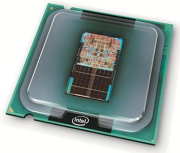


Procesadores IA-32 e Intel® 64 - Tareas

Alejandro Furfaro

11 de noviembre de 2025

Temario

Temario

Definiciones

Definiciones

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

Definiciones

Tarea: 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*).

Definiciones

Tarea: 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.

Definiciones

Tarea: 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

Definiciones

Espacio de ejecución: Es el conjunto de secciones de código, datos, y pila que componen la tarea.

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.

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

Definición

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).

Context Switch

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

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.

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.

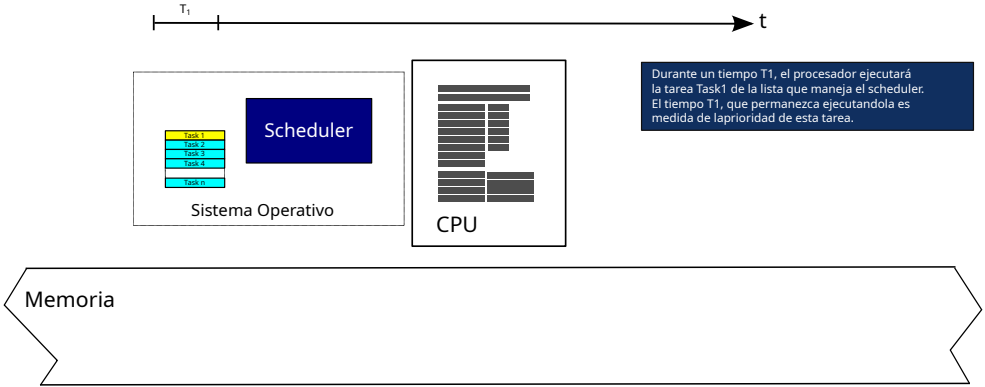
Entra en juego el Sistema Operativo

- 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.

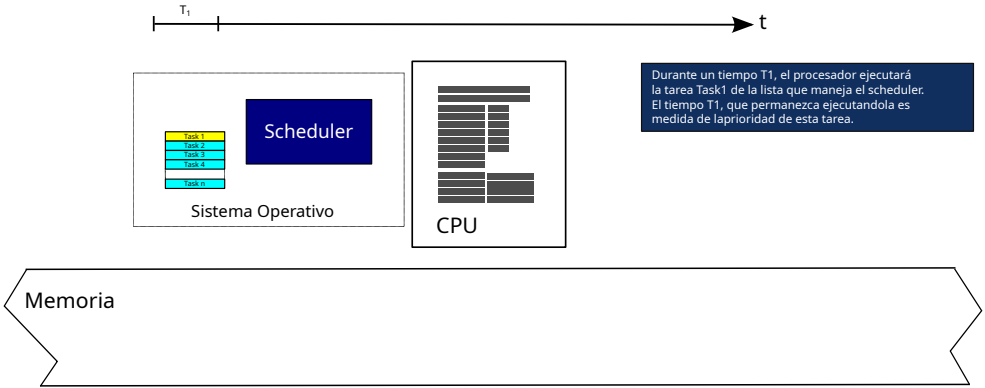
Entra en juego el Sistema Operativo

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

Integrando lo visto hasta aquí

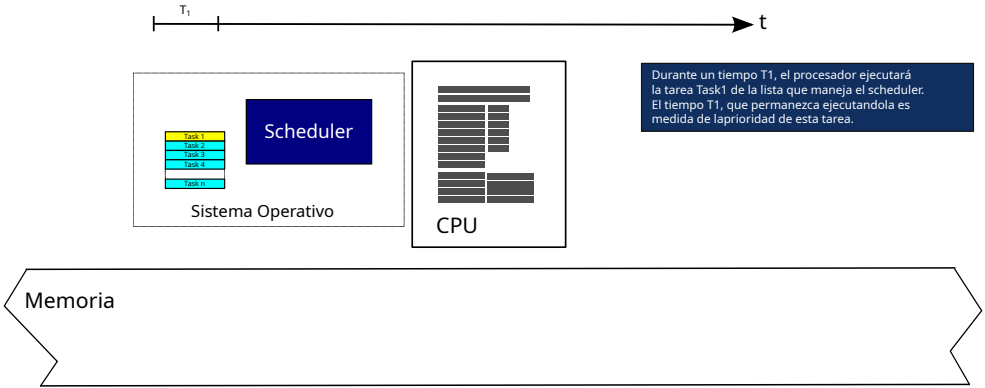


Integrando lo visto hasta aquí



Durante un tiempo T1, el procesador ejecuta la tarea *Task1* de la lista que maneja el ***scheduler***.

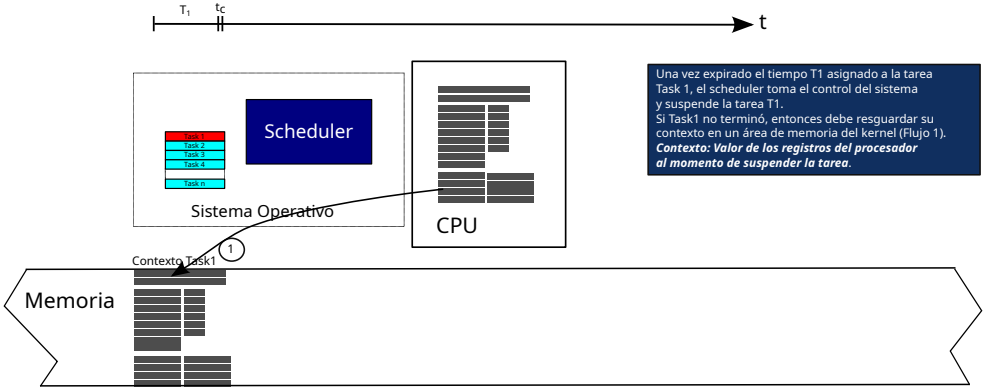
Integrando lo visto hasta aquí



Durante un tiempo T1, el procesador ejecuta la tarea *Task1* de la lista que maneja el ***scheduler***.

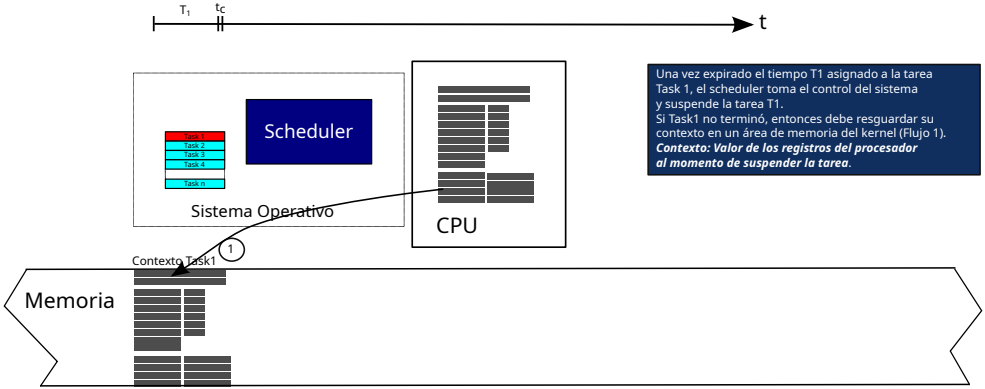
El tiempo T1, que se le asignó para ejecución es medida de la prioridad de esta tarea.

