Teórica 7

Tareas

Unidad de trabajo que un procesador puede despachar, ejecutar, y detener a voluntad, bajo la forma de:

- La instancia de un programa (o, expresado en términos del lenguaje de teoría de Sistemas Operativos, proceso, o thread).
- Un handler de interrupción, o excepción.
- Un servicio del S.O. (por ejemplo en Linux, cualquiera de las Syscall).

Definiciones:

- **Espacio de ejecución:** Es el conjunto de secciones de código, datos, y pila que componen la tarea.
- **Contexto de ejecución:** Es el conjunto de valores de los registros internos del procesador en cada momento.
- **Espacio de Contexto de ejecución:** Se compone de un bloque de memoria en el que el S.O. almacenará el contexto completo de ejecución del procesador.

Context switch

> Salvar y restaurar el estado computacional o contexto de dos procesos o threads cuando se suspende la ejecución del primero (se salva su contexto) para pasar a ejecutar el segundo (se restaura su contexto). Puede incluir, o no, el resguardo y restauración del espacio de memoria (procesos o threads).

Primeras conclusiones
Las tareas en un computador no operan simultáneamente, sino serialmente pero conmutando de una a otra a gran velocidad. Esto significa que se producen cientos o miles de context switches por segundo. Nuestros sentidos no captan la intermitencia de cada tarea, creándose una *sensación de simultaneidad*.

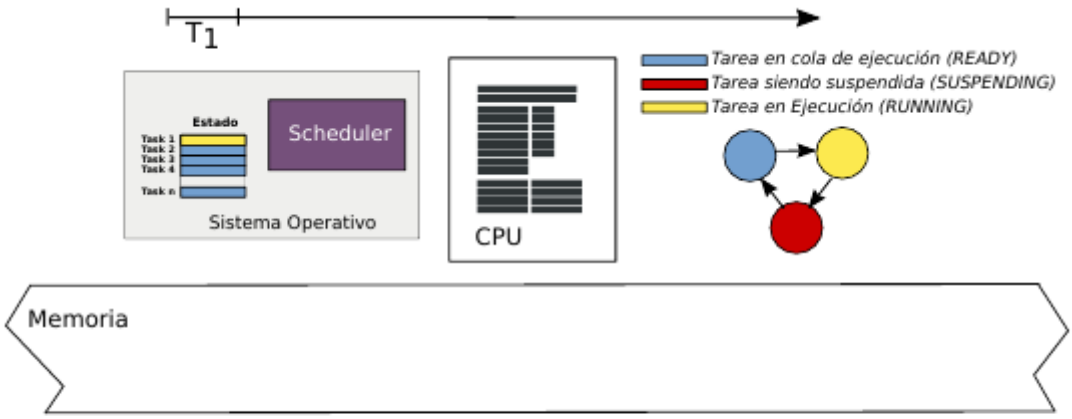
Entra en juego el Sistema Operativo

En particular, el Sistema Operativo tiene un módulo de software llamado **scheduler**, que trabaja con una lista de tareas a ejecutar.

