Lecture 4

Tasks in PM

Task
An 80386 task is an entity that possesses unique **task state** and individual **virtual address** space.

From the point of view of an operating system tasks are being assigned to these “jobs”, “applications”, “threads”, etc. that require individual address spaces and unique task state segments. There is no requirement from 80386 processor to assign tasks to any of thread and application. Multitasking mechanisms of 80386 processor and operating systems are not the same functionalities. The 80386 hardware multitasking constitutes only a comfortable backup for OS task management.

Task state
By task state the **Task State Segment** (TSS) is understood. TSS is the area of memory that holds the task's registers state and other information.

Virtual address space
There is the possibility for each task to have its own LDT holding descriptors that define the task's individual address space.

Task State Segment (see Fig. 4.1)
It is a separated memory area holding the following information: the state of task's registers, stack segment selectors for all privilege levels, selector for LDT, I/O ports permission bitmap.

Tasks in PM

TSS

The minimum size of TSS is 104 bytes. With such size it holds the state of task's registers. Permission bitmap is not included into TSS.

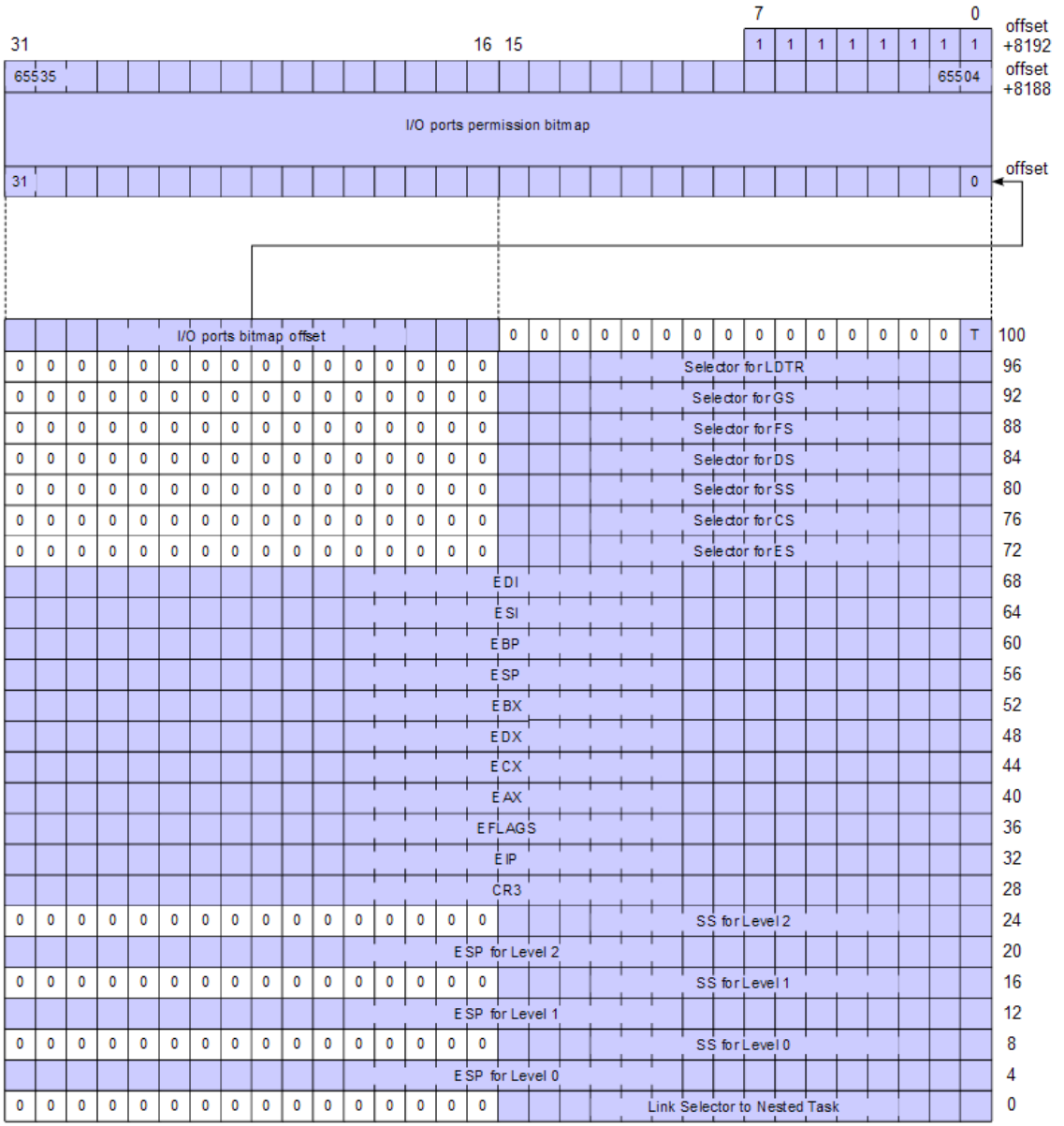
I/O ports permission bitmap – each bit with index *n* holds permission for access to one (with index *n*) of 65536 one-byte wide ports (0 – no access, 1 – access allowed).