Integrando lo visto hasta aquí



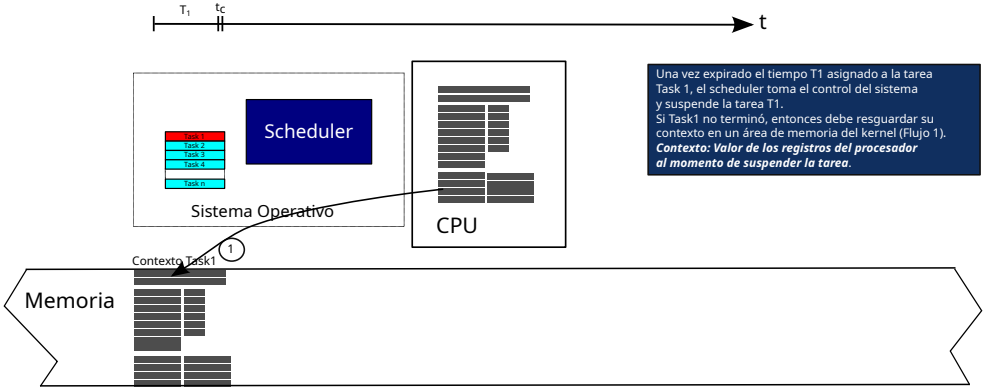
Una vez expirado el tiempo T1 asignado a la tarea Task 1, el scheduler toma el control del sistema y suspende la tarea T1.
Si Task1 no terminó, entonces debe resguardar su contexto en un área de memoria del kernel (Flujo 1).
Contexto: Valor de los registros del procesador al momento de suspender la tarea.

Integrando lo visto hasta aquí



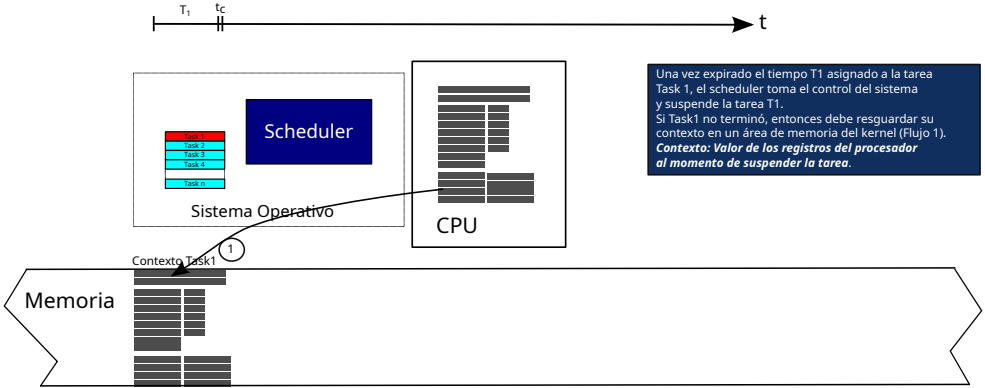
Una vez expirado T1, el ***scheduler*** toma el control del sistema y como la tarea *Task1* no finalizó, la suspende, asignándole estado **SUSPENDING**.

Integrando lo visto hasta aquí



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).

Integrando lo visto hasta aquí

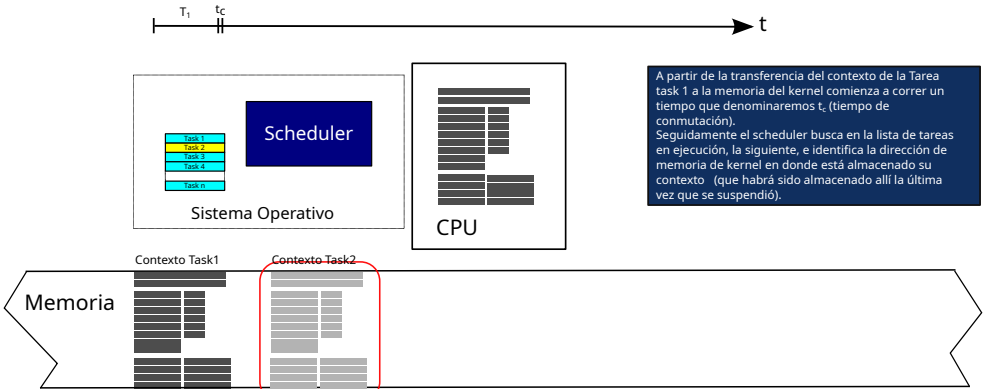


Una vez expirado T1, el ***scheduler*** toma el control del sistema y como la tarea *Task1* no finalizó, la suspende, asignándole estado **SUSPENDING**.

Resguarda el contexto de *Task1* en un área de **memoria del kernel** (Flujo 1).

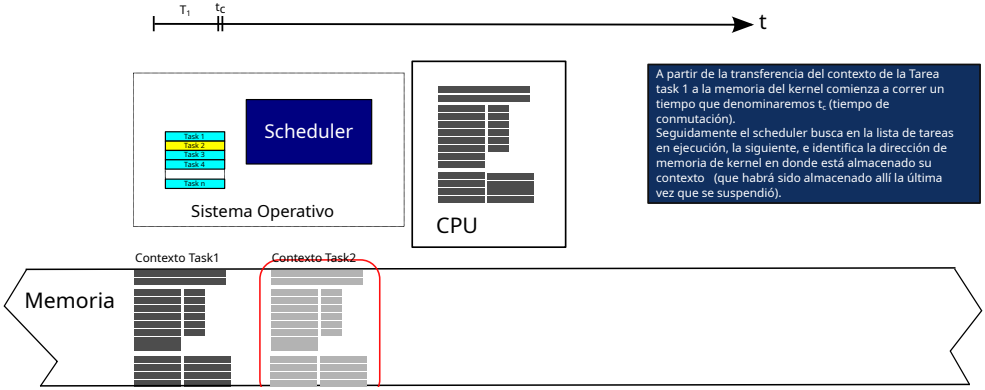
Contexto: Valor de los registros del procesador al momento de suspender la tarea.

Integrando lo visto hasta aquí



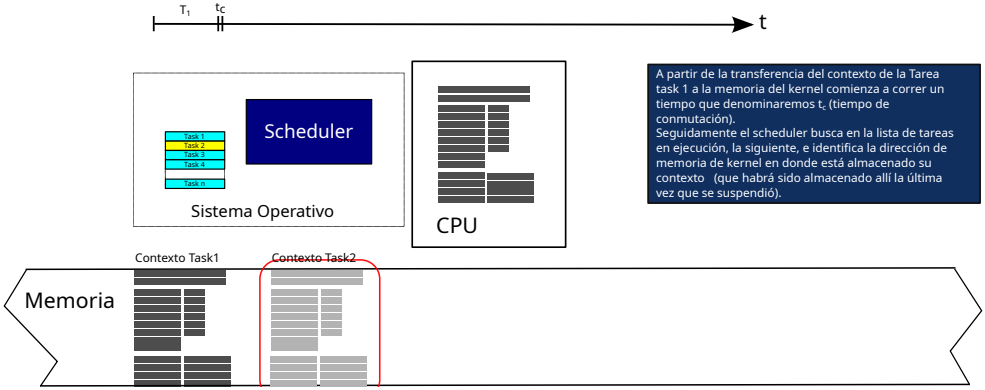
A partir de la transferencia del contexto de la Tarea task 1 a la memoria del kernel comienza a correr un tiempo que denominaremos t_c (tiempo de conmutación).
Seguidamente el scheduler busca en la lista de tareas en ejecución, la siguiente, e identifica la dirección de memoria de kernel en donde está almacenado su contexto (que habrá sido almacenado allí la última vez que se suspendió).

