Programmazione di sistema

Iniziato	venerdì, 14 maggio 2021, 20:03
Stato	Completato
Terminato	venerdì, 14 maggio 2021, 22:10
Tempo impiegato	2 ore 6 min.
In ritardo	6 min. 40 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 Byte, con indirizzi espressi in codice esadecimale) viene riportata anche l'operazione di lettura (R) / scrittura (W): R 33F5, R 2A64, W 0AD3, W 2E7E, R 08C8, W 13D1, R 094E, R 3465, W 32A0, R 1BBA, W 0CE6, R 1480, R 3294, R 2AB8. Supponiamo che gli indirizzi fisici e logici siano rappresentati su 16 bit, la dimensione della pagina sia 2 KByte e 3C02 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) Calcolare la stringa degli accessi alle pagine.

C) Simulare un algoritmo di sostituzione delle pagine di tipo PFF (Page Fault Frequency), assumendo un'unica soglia di tempo C = 2. Rappresentare il resident set (frame fisici contenenti pagine logiche) dopo ogni accesso alla memoria. Indicare esplicitamente le attivazioni dell'algoritmo (la procedura di selezione della vittima). Rappresentare anche (con un pedice o altra notazione) i bit di riferimento associati a pagine / frame.

L'algoritmo PFF (approssimato) può essere riassunto come segue:

- *L'algoritmo viene attivato ad ogni page fault, non ad ogni accesso a pagina.*
- *A seconda dell'intervallo di tempo TAU dal page fault precedente:*
  - *se TAU < C, cioè se la frequenza di page fault è maggiore di quella desiderata, aggiungere un nuovo frame al Resident Set del processo che ha fatto page faulting.*
  - *se TAU >= C, cioè se la frequenza di page fault è OK, rimuovere dal resident set del processo che ha fatto pa ge fault tutte le pagine con bit di riferimento a 0; quindi azzerare (impostare 0) il bit di riferimento di tutte le altre pagine nel Resident Set.*

NOTA: quando si valuta TAU, considerare solo gli istanti di tempo "tra" i due page fault, quindi ad esempio quando si ve rificano due page fault ai tempi 7 e 8, TAU = 0, quando due page fault sono ai tempi 12 e 15, TAU = 2). I bit di riferiment o vengono posti a 1 a ciascun accesso e azzerati come ultimo passaggio dell'algoritmo di sostituzione. Quindi, ogni volt a che i bit di riferimento vengono azzerati, questo include la pagina relativa al page fault (il riferimento corrente).

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

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

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

3c02 = 0011 1,100 0000 0010.

L'indice massimo di pagina è 00111 (binario) = 7(decimale) => Quindi lo spazio degli indirizzi ha 8 pagine.

Frammentazione interna: l'ultima pagina viene utilizzata fino al Byte all'offset 100 0000 0010 = 1K+2 = 1026

Complessivamente, 1K+3 Byte sono utilizzati nell'ultima pagina

Fr.Int. = 2K-(1K+3) = 1021 Bytes.

B) Calcolare la stringa degli accessi alle pagine.

Indirizzi logici (p, d) (pagina, scostamento).

Una pagina contiene 2 KByte, quindi lo scostamento/displacement (d) necessita di 11 bit, mentre p richiede 5 bit.

Accessi espressi in binario: R (0011 0,011 1111 0101), R (0010 1,010 0110 0100), W (000 1,010 1101 0011), …..

Stringa degli accessi (p è sufficiente, d non necessario):

6 (2*3+0), 5 (2*2+1), 1, 5, 1, 2, 1, 6, 6, 3, 1, 2, 6, 5

si noti che p, dato dai 5 bit più significativi dell'indirizzo logico, può essere rapidamente calcolato come il doppio della prima cifra esadecimale + MSB (bit più significativo) della seconda cifra esadecimale.

