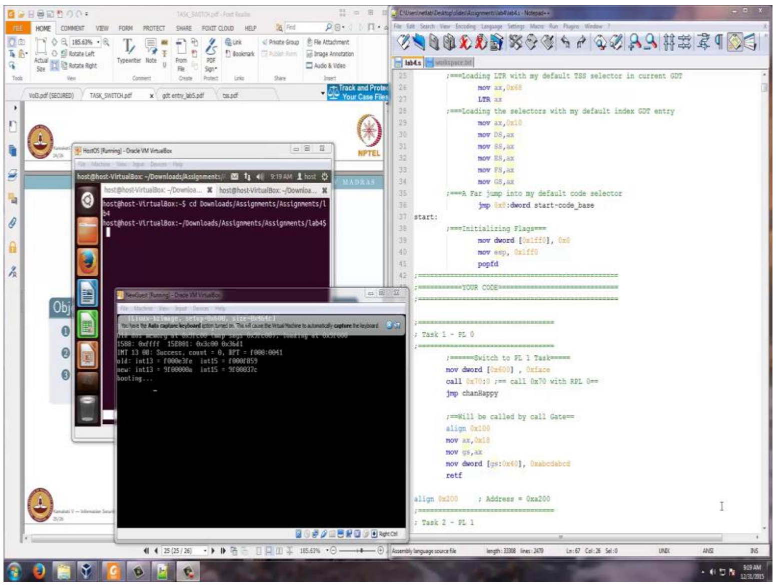
Information Security – II
Prof. V. Kamakoti
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture - 31
Lab4 Part 1 - Week 6

(Refer Slide Time: 00:09)



```
host@host-VirtualBox:~$ cd Downloads/Assignments/Assignments/l
b4
host@host-VirtualBox:~/Downloads/Assignments/Assignments/lab4$
```

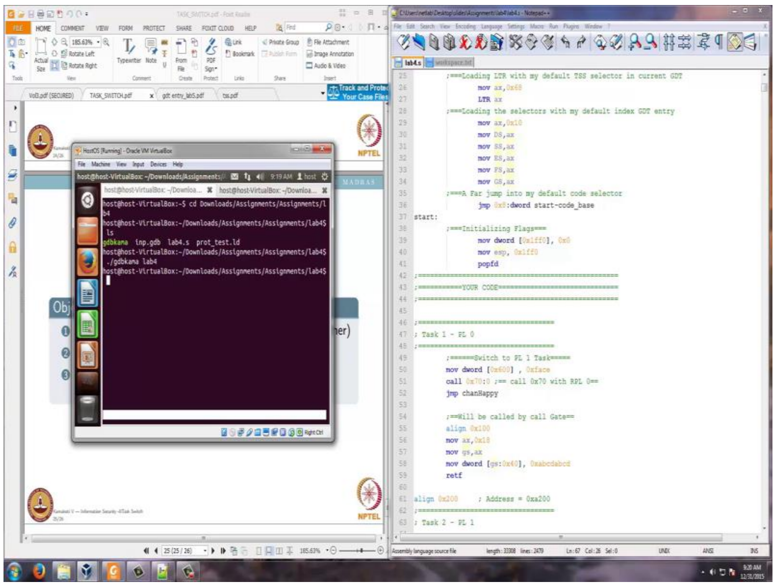
```
[Linux-bzImage, setup=0x600, size=0x4b4c]
Old dos memory at 0x9fc00 (map segs 0x9fc00); loading at 0x9f000
1588: 0xffff  15E801: 0x3c00 0x3641
INT 13 08: Success, count = 0, BPT = f000:0041
old: int13 = f000e3fe  int15 = f000f859
new: int13 = 9f00000a  int15 = 9f00037c
booting...
```

```
        ;===Loading LTR with my default TSS selector in current GDT
                mov ax,0x68
                LTR ax
        ;===Loading the selectors with my default index GDT entry
                mov ax,0x10
                mov DS,ax
                mov SS,ax
                mov ES,ax
                mov FS,ax
                mov GS,ax
        ;===A Far jump into my default code selector
                jmp 0x8:dword start-code_base
start:
        ;===Initializing Flags===
                mov dword [0x1ff0], 0x0
                mov esp, 0x1ff0
                popfd
;====================================================
;===========YOUR CODE================================
;====================================================


;==================================
; Task 1 - PL 0
;==================================
        ;======Switch to PL 1 Task=====
        mov dword [0x600] , 0xface
        call 0x70:0 ;== call 0x70 with RPL 0==
        jmp chanHappy

        ;==Will be called by call Gate==
        align 0x100
        mov ax,0x18
        mov gs,ax
        mov dword [gs:0x40], 0xabcdabcd
        retf

align 0x200     ; Address = 0xa200
;==================================
; Task 2 - PL 1
```

C:\Users\netlab\Desktop\slides\Assignments\lab4\lab4.s - Notepad++
lab4.s
workspace.txt
HostOS [Running] - Oracle VM VirtualBox
NewGuest [Running] - Oracle VM VirtualBox
You have the Auto capture keyboard option turned on. This will cause the Virtual Machine to automatically capture the keyboard
TASK_SWITCH.pdf
gdt entry_lab5.pdf
tss.pdf
NPTEL
Assembly language source file
length : 33308   lines : 2479
Ln : 67   Col : 26   Sel : 0
UNIX
ANSI
INS
9:19 AM
12/31/2015

Welcome to assignment number 4 or lab 4. Please ensure that your guest machine is running and also your host machine, you are logged into host machine.

