# Sistemi Operativi
# Compitino Advanced Secondo Periodo 6 aprile 2004

1. (2 points) Si illustrino le differenze in termini di modalità di connessione e funzionamento, nonché di performance, fra un File Server Stateful (a stato) e uno Stateless (senza stato).

2. Si consideri un sistema a paginazione a tabella delle pagine invertita (con hash); spazio di indirizzamento fisico a 40 bit e logico a 64 bit, pagine da 4K, 4GB di RAM. Ogni entry nella page table occupa 64 bit. Supponiamo che un processo acceda alle seguenti pagine del suo spazio virtuale: 0x780E20, 0x780F22, 0xFFFFFFFFFFFFE3FF e 0xFFFFFFFFFFFFFFFF.

   (a) (1½ points) Quanti frame devono essere allocati al processo per contenere queste pagine e quanto serve (nel caso peggiore) al sistema di paginazione?

   > Visto che le pagine sono da 4KB gli ultimi 12bits di ogni indirizzo logico riferiscono all'offset e quindi le pagine logiche riferite sono 3 (0x780, 0xFFFFFFFFFFFFE e 0xFFFFFFFFFFFFF). Al sistema di paginazione alla peggio serviranno altri 3 frames per mantenere le relative traduzioni.

   (b) (5½ points) Si valuti la differenza di performance sull'accesso medio in memoria (EAT) fra questo sistema e uno a paginazione a 4 livelli. Si assuma un tempo di accesso in cache $t_c$, hit ratio $\alpha$, accesso in RAM $t_m$, un'hash per la tabella invertita che mediamente trova il riferimento in 2 passi, page fault rate $p$ e tempo di recupero pagina da disco $t_d$ mentre per ognuno dei primi 3 livelli si usano 13bit. Si assuma inoltre nel caso a livelli di avere una località talmente forte da far mantenere in memoria tutti i 4 livelli della tabella.

   > **pagine invertite** Viste le dimensioni della tabella possiamo assumere sia sempre mantenuta tutta in memoria, quindi:
   >
   > - ad ogni accesso attendiamo che la cache tenti di risolvere l'indirizzo;
   > - se va male, con frequenza $1-\alpha$, dobbiamo accedere alla tabella per la ricerca e il tempo medio di ricerca in tabella è $t_r = 4t_m$ (2 accessi per passo);
   > - ora, se va male, con frequenza $p$ dobbiamo accedere al disco;
   > - poi (in ogni caso) facciamo l'accesso in RAM.
   >
   > Quindi $t_{EAT} = t_c + (1-\alpha)(t_r + pt_d) + t_m = t_c + (1 + 4(1-\alpha))t_m + p(1-\alpha)t_d$.
   >
   > **4 livelli** Viste le dimensioni della tabella *non* possiamo assumere venga manenuta *tutta* in memoria.
   >
   > - ad ogni accesso attendiamo che la cache tenti di risolvere l'indirizzo;
   > - se va male, con frequenza $1-\alpha$, dobbiamo accedere alla tabella di primo livello per la ricerca, che sta in RAM;
   > - poi andiamo a leggere la parte necessaria della tabella di secondo livello. Per ipotesi l'abbiamo in RAM.
   > - analogamente per gli altri 2 livelli;
   > - poi con frequenza $p$ dobbiamo fare l'accesso a disco;
   > - infine (in ogni caso) facciamo l'accesso in RAM.
   >
   > Quindi $t_{EAT} = t_c + (1-\alpha)(4t_m + pt_d) + t_m = t_c + (1 + 4(1-\alpha))t_m + p(1-\alpha)t_d$.
   >
   > La differenza è quindi 0.

3. Abbiamo un sistema con un bus verso la memoria a 400MHz che trasferisce 64 bit in 10ns. Abbiamo dei dischi collegati mediante un bus PCI a 133Mhz che trasferisce 32 bit in 20 ns.

   Consideriamo di avere un disco con sustained transfer rate da 40Mbs che quindi riesce a fornire 4 bytes ogni 50ns.

   (a) (2 points) Effettuando un trasferimento con DMA qual'è la percentuale di cicli di accesso in memoria che viene rubata alla CPU?

trasferiamo 64 bits ogni 100ns e rubiamo alla CPU 10ns quindi 10%

Supponendo di usare 2 dischi, con le stesse caratteristiche, funzionanti in striping (RAID level 0) per aumentare la performance

(b) (2 points) quale transfer rate verso la memoria riusciamo ad ottenere?

possiamo mandare in parallelo i dischi, ogni 50ns abbiamo 64bit che mandiamo al bridge in 40ns quindi riusciamo effettivamente a raddoppiare il transfer rate portandolo a 80MBs

(c) (2 points) qual'è la percentuale di cycle stealing in questo caso?

trasferiamo 64 bits ogni 50ns e rubiamo alla CPU 10ns quindi 20%

4. (6½ points) Si consideri un disco con velocità di rotazione pari a 7200 rpm, tempo medio di seek $t_S$ pari a 7 msec, tempo track-to-track $t_T$ di 1ms, blocchi da 4K, traccie da 1024 blocchi. Si ha un file da 32MB allocato (per quanto possibile) tutto contiguo. Si valuti l'incremento di tempo per la lettura sequenziale completa usando una FAT32 cached rispetto a NTFS.

Con NTFS

Il file è composto da $\lceil \frac{32 \cdot 2^{20} B}{4 \cdot 2^{10} B} \rceil = 8 \cdot 2^{10} = 8192$ blocchi. Per essere il più possibile contiguo il file sarà logicamente di un solo run e fisicamente spezzato in 8 parti ognuna di intera traccia.