Integrando lo visto hasta aquí



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).

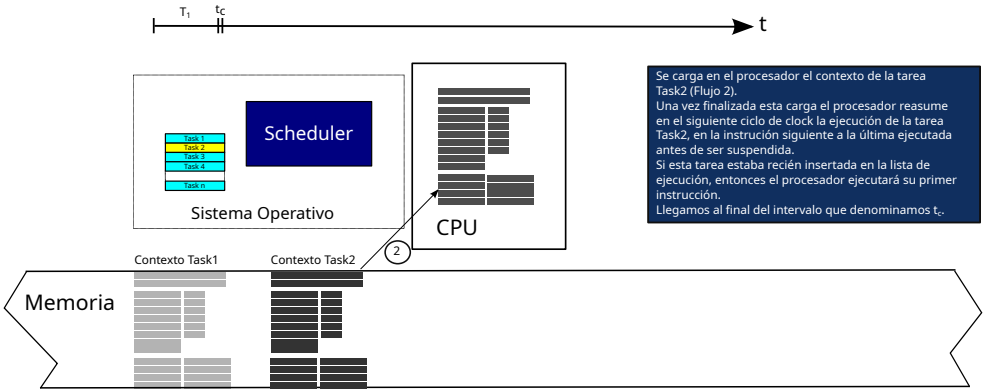
Integrando lo visto hasta aquí



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).

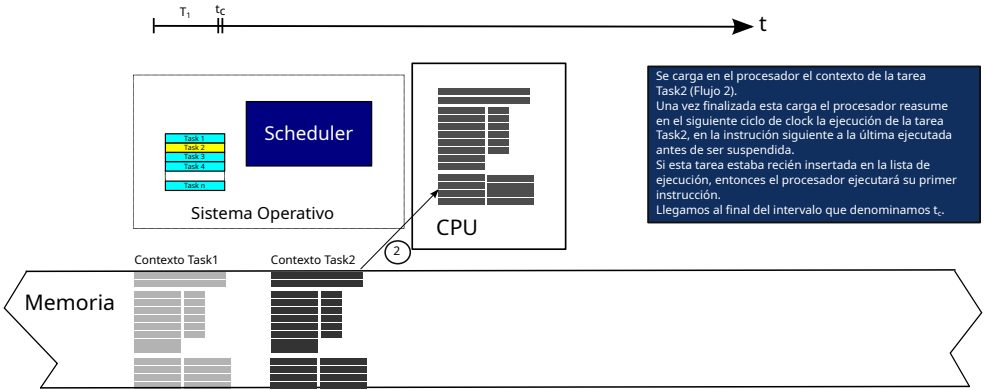
Luego el ***scheduler*** busca la siguiente tarea en la lista para ejecutar (*Task2*), identifica la dirección de memoria de kernel en donde está su contexto, y marca a la tarea **RUNNING**.

Integrando lo visto hasta aquí



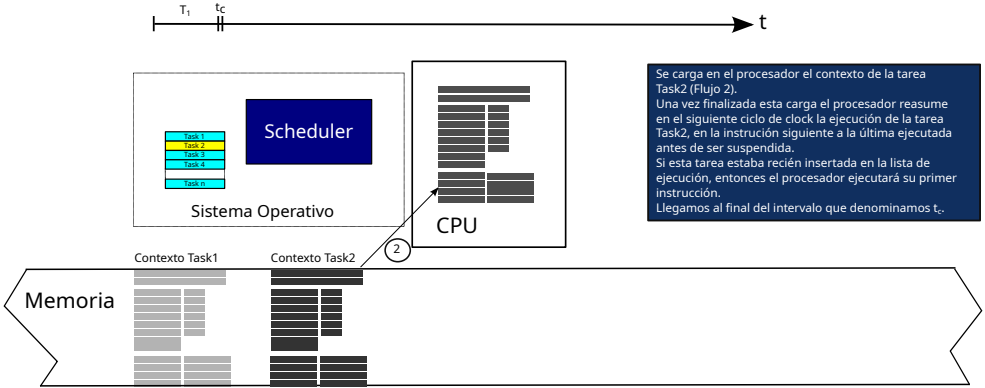
Se carga en el procesador el contexto de la tarea Task2 (Flujo 2).
Una vez finalizada esta carga el procesador reasume en el siguiente ciclo de clock la ejecución de la tarea Task2, en la instrución siguiente a la última ejecutada antes de ser suspendida.
Si esta tarea estaba recién insertada en la lista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

Integrando lo visto hasta aquí



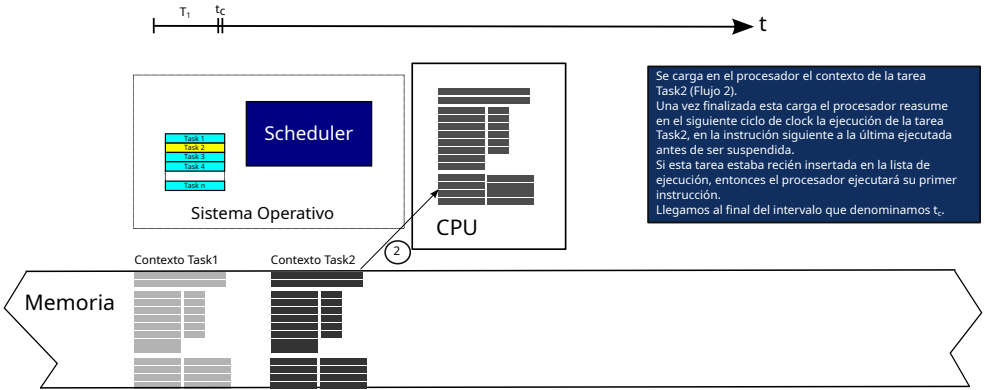
Se carga en el procesador el contexto de la tarea *Task2* (Flujo 2).

Integrando lo visto hasta aquí



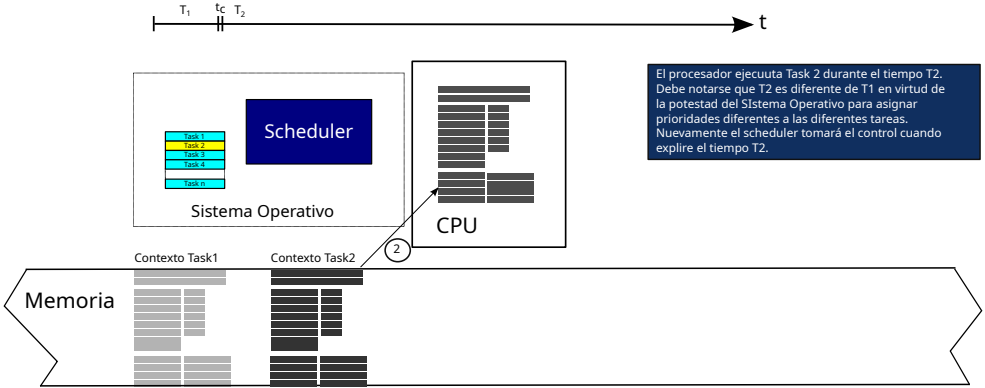
Se carga en el procesador el contexto de la tarea *Task2* (Flujo 2).
En el siguiente ciclo de clock, se reasume la ejecución de *Task2*, en la instrucción siguiente a la última ejecutada antes de ser suspendida. Si *Task2* fue recién insertada en la lista de ejecución, entonces se ejecutará su primer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

