Do I need a TSS?

Moderators: mystran, JAAman, Owen, Combuster, quok, os64dev, Candy, chase, pcmattman, sortie, AJ, Brendan, 01000101, carbonBased, Antti, thepowersgang, JamesM, kmcguire

 Page 1 of 1 [10 posts]

Author	Message
pcmattman	**Post subject:** Do I need a TSS? **Posted:** Tue Apr 10, 2007 9:03 pm

offline

Member

Joined: Sun Jan 14, 2007 9:15 pm
Posts: 2550
Location: Sydney, Australia (I come from a land down under!)

I've setup a working multitasking system, one that would be expected of a microkernel. Basically, each process is a structure:

Code:

```
// process structure
typedef struct tagPROCESS {
   struct regs myregs;
   int stackstart;
   int status;
   char name[64];
} PROCESS;
```

On initialization it's setup like so:

Code:

```
   // setup the registers and other data for the process
   ProcessList[i].myregs.eip = addr;
   ProcessList[i].myregs.esp = (unsigned int) stackSpace;
   ProcessList[i].myregs.ebp = 0;
   ProcessList[i].myregs.eax = 0;
   ProcessList[i].myregs.ebx = 0;
   ProcessList[i].myregs.ecx = 0;
   ProcessList[i].myregs.edx = 0;
   ProcessList[i].myregs.edi = 0;
   ProcessList[i].myregs.esi = 0;
   ProcessList[i].myregs.cs = 0x08;
   ProcessList[i].myregs.ds = 0x10;
```

```
   ProcessList[i].myregs.es = 0x10;
   ProcessList[i].myregs.fs = 0x10;
   ProcessList[i].myregs.gs = 0x10;
   ProcessList[i].myregs.ss = 0x10;
   if( isRealMode == 1 )
   {
      ProcessList[i].myregs.eflags = EFLAGS_VM | EFLAGS_IOPL0 |
EFLAGS_IF | 0x20; // v8086 mode
   }
   else
   {
      ProcessList[i].myregs.eflags = 0x0202;
   }
   ProcessList[i].myregs.useresp = (unsigned int) stackSpace;
```

I had tried to initialize a stack but that didn't work (*stackSpace-- = whatever...).

Now, when I try to run a real mode program Bochs crashes. The log at the time of failure is like so:

Code:

```
00203256766e[CPU0 ] allow_io(): TR doesn't point to a valid
32bit TSS
00203256766p[CPU0 ] >>PANIC<< get_SS_ESP_from_TSS: TR is bogus
type (3)
00203256766i[SYS  ] Last time is 1176260980
00203256766i[CPU0 ] v8086 mode
00203256766i[CPU0 ] CS.d_b = 16 bit
00203256766i[CPU0 ] SS.d_b = 16 bit
00203256766i[CPU0 ] | EAX=00008744  EBX=00000000  ECX=0000a3fe
EDX=000000f0
00203256766i[CPU0 ] | ESP=0020b5aa  EBP=00000000  ESI=00001ff0
EDI=00000002
00203256766i[CPU0 ] | IOPL=0 id vip vif ac VM RF nt of df IF tf
SF zf af PF cf
00203256766i[CPU0 ] | SEG selector     base    limit G D
00203256766i[CPU0 ] | SEG sltr(index|ti|rpl)     base    limit
G D
00203256766i[CPU0 ] |  CS:0008( 0001| 0|  3) 00000080 0000ffff
0 0
00203256766i[CPU0 ] |  DS:0000( 0002| 0|  3) 00000000 0000ffff
0 0
00203256766i[CPU0 ] |  SS:0010( 0002| 0|  3) 00000100 0000ffff
0 0
00203256766i[CPU0 ] |  ES:0000( 0002| 0|  3) 00000000 0000ffff
0 0
00203256766i[CPU0 ] |  FS:7ade( 0002| 0|  3) 0007ade0 0000ffff
0 0
00203256766i[CPU0 ] |  GS:000a( 0002| 0|  3) 000000a0 0000ffff
0 0
00203256766i[CPU0 ] | EIP=00001f8a (00001f8a)
```

```
00203256766i[CPU0 ] | CR0=0x00000011 CR1=0 CR2=0x00000000
00203256766i[CPU0 ] | CR3=0x00000000 CR4=0x00000000
00203256766i[CPU0 ] >> insb byte ptr es:[di], dx : 656C
00203256766i[     ] restoring default signal behavior
00203256766i[CTRL ] quit_sim called with exit code 1
```

The problem is, I don't have any TSS whatsoever, and haven't needed one. Can anyone explain this? I can post code if you need it.

