Programmazione di sistema

Iniziato	martedì, 18 ottobre 2022, 14:59
Stato	Completato
Terminato	martedì, 18 ottobre 2022, 14:59
Tempo impiegato	11 secondi
Valutazione	**0,00** su un massimo di 15,00 (**0**%)

Domanda 1

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 il seguente frammento di codice:

```
#define N 2048

float M[N][2*N],V[2*N*N], x;
int i,j,t;
...
for (t=0; t<2*N*N; t++) {
  i = t/(2*N);
  j = t%(2*N);
  x = M[i][j];
  V[t] = x;
  M[i][j] = x*2;
}
```

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 0xDB28A800;
- 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 che le variabili i, j, t e x 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?

N = 2048 = 2K
N*N = 4M = 4096K
2KB/4B = 512, quindi 1 pagina contiene 512 float

|V| = 2*N*N*sizeof(float) = 2*4M * 4B = 32MB = 32MB/2KB pagine = 16K pagine = 16384 pagine
|M| = |V| = 32MB = 16K pagine
L'indirizzo di partenza 0xDB28A800 è multiplo della dimensione di una pagina (termina con 11 zeri), quindi non sono necessarie correzioni,

B) Supponiamo ora che le variabili i, j, t e x 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)?

numero di iterazioni for: 2*N*N = 8M

per ogni iterazione: 1 R (su M) + 2 W (V e M)

$N_R = 8M$
$N_w = 16M$
$N_T = N_R + N_W = 24M$

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.

i e j sono calcolati in modo tale da percorrere M in modo sequenziale per righe .
Quindi tutte le pagine di V e M sono lette/scritte completamente non appena vi si accede.
Il numero totale di accessi non locali è pari al numero di pagine

$N_{NL} = 16K$

$N_L = N_T - N_{NL}$
$L = N_L/N_T = (N_T - N_{NL})/N_T = 1 - N_{NL}/N_T = 1 - 16K/24M = 1 - 2/3K = (3K-2)/3K \approx 1$

D) 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 $N_{PF} = 16K$

Domanda 2

Risposta non data

Punteggio max.: 3,00

TUTTE LE RISPOSTE SÌ / NO DEVONO ESSERE MOTIVATE. QUANDO I RISULTATI SONO NUMERI, SONO NECESSARI IL RISULTATO FINALE E I RELATIVI PASSI INTERMEDI (O FORMULE)

Si consideri un "disk scheduler" che gestisce richieste di accesso a blocchi logici su un disco di tipo HDD.
La funzione `enqNextBlk` (abbreviata `eNB`) viene utilizzata per mettere nella coda la richiesta di accesso al blocco `blk`. La funzione `deqNextBlk` viene chiamata per ottenere il nuovo blocco a cui fare accesso (il blocco schedulato) ogni volta che il disco è nuovamente pronto per una operazione. Per semplicità, non si distingue tra operazioni di lettura e scrittura.
Supponendo la seguente sequenza di chiamate, con coda di schedulazione inizialmente vuota e blocco corrente head (posizione iniziale della testina di lettura/scrittura) inizializzato al valore 500:

```
eNB(3050) eNB(5150) N=dNB() eNB(1000) eNB(1050)
N=dNB() eNB(1200) N=dNB() N=dNB() N=dNB()
```

Si indichino i valori successivi di head e di N (accessi a blocchi su disco effettuati) e si rappresenti la coda dopo ogni chiamata a `enqNextBlk` (`eNB`). Si adotti una politica di schedulazione SSTF (Shortest Seek Time First), nella quale dNB ritorna **il blocco più vicino (in una qualunque delle due direzioni) alla testine di lettura scrittura (head)**.

Si calcoli la lunghezza del percorso totale della testina di lettura/scrittura, misurato in numero di cilindri (ogni cilindro contiene 50 blocchi).

Risposta:

Operazione	Head	Coda (dopo l'operazione)	N
	500		
eNB(3050)		3050	
eNB(5150)		3050, 5150	
N=dNB()	3050	5150	3050
eNB(1000)		5150,1000	
eNB(1050)		5150,1000,1050	
N=dNB()	1050	5150,1000	1050
eNB(1200)		5150,1000,1200	
N=dNB()	1000	5150,1200	1000
N=dNB()	1200	5150	1200
N=dNB()	5150		5150

Successione posizioni head (in cilindri): 10,61,21,20,24,103
Distanza totale = (61-10)+(61-21)+(21-20)+(24-20)+(103-24) = 51+40+1+4+79 = 175

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 una tabella di HASH. Perchè la soluzione di IPT + tabella di HASH + diversa da una soluzione con PT basata solamente su tabella di Hash?

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

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 se non si usasse una tabella di HASH occorrerebbe una ricerca lineare del numero di pagina logica (p) nella IPT.
Le due soluzioni differiscono perché, pur essendo molto simili in termini di prestazioni della tabella di HASH, in un caso (con la IPT, vengono inseriti direttamente nelle liste di chaining della tabella di HASH le entry della IPT, mentre nel caso si HASH semplice occorrono le classiche allocazioni e deallocazioni di elementi ad ogni inserimento/cancellazione dalla HASH. Quindi soluzione IPT+HASH è più efficiente nell'uso della memoria.