Integrando lo visto hasta aquí



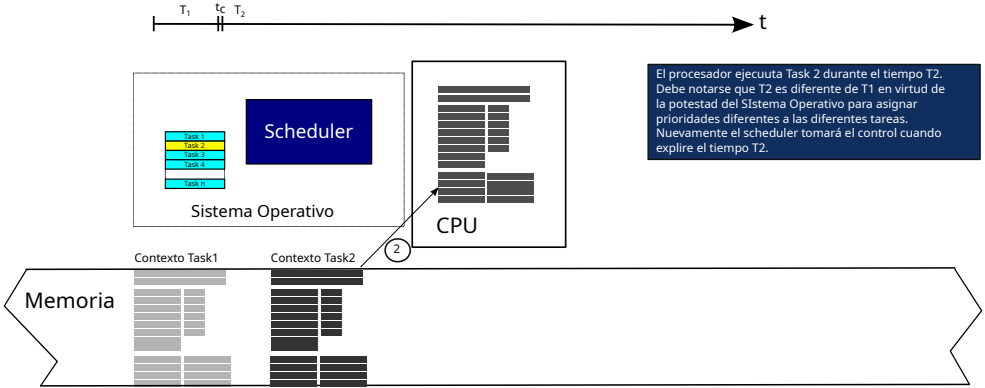
Se carga en el procesador el contexto de la tarea *Task2* (Flujo 2).
En el siguiente ciclo de clock, se reasume la ejecución de *Task2*, en la instrucción siguiente a la última ejecutada antes de ser suspendida. Si *Task2* fue recién insertada en la lista de ejecución, entonces se ejecutará su primer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

Integrando lo visto hasta aquí



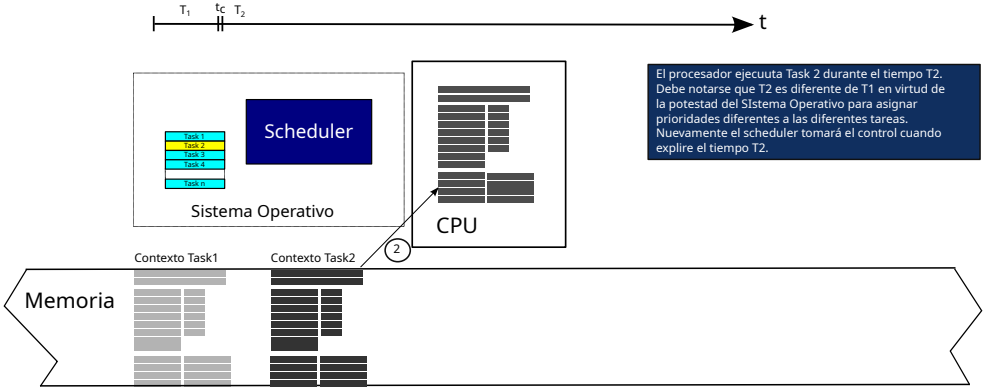
El procesador ejecuuta Task 2 durante el tiempo T2. Debe notarse que T2 es diferente de T1 en virtud de la potestad del SIstema Operativo para asignar prioridades diferentes a las diferentes tareas. Nuevamente el scheduler tomará el control cuando explire el tiempo T2.

Integrando lo visto hasta aquí



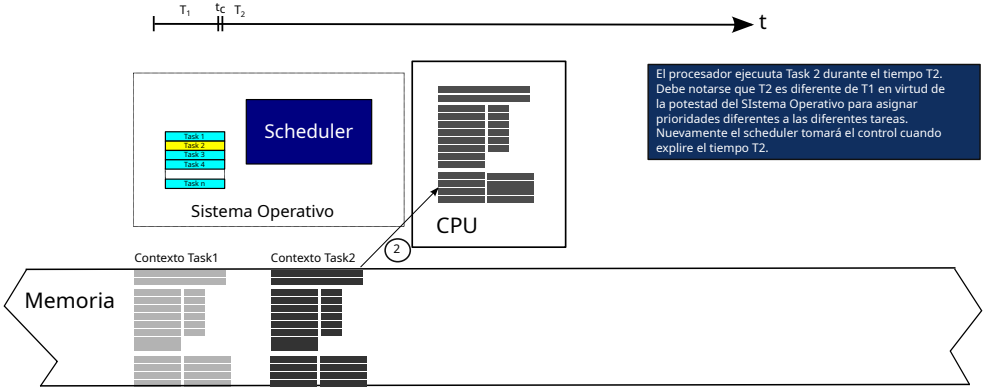
El procesador ejecuta *Task2* durante el tiempo T2.

Integrando lo visto hasta aquí



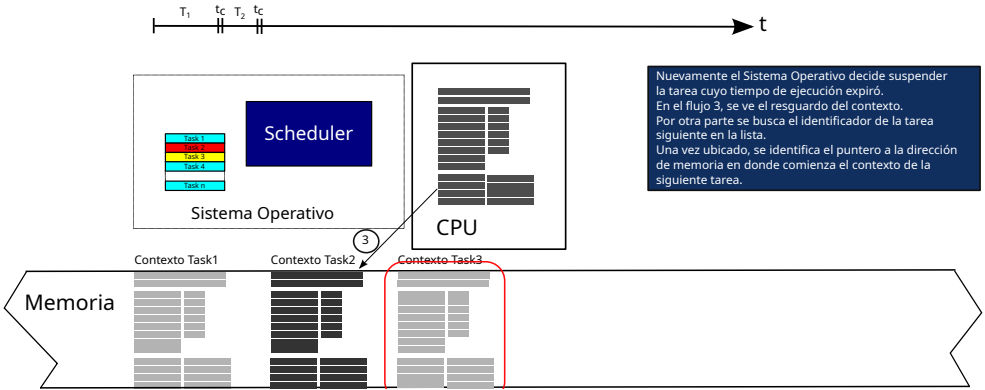
El procesador ejecuta *Task2* durante el tiempo T2.
Notar que T2 $\neq$ T1 ya que el Sistema Operativo asigna prioridades diferentes a cada tarea.

Integrando lo visto hasta aquí



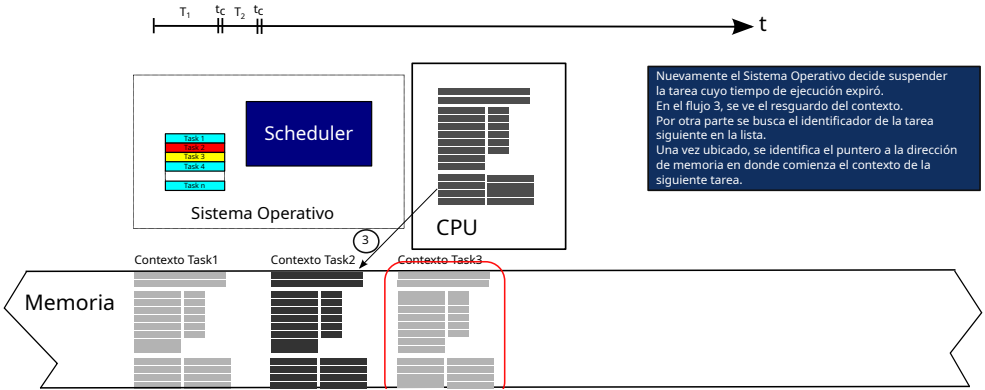
El procesador ejecuta *Task2* durante el tiempo T2.
Notar que T2 $\neq$ T1 ya que el Sistema Operativo asigna prioridades diferentes a cada tarea.
Nuevamente el ***scheduler*** tomará el control cuando expire el tiempo T2.

