Programmazione di sistema

Iniziato	martedì, 18 ottobre 2022, 15:03
Stato	Completato
Terminato	martedì, 18 ottobre 2022, 15:03
Tempo impiegato	6 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 DEVONO ESSERE MOTIVATE. QUANDO I RISULTATI SONO NUMERI, SONO RICHIESTI SIA IL RISULTATO FINALE SIA I RELATIVI PASSI INTERMEDI (O FORMULE)

Sia dato il frammento di programma rappresentato*:*

```
...
#define N (128*1024)
int i,j;
int A[1024],B[N]
...
for(j=0; j<N; j++) {
  B[j] = 0;
}
for(i=0; i<1024; i++) {
  j = A[i];
  if (j>=0 && j<N)
    B[j]++;
}
...
```

Il codice macchina generato da tali istruzioni viene eseguito in un sistema con memoria virtuale gestita mediante paginazione, con **pagine di 2Kbyte**, utilizzando come politica di sostituzione pagine la **SECOND CHANCE**. Si sa che un `int` ha dimensione **32 bit** e che le istruzioni in codice macchina, corrispondenti al frammento di programma visualizzato, sono contenute in una sola pagina. Si supponga che `A` e `B` siano allocati ad indirizzi logici contigui (prima `A`, poi `B`), a partire dall'indirizzo logico 0x74240A00.

A) Quante pagine (e frame) sono necessarie per contenere i vettori A e B?

B) Ipotizzando che le variabili i, j siano allocate in registri della CPU, quanti accessi in memoria (in lettura e scrittura) fa il programma proposto, per accedere a dati (non vanno conteggiati gli accessi a istruzioni)?

(siccome il costrutto `if (j>=0 && j<N)` *condiziona un'istruzione, si calcolino sia il caso minimo che il caso massimo).*

C) Calcolare il numero di page fault generati dal programma proposto, supponendo che siano allocati per esso **10 frame**, di cui uno utilizzato (già all'inizio dell'esecuzione) per le istruzioni. (considerare nuovamente caso minimo e massimo, e motivare le risposta)

A) Quante pagine (e frame) sono necessarie per contenere i vettori A e B?

|A| = 1024*sizeof(int) = 4KB = 4KB/2KB pagine = 2 pagine

|B| = 128*1024*sizeof(int) = 512KB = 512KB/2KB pagine = 256 pagine

(Si è per ora considerata la pagina unicamente come unità di misura, vediamo ora l'effettiva allocazione di pagine logiche, che corrisponderanno a frame fisici)

L'indirizzo di partenza 0x74240A00 NON è multiplo di pagina (dovrebbe terminare con 11 bit a 0, invece termina con 01 seguito da 9 bit al valore 0), ma inizia a 1/4 di pagina logica. Quindi sono necessarie correzioni: A sta su 3 pagine e B su 257 (la prima in comune con A).

B) Ipotizzando che le variabili i, j siano allocate in registri della CPU, quanti accessi in memoria (in lettura e scrittura) fa il programma proposto, per accedere a dati (non vanno conteggiati gli accessi a istruzioni)?
(siccome il costrutto `if (j>=0 && j<N)` *condiziona un'istruzione, si calcolino sia il caso minimo che il caso massimo).*

Il primo for fa N iterazioni (N = 128K), in ognuna delle quali si fa una scrittura (B[j]=0): 128K write

Il secondo for fa 1024 (1K) iterazioni, in ognuna delle quali si fa una read (j=A[i]), mentre l'istruzione B[j]++, che corrisponde a una read e una write (B[j] = B[j]+1) può essere eseguita o meno, in funzione dei dati.

Caso minimo: *128K write + 1K read*

Caso massimo: *128K write + 1K read + 1K read + 1K write = 129K write + 2K read*

C) Calcolare il numero di page fault generati dal programma proposto, supponendo che siano allocati per esso **10 frame**, di cui uno utilizzato (già all'inizio dell'esecuzione) per le istruzioni. (considerare nuovamente caso minimo e massimo, e motivare le risposta)

La prima iterazione (su B) genera 257 PF (accede a tutte le pagine di B in sequenza)

La seconda iterazione accede a tutte le (3) pagine di A (quindi 3 PF).