C) Simulare un algoritmo di sostituzione delle pagine di tipo PFF (Page Fault Frequency), assumendo un'unica soglia di tempo C = 2. Rappresentare il resident set (frame fisici contenenti pagine logiche) dopo ogni accesso alla memoria. Indicare esplicitamente le attivazioni dell'algoritmo (la procedura di selezione della vittima). Rappresentare anche (con un pedice o altra notazione) i bit di riferimento associati a pagine / frame.

Tempo	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Accessi	6	5	1	5	1	2	1	6	6	3	1	2	6	5
Resident Set	6	6	6	6	6	6_0	6_0	6	6	6_0	6_0	6_0	6	6
		5	5	5	5	5_0	5_0	5_0	$\mathbf{5_0}$	3_0	3_0	3_0	3_0	3_0
			1	1	1	1_0	1	1	1	1_0	1	1	1	1
						2_0	2_0	2_0	2_0			2	2	2
														5
Page Faults	*	*	*			*				*		*		*
Attivazione dell'algoritmo						x				x				

D) 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: 7

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)

Considerare un file system Unix-like, basato su inode, con 13 puntatori / indici (10 diretti, 1 singolo indiretto, 1 doppio indiretto e 1 triplo indiretto). I puntatori / indici hanno una dimensione di 32 bit e i blocchi del disco hanno una dimensione di 2 KB. Il file system risiede su una partizione del disco di 400 GB, che **include sia blocchi di dato che blocchi di indice.**

A) Supponendo che tutti i metadati (eccetto i blocchi indice) abbiano dimensioni trascurabili, calcolare il numero massimo di file che il file system può ospitare, utilizzando l'indicizzazione indiretta singola (N1) e l'indicizzazione indiretta tripla (N3).

B) Dato un file binario di dimensione 20490.5KB, calcolare esattamente quanti blocchi indice e blocchi di dato occupa il file.

C) Considerare lo stesso file della domanda B, su cui viene chiamata l'operazione lseek (fd, offset, SEEK_SET) per posizionare l'offset del file per la successiva operazione di lettura / scrittura (SEEK_SET significa che l'offset viene calcolato dall'inizio del file). Supponiamo che fd sia il descrittore di file associato al file (già aperto), e offset = 0x00800010. Calcolare il numero di blocco logico (numerazione relativa ai blocchi del file, numerati a partire da 0) in corrispondenza del quale viene spostata la posizione (da lseek). Se il file utilizza un'indicizzazione indiretta singola (o doppia / tripla), calcolare quale riga del blocco di indici esterno contiene l'indice del blocco di dati (o l'indice del blocco di indici più interno).

A) Supponendo che tutti i metadati (eccetto i blocchi indice) abbiano dimensioni trascurabili, calcolare il numero massimo di file che il file system può ospitare, utilizzando l'indicizzazione indiretta singola (N1) e l'indicizzazione indiretta tripla (N3).

Osservazioni generali
Un blocco indice contiene 2KB / 4B = 512 puntatori / indici.
La partizione contiene 400 GB / 2 KB = 200 M blocchi

Calcolo dei numeri massimi N1 / N3 *(per file con indicizzazione indiretta singola / tripla)*
Per calcolare il numero massimo di file, dobbiamo considerare l'occupazione minima. Dobbiamo considerare sia i blocchi di dati che i blocchi di indice.

Usiamo MIN_1 e MIN_3 per l'occupazione minima dei due tipi di file:

MIN_1 = (10 + 1) blocchi di dati + 1blocco indice
MIN_3 = (10 + 512 + 512^2 +1) blocchi di dati+
1 (singolo) + 1 + 512 (doppio) + 1 + 1 + 1 (triplo) blocchi indice
= 16 + 1024 + 512^2 = 1040 + 512^2

N1 = floor(200M / 12) = 17476266 = 16,66 M
N3 = floor(200M / (1040 + 512^2)) = 796

B) Dato un file binario di dimensione 20490.5KB, calcolare esattamente quanti blocchi indice e blocchi di dati occupa il file.