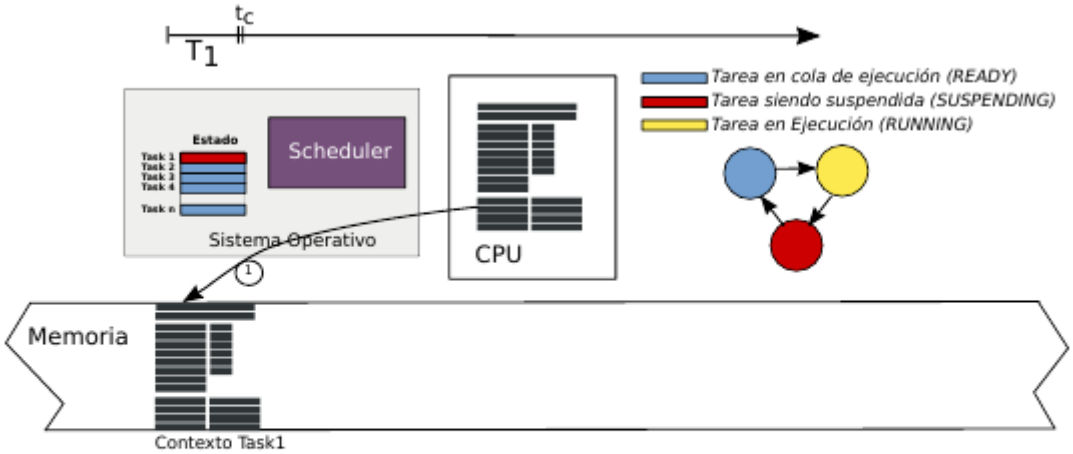
- El scheduler define un intervalo de tiempo llamado time frame, el que a su vez con ayuda de un temporizador es dividido en intervalos mas pequeños, que se convierten en la unidad de tiempo.
- El arte de manejo de prioridades consiste entonces en *asignarle a cada tarea un porcentaje del time frame*, medido en una cantidad de intervalos unitarios. **Es decir, le asigna a cada tarea una prioridad.**

- Cada tarea tiene así unos milisegundos para progresar, expirados los cuales suspende una tarea y despacha para su ejecución la siguiente de la lista.