(Refer Slide Time: 00:28)

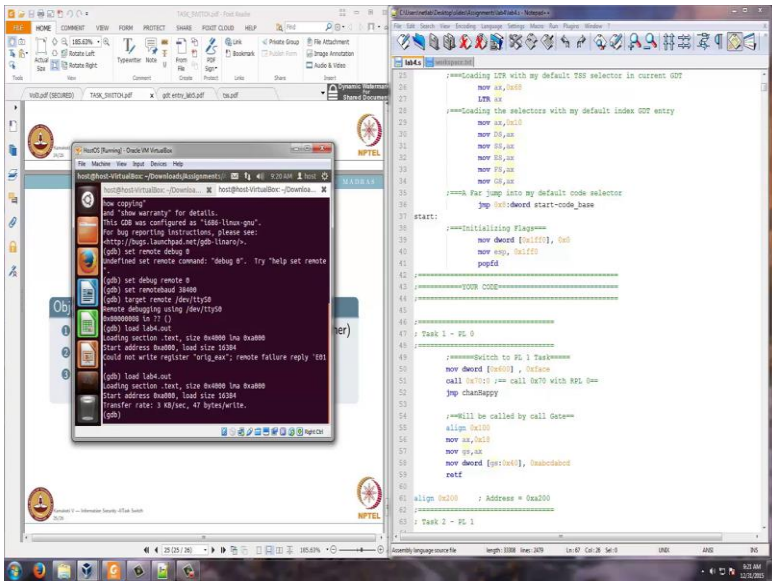


```
host@host-VirtualBox:~$ cd Downloads/Assignments/Assignments/l
b4
host@host-VirtualBox:~/Downloads/Assignments/Assignments/lab4$
 ls
gdbkama  inp.gdb  lab4.s  prot_test.ld
host@host-VirtualBox:~/Downloads/Assignments/Assignments/lab4$
 ./gdbkama lab4
host@host-VirtualBox:~/Downloads/Assignments/Assignments/lab4$
```

```
        ;===Loading LTR with my default TSS selector in current GDT
                mov ax,0x68
                LTR ax
        ;===Loading the selectors with my default index GDT entry
                mov ax,0x10
                mov DS,ax
                mov SS,ax
                mov ES,ax
                mov FS,ax
                mov GS,ax
        ;===A Far jump into my default code selector
                jmp 0x8:dword start-code_base
start:
        ;===Initializing Flags===
                mov dword [0x1ff0], 0x0
                mov esp, 0x1ff0
                popfd
;=====================================================
;===========YOUR CODE=================================
;=====================================================

;==================================
; Task 1 - PL 0
;==================================
        ;======Switch to PL 1 Task=====
        mov dword [0x600] , 0xface
        call 0x70:0 ;== call 0x70 with RPL 0==
        jmp chanHappy

        ;==Will be called by call Gate==
        align 0x100
        mov ax,0x18
        mov gs,ax
        mov dword [gs:0x40], 0xabcdabcd
        retf

align 0x200     ; Address = 0xa200
;==================================
; Task 2 - PL 1
```

Go to lab 4, and there you will find the lab four dot s, compile it, and now go to the log into the GDB.

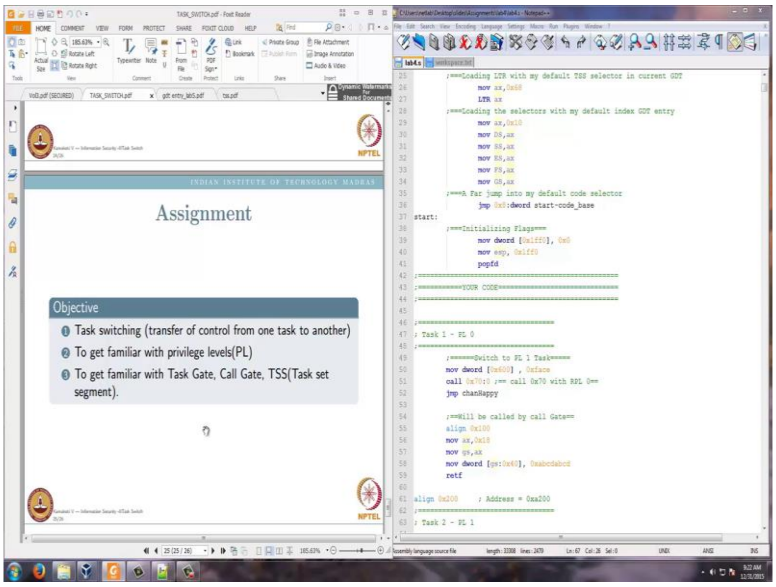
(Refer Slide Time: 00:48)



```
how copying"
and "show warranty" for details.
This GDB was configured as "i686-linux-gnu".
For bug reporting instructions, please see:
<http://bugs.launchpad.net/gdb-linaro/>.
(gdb) set remote debug 0
Undefined set remote command: "debug 0".  Try "help set remote
".
(gdb) set debug remote 0
(gdb) set remotebaud 38400
(gdb) target remote /dev/ttyS0
Remote debugging using /dev/ttyS0
0x00000000 in ?? ()
(gdb) load lab4.out
Loading section .text, size 0x4000 lma 0xa000
Start address 0xa000, load size 16384
Could not write register "orig_eax"; remote failure reply 'E01
'
(gdb) load lab4.out
Loading section .text, size 0x4000 lma 0xa000
Start address 0xa000, load size 16384
Transfer rate: 3 KB/sec, 47 bytes/write.
(gdb)
```

