

Computer Organization and Architecture
Prof. V. Kamakoti
Department of Computer Science and Engineering
Indian Institute of Technology, Madras

Lecture – 38
(Part – 2)
Lab 4: Task Switching (Contd)

So, in the lab; so, another important aspect of operating system is that we need to do task switching. So, we already had some notion of what is a layered operating system. So, there are 4 layers normally in an operating system we could have upto 4 layers a kernel, then there can be some system programs then there could be development programs like compiler etcetera and then user application program, if you really want to build a capability based operating system right.

So, we need to assign certain privilege level for each of these programs or each of this object fitou will say. So, we will have a privilege level 0 for the kernel, privilege level 1, for the system software like device drivers etcetera privilege level 2 for these system software like compilers and the privilege level 3 development software like compilers and privilege level 3 for this. That means, if you look at a processor It basically works across 4 privilege level.

So, there will be a context switch from the kernel to a middle layer to another layer. So, the processor can work in one of the 4 privilege levels namely privilege level 0 privilege level 1 2 and 3. Now so, a task which is nothing but a process which is a program in execution as to basically execute in one of these privilege level.

(Refer Slide Time: 01:46)



INDIAN INSTITUTE OF TECHNOLOGY MADRAS

What is a task ?

Definition

Task is a unit of work performed by processor to :

- a) execute process
- b) serve operating system utility
- c) handle interrupt or exception

 Copyright © V — Information Security - IITask Switch 12/28 

And what you are trying to teach through this 4th assignment is how in the Intel platform how do you move from privilege level 0 to 1 1 to 2 2 to 3. And then there is always a reason for us to basically been one privilege level and try to access something else and some other privilege level. For example, simple hello word I want execute printer a simple you know matrix multiplication I want to do my lock right. So, when I am executing a c program I going that is a need for me to go and have a system call. I am going to have a system call basically wanting the operating system to satisfy some needs of this program. So, there is always a necessity for we to be at a higher level and switch down to a lower and come back to that higher level right. Right, then there are other needs like for example, les us take setting password right.

So, your updating your password right, in your unique system where does your password get stored? Slash etcetera slash password slash etcetera. Slash password is the file in slash etcetera directory is password file. That password that file can you write into that password file? No, but you are writing into that password file, because you say password and you enter a new password it is get written to that file. Though that file by principal is only editable by the root are the super user, you are able to know as normal user you are able to go and update that file and come back. Means even though I have

a very privileged object inside my operating system there is a need for a less privileged process to come and manipulate and do something, but with subject to certain rules and regulations. What is the rules rule and regulation a single rule and for updating your

password? You can go and change your password there, a your shadow password basically password not stored like that it basically store ash and stored.

So, you can go and change your ash of your line in that particular file. You can not go and change his line. So, I have a selected way I have a selective way of permitting you to go and change an object, which is set a very higher privilege level, do you understand this? Right, and So, the underline architecture should have certain hardware mechanism to allow you to do this right. So, this what we are trying cover in this task switching right. This understanding is quite necessary because tomorrow when you look at operating system then you will face this question which would asked and then you the answer is not just software. Answer is both software plus hardware together. If you want a really full prof mechanism for password updation then it is in my opinion it is going to be a careful mixture of hardware and software together. And as a part of this course we need to teach you what is that hardware component. Are you able to follow? Am I setting the stage correct ok.

Now So, all is about task I have a task this task is executing at privilege 0, I want execute it is privilege or demote it is privilege to 3 3 as lesser privilege then this something is working at 3 I want exculate privilege to 0 vice versa. So, that is what this entire assignment is about we are made it very short. So, that you know we are given you lot more template. So, that you can complete it within a week you spend this Saturday Sunday and you can finish it off very, very going to be very simple we are not putting lot of pressure on that, but please understand this. So, when I have a task I want to keep executing the task. And there is a need to switch from one task to another task which is call that context switch right. So, I stop this task and starting executing this task. And if this context switch is not done with enough amount of security then we may land up with

(Refer Slide Time: 06:16)



INDIAN INSTITUTE OF TECHNOLOGY MADRAS

What is Task Management and why it's important for information security ?

Task management
How hardware handles multiple tasks and protection among them

- a) Task execution
- b) Task switch(Context switch)

If task management is not made protected
access or corruption of secured data(belong to higher privilege user task) by lower privilege user task—**threat to information security**

 Information Security - IIT Madras 

Lot of issue in information security, this become potentially threat to information security right.

