Esercizi Terranova

Cap 1.

Il tempo di overhead è un parametro fondamentale per lo studio delle prestazioni di un SO. Esso rappresenta il tempo medio di CPU necessario per eseguire i moduli del kernel. È spesso fornito in percentuale rispetto al tempo totale di utilizzo della CPU. Si può calcolare con:

$$Overhead\ \% = \frac{Tempo\ CPU\ per\ l'esecuzione\ dei\ moduli\ del\ kernel}{Tempo\ totale\ di\ utilizzoCPU}$$

Ovviamente, più il tempo è basso, maggiore sarà la quantità di tempo CPU che si può utilizzare per i processi utente.

Nei sistemi a livelli l'inserimento di più strati implica un sostanziale aumento dell'Overhead, diminuendo l'efficienza del sistema stesso

Cap. 2 Processi & Thread

Supponiamo che sia in esecuzione il processo utente 3, quando si verifica un'interruzione dal disco: il program counter di questo processo, la parola di stato del programma e magari uno o più registri vengono messi sullo stack corrente dall'HW dedicato alle interruzioni e la CPU salta all'indirizzo specificato nell'interrupt vector. Questo è tutto quello che fa l'HW; da qui in poi è tutto in mano al SW. Azioni come il salvataggio dei registri o l'inizializzazione dello stack pointer, vengono svolte da una piccola routine in linguaggio assembler. Quando questa routine termina, chiama una procedura C per terminare il resto del lavoro per lo specifico tipo di interrupt.

Quando questa ha svolto il suo compito, viene richiamato lo scheduler per vedere qual è il prossimo processo da eseguire. Dopodiché il controllo viene passato al codice in linguaggio assembler perché vengano caricati i registri e la mappa di memoria per il nuovo processo corrente. Tutte queste azioni di salvataggio e ripristino vengono chiamate **context switching**.

Thread

4 thread da gestire su sistema single core (pseudo parallelismo)
T1 -> T2 -> T3 -> T4 -> T1 -> T2 -> T3 -> T4 -> ...

4 thread da gestire su sistema multi core (parallelismo puro)
core 0 T1 -> T3 -> T1 -> T3 -> ...
core 1 T2 -> T4 -> T2 -> T4 -> ...

Semafori

Supponiamo di avere 3 processi che condividono una variabile x e che i loro pseudo-codici siano i seguenti:

```
P1:
wait(S)          P2:              P3:
x=x-2            wait(R)          wait(T)
signal(T)        x=x+2            if (x<0) signal(R)
wait(S)          signal(T)        wait(T)
x=x-1            wait(R)          print(x)
signal(T)
```

Determinare l'output del processo P3 assumendo che il valore iniziale di x è 1 e che i 3 semafori abbiano i seguenti valori iniziali: S=1, R=0, T=0.

P2 e P3 attendono su due variabili (R e T) che sono inizialmente poste a zero, dunque vengono sospesi fino a una successiva signal che li faccia uscire dalla coda di attesa. P1, invece, attende sulla variabile S, che inizialmente è posta a uno, dunque non entra in sospensione, ma semplicemente pone il semaforo associato a 0, e continua l'esecuzione portando x, dal valore 1 al valore 1-2=-1. Adesso P1 sblocca un processo dalla coda di T (cioè P3), con signal(T), e torna immediatamente ad attendere su S (su cui, almeno in questo codice, non verrà mai più fatta una signal, e che quindi resterà in attesa per sempre...). Il processo sbloccato (P3) controlla il valore di x; lo trova pari a -1, e -1<0, quindi entra nel ramo then dell'if, eseguendo signal(R), che sblocca un processo dalla coda di R (P2), e torna in attesa su T. Il processo sbloccato dalla coda di R (P2), porta x da -1 a -1+2=1 e immediatamente sblocca un processo dalla cosa di T. L'unico che era presente in tale attesa era P3 da pochissimo fa, che, adesso, può finalmente fare la print(x): siccome x contiene 1, P3 stamperà 1, e questa è la risposta.

Algoritmi di schedulazione

First Come First Served (FCFS)

Abbiamo i seguenti processi da elaborare con l'algoritmo FCFS:

Processo	Arrivo	Durata
P1	0	7
P2	2	4
P3	4	1
P4	5	4

P1 arriva al s0 e andrà subito in elaborazione e ci rimarrà fino al complemento in s7. P2 arriva a s2 e attenderà i 5s del rimanente completamento di P1 per andare in esecuzione ed essere completato all's9. Così via per gli altri, dove P3 e P4 attendono entrambi s7 e vengono completati in P3=8, P4=11.
Tempi di attesa: P1=0, P2=5, P3=7, P4=7 (media 4,75)
Tempi di completamento: P1=7, P2=9, P3=8, P4=11 (media 8,75)