Set debug remote 0, set remotebaud 38400, target remote slash dev slash tty s 0. So, we are inside this, and now we can load lab four dot out.

(Refer Slide Time: 01:54)



INDIAN INSTITUTE OF TECHNOLOGY MADRAS

Assignment

Objective

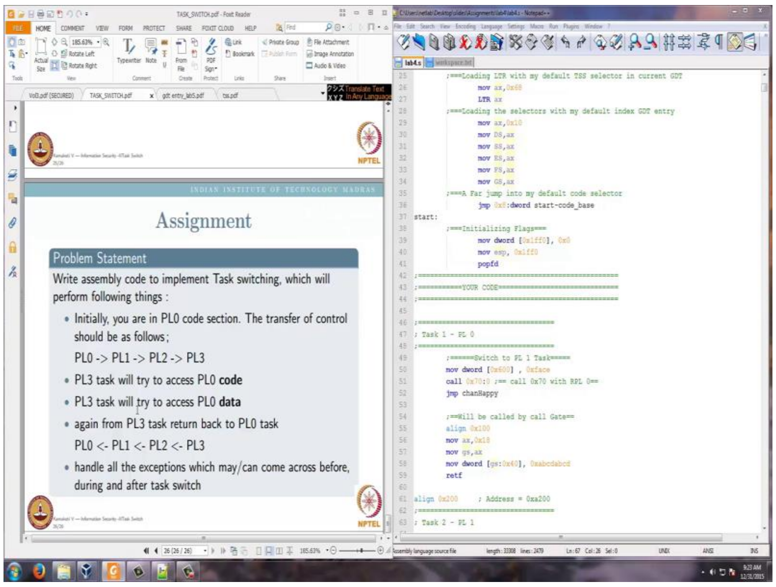
1. Task switching (transfer of control from one task to another)
2. To get familiar with privilege levels(PL)
3. To get familiar with Task Gate, Call Gate, TSS(Task set segment).

```
        ;===Loading LTR with my default TSS selector in current GDT
                mov ax,0x68
                LTR ax
        ;===Loading the selectors with my default index GDT entry
                mov ax,0x10
                mov DS,ax
                mov SS,ax
                mov ES,ax
                mov FS,ax
                mov GS,ax
        ;===A Far jump into my default code selector
                jmp 0x8:dword start-code_base
start:
        ;===Initializing Flags===
                mov dword [0x1ff0], 0x0
                mov esp, 0x1ff0
                popfd
;==================================================
;==========YOUR CODE===============================
;==================================================

;==================================
; Task 1 - PL 0
;==================================
        ;======Switch to PL 1 Task=====
        mov dword [0x600] , 0xface
        call 0x70:0 ;== call 0x70 with RPL 0==
        jmp chanHappy

        ;==Will be called by call Gate==
        align 0x100
        mov ax,0x18
        mov gs,ax
        mov dword [gs:0x40], 0xabcdabcd
        retf

align 0x200     ; Address = 0xa200
;==================================
; Task 2 - PL 1
```

Before, we execute this lab four dot out, let us go and see what this assignment is all about. This assignment teaches you, till now what we have done in the lab 1, lab 2 and lab 3, we have not gone beyond privilege level zero. Now, we will go to privilege levels more than 0. So, what we are trying to do in this assignment is to understand, how we can switch the privilege levels and if I want to switch privilege level, basically we are doing it using tasks. So, we will be creating for tasks, tasks 0, task 1, task 2, and task 3; task 0 will execute at privilege zero, task 1 at privilege one, task 2 at privilege two and task 3 at privilege three.

We will switch from PL 0 to PL 1 to PL 2 to PL 3. And that PL 3, we will come and access a code which is PL 1 through a call gate and then go back, and also prove certain things like we will try and do at privilege three we will try and access a segment which is privilege one and then there will be a segmentation fault. So, this is basically have to understand the protection that is offered by the Intel S86 the S86 architecture Intel or AMD right and this is what we are going to achieve in this assignment.