Integrando lo visto hasta aquí



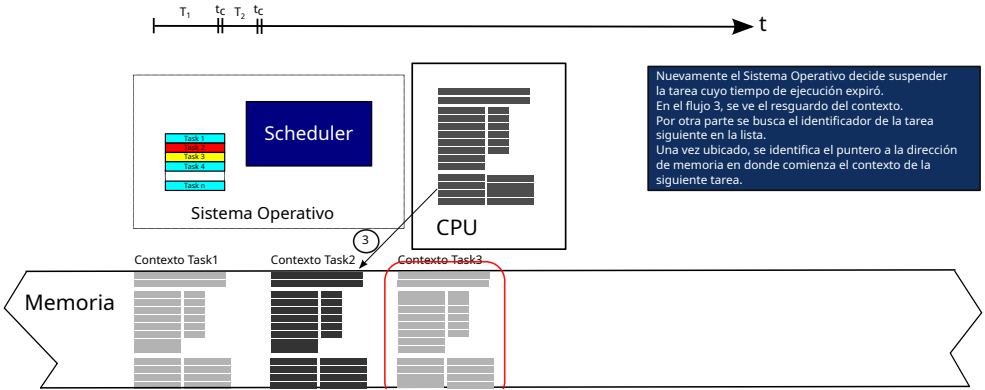
Nuevamente el Sistema Operativo decide suspender la tarea cuyo tiempo de ejecución expiró.
En el flujo 3, se ve el resguardo del contexto.
Por otra parte se busca el identificador de la tarea siguiente en la lista.
Una vez ubicado, se identifica el puntero a la dirección de memoria en donde comienza el contexto de la siguiente tarea.

Integrando lo visto hasta aquí



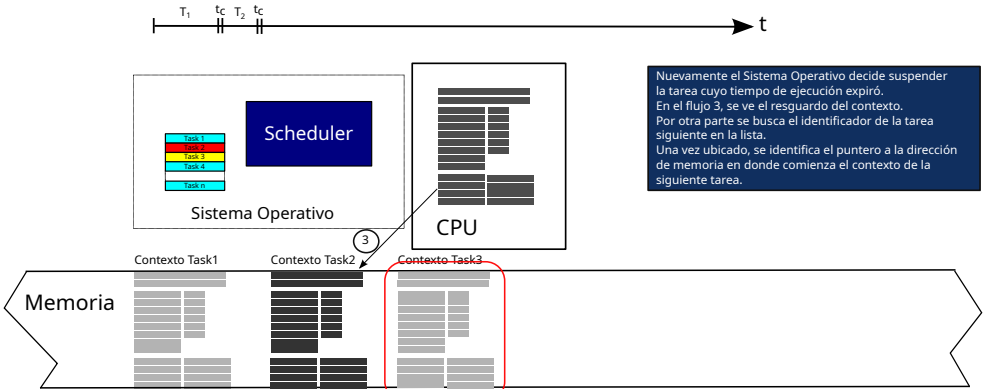
El Sistema Operativo pone *Task2* **SUSPENDING** ya que su tiempo de ejecución expiró.

Integrando lo visto hasta aquí



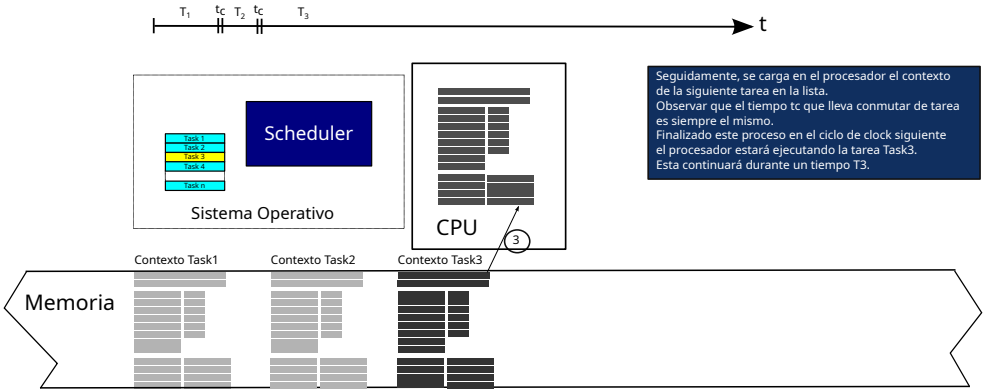
El Sistema Operativo pone *Task2* **SUSPENDING** ya que su tiempo de ejecución expiró.
En el flujo 3, se ve el resguardo del contexto. Por otra parte se busca el identificador de la tarea siguiente en la lista.

Integrando lo visto hasta aquí



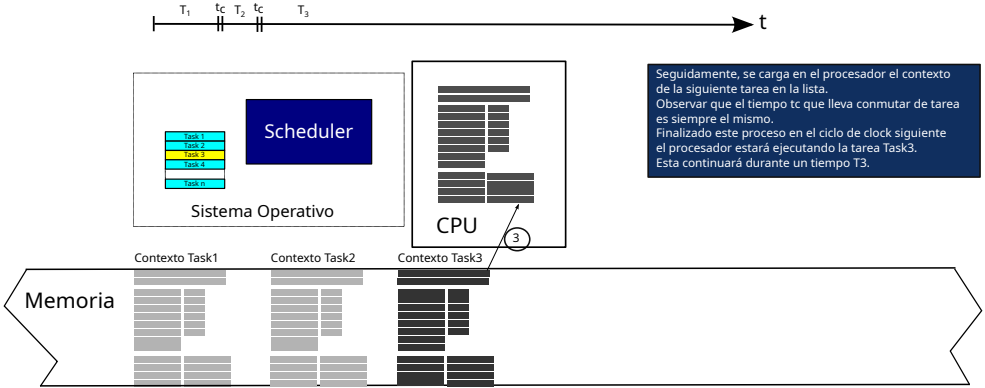
El Sistema Operativo pone *Task2* **SUSPENDING** ya que su tiempo de ejecución expiró.
En el flujo 3, se ve el resguardo del contexto. Por otra parte se busca el identificador de la tarea siguiente en la lista.
Una vez ubicado, se identifica el puntero a la dirección de memoria en donde comienza el contexto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

Integrando lo visto hasta aquí



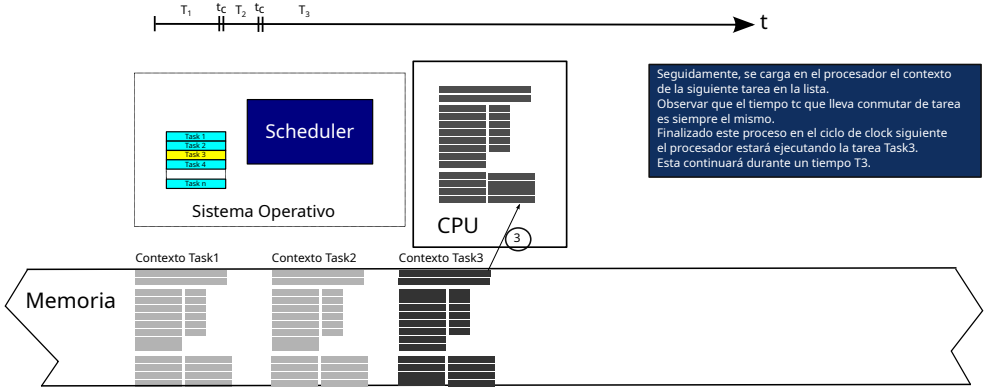
Seguidamente, se carga en el procesador el contexto de la siguiente tarea en la lista.
Observar que el tiempo tc que lleva conmutar de tarea es siempre el mismo.
Finalizado este proceso en el ciclo de clock siguiente el procesador estará ejecutando la tarea Task3.
Esta continuará durante un tiempo T3.

