

Multitasking in ia-32 – Spiegazioni varie...

Pubblicato da **Ritz** il 28/01/2001

Livello **avanzato**

Introduzione

Salve a tutti. Senza soffermarmi troppo nell'introduzione, vi dico subito che in questo tute vedremo come nei processori della famiglia i386 viene gestito il cosiddetto "multitasking" di cui tanto si sente parlare.

Iniziamo

Per affrontare questo tutorial vi basta una classica guida sul protected mode (ne trovate a centinaia in inglese, se ne volete una in italiano prendete la mia :P)

Nelle architetture ia32, un task e' una "unita' di lavoro che il processore puo' preparare, eseguire e sospendere". Esso puo' essere utilizzato in svariate situazioni: come processo a parte, come servizio di sistema, come interrupt handler, e cosi' via.

Il protected mode (detto anche pm, o pmode) offre al programmatore (a livello di sistema operativo) la possibilita' di eseguire piu' task "contemporaneamente" tramite alcune strutture che trattero' in seguito, non prima di aver spiegato il significato delle virgolette che ho usato.

Infatti, il multitasking inteso come capacita' di eseguire contemporaneamente piu' istruzioni a livello processore, cioe' un multitasking reale, e' impossibile in architetture monoprocesso, dove cioe' c'e' solo 1 processore che si occupa di eseguire le istruzioni macchina una per volta (ovviamente).

Il multitasking come viene solitamente inteso, pero', consiste in realta' nel dividere tra i vari processi la possibilita' di utilizzare il processore stesso.

Nella pratica cio' significa che mentre stiamo eseguendo un prg qualsiasi (in un so opportuno ovviamente =)) possiamo decidere di farne partire un altro, consci del fatto che sara' il SO stesso a decidere i tempi di esecuzione che i due dovranno osservare (attraverso appositi algoritmi di scheduling).

Fatta questa piccola nota, procediamo.

Struttura di un Task-State Segment

^^

Cio' che identifica un task e' il suo Task-State Segment (TSS). Il TSS infatti e' una struttura che specifica lo spazio di esecuzione del task stesso, i suoi code/data/stack segment, i meccanismi per il task linking, e cosi' via. Ecco come esso si presenta:

```
typedef struct
{
    /* i80386 TSS */

    unsigned BackLink;
    unsigned ESP0;
    unsigned SS0;
    unsigned ESP1;
    unsigned SS1;
    unsigned ESP2;
    unsigned SS2;
    PGTBL **PGDIR;
    unsigned EIP;
    unsigned EFLAGS;
    unsigned EAX;
    unsigned ECX;
    unsigned EDX;
    unsigned EBX;
    unsigned ESP;
    unsigned EBP;
    unsigned ESI;
    unsigned EDI;
    unsigned ES;
    unsigned CS;
    unsigned SS;
    unsigned DS;
    unsigned FS;
    unsigned GS;
    unsigned LDT;
    unsigned IO_bitmap_offset;
```

```

/* End of i80386 TSS. */
} TSS;

```

Insomma si tratta di un array di 26 elementi da 32bit ciascuno. Da notare che per il caso di elementi a 16bit (BackLink, SS0, SS1, SS2, ES, CS, DS, FS, GS) il corrispondente spazio all'interno del TSS viene riempito solo per i 16 bit + bassi della "cella" mentre i rimanenti 16 bit + alti sono riservati e devono essere settati a 0. Fa eccezione l'ultimo elemento, che occupa quelli + alti lasciando riservati i rimanenti (a parte il primo bit dell'ultimo elemento, che costituisce il flag T). Ecco un descrizione dei vari elementi nell'ordine in cui li troviamo:

- ****PGDIR**: un puntatore (indirizzo fisico) della page dir usata dal task.
- **EIP**: stato dell'Instruction Pointer prima del task switch.
- **EFLAGS**: stato degli EFAG prima dello switch.
- **EAX --> EDI**: stato dei general purpose register prima dello switch.
- **ES --> DS**: vi lascio immaginare =).
- **LDT**: contiene il selettore per l'LDT del task.
- **T**: questo flag (praticamente il bit 801, ovvero tra l'LDT selector e l'IO_bitmap_offset) se settato causa una task-switch exception se avviene uno switch verso il task identificato da questo tss.
- **IO_bitmap_offset**: contiene un offset a 16bit che punta, a partire dalla base del TSS, all'IO permission bit map e all'interrupt redirection bit map, che cmq non ci interessano molto.

