Programmazione di sistema

Iniziato	venerdì, 14 maggio 2021, 20:00
Stato	Completato
Terminato	venerdì, 14 maggio 2021, 22:05
Tempo impiegato	2 ore 4 min.
In ritardo	4 min. 4 secondi
Valutazione	Non ancora valutato

Domanda 1

Completo

Punteggio max.: 3,00

TUTTE LE RISPOSTE SÌ / NO VANNO MOTIVATE. PER LE RISPOSTE NUMERICHE, SONO RICHIESTI SIA I RISULTATI CHE I PASSAGGI INTERMEDI (O FORMULE) RILEVANTI

Si consideri il seguente frammento di codice:

```
#define N 512

float M[2*N][N],V[2*N*N];
int i,j,k,t;
...
for (i=t=0; i<2*N; i++) {
  k = (i<N) ? N-i : i-N+1;
  for (j=0; j<k; j++) {
    V[t++] = M[i][j];
  }
}
```

Il codice macchina generato da tali istruzioni viene eseguito su un sistema con gestione della memoria basata su demand paging (paginazione a richiesta), pagine da 2 KB, utilizzando una politica di sostituzione pagine working set (versione esatta) con delta=10.
Si sa che:

- la dimensione di un `float` è di 32 bit;
- il segmento di codice (istruzioni in codice macchina) ha dimensioni inferiori a una pagina;
- `M` e `V` sono allocati ad indirizzi logici contigui (prima M, poi V), a partire dall'indirizzo logico 0xA720AC00;
- la matrice `M` è allocata seguendo la strategia "row major", ovvero per righe (prima riga, seguita da seconda riga, ecc..)

Si risponda alle seguenti domande

A) Quante pagine (e frame) sono necessarie per contenere la matrice e il vettore?

B) Supponiamo ora che le variabili i, j, k e t siano allocate in registri della CPU (accedendo a tali registri non si fa quindi alcun accesso in memoria RAM), quanti accessi totali a memoria $N_T = N_W+N_R$ (N_R per la lettura e N_W per la scrittura di dati) produce il programma proposto (non vanno conteggiati gli accessi a istruzioni)?

C) Sia N_T la quantità totale di riferimenti a dati in memoria (omettiamo per semplicità il fetch delle istruzioni) e sia N_L il numero di riferimenti (tra gli N_T totali) a una pagina già letta nei precedenti 10 accessi. Definiamo come **località** del programma per i dati il rapporto $L = N_L/N_T$. Si calcoli la località del programma proposto.

D) Si calcoli il numero di page fault generati dal programma proposto. (Motivare la risposta)

A) Quante pagine (e frame) sono necessarie per contenere la matrice e il vettore?

|V| =
|M| =

B) Supponiamo ora che le variabili i, j, k e t siano allocate in registri della CPU (accedendo a tali registri non si fa quindi alcun accesso in memoria RAM), quanti accessi totali a memoria $N_T = N_W + N_R$ (N_R per la lettura e N_W per la scrittura di dati) produce il programma proposto (non vanno conteggiati gli accessi a istruzioni)?

N_R =
N_w =
N_T =

C) Sia N_T la quantità totale di riferimenti a dati in memoria (omettiamo per semplicità il fetch delle istruzioni) e sia N_L il numero di riferimenti (tra gli N_T totali) a una pagina già letta nei precedenti 10 accessi. Definiamo come **località** del programma per i dati il rapporto $L = N_L/N_T$. Si calcoli la località del programma proposto.

N_L =
L =

D) Si calcoli il numero di page fault generati dal programma proposto. (Motivare la risposta)

N_{PF} =

A) Quante pagine (e frame) sono necessarie per contenere la matrice e il vettore?

