PDS – Os Internals – Esame del 13/1/2025

Cognome e Nome:

Matricola:

1) memoria virtuale

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 (eventualmente gerarchica) PT. Si dica di ognuno 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 la coppia (pid,frame)
2. La chiave di ricerca è la coppia (pid,pagina)
3. Per migliorare le prestazioni si sostituisce la IPT con una tabella di HASH
4. Per migliorare le prestazioni si aggiunge alla IPT una tabella di HASH

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). Si supponga che 4GB di RAM sia allocato 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

A) le dimensioni della tabella delle pagine (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.

B) 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.

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, in quanto una PT standard ha dimensione proporzionale all'address space (nel fare quest'affermazione si assume che le entry nelle due tabelle abbiano dimensione uguale).
4. SI. Perche' l'IPT e' unica per tutti i processi, mentre la somma di piú PT per piú processi potrebbe avere dimensione superiore.

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

1. NO. Il frame corrisponde all'indice
2. SI. Il contenuto di una cella della IPT è proprio questo
3. NO. Non basta sostituire. O meglio, se si sostituisce con una tabella di HASH si tratta di un'altra soluzione
4. SI. E' proprio quello che si fa di solito per migliorare le prestazioni della IPT

Per i calcoli delle dimensioni, si trascurano eventuali bit di validità/modifica.

A) DIMENSIONI di PT e IPT

Page Table standard:

N pagine = 48GB/4KB = 12M (corrisponde al numero di celle/righe nella PT)
Una cella della PT occupa 4B (deve contenere 28 bit, quindi per allineamento si arriva a 32 bit)
|Page Table| = 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. E' sufficiente rappresentare 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

B) Spazio di indirizzamento virtuale

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 1G/4 = 256M. Siccome ogni pagina ha dimensione 4KB, lo spazio di indirizzamento virtuale ha dimensione massima (1G/4)*4KB = 1TB.

2) Organizzazione di un disco

Sia dato un disco organizzato con blocchi fisici e logici di dimensione 8KB. 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 33,33% (quindi 1 blocco libero ogni 3 usati), si calcoli (in funzione di NM) la dimensione massima per un intervallo contiguo di blocchi liberi, assumendo la configurazione più favorevole della bitmap (favorevole significa in grado di ottenere partizioni libere più grandi). Si dia la stessa risposta anche assumendo la configurazione della bitmap meno favorevole.
C) Si supponga che un file control block (FCB) abbia dimensione 256B e NM/4 blocchi di metadati siano riservati agli FCB, per un massimo di 16K file. Si calcolino ND, NM e NB. Si esprima anche la dimensione della bitmap e della partizione A, espressa in Byte.

Risposte

A) Si calcoli il rapporto ND/NM:

|bitmap| = NM/4 blocchi = NM/4*8K*8 bit = 16K*NM bit
ogni bit della bitmap corrisponde a uno degli ND blocchi di dati
ND = 16K*NM
ND/NM = 16K

B) Supponendo che la bitmap indichi un rapporto blocchi liberi / usati del 33,33% (quindi 1 blocco libero ogni 3 usati), si calcoli (in funzione di NM) la dimensione massima per un intervallo contiguo di blocchi liberi, assumendo la configurazione più favorevole della bitmap (favorevole significa in grado di ottenere partizioni libere più grandi). Si dia la stessa risposta anche assumendo la configurazione della bitmap meno favorevole.

$N_{free}/N_{used} = 1/3 \Rightarrow N_{free} = 0.25*(N_{free}+N_{used}) = 0.25*(N_{bits}) = 0.25*16K*NM$ bits = 4K*NM bits

La situazione più favorevole è quando tutti i blocchi liberi sono contigui:
$N_{good} = 4K*NM$

La situazione meno favorevole è quando tutti i blocchi liberi non hanno altri blocchi liberi adiacenti:
$N_{bad} = 1$

C) Si supponga che un file control block (FCB) abbia dimensione 256B e NM/4 blocchi di metadati siano riservati agli FCB, per un massimo di 16K file. Si calcolino ND, NM e NB. Si esprima anche la dimensione della bitmap e della partizione A, espressa in Byte.

Il numero massimo di file è 16K
Un blocco può contenere 8KB/256B = 32 FCB
Il numero massimo di FCBs is 32*NM/4 = 8*NM
Poiché i file corrispondono agli FCB:
8*NM = 16K
NM = 2K
ND = 16K*NM = 32M
NB = ND + NM = 32M + 2K
|A| = (32M+2K)*8KB = 256GB + 16MB
|bitmap| = NM/4*8KB = 4MB

3) link/load, IPT e TLB

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 e si spieghi perchè la soluzione di IPT + tabella di HASH e' 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? Il "valid" bit presente in un entry della TLB e' una semplice copia del "valid" bit presente nella Page Table (motivare) ?