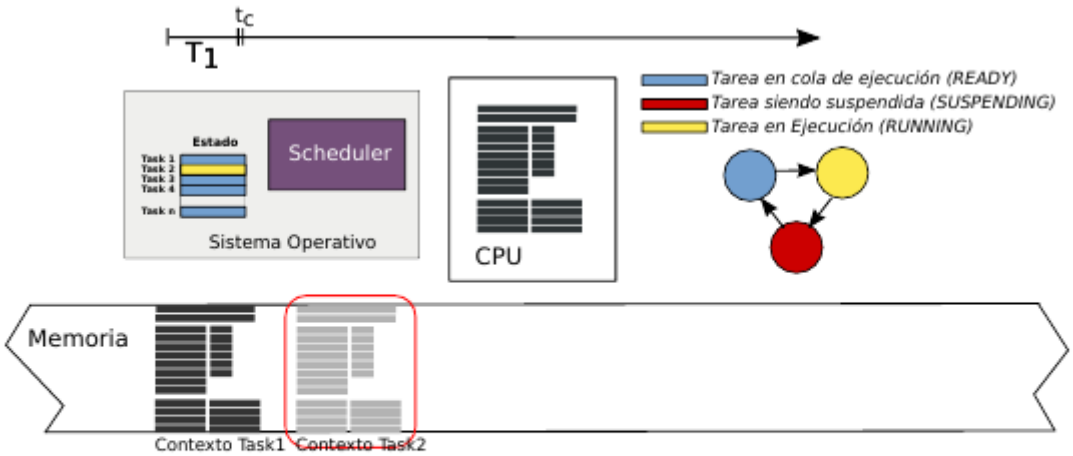
Ejemplo



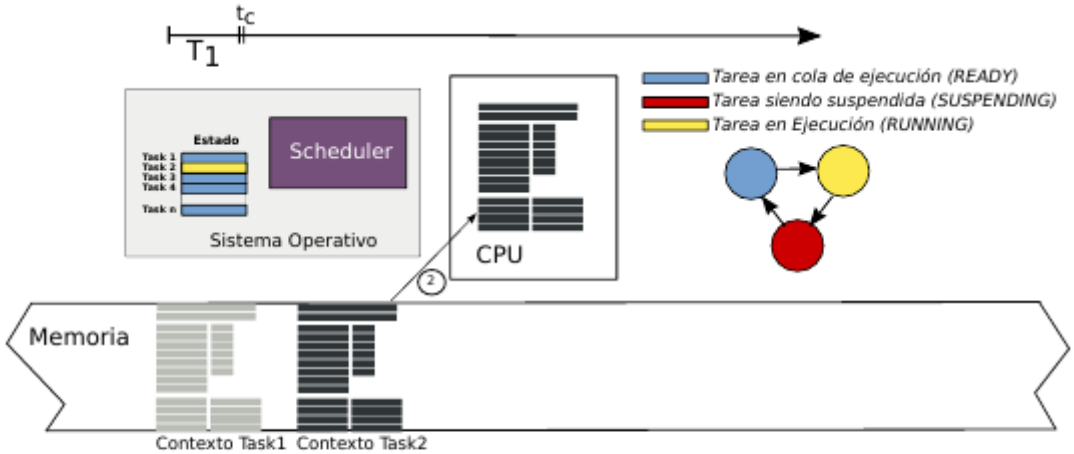
Durante un tiempo T1, el procesador ejecuta la tarea *Task1* de la lista que maneja el ***scheduler***.
El tiempo T1, que se le asignó para ejecución es medida de la prioridad de esta tarea.



Una vez expirado T1, el ***scheduler*** toma el control del sistema y como la tarea *Task1* no finalizó, la suspende, asignándole estado **SUSPENDING**.
Resguarda el contexto de *Task1* en un área de **memoria del kernel** (Flujo 1).
Contexto: Valor de los registros del procesador al momento de suspender la tarea.

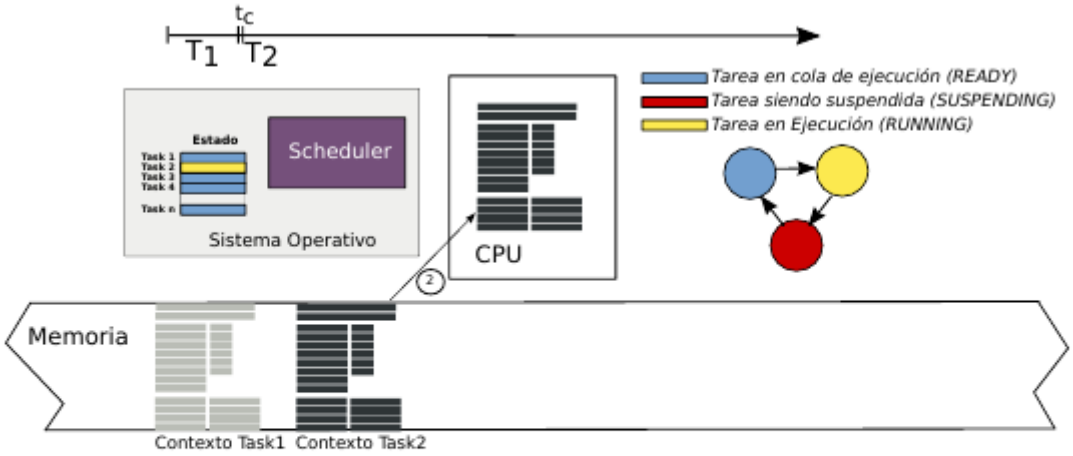


A partir del inicio de la transferencia del contexto de *Task1* a la **memoria del kernel** inicia un intervalo que denominaremos t_c (tiempo de conmutación). Seguidamente el ***scheduler*** busca la siguiente tarea en la lista de tareas en ejecución (en el gráfico *Task2*), identifica la dirección de memoria de kernel en donde está almacenado su contexto, y marca a la tarea RUNNING.

