Programmazione di sistema

Iniziato	sabato, 22 ottobre 2022, 17:03
Stato	Completato
Terminato	sabato, 22 ottobre 2022, 17:05
Tempo impiegato	1 min. 25 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)

Si consideri, per un dato processo, la seguente stringa di accessi alla memoria. Per ogni indirizzo (indirizzamento per Byte, con indirizzi espressi in codice esadecimale) viene riportata anche l'operazione di lettura (R) / scrittura (W): R 63F5, R 5A64, W 1AD3, W 6E7E, R 18C8, W 23D1, R 194E, R 5465, W 62A0, R 3BBA, W 1CE6, R 1480, R 6294, R 5AB8. Si supponga che gli indirizzi fisici e logici siano rappresentati su 16 bit, la dimensione della pagina sia 4KByte e A817 sia l'indirizzo massimo utilizzabile dal programma (il limite superiore dello spazio degli indirizzi).

A) Calcolare la dimensione dello spazio di indirizzamento (espressa come numero di pagine) e la frammentazione interna.

B) Simulare un algoritmo di sostituzione delle pagine di tipo LRU (Least Recently Used), supponendo di avere a disposizione 3 frame. Rappresentare il resident set (frame fisici contenenti pagine logiche) dopo ogni accesso alla memoria.

C) Mostrare i page fault (accessi a pagine esterne al resident set) e calcolare il loro conteggio complessivo.

D) spiegare brevemente perché la versione esatta dell'algoritmo LRU è inefficiente e quindi necessita di versioni approssimate.

A) Calcolare la dimensione dello spazio di indirizzamento (espressa come numero di pagine) e la frammentazione interna.

Si noti che una pagina ha dimensione 4KB, quindi servono 12 bit di offset/displacement

Numero totale di pagine nello spazio d'indirizzamento (comprese quelle non nella stringa di riferimento):

A817 = 1010 1000 0001 0111.

L'indice massimo di pagina è 1010 (binario) = 10(decimale) = A(esadecimale) => Quindi lo spazio degli indirizzi ha 11 pagine.

Frammentazione interna: l'ultima pagina viene utilizzata fino al Byte all'offset 1000 0001 0111 = 2K+23 = 2071

Complessivamente, 2K+24 Byte sono utilizzati nell'ultima pagina

Fr.Int. = 4K-(2K+24) = 2024 Bytes.

B) Simulare un algoritmo di sostituzione delle pagine di tipo LRU (Least Recently Used), supponendo di avere a disposizione 3 frame. Rappresentare il resident set (frame fisici contenenti pagine logiche) dopo ogni accesso alla memoria.

Tempo	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Accessi	6	5	1	6	1	2	1	5	6	3	1	1	6	5
Resident Set	6	6	6	6	6	6	**6**	5	5	**5**	1	1	1	1
		5	5	5	**5**	2	2	**2**	6	6	6	6	6	6
			1	1	1	1	1	1	**1**	3	3	3	**3**	5
Page Faults	*	*	*			*		*	*	*	*			*

C) Mostrare i page fault (accessi a pagine esterne al resident set) e calcolare il loro conteggio complessivo.

I page fault sono stati mostrati nella tabella.

Numero totale di PF: 9

D) spiegare brevemente perché la versione esatta dell'algoritmo LRU è inefficiente e quindi necessita di versioni approssimate.

Perché per determinare la vittima nel caso di un PF occorre cercare la pagina con ultimo tempo di accesso più lontano. Questo si può realizzare solo con tecniche che implicano un'operazione ad ogni accesso in memoria (quindi non solo nel caso di PF):

a) tenere traccia, per ogni pagina, del tempo di ultimo accesso (questa è l'operazione da ripetere ad ogni accesso) e poi fare una ricerca di minimo quando si cerca la vittima

oppure

b) gestire uno stack di pagine doppio linkate, spostando la pagina cui si accede in testa allo stack, ad ogni accesso (questo evita la ricerca del minimo, ma richiede un'operazione costosa ad ogni accesso

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 4KB. 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/4 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 50% (quindi 1 blocco libero ogni 2 usati), si calcoli (in funzione di NM) la dimensione massima per un intervallo contiguo di blocchi liberi, assumendo la configurazione più favorevole della bitmap.

C) Si spieghi brevemente perché, a seconda che il disco sia realizzato con tecnologia SSD oppure HDD, può essere necessario realizzare diverse politiche di scheduling per le richieste di accesso a tale disco (NON è necessario spiegare quali siano le politiche o strategie di scheduling)

A) Si calcoli il rapporto ND/NM:

|bitmap| = NM/4 blocchi = NM/4*4K*8 bit = 8K*NM bit
ogni bit della bitmap corrisponde a uno degli ND blocchi di dati
ND = 8K*NM
ND/NM = 8K

B) Supponendo che la bitmap indichi un rapporto blocchi liberi / usati del 50% (quindi 1 blocco libero ogni 2 usati), si calcoli (in funzione di NM) la dimensione massima per un intervallo contiguo di blocchi liberi, assumendo la configurazione più favorevole della bitmap.