Bit T is used for debug purposes and when set makes task trapping directly upon entry.

The location and size of Task State Segment is defined with the aid of the **Task Descriptor**.

TSS descriptors are allowed only in GDT.

Fig. 4.1 Task State Segment



TSS Descriptor
The form of TSS Descriptor is presented in Fig. 4.2.

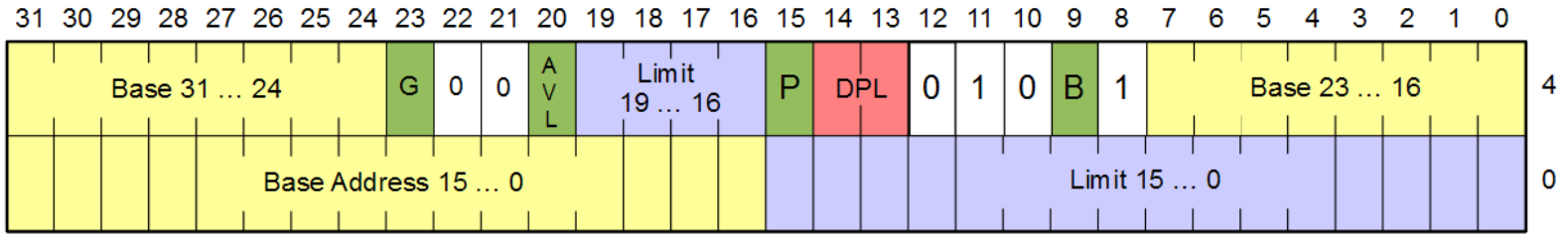


Fig. 4.2 Task State Segment Descriptor

- **Base address**: linear address of the first byte of TSS,
- **Limit**: maximum offset within TSS,
- **DPL**: privilege of TSS descriptor,
- **G**: granularity bit required for TSS size interpretation (0 – byte granularity, 1 – 4 kB granularity),
- **P** (present): indicates whether TSS is present in physical memory (P=1) or not (P=0),
- **AVL**: for operating system use,
- **B** (busy flag): indicates whether task is currently busy (B=1) or not (B=0). A busy task is the one that is running or is suspended.

Tasks in PM

Task transfer

The task transfer or **task-switching** in 80386 processors is realized with ordinary instructions: intersegment JMP, intersegment CALL, INT *n* or IRET. A task switch is performed by specifying the TSS selector or a **task gate** in the destination field of instruction.

The tasks involved in task switching are called: *outgoing* and *incoming*. The TSS of a current task is specified by its descriptor kept in **task register** TR register. Task register can be loaded with use of LTR instruction. The TSS of incoming task is indicated by the argument of a task switching instruction.

The 80386 saves the state of outgoing (current) task in TSS indicated by TR register. Then it uses the TSS of incoming task as the source of new registers' values.

Items saved in TSS of *outgoing* task:
- eight 32-bit general registers,
- six segment registers,
- EFLAG register.

Items taken from TSS of *incoming* task:
- eight 32-bit general registers,
- six segment registers,
- EFLAG register,
- LDTR contents to establish task's virtual address space,
- CR3 to establish task's page directory,
- T-bit for debugging purposes.

Task Gate

A task gate descriptor holds the selector of a TSS descriptor. Task gates can be hold in GDT, LDT and IDT.