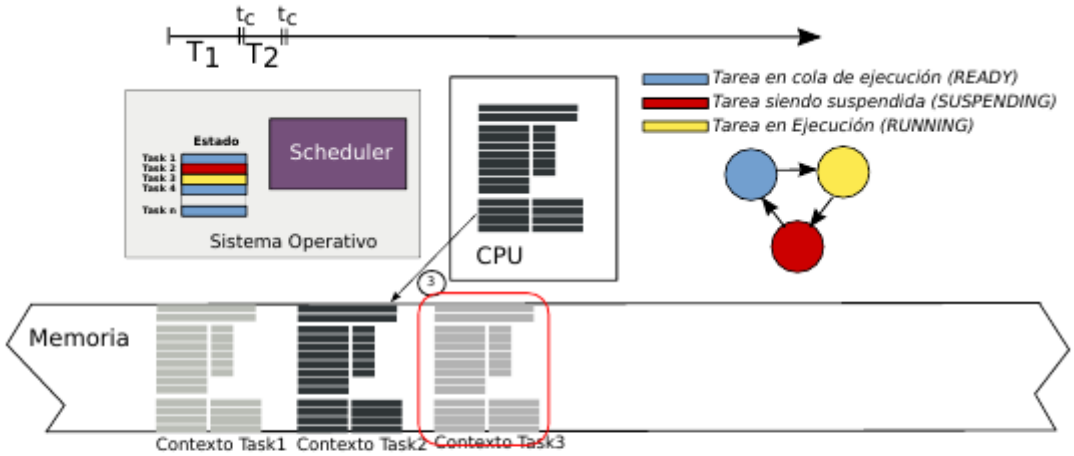


Se carga en el procesador el contexto de la tarea *Task2* (Flujo 2).
En el siguiente ciclo de clock, se reasume la ejecución de *Task2*, en la instrucción siguiente a la última ejecutada antes de ser suspendida.
Si *Task2* fue recién insertada en la lista de ejecución, entonces el procesador ejecutará su primer instrucción.
Llegamos al final del intervalo que denominamos t_c.

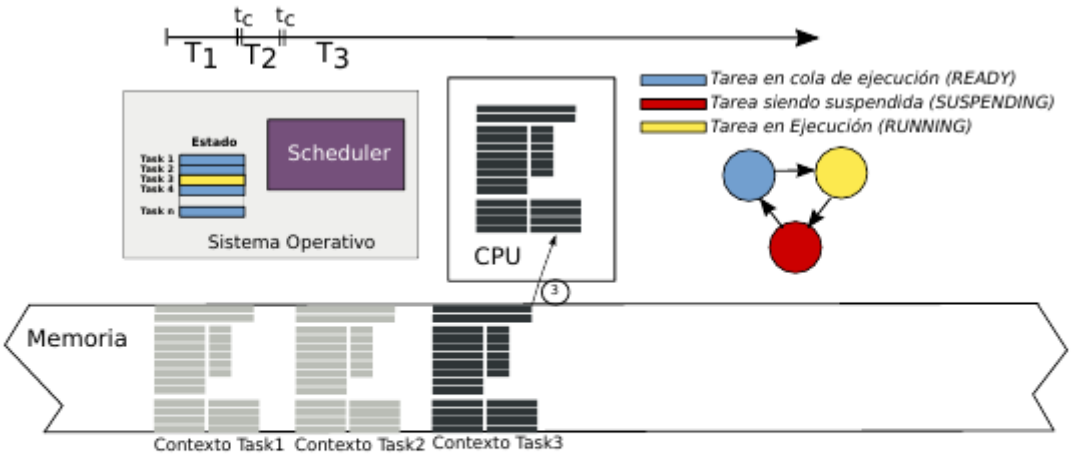
Hay un pequeño porcentaje del time frame que esta destinado no a las tareas, sino al context switch. Entonces hay que optimizarlo para que ocupe el menor tiempo posible.