If you look at all the attack that have happened in the past many of them or because of this problems in switching between one function to another function or one task to another task. Now let us go and study this in great detail, how in Intel as put certain security mechanism for task switching. Now first and foremost let us understand that I have a task I am switching into another task. So, what is it that I need to save? And that is call the context of the task. The context we have already define what is the context of a process, it is the minimum amount of information that is necessary to restart that task from exactly the point where it is stop and that is what we mean by this mechanism. So, let us see, what is it.

(Refer Slide Time: 07:12)

INDIAN INSTITUTE OF TECHNOLOGY MADRAS

Task-state segment(TSS)

31	15	0
00 Flag Word Address		
12 LDT Segment Selector		
05		
10		
15		
20		
25		
30		
35		
40		
45		
50		
55		
60		
65		
70		
75		
80		
85		
90		
95		
100		
105		
110		
115		
120		
125		
130		
135		
140		
145		
150		
155		
160		
165		
170		
175		
180		
185		
190		
195		
200		
205		
210		
215		
220		
225		
230		
235		
240		
245		
250		
255		
260		
265		
270		
275		
280		
285		
290		
295		
300		
305		
310		
315		
320		
325		
330		
335		
340		
345		
350		
355		
360		
365		
370		
375		
380		
385		
390		
395		
400		
405		
410		
415		
420		
425		
430		
435		
440		
445		
450		
455		
460		
465		
470		
475		
480		
485		
490		
495		
500		
505		
510		
515		
520		
525		
530		
535		
540		
545		
550		
555		
560		
565		
570		
575		
580		
585		
590		
595		
600		
605		
610		
615		
620		
625		
630		
635		
640		
645		
650		
655		
660		
665		
670		
675		
680		
685		
690		
695		
700		
705		
710		
715		
720		
725		
730		
735		
740		
745		
750		
755		
760		
765		
770		
775		
780		
785		
790		
795		
800		
805		
810		
815		
820		
825		
830		
835		
840		
845		
850		
855		
860		
865		
870		
875		
880		
885		
890		
895		
900		
905		
910		
915		
920		
925		
930		
935		
940		
945		
950		
955		
960		
965		
970		
975		
980		
985		
990		
995		
1000		

- LDT segment selector : segment selector for the task LDT
- Segment selectors : ES, CS, SS, DS, FS, and GS registers prior to the task switch.
- EIP : instruction pointer prior to task switch
- CR3 : base physical address of the page directory to be used by the task

Computer V — Information Security — ITTask Switch

So, I will just quickly go into that what you call as So, this is the whatever you see here. This is the task state segment right, this task state segment basically as So, it as a place for all your general purposes suggested look at this index 40 etcetera.

It as a place for all your general purpose suggested ES CCS CDS CBS TSP VPSI IEDA right than it as a place for reflags because your flags are very important right. So, I might have done in add operation. And then I would have I would have done a compare operation. And then the root have been a context switch next time I want to jump on 0, the flag is very important for me to save, correct?

Why should we save the flag I would done a compare impression. Next time I say j inside. Before j inside and after the compare there was a context switch. Then I need to save the flags right. So, flags are to be saved. The instruction pointer of the next instruction to be executed that is saved here EIP, if you look at and I say told you your cr 3 is also save because every processor can have it is own past table.

So, here cr 3 is also saved. Then the content of all your GS FS ES SS CS CSS, all this are saved. And then I could have something called GDT that is global disrupter table I will have a local disrupter table for some of my local memory accesses, which is creating by the

operating system. That LDT segment selector is also saved, which will point you to the LDT where the segments are there. So, other than that please look that there are 3 task segments SS 0 SS 1 SS 2. Basically ESP 0 ESP 1 ESP 2 there are task pointers of

that. And those are basically created. Why it is created? Because when I when I am a program and suddenly I want to go to a task 0 right, task 0 it is a interrupt service routine right, now what stack will task 0 use.

It will not use this 3 stack, I do not want it to use. I am a privilege level 3 code I am executing and I am having an SSP ESP 3 stack, a stack with privilege level a stack whose segment is privilege level 3. I do not want my other program to use the same stack. Because when it comes out then some information of that stack can be leaked. Because if it is using the same stack; that means, what is going to happen it is going to write into your that privilege 3 data segment and then any other fellow can come and access this privilege 3 data segment, you are able to understand? So, if I am a privilege 3 and I am calling kernel level routine it also uses a same privilege 3 stack when that kernel level routine is executing there can be a context switch and another privilege 3 level code can start accessing and it can go on now and screw of the kernel level 3 data structure you are getting this?