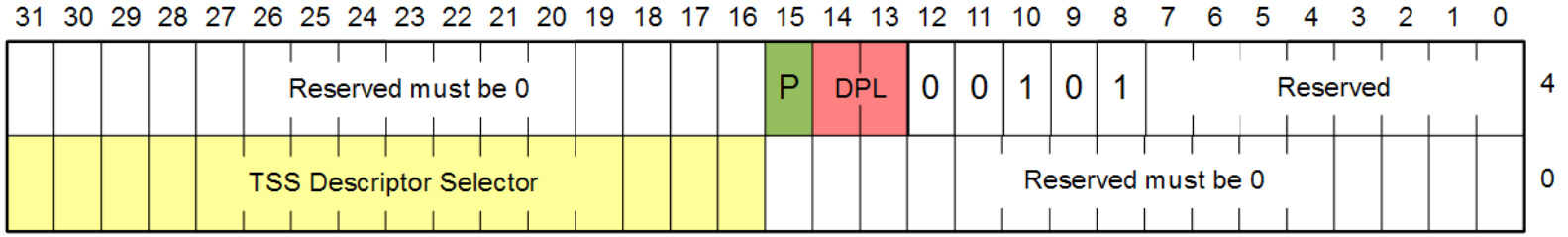


Fig. 4.5 Task Gate Descriptor

- **TSS Descriptor Selector**: holds the selector of descriptor for TSS,
- **P** (present): indicates whether TSS is present in physical memory (P=1) or not (P=0),
- **DPL**: the privilege level of the task gate.

Tasks in PM

Transfer through Task State Segments
There is the possibility to transfer control via a TSS directly with providing the selector of its descriptor as an argument of a transfer instruction. It should be noticed that TSS descriptors can be hold only in GDT what may cause some limitations.

Access checks:
- maximum of CPL and RPL must be <= DPL of TSS descriptor,
- TSS must be present,
- EIP must be within limits of TSS code segment.

Transfer through Task Gates
The advantages of using Task Gates are as follows: Task Gates can be hold in GDT, LDT or IDT, one TSS can have several gates making task switch possible in several ways (interrupt, direct call, etc.), they provide lower privileged software with task switching capabilities even of TSS with low DPLs.

Access checks:
- maximum of CPL and RPL must be <= **Task Gate DPL**,
- task gate must be present,
- TSS must be present,
- EIP must be within limits of TSS code segment.

Redirection via Call Gates

Call Gates

Redirection via a call gate is performed with intersegment CALL instruction. It should be noted that call gates enable control transfer between different privilege levels. However there is no possibility of redirection into lower privilege level. Such a CALL instruction can lead either to another code segment at the same or higher privilege level. Hence call gates constitute comfortable mechanisms for calling operating systems service routines. Call gates can be hold in GDT and LDT. In Figure 4.6 the structure of the call gate descriptor is presented.