The FORGE Operating System | Pedigree | Blog | @pcmattman on Twitter

Top profile

mystran

offline

Member

Joined: Thu Mar 08, 2007 11:08 am
Posts: 670

Post subject: **Posted:** Tue Apr 10, 2007 10:42 pm

You need one TSS per processor. You need to store esp0 there. Rest of the TSS contents are irrelevant, unless you specifically want to use them for something, but when you come back to ring0 from ring3 (or whatever) the processor is going to load a new stack pointer from the TSS, whether you want it or not, and therefore you need one TSS for each processor.

So.. mm... yes.. you need one.

The real problem with goto is not with the control transfer, but with environments. Properly tail-recursive closures get both right.

Top profile

pcmattman

offline

Member

Joined: Sun Jan 14, 2007 9:15 pm
Posts: 2550
Location: Sydney, Australia (I come from a land down under!)

Post subject: **Posted:** Tue Apr 10, 2007 10:54 pm

Ok... can you explain how exactly I'm meant to set up the TSS and how to use it?

The FORGE Operating System | Pedigree | Blog | @pcmattman on Twitter

Top profile

Brynet-Inc

offline

Member

Post subject: **Posted:** Tue Apr 10, 2007 11:11 pm

Well, Wikipedia is always a valuable resource

http://en.wikipedia.org/wiki/Task_State_Segment

Apparently, There is documentation available here:
http://www.intel.com/products/processor ... /index.htm

http://www.osdev.org/wiki/Context_Switching might also be relevant..

Joined: Tue Oct 17, 2006 9:29 pm
Posts: 2410
Location: Canada

Twitter: @canadianbryan. Award by smcerm, I stole it. Original was larger.

UNIX&BSD's, your aspirations, only to be imitated!
Windows, are an opening in an otherwise solid and opaque surface through which light and, sometimes, even air can pass through; nothing more.

Last edited by Brynet-Inc on Tue Apr 10, 2007 11:14 pm, edited 1 time in total.

Top profile

mystran **Post subject:** **Posted:** Tue Apr 10, 2007 11:13 pm

offline

Member

Joined: Thu Mar 08, 2007 11:08 am
Posts: 670

Well you allocate one, then you load one into the task register, and then you set esp0 to whatever you want your ESP to be when you enter your kernel.

Look into the Intel manual. It tells everything you need to know about those. You can skip all the crazy stuff about actually using them for something other than ESP0. For all practical purposes, that's the only field you need to care about.

Basicly, you need an entry in your GDT (or LDT if you bother with those.. I don't).. and then you need a structure like this:

Code:

```
typedef volatile struct __tss_struct {
    unsigned short   link;
    unsigned short   link_h;

    unsigned long   esp0;
    unsigned short   ss0;
    unsigned short   ss0_h;

    unsigned long   esp1;
    unsigned short   ss1;
    unsigned short   ss1_h;

    unsigned long   esp2;
    unsigned short   ss2;
    unsigned short   ss2_h;

    unsigned long   cr3;
    unsigned long   eip;
    unsigned long   eflags;

    unsigned long   eax;
    unsigned long   ecx;
```

```
    unsigned long   edx;
    unsigned long    ebx;

    unsigned long   esp;
    unsigned long   ebp;

    unsigned long   esi;
    unsigned long   edi;

    unsigned short   es;
    unsigned short   es_h;

    unsigned short   cs;
    unsigned short   cs_h;

    unsigned short   ss;
    unsigned short   ss_h;

    unsigned short   ds;
    unsigned short   ds_h;

    unsigned short   fs;
    unsigned short   fs_h;

    unsigned short   gs;
    unsigned short   gs_h;

    unsigned short   ldt;
    unsigned short   ldt_h;

    unsigned short   trap;
    unsigned short   iomap;

} tss_struct;
```

Then you allocate one of those, and then you put a descriptor in your GDT and load the task registers with code something like this:

Code:

```
void tss_load(unsigned long cpu_num) {

    unsigned long tss_addr;

    tss_addr = (unsigned long) tss_table + cpu_num;

    /* build descriptor */
    gdt[5 + cpu_num] =
        /* base */
        ((unsigned long long) (tss_addr &       0x00ffffff) <<
16)
        + ((unsigned long long) (tss_addr & 0xff000000) << 32)
        /* attributes, 32-bit TSS, present,  */
```

```
        +
 0x0000890000000000LL
        /* limit, it's less than 2^16 anyhow, so no need for th
eupper nibble */
        + (sizeof(tss_struct)) - 1;

    asm volatile("ltr %%ax": : "a" ((5 + cpu_num)<<3));

}
```

And once you've done that, you can keep the pointer to the tss somewhere, if you allocated it dynamically (I just allocate them statically as a table, so I just need the address of the table) and then you just put your kernel datasegment descriptor into ss0 like this:

Code:

```
    tss->ss0 = 0x10;
```

and if you care about the IO bitmap, you could do something with those, but I don't bother with them, so I just:

Code:

```
    /* set to point beyond the TSS limit */
    tss->iomap = (unsigned short) sizeof(tss_struct);
```

And then the only thing from there on you need to do is put into esp0 the value you want in your ESP when you enter your kernel from outside ring0.

