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

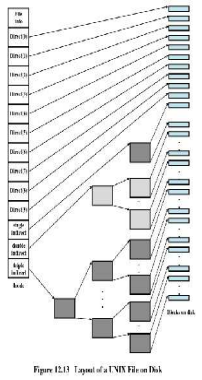
Sistemi Operativi 1 — A.A. 2004-2005, prova scritta del 20 settembre 2005

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

Domanda 1

1. * Cosa contiene l'inode nel filesystem di un sistema operativo UNIX? come è correlato l'inode al concetto di nome di file e di directory?

Un inode descrive vari attirbuti di un file (utente, gruppo, permessi, data/ora di modifica e accesso, ecc) e quali blocchi del disco contengono il file. Se le informazioni sui blocchi del file non entrano in un singolo inode se ne usano altri organizzati in una struttura ad albero. L'inode contiene anche il numero di rifierimenti ad esso. Infatti un file può avere più nomi (hard link). Una directory in unix contiene coppie (nome, inode id).



2. * Mettiti nei panni del progettista di un software per gestire un filesystem. Supponi di voler utilizzare l'algoritmo di I/O scheduling C-SCAN e di voler ottimizare la lettura di file molto lunghi. Quale è il modo migliore di sistemare i blocchi del file sul disco? in particolare quale è il modo migliore di sistemare i blocchi dei file che occupano più di una traccia?

Il modo migliore di sistemare i blocchi è in settori consecutivi all'interno di una stessa traccia (questo è vero per qualsiasi algoritmo di disk-scheduling) e in tracce consecutive nel senso di movimento della testina durante la lettura.

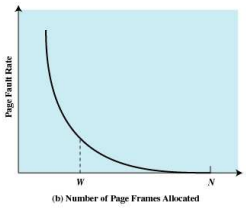
Domanda 2

1. * Un processo accede alle pagine in memoria con la seguente sequenza: 1234521121332213. Mostra il working set per Δ=4 dopo ciascun accesso.

pagina acceduta	***1***	***2***	***3***	***4***	***5***	***2***	***1***	***1***	***2***	***1***	***3***	***3***	***2***	***2***	***1***	***3***
working set	1	2	3	4	5	2	1	1	2	1	3	3	2	2	1	3
(l'ordine in cui disponi le pagine nelle caselle è irrilevante, scegli quello che più ti fa comodo)		1	2	3	4	5	2	2	1	2	1	1	3	3	2	1
			1	2	3	4	5	5			2	2	1		3	2
				1	2	3	4									

2. ** Descrivi l'approccio Page Fault Frequency e spiega perché tale approccio può essere considerato una approssimazione della tecnica del working set. Elenca pregi e difetti di tale approccio.

PFF si basa sull'ipotesi che la frequenza di page faults al variariare della dimensione del resident set circa in modo inversamente proporzionale. Si fissa un valore “ottimale” *f* della frequenza di page faults (che assumiamo corrispondere alla situazione |RS|=|WS|). Ad ogni page fault si stima la PFF, se PFF>*f* si aumenta |RS| se PFF<*f* si diminuisce |RS|. Il pregio di PFF è che è facilmente implementabile. Svantaggio: PFF richiede molta memoria durante i transitori tra vari “regimi di località”.



Sistemi Operativi 1 — A.A. 2004-2005, prova scritta del 20 settembre 2005

Domanda 3

1. * Supponi, in una architettura con 4 gigabyte di indirizzamento virtuale, che un processo occupi per lo stack gli indirizzi più alti e per il codice e i dati gli indirizzi più bassi. Quante page table entry servono se la pagina è di 4 kilobyte? Fai una assunzione per la grandezza della page table entry (giustificala) e calcola la grandezza della page table del processo.