- caso minimo, i valori di j sono tali da non accedere mai a B
- caso massimo, trova valori per j tali da accedere a tutte la 257 pagine di B, in un ordine tale da generare un PF per ognuna delle 1024 iterazioni (quindi più di un PF per pagina). *NOTA: correzione rispetto alla versione originale della soluzione, che sottostimava il numero di PF.*

In totale

caso minimo: 257+3 = 260 PF

caso massimo: 257+3+1024 = 1284 PF

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 di tipo “solid state” (SSD).

Una delle tecniche usate per prolungare la vita di un dispositivo SSD è il cosiddetto “wear leveling”, che consiste nel mantenere una tabella (mappa) di corrispondenza tra indirizzi logici e fisici per i blocchi su disco. Implementata in varie modalità, tale tecnica cerca di “spostare” un dato blocco logico da un blocco fisico (pagina, nella terminologia SSD) a un altro in corrispondenza a un’operazione di “ri-scrittura”.

A) Perché è opportuna questa operazione?

B) Da cosa nasce il possibile vantaggio, da un numero minore complessivo di scritture su disco, oppure da qualcos’altro?

C) E’ possibile che un disco SSD continui a funzionare correttamente una volta che una o più pagine (blocchi) fisiche siano marcate come inutilizzabili (in quanto hanno raggiunto il numero massimo di riscritture)? (Se si, spiegare come, se no, motivare)

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

Una delle tecniche usate per prolungare la vita di un dispositivo SSD è il cosiddetto “wear leveling”, che consiste nel mantenere una tabella (mappa) di corrispondenza tra indirizzi logici e fisici per i blocchi su disco. Implementata in varie modalità, tale tecnica cerca di “spostare” un dato blocco logico da un blocco fisico (pagina, nella terminologia SSD) a un altro in corrispondenza a un’operazione di “ri-scrittura”.

A) Perché è opportuna questa operazione?

B) Da cosa nasce il possibile vantaggio, da un numero minore complessivo di scritture su disco, oppure da qualcos’altro?

La durata di un disco SSD è limitata dal numero massimo di cancellazioni. Le cancellazioni sono effettuate per blocchi/pagine e hanno impatto anche sul numero massimo di scritture, in quanto una scrittura (ri-scrittura) deve essere preceduta da cancellazione.
A) Le tecniche di “wear leveling”, disaccoppiando i blocchi logici da quelli fisici, permettono di distribuire cancellazioni e riscritture in modo più uniforme su tutto il disco, riducendo pertanto i numeri massimi di cancellazioni per un dato blocco/pagina.

B) il numero totale di cancellazioni e scritture non cambia, viene solamente distribuito in modo più uniforme su tutti i blocchi.

C) E’ possibile che un disco SSD continui a funzionare correttamente una volta che una o più pagine (blocchi) fisiche siano marcate come inutilizzabili (in quanto hanno raggiunto il numero massimo di riscritture)? (Se si, spiegare come, se no, motivare)

Si. E’ possibile. E’ sufficiente mantenere un flag valid/invalid associato a ogni pagina (blocco). Questo flag va considerato nel momento in cui si fa l’associazione tra blocco logico e fisico, e ovviamente diminuisce leggermente il numero di blocchi fisici disponibili.

Domanda 3

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

Si considerino i due tipi di sincronizzazione, relativi a operazioni di I/O: **sincrono** e **asincrono**.

A) Si spieghino le principali differenze tra gli I/O dei due tipi. Si considerino poi I/O bloccante e non bloccante: si tratta di sinonimi di asincrono e sincrono? Ci sono differenze (tra bloccante/non-bloccante e sincrono/asincrono)? (se no, perché, se si, quali?).

B) Un I/O sincrono può essere effettuato in polling oppure è necessario effettuarlo in interrupt? (si risponda alla stessa domanda per il caso dell'I/O asincrono).

C) In quale modo può un processo beneficiare di un I/O asincrono? E' possibile, nel caso di I/O asincrono, scrivere un programma con istruzioni, successive all'I/O, che dipendano dai dati coinvolti (ad es. letti) durante l'I/O ?

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

A) Si spieghino le principali differenze tra gli I/O dei due tipi. Si considerino poi I/O bloccante e non bloccante: si tratta di sinonimi di asincrono e sincrono? Ci sono differenze (tra bloccante/non-bloccante e sincrono/asincrono)? (se no, perché, se si, quali?).

In modo molto sintetico, si può affermare che

I/O bloccante e sincrono siano equivalenti: il processo che effettua I/O ne attende il completamento, in stato di wait.