So, when I when I am doing when I get an interrupt for example, when I get an interrupt right, and my interrupt service routine is a privilege level 0 it will start using the privilege level 0 stack and not the privilege level 3 stack. So, when a process go call has a system called or it has a trap the trap routine or the system call whatever that routine if it is going to execute at say at lower privilege level that will use the stack of the lower privilege, you are getting this, right? So, so that is why when you are setting up this task state segment you create a stack for privilege 0 privilege 1 and privilege 2, because in case that program that your executing which is privilege level k. Once you have a kernel level service or a system level service of a lower privilege level the stack that that service will use is not your stack, but this stack that is given here ok.

So, the entire task state segment is set up by whom? By the by the operating system, so, when you are spawning a task immediately I create one segment for you like this right. And then So, this segment whenever there is a context switch when you are moving out the latest value of all these things will be saved in the segment. So, there are 2 processes working, when he is working after he finishes it is his latest value will be stored in this task state segment. And we go the next fellow we load all the things from his task state segment and start executing. After he finishes say round robin he finishes his that state

will be stored in his task state segment, and he should reach state we will reload from his task state segment and start executing.

Like this so, this is basically what we call as the process control block from the operating system point of view, this TSS is basically called as a what? A process control block. This is an hardware image of a process control block, are you able to follow? Yes or no?

(Refer Slide Time: 12:58)

INDIAN INSTITUTE OF TECHNOLOGY MADRAS

Method 2 : Task gate descriptor

DPL - Descriptor Privilege Level
 P - Segment Present
 TYPE - Segment Type
 Reserved

- indirect reference to a task
- can be stored in GDT, LDT or IDT
- (TSS Segment Selector) —> TSS descriptor
- TSS descriptor access rule :
CPL/RPL calling Task gate selector \leq DPL of Task gate

Camelot V — Information Security :ITask Switch
12/28

So now, what happens is, So how do you switch from one task to another? So, there are 2 levels of things first is there something call a task gate right, the task gate is a part of your GDT or LDT or IDT task gate is a part of your global descriptor it is another descriptor like another segment descriptor. And what it will store there is a descriptor privilege level there DPL there is a present bit of course, that should be 1. And then there is a type 0 0 1 0 1, that essentially means this is a task gate. And then there is a selector. So, there is a selector here, this selector will tell you which task state segment you need to go right, now how will this selector look?

(Refer Slide Time: 13:54)

INDIAN INSTITUTE OF TECHNOLOGY MADRAS

TSS descriptor

- segment descriptor for TSS
- each TSS should have only one TSS descriptor(to ensure one busy flag per task)
- task can be dispatched only if TSS descriptor is **accessible**
i.e $CPL \leq DPL$ of TSS
- Busy flag (B) : whether the task is busy (task is currently running or is suspended)

TSS Descriptor

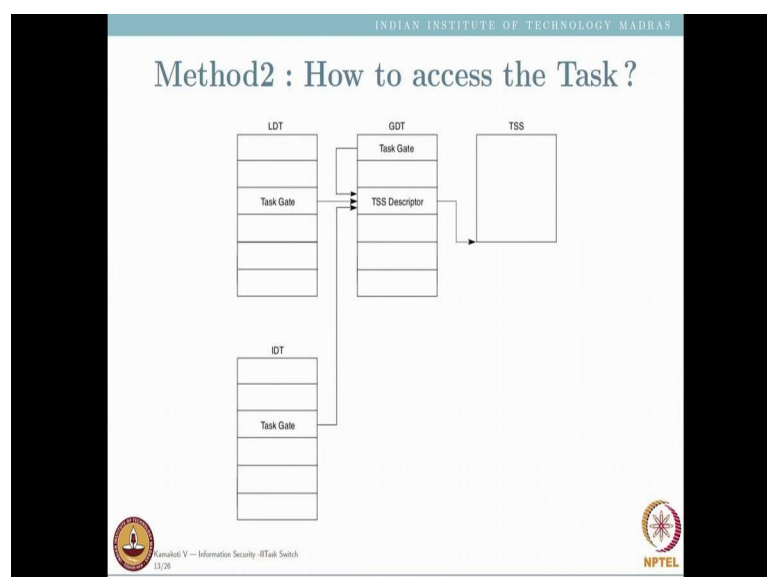
31		24 23 22 21 20 19										16 15 14 13 12 11										8 7		0	
Base 31:24		G	D	A	V	L	Limit 19:16				P	D	Type				Base 23:16		4						
31		16 15										8		0											
Base Address 15:00												Segment Limit 15:00										0			

AVL Available for use by system software
B Busy flag
BASE Segment Base Address
DPL Descriptor Privilege Level
G Granularity
LIMIT Segment Limit
P Segment Present
TYPE Segment Type

Amalath V — Information Security - IITask Switch

So, this a DSS descriptor this will tell you where is the base address, this is a normal you know segment selector right, it will tell you what is the base address of that TSS that 104 bits and then what is the limit then the type please note it should be 0 1 0 B 1. And then you have a DPL and this granularity and this is the limit. So, a TSS descriptor will be as same as the you know Normal descriptor.