N = 512 = 1/2K
2*N = 1K
N*N = 1/4M = 256K
2KB/4B = 512, quindi 1 pagina contiene 512 float

|V| = 2*N*N*sizeof(float) = 2*256K * 4B = 2MB = 2MB/2KB pagine = 1K pagine = 1024 pagine
|M| = |V| = 2MB = 1024 pagine
L'indirizzo di partenza 0xA720AC00 NON è multiplo della dimensione di una pagina (termina con 10 zeri, mentre dovrebbe terminare con 11 bit a zero), inizia invece a ½ di pagina (C in binario è 1100).
Dobbiamo quindi modificare leggermente i nostri risultati precedenti: V si sovrappone a 1025 pagine e M a 1025 pagine (la prima condivisa con V): 2049 pagine in totale.

Supponiamo ora che le variabili i, j, k e t siano allocate in registri della CPU (accedendo a tali registri non si fa quindi alcun accesso in memoria RAM), quanti accessi totali a memoria $N_T = N_W+N_R$ (N_R per la lettura e N_W per la scrittura di dati) produce il programma proposto (non vanno conteggiati gli accessi a istruzioni)?

Notazione: N_i numero iterazioni del for esterno
N_j numero iterazioni del for interno
Soluzione:

```
for (i=0;i<2*N;i++){ // Ni = 2*N = 1K iterazioni
// per le prime N iterazioni di i, k varia da N-1 a 0
// per le successive N iterazioni di i, k varia da 0 fino a N-1
for (j=0;j<k;j++){ // il valore medio di k è (N+1)/2, ripetuto per tutti i valori di i
// Nj = 2*SUM_{k=1..N}(k) = 2N(N+1)/2 = N(N+1) iterazioni,
Nj = 512*513 = 256,5K = 262656
V[t++] = M[i][j]; // NR = Nj letture, Nw = Nj scritture, NT = NR + NW = 2*Nj = 513K
```

Soluzione alternative (veloce).
La matrice proposta è una matrice rettangolare che può essere vista come l'unione di due matrici quadrate sovrapposte aventi dimensione `N*N`. A causa del modo in cui `k` viene calcolato, il suo valor medio è `(N+1)/2`, quindi il numero medio di iterazioni `j` per ogni valore di `i` è `(N+1)/2`. Complessivamente, il numero di iterazioni del loop interno è `2N*(N+1)/2`, con una lettura ed una scrittura per ogni iterazione. Quindi $N_j = N*(N+1) = 512*513$, $N_R = N_w = N_j$ (una lettura + una scrittura per iterazione). $N_T = N_R + N_W = 2*N_j$

Sia N_T la quantità totale di riferimenti a dati in memoria (omettiamo per semplicità il fetch delle istruzioni) e sia N_L il numero di riferimenti (tra gli N_T totali) a una pagina già letta nei precedenti 10 accessi. Definiamo come **località** del programma per i dati il rapporto $L = N_L/N_T$. Si calcoli la località del programma proposto.

Ogni riga di M si sovrappone a due pagine, poiché la prima pagina contiene solo metà della prima riga e l'ultima riga è condivisa con V. Non tutte le celle di M vengono lette, ma tutte le pagine di M vengono lette almeno una volta: l'unico accesso non locale è il primo a ogni pagina.

V è scritto sequenzialmente, ma non completamente: solo per le prime N_j celle, con N_j = `512*513` (come precedentemente calcolato). V condivide (metà e metà, quindi 1 KB = 256 float) la prima pagina con M. Le pagine rimanenti ammontano a *ceil* ((N_j - 256) / 512) = *ceil* (512,5) = 513 pagine.

Le due prime righe sono nella stessa pagina dell'ultima pagina di M.
Per tutte le pagine a cui si accede, abbiamo solo un accesso non locale (la pagina condivisa da M e V ha 2 accessi non locali, uno durante la scrittura di V e uno durante la lettura di M)

N_{NL} = 1025 (pagine di M) + 1 (pagina di V condivisa con M) + 513 (altre pagine di V) = 1539

$N_L = N_T - N_{NL}$
$L = N_L/N_T = (N_T - N_{NL})/N_T = 1 - N_{NL}/N_T = 1 - 1539/(1024*513) = 1 - 0.0029 =$ 0,9971

Si calcoli il numero di page fault generati dal programma proposto. (Motivare la risposta)

Non è necessaria una simulazione dettagliata, poiché (a causa dell'elevata località del programma) è possibile stimare facilmente i page fault.