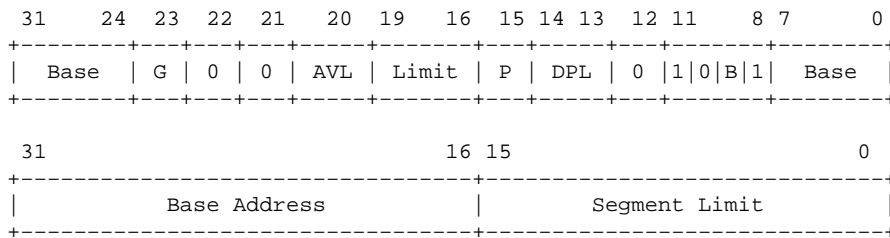
Task Switching
 ^^^^^^^^^^^^^^^

Per cambiare task (task switching) possiamo agire in 4 modi diversi:

- jmp/call a un TSS descriptor nella gtd
- jmp/call a un task gate descriptor nella gdt/ldt
- un vettore di int/excep che punta al task gate di cui sopra
- esecuzione di un iret (in tal caso lo switching avviene, al contrario, verso il chiamante iniziale)

Metodo 1

Se decidiamo di fare in questo modo, allora dobbiamo prima installare un tss descriptor della gdt (_non_ possiamo utilizzare ldt o idt) ovviamente da 64 bit. Ecco come esso si presenta (disegno assolutamente non in scala ;):



Similmente agli altri segment descriptor, i 3 campi "Base Address" offrono un indirizzo logico del primo byte del segmento (il TSS nel nostro caso). I 2 campi "Segment Limit" offrono la grandezza del TSS stesso e seguono la regola del flag G di granularita'. Il DPL indica il livello di privilegio del segmento. Il flag B (busy) indica se il task e' occupato: poiche' i task non sono ricorsivi, un task non puo' chiamare se stesso (da notare che i bit 8 --> 11 corrispondono al campo "Type" di un comune segment descriptor, solo che in questo caso viene lasciato solo il flag B, data la particolarita' del segmento ;)) e il flag B serve proprio per controllare che questo non avvenga (in caso contrario si potrebbero avere perdite di info riguardanti il task). I flag P e AVL indicano rispettivamente se il segmento e' presente in memoria e se e' utilizzabile dal so. Per maggiori info sui segment descriptor vi rimando al mio tute sul memory management nel pmode.

Metodo 2

Tramite un task gate descriptor possiamo saltare indirettamente a un task. Esso puo' essere installato nella gdt, ldt o idt. Eccone di

| |
+-----+

Ma una volta che tutto e' installato correttamente, come avviene lo switch vero e proprio tra 2 task? Ecco cosa fa il processore mentre cambia per cambiare task (paso dal solito tute sul pmode):

- 1- Ricava il TSS segment selector (dal task gate o dall'operando dell'istro in caso di switch diretto).
- 2- Controlla che la CPL del task vecchio e l'RPL del segment selector siano minori o uguali al DPL del TSS descriptor o del task gate. Le exception e int (a parte gli int n) possono permettere di switchare task senza questi check.
- 3- Controlla che il TSS descriptor sia segnato come presente e abbia un limite maggiore e uguale a 0x67.
- 4- Controlla se il nuovo task e' disponibile o occupato.
- 5- Controlla che i TSS vecchi e nuovi e i vari descrittori siano mappati in memoria.
- 6- Se e' utilizzati un JMP o IRET, viene cancellato il flag B, se si e' utilizzata una CALL, exception e int il flag B viene lasciato attivo.
- 7- Se si e' usato un IRET per lo switch viene cancellato il flag NT in un'immagine temporanea degli EFLAGS che viene salvata; in caso contrario non viene modificata nulla.
- 8- Salva il corrente (vecchio) task nel suo TSS. **ORA AVVIENE LO SWITCH VERO E PROPRIO**
- 9- Se lo switch e' stato eseguito tramite una CALL, except o int il setta il flag NT in un'immagine nel nuovo TSS. Se e' stato eseguito un IRET, il processore rimette a posto il flag NT. Se e' stato usato un JMP viene lasciato tutto cosi' com'e'.
- 10- Se il task switch e' stato eseguito tramite un IRET viene mantenuto acceso il flag B del nuovo TSS descriptor; in caso contrario, il flag viene acceso.
- 11- Setta il flag TS nell'immagine del CR0 nel nuovo TSS.
- 12- Carica il task register col selettore e descrittore del nuovo TSS.
- 13- Carica il nuovo stato dello stack dal proprio TSS.
- 14- Inizia l'esecuzione del nuovo task.