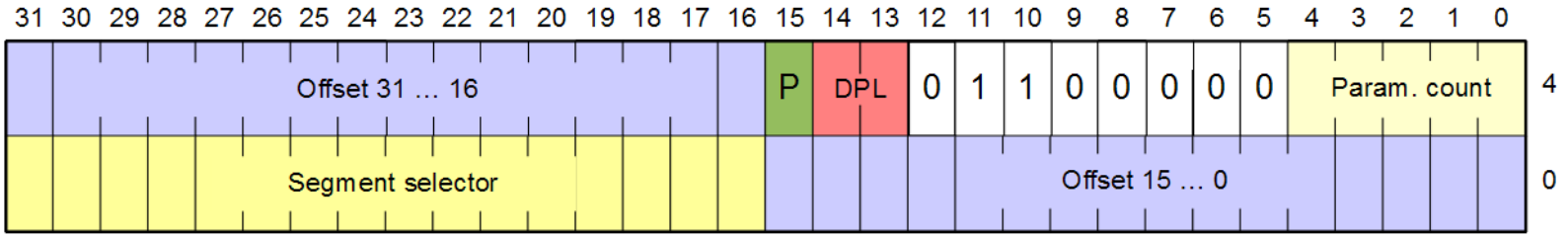


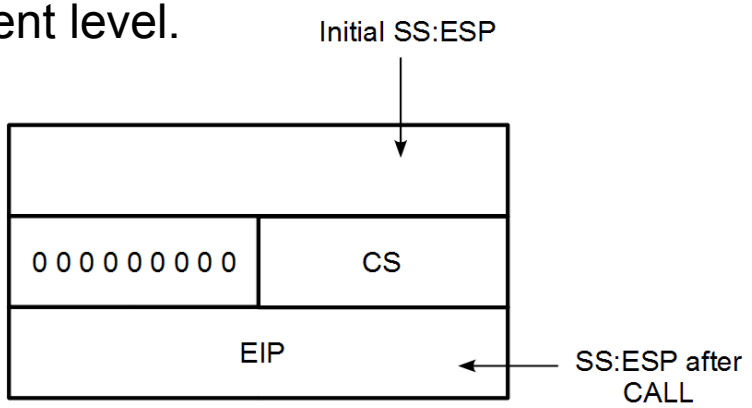
Fig. 4.6 Call Gate Descriptor

- **Segment selector**: segment selector of the destination code segment,
- **Offset**: offset within code segment of routine to be called,
- **DPL**: the privilege level of call gate descriptor,
- **P**: indicates whether gate is valid (P=1) or not (P=0),
- **Param. count**: indicates the number of 32-bit parameters required by called routine.

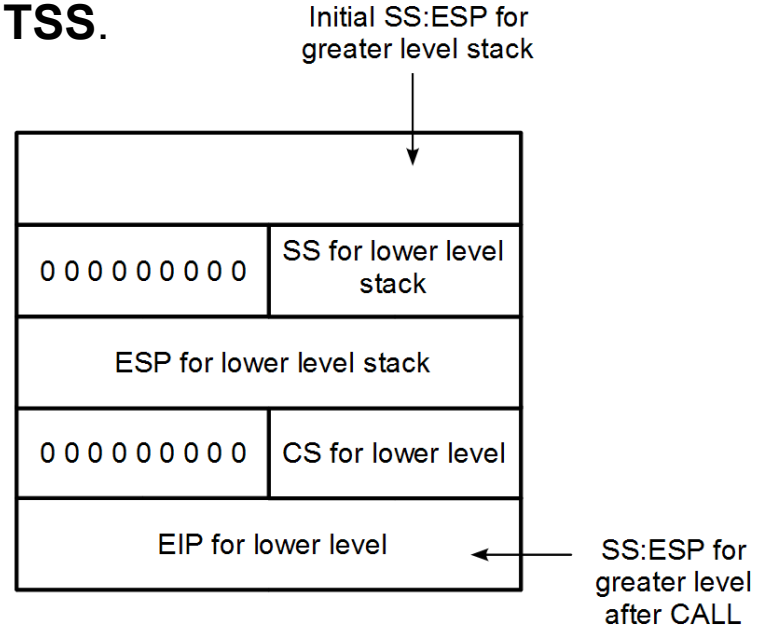
Redirection via Call Gates

Redirection via Call Gates without Parameter Autocopy

When redirection is performed at the same privilege level the CS and EIP (return address) are pushed on the stack of a current level.



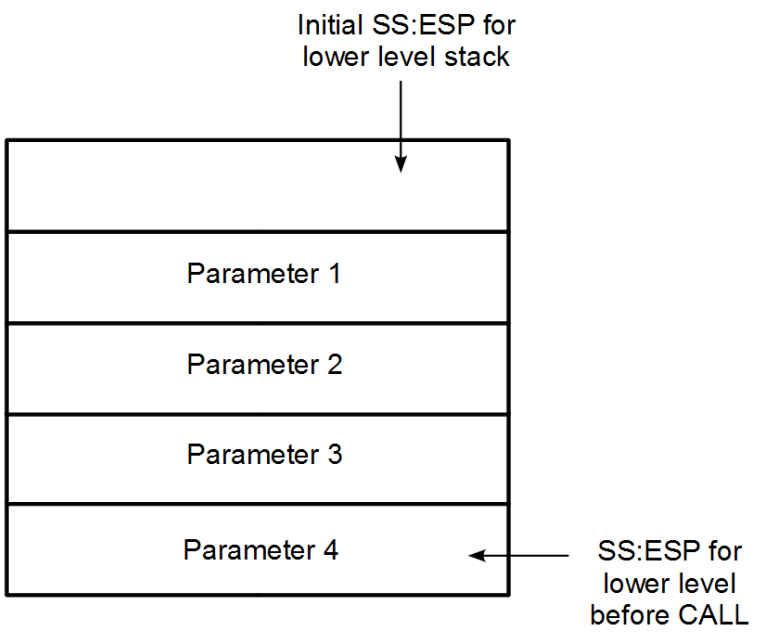
When a call gate redirects control to greater privilege level both return address and SS and ESP are pushed on the stack of a greater privilege level. **The SS:ESP of a greater privilege stack is taken from current TSS**.

