Cognome:_____ **Nome:** _____ **Matricola:**_____

Sistemi Operativi — A.A. 2005-2006, prova scritta del 7 luglio 2006

Libri e appunti chiusi. Vietato comunicare con chiunque. Vietato l'uso di cellulari, calcolatrici, palmari e affini. Tempo a disposizione: 60 minuti. Le domande sono etichettate con 1,2 o 3 asterischi:

* = domanda semplice, valutazione alta, rispondi a queste prima delle altre

** = domanda di media difficoltà

*** = domanda difficile, valutazione bassa, rispondi dopo aver risposto alle altre

1. * Cosa è il Translation Lookaside Buffer? Mostra uno **schema** architetturale per la traduzione degli indirizzi di memoria virtuale in indirizzi fisici che usa il Translation Lookaside Buffer. Commentalo **brevemente**.

Per la descrizione del TLB e dello schema vedi libro.

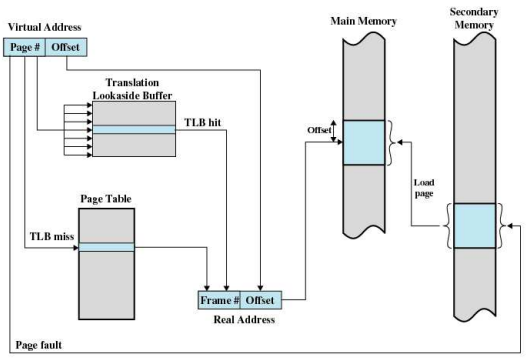


Figure 8.7 Use of a Translation Lookaside Buffer

2. * Che cosa è un "interrupt"? Descrivi **analogie e differenze** con il concetto di system call.

Un interrupt è una interruzione asincrona del normale flusso di esecuzione provocata da un evento esterno (timer, i/o, ecc). L'interrupt provoca la chimata di un "interrupt handler routine" che è parte del sistema operativo ed eseguito in modalità privilegiata. Ogni interrupt da luogo ad un mode switch. Viene salvato il program counter e, poiché l'interruzione è asincrona, anche lo stato della cpu, per essere ripristinato dopo l'esecuzione dell'interrupt handler routine.

Analogie con system call:
- il flusso di esecuzione viene interrotto, si salva il program counter e almeno parte dello stato e che poi verrà ripristinato
- la routine chiamata viene eseguita in modalità privilegiata (mode switch)

Differenze rispetto alle system call:
- in una system call la chiamata avviene in maniera sincrona, cioè prevista dal programmatore
- lo stato della cpu può essere usato per passare i parametri alla system call e per ritornare risultati

3. * Analogie e differenze tra le tecniche memoria virtuale e disk caching.