(Refer Slide Time: 03:17)



INDIAN INSTITUTE OF TECHNOLOGY MADRAS

Assignment

Problem Statement

Write assembly code to implement Task switching, which will perform following things :

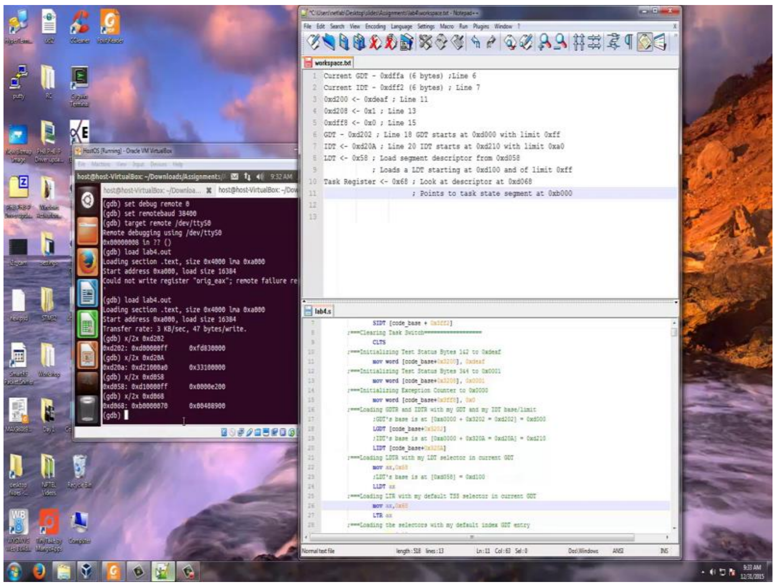
- Initially, you are in PL0 code section. The transfer of control should be as follows;
  PL0 -> PL1 -> PL2 -> PL3
- PL3 task will try to access PL0 **code**
- PL3 task will try to access PL0 **data**
- again from PL3 task return back to PL0 task
  PL0 <- PL1 <- PL2 <- PL3
- handle all the exceptions which may/can come across before, during and after task switch

```
        ;===Loading LTR with my default TSS selector in current GDT
                mov ax,0x68
                LTR ax
        ;===Loading the selectors with my default index GDT entry
                mov ax,0x10
                mov DS,ax
                mov SS,ax
                mov ES,ax
                mov FS,ax
                mov GS,ax
        ;===A Far jump into my default code selector
                jmp 0x8:dword start-code_base
start:
        ;===Initializing Flags===
                mov dword [0x1ff0], 0x0
                mov esp, 0x1ff0
                popfd
;=================================================
;===========YOUR CODE=============================
;=================================================

;=================================
; Task 1 - PL 0
;=================================
        ;======Switch to PL 1 Task=====
        mov dword [0x600] , 0xface
        call 0x70:0 ;== call 0x70 with RPL 0==
        jmp chanHappy

        ;==Will be called by call Gate==
        align 0x100
        mov ax,0x18
        mov gs,ax
        mov dword [gs:0x40], 0xabcdabcd
        retf

align 0x200     ; Address = 0xa200
;=================================
; Task 2 - PL 1
```

So, to sum up here, we are actually in PL0 code section always. Till now we have seen that in all the remaining three labs that we have covered so far. Now what we will do is we do this PL 0 to PL 1 to PL 2 to PL 3; In PL 3 from PL 3, we will go and excess PL 0 code from PL and through a call gate. And again at PL 3, we will try to access at PL 0 or PL 1 data and then this will create segmentation for, it gives general protection fault or a segmentation fault, we will see what sort of fault it is going to create. And then we will come back and then after that we will return from PL 3 to PL 2 to PL 1 to PL 0. And here also we will see that since PL 3 is call causing at PL 0 task we are sorry PL 3 is trying to access the PL 0 data, there is interrupt that is happening and the inter service routine will work at privilege 0.

So, will also show that this task corresponding to PL 3 is being used PL 0 is being used there is a task switch that is happening which I had explained and that is very, very important from the point of protection. And then we will sum up this with all these things. So, that we have full understanding of what has happening in this whole step. So, with this as the background let us start looking at the code. So, we will now see what is happening here, very quickly. Now as you see that we need to go and look at much more deeper understanding of the first few parts of the code; so we are storing the…