Shortest Job First (SJF)

Abbiamo i seguenti processi da elaborare con l'algoritmo SJF:

Processo	Arrivo	Durata
P1	0	7
P2	2	4
P3	4	1
P4	5	4

P1 arriva al s0 e andrà subito in elaborazione e ci rimarrà fino al complemento in s7. In quest'ultimo istante sono arrivati tutti gli altri processi, si sceglie quindi quello con durata minore che è P3 e viene eseguito, e così via.
Ordine di elaborazione: P1, P3, P2, P4
Tempi di attesa: P1=0, P2=6, P3=3, P4=7 (media 4)
Tempi di completamento: P1=7, P2=10, P4=8, P4=11 (media 8)

Shortest Remaining Time Next (SRTN, sistemi batch)

Supponiamo di avere i seguenti cinque processi con relative durate e tempi di arrivo:

processi	*P1*	*P2*	*P3*	*P4*	*P5*
durata	*7*	*5*	*2*	*4*	*4*
tempo di arrivo	*0*	*3*	*4*	*6*	*9*

Qual è il processo che sarà schedulato per ultimo?

Abbiamo detto che l'SRTN esegue prima i processi più veloci; essendo preemptive, nel momento in cui sta elaborando un processo ne arriva uno più veloce, sospende quello in corso e passa al più veloce. Quindi lo schema di esecuzione sarà (in grassetto, il processo a cui passa l'esecuzione):

t=0	t=3	t=4	t=6	t=9
P1=7	**P1=4**, P2=5	P1=3, P2=5, **P3=2**	**P1=3**, P2=5, P3=0, P4=4	P1=0, P2=5, P3=0, **P4=4**, P5=4

t=13	t=17	t=22
P1=0, P2=5, P3=0, P4=0, **P5=4**	P1=0, **P2=5,** P3=0, P4=0, P5=0	P1=0, P2=0**,** P3=0, P4=0, P5=0

Il processo schedulato per ultimo sarà P2.

Scheduling Round Robin (RR, sistemi interattivi)

Abbiamo i seguenti processi in coda con relativa durata in millisecondi, e quanto di tempo stabilito di 20 ms:

Processi	P1	P2	P3	P4
Durata (millisecondi)	30	15	60	45

Verranno eseguiti nel seguente ordine:

- P1 (interrotto dopo 20 ms, ne rimangono altri 10)
- P2 (termina la propria esecuzione perché dura meno di 20 ms)
- P3 (interrotto dopo 20 ms, ne rimangono altri 40)
- P4 (interrotto dopo 20 ms, ne rimangono altri 25)
- P1 (termina la propria esecuzione perché necessitava di meno di 20 ms)
- P3 (interrotto dopo 20 ms, ne rimangono altri 20)
- P4 (interrotto dopo 20 ms, ne rimangono altri 5)
- P3 (termina la propria esecuzione perché necessitava di esattamente 20 ms)
- P4 (termina la propria esecuzione)

Overhead %

Supponiamo di utilizzare un algoritmo di scheduling preemptive come il Round-Robin: assumendo di avere un context switch effettuato in 5 ms e di usare quanti di tempo lunghi 50 ms, a quanto ammonta la percentuale di overhead per la gestione dell'interlacciamento dei processi?

$$\frac{5}{50+5} = 0{,}09 \cdot 100 = 9\%$$

Cap 4

Consideriamo un sistema che fa uso di memoria virtuale con le seguenti caratteristiche: uno spazio di indirizzamento virtuale da 1 GB, un numero di pagina virtuale a 22 bit e un indirizzo fisico a 20 bit. Determinare esattamente quanti frame fisici ci sono in memoria.

Spazio di indirizzamento virtuale $1GB = 1024^3\, byte = (2^{10})^3 = 2^{30}\, byte$

Con un **numero di pagina virtuale a 22 bit** possiamo individuare un numero di pagine pari a:

2^{22} pagine

Adesso possiamo calcolare la dimensione di una pagina con $\frac{dimensione\, indirizzo\, virtuale}{numero\, di\, pagione}$

$$dimensione\, di\, una\, pagina = \frac{2^{30}}{2^{22}} = 2^8 = 256$$

L'indirizzamento fisico è a 20 bit quindi la dimensione totale della RAM è di 2^{20} byte Sapendo che la dimensione di una pagina è 256 byte il numero di frame in memoria RAM è:

$$numero\, di\, frame = \frac{2^{20}}{256} = \frac{2^{20}}{2^8} = 2^{12} = 4096$$

In un sistema che usa paginazione, l'acceso al TLB richiede 150ns, mentre l'accesso alla memoria richiede 400ns. Quando si verifica un page fault, si perdono 8ms per caricare la pagina che si sta cercando in memoria. Se il page fault rate è il 2% e il TLB hit il 70%, indicare l'EAT ai dati.