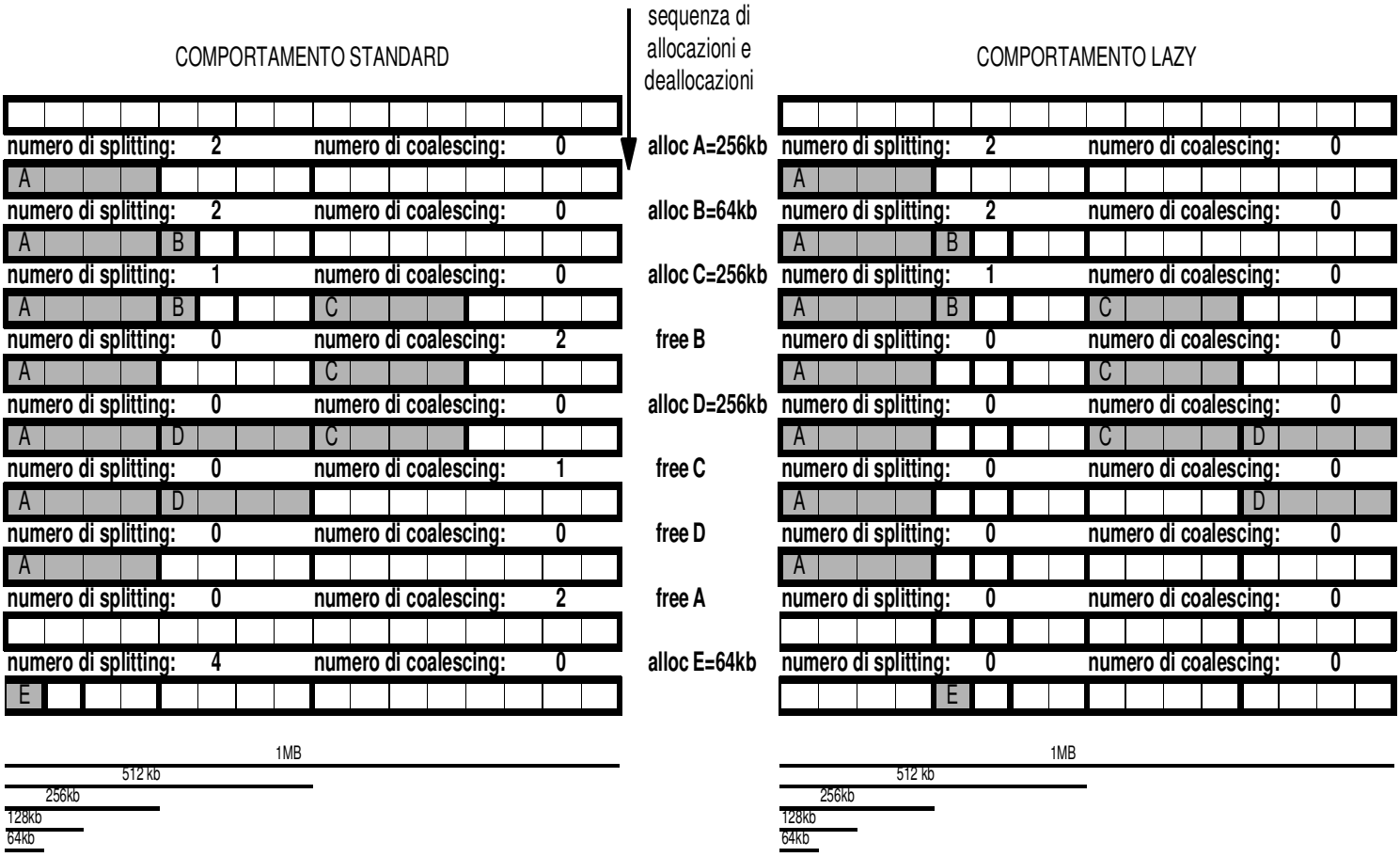
Analogie:
- entrambe le tecniche mirano a tenere in memoria solo dati e programmi acceduti di frequente

Differenze:
- le tecniche agiscono in domini differenti
  virtual memory: processi, pagine, segmenti
  disk caching: files

	ram		disk
virtual memory	rarely used processes, pages or segments	→	swap area
disk caching	disk cache	←	frequently used parts of files

4. ** Considera una zona di memoria di 1 megabyte gestita tramite un buddy system. Considera la sequenza di allocazioni e deallocazioni mostrata al centro dello schema (il tempo scorre verso il basso). Mostra come lo spazio

di memoria è suddiviso dopo ogni operazione e quali blocchi sono allocati (come nell'esmpio). Indica anche quanti splitting o quanti coalescing sono effettuati. Mostra sulla sinistra il comportamento del buddy system standard (coalescing appena possibile) e sulla destra il comportamento di quello con un approccio lazy (coalescing quando necessario).



5. ** Considera un sistema con architettura del kernel “execution within user process”. In tale sistema sono presenti due processi: A I/O bound e B puramente cpu bound (non fa mai chiamate di sistema). All'inizio A è running e B è ready. Lo scheduler dà sempre priorità ad A con una politica preemptive che si attiva ogni volta che A esce dallo stato di blocco. Il processore esegue di volta in volta A, B, mode switching, dispatcing, system call e interrupt handler. Mostra schematicamente, nella seguente tabella, l'ordine con cui tali attività vengono eseguite (una sola croce per ciascuna colonna).

		tempo →															
user mode	A (i/o bound)	x										x					
	B (cpu bound)						x										...
mode switch			x			x		x			x		x			x	
kernel mode	dispatching				x					x					x		
	system call			x										x			
	interrup handler								x								

6. ** Elenca **sinteticamente** il contenuto della struttura Inode nei sistemi UNIX. Mostra la **struttura gerarchica** che i file system dei sistemi UNIX utilizzano per tenere traccia dei blocchi di un file e che ha la sua radice nell'Inode.