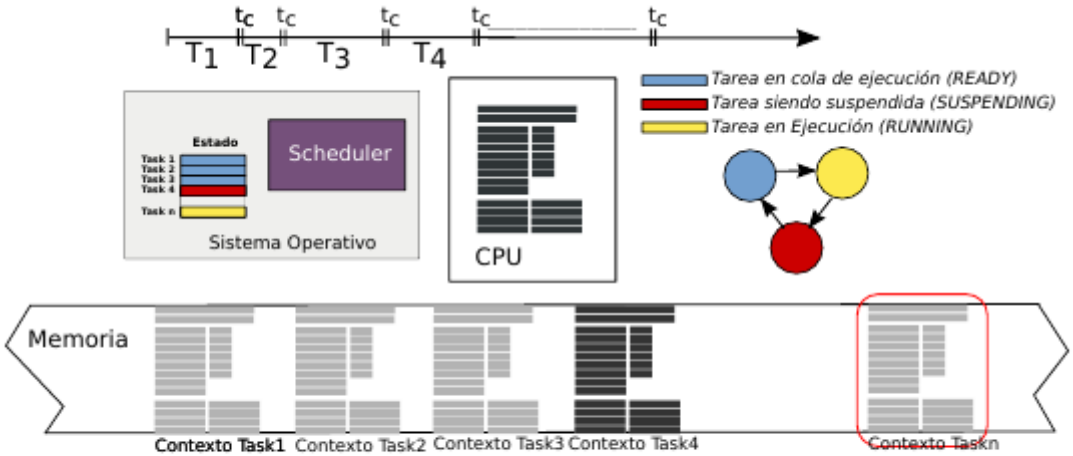
El procesador ejecuta *Task2* durante el tiempo T2.
Notar que T2 != T1 ya que el Sistema Operativo asigna prioridades diferentes a cada tarea.
Nuevamente el ***scheduler*** tomará el control cuando expire el tiempo T2.



El Sistema Operativo pone *Task2* **SUSPENDING** ya que su tiempo de ejecución expiró.
En el flujo 3, se ve el resguardo del contexto. Por otra parte se busca el identificador de la tarea siguiente en la lista.
Una vez ubicado, se identifica el puntero a la dirección de memoria en donde comienza el contexto de la siguiente tarea.



Identificada la tarea siguiente, se carga su contexto en el procesador.
Observar que el tiempo t_c es siempre el mismo independientemente de las tareas.
Finalizado este proceso en el ciclo de clock siguiente el procesador estará ejecutando la tarea *Task3*.
Esta continuará durante un tiempo T3.



Finalmente llega el turno de la última tarea en la lista.
Una vez completada esta el ***scheduler*** volverá al inicio de la lista y reasumirá la tarea *Task1*.

Modelo de Programación de Sistemas

- El modelo de programación de Aplicaciones de una CPU tiene una visión restringida del sistema en su conjunto.
- El modelo de programación de Sistemas, incluye un conjunto de recursos de hardware que amplía la capacidad de la CPU de resolver funciones que le permitan a un Sistema Operativo proveer los servicios y el entorno de programación que “ven” las aplicaciones, de manera mucho mas veloz y eficiente.
- De hecho, el manejo de las excepciones y de las interrupciones de hardware que hemos visto anteriormente es claramente el terreno del Sistema Operativo.
- Y hay muchas mas funciones para proveer a un Sistema Operativo desde el hardware que no son necesarias para las aplicaciones.

Scheduling de Tareas

Cualquier Sistema Operativo posee un módulo llamado scheduler, que le permite administrar la ejecución de tareas / procesos.
Podemos considerar al scheduler dividido en dos capas:

- Una de alto nivel en la que se implementan una o mas políticas para manejar el cambio de tareas, y que se traducen en diferentes algoritmos de scheduling.
- Una de bajo nivel en la que se realiza el *context switch*.

En este punto nos vamos a concentrar en el módulo de *context switch*, dejando para mas adelante los algoritmos de scheduling.
El *context switch* como veremos es físicamente dependiente del procesador y se escribe en assembler.