Esiste esattamente un page fault per ogni accesso non locale, quindi N_{PF} = 1539

Domanda 2

Risposta non data

Punteggio max.: 3,00

TUTTE LE RISPOSTE SÌ / NO VANNO MOTIVATE. PER LE RISPOSTE NUMERICHE, SONO RICHIESTI SIA I RISULTATI CHE I PASSAGGI INTERMEDI (O FORMULE) RILEVANTI

Sia dato un disco organizzato con blocchi fisici e logici di dimensione 8KB. Il disco contiene più partizioni: la partizione A, di NB blocchi, è formattata per un file system che alloca staticamente NM blocchi per i metadati (che includono directory, file control blocks e una bitmap per la gestione dello spazio libero) e ND blocchi per i dati dei file. La bitmap ha un bit per ciascuno degli ND blocchi di dati. NM/2 blocchi di metadati sono riservati alla bitmap.

Si risponda alle seguenti domande:

A) Si calcoli il rapporto ND/NM.

B) Supponendo che la bitmap indichi un rapporto blocchi liberi / usati del 25% (quindi 1 blocco libero ogni 4 usati), si calcoli (in funzione di NM) la dimensione massima per un intervallo contiguo di blocchi liberi, assumendo la configurazione più favorevole della bitmap. Si dia la stessa risposta anche assumendo la configurazione della bitmap meno favorevole.

C) Si supponga che un file control block (FCB) abbia dimensione 256B e NM/4 blocchi di metadati siano riservati agli FCB, per un massimo di 16K file. Si calcolino ND, NM e NB. Si esprima anche la dimensione della bitmap e della partizione A, espressa in Byte.

A) Si calcoli il rapporto ND/NM:

|bitmap| = NM/2 blocchi = NM/2*8K*8 bit = 32K*NM bit
ogni bit della bitmap corrisponde a uno degli ND blocchi di dati
ND = 32K*NM
ND/NM = 32K

B) Supponendo che la bitmap indichi un rapporto blocchi liberi / usati del 25% (quindi 1 blocco libero ogni 4 usati), si calcoli (in funzione di NM) la dimensione massima per un intervallo contiguo di blocchi liberi, assumendo la configurazione più favorevole della bitmap. Si dia la stessa risposta anche assumendo la configurazione della bitmap meno favorevole.

N_{free}/N_{used} = 0,25 => N_{free} = 0.20*(N_{free}+N_{used}) = 0.2*(N_{bits}) = 0.2*32K*NM bits = 6.4K*NM bits

La situazione più favorevole è quando tutti i blocchi liberi sono contigui:
N_{good} = 6.4K*NM

La situazione meno favorevole è quando tutti i blocchi liberi non hanno altri blocchi liberi adiacenti:
N_{bad} = 1

C) Si supponga che un file control block (FCB) abbia dimensione 256B e NM/4 blocchi di metadati siano riservati agli FCB, per un massimo di 16K file. Si calcolino ND, NM e NB. Si esprima anche la dimensione della bitmap e della partizione A, espressa in Byte.

Il numero massimo di file è 16K
Un blocco può contenere 8KB/256B = 32 FCB
Il numero massimo di FCBs is 32*NM/4 = 8*NM
Poiché i file corrispondono agli FCB:
8*NM = 16K
NM = 2K
ND = 32K*NM = 64M
NB = ND + NM = 64M + 2K
|A| = (64M+2K)*8KB = 512GB + 16MB
|bitmap| = NM/2*8KB = 8MB

Domanda 3

Risposta non data

Punteggio max.: 3,00

TUTTE LE RISPOSTE SÌ / NO VANNO MOTIVATE. PER LE RISPOSTE NUMERICHE, SONO RICHIESTI SIA I RISULTATI CHE I PASSAGGI INTERMEDI (O FORMULE) RILEVANTI

Si risponda alle seguenti domande sulla gestione della memoria:

A) Si consideri il caricamento dinamico (dynamic loading) e il link dinamico (dynamic linking). È possibile caricare dinamicamente un programma senza cha sia necessario il dynamic linking? Il dynamic linking richiede che un programma sia anche caricabile dinamicamente (dynamic loading)?

