere diversa (in quanto nelle partizioni si sono caricate informazioni relative ai processi).

Configurazione dopo allocazione a processo P32



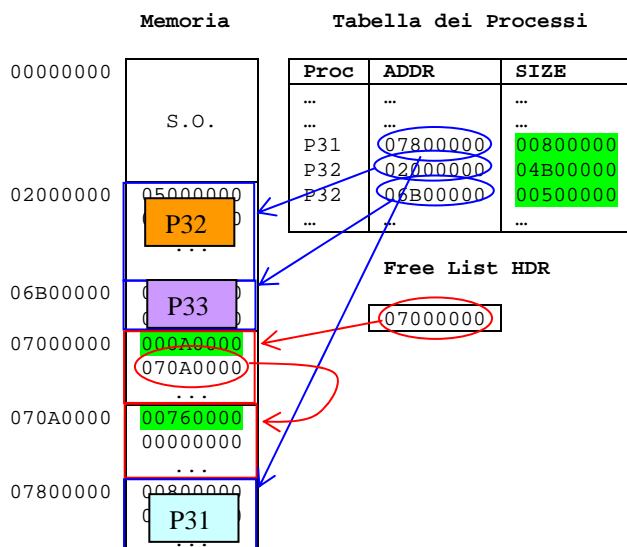
C'è una sola partizione di dimensione adeguata a P32, quella che inizia all'indirizzo 02000000, di dimensione 05000000 (= 80 MB). Una volta allocati 75 MB, resta una partizione libera residua di 5 MB, che va in testa alla free list.

Configurazione dopo allocazione a processo P33



La prima partizione nella free list è esattamente di dimensione 5MB, quindi viene rimossa dalla free list e allocata a P33.

Configurazione dopo termine del processo P30



La partizione allocata a P3 diviene libera e va in testa alla free list. A seconda della politica di ottimizzazione delle partizioni, si potrebbero unificare le due partizioni libere (adiacenti) in una sola partizione.

In forma più compatta, la soluzione può essere formulata come segue

Situazione iniziale

La memoria è suddivisa in 5 partizioni. La prima è allocata al Sistema Operativo, delle restanti 4 partizioni, 2 sono allocate ai processi P30 e P31, e 2 sono nella free list. Per le partizioni nella free list i primi 8 Byte contengono dimensione della partizione e puntatore alla successiva. Per le partizioni allocate, l'informazione contenuta può essere di altro tipo.

Si consiglia di mantenere indirizzi e dimensioni in esadecimale, ma per completezza si riportano anche i dati in sistema decimale. L'elenco delle partizioni iniziali, con indirizzo di partenza e dimensione, è:

Esadecimale

(00000000,02000000), (02000000,05000000), (07000000,000A0000), (070A00000,00760000), (07800000,00800000)

Decimale (con potenze di 2)

(0,32M), (32M,80M), (112M,640K), (112M+640K,7M+384K), (120M,8M)

Decimale

(0,33554432), (33554432,83886080), (117440512,655360), (118095872,7733248), (125829120,8388608)

Le partizioni iniziali sono

Free list: (070A00000,00760000), (02000000,05000000)

P30: (07000000,000A0000)

P31: (07800000,00800000)

Allocazione a P32. Servono 75MB, che vengono allocate dalla partizione di dimensione 80MB, da cui resta una partizione libera (di 5MB) in testa alla free list. Si rappresentano solo le variazioni. ($75 = 4B_{Hex}$)

Free list: (06B00000,00500000), (070A00000,00760000)

P32: (02000000,04B00000)

Allocazione a P33. Servono 5MB, che vengono allocate dalla partizione di dimensione 5MB, in testa alla free list.

Free list: (070A00000,00760000)

P33: (06B00000,00500000)

Termine di P30. La partizione allocata a P30 (di dimensione 640K va in testa alla free list.

