Programmazione di Sistema
13 luglio 2016 (teoria)

Si prega di rispondere in maniera leggibile, descrivendo i passaggi e i risultati intermedi. Non è possibile consultare alcun materiale. Durata della prova 70 minuti. Sufficienza con punteggio >= 8. Prima e seconda parte possono essere sostenute in appelli diversi. La presenza a una delle due parti annulla automaticamente un'eventuale sufficienza già ottenuta (per la stessa parte): viene intesa come rifiuto del voto precedente.

1. *(3 punti)* Che cosa si intende, nel contesto della gestione della memoria virtuale da parte di un sistema operativo, con il termine paginazione a richiesta (demand paging)?

R	È una strategia di gestione della memoria che sfrutta la suddivisione della memoria in pagine (frame) caricando in memoria i frame corrispondenti alle pagine solo nel momento in cui servono realmente. Le pagine cui non si accede mai, pertanto, non saranno mai caricate in memoria fisica.

E' possibile effettuare la paginazione a richiesta con allocazione contigua di memoria? *(motivare la risposta).*

R	I due tipi di allocazione sono concettualmente diversi. Nell' allocazione contigua non si sfrutta la suddivisione in pagine della memoria allocando spazio per l'intero progamma. La paginazione, tra l'altro, è proprio un meccanismo adottato per evitare l'allocazione contigua.

Si supponga di avere paginazione a richiesta con una tabella delle pagine a due livelli e MMU con TLB. Supponendo che un page fault sia servito in 5 millisecondi, che un accesso (lettura/scrittura) in RAM impieghi 100 microsecondi, un accesso alla TLB 30 nanosecondi, quanto costa in media un accesso in memoria, con TLB hit del 99 %, nel caso in cui non ci sia un page fault? E nel caso di un page fault?

R	Accesso senza page fault: $T_1 = TLB_{hit} * (T_{TLB} + T_{mem}) + (1 - TLB_{hit}) * (T_{TLB} + 3 * T_{mem})$ = $0.99 + (30 + 10^{-3} + 100) + 0.01 * (3 * 100 + 30 * 10^{-3}) = 102.3\ \mu s\ \approx 102 \mu s$ Accesso con page fault (si noti che se c'è page fault c'è sicuramente TLB miss): $T_2 = T_{TLB} + 3 * T_{mem} + T_{pf} \approx T_{pf} \approx\ 5\ ms$

Quale deve essere la frequenza di page fault per garantire tempi di accesso in memoria di (al massimo) 200 microsecondi? Si tratta di frequenza massima o minima?

R	Si detta p la frequenza di page fault $T_{mem} = p * T_{pf} + (1 - p) * T_1 \leq 200\ \mu s$ $T_{mem} = p * 5 * 10^3\ \mu s + (1 - p) * 102\ \mu s \leq 200\ \mu s$ $p * (5000 - 102)\ \leq$ 200 - 102 $p * (5000 - 102)\ \leq$ 98 / 4898 ≈ 0.02

2. *(4 punti)* Sia dato un file di dimensione 80MB, in un file system di dimensione complessiva 200GB. Si vogliono confrontare le possibili organizzazioni di tale file, su file system con allocazione non contigua, secondo gli schemi:
   - *"linked list allocation"*
   - "File Allocation Table (FAT)"
   - *"indexed allocation"* (attenzione: non iNode, ma semplice organizzazione a indici)

   Si supponga che i blocchi su disco (sia per i dati che per gli eventuali indici) abbiano dimensione 4KB, e che i puntatori e gli indici abbiano dimensione 32 bit. Si dica, per ognuna delle 3 soluzioni:

- quanti blocchi (di dato e/o di indice) sono necessari per il file (attenzione: Si richiede il conteggio ESATTO, eventualmente espresso in termini di potenze di 2, ricordando che 1K = 2^{10} e 1M = 2^{20} !) ?

R	**"linked list allocation":** 80MB / (4KB-4B) =20501 blocchi dato, non ci sono blocchi di indice **"FAT":** 80MB / 4KB = 20K = 20480 blocchi dato, non ci sono blocchi di indice **"indexed allocation":** 80MB / 4KB = 20K = 20480 blocchi dato (ad ognuno è associato un indice) Siccome un blocco di indici contiene 1K indici (4KB/4B), i blocchi indice sono 20K/1K = 20 di secondo livello, più 1 di primo livello, in totale 21.