B) Si spieghi brevemente perché un'Inverted Page Table necessiti di un campo pid (ID del processo) in ciascuna delle sue entry, mentre ciò non è vero per una tabella di pagine standard.

C) Si consideri una CPU dotata di TLB: la TLB può contenere entry di più processi o è vincolata a contenere entry per un solo processo?

D) Sia data una CPU a 64 bit, la quale gestisce indirizzi fisici di soli 42 bit e paginazione gerarchica, indirizzi logici a 64 bit suddivisi in (p1, p2, d), di dimensioni (40,12,12) bit, rispettivamente. Si calcoli il numero minimo di bit necessari per una entry nella TLB, considerando che ogni entry di TLB include i bit di "validità", "modifica" e 3 bit di protezione della pagina.

A) È possibile caricare dinamicamente (dynamic loading) un programma senza cha sia necessario il dynamic linking?

Sì. Sebbene il link dinamico possa essere combinato al caricamento (load) dinamico, non è obbligatorio: un programma può essere linkato staticamente e caricato in modo dinamico/incrementale, ad es. caricamento dinamico controllato dal programma (program-based dynamic loading).

Nota: caricamento (load) dinamico significa chei pezzi di un programma vengono caricati in memoria solo se/quando necessari, il link dinamico significa che i riferimenti tra moduli vengono risolti in fase di esecuzione (è particolarmente utile per le librerie condivise/shared).

Il dynamic linking richiede che un programma sia anche caricabile dinamicamente (dynamic loading)?

No. Ancora una volta, sebbene il caricamento dinamico possa sfruttare il link dinamico, un programma può essere linkato dinamicamente (ad esempio a una libreria condivisa già caricata in memoria) senza caricamento (load) dinamico

B) Si spieghi brevemente perché un'Inverted Page Table necessiti di un campo pid (ID del processo) in ciascuna delle sue entry, mentre ciò non è vero per una tabella di pagine standard.

Perché l'IPT include "per costruzione" coppie pagina/frame per tutti i processi in un sistema, siccome l'IPT è mappata sulla memoria fisica, e NON su un determinato processo.
Quindi ogni entry ha bisogno di un pid, poiché diversi processi potrebbero avere gli stessi valori p (pagina), ovviamente associati a frame diversi. D'altra parte, una tabella di pagine standard è in genere una tabella per processo, quindi non necessita dei pid. Se una tabella di pagine standard è una PT a livello di sistema, quindi condivisa dai processi, allora è necessario anche un campo pid in ciascuna entry.

C) Si consideri una CPU dotata di TLB: la TLB può contenere entry di più processi o è vincolata a contenere entry per un solo processo?

Esistono entrambi i tipi di TLB: quelle contenenti entry per più processi e quelle contenente solo entry per il processo attualmente attivo. Queste ultime necessitano di funzionalità di azzeramento/reset in corrispondenza a ciascun context switch

D) Sia data una CPU a 64 bit, la quale gestisce indirizzi fisici di soli 42 bit e paginazione gerarchica, indirizzi logici a 64 bit suddivisi in (p1, p2, d), di dimensioni (40,12,12) bit, rispettivamente. Si calcoli il numero minimo di bit necessari per una entry nella TLB, considerando che ogni entry di TLB include i bit di "validità", "modifica" e 3 bit di protezione della pagina.