N_{free}/N_{used} = 0,5 => N_{free} = 1/3*(N_{free}+N_{used}) = 1/3*(N_{bits}) = 0.33*8K*NM bits = 2.67K*NM bits

La situazione più favorevole è quando tutti i blocchi liberi sono contigui:
N_{good} = 2.67K*NM

C) Si spieghi brevemente perché, a seconda che il disco sia realizzato con tecnologia SSD oppure HDD, può essere necessario realizzare diverse politiche di scheduling per le richieste di accesso a tale disco (NON è necessario spiegare quali siano le politiche o strategie di scheduling)

Perché con la tecnologia HDD, basata su dischi magnetici con movimenti meccanici, è opportuno servire le richieste di accesso in modo da minimizzare il tempo di seek e la latenza rotazionale: questo in genere porta a cercare di ridurre la differenza tra gli indici di blocco (oppure di cilindro) tra richieste successive.
Al contrario, le tecnologie SSD non hanno tempi di seek e latenza rotazionale, per cui si può evitare la schedulazione (NOOP) o limitarsi a tenere conto dell'asimmetria tra letture e scritture, e/o dell'eventuale ottimizzazione per accessi a blocchi adiacenti

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

A) Si descrivano brevemente, nell'ambito di un file system, i file organizzati come:

1) Sequenza di byte
2) Struttura a record
3) Struttura complessa, ad esempio file con indice.

B) Di ognuno dei tipi di organizzazione, si dica se consente l'accesso sequenziale e/o quello diretto.

C) Si dica poi se e/o quali delle organizzazioni possono (o non possono) essere realizzate mediante file ad allocazione contigua, liste di blocchi, FAT e organizzazioni ad indici quali ad esempio gli inode (motivare la risposta).

LE RISPOSTE SONO FORMULATE IN INGLESE IN QUANTO COMUNI AL CORSO IN INGLESE DI SYSTEM AND DEVICE PROGRAMMING

A) Si descrivano brevemente, nell'ambito di un file system, i file organizzati come:

1) Sequenza di byte
2) Struttura a record
3) Struttura complessa, ad esempio file con indice.

A)

Si tratta di formati di file visti e utilizzati a livello di applicazione.
1) Tutti i file possono essere visti come sequenza di byte
2) Il record è un'astrazione di più alto livello rispetto al byte. Si possono avere record a lunghezza fissa o variabile: ad esempio i file testo con righe terminate da end-of-line (\n) sono del secondo tipo. Un file risulterà come una sequenza di record,
3) Strutture più complesse di file prevedono dati e metadati, organizzati ad esempio in header e/o sezioni. I file ad indici sono un esempio: un file di indici (oppure la parte iniziale del file) contiene una tabella con coppie (chiave, puntatore) tali da permettere ricerca efficiente: il puntatore punta in modo diretto al dato nel file di dati o nella seconda parte del file.

B) Di ognuno dei tipi di organizzazione, si dica se consente l'accesso sequenziale e/o quello diretto.

1. Il file organizzato a byte permette sia accesso sequenziale che diretto (se si sa a quale byte accedere)
2. I file con record a lunghezza fissa permettono sia accesso sequenziale che diretto (dato il numero del record e la dimensione, si calcola il byte cui accedere). I record a lunghezza variabile consentono solo accesso sequenziale
3. I file a struttura complessa (es.a indici) consentono sia accesso sequenziale che diretto, a patto che (per il diretto) si possa accedere in modo efficiente all'indice e/o all'header in cui reperire il puntatore al dato)

C) Si dica poi se e/o quali delle organizzazioni possono (o non possono) essere realizzate mediante file ad allocazione contigua, liste di blocchi, FAT e organizzazioni ad indici quali ad esempio gli inode (motivare la risposta).

Tutti i tre tipi di file sono compatibili con tutte le organizzazioni. Si tratta di formati gestiti a livelli diversi: il primo a livello applicazione, mentre gli ultimi a livello "logical file system" e "file organization module"

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)

Si consideri un processo utente su OS161

A) Spiegare brevemente la differenza tra switchframe e trapframe. Quale viene utilizzato per avviare un processo utente? Quale per gestire una chiamata di sistema? Quale per la thread_fork?

B) Lo switchframe è allocato nello stack utente o nello stack di processo a livello di kernel? (motivare)

C) Perché i dispositivi di IO sono mappati su kseg1? Sarebbe possibile mappare un dispositivo di IO in kseg0?

D) Cosa succede se un processo utente effettua una write(fd, buf, nb), dove buf è un indirizzo in kseg0? Cosa dovrebbe fare la syscall SYS_write per gestire correttamente questo caso?

A) Spiegare brevemente la differenza tra switchframe e trapframe. Quale viene utilizzato per avviare un processo utente? Quale per gestire una chiamata di sistema? Quale per la thread_fork?