Blocchi dati: ceil(20490.5KB/2KB) = 10246
Framm. Int. = 1-0.25 blocks = 0.75 blocks = (1024+512)B = 1536B
Blocchi indice
Blocchi indice singolo: 1
Blocchi indice interni: ceil((10246-10-512)/512) = 19
Blocco indice esterno: 1 (è sufficiente il doppio indiretto)
Blocchi indice totali: 1+19+1 = 21

C) Considerare lo stesso file della domanda B, su cui viene chiamata l'operazione lseek (fd, offset, SEEK_SET) per posizionare l'offset del file per la successiva operazione di lettura / scrittura. Supponiamo che fd sia il descrittore di file associato al file (già aperto), con offset = 0x00800010. Calcolare il numero di blocco logico (relativo ai blocchi di file, numerati a partire da 0) in corrispondenza del quale viene spostata la posizione. Se il file utilizza un'indicizzazione indiretta singola (o doppia / tripla), calcolare quale riga del blocco dell'indice esterno contiene l'indice del blocco di dati (o l'indice del blocco dell'indice interno).

Indice del blocco dati: 0x00800010/2K = (8M+2)/2K = 4K = 0x1000
10 blocchi di dati sono diretti
512 blocchi di dati sono a singoli indiretti
Il blocco è al doppio livello indiretto
Indici esterni = (4K-512-10)/512 = 6

Domanda 3

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)

Rispondere alle seguenti domande sulla gestione della memoria:

A) Si Considerino gli algoritmi di scheduling del disco. Cosa ottimizzano gli algoritmi scan e C-scan e a quale tecnologia di disco si rivolgono? (motivare)

B) Che cos'è la schedulazione ***noop*** e perché viene adottata con i dischi NVM (Non-Volatile Memory)?

C) Si consideri un disco HDD con latenza di rotazione massima (tempo per un'intera rotazione del disco) 6 ms, tempo medio di seek 4 ms, un overhead del controller di 0,2 ms e velocità di trasferimento (transfer rate) 2 Gbit / s. Supponiamo che i blocchi del disco abbiano una dimensione di 8 KB, calcolare la velocità di rotazione (misurata in rpm, ovvero giri al minuto), la latenza di rotazione media (considerare mezza rotazione), il tempo di trasferimento per un blocco e il tempo di IO (complessivo) medio per trasferire 2 blocchi adiacenti (in una singola operazione I / O)

A) Si Considerino gli algoritmi di scheduling del disco. Cosa ottimizzano gli algoritmi scan e C-scan e a quale tecnologia di disco si rivolgono? (motivare)

Conosciuti anche come algoritmi dell "ascensore", riducono al minimo il tempo di seek e la relativa distanza di seek, ovvero la distanza complessiva percorsa dalla testina del disco durante la localizzazione del cilindro richiesto. La distanza è proporzionale alla somma delle differenze tra le coppie di successivi indici di blocco disco (o indici di cilindro) serviti.
Nella "scan", le richieste del disco vengono servite in entrambe le direzioni (su e giù), mentre in c-scan sono servite in una sola direzione.
Gli algoritmi hanno senso solo con HDD (dischi magnetici), dove l'overhead di ricerca è correlato alla distanza di ricerca. Gli SSD non hanno tempo di ricerca.

B) Che cos'è la schedulazione ***noop*** e perché viene adottata con i dischi NVM (Non-Volatile Memory)?

Schedulazione NOOP significa nessuna operazione o nessuna schedulazione, quindi le richieste possono essere servite con una strategia FIFO pura. Viene adottata con gli NVM, in quanto non hanno tempo di ricerca (né latenza rotazionale).