(Refer Slide Time: 05:34)



```
Current GDT - 0xdffa (6 bytes) ;Line 6
Current IDT - 0xdff2 (6 bytes) ; Line 7
0xd200 <- 0xdeaf ; Line 11
0xd208 <- 0x1 ; Line 13
0xdff8 <- 0x0 ; Line 15
GDT - 0xd202 ; Line 18 GDT starts at 0xd000 with limit 0xff
IDT <- 0xd20A ; Line 20 IDT starts at 0xd210 with limit 0xa0
LDT <- 0x58 ; Load segment descriptor from 0xd058
            ; Loads a LDT starting at 0xd100 and of limit 0xff
Task Register <- 0x68 ; Look at descriptor at 0xd068
                      ; Points to task state segment at 0xb000
```

```
(gdb) set debug remote 0
(gdb) set remotebaud 38400
(gdb) target remote /dev/ttyS0
Remote debugging using /dev/ttyS0
0x00000000 in ?? ()
(gdb) load lab4.out
Loading section .text, size 0x4000 lma 0xa000
Start address 0xa000, load size 16384
Could not write register "orig_eax"; remote failure re
.
(gdb) load lab4.out
Loading section .text, size 0x4000 lma 0xa000
Start address 0xa000, load size 16384
Transfer rate: 3 KB/sec, 47 bytes/write.
(gdb) x/2x 0xd202
0xd202: 0xd00000ff      0xfd830000
(gdb) x/2x 0xd20A
0xd20a: 0xd21000a0      0x33100000
(gdb) x/2x 0xd058
0xd058: 0xd10000ff      0x0000e200
(gdb) x/2x 0xd068
0xd068: 0xb0000070      0x00408900
(gdb)
```

```
7           SIDT [code_base + 0x3ff2]
8       ;===Clearing Task Switch==================
9           CLTS
10      ;===Initializing Test Status Bytes 1&2 to 0xdeaf
11          mov word [code_base+0x3200], 0xdeaf
12      ;===Initializing Test Status Bytes 3&4 to 0x0001
13          mov word [code_base+0x3208], 0x0001
14      ;===Initializing Exception Counter to 0x0000
15          mov word [code_base+0x3ff8], 0x0
16      ;===Loading GDTR and IDTR with my GDT and my IDT base/limit
17          ;GDT's base is at [0xa000 + 0x3202 = 0xd202] = 0xd000
18          LGDT [code_base+0x3202]
19          ;IDT's base is at [0xa000 + 0x320A = 0xd20A] = 0xd210
20          LIDT [code_base+0x320A]
21      ;===Loading LDTR with my LDT selector in current GDT
22          mov ax,0x58
23          ;LDT's base is at [0xd058] = 0xd100
24          LLDT ax
25      ;===Loading LTR with my default TSS selector in current GDT
26          mov ax,0x68
27          LTR ax
28      ;===Loading the selectors with my default index GDT entry
```

So, let me just go here we need to understand several parts, we are storing the GDT, current GDT is stored at 0x code basis a000 plus 3ffa, dffa and this is 6 bytes. This is line number 6. Now, we are storing the current IDT at 0xdff2 level is again 6 bytes at line number seven. So, can you open it separately, so we will open it separately. We will go up and down here. So, I am doing this assignment in full, so that we have a good understanding. We move 0 to d200, 0xd200 is set to 0xf this is line 11; 0xd208 is set to 1, this is line number 13. Then 0xdff8 is set to 0, and this is line number 15. Why, dff8, because code basis a000, a000 plus 3fff8 is dfff8. And also note that there will be a data segment to which this is adding the current data segment which it is working at starts at 0 we have explained it earlier its starts at 0 and we also show that it is staring at 0 shortly.

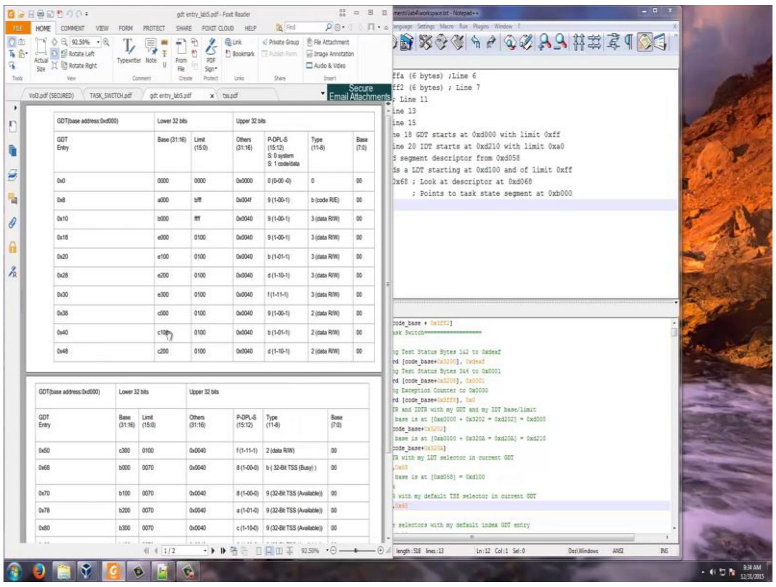
Now we load. So, let us go to line number 18, we load GDT with what is that in 0xtd202 which is line number 18. So, let us see the 6 bytes there. Now, as you see, to the GDT starts at the d000 and as a limit of ff, GDT start at d000 and also limit of ff. So, the remaining four, which has 0 here, 0000 d000, 00ff is the limit. So, this shows that GDT starts at 0xd000 with limit 0xff. Now similarly your IDT is loaded with 0xd20a this is by line 20. So, let us see what is there at that point. So, it starts at d210, IDT starts at 0xd210 with limit 0xa0, you see here. So, d2 0000, d 1 0 is the base and 00a0 is the limit. Now, the most important thing that we need to know here is about the current IDT, current

LDT.