Convertendo tutti i tempi in ms, abbiamo:

- tempo di accesso al TLB: 150 ns = $150 \cdot 10^{-6}$ ms;
- tempo di accesso alla memoria: 400 ns = $400 \cdot 10^{-6}$ ms;
- tempo di gestione del page fault: 8 ms;
- page fault rate: 2% = 0,02;
- TLB hit: 70% = 0; 7.

Quindi

$EAT = TLB\ hit\ \cdot (150 \cdot 10^{-6}) + (1 - page\ fault\ rate) \cdot (150 \cdot 10^{-6} + 400 \cdot 10^{-6}) + page\ fault\ rate\ \cdot (150 \cdot 10^{-6} + 8 + 400 \cdot 10^{-6})$
$= 0{,}7 \cdot (150 \cdot 10^{-6}) + 0{,}28 \cdot (150 \cdot 10^{-6} + 400 \cdot 10^{-6}) + 0{,}02 \cdot (150 \cdot 10^{-6} + 8 + 400 \cdot 10^{-6})$
$= 0{,}7 \cdot (1{,}5 \cdot 10^{-4}) + 0{,}28 \cdot (1{,}5 \cdot 10^{-4} + 4 \cdot 10^{-4}) + 0{,}02 \cdot (1{,}5 \cdot 10^{-4} + 8 + 4 \cdot 10^{-4})$
$= 1{,}05 \cdot 10^{-4} + 1{,}54 \cdot 10^{-4} + 0{,}160011$
$= 0{,}1627\ ms$

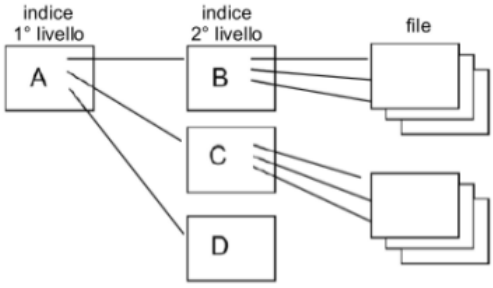
Facendo uso di una lista concatenata, che ha blocchi da 4kb e un numero di blocco di 32bit, quanti blocchi occorrono per memorizzare un file da 40kb?

Un blocco della lista è di 4KB, ovvero $4 \cdot 2^{10}$ byte.
Ogni blocco della lista concatenata ha un solo puntatore che specifica il numero del blocco successivo.
Questo numero è di 32bit, ovvero di 4byte.
Dunque ogni blocco della lista ha uno spazio per i dati di:
$(4 \cdot 2^{10} - 4)$byte = (4·1024 - 4)byte = (4096 - 4)byte = 4092byte.
Il file è di 40KB, quindi $(40 \cdot 2^{10})$ byte = 40960byte.
La dimensione del file diviso lo spazio disponibile per ogni blocco è: 40960 / 4092 = 10,01.
Sono serviti poco più di 10 blocchi, cioè 11 blocchi.

Supponiamo di utilizzare un file system UNIX basato su i-node particolari che contengono i seguenti campi: 12 indirizzi diretti a blocchi di dati, 1 indirizzo ad un blocco indiretto singolo e 1 indirizzo ad un blocco indiretto doppio. Supponendo di avere numeri di blocchi a 32 bit e blocchi su disco da 1 kb, indicare esattamente la dimensione massima (in kb) supportata da un simile i-node. Esplicitare il calcolo utilizzato.

L'indirizzamento indiretto ha blocchi da 1kb pieno di indirizzi a 32 bit, quindi sono 12 blocchi di indirizzo diretto + 256 di indiretto + 256^2 di indiretto doppio = 65804 blocchi.

Illustrare il meccanismo di allocazione indicizzata dei file. Si faccia riferimento ad un file system con indici a due livelli, dimensione del blocco logico pari a 512 byte e indirizzi di 4 byte. Come si alloca un file di 1 Mb? Quanto blocchi servono? Come si accede al suo 400° blocco? Come si accede al byte 236.448? Qual è la dimensione massima di un file? Quanti blocchi occupa complessivamente?