Task Linking ^^^^^^^^^^^^

Il BackLink del TSS e il flag NT sono utilizzati per far tornare l'esecuzione da un task. Infatti, il flag NT indica se il task corrente e' nidificato o meno. In caso affermativo, un iret avra' come conseguenza il ritorno al task di livello subito piu' alto (precedente, chiamante, insomma avete capito =)) della gerarchia, identificato a sua volta dal BackLink (selettore). Infatti, durante uno switch tramite call, int o exception, il processore copia il selettore del tss corrente proprio nel campo backlink del nuovo tss, e poi accende il flag NT. Ovviamente, ci possono essere piu' livelli di "nesting", cioe' nidificazione.

[NOTA]

Come avrete capito, se il task switch viene effettuato tramite un jmp, il nuovo task non e' nidificato: infatti il flag NT non viene modificato e il campo BackLink del nuovo tss non e' usato.

[FINE NOTA]

Task Address Space ^^^^^^^^^^^^^^^^

Ovviamente ad ogni nuovo task corrispondera' un proprio segmento code/data/stack, ed eventualmente anche una propria ldt. Se il paging non e' abilitato, tutti gli indirizzi lineari saranno mappati come fisici.

Se invece il paging e' attivato, il mapping lineare/fisico puo' avvenire in 2 modi: tramite un'unica page dir per tutti i task o tramite diverse page dir per ogni task (poiche' cr3 e' caricato ad ogni switch, infatti, ogni task puo' avere differenti page dir).

Il secondo metodo implica che stessi indirizzi lineari in task diversi possono essere mappati in indirizzi fisici completamente diversi, in quanto le entry della page dir possono puntare a diverse page table che puntano a loro volta a diverse pagine fisiche in memoria.

In ogni caso, i tss di ogni task devono sempre e cmq rimanere un uno spazio fisico condiviso a cui ogni task puo' accedere. Cio' e' necessario affinche' l'indirizzo del tss non cambi durante uno switch :P.

Per condividere dati tra task si puo' quindi procedere in 3 modi: - attraverso descriptor nella gdt. Poiche' tutti i task hanno accesso alla gdt, se alcuni descriptor in essa presenti puntano a segmenti nello spazio lineare mappati in un'area fisica comune a tutti i task, allora essi possono condividere dati e codice in quei segmenti.

- attraverso una ldt condivisa. Se dei task usano la stessa ldt e i descriptor in essa presenti sono mappati in un'area fisica comune, essi possono condividere dati o codice (questo metodo ovviamente e' piu' restrittivo del precedente).

- attraverso descriptor in diverse ldt che sono mappati in indirizzi comuni nello spazio lineare. Se tale area lineare comune e' mappata nella stessa area fisica, questi descriptor permetteranno di condividere segmenti. Questo metodo e' ovviamente ancora piu' restrittivo del precedente.

Conclusioni

Bene, e' tutto anche per oggi.

Spero che questo tute vi sia stato utile e vi abbia incuriosito.

Ovviamente molte delle info in esso contenute sono state prese dai vari manuali intel disponibili per il download in <http://developer.intel.com>.

Un saluto a tutti, in particolare a mrcode, che mi ha convinto a scrivere questo tute. Andate a <http://web.tiscalinet.it/mrcode> e girate bene quel suo sito, che merita davvero!

Byzz,

Ritz

Attenzione

Questo tutorial e' stato scritto a scopo unicamente didattico. L'autore non si assume alcuna responsabilita' dell'utilizzo che ne verra' fatto.