Per gli Inode vedi materiale didattico

Cognome:_____ **Nome:** _____ **Matricola:**_____

Sistemi Operativi — A.A. 2005-2006, prova scritta del 7 luglio 2006

La struttura gerarchica è la seguente

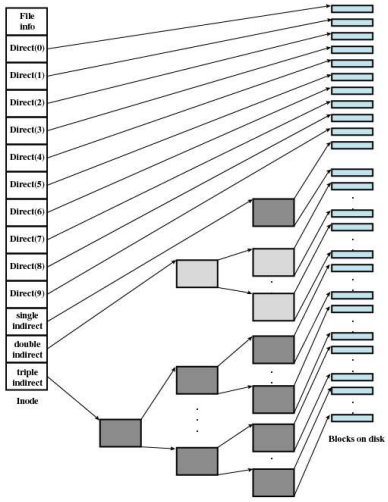


Figure 12.13 Layout of a UNIX File on Disk

7. ** Un processo fa in una prima fase 10000 accessi su 20 pagine. Le 20 pagine sono accedute in maniera del tutto casuale. In una seconda fase fa 100 accessi casualmente distribuiti su 40 pagine. In una terza fase fa 10000 accessi casualmente distributi su 10 pagine. Supponi di voler adottare una strategia che fa si che il resident set sia sempre uguale al working set. Supponi inoltre di poter scegliere Δ tra i valori 10, 20, 40 e 100. **Fai delle considerazioni sull'opportunità o meno di scegliere ciascuno di tali valori.**

fase 1: 10000 accessi su un insieme A di 20 pagine
fase 2: 100 su un insieme B di 40 pagine
fase 3: 10000 su un insieme C di 10 pagine

supponiamo che gli insiemi di pagine A, B e C accedute nelle varie fasi sia disgiunto altrimenti la rilevanza delle seguenti considerazioni dipende dal grado di sovrapposizione degli insiemi.

Δ = 10: |RS| ≤10, molti fault nella prime due fasi.
Δ = 20: |RS| ≤ 20, molti fault nella fase 2 ma questa è molto breve. Il numero di fault in tale fase è compreso tra 40 e 100.
Δ = 40: |RS| ≤ 40, gran parte dei fault sono evitati anche nella seconda fase
Δ = 100: |RS| ≤100, nelle transizioni di fase si occupa piu' spazio del necessario perche' si tengono nel resident set i working set di una fase e della successiva.

8. ** In quali casi round robin è unfair? mostra uno **schema** che lo rende fair e descrivilo disegnando **la rete di code** e commentandola **sinteticamente**.

Sistemi Operativi — A.A. 2005-2006, prova scritta del 7 luglio 2006

Round robin è unfair verso i processi che vanno in blocco prima dello scadere del loro quanto di tempo. Lo schema che affronta questo problema è quello del Virtual Round Robin (VRR). Vedi libro per la descrizione.