4 GB di indirizzamento virtuale → 32 bit per gli indirizzi.
4 KB per pagina → 12 bit per offset all'interno di una pagina e 32-12=20 per l'identificatore di una pagina → 2^{20} page table entry (PTE).
Assumiamo che 4GB sia anche lo spazio di indirizzamento fisico → 20 bit per id frame → almeno 3 byte per PTE (o 4 byte: hardware più semplice, PTE allineate a 2^2 byte, e più spazio per bit ausiliari).
La grandezza massima della page table è di $3*2^{20}$ o $4*2^{20}$ bytes (cioè 3 o 4 Mbytes).
Poiché il processo occupa sia le zone basse che le zone alte la tabella delle pagine ha dimensione massima.

2. ** Confronta l'architettura "tabella delle pagine a più livelli" con l'architettura "inverted page table" e mostra quali sono i pro e i contro dei due approcci.

Entrambe le architetture mirano a mitigare il problema della grandezza della tabelle delle pagine.
Tabella delle pagine a più livelli.
pro: permette di non avere in memoria l'intera tabella.
contro: per una singola lettura si possono avere molti page fault (al massimo pari al numero dei livelli) e che comunque si devono fare un numero di accessi a memoria in più pari al numero dei livelli.
Inverted page table
pro: mantiene in memoria un numero di entry pari al numero di frame della memoria fisica (Le entry contengono l'id della pagina. Si applica una funzione hash al numero della pagina per calcolare il numero del frame. L'accesso alla tabella può confermare che tale frame contiene la pagina cercata o meno. Le collisioni sono gestite mediante la tecnica del chaining.) Spesso è sufficiente un solo accesso a memoria.
contro: in caso di collisioni si deve percorrere la catena. Il numero di accessi a memoria non è costante.

Domanda 4

1. * Devi progettare lo scheduling di un sistema in cui sono presenti processi che interagiscono con l'utente e decidi per un aproccio round robin. In base a quali considerazioni decidi la durata del quanto di tempo?

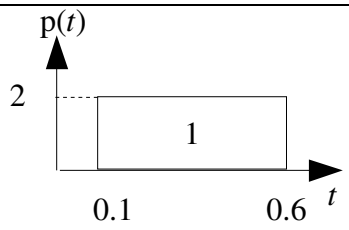
Il quanto di tempo deve essere scelto in modo che, la maggior parte delle volte, il quanto sia superiore alla durata del cpu burst delle applicazioni interattive.

2. ** Supponi che la durata del quanto di tempo di un certo sistema round robin sia di 0.5 secondi. Sono presenti due processi, uno interattivo e l'altro cpu bound. Se il processo interattiva ha durata del cpu burst distribuita uniformemente (densità di probabilità uniforme) tra 0.1 e 0.6 secondi. Quale sarà la distribuzione (densità di probabilità) del tempo di risposta percepito dall'utente?

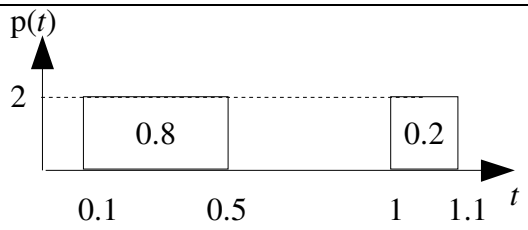
La durata del burst percepito dall'utente è distribuita uniformemente tra 0.1 e 0.5 e tra 1.0 e 1.1 con altezza 2. Infatti, se il processo interattivo ha un burst di più di 0,5sec il suo quanto scade. Il processo cpu bound prende sempre tutto il suo quanto di tempo.