And that's about it.

Pretty uninteresting structure.

Like I said, the internal manual knows the details.

The real problem with goto is not with the control transfer, but with environments. Properly tail-recursive closures get both right.

Top profile

pcmattman **Post subject:** **Posted:** Wed Apr 11, 2007 12:45 am

offline

Member

Brynet-Inc wrote:

Well, Wikipedia is always a valuable resource

Joined: Sun Jan 14, 2007 9:15 pm
Posts: 2550
Location: Sydney, Australia (I come from a land down under!)

> http://en.wikipedia.org/wiki/Task_State_Segment
>
> Apparently, There is documentation available here:
> http://www.intel.com/products/processor ... /index.htm
>
> http://www.osdev.org/wiki/Context_Switching might also be relevant..

I tried the Intel manuals, they were my first point of reference. The problem is that unless I have a general idea of the code required I get exremely confused.

Now my question is: what is the access and granuality of the TSS entry in the GDT?

The FORGE Operating System | Pedigree | Blog | @pcmattman on Twitter

Top profile

Combuster

offline

Member

Joined: Wed Oct 18, 2006 3:45 am
Posts: 8617
Location: On a balcony in Vianden, watching muppets

Post subject: **Posted:** Wed Apr 11, 2007 1:43 am

the only required bits are the Present bit and the type field. Base and Size are set to to the values you want. the G bit only says that the size is in 4k pages instead of bytes when its set. The remaining fields are just zeroed out

---

"Certainly avoid yourself. He is a newbie and might not realize it. You'll hate his code deeply a few years down the road." - Sortie
[My OS] [VDisk/SFS]

Top profile

pcmattman

offline

Member

Joined: Sun Jan 14, 2007 9:15 pm
Posts: 2550
Location: Sydney, Australia (I come from a land down under!)

Post subject: **Posted:** Wed Apr 11, 2007 1:50 am

Ummm... this isn't working:

Code:

```
00021699414e[CPU0 ] LTR: doesn't point to an available TSS descriptor!
00021699414e[CPU0 ] interrupt(): gate descriptor is not valid sys seg
00021699414e[CPU0 ] interrupt(): gate descriptor is not valid sys seg
```

The code I'm using for TSS initialization:

Code:

```
// TSS functions

#include "sys/mattise.h"
#include "sys/process.h"

// the system TSS
TSS_t MyTSS;

// loads the TSS - defined in the GDT file
void LoadTSS( TSS_t* tss_table );

// initializes the tss handling
void InitTss()
{
   // start by loading the TSS address into the GDT
   LoadTSS( &MyTSS );

   // now fill each value
   int i;
   for( i = 0; i < 250; i++ )
   {
      // set values necessary
      MyTSS.ss0 = 0x10;

      // now set the IO bitmap (not necessary, so set above
limit)
      MyTSS.iomap = ( unsigned short ) sizeof( TSS_t );
   }
}

// returns a pointer to an entry
void GetTSS( TSS_t** ent, int id )
{
   *ent = &MyTSS;
}
```

