Programmazione di sistema

Iniziato	martedì, 18 ottobre 2022, 15:02
Stato	Completato
Terminato	martedì, 18 ottobre 2022, 15:02
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)

A) Si considerino le affermazioni che seguono, a proposito di possibili vantaggi e svantaggi di una inverted page table (IPT), rispetto a una tabella delle pagine standard (PT). Si dica di ognuna se sia vera o falsa, motivando la risposta

Vantaggi: L'IPT permette di risparmiare di memoria:

1. Si risparmia sempre memoria
2. Dipende dalle dimensioni della RAM, dal numero di processi e dal loro spazio di indirizzamento virtuale
3. Si risparmia sempre memoria quando lo spazio di indirizzamento di un processo è maggiore della dimensione della RAM
4. Si può risparmiare anche in casi in cui lo spazio di indirizzamento di ogni processo è inferiore alla dimensione della RAM

Svantaggi: L'IPT è lenta, perché non garantisce accesso diretto ma occorre una ricerca

1. La chiave di ricerca è il frame
2. La chiave di ricerca è la pagina
3. La chiave di ricerca è la coppia (pid,frame)
4. La chiave di ricerca è la coppia (pid,pagina)
5. Per migliorare le prestazioni si sostituisce la IPT con una tabella di HASH
6. Per migliorare le prestazioni si aggiunge alla IPT una tabella di HASH

B) Sia dato un processo avente spazio di indirizzamento virtuale di 48 GB, in un sistema dotato di 16GB di RAM, con architettura a 64 bit (in cui si indirizza il Byte) e gestione della memoria paginata (pagine/frame da 4KB) per i soli processi user, quindi non per il kernel. Si supponga che 4GB di RAM siano allocati in modo statico al kernel. Si vogliono confrontare una soluzione basata su tabella delle pagine standard (una tabella per ogni processo) e una basata su IPT. Si calcolino

B1) Le dimensioni della PT (a un solo livello) per il processo e della IPT. Si ipotizzi che il `pid` di un processo possa essere rappresentato su 12 bit. Si utilizzino 28 bit per gli indici di pagina e/o di frame (nella PT o nella IPT) e *si tenga conto che, per allineamento, una cella di IPT o PT può solo essere di 32 o 64 bit.*

B2) Si dica infine, utilizzando la IPT proposta (12 bit di pid, 28 bit bit per un indice di pagina/frame), quale è la dimensione massima possibile per lo spazio di indirizzamento virtuale di un processo.

A) Si considerino le affermazioni che seguono, a proposito di possibili vantaggi e svantaggi di una inverted page table (IPT), rispetto a una tabella delle pagine standard (PT). Si dica di ognuna se sia vera o falsa, motivando la risposta

Vantaggi: *L'IPT permette di risparmiare di memoria:*

1. ***NO****. Non si risparmia nel caso di pochi processi con spazio di indirizzamento piccolo*
2. ***SI,*** *per motivi simili alla risposta 1*
3. ***Si****, purché la dimensione venga misurata in termini di righe/celle nella tabella. Infatti il numero di righe nella PT è maggiore del numero di righe nella IPT. Si potrebbe infatti considerare l'eccezione in cui (siccome c'è anche il pid) un entry della IPT potrebbe occupare più memoria di uno nella PT*
4. ***SI.*** *Perché la IPT è unica per tutti i processi.*

Svantaggi: *L'IPT è lenta, perché non garantisce accesso diretto ma occorre una ricerca*

1. ***NO****. Il frame corrisponde all'indice nella tabella*
2. ***SI,*** *se si intende che la chiave di ricerca "include" l'indice di pagina.* ***NO****, se di considera che non basta (c'é anche il pid)*
3. ***NO****. Il frame corrisponde all'indice dell'entry nella tabella*
4. ***SI.*** *Il contenuto di una cella della IPT è proprio questo*
5. ***NO.*** *Non basta sostituire. O meglio, se si sostituisce con una tabella di HASH si tratta di un'altra soluzione*
6. ***SI.*** *E' proprio quello che si fa di solito per migliorare le prestazioni della IPT*

B1) Dimensioni di PT e IPT

Page Table standard:

N pagine = 48GB/4KB = 12M (corrisponde al numero di celle/righe/entry nella PT)

Una cella della PT occupa 4B (deve contenere 28 bit, quindi per allineamento si arriva a 32 bit)

|PT| = 12M*4B = **48MB**

IPT

La tabella, unica per tutti i processi, ha dimensione fissa, in quanto contiene un indice di pagina per ogni frame in RAM. Quindi non dipende dal processo. E' sufficiente rappresentare nella IPT 12GB = 16GB – 4GB (si esclude la RAM allocata al kernel).