I/O non bloccante permette al processo di proseguire, I/O asincrono è di fatto NON bloccante, con l'aggiunta di tecniche che permettano di gestire (successivamente) il completamento dell'I/O. Tali tecniche sono: funzioni di `wait` (dipendono dal sistema operativo) che consentano di attendere il completamento dell'I/O, oppure funzioni di tipo *"callback",* richiamate in modo automatico dal sistema, al completamento dell'I/O (attenzione: sono funzioni che vanno scritte dallo user)

B) Un I/O sincrono può essere effettuato in polling oppure è necessario effettuarlo in interrupt? (si risponda alla stessa domanda per il caso dell'I/O asincrono).

Polling e interrupt sono due modalità diverse per gestire un dispositivo di I/O. Ovviamente il polling risulta meno efficiente, con rare eccezioni. Si tratta tuttavia di un problema interno ai driver (moduli del Kernel) e quindi indipendenti dal processo user. In sostanza, realizzare una system call read/write in modo sincrono o asincrono, può esser fatto con driver che lavorino sia in modo polling che interrupt,

C) In quale modo può un processo beneficiare di un I/O asincrono? E' possibile, nel caso di I/O asincrono, scrivere un programma con istruzioni, successive all'I/O, che dipendano dai dati coinvolti (ad es. letti) durante l'I/O ?

Il processo può beneficiare in quanto può eseguire altre istruzioni mentre l'I/O è in corso. Si tratta quindi di una possibile forma di concorrenza. Il processo non può, durante l'I/O, utilizzare i dati coinvolti. Se deve farlo, occorre sincronizzarsi (e aspettare) sulla relativa (dell'I/O asincrono) operazione di wait.

Domanda 4

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)

Sia dato un sistema OS161. Ci consideri la funzione sys_read, utilizzata per realizzare la system call read. Se ne propone una realizzazione, limitata al caso di standard input.

```
int
sys_read(int fd, userptr_t buf, int size) {
  if (fd == STDIN_FILENO) {
    kgets((char *)buf,size);
    return size;
  }
  else {
    kprintf("sys_read implemented only on stdio\n");
    return -1;
  }
}
```

Si risponda alle seguenti domande:

A) E' possibile che la funzione sia richiamata direttamente da un processo user (ad esempio mediante una chiama interna a palin.c, sort.c, forktest.c)? (motivare la risposta)

B) La funzione è errata, in quanto la chiamata a kgets, così come effettuata, rischia di fare un accesso scorretto in memoria. Si spieghi il perché e si proponga una possibile correzione al problema.

C) Perché la funzione ritorna size, qualora non ci sia errore? E' possibile che, in una realizzazione più completa della funzione, il valore ritornato sia maggiore o minore di size?

A) E' possibile che la funzione sia richiamata direttamente da un processo user (ad esempio mediante una chiama interna a palin.c, sort.c, forktest.c)? (motivare la risposta)

No. Non è possibile in quanto si tratta di una parte del kernel. Un processo user non "vede", come richiamabili, le funzioni del kernel. L'unico modo che ha per attivare una funzione del kernel è la system call.

B) La funzione è errata, in quanto la chiamata a kgets, così come effettuata, rischia di fare un accesso scorretto in memoria. Si spieghi il perché e si proponga una possibile correzione al problema.

La kgets genera una stringa, in cui, oltre ai caratteri acquisiti, aggiunge il terminatore ('\0'). Fatta in questo modo, la funzione modifica non size byte, ma size+1 byte in buf, il che è errato. Per ovviare a tale problema si può effettuare kgets su un altro vettore temporaneo (opportunamente allocato), quindi copiare i caratteri in buf, oppure usare per input getch invece di kgets). *(Si vedano le implementazioni fornite sulle slide e nelle soluzioni proposte per i laboratori).*

C) Perché la funzione ritorna size, qualora non ci sia errore? E' possibile che, in una realizzazione più completa della funzione, il valore ritornato sia maggiore o minore di size?

Perché la funzione deve ritornare il numero di byte correttamente acquisiti: in linea di massima questo valore corrisponde a size, cioè il numero di byte da acquisire in input. Si, è possibile che il valore ritornato sia minore di size (non maggiore), qualora, specie per un input da file, non sia possibile acquisire tutti i byte richiesti (ad esempio perché il file è terminato).

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)

Sia dato un sistema operativo OS161.