So, please see that LDT is loaded with 0x58. So, where is 0x by a it should be x slash 2x0d058 because 58 is in the GDT and as you know that the GDT starts at d000 here as you use here on you are right hand side. So, we load segment descriptor from 0xd058 and what is that at d058. Please note that, this is giving me LDT that starts at 0xd100 and of size 00ff. And this e 2 basically tells you that is LDT descriptor. So, this loads LDT, loads a LDT starting at 0xd100 and of limit 0xff. Now let us go to line 27 your task register is loaded with that segment descriptor basically 0x68.

So, look at descriptor in again this is GDT at 0xd068, why again d068, because d000 is where you are GDT is starts. So, let us go and look at the descriptor at x dash 2xd068. This basically gives as if you look at 4 0 8 9, this is a pointer to a task state segment this is the TSS descriptor which is staring at d000, points to task state segment at 0xd000. Please note that 70 is the limit. So, it is given 70 is the limit, and you know the task descriptor can be of at most sense 104 bytes. So, I have giving something more than a 104 bytes, this is the basically limit of that task state segment descriptor.

(Refer Slide Time: 14:19)



GDT(base address:0xd000)	Lower 32 bits		Upper 32 bits			
GDT Entry	Base (31:16)	Limit (15:0)	Others (31:16)	P-DPL-S (15:12) S: 0 system S: 1 code/data	Type (11-8)	Base (7:0)
0x0	0000	0000	0x0000	0 (0-00 -0)	0	00
0x8	a000	bfff	0x004f	9 (1-00-1)	b (code R/E)	00
0x10	b000	ffff	0x0040	9 (1-00-1)	3 (data R/W)	00
0x18	e000	0100	0x0040	9 (1-00-1)	3 (data R/W)	00
0x20	e100	0100	0x0040	b (1-01-1)	3 (data R/W)	00
0x28	e200	0100	0x0040	d (1-10-1)	3 (data R/W)	00
0x30	e300	0100	0x0040	f (1-11-1)	3 (data R/W)	00
0x38	c000	0100	0x0040	9 (1-00-1)	2 (data R/W)	00
0x40	c100	0100	0x0040	b (1-01-1)	2 (data R/W)	00
0x48	c200	0100	0x0040	d (1-10-1)	2 (data R/W)	00

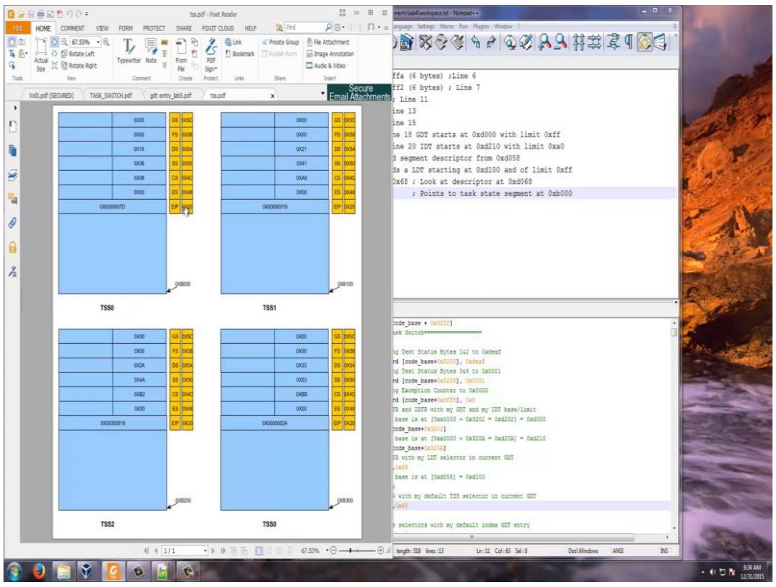
GDT(base address:0xd000)	Lower 32 bits		Upper 32 bits			
GDT Entry	Base (31:16)	Limit (15:0)	Others (31:16)	P-DPL-S (15:12)	Type (11-8)	Base (7:0)
0x50	c300	0100	0x0040	f (1-11-1)	2 (data R/W)	00
0x68	b000	0070	0x0040	8 (1-00-0)	b (32-Bit TSS (Busy))	00
0x70	b100	0070	0x0040	8 (1-00-0)	9 (32-Bit TSS (Available))	00
0x78	b200	0070	0x0040	a (1-01-0)	9 (32-Bit TSS (Available))	00
0x80	b300	0070	0x0040	c (1-10-0)	9 (32-Bit TSS (Available))	00

```
ffa (6 bytes) ;Line 6
ff2 (6 bytes) ; Line 7
; Line 11
ine 13
ine 15
ne 18 GDT starts at 0xd000 with limit 0xff
ine 20 IDT starts at 0xd210 with limit 0xa0
d segment descriptor from 0xd058
ds a LDT starting at 0xd100 and of limit 0xff
0x68 ; Look at descriptor at 0xd068
     ; Points to task state segment at 0xb000
```

```
code_base + 0x3ff2]
ask Switch*******************

ng Test Status Bytes 1&2 to 0xdeaf
rd [code_base+0x3200], 0xdeaf
ng Test Status Bytes 3&4 to 0x0001
rd [code_base+0x3208], 0x0001
ng Exception Counter to 0x0000
rd [code_base+0x3ff8], 0x0
TR and IDTR with my GDT and my IDT base/limit
 base is at [0xa0000 + 0x3202 = 0xd202] = 0xd000
code_base+0x3202]
 base is at [0xa0000 + 0x320A = 0xd20A] = 0xd210
code_base+0x320A]
TR with my LDT selector in current GDT
,0x58
 base is at [0xd058] = 0xd100

R with my default TSS selector in current GDT
,0x68

 selectors with my default index GDT entry
```

