

PROGETTO DI SISTEMI OPERATIVI

Ingegneria Informatica

13 settembre 2010

(teoria)

(si prega di rispondere descrivendo i passaggi e i risultati intermedi)

1. (6 punti) Sia dato un sistema con paginazione che utilizza una “inverted page table” (IPT). Si vuole realizzare in linguaggio C una funzione che effettui la ricerca nella IPT. Il prototipo della funzione è:

```
unsigned IPTsearch (unsigned *IPT, unsigned IPTsize, unsigned pagenum);
```

I parametri rappresentano, rispettivamente, il puntatore alla IPT, la sua dimensione (numero di entry) e in numero di pagina (logica). Il valore di ritorno è il numero di frame (fisico). Si supponga che la IPT sia realizzata senza l'utilizzo di una tabella di HASH per velocizzare le ricerche. Si richiede di:

- Realizzare la funzione in linguaggio C
 - Discutere l'efficienza (o il costo) in termini di velocità di ricerca, illustrando il miglioramento ottenibile mediante una tabella di HASH
 - Esprimere (mediante espressioni matematiche) i tempi medio e massimo di un accesso in memoria, ipotizzando l'utilizzo di una TLB con hit ratio α e la IPT proposta. Si indichino con $|M|$ e $|P|$ le dimensioni della memoria fisica (totale) e di una pagina (o frame).
2. (6 punti) Si descrivano le caratteristiche principali del “memory mapping” di un file, nell'ambito di un sistema operativo. Si dica, in particolare, quale utilizzo se ne può fare sia nell'ambito di un processo che nell'ambito della comunicazione tra più processi. Tenendo conto che il “memory mapped file” è un caso particolare di “memory-mapped I/O”, si spieghi ch'è cosa è e perché è utile lo “unified buffer-cache”.
3. (6 punti) Sia dato un file system simile a Unix, basato su inode aventi 13 puntatori (10 diretti, 1 indiretto singolo e 1 doppio). I puntatori hanno dimensione di 32 bit e i blocchi di dato dei file hanno dimensione 4 Kbyte. Il blocco di indice indiretto doppio contiene un puntatore riservato. Qualora il livello doppio non sia sufficiente, si ricorre ad un ulteriore blocco di indice indiretto doppio (puntato da tale puntatore). Si ha quindi una lista di indici indiretti doppi. Supponendo che un file abbia la dimensione di 210123045 Byte, si determini il numero totale di blocchi (di dato e di indice) utilizzati per tale file (illustrandone la struttura). Si calcoli poi, per tale file, la frammentazione dovuta all'organizzazione basata su inode.
4. (6 punti) Siano date due funzioni di lettura da dispositivi di I/O, una di tipo sincrono (`readSynch`) e una di tipo asincrono (`readAsynch`). Nel seguito sono proposte alcune chiamate alle funzioni:

```
readSynch(fp1, buf1, sizeof(buf_t),...);
readAsynch(fp2, buf2, sizeof(buf_t),...);
a = buf1;
b = buf2;
readSynch(fp1, buf1, sizeof(buf_t),...);
readAsynch(fp2, buf2, sizeof(buf_t),...);
...
```

Il programma non è completo (si sono utilizzati i ... per indicare parti mancanti). Le funzioni `readSynch` e `readAsynch` realizzano letture sincrone/asincrone tra un file ed un buffer, di cui si forniscono il puntatore e la dimensione. Si dica se il programma può funzionare, nella forma proposta, e/o come va modificato, o a quali condizioni lo si può utilizzare, affinché le chiamate ad I/O funzionino correttamente. Si chiede, in particolare, di discutere la consistenza dei dati presenti in `buf1` e `buf2` dopo le letture.

5. (6 punti) Sia riportano nel seguito (retro del foglio) alcune parti del file `dumbvm.c` di OS161. Le righe sono numerate. Si sono volutamente riportate alcune parti ritenute più significative. Si spieghi brevemente lo scopo delle funzioni riportate. Si dica poi come viene gestito in tale contest il rilascio di uno spazio di memoria. E' possibile liberare una parte di memoria, affinché sia disponibile per una nuova allocazione? In caso positivo, si dica perché; in caso negativo, si suggerisca una modifica (non è richiesto il codice C, ma sono un'indicazione di strutture dati e/o algoritmi) ai programmi sotto-elencati, tale da consentire tale liberazione e ri-allocazione.

```

00156 struct addrspace *
00157 as\_create(void)
00158 {
00159     struct addrspace *as = kmallocc(sizeof(struct addrspace));
00160     if (as==NULL) {
00161         return NULL;
00162     }
00163
00164     as->as_vbase1 = 0;
00165     as->as_pbase1 = 0;
00166     as->as_npages1 = 0;
00167     as->as_vbase2 = 0;
00168     as->as_pbase2 = 0;
00169     as->as_npages2 = 0;
00170     as->as_stackbase = 0;
00171
00172     return as;
00173 }
00174
00175 void
00176 as\_destroy(struct addrspace *as)
00177 {
00178     kfree(as);
00179 }
00180
00181 void
00182
00183 ...
00197 int
00198 as\_define\_region(struct addrspace *as, vaddr\_t vaddr, size\_t sz,
00199                  int readable, int writeable, int executable)
00200 {
00201     size\_t npages;
00202
00203     /* Align the region. First, the base... */
00204     sz += vaddr & ~(vaddr\_t)PAGE\_FRAME;
00205     vaddr &= PAGE\_FRAME;
00206
00207     /* ...and now the length. */
00208     sz = (sz + PAGE\_SIZE - 1) & PAGE\_FRAME;
00209
00210     npages = sz / PAGE\_SIZE;
00211
00212
00213 ...
00216
00217     if (as->as_vbase1 == 0) {
00218         as->as_vbase1 = vaddr;
00219         as->as_npages1 = npages;
00220         return 0;
00221     }
00222
00223     if (as->as_vbase2 == 0) {
00224         as->as_vbase2 = vaddr;
00225         as->as_npages2 = npages;
00226         return 0;
00227     }
00228
00229     /*
00230      * Support for more than two regions is not available.
00231      */
00232     kprintf("dumbvm: Warning: too many regions\n");
00233     return EUNIMP;
00234 }