Quindi iniziamo a leggere la MFT in un tempo di $t_S + t_L + t_B$, poi andiamo a leggere il primo pezzo in un tempo di $t_S + t_L + 1024 t_B$. Poi leggiamo ogni altra traccia in $t_T + 1024 t_B$. Totale $2t_S + 2t_L + 7t_T + (1024 \cdot 8 + 1)t_B$.

Con FAT

Ogni blocco della FAT conterrà i puntatori di 1024 blocchi del file.

Quindi iniziamo a leggere il primo blocco della FAT in un tempo di $t_S + t_L + t_B$, poi andiamo a leggere il primo pezzo del file in un tempo di $t_S + t_L + 1024 t_B$. Per ogni altro pezzo è tutto lo stesso. Totale $8(2t_S + 2t_L + 1025 t_B)$.

La differenza è $14t_S + 14t_L + 7t_B - 7t_T = \ldots$.

5. (7 points) Scrivere uno pseudo-programma C che si occupa di sincronizzare l'orologio di sistema mediante un servizio d'orologio remoto (il cui IP è specificato da linea di comando).

Il programma deve garantire di impostare l'orologio di sistema con una differenza di massimo 50ms da quello del server e sa di funzionare in un sistema operativo con ticks da 10ns. Se entro 10 tentativi di comunicazione con il server non riesce nell'intento il programma termina con un messaggio d'errore.

[si assuma tranquillamente che l'orologio di sistema sappia misurare 50ms con ottima precisione]

```
includere gli headings

definire le costanti

int main (unsigned argc, char **argv)
{
  int sock;
  char inputline[LINESIZE];
  struct_per_systemclock time1, time2;
  struct_per_la_socket client, server;
```

```
  struct_per_indirizzi_IP *host;
  altre_vars;

  if (argc != 2) {
    fprintf(stderr, "usage: %s <server address>\n", argv[0]);
    exit(2);
  }

  sock = crea_socket;
  if (sock < 0) {
    perror("creating socket");
    exit(1);
  }

  client=prepariamo_struttura(costanti_client);

  if ( leghiamo_indirizzo(sock,client) < 0) {
    perror("bind failed");
    exit(1);
  }

  if ( (host=recuperiamo_indirizzo_da(argv[1]) == NULL) {
    perror("unknown host");
    exit(1);
  }

  server=prepariamo_struttura(host);

  for (i=0; i<10; i++) {
      /* spediamo un pacchetto dummy per svegliare il server */
      send_pacchetto(sock, server);

      time1 = leggi_systemclock();

      /* riceviamo la risposta, contenente la data locale */
      receive_pacchetto(sock, inputline, server);

      time2 = leggi_systemclock();

      if ( calcola_differenza_orari(time1,time2) < costante_50ms) {
        imposta_systemclock(inputline);
        exit(0);
      }
  }

  perror("cannot get accurate time");
  exit(1);
}
```

6. (a) (1 point) In un sistema con file system journaled che faccia caching sulle scritture dei blocchi dei files potrebbe aver senso il prevedere un processo di sistema che, di tanto in tanto, vada a copiare su disco i blocchi dati modificati?

Sì, perché solitamente i file system journaled mantengono sui log solo gli attributi, mentre i blocchi dati sono soggetti al normale meccanismo di caching con scrittura asincrona.

(b) (1½ points) Esiste un sistema operativo reale che fa ciò?

Unix, la system call `fsync()` serve per forzare la scrittura su disco di tutti i blocchi dati e attributi modificati di un file.

7. (6 points) Al driver di un disco arrivano, nell'ordine (e nello "stesso" istante di tempo), richieste per i cilindri 24, 41, 12, 6, 8, 26, 42. Supponendo che la testina si trovi inizialmente sul cilindro 22, e che il tempo necessario per spostare la testina di 1 cilindro sia di 6ms, quanto tempo viene speso per lo spostamento della testina per servire tutte le richieste con la politica LOOK (direzione iniziale sia ascendente), FCFS e SSTF rispettivamente?

Per FCFS: lo spostamento totale è dato dalla somma delle differenze tra il cilindro attuale e il successivo, partendo dal cilindro 22.

$$2 + 17 + 29 + 6 + 2 + 18 + 16 = 90 \text{ cilindri} = 540ms$$

Per SSTF: lo spostamento totale è dato dalla somma delle differenze tra il cilindro attuale e il più vicino tra quelli rimanenti, partendo dal cilindro 25.

$$2 + 2 + 14 + 4 + 2 + 35 + 1 = 60 \text{ cilindri} = 360ms$$

Per LOOK: lo spostamento totale è dato dalla distanza tra il cilindro attuale (22) e il massimo nella direzione attuale (42), più la distanza da tale massimo al cilindro minimo (6):

$$(42 - 22) + (42 - 6) = 56 \text{ cilindri} = 336ms$$

8. (a) (3 points) Si illustrino le problematiche che si dovrebbe affrontare per sviluppare un sistema operativo su un multiprocessore NUMA. Si pensi in particolare a dove e come mantenere le strutture dati del kernel.

   (b) (2 points) Che differenza in scalabilità ci si può aspettare (sempre su questa architettura) fra un Sistema Operativo a microkernel e uno monolitico?

   (c) (2 points) Si possono avere deadlock "locali" dovuti all'uso di risorse del kernel tutte situate in una memoria locale? E ci possono essere deadlock "globali" che coinvolgono risorse del kernel che si trovano in differenti memorie locali? Quale soluzione per evitarli, in caso, appare più adeguata?