Capa de bajo nivel: Context Switch

Un Context Switch genérico debe llevar a cabo las siguientes acciones:

1. Resguardar los registros core y cualquier otro del modelo de programador de aplicaciones en un área de memoria accesible solo para el kernel.
2. En general el contexto forma parte de una estructura mas amplia llamada genéricamente TCB (por Task Control Block) o en la jerga de los sistemas operativos de propósito general PCB (Process en lugar de Task).
3. Determinar la ubicación del TCB de la siguiente tarea en la lista de tareas en ejecución.
4. Extraer el contexto de la nueva tarea y copiar registro a registro en la CPU.

Capa de alto nivel: Políticas de Scheduling

La capa de alto nivel llama al context switch cuando necesita conmutar tareas, en base a criterios definidos en su algoritmo.
Antes de invocar al context switch evalúa una serie de variables:

1. prioridad,
2. Preemption,
3. Atomicidad de operaciones,
4. concurrencia,
5. limitaciones en tiempo (latency),
6. paralelismo (en caso de que el sistema cuente con mas de una CPU), entre otros.
   Por lo tanto un **scheduler** resulta un código que puede oscilar entre algo muy sencillo, casi trivial, hasta algoritmos muy complejos que puedan considerar los aspectos señalados en el ítem anterior.

Conclusiones
Approach bootom-up

> Intentaremos abordar el diseño de un scheduler mas o menos sencillo comenzando por la parte basal: *El Context Switch.*
> Esta es la parte dependiente del hardware. La capa superior interfaz consistente

> mediante puede reutilizarse, para lo cual es conveniente escribirla en C, para segurar su portabilidad.

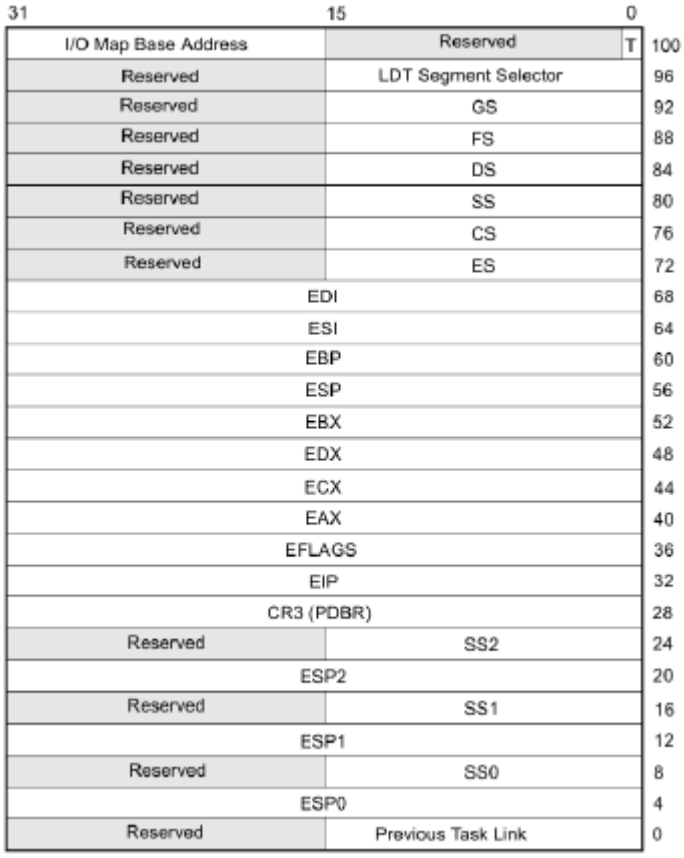
Recursos para manejo de tareas en IA-32

Arquitectura

Provee recursos especiales para manejo de tareas:

1. Segmento de estado de tarea (TSS).
2. Descriptor de TSS
3. Descriptor de Puerta de Tarea
4. Registro de tarea
5. Flag NT (bit 14 de EFLAGS)