(Refer Slide Time: 14:29)



So, this is how we access, now there is a task gate if I want to jump to another privilege level, I jump on this task gate right. This task gate will point a TSS descriptor. Because

this is a task gate right, this task gate will point a TSS descriptor. That TSS descriptor will point you a basic address of 104 bits.

(Refer Slide Time: 15:00)

INDIAN INSTITUTE OF TECHNOLOGY MADRAS

Method1 : Task-state segment(TSS)

10	16	0
100	100	100
101	101	101
102	102	102
103	103	103
104	104	104
105	105	105
106	106	106
107	107	107
108	108	108
109	109	109
110	110	110
111	111	111
112	112	112
113	113	113
114	114	114
115	115	115
116	116	116
117	117	117
118	118	118
119	119	119
120	120	120
121	121	121
122	122	122
123	123	123
124	124	124
125	125	125
126	126	126
127	127	127
128	128	128
129	129	129
130	130	130
131	131	131
132	132	132
133	133	133
134	134	134
135	135	135
136	136	136
137	137	137
138	138	138
139	139	139
140	140	140
141	141	141
142	142	142
143	143	143
144	144	144
145	145	145
146	146	146
147	147	147
148	148	148
149	149	149
150	150	150
151	151	151
152	152	152
153	153	153
154	154	154
155	155	155
156	156	156
157	157	157
158	158	158
159	159	159
160	160	160
161	161	161
162	162	162
163	163	163
164	164	164
165	165	165
166	166	166
167	167	167
168	168	168
169	169	169
170	170	170
171	171	171
172	172	172
173	173	173
174	174	174
175	175	175
176	176	176
177	177	177
178	178	178
179	179	179
180	180	180
181	181	181
182	182	182
183	183	183
184	184	184
185	185	185
186	186	186
187	187	187
188	188	188
189	189	189
190	190	190
191	191	191
192	192	192
193	193	193
194	194	194
195	195	195
196	196	196
197	197	197
198	198	198
199	199	199
200	200	200
201	201	201
202	202	202
203	203	203
204	204	204
205	205	205
206	206	206
207	207	207
208	208	208
209	209	209
210	210	210
211	211	211
212	212	212
213	213	213
214	214	214
215	215	215
216	216	216
217	217	217
218	218	218
219	219	219
220	220	220
221	221	221
222	222	222
223	223	223
224	224	224
225	225	225
226	226	226
227	227	227
228	228	228
229	229	229
230	230	230
231	231	231
232	232	232
233	233	233
234	234	234
235	235	235
236	236	236
237	237	237
238	238	238
239	239	239
240	240	240
241	241	241
242	242	242
243	243	243
244	244	244
245	245	245
246	246	246
247	247	247
248	248	248
249	249	249
250	250	250
251	251	251
252	252	252
253	253	253
254	254	254
255	255	255
256	256	256
257	257	257
258	258	258
259	259	259
260	260	260
261	261	261
262	262	262
263	263	263
264	264	264
265	265	265
266	266	266
267	267	267
268	268	268
269	269	269
270	270	270
271	271	271
272	272	272
273	273	273
274	274	274
275	275	275
276	276	276
277	277	277
278	278	278
279	279	279
280	280	280
281	281	281
282	282	282
283	283	283
284	284	284
285	285	285
286	286	286
287	287	287
288	288	288
289	289	289
290	290	290
291	291	291
292	292	292
293	293	293
294	294	294
295	295	295
296	296	296
297	297	297
298	298	298
299	299	299
300	300	300
301	301	301
302	302	302
303	303	303
304	304	304
305	305	305
306	306	306
307	307	307
308	308	308
309	309	309
310	310	310
311	311	311
312	312	312
313	313	313
314	314	314
315	315	315
316	316	316
317	317	317
318	318	318
319	319	319
320	320	320
321	321	321
322	322	322
323	323	323
324	324	324
325	325	325
326	326	326
327	327	327
328	328	328
329	329	329
330	330	330
331	331	331
332	332	332
333	333	333
334	334	334
335	335	335
336	336	336
337	337	337
338	338	338
339	339	339
340	340	340
341	341	341
342	342	342
343	343	343
344	344	344
345	345	345
346	346	346
347	347	347
348	348	348
349	349	349
350	350	350
351	351	351
352	352	352
353	353	353
354	354	354
355	355	355
356	356	356
357	357	357
358	358	358
359	359	359
360	360	360
361	361	361
362	362	362
363	363	363
364	364	364
365	365	365
366	366	366
367	367	367
368	368	368
369	369	369
370	370	370
371	371	371
372	372	372
373	373	373
374	374	374
375	375	375
376	376	376
377	377	377
378	378	378
379	379	379
380	380	380
381	381	381
382	382	382
383	383	383
384	384	384
385	385	385
386	386	386
387	387	387
388	388	388
389	389	389
390	390	390
391	391	391
392	392	392
393	393	393
394	394	394
395	395	395
396	396	396
397	397	397
398	398	398
399	399	399
400	400	400
401	401	401
402	402	402
403	403	403
404	404	404
405	405	405
406	406	406
407	407	407
408	408	408
409	409	409
410	410	410
411	411	411
412	412	412
413	413	413
414	414	414
415	415	415
416	416	416
417	417	417
418	418	418
419	419	419
420	420	420
421	421	421
422	422	422
423	423	423
424	424	424
425	425	425
426	426	426
427	427	427
428	428	428
429	429	429
430	430	430
431	431	431
432	432	432
433	433	433
434	434	434
435	435	435
436	436	436
437	437	437
438	438	438
439	439	439
440	440	440
441	441	441
442	442	442
443	443	443
444	444	444
445	445	445
446	446	446
447	447	447
448	448	448
449	449	449
450	450	450
451	451	451
452	452	452
453	453	453
454	454	454
455	455	455
456	4	

