Programmazione di Sistema
5 luglio 2019 (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. Le due parti 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. ***LE RISPOSTE SI/NO VANNO MOTIVATE***

Soluzione proposta: *si intenda la soluzione come indicativa, in particolare nel caso di domande a risposta aperta.*

1. *(3.5 punti)* Si consideri la seguente sequenza di riferimenti in memoria nel caso di un programma in cui, per ogni accesso (indirizzi in esadecimale, si indirizza il Byte), si indica se si tratta di lettura (R) o scrittura (W): **R 3F5**, R 364, **W 4D3**, W 47E, R 4C8, W 2D1, R 465, W 2A0, **R 3BA**, W 4E6, R 480, R 294, R 0B8, R 14E.
   Supponendo che sia indirizzi fisici che logici siano su 12 bit, che si usi paginazione con pagine di dimensione 128 Byte e che il massimo indirizzo utilizzabile dal programma sia C10, si dica quante pagine sono presenti nello spazio di indirizzamento del programma e se ne calcoli la frammentazione interna.

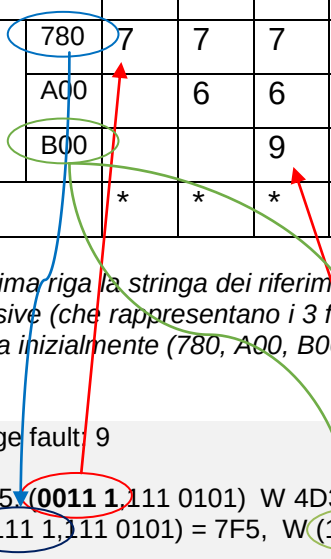
   *Numero totale di pagine indirizzabili (incluse quelle fuori dai riferimenti proposti): C10 = 1100 0, 001 0000. Il massimo indice di pagina 2*C (esadecimale) = 2*12(decimale) = 24 => In totale lo spazio indirizzabile comprende 25 pagine. Metodo alternativo:* $C10_H + 1 = 3089_{10}$, $NP = \lceil 3089/128 \rceil = 25$

   Frammentazione interna: L'ultima pagina è occupata fino all'offset 0010000 (16) => fr. Int. = 128-17 = 111 Byte. Alternativa: 128 – 3089%128 = 128-17 = 111 (Byte)

   Si determini la stringa dei riferimenti a pagine (Si consiglia di passare da esadecimale a binario, per determinare correttamente il numero di pagina e, se necessario, il displacement/offset). Si utilizzi un algoritmo di sostituzione pagine di tipo LRU (Least Recently Used). Si assuma che siano disponibili 3 frame, agli indirizzi fisici (espressi in esadecimale) 780, A00, B00. Si richiede la visualizzazione (dopo ogni accesso) del resident set (i frame fisici contenenti pagine logiche).
   Determinare quali e quanti page fault (accessi a pagine non presenti nel resident set) si verificheranno.
   Si dica infine a quali indirizzi fisici vengono effettuati gli accessi (tra quelli sopra elencati) R 3F5, W 4D3, R 3BA

   *Indirizzi logici (p,d) (pagina,displacement). Una pagina contiene 128 Byte, quindi per d servono 7 bit, per p 5 bit. Riferimenti in binario: R (0011 1,111 0101), R (0011 0,110 0011), W (0100 1,101 0011). Stringa dei riferimenti (basta p non serve d): 7 (2*3+1), 6 (2*3+0), 9, 8, 9, 5, 8, 5, 7, 9, 9, 5, 1, 2 (si noti che p, dato dai 5 bit più significativi dell'indirizzo logico, può essere velocemente calcolato come il doppio della prima cifra esadecimale + il bit più significativo della seconda cifra esadecimale).*

Riferimenti		7	6	9	8	9	5	8	5	7	9	9	5	1	2
Resident Set	780	7	7	7	8		8			8	9			9	2
	A00		6	6	6		5			5	5			5	5
	B00			9	9		9			7	7			1	1
Page Fault		*	*	*	*		*			*	*			*	*



Indicare nella prima riga la stringa dei riferimenti a pagine (rappresentate a scelta in esadecimale o decimale), nelle tre successive (che rappresentano i 3 frame del resident set), le pagine allocate nei corrispondenti frame. La free frame list sia inizialmente (780, A00, B00). Nell'ultima riga si indichi la presenza o meno di un Page Fault.