Free list: (07000000,000A0000), (070A00000,00760000)

Le due partizioni libere potrebbero essere unificate in

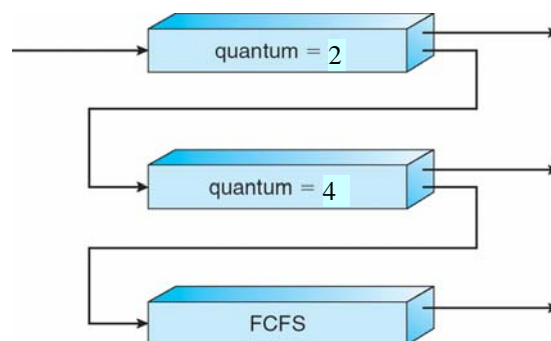
Free list: (07000000,00800000)

2. (7 punti) Cinque processi, identificati dalle lettere A-E, arrivano all'elaboratore agli istanti di tempo 0, 2, 2, 1, 5, rispettivamente. I processi hanno tempi di esecuzione di 4, 7, 5, 9 e 10 unità di tempo, rispettivamente. I processi D e E, effettuano una richiesta di I/O dopo le prime 5 unità di tempo di esecuzione. Si supponga che tali richieste di I/O siano soddisfatte in 4 unità di tempo. Descrivere (mediante diagramma di Gantt) la sequenza di esecuzione dei job su un sistema dotato di 1 CPU e calcolare i tempi di turnaround individuale (per ognuno dei processi) e globale, trascurando i tempi dovuti allo scambio di contesto, per una schedulazione di tipo multi-level feedback queue, con 3 code, di cui le prime due gestite con criterio round-robin (preemption con quanti di tempo rispettivamente 2 e 4), e la terza con strategia FCFS (FIFO). Le tre code hanno priorità relative crescenti (numero inferiore indica maggiore priorità).

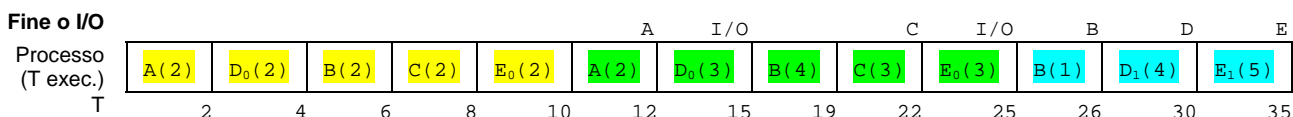
Soluzione

Lo schema proposto è un caso particolare di multilevel queue, simile all'esempio descritto nei lucidi *Silberschatz, Galvin and Gagne 2009 (cap 5, pp. 26-27)*. Lo schema delle tre code è riportato in figura. In pratica

- i processi entrano nella prima coda (gestita FIFO), quando vanno in esecuzione hanno quanto di tempo 2
- dopo l'esecuzione, se non hanno terminato, rientrano nella seconda coda, con quanto 4
- dopo l'esecuzione, se non hanno terminato, rientrano nella terza coda, dalla quale andranno in esecuzione senza preemption.



Nel caso in cui ci siano processi, pronti all'esecuzione, in più di una coda, ha priorità la prima, poi la seconda e infine la terza. In altri termini, i processi di durata superiore a 6 passano una volta in ognuna delle code. I processi nella seconda coda non vanno in esecuzione se non quando la prima coda è vuota. Analogamente per la terza coda.



Tra parentesi la durata dell'esecuzione. In caso di contemporaneità, si è utilizzato (in modo arbitrario) un criterio alfabetico.

I processi D ed E sono stati suddivisi in due CPU burst (D₀-D₁, E₀-E₁), tra cui, per I/O, trascorre un tempo >4. Indichiamo con Q0, Q1 e Q2 le tre code. In giallo le esecuzioni a partire dalla prima coda Q0 (RR quanto 2), in verde dalla seconda Q1 (RR quanto 4), in azzurro dalla terza Q2 (FCFS).

Ecco (per completezza) alcune configurazioni delle code. Tra parentesi si indicano, per le code, il tempo (Q0(2) significa coda Q0 al tempo 2, prima di fare accodamenti e o preemption, Q0(2+) significa coda Q0 immediatamente dopo il tempo 2), per i processi in coda si indicano i tempi di ingresso in coda (D₀(1) significa D₀ entrato in coda al tempo 1).