now come to the if that a task gate it will point TSS descriptor it come to a DSS descriptor from the TSS descriptor, it will basically upload your new task and you will

start executing from where that tip in the TSS. TSS as an EIP, you start executing with the privilege level of what that CS segment.

So, I go from 0 to 3 like this are 3 to 0 back, are you able to follow? Yes or no? So, so this is something like this 0 x 7 9, will store yeah TSS task switch. So, you just say 0 x 7

9. And that will point to a TSS segment selector let it be a 0 x 7 8. And the you go to 0 x 7 8 or whatever and you basically go to the TSS segment, upload everything and start executing. While doing this we also as to make sure that if I am an third I am I am there is a DPL right, there is a descript privilege level for this task gate and also for this task descriptor. All of them should be you know as the descriptor privilege level like the task gate should be have 3 means then 3 2 1 or 0 can jump through it. If a put it two; that means, a 3 level process cannot jump to this ok.

So, that is why I adjust the value of DPL of the task gate and task descriptor to basically tell who can come and jump to me, who can come use me? I have such that level of privilege. So, this is the way I can switch from one task to another task. And now let me introduce one very important concept, suppose what happens when there is interrupt? What happen when there is an interrupt? Immediately something is loaded on the stack and then you go to the interrupt service routine you your right, now when as a privilege 3 code and I have stack fault that also an trap right, that not cause any problem because whatever I am going push in a privilege level 0 stack. My privilege level 3 stack is what is getting effect there is stock fall privilege level 3 stack there is a privilege level 0 there is a another privilege level 0 stack.

So, privilege level 3 stack something happens, but I am going to only update the privilege level 0 stack, correct? But if a privilege level 0 code itself as a stack fault; that means, I am trying when there is a fault, I am trying to load when there is trap I am trying to load something and then calling the interrupt service routine. So, there are 3 step there is a trap that has occurred then I am loading something into that stack and then calling the interrupt routine. So, hardware does 3 step there. While I am trying to load into the stack itself I land up with another problem, then that is called a double fault. It is called a double fault, please understand. I am a process, I have a trap I encounter a trap, as a part of do solve servicing this trap what is should do? I do I push something into the stack and then call the interrupt

service routine, the trap service routine does something and then returns back, when I am trying to push something before even calling the