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

Sistemi Operativi 1 — A.A. 2004-2005, prova scritta del 20 settembre 2005



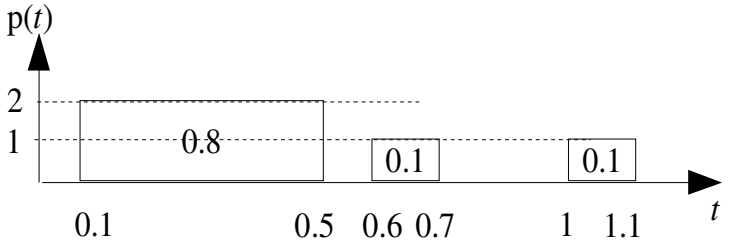
densità di probabilità della durata del cpu busrt del processo interattivo



densità di probabilità del tempo di risposta percepito per il processo interattivo

3. *** Se il secondo processo invece di essere cpu bound avesse un cpu burst di 0,1sec con probabilità ½ e di 0,5sec con probabilità ½, quale sarebbe la distribuzione del tempo di risposta percepito dall'utente del processo interattivo supponendo che l'altro sia ready (supponi indipendente la durata dei cpu burst dei due processi)?

Uniforme tra 0.1 e 0.5 con altezza 2 e tra 0.6 e 0.7 con altezz 1 e tra 1 e 1.1 con altezza 1. Infatti, se il processo interattivo ha un burst di più di 0,5sec il suo quanto scade. L'altro processo può occupare la cpu per 0.1 o 0.5 secondi con probabilità ½. La probabilità di avere cpu burst tra 0,5 e 0,6 per il processo interattivo viene divisa fra questi due casi.



Domanda 5

3. * Descrivi l'architettura dei sistemi operativi “process-based” (anche detta microkernel), descrivendo vantaggi e svantaggi rispetto a quella “execution within process” (stile UNIX).

Un microkernel realizza nello spazio utente alcune delle sue funzionalità tipiche (es. il filesystem). I suoi vantaggi sono modularità (che permette di cambiare il comportamento del sistema cambiando un processo) e robustezza (che permette di avere un kernel stabile anche quando un modulo va in crash). I suoi svantaggi sono relativi all'efficienza poiché i servizi offerti dal systema richiedono molti più process switch.

4. **Considera un sistema operativo di tipo “execution within process” con scheduling non-preemptive. Supponi che ciascun *processo utente* possa richiedere un servizio di I/O S mediante system call bloccante. Conta il numero di process switch e di mode switch complessivo nel caso in cui si abbiano *p* processi che fanno ciascuno solo *r* richieste al servizio S, che non ci siano altri processi nel sistema e ci sia sempre almeno un processo pronto quando viene effettuata la richiesta S.

I process switch vengono effettuati solo nel momento in cui un processo va in blocco (scheduling non preemptive) quindi abbiamo *p r* process switch. Un mode switch user-kernel viene effettuato quando un processo va in blocco e un mode switch kernel-user viene effettuato quando un processo viene schedulato. Quindi abbiamo 2 *p r* mode switch (2 per ciascun process switch).

5. *** Stessa situazione di prima ma con architettura a microkernel. Supponi che per eseguire S il sistema operativo debba attivare un *processo di sistema* il quale esegue altre *n* system call bloccanti che non

richiedono l'attivazione di ulteriori processi di sistema. Il processo di sistema ha priorità maggiore, e interrompe i processi utente quando diviene *ready*. Quando ottiene la cpu serve tutte le richieste in coda. Calcola il numero massimo di process switch e di mode switch per l'esecuzione dei p processi. Supponi che ci sia sempre un processo pronto quando viene chiamata una system call bloccante.

Il caso peggiore si ha quando il processo di sistema serve una sola richiesta alla volta (se servo più richieste per volta ho meno attivazioni e quindi meno process switch). Inoltre facciamo l'ipotesi che il processo di sistema lavori in user mode.
Per ciascuna delle $p\ r$ invocazioni di S si hanno 2 process switch, uno per attivare il processo di sistema e uno per tornare a uno dei processi utente. Ciò da luogo a $2\ p\ r$ process switch.
Per ciascuna invocazione del processo di sistema esso esegue n system call bloccanti ciascuna provoca la schedulazione di uno dei processi utente e relativo ritorno quando il processo di sistema non ridiventa ready. Ciò da luogo a $2\ n\ p\ r$ process switch. In totale si hanno $2\ p\ r\ (n + 1)$ process switch. I mode switch sono $4\ p\ r\ (n + 1)$.