Q0(2) = (D₀(1), B(2))
 Q0(2+) = (D₀(1), B(2), C(2)), Q1(2+) = (A(2))
 Q0(5) = (C(2)), Q1(5) = (A(2), D₀(4))
 Q0(10) = (), Q1(10) = (A(2), D₀(4), B(6), C(8))
 Q1(18) = (C(8), E₀(14)), Q2(18) = (A(12))
 ...

3. (5 punti) Si descriva la funzione `getblk`, nell'ambito della gestione del "buffer cache" UNIX. Si richiede, in particolare, di spiegare i dati ricevuti in ingresso e ritornati in uscita, i principali casi in cui la funzione si trova ad operare, e le strategie algoritmiche adottate.

La funzione ottiene dal buffer cache il un buffer di dati corrispondente a un dato numero di blocco su disco. La ricerca veloce di un buffer associato ad un blocco viene garantita da una tabella di hash. I buffer possono essere contemporaneamente in una tabella di hash e in free-list, per cui sono dotati di 4 puntatori (avanti e indietro, per entrambe le strutture dati, per garantire cancellazione $O(1)$).

La funzione riceve come parametri

- Numero del dispositivo (da cui fare input)
- Numero del blocco

Ritorna come risultato un buffer, in stato "locked", da usare per il blocco

La funzione, finchè non riesce a ritornare un buffer, itera su due possibili casi principali (con sotto-casi)

- Blocco presente nel buffer cache (nella tabella hash)
 - Se locked (un altro processo ci sta lavorando), aspetta e riprova
 - Se non è locked, segna come locked, toglie blocco da free-list e lo ritorna
- Blocco non presente nel buffer cache. Cerca un blocco nella free-list
 - Se free-list vuota, aspetta e ritenta
 - Se free-list non vuota, toglie il primo buffer
 - Se "delayed-write" attiva write asincrona e riprova (cerca un altro blocco libero)
 - Se buffer normale, toglie dalla tabella di hash (perde la vecchio blocco) e mette nella tabella di hash in corrispondenza a nuovo blocco

Se i dati nel buffer non sono validi, sarà la funzione chiamante (es. `bread`) a inserirli nel buffer (leggendo il blocco su disco).

4. (5 punti) Si descriva brevemente la politica di paginazione a richiesta di tipo "working set". Perché la politica, applicata in modo esatto, risulta inefficiente? Come è possibile approssimare la politica in modo da migliorarne l'efficienza in termini di tempo? Che cosa si perde in termini di memoria utilizzata?

Nell'ambito della gestione di memoria virtuale con paginazione a richiesta, il working set di un processo al tempo t_i ($WS(t_i)$), è l'insieme di pagine cui si è fatto accesso in un intervallo di tempo Δ (di durata prefissata). La dimensione del working set non è fissa, ma varia in funzione della località degli accessi. Ad es. $|WS(t_i)|=1$ se il processo accede sempre alla stessa pagina, $|WS(t_i)|=\Delta$ se accede ad ogni istante a una nuova pagina.

Una politica di paginazione basata su working-set cerca di mantenere nel resident-set, ad ogni istante, tutte e sole le pagine del working-set. Tale politica, applicata in modo esatto, risulta inefficiente, perché richiede un potenziale aggiornamento del resident-set ad ogni accesso in memoria (sia con che senza page-fault).

Politiche che approssimano la strategia working-set cercano di

- ridurre il numero di aggiornamenti del resident-set, effettuandoli in corrispondenza ai soli page-fault o ad intervalli di tempo fissi
- sovra-dimensionare il working-set (il resident-set contiene quindi più pagine) misurando il tempo in macro-intervalli (basati su uso di reference bits).

5. (5 punti) Si consideri un disco magnetico di 40 GByte, caratterizzato dalla seguente geometria: 4 dischi, 8 testine di lettura/scrittura, 512K cilindri. Si ricorda che un cilindro è dato dalle tracce (aventi lo stesso raggio) accessibili in parallelo dalle 8 testine. Supponendo che il disco contenga due partizioni di ugual dimensione, formattate con file system che utilizzano blocchi di 4KByte, quanti settori (di cilindro) deve avere il disco, affinché ad ogni blocco dei file system corrisponda un settore fisico su disco? Quanti settori conterrà un cilindro?