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

Domanda 4

Risposta non data

Punteggio max.: 3,00

SE I RISULTATI SONO NUMERI, RIPORTARE PASSAGGI INTERMEDI RILEVANTI E/O FORMULE USATE
LE RISPOSTE SI/NO VANNO MOTIVATE

Sia dato un sistema OS161.
Si vuol realizzare il supporto (parziale, in quanto non si gestisce il parametro `flags`) per la system call `waitpid`, di cui si fornisce il prototipo:

```
pid_t waitpid (pid_t pid, int *returncode, int flags);
```

Si è già realizzata una funzione `proc_wait`, avente prototipo:

```
int proc_wait(struct proc *proc);
```

che dato il puntatore al descrittore di un processo, ne attende la fine (del processo) e ne ritorna lo stato di terminazione.

Si realizzi una funzione `sys_waitpid`, che possa essere richiamata nella `syscall` mediante:

```
case SYS_waitpid:
  retval = sys_waitpid(tf->tf_a0, (userptr_t)tf->tf_a1);
  break;
```

ATTENZIONE: Di una eventuale tabella dei processi, è sufficiente fornire la dichiarazione e l'uso all'interno di `sys_waitpid`, nonché una breve descrizione a parole. Non è necessaria la gestione della tabella al boot e alla creazione/distruzione di un processo.

```
/* FILE proc.c

The process table is an array where the i-th entry contains the poi
nter to the
struct proc of the process with pid == i.
The process table has to be updated at each process creation/destru
ction
Function proc_from_pid returns the process, given the pid, using th
e process table
Let’s use a static array (for MAX_PROC processes) – dynamic allocat
ion is an alternative optiom
*/
struct proc *processTable[MAX_PROC];

proc_t *proc_from_pid(pid_t pid) {
 return processTable[pid];
}

/* File proc_syscalls.c (or equivalent)

NOTE: we assume process destruction done in proc_wait
*/
pid_t sys_waitpid (pid_t pid, userptr_t returncodeP) {
  struct proc *proc = proc_from_pid(pid); // from pid to pointer
  if(proc==NULL) return -1; // or another error code
    *returncodeP = proc_wait(proc); // wait and assign returned val
  return pid; // or another value indicating success
}
```

Domanda 5

Risposta non data

Punteggio max.: 3,00

TUTTE LE RISPOSTE SÌ / NO DEVONO ESSERE MOTIVATE. QUANDO I RISULTATI SONO NUMERI, SONO NECESSARI IL RISULTATO FINALE E I RELATIVI PASSI INTERMEDI (O FORMULE)

Si consideri, in un sistema OS161, le definizione delle struct uio e iovec qui riportate (iovec in forma semplificata ma equivalente all'originale):

```
struct uio {
  struct iovec *uio_iov;
  unsigned uio_iovcnt;
  off_t uio_offset;
  size_t uio_resid;
  enum uio_seg uio_segflg;
  enum uio_rw uio_rw;
  struct addrspace *uio_space;
};
struct iovec {
  void *iov_base;
  size_t iov_len;
};
```

A) Spiegare brevemente la funzione svolta dai campi uio_segflg, uio_rw e uio_space;

B) Nel realizzare la system call sys_write, ci si aspetta di che il campo uio_segflg indichi un segmento kernel o user? (motivare)

C) Si dica poi perchè la struct iovec, puntata dal campo uoi_iov, è una struct esterna. Sarebbe possibile evitare la struct aggiuntiva inserendo direttamente i campi iov_base e iov_len nella struct uoi, come riportato qui sotto? (motivare la risposta)

```
struct uio {
  void *iov_base;
  size_t iov_len;
  unsigned uio_iovcnt;
  off_t uio_offset;
  size_t uio_resid;
  enum uio_seg uio_segflg;
  enum uio_rw uio_rw;
  struct addrspace *uio_space;
};
```

A) Spiegare brevemente la funzione svolta dai campi uio_segflg, uio_rw e uio_space;

- uio_segflg: serve a specificare se i dati interessati all'io siano relativi a un segmento user o a memoria kernel. Nel caso di memoria user si discrimina ulteriormente tra segmento codice o dati
- uio_rw: specifica se si tratti di lettura o scrittura
- uio_space: serve solo nel caso di IO relativo a segmento user, e indica l'addrspace da usare per la conversione logico-fisica degli indirizzi.

B) Nel realizzare la system call sys_write, ci si aspetta di che il campo uio_segflg indichi un segmento kernel o user? (motivare)

Un segmento user (in particolare UIO_USERSPACE) in quanto la sys_write scrive su file dati provenienti da un processo user.

C) Si dica poi perchè la struct iovec, puntata dal campo uoi_iov, è una struct esterna. Sarebbe possibile evitare la struct aggiuntiva inserendo direttamente i campi iov_base e iov_len nella struct uoi, come riportato qui sotto? (motivare la risposta)

Perchè il puntatore uio_iov punta alla casella iniziale di un vettore di struct iovec. La soluzione proposta sarebbe compatibile solo con un vettore di una sola casella.
Benchè nei casi visti (load_elf) il vettore abbia in effetti una sola casella, non si tratta del caso generale.