C) Si consideri un disco HDD con latenza di rotazione massima (tempo per un'intera rotazione del disco) 6 ms, tempo medio di seek 4 ms, un overhead del controller di 0,2 ms e velocità di trasferimento (transfer rate) 2 Gbit / s. Supponiamo che i blocchi del disco abbiano una dimensione di 8 KB, calcolare la velocità di rotazione (misurata in rpm, ovvero giri al minuto), la latenza di rotazione media (considerare mezza rotazione), il tempo di trasferimento per un blocco e il tempo di IO (complessivo) medio per trasferire 2 blocchi adiacenti (in una singola operazione I / O)

*Vel. Rot..: 60s * 1/6ms = 10000 rpm*
Lat. Rot. media = 6ms / 2 = 3ms
*Tempo di trasferimento (1 blocco) = 8KB / (2Gbit/s) = (8KB * 8 / 2GB) s = 32 + 10^{-6} s = 0.032 ms*

*Tempo di IO medio (2 blocchi) = Lat. Rot. media + ricerca media + Tempo di trasferimento + overhead controller = 3ms + 4ms + 2*0.032ms + 0.2ms = 7.264ms*

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 la gestione della console in OS161 e le funzioni scritte di seguito.

```
void putch_intr(struct con_softc *cs, int ch) {
  P(cs->cs_wsem);
  cs->cs_send(cs->cs_devdata,ch);
}

void con_start(void *vcs) {
  struct con_softc *cs = vcs;
  V(cs->cs_wsem);
}
```

Si risponda alle seguenti domande

A) Quale delle due funzioni viene chiamata (direttamente) dalla funzione di console **putch** (int ch) e quale viene chiamata quando riceve un interrupt dalla console seriale (motivare)?

B) Fornire la definizione del campo cs_wsem nella struct con_softc

C) Fornire un'implementazione delle due funzioni, sostituendo il semaforo cs_wsem con una condition variable (definire anche i campi necessari in struct con_softc)

A) Quale delle due funzioni viene chiamata (direttamente) dalla funzione di console **putch** (int ch) e quale viene chiamata quando riceve un interrupt dalla console seriale (motivare)?

putch_intr viene chiamata dalla funzione putch, poiché riceve il carattere da stampare, attende che la console seriale sia pronta (`P(cs->cs_wsem);`), quindi chiama la funzione send (cs_send) del driver del dispositivo della console.
Mentre con_start viene attivata dall'interrupt hardware relativo alla console seriale che significa “console pronta a ricevere un nuovo output”, da qui la chiamata a `V(cs->cs_wsem);` che segnala a putch_intr (in attesa) che un nuovo output può essere fatto.

B) Fornire la definizione del campo cs_wsem della struct con_softc

```
struct con_softc {
  ...
  struct semaphore *cs_wsem;
  ...
}
```

C) Fornire un'implementazione delle due funzioni, sostituendo il semaforo cs_wsem con una condition variable (definire anche i campi necessari in struct con_softc)

Abbiamo bisogno di un condition variable e di un lock.
Si consideri che è possibile effettuare più chiamate simultanee a putch. Solo uno alla volta verrà rilasciata dal semaforo. Come alternativa, utilizziamo un flag “ready”, un lock per mutua esclusione e la condition variabile per fare wait / signal.

```
struct con_softc {
  ...
  struct lock *cs_wlk;
  int wready;
  struct lock *cs_wcv;
  ...
}

void putch_intr(struct con_softc *cs, int ch) {
  /* attende che la console sia pronta */
  lock_acquire(cs->cs_wlk;
  while (!cs->wready) {
    cs_wait(cs->cs_wcv, cs->cs_wlk);
  }
  cs->wready = 0;
  lock_release(cs->cs_wlk;
  /* rilascia il lock, wready verrà settato da con_start al termine
dell'output*/
  cs->cs_send(cs->cs_devdata,ch);
}

void con_start(void *vcs) {
  struct con_softc *cs = vcs;
  /* imposta il flag ready e riattiva un thread eventualmente in attesa */
  lock_acquire(cs->cs_wlk;
  cs->wready = 1;
  cs_signal(cs->cs_wcv,cs->wlk);
  lock_release(cs->cs_wlk;
}
```

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 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) Un processo utente può eseguire una write(fd, buf, nb), dove buf è un indirizzo in kseg0?

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) Un processo utente può eseguire una write(fd, buf, nb), dove buf è un indirizzo in kseg0?

NO, per mancanza di privilegi. 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.