Now, let us go and see that in details here. So, we will just see here. So, this is three thousand. So, let us go and see what was there in 0x68, 0x68b000 limit is 70, it is a 32 bit TSS. So, this is the entire GDT that we are loaded here.

(Refer Slide Time: 14:58)



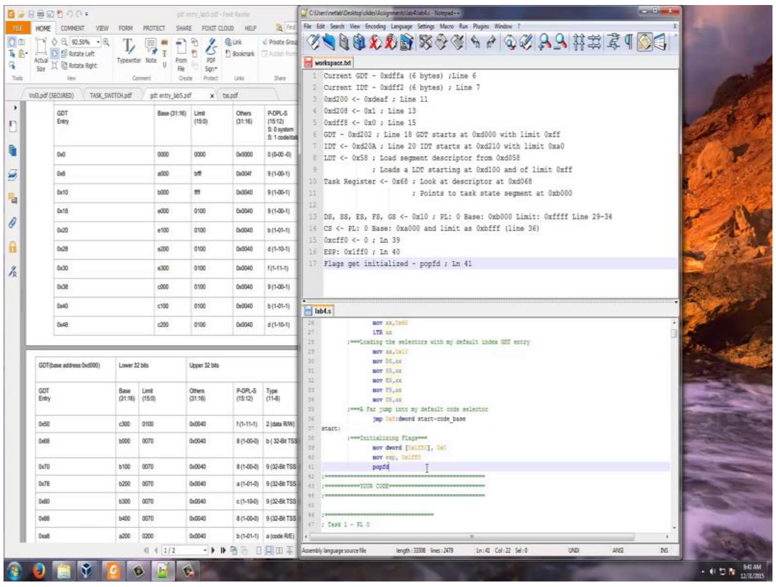
```
ffa (6 bytes) ;Line 6
ff2 (6 bytes) ; Line 7
; Line 11
ine 13
ine 15
ne 18 GDT starts at 0xd000 with limit 0xff
ine 20 IDT starts at 0xd210 with limit 0xa0
d segment descriptor from 0xd058
ds a LDT starting at 0xd100 and of limit 0xff
0x68 ; Look at descriptor at 0xd068
     ; Points to task state segment at 0xb000
```

```
code_base + 0x3ff2]
ask Switch=================

ng Test Status Bytes 1&2 to 0xdeaf
rd [code_base+0x3200], 0xdeaf
ng Test Status Bytes 3&4 to 0x0001
rd [code_base+0x3208], 0x0001
ng Exception Counter to 0x0000
rd [code_base+0x3ff8], 0x0
TR and IDTR with my GDT and my IDT base/limit
 base is at [0xa0000 + 0x3202 = 0xd202] = 0xd000
code_base+0x3202]
 base is at [0xa0000 + 0x320A = 0xd20A] = 0xd210
code_base+0x320A]
TR with my LDT selector in current GDT
,0x58
 base is at [0xd058] = 0xd100

R with my default TSS selector in current GDT
,0x68

 selectors with my default index GDT entry
```

Now, the TSS, now will be like this. It is starting at b000, and you are e i p will be stored at b000 20 cs, ds etcetera, etcetera, etcetera. So, this is how your task segment 0 will look like. Currently, we are loading everything and we are assorting a task state segment, please note that these things are already loaded by the operating system. So, currently if you look at the task state segment, these thing when will be uninitialized, because this current task is executing. The moment I am going to do a task switch from TSS 0 to say TSS 1, then all this values get filled up there and then we go. Now, so this is currently for the current task that is at the first task that is executing, the values for these cs, ss and all these things will be initialized by the operating system and when then there is a context switch then that this TSS 0 will get filled with the relevant values.

So, first thing is we are just allocating a TSS segment for the PL 0, but the initialization of all these registers etcetera will not happen from this, because the operating system will do it for the first 0 to task it has and then when we do a task switching then the current values of TSS 0 would be at the time of the task switching will be stored here. And then when we these values say suppose ago from TSS 0 to TSS 1 then these values will be automatically loaded. So, the operating system will set up these values TSS 1, 2 and 3, it should be TSS 3, this is set values for these, but for the first task, it is setting the value because it is allocating the TSS segment itself. So, this is something that we need to understand here. So, now, I have allocated a task is a segment details b000 for the current task.