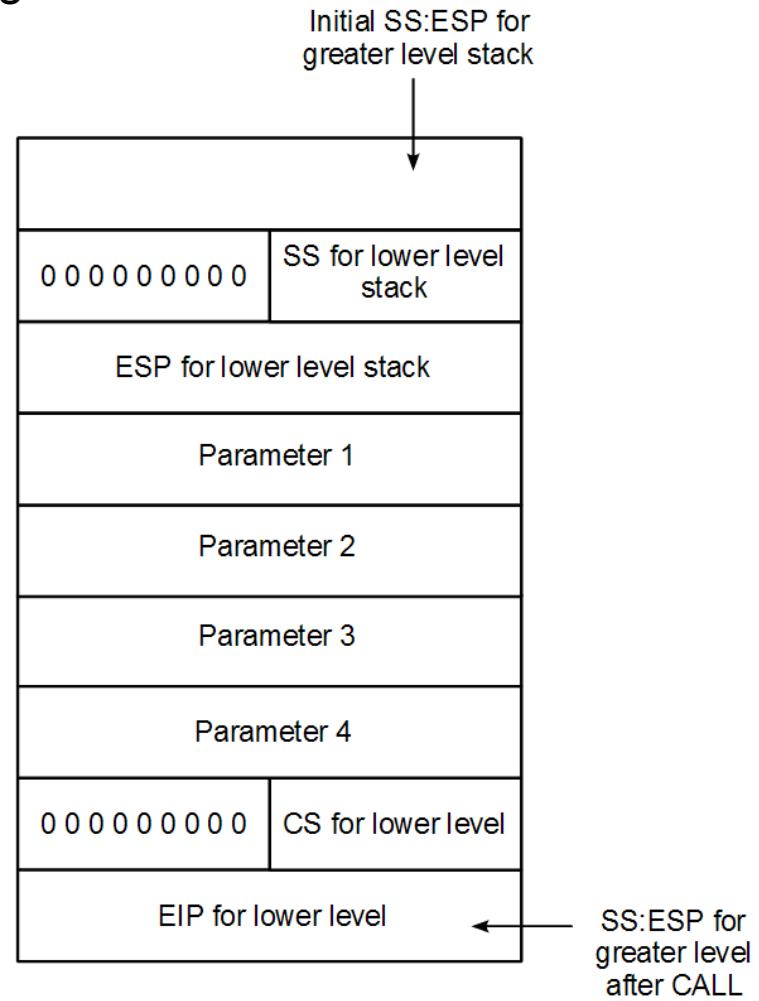


Redirection via Call Gates

Redirection via Call Gates with Parameter Autocopy

The calling procedure must place the calling parameters on its stack before performing an intersegment CALL. If a call gate transfers control to a greater privilege level there is the parameter autocopy option available. Simply the number of 32-bit values indicated by Param. count field is automatically copied from lower level to greater level stack.





In this case when returning from called procedure RET *n* instruction must be used where *n* is the length in bytes of calling parameters.

Redirection via Call Gates

Access checks

During transfer redirection via Call Gates the following (excerpt) access checks are performed:

- maximum of RPL and CPL must be <= DPL of a call gate,
- call gate must be present,
- call gate segment cannot be NULL,
- call gate code segment selector must be within limits of descriptor table,
- code segment descriptor must indicate a code segment,
- code segment must be present,
- DPL of code segment must be <= CPL.

Hints for User/Supervisor Application

Initially dispatching a user-level program

Initially PM application starts in Real Mode and next it is switched into Protected Mode operating at supervisor level. The user level program can be triggered in two ways: with task switching mechanism, user control redirection functionality.

While considering a control redirection method it should be noted that there is no possibility to transfer control to an user application using CALL due to privilege levels dependencies (more privileged code cannot initiate transfer to less privileged code). In such case we pretend that the supervisor code was called by the user code, properly prepare supervisor level stack (see previous slides) and RETurn to user code.