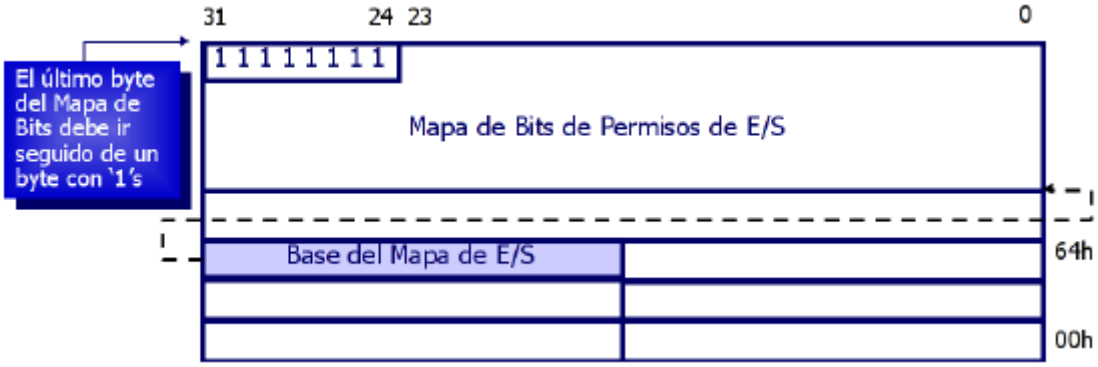
TSS y el Espacio de Contexto



31	15	0	
I/O Map Base Address	Reserved	T	100
Reserved	LDT Segment Selector		96
Reserved	GS		92
Reserved	FS		88
Reserved	DS		84
Reserved	SS		80
Reserved	CS		76
Reserved	ES		72
EDI			68
ESI			64
EBP			60
ESP			56
EBX			52
EDX			48
ECX			44
EAX			40
EFLAGS			36
EIP			32
CR3 (PDBR)			28
Reserved	SS2		24
ESP2			20
Reserved	SS1		16
ESP1			12
Reserved	SS0		8
ESP0			4
Reserved	Previous Task Link		0

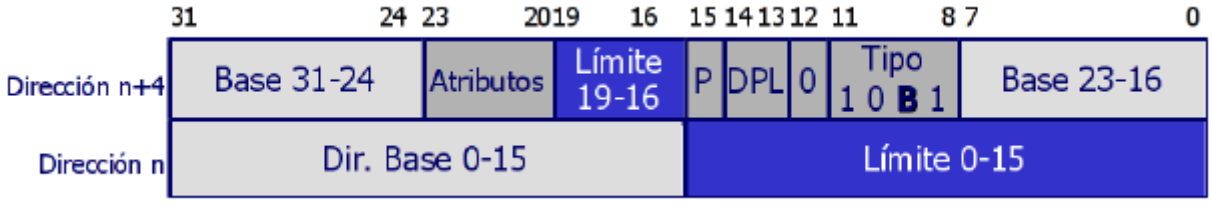
1. Es el lugar de memoria previsto en los procesadores IA-32 como espacio de contexto de cada tarea.
2. El tamaño mínimo de este segmento es 67h.
3. Se guardan los valores de SS y ESP para los stacks de nivel 2, 1, y 0. El del nivel 3 eventualmente estará en los registros SS:ESP.
4. El Flag T genera una excepción de Debug cada vez que se conmuta de tarea (Pentium Pro en adelante), si está en '1'.
5. I/O Map Base Address: Offset de 16 bits desde el inicio del TSS hasta el inicio del Mapa de permisos de E/S