(Refer Slide Time: 17:10)



GDT Entry	Base (31:16)	Limit (15:0)	Others (31:16)	P-DPL-S (15:12) S: 0 system S: 1 code/data
0x0	0000	0000	0x0000	0 (0-00 -0)
0x8	a000	bfff	0x004f	9 (1-00-1)
0x10	b000	ffff	0x0040	9 (1-00-1)
0x18	e000	0100	0x0040	9 (1-00-1)
0x20	e100	0100	0x0040	b (1-01-1)
0x28	e200	0100	0x0040	d (1-10-1)
0x30	e300	0100	0x0040	f (1-11-1)
0x38	c000	0100	0x0040	9 (1-00-1)
0x40	c100	0100	0x0040	b (1-01-1)
0x48	c200	0100	0x0040	d (1-10-1)

GDT(base address:0xd000)	Lower 32 bits		Upper 32 bits		
GDT Entry	Base (31:16)	Limit (15:0)	Others (31:16)	P-DPL-S (15:12)	Type (11-8)
0x50	c300	0100	0x0040	f (1-11-1)	2 (data R/W)
0x68	b000	0070	0x0040	8 (1-00-0)	b (32-Bit TSS
0x70	b100	0070	0x0040	8 (1-00-0)	9 (32-Bit TSS
0x78	b200	0070	0x0040	a (1-01-0)	9 (32-Bit TSS
0x80	b300	0070	0x0040	c (1-10-0)	9 (32-Bit TSS
0x88	b400	0070	0x0040	8 (1-00-0)	9 (32-Bit TSS
0xa8	a200	0200	0x0040	b (1-01-1)	a (code R/E)

```
Current GDT - 0xdffa (6 bytes) ;Line 6
Current IDT - 0xdff2 (6 bytes) ; Line 7
0xd200 <- 0xdeaf ; Line 11
0xd208 <- 0x1 ; Line 13
0xdff8 <- 0x0 ; Line 15
GDT - 0xd202 ; Line 18 GDT starts at 0xd000 with limit 0xff
IDT <- 0xd20A ; Line 20 IDT starts at 0xd210 with limit 0xa0
LDT <- 0x58 ; Load segment descriptor from 0xd058
            ; Loads a LDT starting at 0xd100 and of limit 0xff
Task Register <- 0x68 ; Look at descriptor at 0xd068
                      ; Points to task state segment at 0xb000

DS, SS, ES, FS, GS <- 0x10 ; PL: 0 Base: 0xb000 Limit: 0xffff Line 29-34
CS <- PL: 0 Base: 0xa000 and limit as 0xbfff (line 36)
0xcff0 <- 0 ; Ln 39
ESP: 0x1ff0 ; Ln 40
Flags get initialized - popfd ; Ln 41
```

```
26              mov ax,0x68
27              LTR ax
28          ;===Loading the selectors with my default index GDT entry
29              mov ax,0x10
30              mov DS,ax
31              mov SS,ax
32              mov ES,ax
33              mov FS,ax
34              mov GS,ax
35          ;===A far jump into my default code selector
36              jmp 0x8:dword start-code_base
37  start:
38          ;===Initializing Flags===
39              mov dword [0x1ff0], 0x0
40              mov esp, 0x1ff0
41              popfd
42  ;================================================
43  ;==========YOUR CODE==============================
44  ;================================================
45
46  ;=================================
47  ; Task 1 - PL 0
```

And now I am moving all ds, ss, es, fs, gs everything as 0 extends, so what is… So all your ds, ss, ef, fs and gs will have 0 extends, this means what this means let us go the tenth entry here b000 is base, fff is limit and it is type data read write and it is privilege zero; privilege zero this is means privilege is zero, base is 0xb000, and limit is ffff. And therefore, all and this is done by whom by lines 29 to 34. Now in line number 36, actually I jump to the next where if you say d word start minus code base start is here code base is above here code base is staring here. So, start minus code base essentially gives me how many bytes these things these instructions up to this, till including this. So, from the beginning to this, the total bytes consumed by these instructions from the beginning of this program to this that is what is start minus code base.

So, now, I say jump 0x8, what is the eight descriptor, eight descriptor starts it a000. So, a000 plus the total number of ways consumed by this to this, that will be nothing but this address. So, when I do this jump 0x8 colon from start minus code base, I actually jump into this instruction, but what I have achieved is my code segment now will be different code segment that is the one starting at a000 with privilege zero and limit the ffff. Now, my cs becomes PL 0, it base has 0xa000 and limit as 0xbfff, and this is done by line 36 of code. Now I am initializing the flag to be 0 here. So, I am making the location 0x what is ds, ds is now b000, b000 plus 1ff0, 0xcff0, I making it 0 and I making ESP point to 0x1ff0. So, when I say pop f t, now flags get initialized and initializing flags. So, this is line 39 to 41, this is line 39, line 40, pop f d popped as stack. So, it pops the value zero into the flag register. So, this is the line 41. So, we reach this stage and now we start our code.