Numero totale di page fault: 9

Indirizzi logici: R 3F5 (**0011 1**,111 0101) W 4D3: (**0100 1**,101 0011) R 3BA: (**0011 1**, 011 1010)
Indirizzi fisici: R (0111 1,111 0101) = 7F5, W (1011 0,101 0011) = B53, R (1011 0, 011 1010) = B3A

Attenzione: La pagina logica 7 viene posta in due frame diversi in due momenti diversi

2. *(3.5 punti)* Siano dati due file system F1 e F2, su due volumi, basati rispettivamente su FAT e su Inode. I puntatori hanno dimensione di 32 bit, i blocchi hanno dimensione 4KB, entrambi i volumi hanno una parte riservata ai metadati (direttori, FAT e FCB (File Control Block) oppure Inode, blocchi indice, ecc.) e una parte ai blocchi di dato. Si supponga che lo spazio riservato ai blocchi di dato sia lo stesso per F1 e F2. Si sa che la FAT occupa 150 MB.

   Quanti blocchi di dato possono contenere complessivamente, al massimo, i file presenti nei due file system?

   - N blocchi F1: *Ad ogni entry nella FAT corrisponde un blocco dato N_{F1} = 150 MB / 4B = 37,5 M blocchi*

   - N blocchi F2: *$N_{F2} = N_{F1}$ = 37,5 M blocchi (per definizione)*

   Quale è il numero massimo di blocchi dato liberi ?

   - F1: *(stesso valore di quelli occupati) 37,5 M blocchi*

   - F2: 37,5 *M blocchi (come sopra)*

   Si supponga che i blocchi dato liberi di F1 siano gestiti mediante free list.

   - La free list di F1 può essere rappresentata direttamente nella FAT?

     (SI/NO, motivare) SI. La FAT contiene sia le liste dei riferimenti a blocchi occupati dai file che la free-list. Non ha senso fare altrimenti, in quanto gli indici (nella FAT) dei blocchi liberi sono disponibili (non usati per file), quindi utilizzabili per la free list.

   Si vorrebbero organizzare blocchi liberi di F2 mediante lista concatenata di blocchi indice, ognuno dei quali contiene un puntatore al prossimo blocco indice in lista e N_{Free} puntatori a blocchi di dato liberi.

   - E' possibile adottare tale rappresentazione (mentre per i file si usa una rappresentazione basata su Inode)?

     (SI/NO, motivare) SI. La rappresentazione ad Inode non rende necessario che alche i blocchi liberi siano di fatto come un file aggiuntivo accessibile mediante Inode. Volendo, si potrebbe usare una bitmap o una lista semplice di blocchi. La scelta "lista di blocchi indice, ognuno contenente puntatori a blocchi liberi", è quindi lecita.

   - (Se SI alla risposta precedente)
     - *Quale è il valore di N_{Free}? N_{Free} = 4KB/4B-1 = 1023 (numero totale di puntatori in un blocco – il puntatore per il prossimo in lista concatenata)*
     - *Quanti blocchi indice servono, al massimo, per la free list di F2? Ogni blocco indice punta a N_{Free} = 1023 blocchi liberi. Il massimo numero di blocchi liberi è N_{F2} = 37,5 M blocchi. Il numero massimo di blocchi indice per la free list è $N_{IndFree}$ = 37,5 M / 1023 = 37,5 K *1024/1023 ≈ 37,5 K*

   - ~~(Se NO alla risposta precedente)~~
     ~~Dire come andrebbero organizzati i blocchi liberi~~
     ……
     ……

   Siano dati un file "a.mp4" (in F1) di dimensione 317 MB e un file "b.mp4" (in F2) di dimensione 751 MB.