Integrando lo visto hasta aquí



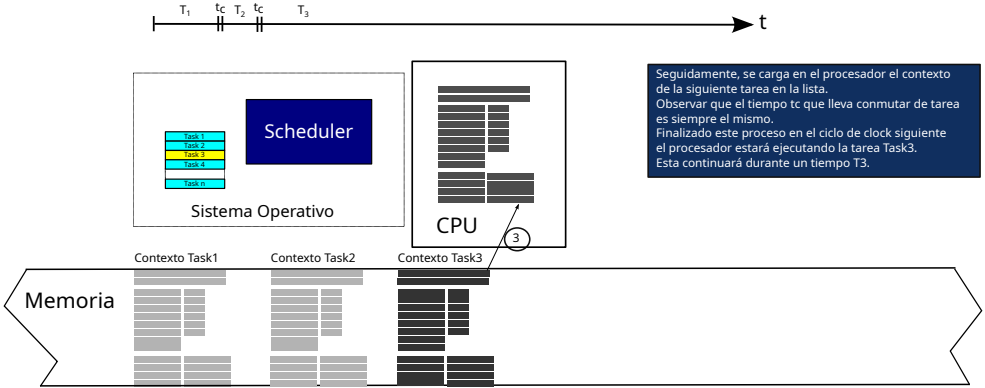
Identificada la tarea siguiente, se carga su contexto en el procesador.

Integrando lo visto hasta aquí



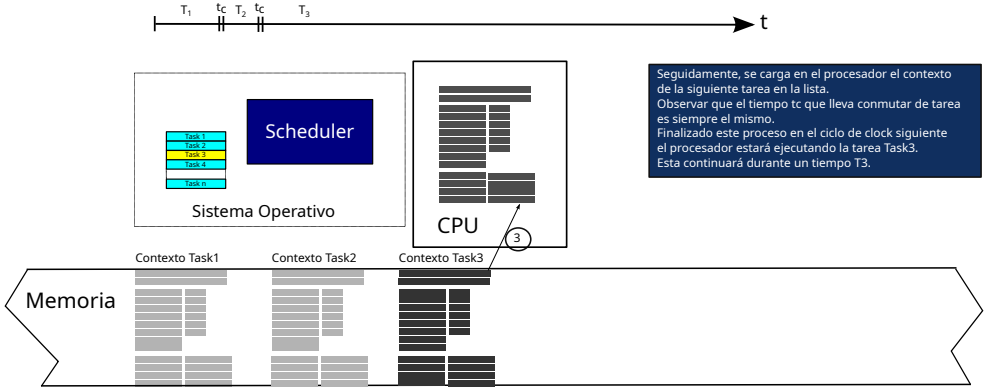
Identificada la tarea siguiente, se carga su contexto en el procesador.
Observar que el tiempo t_c es siempre el mismo independientemente de las tareas.

Integrando lo visto hasta aquí



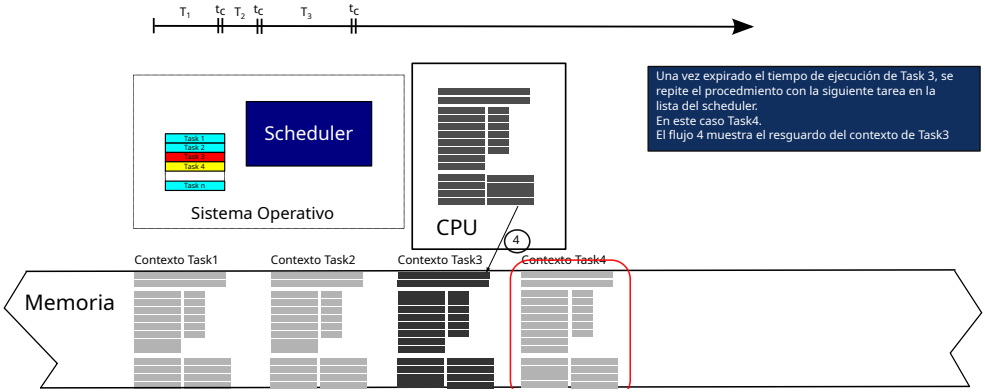
Identificada la tarea siguiente, se carga su contexto en el procesador.
Observar que el tiempo t_c es siempre el mismo independientemente de las tareas.
Finalizado este proceso en siguiente ciclo el procesador estará ejecutando la tarea *Task3*.

Integrando lo visto hasta aquí

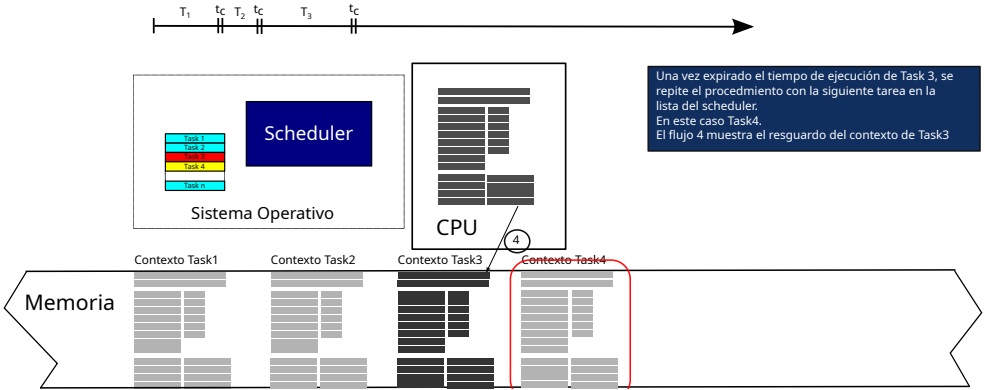


Identificada la tarea siguiente, se carga su contexto en el procesador.
Observar que el tiempo t_c es siempre el mismo independientemente de las tareas.
Finalizado este proceso en siguiente ciclo el procesador estará ejecutando la tarea *Task3*.
Esta continuará durante un tiempo T3.