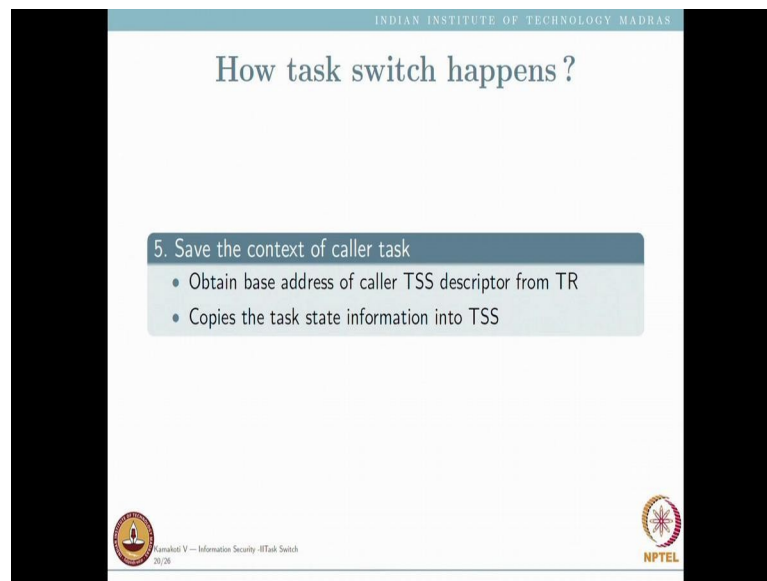
interrupt service routine after I have got a trap and before calling the interrupt service routine when I am trying to push something into a stack.

If there is another problem that is coming that is called a double fault, understand? So, what is a double fault? I have asked n times I asked in the interview for So many people last 17 year of my stay at IIT madras. Everybody says that when the interrupt service routine is executing, interrupt service routine lands up with another interrupt then it say double fault, no. When a interrupt service routine start executing it is another process by itself it creates a fault then it is it is fault it is not a double fault. I have an interrupt fault, I have fault when I am trying to process that fault before even calling the interrupt service routine I will land up with another fault that is called a double fault. There is one very good instant, a privilege level 0 stack there is a privilege level 0 code which is trying to execute an which has stack fault, and interrupt service routine is also privilege level 0 and they share the stack essentially then I land up with a stack fault.

When there is a double fault what I can do? I have to do a completely some other process as to come and handle. I can not I can not handle this fault like a function call. Because function call will need a stack and my stack is ruined right. So, I have to go and have a trap I have to go and call entirely different process which use entirely different stack to handle this, are you able to get this? So, that is one of the reason in IDT right, in the interrupt descriptor table I will have lot of interrupt gate and trap gates, but for at least one fault I need to have a task gate. Which one instead if this interrupt comes I need to completely do a task switching right, because I have to do completely because, and one of that prominent cases for that you could have other devise driver also at as a part of it, but one of the prominent cases for this is the double fault. Because double fault as of today when we perceive it is because the stack got corrupted and if the stack got corrupted I cannot use a interrupt or a trap gate to solve it.

Because interrupt and trap gate heavily realised upon the this stack again. So, that is a reason I will have a double fault which will take you to entire different process. And the process will start you know servicing this fault. So, that is why we try and use it task gate in your in your IDT, are you able to follow? Please understand and appreciate what is a double fault, it is not a interrupt service routine creating a another fault. It is not double fault there is a certain different here I hope you followed.

(Refer Slide Time : 23:03).



So, when I do a call what is a different between a jump and the call I have to return back. When I do a call on as a task switch I call some other process; that means, I some completely come out and he execute, after he finish and he return I have to come back here. Please note that in this in this there is a previous task link 0, if you look at 0 there is a previous task link. This will point to the previous task state segment.

Student: Sir.

Yes.

Student: Note different 2 task have the same privilege level they still different rage so.

They are still different they.

Student: (Refer Time: 23:42) they are like inter process protection problem here.

So, that we will that a good question, but as of now yes there can be inter falls of protection. How would we handle that I can we will just basically see after this step, but now can you see that I if do a call my previous link will be stored here. So, after you return automatically this previous task link will go there and it will load the context of it. So, so when I do nested calls I move from one task to another start

task as a nested call please note that the previous task link will be updated. So, the I could come back to the previous task. So, that is also very important.

(Refer Slide Time: 24:28)

INDIAN INSTITUTE OF TECHNOLOGY MADRAS

Method 1 : Task register

The diagram illustrates the Task Register Method 1. It shows a Task Register with three fields: Selector, Base Address, and Segment Limit. The Selector points to a TSS Descriptor in a GDT table. The Base Address and Segment Limit are used to calculate the Visible Portion (Base Address + Segment Limit) and the Invisible Portion (Base Address). The TSS Descriptor points to a TSS structure, which is divided into a Visible Portion and an Invisible Portion. The Visible Portion is used for the Base Address and Segment Limit calculation.