Ogni riga della IPT contiene almeno 12bit (pid) + 28 bit (pagina) = 40 bit. Quindi per allineamento si arriva a 64bit = 8B.

N frame = 12GB/4KB = 3M

|IPT| = 3M*8B = **24MB**

B2) Spazio di indirizzamento

NOTA
In questo caso non si possono utilizzare completamente i 64 bit di indirizzo, in quanto per gli indici di pagina c'è un limite di 28 bit.

Il numero massimo di pagine virtuali di un processo è quindi limitato dalla dimensione degli indici di pagina (28 bit): tale numero è quindi 2^28 = 1G/4 = 256M.
Siccome ogni pagina ha dimensione 4KB, lo spazio di indirizzamento virtuale ha dimensione massima
|address space| = (1G/4)*4KB = **1TB**.

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

A) Sia dato un disco organizzato con struttura RAID. Sono dati il tempo medio tra guasti (MTTF: Mean Time To Failure), il tempo medio per riparazione (MTTR: Mean Time To Repair), e il tempo medio tra perdite di dato (MTTDL: Mean Time To Data Loss). E' possibile che (motivare le risposte)

A1) MTTR > MTTF?

A2) MTTDL>10*MTTF?

B) Si consideri una struttura RAID con 2 dischi in configurazione *"mirrored"*. Se ognuno dei dischi può guastarsi in modo indipendente dall'altro, con MTTF = 20000 ore, quanto deve essere MTTR per garantire MTTDL > 100*MTTF)?

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

A) Sia dato un disco organizzato con struttura RAID. Sono dati il tempo medio tra guasti (MTTF: Mean Time To Failure), il tempo medio per riparazione (MTTR: Mean Time To Repair), e il tempo medio tra perdite di dato (MTTDL: Mean Time To Data Loss). E' possibile che (motivare le risposte)

A1) MTTR > MTTF?

No, non è possibile, o meglio, non è realistico. MTTF è dell'ordine di anni in quanto si tratta del tempo che intercorre tra due guasti, mentre MTTR è di ore o giorni: si tratta del tempo necessario per riparare (o sostituire e poi ripristinare i dati) un disco guasto

A2) MTTDL>10*MTTF?

SI. E' possibile, ma dipende dal tempo MTTR. L'obiettivo dei dischi RAID è proprio quello di rendere MTTDL maggiore, o molto maggiore, di MTTF

B) Si consideri una struttura RAID con 2 dischi in configurazione *"mirrored"*. Se ognuno dei dischi può guastarsi in modo indipendente dall'altro, con MTTF = 20000 ore, quanto deve essere MTTR per garantire MTTDL > 100*MTTF)?

(per una spiegazione dettagliata si vedano i lucidi Silberschatz modificati: ***ch11 (con aggiunte su dischi RAID)****)*

La formula a usare è: MTTDL = MTTF^2/(2*MTTR)
il vincolo richiesto è: MTTDL > 100*MTTF

(sostituendo)

MTTF^2/(2*MTTR) > 100*MTTF
MTTF > 200*MTTR
da cui
MTTR < MTTF/200 = 20000/200 = 100 (è sufficiente effettuare una riparazione in meno di 100 ore, cioè circa 4 giorni)

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 consideri il problema della gestione di una richiesta di IO a blocchi realizzato mediante DMA.

A) Che cosa si intende, in questo contesto, con il termine "*cycle stealing*"?

B) Perché il trasferimento in DMA è vantaggioso rispetto all'IO programmato?

C) Immaginando di dover trasferire 10MB di dati da disco a memoria RAM, quanti Byte transitano sul bus dati della RAM nei due casi (trasferimento in DMA e IO programmato)? (motivare).

D) Qualora si volesse usare per l'IO, mediante DMA, un buffer in memoria Kernel, invece di usare direttamente la sorgente/destinazione in memoria user, il DMA controller farebbe più o meno trasferimenti, oppure lo stesso numero, rispetto alla versione senza buffer? (motivare)

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

Si consideri il problema della gestione di una richiesta di IO a blocchi realizzato mediante DMA.

A) Che cosa si intende, in questo contesto, con il termine "*cycle stealing*"?

Il "cycle stealing" è la sottrazione (con attesa per la CPU) di cicli di BUS alla CPU, mentre il DMA ha il controllo dei BUS di accesso alla RAM. Viene gestito completamente dall'HW, quindi l'effetto sulla CPU è solo quello di vedere istruzioni di lettura/scrittura che durano più cicli di clock

B) Perché il trasferimento in DMA è vantaggioso rispetto all'IO programmato?