Una entry di TLB ha (p, f), cioè pagina (compresi sia p1 sia p2), frame, oltre ai bit aggiuntivi (ne sono necessari 5)
|f| vale 42-12 (offset rimosso dall'indirizzo fisico)
|TLB-entry| = 40+12+30+5 = 87 bit

Domanda 4

Completo

Punteggio max.: 3,00

TUTTE LE RISPOSTE SÌ / NO VANNO MOTIVATE. PER LE RISPOSTE NUMERICHE, SONO RICHIESTI SIA I RISULTATI CHE I PASSAGGI INTERMEDI (O FORMULE) RILEVANTI

Si considerino tre kernel thread in OS161, i quali implementano un'attività di trasferimento dati basata sul modello produttore/consumatore (due produttori, un consumatore). I thread condividono una struttura C di tipo `struct prodCons` definita come segue:

```
#define NumP 2

struct prodCons {
  int cnt[NumP];
  int size[NumP];
  struct lock *pc_lk;
  struct cv *pc_cv;
  … /* data buffer handling – omitted */
};
```

Il produttore `i` (con `i = 0` oppure `1`) aggiorna le `i`-esime entry nei vettori `cnt` e `size`. Il consumatore legge entrambi i vettori. Il lock viene utilizzato per la mutua esclusione sui vettori condivisi. Prima di lavorare sui dati, il consumatore deve attendere che una condizione sia vera: ossia che (`cnt[0]+cnt[1]) > minCnt` oppure che (`size[0]+size[1]) > minSize` *(sebbene `cnt` e `size` siano campi di una struttura C, il prefisso, nome della struttura, è stato omesso qui per semplicità)*.
Questo avviene chiamando la funzione:

```
void consumerWait(struct prodCons *pc, int minCnt, int minSize);
```

Si risponda alle seguenti domande:

A) La struttura condivisa può essere posizionata nello stack di thread o dovrebbe essere una variabile globale o altro?

B) Dato che i campi `pc_lk` e `pc_cv` sono puntatori, dove dovrebbero essere chiamate `lock_create()` e `cv_create()`? Nei thread produttori, nel thread consumatore? Altrove?

C) Si fornisca un'implementazione della funzione `consumerWait`. Non è necessario alcun codice produttore e/o consumatore, solo la funzione.

A) La struttura condivisa può essere posizionata nello stack di thread o dovrebbe essere una variabile globale o altro?

B) Dato che i campi `pc_lk` e `pc_cv` sono puntatori, dove dovrebbero essere chiamate `lock_create()` e `cv_create()`? Nei thread produttori, nel thread consumatore? Altrove?

C) Si fornisca un'implementazione della funzione `consumerWait`. Non è necessario alcun codice produttore e/o consumatore, solo la funzione.

```
void consumerWait(struct prodCons *pc, int minCnt, int
minSize) {


}
```

A) La struttura condivisa può essere posizionata nello stack di thread o dovrebbe essere una variabile globale o altro?

Ogni thread ha il proprio stack di thread, quindi i dati condivisi non possono risiedere lì. Una variabile globale è l'opzione più comune. Altre opzioni includono l'allocazione dinamica (con kmalloc), se gestita correttamente.

B) Dato che i campi `pc_lk` e `pc_cv` sono puntatori, dove dovrebbero essere chiamate `lock_create()` e `cv_create()`? Nei thread produttori, nel thread consumatore? Altrove?

La chiamata potrebbe essere fatta in ciascuno di essi, a condizione che si sincronizzino correttamente: ad es. un thread esegue le inizializzazioni, gli altri thread attendono.
Ma una soluzione più comune potrebbe essere che le inizializzazioni vengano eseguite da un altro thread, ossia il thread padre/principale, il quale sta creando i produttori e il consumatore.

C) Si fornisca un'implementazione della funzione `consumerWait`. Non è necessario alcun codice produttore e/o consumatore, solo la funzione.

```
/* le istruzioni di controllo/gestione degli errori sono
omesse per semplicità */
void consumerWait(struct prodCons *pc, int minCnt, int
minSize) {
  /* lock per  mutua esclusione (anche i produttori accedono
usando il lock */
  lock_acquire(pc->pc_lk);
  /* un produttore puo' chiamare cv_signal ad ogni modifica
di cnt/size oppure
     solo quando sa che la condizione attesa dal consumatore
è raggiunta.
     in ogni caso occorre un ciclo, motivato dalla semantica
Mesa */
  while ((pc->cnt[0]+pc->cnt[1])<=minCnt && (pc->size[0]+pc-
>size[1])<=minSize) {
    cv_wait(pc->pc_cv, pc->pc_lk);
  }
  lock_release(pc->pc_lk);
}
```