Le due partizioni occupano globalmente l'intero disco (40 GB). La dimensione totale $|D|$ del disco è data dal numero totale di settori $\#S$, moltiplicati per la dimensione di un settore $|S|$. Un settore ha dimensione 4 KB (in quanto settori e blocchi coincidono). Il numero totale di settori è quindi

$$\#S = |D| / |S| = 40\text{GB} / 4\text{KB} = 10 \text{ M settori} = 10485760 \text{ settori}$$

Siccome il numero totale di cilindri $\#C$ è 512K, il numero totale di settori per cilindro $\#S/C$ è

$$\#S/C = \#S / \#C = 10\text{M} / 512\text{K} = 20$$

PROGETTO DI SISTEMI OPERATIVI
Ingegneria Informatica (a.a. 2007/2008 e precedenti)
20 aprile 2009

(teoria)

(si prega di rispondere descrivendo i passaggi e i risultati intermedi)

1. (8 punti) Sia dato un sistema di memoria virtuale con paginazione a richiesta (*demand paging*).
 - Si dica che cos'è e come viene gestito un "page fault".
 - Si dica poi che cos'è e a cosa serve il bit di validità
 - Che cos'è l'anomalia di Belady ?
 - Che cosa si intende con i termini "resident set" e "working set" ?
 - Sia data la seguente sequenza di riferimenti a pagine: 4, 5, 1, 2, 1, 4, 1, 3, 4, 3, 2, 1, 4
Si determini, per ognuno degli accessi in memoria, la presenza o meno di un page fault, utilizzando come algoritmo di rimpiazzamento l'LRU e il second chance (è richiesta la visualizzazione del resident set dopo ogni riferimento).
2. (7 punti) Cinque processi, identificati dalle lettere A-E, arrivano all'elaboratore agli istanti di tempo 0, 2, 2, 1, 5, rispettivamente. I processi hanno tempi di esecuzione di 4, 7, 5, 9 e 10 unità di tempo, rispettivamente. I processi D e E, effettuano una richiesta di I/O dopo le prime 5 unità di tempo di esecuzione. Si supponga che tali richieste di I/O siano soddisfatte in 4 unità di tempo. Descrivere (mediante diagramma di Gantt) la sequenza di esecuzione dei job su un sistema dotato di 1 CPU e calcolare i tempi di turnaround individuale (per ognuno dei processi) e globale, trascurando i tempi dovuti allo scambio di contesto, per una schedulazione di tipo multi-level feedback queue, con 3 code, di cui le prime due gestite con criterio round-robin (preemption con quanti di tempo rispettivamente 2 e 4), e la terza con strategia FCFS (FIFO). Le tre code hanno priorità relative crescenti (numero inferiore indica maggiore priorità).
3. (5 punti) Si descriva, nell'ambito dei sistemi operativi UNIX, la gestione del "buffer cache" UNIX. A cosa serve ? Come viene gestita la ricerca di un blocco ? Cosa è la free list ?
4. (5 punti) Si descriva brevemente il meccanismo di gestione di un driver generico per i dispositivi di I/O, identificando le funzioni svolte da processo utente, processo driver e interrupt handler. Che cosa si intende con il termine *interrupt vector* (vettore di interruzione) ? Che differenza c'è, nei sistemi UNIX, tra un driver per dispositivo a caratteri e a blocchi ?
5. (5 punti) Si consideri un disco magnetico di 40 GByte, caratterizzato dalla seguente geometria: 4 dischi, 8 testine di lettura/scrittura, 512K cilindri. Si ricorda che un cilindro è dato dalle tracce (aventi lo stesso raggio) accessibili in parallelo dalle 8 testine. Supponendo che il disco contenga due partizioni di ugual dimensione, formattate con file system che utilizzano blocchi di 4KByte, quanti settori (di cilindro) deve avere il disco, affinché ad ogni blocco dei file system corrisponda un settore fisico su disco ? Quanti settori conterrà un cilindro ?