- Two components :
 - Visible** /software controlled part - 16-bit segment selector
 - Invisible** /processor controlled part- 32 bit base,16 bit limit and descriptor attributes for TSS descriptor(caching purpose)
- Instructions :
 - LTR (Load task register) : Visible and invisible portion from TSS
 - STR (store task register) : Visible portion into GPR or memory

© 2008 NPTEL

So, this is 1. So, with this amount of data now let us go and look at one more for this option. One more thing is where is call gate it is not there I will just basically explain.

So, another thing is one of the entries that you can have in your descriptor table GDT LDT is set of a call gate. In the call gate what you have is that you basically say that you give a code segment and then index within that code segment right, you give a code segment. Then index between a code segment now we say privilege level 3 we can call this if we calls this then what we can do we can go and execute that code from that index only right, followed so and but that code can be in whatever lesser privilege level like. So, this helps in the context of a password. So, so I want to change the password. So, I type password this will point to me a process through a call gate mechanism, and I go and execute that process there right, but I am unable to execute that process of a higher privilege. And which will and that alone will execute I can not have an influence on that process it will execute and return back right.

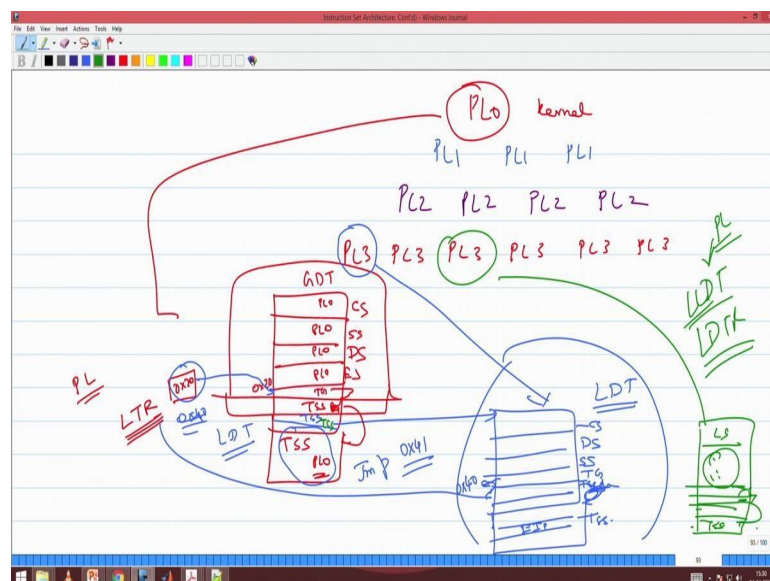
So, essentially I am saying I am privilege level 3 I am poor he is privilege 0 etcetera, password I can even touch him, but you are a nice man. So, please go and deposit that you tell the call gate right, the call gate will gate some routine which will go and deposit

and come. So, the call gate it will tell me or vice versa the call gate call gate, not tooth brush. So, call gate, call gate. The call gate will tell me that hey you want to do this action you can do. You tell me I will do you will not, but you can tell me, and I will do it

for you. So, what you tell him a this is my new password go and save it. This call gate you tell the call gate through the call gate and that fellow take your data (Refer Time: 26:48) and give it back.

So, call gate is another mechanism where a privilege level 3 code, can go and access a privilege level 0 segment through a privilege level 0 code which is accessed through the call gate. So, this gives without doing all this task switching is essentially you see it is a very shitty thing correct? I have to do So many things there to get this task switch. I have to go to one task gate then one TSS descriptor, then to TSS and then from there load everything here, store everything currently load the new thing then start with one [FL] will over. One full, one year will get over. So, that is a very heavily you know intensive compute intensive task. So, call gate gives you an ex very good advantage of you know enhancing or enhancing your privilege level. Is it fine? So, you will also implement call gate here ok.

(Refer Slide Time: 27:51)



Now, I will go to this (Refer Time: 27:51). In the lab say please now note, now there is a PL 0 kernel I will I will do one small story here completely like how we jump across. Then there are blue colour PL 1, PL 1, PL 1. Then there are some PL twos then there are lot of PL

threes PL 3 PL 3 PL 3. Now when PL 0 start executing, now there is a GDT, in which there are some code segments task segment blah blah blah, these are all privilege

level 0 segment. These are all privilege level 0 segments and then there is one task state segment for the privilege level 0 code.

So, the GDT ends here this is the GDT then there is a task state segment. So, there is a task gate here, which points to a TSS descriptor. Which in turn point to this all these are here. Now what happen is when the kernel comes it start executing right, it also makes there is execution call there is a instruction call LTR which is a privilege instruction. And this LTR it will store the index of this task gate let we store at 0 x 2 0 or something I will store 0 x 2 0 here. This is store the index of the task gate so now, PL 0 code is executing, each context available in this TSS. Because the task gate points to TSS descriptor which is in the GDT and that will point to this TSS. And this (Refer Time: 30:15) ok.