Risposte

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 che i 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. A questo si puó aggiungere che la IPT+HASH è globale, mentre le PT basate su HASH sono tipicamente per singoli processi.

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? Il "valid" bit presente in un entry della TLB e' una semplice copia del "valid" bit presente nella Page Table (motivare) ?

> Esistono entrambi i tipi di TLB: quelle contenenti entry per più processi (in queste c'è un campo aggiuntivo detto ASID: Address Space IDentifier) e quelle contenente solo entry per il processo attualmente attivo. Queste ultime necessitano di funzionalità di azzeramento/reset in corrispondenza a ciascun context switch. Il valid vit ha significati diversi nella TLB e nella PT: nella TLB significa che l'entry è libero/disponibile, nella PT significa che la pagina logica non è associata a un frame.

4) Os161 - project

É dato un Sistema OS161. Si supponga di aver aggiunto le istruzioni seguenti a `kern/conf/conf.kern`

```
defoption project
optfile project syscall/project.c
```

e di aver creato il file `PROJECT` in `kern/conf`, copiato dal file `DUMBVM`.

A) Si dica se le azioni descritte nel seguito, piú l'esecuzione (in `kern/conf`) di `./config PROJECT`, sono sufficienti affinché il file opzionale `syscall/project.c` sia compilato quando si eseguono (in `kern/compile/PROJECT`) i comandi

```
bmake depend
bmake
```

B) Quale file, tra `project.h` e `opt-project.h` viene generato automaticamente dal comando `./config PROJECT`? Il file viene sempre generato, oppure solo se l'istruzione seguente compare nel file `PROJECT`?

```
options project
```

C) Cosa contiene il file (quello generato automaticamente, citato nella domanda precedente)?

D) Si supponga di inserire in `main.c` l'istruzione

```
project_init();
```

Considerando che la funzione `project_init()` e' implementata nel file `syscall/project.c`, come si puo' fare in modo che l'istruzione sia considerate e compilate solo nelle versioni del kernel in cui l'opzione `project` e' abilitata?

Risposte

A)

NO. Non sono sufficienti. Le istruzioni definiscono l'opzione e la dipendenza del file `syscall/project.c` dall'opzione. Ma l'opzione va abilitata in modo esplicito, nel file `PROJECT`. Diversamente, il file `project.c` non sará compilato, né considerato per la generazione delle dipendenze.

B)

`opt-project.h`. Il file viene generato anche se `PROJECT` non contiene `options project`. Affinchè il file .h venga generato, è sufficiente che `conf.kern` contenga la definizione dell'opzione `project`. Un file `project.h` potrá, se necessario o appropriato, essere creato "manualmente".

C)

`opt-project.h`, oltre alla protezione da inclusione multipla, contiene solo una direttiva al pre-compilatore:
`#define OPT_PROJECT 1`
o
`#define OPT_PROJECT 0`
che dipende dal fatto che il file `PROJECT` contenga o meno l'istruzione `options project`

D)

Va usata la compilazione condizionale, mediante la macro `OPT_PROJECT`, definita in `opt-project.h`.
L'intestazione del file `main.c`, (o di un .h incluso da questo) deve contenere

```
#include "opt-project.h"
```

e andranno condizionate le istruzioni (in `main.c`) come segue

```
#if OPT_PROJECT
project_init();
#endif
```

5) OS161 – user process

Si consideri un processo user OS161 (a ogni domanda si dia una risposta e una motivazione).

A) Quale, tra trapframe e switchframe, viene usato per gestire una system call?

B) Un trapframe viene allocato nello user stack o nello stack a livello kernel di un processo?

C) Perché i dispositivi di IO sono mappati in kseg1? É possibile che un dispositivo di IO sia mappato in kseg0?

D) Cosa succeed se un processo user chiama read(fd, buf, nb), dove buf é un indirizzo in kseg0? Cosa dovrebbe fare la system call SYS_read, al fine di gestire correttamente questo caso?

Risposte

A)

Viene utilizzato un trapframe, perché una chiamata di sistema viene gestita tramite un protocollo trap/interrupt (una chiamata di sistema è una sorta di interruzione software)

B)

Nello stack del kernel, poiché trapframe è una struttura dati del kernel (sebbene utilizzata per gestire il processo utente). Sarebbe un problema lasciarlo visibile in modalità utente, poiché un programma utente potrebbe corromperlo.

C)

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

D)

Questo è chiaramente un errore di programmazione, poiché la destinazione di una lettura (un programma utente) è un indirizzo del kernel. La chiamata di sistema SYS_read dovrebbe controllare e gestire correttamente l'errore ed evitare di eseguire l'operazione di lettura.