Mapa de bits de E/S



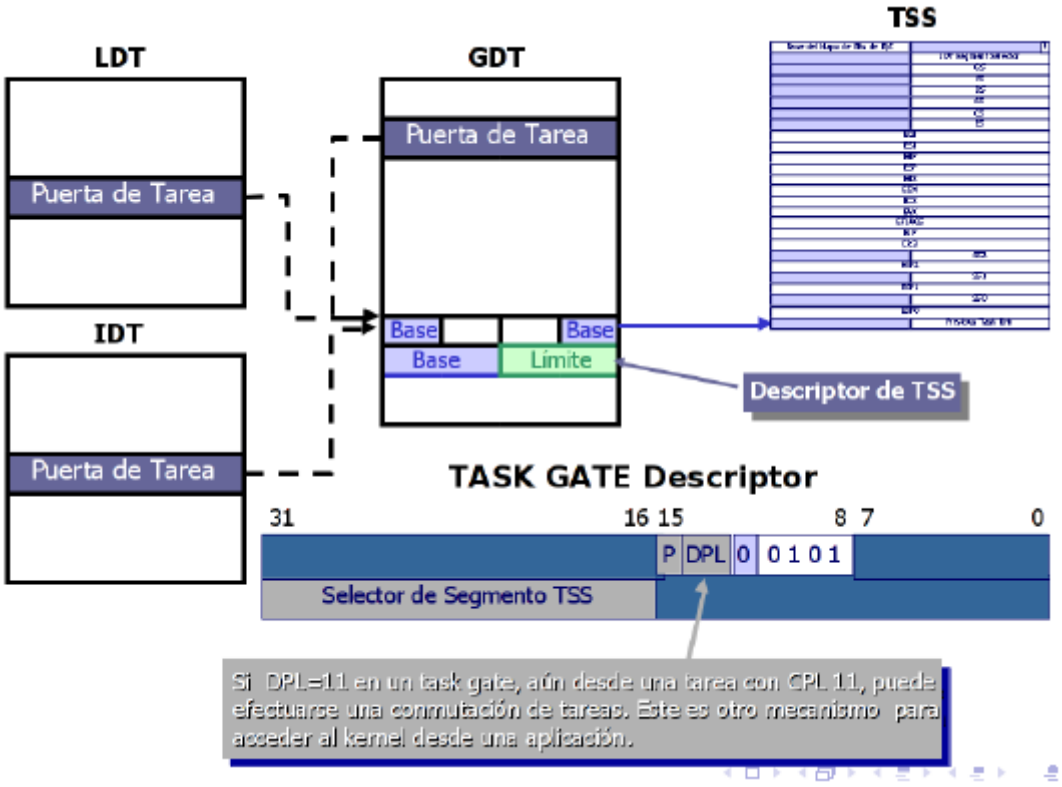
En Modo Protegido, por default una tarea que ejecuta con CPL=11 no puede ejecutar instrucciones de acceso a E/S, es decir IN, OUT, INS, y OUTS.
El procesador utiliza el par de bits **IOPL** del registro **EFLAGS**, para modificar este comportamiento default, de manera selectiva para cada tarea.
Para una determinada tarea con CPL=11, si desde el S.O. se pone el campo **IOPL** de **EFLAGS** en 11, se habilita el acceso a las direcciones de E/S cuyos bit correspondientes estén seteados en el Bit Map de permisos de E/S.
Así se habilita el acceso a determinados ports para determinadas tareas.
En este caso el TSS mide mas de 104 bytes (su límite será mayor que 67h).

Descriptor de TSS



- El Bit **B** (Busy) sirve para evitar recursividad en el anidamiento de tareas. Nos referiremos a él con mas detale cuando analicemos el anidamiento de tareas.
- El Límite debe ser mayor o igual a 67h (mínimo tamaño del segmento es 0x68, o 10310 . De otro modo se genera una excepción TSS inválido, tipo 0x0A.

Task Gate


TSS
LDT
GDT
Puerta de Tarea
Puerta de Tarea
Base
Base
Base
Límite
IDT
Puerta de Tarea
Descriptor de TSS
TASK GATE Descriptor
31
16 15
8 7
0
P
DPL
0
0 1 0 1
Selector de Segmento TSS
Si DPL=11 en un task gate, aún desde una tarea con CPL 11, puede efectuarse una conmutación de tareas. Este es otro mecanismo para acceder al kernel desde una aplicación.