Code for GDT

Code:

```
#include "sys/mattise.h"
#include "sys/process.h"

/* Defines a GDT entry. We say packed, because it prevents the
*  compiler from doing things that it thinks is best: Prevent
*  compiler "optimization" by packing */
struct gdt_entry
{
    unsigned short limit_low;
    unsigned short base_low;
```

```
    unsigned char base_middle;
    unsigned char access;
    unsigned char granularity;
    unsigned char base_high;
} __attribute__((packed));

// definitions of gate attributes

#define LDT         0x200   // ldt segment
#define TASK      0x500   // task gate
#define TSS         0x900   // tss
#define CALL      0x0C00   // 386 call gate
#define INT         0x0E00   // 386 interrupt gate
#define TRAP      0x0F00   // 386 trap gate
#define DATA      0x1000   // data segment
#define CODE      0x1800   // code segment

#define DPL3      0x6000   // dpl3
#define DLP2      0x4000   // dpl2
#define DPL1      0x2000   // dpl1
#define DPL0      0x0000   // dpl0
#define PRESENT      0x8000   // present
#define NPRESENT   0x8000   // not present
                     // present is set by default, non-present
to turn it off
                     // present does the same

#define ACC         0x100   // accessed (ds/cs)
#define WRITE      0x200   // writable (cs)
#define READ      0x200   // readable (cs)
#define BUSY      0x200   // busy (cs)
#define EXDOWN      0x400   // expand down (ds)
#define CONFORM      0x400   // conforming (cs)
#define BIG         0x40   // default to 32bit
#define BIG_LIM      0x80   // limit in 4k units

/* Special pointer which includes the limit: The max bytes
*  taken up by the GDT, minus 1. Again, this NEEDS to be packed
*/
struct gdt_ptr
{
    unsigned short limit;
    unsigned int base;
} __attribute__((packed));

/* Our GDT, with 3 entries, and finally our special GDT pointer
*/
struct gdt_entry gdt[15];
struct gdt_ptr gp;

/* This will be a function in start.asm. We use this to
properly
*  reload the new segment registers */
extern void gdt_flush();
```

```
/* Setup a descriptor in the Global Descriptor Table */
void gdt_set_gate(int num, unsigned long base, unsigned long
limit, unsigned char access, unsigned char gran)
{
    /* Setup the descriptor base address */
    gdt[num].base_low = (base & 0xFFFF);
    gdt[num].base_middle = (base >> 16) & 0xFF;
    gdt[num].base_high = (base >> 24) & 0xFF;

    /* Setup the descriptor limits */
    gdt[num].limit_low = (limit & 0xFFFF);
    gdt[num].granularity = ((limit >> 16) & 0x0F);

    /* Finally, set up the granularity and access flags */
    gdt[num].granularity |= (gran & 0xF0);
    gdt[num].access = access;
}

/* Should be called by main. This will setup the special GDT
*  pointer, set up the first 3 entries in our GDT, and then
*  finally call gdt_flush() in our assembler file in order
*  to tell the processor where the new GDT is and update the
*  new segment registers */
void gdt_install()
{
    /* Setup the GDT pointer and limit */
    gp.limit = (sizeof(struct gdt_entry) * 3) - 1;
    gp.base = (unsigned int) &gdt;

    /* Our NULL descriptor */
    gdt_set_gate(0, 0, 0, 0, 0);

    /* The second entry is our Code Segment. The base address
    *  is 0, the limit is 4GBytes, it uses 4KByte granularity,
    *  uses 32-bit opcodes, and is a Code Segment descriptor.
    *  Please check the table above in the tutorial in order
    *  to see exactly what each value means */
    gdt_set_gate(1, 0, 0xFFFFFFFF, 0x9A, 0xCF);

    /* The third entry is our Data Segment. It's EXACTLY the
    *  same as our code segment, but the descriptor type in
    *  this entry's access byte says it's a Data Segment */
    gdt_set_gate(2, 0, 0xFFFFFFFF, 0x92, 0xCF);
}

// loads the TSS
void LoadTSS( TSS_t* tss_table )
{
   // address of the tss
    unsigned long base;

   // fill it
    base = (unsigned long) tss_table;
```

```
   // size of the TSS
   int size = sizeof( TSS_t );

   gdt_set_gate( 3, base, base + size, TSS, 0xCF );

   // location of task register to load
   int tra = 3 << 3;

   // load it
   __asm__ __volatile__ ( "ltr %%ax" : : "a" ( tra ) );

   // flush and install changes
   // must be here because this is called after GDT
installation
    gdt_flush();
}
```

The FORGE Operating System | Pedigree | Blog | @pcmattman on Twitter

Top profile

Combuster

offline

Member

Joined: Wed Oct 18, 2006 3:45 am
Posts: 8617
Location: On a balcony in Vianden, watching muppets

Post subject: **Posted:** Wed Apr 11, 2007 2:24 am

Passing large numbers as signed chars? you should check the #defines

Another anomaly you might want to look at (the documentation is bogus):

Code:

```
#define PRESENT      0x8000   // present
#define NPRESENT   0x8000   // not present
                     // present is set by default, non-present
to turn it off
                     // present does the same
```

"Certainly avoid yourself. He is a newbie and might not realize it. You'll hate his code deeply a few years down the road." - Sortie
[My OS] [VDisk/SFS]

Top profile

pcmattman

offline

Member

Joined: Sun Jan 14, 2007 9:15 pm

Post subject: **Posted:** Wed Apr 11, 2007 2:31 am

The defines converted from assembly code over on osdever.net.

One of the few things that really confused me in the kernel development tutorial was the GDT, mainly because it was more of a case of 'here's some code' than actual explanation of which bits go where and whatnot.

Posts: 2550
Location: Sydney, Australia (I come from a land down under!)

The FORGE Operating System | Pedigree | Blog | @pcmattman on Twitter

Top profile

Display posts from previous: All posts ▼ Sort by Post time ▼ Ascending ▼ Go

newtopic postreply **Page 1 of 1** [10 posts]

Board index » Operating System Development » OS Development All times are UTC - 6 hours

Who is online

Users browsing this forum: beyondsociety and 5 guests

You **cannot** post new topics in this forum
You **cannot** reply to topics in this forum
You **cannot** edit your posts in this forum
You **cannot** delete your posts in this forum
You **cannot** post attachments in this forum

Search for: Go

Jump to: OS Development ▼ Go