   - Quanti blocchi di dato occupa a.mp4 ? *317 MB / 4 KB = 317/4 K = 79,25 K blocchi*

   - Quanti indici nella FAT sono utilizzati per a.mp4 ? *79,25 K (uguale al numero di blocchi dato)*

   - Quanti blocchi di dato e di indice (escluso l'Inode) occupa b.mp4 ?
     DATO: 751 MB / 4 KB = 751/4 K = 187,75 K
     INDICE: 10 blocchi dato sono puntati direttamente all'Inode. Un blocco indice punta a 1024 = 1K blocchi dato.
     Indice indiretto singolo: 1 blocco indice
     Indici indiretti doppi: a primo livello 1, a secondo livello $\lceil$(187,75 K-10) / 1K$\rceil$ - 1 = 187 (basta indiretto doppio)
     Totale: 1 (singolo) + 1 (doppio primo livello) + 187 (doppio secondo livello) = 189

3. *(3 punti)* Sia dato il modulo di un kernel che deve gestire la schedulazione delle richieste di accesso a un disco.
3a) Perché le politiche SCAN e C-SCAN non servono con dischi SSD?

Perchè le politiche cercano di ridurre la distanza totale percorsa (misurata in differenze tra indici di blocco/cilindro): questo ha senso per i dischi magnetici, nei quali si riduce il tempo di posizionamento delle testine. Per i dischi SSD non ci sono penalità (costi) legati alla differenza tra indici di blocco.

E possibile che il sistema abbia un disco magnetico (A) di 500GB e uno SSD (B) di 200GB, organizzati in modo tale da formare un unico volume?

Si. La tecnologia del disco viene vista al livello basso dei driver dei dispositivi. Partizionamento e formattazione dei dischi permettono sia di ricavare più volumi da un solo disco che di unire più dischi in un unico volume. Le prestazioni potranno ovviamente variare a seconda del dispositivo.

(Se Si alla domanda precedente) quale dimensione ha il volume? 500GB + 200GB = 700GB

3b) Nel sistema (*ATTENZIONE: si tratta di un'altra sotto-domanda, indipendente dalla precedente*) si vogliono supportare paginazione e swapping. E' più efficiente una partizione di swap raw oppure uno swapfile in un filesystem esistente? (motivare la risposta)
E' più efficiente, in termini di tempi di accesso, la partizione raw, in quanto si gestiscono/utilizzano direttamente blocchi fisici su disco, senza passare dai vari livelli del file system.

3c) Si vuole realizzare un nuovo tipo di filesystem, nel quale è possibile avere sia file che direttori condivisi (l'albero di direttori è un DAG!).
Perché eventuali cicli nel grafo dei direttori rappresentano un problema?
Appunto perché con i cicli il grafo non è un DAG, cioè un direttorio avrebbe come discendente/successore nell'albero/DAG un suo antenato/predecessore! Questo va evitato.

Quali strategie è possibile adottare per ovviare al problema (dei cicli)?
a) Condividere solo file (le foglie) e non i direttori (ma questo limiterebbe la richiesta: si vogliono "sia file che direttori condivisi")
b) Garbage collection: si accettano i cicli (per non fare controlli ad ogni operazione di link/condivisione di file/direttorio), ma periodicamente si fa eseguire sul file system una funzione di controllo che individua e rimuove eventuali cicli
c) Ad ogni generazione di un nuovo link (a file/direttorio esistente) si fa un controllo di presenza di eventuali cicli, che determina la validità o meno dell'operazione.

4. *(4 punti)* Sia dato un sistema operativo OS161.