Now, I want to go let us say I want to go to a PL 3 code right, are you able to follow what I am saying? So, immediately what this GDT this fellow will do he will set up an LDT for this. And then in the LDT this is the local descriptor table, he will create the CS DS

SS right, and then he will put one task gate one TSS descriptor. And then another TSS forever in same memory this will point to this for PL 3. This PL 3 process and then he will have a he will have a task gate. So, you will do this and this fellow will jump on this task gate. Let us say this is 0 x 40 something he will jump on this 0 x 41. This is a LDT right? He will jump on this task gate. In this task gate you would have initialized the EIP here, and it will start executing from the tip point you are getting this? Yes.

Student: (Refer Time: 31:42)

Student: we can not have TSS descriptor LDT.

(Refer time: 31:50) TG, only you can have right, you can not have TSS descriptor [FL] let that TSS ok.

But TGA can TG can only TSS. So, this TG will be pointing to this and the TSS will be somewhere in the value. Now this fellow will be executing. And then he wants to come back, what he just? He just again say jump back or return. When it return so, when I do this jump your LTR will now point to this TG. Now your LTR will become 0 x 40 when this is executing. When it just black a return become lx 0 20. When I go from this process to this process I will save all the details of this fellow, and then

load all the details from this TSS (Refer Time: 32:52) right.

Now, please note that this is also in privilege level 0, this is also in privilege level 0. The PL 3 process which I am is quant this process namely cannot go and adjust the value of this LDT, because this is the it can use the value also (Refer Time: 33:13). So, essentially it can only access the TSS SS the etcetera, that only the descriptor that are stored here those memory only it can access. It cannot access anything more than that. Because every memory access is governed through a descriptor and that descriptor has a base and limit.

So, the privilege 3 process will only use this and it that is all. And TSS also will have a pointer past table you can use only those page pages, you cannot it cannot go and touch any part of, for example, it cannot touch this GDT because none of the descriptors in the LDT right, points to the GDT include the GDT. So, I can not go and change anything as a PL 3 process I can not go and anything in the GDT. Because it should have a segment descriptor here with privilege level 3 which should allow which includes the address where GDT is store and that will not (Refer Time: 34:07). So, I can not go and change the GDT, got this? Are you able to follow? Yes or no yes? Ok.

Now, what happens? Now I will come back here again now another PL 3 we want to start, let us take green fellow PL 3. Again this fellow will put another LDT, another LDT entry again there will CS DS (Refer Time: 34:34) there will be a TG (Refer Time: 34:35) then there will be a TSS. And there will be a TSS descriptor for that also imaging. So, when I count this I will just start executing with this EIP in this TSS. And I will execute and again come back. Now this process can touch only the data segments that are available in this LDT. Because all here there are PL 0, all here are only PL 3 it cannot access this because the LDT are registered can be changed only by an ILDT instruction which is already privileged. So, I can not go and adjust my LDT. So, the LDT is basically given by the kernel, I can use only the segments that are available and I adjust in such a way that these 2 (Refer Time: 35:24).

So, by using an LDT a privilege level 3 for both of privilege level 3 process, but this can not access that fellow memory and that fellow cannot access this fellow memory yes.

Student: what about (Refer Time: 35:35).

Each will have its own stack here this access this access. So, any memory I can touch only I can use I can use only the descriptor that is given the LDT. I can not use any of the descriptor in GDT. Why because there are GDT why because there are privilege level 0. I can use only the descriptor in my LDT. And I can not change my LDT. I want change my LDT go back to the operating. So, this type of mechanism gives obsolete isolation between 2 processes that are running, correct? So, this is this is so, then then when I am move out again I save everything in this task state segment. And then I again load everything from this you know the PL 0 task state segment, again I go. Now I want to restart this again I jump to this and then call this and then go back. So, are you able to follow see how I move from this. So, a notion of a LDT and that the LDT can be privilege and your LDTR is privilege registered.

So, thus gives us way by which I could have inter process protection between several processes of privileged level three. So, this is another thing that you need to keep in mind. So, your GDT and there is a GDT for the entire system there is a per process LDT. And the per process LTD also is created by the operating system and it is put in area of maximum privilege level. So, that nobody can change it. Are you able to follow? Any doubts? So, with this I think all my lecture with respect to lab is over. So, you have any doubts you can ask me.