Integrando lo visto hasta aquí

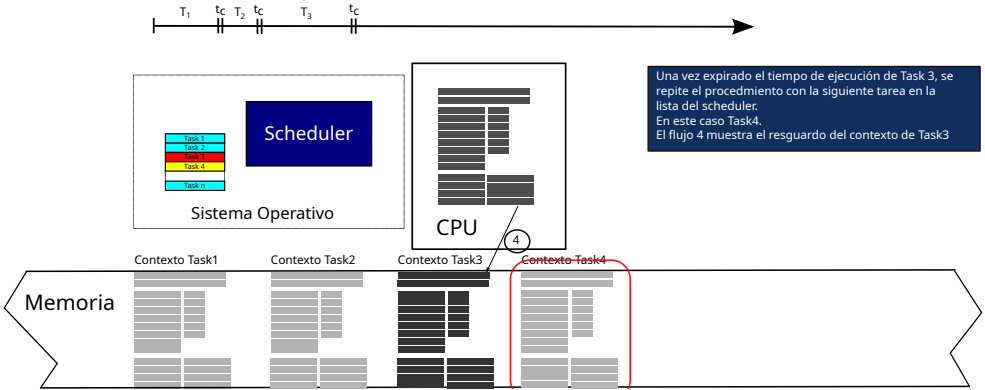


Integrando lo visto hasta aquí



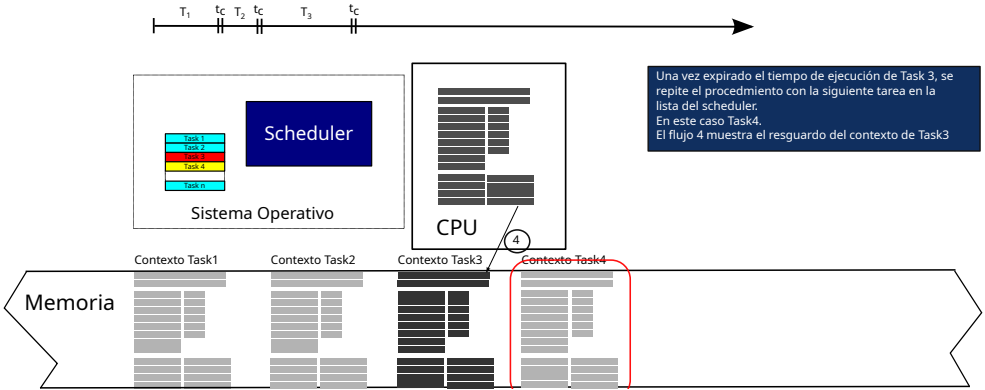
Una vez expirado el tiempo de ejecución de *Task3*, se repite el procedimiento con la siguiente tarea en la lista del ***scheduler***.

Integrando lo visto hasta aquí



Una vez expirado el tiempo de ejecución de *Task3*, se repite el procedimiento con la siguiente tarea en la lista del ***scheduler***.
En este caso *Task4*.

Integrando lo visto hasta aquí

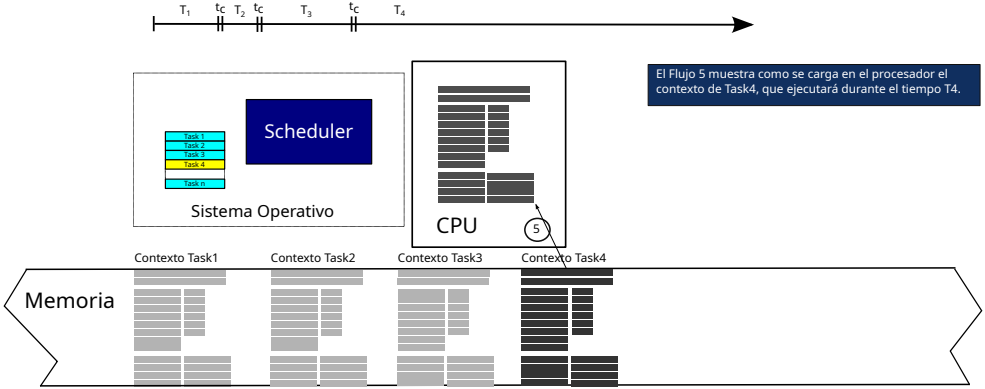


Una vez expirado el tiempo de ejecución de *Task3*, se repite el procedimiento con la siguiente tarea en la lista del ***scheduler***.

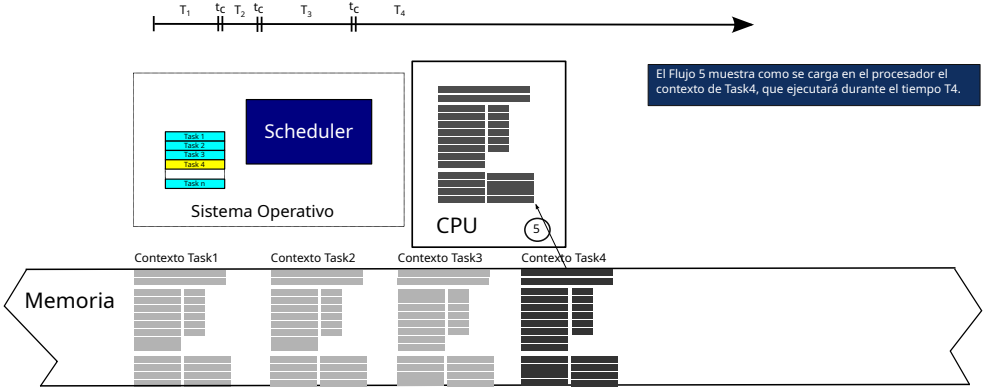
En este caso *Task4*.

El flujo 4 muestra el resguardo del contexto de *Task3*.

Integrando lo visto hasta aquí

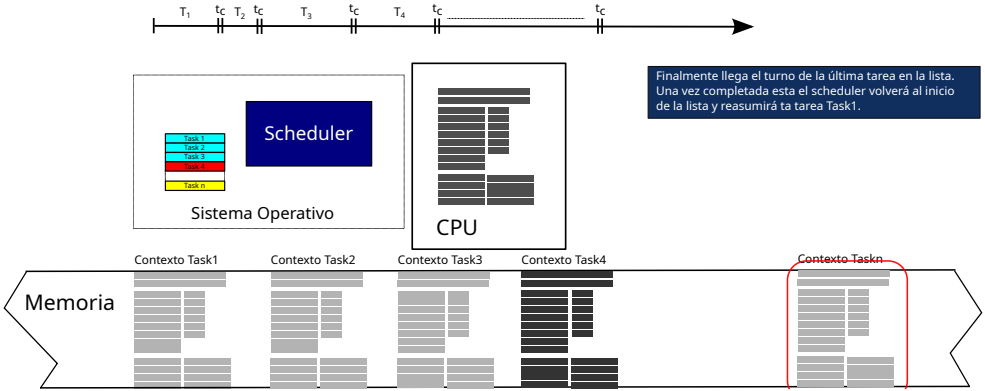


Integrando lo visto hasta aquí

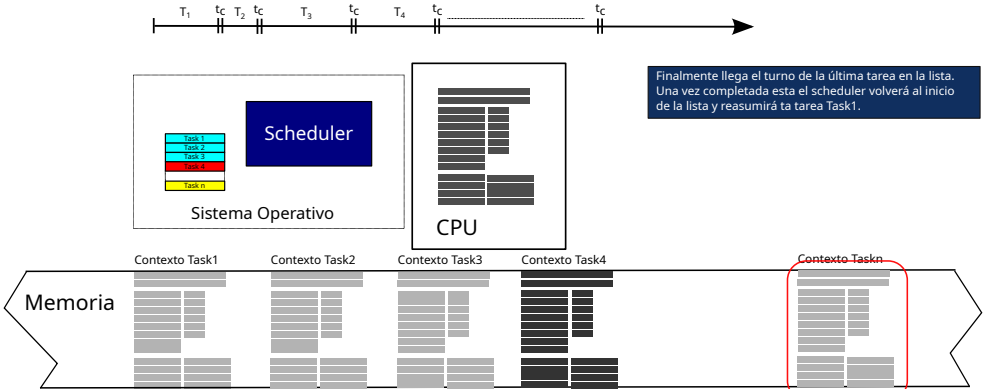


El Flujo 5 muestra como se carga en el procesador el contexto de *Task4*, que ejecutará durante el tiempo T4.