A) Si spieghi perché, in un contesto multi-core (sono presenti più CPU) non è possibile realizzare la mutua esclusione semplicemente disabilitando e riabilitando l'interrupt.

B) Dato codice (ridotto alle parti essenziali) delle funzioni di semaforo P e V, riportate in seguito

```
void P(struct semaphore *sem) {
   spinlock_acquire(&sem->sem_lock);
   while (sem->sem_count == 0) {
     wchan_sleep(sem->sem_wchan, &sem->sem_lock);
   }
   sem->sem_count--;
   spinlock_release(&sem->sem_lock);
}
void V(struct semaphore *sem) {
   spinlock_acquire(&sem->sem_lock);
   sem->sem_count++;
   KASSERT(sem->sem_count > 0);
   wchan_wakeone(sem->sem_wchan, &sem->sem_lock);
   spinlock_release(&sem->sem_lock);
}
```

Si risponda alle seguenti domande:

1) a cosa serve lo spinlock (in entrambe le funzioni)?
2) perché la P contiene un ciclo while anziché un if `(sem->sem_count == 0)`?
3) Perché la `wchan_sleep` riceve come parametro lo spinlock?
4) E' possibile che la chiamata alla `wchan_wakeone` svegli più di un thread in attesa su wait_channel?

C) Spiegare brevemente la differenza tra condition variable e wait_channel.

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)

Sia dato un sistema operativo OS161.

A) Si spieghi perché, in un contesto multi-core (sono presenti più CPU) non è possibile realizzare la mutua esclusione semplicemente disabilitando e riabilitando l'interrupt.

> La mutua esclusione mediante disabilitazione dell'interrupt si basa sull'assunzione che la concorrenza possa essere realizzata unicamente interrompendo il processo/thread attualmente in esecuzione. Questo è possibile solo in un sistema single-core. In un multi-core disabilitare l'interrupt su una CPU (core) non previene l'esecuzione di un altro processo/thread su un'altra CPU (core).

B) Dato codice (ridotto alle parti essenziali) delle funzioni di semaforo P e V, riportate in seguito

```
void P(struct semaphore *sem) {
   spinlock_acquire(&sem->sem_lock);
   while (sem->sem_count == 0) {
     wchan_sleep(sem->sem_wchan, &sem->sem_lock);
   }
   sem->sem_count--;
   spinlock_release(&sem->sem_lock);
}
void V(struct semaphore *sem) {
   spinlock_acquire(&sem->sem_lock);
   sem->sem_count++;
   KASSERT(sem->sem_count > 0);
   wchan_wakeone(sem->sem_wchan, &sem->sem_lock);
   spinlock_release(&sem->sem_lock);
}
```

Si risponda alle seguenti domande:

1) a cosa serve lo spinlock (in entrambe le funzioni)?
2) perché la P contiene un ciclo while anziché un if `(sem->sem_count == 0)`?
3) Perché la `wchan_sleep` riceve come parametro lo spinlock?
4) E' possibile che la chiamata alla `wchan_wakeone` svegli più di un thread in attesa su wait_channel?

1) lo spinlock serve a garantire l'accesso in mutua esclusione al contatore `sem->sem_count`. Va aggiunto che lo spinlock è necessario per poter utilizzare il wait channel (proprio perché si suppone che il wait channel sia usato in corrispondenza a una sezione critica).

2) Perché la semantica "Mesa" con cui sono implementati i wait channel, non garantisce che, pur essendo la condizione `sem->sem_count > 0` vera al momento in cui viene eseguita la V, lo sia ancora quando il thread in attesa esce dalla wchan_sleep.

3) Per poterlo rilasciare, e successivamente riacquisirlo prima dell'uscita dalla wchan_sleep.

4) NO. La wchan_wakeone può svegliare solo un thread in attesa. Per svegliare più thread si dovrebbe usare wchan_wakeall.

C) Spiegare brevemente la differenza tra condition variable e wait_channel.

Si tratta di due primitive di sincronizzazione simili nel comportamento. Si potrebbe dire, in modo leggermente improprio, che un wait channel è una condition variable, che invece di usare un lock usa uno spinlock. Proprio per questo, il wchan è una primitiva utilizzabile solo all'interno del kernel. Il wchan è la primitiva fornita in OS161 per realizzare sinconizzazioni di tipo wait/signal, quindi prive di busy wait.