Entrambe le strutture vengono utilizzate per salvare il contesto del processo (registri + altre informazioni): lo switchframe viene utilizzato per il context switch (un cambio del processo in esecuzione su una cpu), il trapframe è correlato a una trap, il che significa che siamo ancora nel contesto del processo ma in modalità kernel. La funzione runprogram utilizza un trapframe per avviare un processo utente con mips_usermode.
Una chiamata di sistema sfrutta il trapframe, poiché viene attivato come trap. La thread_fork usa uno switchframe in quanto prepara un thread pronto per essere inserito nella coda ready.

B) Lo switchframe è allocato nello stack utente o nello stack di processo a livello di kernel? (motivare)

Nello stack a livello di kernel, poiché non può essere visibile in modalità utente

C) Perché i dispositivi di IO sono mappati su kseg1? Sarebbe possibile mappare un dispositivo di IO in kseg0?

I dispositivi di IO devono essere mappati nello spazio del kernel. Sono mappati in kseg1 in quanto non è memorizzato nella cache: un dispositivo di IO non può essere letto / scritto utilizzando la cache, poiché qualsiasi operazione di lettura / scrittura deve essere eseguita sul dispositivo di IO. Per lo stesso motivo, il dispositivo non può essere mappato su kseg0, che è memorizzato nella cache.

D) Cosa succede se un processo utente effettua una write(fd, buf, nb), dove buf è un indirizzo in kseg0? Cosa dovrebbe fare la syscall SYS_write per gestire correttamente questo caso?

Il processo utente non deve poter leggere un indirizzo logico di tipo kernel. Il puntatore buf è l'origine dell'operazione di scrittura, mentre la destinazione è il file, che potrebbe essere un normale file o la console. In generale, qualsiasi indirizzo di memoria legale è corretto come sorgente, ma, dato un processo utente, un puntatore legale dovrebbe essere mappato nello spazio utente, quindi kseg0 è proibito.
Per gestire correttamente questo evento, la syscall SYS_write dovrebbe controllare che il parametro buf sia un indirizzo user, tornando un errore in caso contrario.

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 faccia riferimento al sistema operativo OS161:

A) Si descrivano brevemente le caratteristiche di semafori e locks evidenziandone le differenze.
B) Cosa sono i wait_channel? Per cosa sono utilizzati all'interno del sistema operativo?

C) Si riporta la funzione wchan_wakeone:

```
wchan_wakeone(struct wchan *wc, struct spinlock *lk) {
  struct thread *target;
  KASSERT(spinlock_do_i_hold(lk));
  target = threadlist_remhead(&wc->wc_threads);
  if (target == NULL)  { return; }
  thread_make_runnable(target, false);
}
```

Si motivi l'istruzione KASSERT(spinlock_do_i_hold(lk));
Perché si rimuove un thread dalla lista wc->wc_threads ?

A) Si descrivano brevemente le caratteristiche di semafori e lock, limitandosi al problema della gestione di una sezione critica ed evidenziandone le differenze (tra semafori e lock).

Lock e semafori sono meccanismi di sincronizzazione. Un semaforo ha uso più generale: può' essere usato per gestire la mutua esclusione da una sezione critica ma non solo, mentre il lock serve unicamente per gestire mutua esclusione da regione critica.

Un semaforo è caratterizzato da un numero intero che potrebbe gestire accessi multipli a una sezione critica;.Un lock può essere visto come un mutex, o caso particolare di semaforo in cui il numero massimo di risorse è pari ad 1 (semaforo binario).

La differenza principale tra un semaforo binario e un lock, usati per mutua esclusione, è che mentre il lock ha il concetto di ownership il semaforo no: quindi la sezione critica potrebbe essere rilasciata, per errore, usando un semaforo, da chi non la possiede. Questo non puo' accadere con un lock.

B) Cosa sono i wait_channel? Per cosa sono utilizzati all'interno del sistema operativo?

Il Wait channel: è una struttura/primitiva di sincronizzazione del kernel, che consente operazioni di wait e signal (dette wchan_sleep, wchan_broadcast, wchan_wakeone), connesse a una condizione osservata, di tipo simile alla "condition variable".

È stato introdotto in OS161 come primitiva di basso livello, associata a spinlock (mentre la condition variable è associata a lock). Il wchan può essere utilizzato solo dal kernel thread che ha acquisito lo spinlock.

C) Si riporta la funzione wchan_wakeone:

```
wchan_wakeone(struct wchan *wc, struct spinlock *lk) {
  struct thread *target;
  KASSERT(spinlock_do_i_hold(lk));
  target = threadlist_remhead(&wc->wc_threads);
  if (target == NULL)  { return; }
  thread_make_runnable(target, false);
}
```

Si motivi l'istruzione KASSERT(spinlock_do_i_hold(lk));
Perché si rimuove un thread dalla lista wc->wc_threads ?

L'istruzione KASSERT(…) non è strettamente necessaria per l'esecuzione del programma ma serve ad individuare errori del programmatore in quanto quando viene invocata la funzione in questione lo spinlock deve essere tassativamente posseduto

Per il comportamento della `wchan_wakeone` solo un thread deve essere svegliato. La lista serve in quanto più thread possono essere in attesa. Nel caso specifico si rimuove il primo della lista