Per due motivi principali

- Perché i trasferimenti mediante DMA fanno passare i dati "direttamente" tra IO e RAM (senza passare dalla CPU, con un numero doppio di operazioni
- Perché mentre si trasferiscono dati in DMA la CPU può fare altro (aumentando il grado di multiprogrammazione)

C) Immaginando di dover trasferire 10MB di dati da disco a memoria RAM, quanti Byte transitano sul bus dati della RAM nei due casi (trasferimento in DMA e IO programmato)? (motivare).

Transitano 10MB in DMA e 20MB (i 10MB passano due volte, in quanto debbono passare dai registri della CPU) nel caso di IO programmato.

D) Qualora si volesse usare per l'IO, mediante DMA, un buffer in memoria Kernel, invece di usare direttamente la sorgente/destinazione in memoria user, il DMA controller farebbe più o meno trasferimenti, oppure lo stesso numero, rispetto alla versione senza buffer? (motivare)

Lo stesso numero. Cambia solo la sorgente (o destinazione in RAM) ma il trasferimento in DMA è simile. Poi (con il buffer kernel) occorrerà un'ulteriore copia tra memoria kernel e user, ma questo NON è in DMA.

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. Si consideri la funzione as_copy, avente come obiettivo quello di “duplicare” un address space, cioè creare un nuovo address space, a partire da uno esistente, di cui si fa una copia. La as_copy potrebbe essere usata, ad esempio, per realizzare la system call fork, che genera un processo child a partire da un processo (parent) esistente. Si propone una possibile implementazione della as_copy:

```
int as_copy(struct addrspace *old, struct addrspace **ret) {
  struct addrspace *new;

  dumbvm_can_sleep();
  new = as_create();

  new->as_vbase1 = old->as_vbase1;
  new->as_npages1 = old->as_npages1;
  new->as_vbase2 = old->as_vbase2;
  new->as_npages2 = old->as_npages2;
  new->as_pbase1 = old->as_pbase1;
  new->as_pbase2 = old->as_pbase2;
  new->as_stackpbase = old->as_stackpbase;

   memmove((void *)PADDR_TO_KVADDR(new->as_pbase1), (const void *)PADDR_TO_KV
ADDR(old->as_pbase1), old->as_npages1*PAGE_SIZE);
   memmove((void *)PADDR_TO_KVADDR(new->as_pbase2), (const void *)PADDR_TO_KV
ADDR(old->as_pbase2), old->as_npages2*PAGE_SIZE);
   memmove((void *)PADDR_TO_KVADDR(new->as_stackpbase), (const void *)PADDR_TO
_KVADDR(old->as_stackpbase), DUMBVM_STACKPAGES*PAGE_SIZE);

  *ret = new;
  return 0;
}
```

per la quale si chiede di rispondere alle seguenti domande:

A) Perché il parametro old ha un solo asterisco mentre ret ne ha due?

B) La funzione as_create alloca RAM per i due segmenti (testo e dati) e per lo stack del nuovo address space?

C) La funzione **contiene volutamente degli errori**. Si tratta di errori semantici, legati all’obiettivo della funzione, quindi non di errori sintattici: si individuino tali errori e si spieghi perché lo sono.

D) La funzione memcopy viene qui utilizzata per copiare il contenuto di una parte di RAM fisica in un’altra parte di RAM fisica: che cosa realizza PADDR_TO_KVADDR e perché viene utilizzata in questo contesto, invece di chiamare semplicemente, ad esempio:

memmove((void *)(new->as_pbase1),(const void *)(old->as_pbase1), old->as_npages1*PAGE_SIZE);

Risposte alle seguenti domande:

A) Perché il parametro old ha un solo asterisco mentre ret ne ha due?

perché il parametro old (puntatore all'address space esistente) è un dato in ingresso alla funzione, mentre ret serve per ritornare "by pointer" il risultato, cioè il nuovo address space

B) La funzione as_create alloca RAM per i due segmenti (testo e dati) e per lo stack del nuovo address space?

No. La as_create alloca semplicemente una struct addrspace in memoria kernel (usando internamente kmalloc) ma non alloca segmenti in RAM: non riceve parametri adeguati/sufficienti per tale scopo: servirebbero le dimensioni dei segmenti. L'allocazione dei segmenti può essere invece realizzata tramite as_prepare_load.

C) La funzione **contiene volutamente degli errori**. Si tratta di errori semantici, legati all'obiettivo della funzione, quindi non di errori sintattici: si individuino tali errori e si spieghi perché lo sono.

Le istruzioni

```
new->as_pbase1 = old->as_pbase1;
new->as_pbase2 = old->as_pbase2;
new->as_stackpbase = old->as_stackpbase;
```

sono incompatibili con le `memcopy` che seguono, in quanto la destinazione della memcopy coincide con la sorgente. La duplicazione dell'addrspace andrebbe realizzata mediante allocazione (in RAM isica) di due nuovi segmenti e di un nuovo stack: invece delle assegnazioni riportate sopra occorrerebbe quindi chiamare la `as_prepare_load`.