- quante letture in RAM e quanti accessi a disco (per trasferire un blocco in RAM) sono necessari per leggere il byte n. 22070 all'interno del file? Si consideri un *accesso diretto a file*. Si supponga di non utilizzare buffer cache, e che un puntatore o indice vanga letto in RAM con una sola lettura. Si supponga poi che il File Control Block (FCB) sia già in memoria RAM, e che ogni accesso a disco legga o scriva un blocco di 4KB, e che la FAT (qualora utilizzata) sia già in RAM.
-

R	Il byte 22070 si trova nel blocco 5, cioè il sesto, in tutti e 3 i casi. Con linked list, 22070/4092)=5.393. Con FAT e indici 22070/4096 = 5.388. **"linked list allocation":** 1 accesso in RAM per recuperare il primo puntatore da FCB, 6 letture sul disco per recuperare i 6 blocchi. 6 letture in RAM (5 puntatori e il byte desiderato). **"FAT":** 1 accesso in RAM per recuperare il primo indice da FCB, 5 letture in RAM per scandire la lista nella FAT. 1 lettura di blocco su disco e 1 in RAM per il byte desiderato. **"indexed allocation":** 2 letture in RAM per i puntatori a due blocchi di indici (a due livelli), uno letto in FCB e uno nel blocco di indici di primo livello. 1 lettura dal blocco di indici di secondo livello, per l'indice del blocco dato. 1 lettura di blocco su disco e 1 in RAM per il byte desiderato.

3. *(3 punti)* Sono date le seguenti due funzioni in linguaggio C utilizzate per richieste di accesso a blocchi su disco:

```
void enqNextBlk (queue_t *dq, int blk) {
  listHeadInsert(dq->list,blk);
}
int deqNextBlk (queue_t *dq) {
  dq->currBlk = listExtractNearest(dq->list, dq->currBlk);
  return dq->currBlk;
}
```

La funzione `enqNextBlk` viene utilizzata per mettere nella coda (`dq`) la richiesta di accesso al blocco `blk`. La funzione `deqNextBlk` viene chiamata per ottenere il nuovo blocco a cui fare accesso (il blocco schedulato).
La funzione `listHeadInsert` realizza un inserimento in testa, mentre la `listExtractNearest`, ricevuti come parametri una lista e l'indice `ref` di un blocco, cancella dalla lista, ritornandone l'indice, il blocco per il quale è minima la distanza rispetto a `ref`.
Supponendo la seguente sequenza di chiamate, con coda inizialmente vuota e `dq->currBlk` inizializzato al valore 700, corrispondente alla posizione iniziale della testina (si utilizzano le abbreviazioni `eNB` e `dNB` per le due funzioni):
`eNB(dq,300) eNB(dq,550) N=dNB(dq) eNB(dq,200) eNB(dq,900)`
`N=dNB(dq) eNB(dq,1200) N=dNB(dq) N=dNB(dq) N=dNB(dq)`
Si indichino i valori successivi di N (che indicano gli accessi a blocchi su disco schedulati) e si rappresenti la lista dopo ogni chiamata a `enqNextBlk` (`eNB`).

R1	Coda	Curr	N
	-	700	-
	300	700	-
	550,300	700	-
	300	550	550(150)
	200,300	550	-
	900,200,300	550	-
	900,200	300	300(250)
	1200,900,200	300	-
	1200,900	200	200(100)
	1200	900	900(700)
	-	1200	1200(300)

Si calcoli la lunghezza del percorso totale della testina di lettura/scrittura, misurato in numero di cilindri (ogni cilindro contiene 10 blocchi).

R	15+25+10+70+30 = 150

4. *(4 punti)* Si risponda, nell'ambito a un sistema operativo OS161, alle domande seguenti:

- Che cosa è il *trapframe* ? Quali dati vi vengono memorizzati?

R	Il trapframe è una struttura in cui si memorizzano le informazioni di stato relative ad un processo all'atto di un cambio di contesto. (per es nel caso di una syscall)

- Sia dato un puntatore a *trapframe* `tf`, che cosa sono i campi `tf->tf_a0, tf->tf_a1`, ... ?

R	Sono campi relativi a parametri per la syscall. Nel caso di read/write possono essere ad esempio il file descriptor, il puntatore al buffer di lettura/scrittura e la sua dimensione. Corrispondono ai registri MIPS

- La funzione di libreria C `printf` viene realizzata in OS161 mediante una chiamata a `vprintf`. Quest'ultima, in modo indiretto, chiama `write`. Perché la `printf` non viene realizzata (in modo molto più semplice) chiamando direttamente la `kprintf` (che è già realizzata e funzionante)?

R	La kprintf è una funzione nello spazio kernel e non può essere invocata da una funzione user in modo diretto.

- In OS161, a fronte di un comando `p <file eseguibile>`, la funzione `menu_execute`, chiamata da `menu` (di cui si riporta il sorgente), attiva `cmd_dispatch` (per eseguire il comando). Perché, una volta attivato il programma utente, `menu` passa immediatamente alla prossima chiamata a `kgets` (ed eventualmente `menu_execute`), senza attendere il termine del programma utente?

```
menu(char *args) {
  char buf[64];
  menu_execute(args, 1);
  while (1) {
    kprintf("OS/161 kernel [? for menu]: ");
    kgets(buf, sizeof(buf));
    menu_execute(buf, 0);
  }
}
```

R	- Manca la sys_exit che segnala/ricorda lo stato. - Manca l'attesa da parte del kernel (basata su interrupt) con relativo meccanismo di sincronizzazione (da implementare)