Domanda 5

Risposta non data

Punteggio max.: 3,00

TUTTE LE RISPOSTE SÌ / NO VANNO MOTIVATE. PER LE RISPOSTE NUMERICHE, SONO RICHIESTI SIA I RISULTATI CHE I PASSAGGI INTERMEDI (O FORMULE) RILEVANTI

Si consideri una possibile implementazione delle system call *open()* e *close()* in OS161. La chiamata di sistema *open()* è implementata dalla una funzione `sys_open()`, che internamente ha la seguente istruzione:

```
err = vfs_open(name, flags, &vn);
```

Si risponda alle seguenti domande:

A) `vn` (ossia vnode) è un puntatore o una struttura C?

B) Si supponga che due processi user chiamino *open()* per lo stesso file, ci si aspetta che (selezionare un'opzione e motivarla):

1 - ogni vfs_open creerà un nuovo vnode, copiato in vn
2 - ogni vfs_open creerà un nuovo vnode, il cui puntatore verrà restituito in vn
3 - solo un vnode verrà creato e copiato in vn (quindi ne avremo due copie)
4 - verrà creato solo un vnode e il puntatore ad esso sarà restituito in vn (quindi avremo due puntatori allo stesso vnode)

C) La tabella di sistema dei file aperti è implementata come segue

```
/* system open file table */
struct openfile {
  struct vnode *vn;
  off_t offset;
  unsigned int countRef;
};
struct openfile systemFileTable[SYSTEM_OPEN_MAX];
```

Perché `struct openfile` include un contatore di riferimenti (countRef), mentre i vnode hanno già internamente un contatore di riferimenti? È un duplicato?
È possibile che una determinata entry nell'array `systemFileTable` sia condivisa (puntata) da due processi diversi?

A) `vn` (ossia vnode) è un puntatore o una struttura C?

È un puntatore, dichiarato come
`struct vnode * vn;`
Non può essere una struttura poiché vfs_open restituisce (“by pointer”, quindi con &) un puntatore a un vnode, allocato in una tabella dedicata, non una copia del vnode

B) Si supponga che due processi user chiamino *open()* per lo stesso file, ci si aspetta che (selezionare un'opzione e motivarla):

1 - ogni vfs_open creerà un nuovo vnode, copiato in vn
2 - ogni vfs_open creerà un nuovo vnode, il cui puntatore verrà restituito in vn
3 - solo un vnode verrà creato e copiato in vn (quindi ne avremo due copie)
4 - verrà creato solo un vnode e il puntatore ad esso sarà restituito in vn (quindi avremo due puntatori allo stesso vnode)

> Numero 4, poiché vfs_open restituisce un puntatore e non una copia (quindi 1 e 3 sono esclusi). Anche la 2 è esclusa in quanto è possibile avere un solo vnode per un determinato file. Se un file viene aperto più volte (in modo concorrente), un nuovo vnode verrà creato una sola volta e referenziato in tutte le successive aperture.

C) La tabella di sistema dei file aperti è implementata come segue

```
/* system open file table */
struct openfile {
  struct vnode *vn;
  off_t offset;
  unsigned int countRef;
};

struct openfile systemFileTable[SYSTEM_OPEN_MAX];
```

Perché `struct openfile` include un contatore di riferimenti (countRef), mentre i vnode hanno già internamente un contatore di riferimenti? È un duplicato?

> Perché sia un `vnode` sia una `struct openfile` possono essere condivisi, in modi diversi: un vnode può essere condiviso da più entry in `systemFileTable`. Una `struct openfile` può essere condivisa da più entry nelle process file table, dato uno stesso file condiviso.

È possibile che una determinata entry nell'array `systemFileTable` sia condivisa (puntata) da due processi diversi?

> Sì. Questo caso ad esempio capita durante il fork di un processo, quando (subito dopo la chiamata a fork) le process file table dei processi padre e figlio, che sono due tabelle distinte, puntano a voci condivise, con i campi countRef correttamente incrementati durante il fork stesso.