RISPOSTA ALTERNATIVA: *Se invece si volesse realizzare un duplicato parziale, in grado di condividere all'inizio tutta la RAM del processo originale (quindi seguire la politica **copy-on-write**) allora sarebbero corrette le assegnazioni riportate, ma occorrerebbe eliminare le `memcopy`, che diventerebbero inutili.*

D) La funzione memcopy viene qui utilizzata per copiare il contenuto di una parte di RAM fisica in un'altra parte di RAM fisica: che cosa realizza PADDR_TO_KVADDR e perché viene utilizzata in questo contesto, invece di chiamare semplicemente, ad esempio:

```
memmove((void *)(new->as_pbase1),(const void *)(old->as_pbase1), old->as_npages1*PAGE_SIZE);
```

La PADDR_TO_KVADDR (come si intuisce dal nome) converte da indirizzo fisico a indirizzo logico di kernel, il che si ottiene sommando 0x80000000 (che equivale a porre semplicemente a 1 il bit più significativo).

Nessuna istruzione software può utilizzare ***direttamente*** indirizzi “fisici”, in quanto le istruzioni sono eseguite dalla CPU, nella quale cui la traduzione logico-fisica viene fatta dalla MMU. Quindi, una copia da memoria fisica a memoria fisica, mediante istruzioni del kernel, viene qui fatta generando gli indirizzi logici kernel che corrispondano agli indirizzi fisici su cui si intende lavorare

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)

In un sistema OS161, si vuole realizzare il supporto per la system call `SYS__exit` (che corrisponde alla `void exit (int status);`)

NON SI INTENDE ANCORA SUPPORTARE LA WAITPID, ma unicamente il salvataggio dello stato (`status`), la terminazione del thread e il rilascio dell'address space.

A) Si supponga che la funzione `syscall()`, in corrispondenza al valore `SYS__exit` in `callno` chiami la funzione `my_sys_exit()`, si definisca il prototipo di tale funzione e se ne scriva la chiamata (con parametri attuali) in `syscall()`.

/* METTERE TIPO DI RITORNO */ `my_sys_exit(`**/* METTERE PARAMETRI QUI */**`);`

```
void syscall(struct trapframe *tf) {
  /* ... */
  callno = tf->tf_v0;
  /* ... */
  switch (callno) {
    /* ... */
  case SYS___exit:
    err = my_sys_exit(/* METTERE PARAMETRI QUI */);
    break;
  }
}
```

B) Si osservi il seguente frammento di codice (incompleto) della `my_sys_exit()`. Si dica da dove può o deve provenire il valore status, e a che variabile o campo di struct va assegnato (dire quindi cosa va sostituito ai …). Si identifichino e si commentino gli errori o problemi contenuti nelle successive istruzioni:

```
… my_sys_exit(…) {
   ... = status;
   thread_exit(curthread);
   proc_destroy(curproc);
   as_destroy(curproc->p_addrspace);
}
```

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)

A) Si supponga che la funzione `syscall()`, in corrispondenza al valore `SYS__exit` in `callno` chiami la funzione `my_sys_exit()`, si definisca il prototipo di tale funzione e se ne scriva la chiamata (con parametri attuali) in `syscall()`.

```
// volendo si possono usare altri tipi assimilabili a int oppure unsigned int
int my_sys_exit(int status);
case SYS___exit:
     err = my_sys_exit((int) tf->tf_a0); // si passa il primo parametro, lo stato
    break;
```

B) Si osservi il seguente frammento di codice (incompleto) della `my_sys_exit()`. Si dica da dove può o deve provenire il valore status, e a che variabile o campo di struct va assegnato (dire quindi cosa va sostituito ai …). Si identifichino e si commentino gli errori o problemi contenuti nelle successive istruzioni:

```
… my_sys_exit(…) {
  ... = status;
  thread_exit(curthread);
  proc_destroy(curproc);
  as_destroy(curproc->p_addrspace);
}
```

Il valore di status deve essere ricevuto come parametro e va assegnato a un opportuno campo della struct proc: ad esempio `curproc->status = status;`

ERRORI:

La thread_exit deve essere l'ultima istruzione, in quanto termina l'esecuzione del thread, le succesive due istruzioni non sono quindi eseguite.
La proc_destroy non puo' essere esequita se si vuole mantenere la struct proc contenente il valore salvato di status (poi occorrerà implementare la waitpid)
La as_destroy non potrebbe comunque essere effettuata dopo la proc_destroy, e va chiamata prima di thread_exit