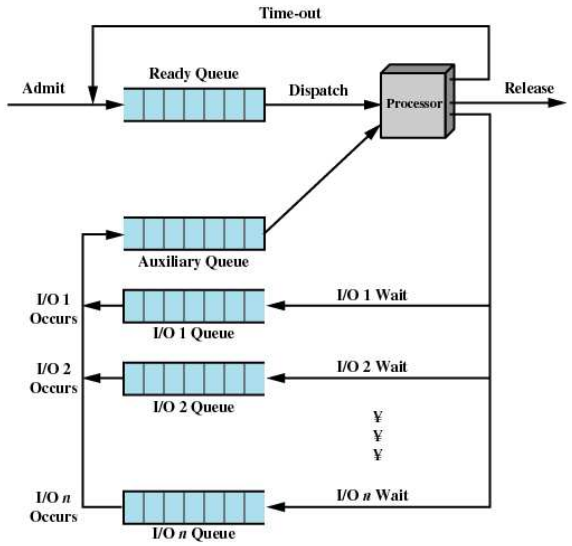


Figure 9.7 Queuing Diagram for Virtual Round-Robin Scheduler

9. *** Considera una strategia di scheduling feedback con n code (0 migliore, n peggiore) in cui la coda viene scelta in base al tempo c dell'ultimo cpu burst (supponi che sia $0<c\leq 1$) e al tempo b di permanenza in blocco per l'ultimo i/o burst (supponi che sia $0\leq b\leq 1$). Una funzione $f(c,b)$ dà il valore della prossima coda. Considera le formule f_1 e f_2 mostrate in tabella (la notazione $\lfloor x \rfloor$ indica l'intero inferiore) e i tre tipi di processi i/o bound, cpu bound e intermedio. Compila i campi min e max con i valori minimi e massimi che può avere la funzione per ciascun tipo di processo. Discuti **sinteticamente,** per ciscuna funzione e ciascun tipo di processo, se il comportamento è desiderabile per l'algoritmo di scheduling o no e perché.

tipi di processi	$f_1(c,b)=\left\lfloor n\cdot\frac{c}{b+1}\right\rfloor$		$f_2(c,b)=\left\lfloor\frac{n}{2}(c-b+1)\right\rfloor$	
c prossimo a 0 (i/o bound)	min 0	max n	min 0	max $\lfloor n/2 \rfloor$
	commento i processi i/o bound vanno sempre in coda 0 comportamento corretto		commento per b prossimo a 1 vale 0 per b prossimo a 0 vale $\lfloor n/2 \rfloor$ i processi i/o bound non vanno mai in code peggiori di $\lfloor n/2 \rfloor$. Il comportamento è comunque **coerente con lo scopo** in quanto un processo con $b=1$ è "più i/o bound" di uno con b prossimo a 0 e quindi deve essere inserito in code migliori.	

Cognome:_____ **Nome:** _____ **Matricola:**_____

Sistemi Operativi — A.A. 2005-2006, prova scritta del 7 luglio 2006

tipi di processi	$f_1(c,b)=\left\lfloor n\cdot\frac{c}{b+1}\right\rfloor$		$f_2(c,b)=\left\lfloor\frac{n}{2}(c-b+1)\right\rfloor$	
$b=0$ (cpu bound)	min *0*	max *n*	min $\lfloor n/2\rfloor$	max *n*
	commento per *c* prossimo a 1 vale *n* per *c* prossimo a 0 vale *0* comportamento coerente con lo scopo (c=1 molto cpu boud e quindi svantaggiatoma **forte variazione in funzione di *c***		commento per *c* prossimo a 1 vale *n* per *c* prossimo a 0 vale $\lfloor n/2\rfloor$ processi cpu bound non vanno mai in code migliori di $\lfloor n/2\rfloor$. Comportamento coerente con lo scopo.	
$b=c$ (comportamento intermedio)	min *0*	max $\lfloor n/2\rfloor$	min $\lfloor n/2\rfloor$	max $\lfloor n/2\rfloor$
	commento per *b=c* prossimo a 1 vale $\lfloor n/2\rfloor$ per *b=c* prossimo a 0 vale *0* i processi i/o bound non vanno mai in code peggiori di $\lfloor n/2\rfloor$ forte variazione in funzione di *b=c* **non coerente con lo scopo, non vi è infatti motivo per tale variabilità.**		commento per *b=c* prossimo a 1 vale $\lfloor n/2\rfloor$ per *b=c* prossimo a 0 vale $\lfloor n/2\rfloor$ comportamento corretto	

commenti complessivi

Nessuna delle due funzioni è pienamente soddisfacente. Tra le due possiamo scegliere f_2 poiché il suo comportamente è sempre coerente con gli obbiettivi dello scheduling anche se presenta ampi intervalli di variabilità.