```

PROGETTO DI SISTEMI OPERATIVI
Ingegneria Informatica (a.a. 2008/2009 e precedenti)
13 settembre 2010

(teoria)

(si prega di rispondere descrivendo i passaggi e i risultati intermedi)

1. (6 punti) Sia dato un sistema con paginazione che utilizza una “inverted page table” (IPT). Si vuole realizzare in linguaggio C una funzione che effettui la ricerca nella IPT. Il prototipo della funzione è:

`unsigned IPTsearch (unsigned *IPT, unsigned IPTsize, unsigned pagenum);`

I parametri rappresentano, rispettivamente, il puntatore alla IPT, la sua dimensione (numero di entry) e in numero di pagina (logica). Il valore di ritorno è il numero di frame (fisico). Si supponga che la IPT sia realizzata senza l'utilizzo di una tabella di HASH per velocizzare le ricerche. Si richiede di:

- Realizzare la funzione in linguaggio C
 - Discutere l'efficienza (o il costo) in termini di velocità di ricerca, illustrando il miglioramento ottenibile mediante una tabella di HASH
 - Esprimere (mediante espressioni matematiche) i tempi medio e massimo di un accesso in memoria, ipotizzando l'utilizzo di una TLB con hit ratio α e la IPT proposta. Si indichino con $|M|$ e $|P|$ le dimensioni della memoria fisica (totale) e di una pagina (o frame).
2. (6 punti) Si descrivano le caratteristiche principali del “memory mapping” di un file, nell'ambito di un sistema operativo. Si dica, in particolare, quale utilizzo se ne può fare sia nell'ambito di un processo che nell'ambito della comunicazione tra più processi. Tenendo conto che il “memory mapped file” è un caso particolare di “memory-mapped I/O, si spieghi ch  cosa   e perch    utile lo “unified buffer-cache”.
3. (6 punti) Sia dato un file system simile a Unix, basato su inode aventi 13 puntatori (10 diretti, 1 indiretto singolo 1 doppio). I puntatori hanno dimensione di 32 bit e i blocchi di dato dei file hanno dimensione 4 Kbyte. Il blocco di indice indiretto doppio contiene un puntatore riservato. Qualora il livello doppio non sia sufficiente, si ricorre ad un ulteriore blocco di indice indiretto doppio (puntato da tale puntatore). Si ha quindi una lista di indici indiretti doppi. Supponendo che un file abbia la dimensione di 210123045 Byte, si determini il numero totale di blocchi (di dato e di indice) utilizzati per tale file (illustrandone la struttura). Si calcoli poi, per tale file, la frammentazione dovuta all'organizzazione basata su inode.
4. (6 punti) Siano date due funzioni di lettura da dispositivi di I/O, una di tipo sincrono (`readSynch`) e una di tipo asincrono (`readAsynch`). Nel seguito sono proposte alcune chiamate alle funzioni:

```
readSynch(fp1, buf1, sizeof(buf_t),...);
readAsynch(fp2, buf2, sizeof(buf_t),...);
a = buf1;
b = buf2;
readSynch(fp1, buf1, sizeof(buf_t),...);
readAsynch(fp2, buf2, sizeof(buf_t),...);
...
```

Il programma non   completo (si sono utilizzati i ... per indicare parti mancanti). Le funzioni `readSynch` e `readAsynch` realizzano letture sincrone/asincrone tra un file ed un buffer, di cui si forniscono il puntatore e la dimensione. Si dica se il programma pu  funzionare, nella forma proposta, e/o come va modificato, o a quali condizioni lo si pu  utilizzare, affin  che le chiamate ad I/O funzionino correttamente. Si chiede, in particolare, di discutere la consistenza dei dati presenti in `buf1` e `buf2` dopo le letture.

5. (6 punti) Si descrivano, nell'ambito della schedulazione di processi, le multilevel feedback queues, facendo notare, in particolare, la politica di schedulazione realizzata e i criteri di ottimalit  scelti a tale scopo.

Con quali criteri vanno realizzate le code al fine di privilegiare

- a. i job di durata breve ?
- b. i job lunghi ?
- c. i job che fanno molto I/O ?

Ha senso utilizzare, in uno schema multilevel, 2 code consecutive di tipo Round-Robin con quanto di tempo uguale (in altri termini, si dica se questo caso differisce da quello di un'unica coda RR) ? (si giustifichi la risposta).