4a) Si supponga di aver definito l'opzione `lab6` (`defoption lab6`) nel file `conf.kern`. Si vuole rendere opzionale (cioè compilata sono nel caso in cui sia abilitata l'opzione `lab6`) l'istruzione in linguaggio C (nella funzione `boot`):

```
Lab6Boot("Aggiunta per LAB6");
```

come occorre adattare il file main.c ? *(si facciano correzioni/aggiunte direttamente sul programma)*

```
/* main.c */
#include <types.h>
#include <kern/errno.h>
...
#include <version.h>
#include "autoconf.h"  // for pseudoconfig
#include "opt-lab6.h"  // oppure inserire questa riga in un .h già incluso

int boot (void) {
   ...

#if OPT_LAB6
   Lab6Boot("Aggiunta per LAB6");
#endif

   ...
}
```

4b) Si spieghi brevemente cosa sono switchframe e trapframe e a quale scopo vengono utilizzati:
switchframe: si tratta di una struttura dati in cui viene salvato il contesto di un thread prima di un context switch (avvicendamento su una CPU). Si trova nello stack di kernel di un thread. Contiene, in particolare, i valori di 10 registri. Lo switchframe viene usato dallo schedulet per salvare/ripristinare lo stato dei thread.

trapframe: una struttura dati simile, in cui viene salvato lo stato per servire una eccezione/trap. Questo vale anche per l'esecuzione di una system call, che è trattata come una trap. Un trapframe viene anche usato per gestire l'avvio di un processo user, quando mips_usermode passa ad eseguire un programma user appena caricato in un addrspace, dato il suo entrypoint: di fatto l'avvio viene assimilato a un ritorno da trap.

In quale parte della memoria virtuale/logica sono allocati i due frame? Memoria kernel o user?
*Lo switchframe è nel kernel stack di un thread (si veda il campo struct switchframe *t_context; di una struct thread). Il trapframe è un dato locale del kernel (quindi in uno stack kernel o nel kernel heap, qualora fosse allocato dinamicamente). Entrambi sono quindi in memoria kernel.*

Di ognuno questi indirizzi logici, si dica se è plausibile (in un sistema OS161 con processore MIPS) come indirizzo iniziale di uno switchframe o trapframe:
A) 0x7FFFFF00 (NO. E' un indirizzo logico USER)
B) 0x80044DB8 (SI. E' un indirizzo logico kernel valido)
C) 0x80056B0A (NO. Pur essendo un indirizzo kernel valido, non è multiplo di 4: per salvare registri serve un frame allineato a indirizzi multipli di 4).

4c) Sia data la porzione della funzione load_elf rappresentata in figura, in cui si vuole leggere dal file ELF l'header del file, avente come destinazione la struct eh.

```
load_elf(struct vnode *v, vaddr_t *entrypoint)
{
    Elf_Ehdr eh;   /* Executable header */
    int result;
    struct iovec iov;
    struct uio ku;
    /*
     * Read the executable header from offset 0 in the file.
     */
    result = VOP_READ(v, &eh, sizeof(eh));
   ...
}
```

Il programma riportato è errato. Si spieghi perché e si proponga la correzione necessaria.

La chiamata a VOP_READ è errata. La lettura va fatta con una strategia diversa: prima si definisce l'operazione da effettuare, mediante `uio_kinit` (per operazione in memoria kernel), usando le variabili ku e iov, poi si chiama VOP_READ, utilizzando ku. La versione corretta è (non è fondamentale l'ordine "esatto" dei parametri a `uio_kinit`):

```
uio_kinit(&iov, &ku, &eh, sizeof(eh), 0, UIO_READ);
result = VOP_READ(v, &ku);
```

Si dica poi (in breve) che cosa sono (e a cosa servono nel programma proposto):

- *il parametro v: è il puntatore a vnode (il file control block) del file ELF da cui si legge*
- *la variabile ku: la struct nella quale va impostata l'operazione di IO (mediante uio_kinit) prima di effettuarla (mediante VOP_READ). Punta a iov (seguente), contiene campi per definire lettura/scrittura kernel/user, e altro*
- *la variabile iov; la struct in cui effettivamente vengono caricati indirizzo in memoria e dimensione, per la destinazione di una VOP_READ (oppure sorgente di una VOP_WRITE)*