L'allocazione indicizzata a due livelli segue questo schema:
Se il blocco logico è di 512 byte e gli indirizzi di 4 byte si hanno 512 / 4 = 128 indirizzi per blocco.
Un file di 1 Mbyte occupa 1M / 512 = 2K = 2048 blocchi (1M = 1024*1024)
2048 blocchi richiedono 2048 indici = 2048/128 = 16 blocchi di 2° livello.
Complessivamente il file occupa 2048 + 16 + 1 = 2065 blocchi.
Il blocco n. 400 è indicizzato dal blocco indice di 2° livello n. 400 / 128 = 3, l'indirizzo è il 400 mod 128 = 16-esimo del blocco.
In generale, l'n-esimo indirizzo occupa i byte da (n - 1) x 4 a (n - 1) x 4 + 3.
Il blocco indice di secondo livello a sua volta è indicizzato dal 4° indirizzo (indirizzo n. 3) dell'indice di 1° livello, che si trova nei byte 12-15 del blocco.
L'accesso al byte 236.448 si risolve così: il byte occupa il blocco 236.448 / 512 = 461 (contando da 0, quindi è il 462-esimo), e si trova all'offset 236.448 mod 512 = 416. Trovato il blocco si procede come sopra.
La dimensione massima di un file è data dal numero massimo di indirizzi utilizzabili per indicizzarlo.
Nell'indice di 1° livello ci sono al massimo 128 puntatori, quindi ci sono al massimo 128 blocchi indice di 2° livello, ciascuno dei quali contiene 128 puntatori ai blocchi del file, per un totale di 128 x 128 = 16384 puntatori, e quindi blocchi del file. Il file può avere una dimensione massima di 16384 x 512 = 8 Mbyte (8.388.608 byte), e occupa complessivamente 16384 + 128 + 1 = 16513 blocchi

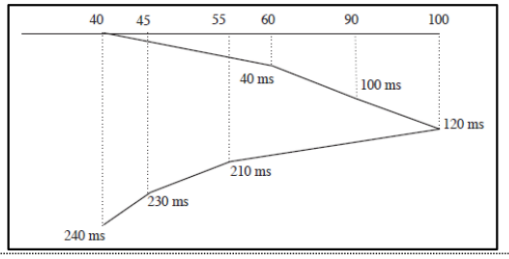
Cap 6 – File System

In un disco con blocchi di 2 Kbyte (= 2^{11} byte), è definito un file system FAT 16. Ogni elemento ha lunghezza di 2 byte e indirizza un blocco del disco. La copia permanente della FAT risiede nel disco a partire dal blocco di indice 0 e una copia di lavoro viene caricata in memoria principale all'avviamento del sistema operativo. Supponendo che la FAT sia dimensionata in base alla massima capacità di indirizzamento dei suoi elementi si chiede:
1) il numero di blocchi dati indirizzabili dalla FAT.
2) il numero di byte e di blocchi del disco occupati dalla FAT,
3) l'indice del primo blocco dati nel disco,
4) quale capacità (in blocchi e in byte deve avere il disco) per contenere tutti i blocchi dati indirizzabili.

1.Il numero di blocchi dati indirizzabili dalla FAT è 2^{16}
2.La FAT ha 2^{16} elementi di 2 byte pertanto occupa 2^{17} byte è 2^6 = 64 blocchi
3.Il primo blocco dati nel disco è quello di indice 64 (i blocchi precedenti sono riservati alla FAT)
4.Per contenere tutti i blocchi indirizzabili, il disco deve avere una capacità di almeno 2^{16} + 26 blocchi e 227 + 217 byte.

(Es. con tempi di arrivo diversi) Si consideri un disco con un intervallo di tracce da 0 a 100, gestito con politica SCAN. Inizialmente la testina è posizionata sul cilindro 40; lo spostamento ad una traccia adiacente richiede 2 ms. Al driver di tale disco arrivano richieste per i cilindri 90, 45, 40, 60, 55, rispettivamente agli istanti 0 ms, 20 ms, 30 ms, 40 ms, 80 ms. Si trascuri il tempo di latenza. (1) In quale ordine vengono servite le richieste? (2) Il tempo di attesa di una richiesta è il tempo che intercorre dal momento in cui è sottoposta al driver a quando viene e ettivamente servita. Qual è il tempo di attesa medio per le cinque richieste in oggetto?

Assumendo una direzione di movimento ascendente all'istante 0, le richieste vengono servite nell'ordine 60, 90, 55, 45, 40



Il tempo di attesa medio per le cinque richieste in oggetto è $\frac{(40-40)+(100-0)+(210-80)+(230-20)+(240-30)}{5} = \frac{0+100+130+210+210}{5} = 130\ ms$

Dati i seguenti dischi:

Disco 1	Disco 2	Disco 3
0010	1001	101x
1001	0101	110x
0110	0100	001x

Trovare i bit di parità x

x = 011 dato che con lo xor risultano tali valori

Dati i seguenti dischi:

101	010	110	100	110
100	111	100	010	110
110	100	110	-	111
111	010	001	101	1100

Trovare gli stripe del disco con il simbolo – .

Per ricostruire il blocco mancante, devo fare solamente lo XOR fra gli elementi delle terza (3a) riga; quindi:

111 XOR 110 = 001 XOR 100 = 101 XOR 110 = 011