Integrando lo visto hasta aquí

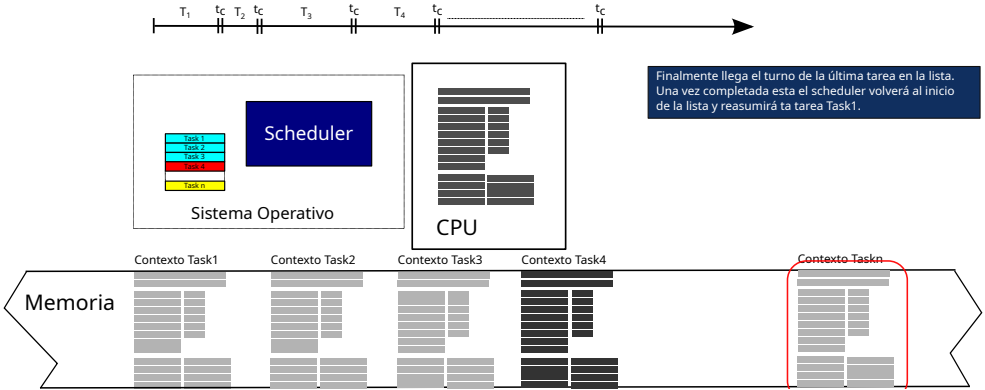


Integrando lo visto hasta aquí



Finalmente llega el turno de la última tarea en la lista.

Integrando lo visto hasta aquí



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*.

Temario

Modelo de Programación de Sistemas

Modelo de Programación de Sistemas

- Hasta ahora hemos trabajado con la vista de una arquitectura basada en el desarrollo de aplicaciones

Modelo de Programación de Sistemas

- Hasta ahora hemos trabajado con la vista de una arquitectura basada en el desarrollo de aplicaciones
  - ✓ Programación en C, C++, o Assembler en entornos Linux, o Windows.

Modelo de Programación de Sistemas

- Hasta ahora hemos trabajado con la vista de una arquitectura basada en el desarrollo de aplicaciones
  - ✓ Programación en C, C++, o Assembler en entornos Linux, o Windows.
  - ✓ Desarrollo de aplicaciones orientadas a objetos en vistosos entornos gráficos.

Modelo de Programación de Sistemas

- Hasta ahora hemos trabajado con la vista de una arquitectura basada en el desarrollo de aplicaciones
  - ✓ Programación en C, C++, o Assembler en entornos Linux, o Windows.
  - ✓ Desarrollo de aplicaciones orientadas a objetos en vistosos entornos gráficos.
  - ✓ Desarrollo de aplicaciones en microcontroladores.

Modelo de Programación de Sistemas

- Hasta ahora hemos trabajado con la vista de una arquitectura basada en el desarrollo de aplicaciones
  - ✓ Programación en C, C++, o Assembler en entornos Linux, o Windows.
  - ✓ Desarrollo de aplicaciones orientadas a objetos en vistosos entornos gráficos.
  - ✓ Desarrollo de aplicaciones en microcontroladores.
- Trabajamos con un conjunto de recursos que se conocen como "**Modelo de programación de Aplicaciones**".

Modelo de Programación de Sistemas

- Hasta ahora hemos trabajado con la vista de una arquitectura basada en el desarrollo de aplicaciones
  - ✓ Programación en C, C++, o Assembler en entornos Linux, o Windows.
  - ✓ Desarrollo de aplicaciones orientadas a objetos en vistosos entornos gráficos.
  - ✓ Desarrollo de aplicaciones en microcontroladores.
- Trabajamos con un conjunto de recursos que se conocen como "**Modelo de programación de Aplicaciones**".
- Este modelo describe los registros de propósito general, las instrucciones, los tipos de datos, los modos de direccionamiento...
  tal parece que estaría todo dicho.

Modelo de Programación de Sistemas

- Hasta ahora hemos trabajado con la vista de una arquitectura basada en el desarrollo de aplicaciones
  - ✓ Programación en C, C++, o Assembler en entornos Linux, o Windows.
  - ✓ Desarrollo de aplicaciones orientadas a objetos en vistosos entornos gráficos.
  - ✓ Desarrollo de aplicaciones en microcontroladores.
- Trabajamos con un conjunto de recursos que se conocen como "**Modelo de programación de Aplicaciones**".
- Este modelo describe los registros de propósito general, las instrucciones, los tipos de datos, los modos de direccionamiento...
  tal parece que estaría todo dicho.
- Esto no es todo

¿Acaso hay mas recursos de desarrollo?

Un paso mas hacia el hardware

Si nuestro requerimiento es diseñar un sistema que permita
almacenar simultáneamente en memoria varias tareas,
administrar su ejecución, evitar que se interfieran,
y regular su acceso a los recursos de hardware disponibles,
evidentemente estamos entrando en un mundo en el que la visión de Programador de Aplicaciones claramente no es suficiente.

¿Acaso hay mas recursos de desarrollo?

¿Acaso hay mas recursos de desarrollo?

- Entonces se trata de desarrollar un conjunto de tareas independientes entre si que contribuyan a implementar el sistema requerido, y un software que las administre.

¿Acaso hay mas recursos de desarrollo?

- Entonces se trata de desarrollar un conjunto de tareas independientes entre si que contribuyan a implementar el sistema requerido, y un software que las administre.
- Necesitamos contar con un SoC cuya CPU tenga capacidades para realizar el propósito planteado.

¿Acaso hay mas recursos de desarrollo?

- Entonces se trata de desarrollar un conjunto de tareas independientes entre si que contribuyan a implementar el sistema requerido, y un software que las administre.
- Necesitamos contar con un SoC cuya CPU tenga capacidades para realizar el propósito planteado.
- Si nos las tiene, se puede desarrollar un sistema como el que nos estamos planteando, de todos modos. Pero no esperemos milagros en cuanto a rendimiento, ni le exijamos demasiado al modelo implementado.

¿Acaso hay mas recursos de desarrollo?

- Entonces se trata de desarrollar un conjunto de tareas independientes entre si que contribuyan a implementar el sistema requerido, y un software que las administre.
- Necesitamos contar con un SoC cuya CPU tenga capacidades para realizar el propósito planteado.
- Si nos las tiene, se puede desarrollar un sistema como el que nos estamos planteando, de todos modos. Pero no esperemos milagros en cuanto a rendimiento, ni le exijamos demasiado al modelo implementado.
- Primer paso: tener claro que CPU necesitamos para el requerimiento de nuestro sistema

Modelo de Programación de Sistemas

Modelo de Programación de Sistemas

- El modelo de programación de Aplicaciones, la CPU tiene una visión restringida del sistema en su conjunto.

Modelo de Programación de Sistemas

- El modelo de programación de Aplicaciones, la 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 más veloz y eficiente.

Modelo de Programación de Sistemas

- El modelo de programación de Aplicaciones, la 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 más 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.

Modelo de Programación de Sistemas

- El modelo de programación de Aplicaciones, la 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 más 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

Scheduling de Tareas

- Cualquier Sistema Operativo posee un módulo llamado **scheduler**, que le permite administrar la ejecución de tareas / procesos.

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:

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*.

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*.

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 más adelante los algoritmos de scheduling.

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 más 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

Capa de bajo nivel: Context Switch

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

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.

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 más amplia llamada genéricamente **TCB** (por **T**ask **C**ontrol **B**lock) o en la jerga de los sistemas operativos de propósito general PCB (**P**rocess en lugar de **T**ask).

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 más amplia llamada genéricamente **TCB** (por **T**ask **C**ontrol **B**lock) o en la jerga de los sistemas operativos de propósito general PCB (**P**rocess en lugar de **T**ask).
  3. Determinar la ubicación del TCB de la siguiente tarea en la lista de tareas en ejecución.

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 más amplia llamada genéricamente **TCB** (por **T**ask **C**ontrol **B**lock) o en la jerga de los sistemas operativos de propósito general PCB (**P**rocess en lugar de **T**ask).
  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

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.

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:

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,

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,

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,

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,

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),

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.

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

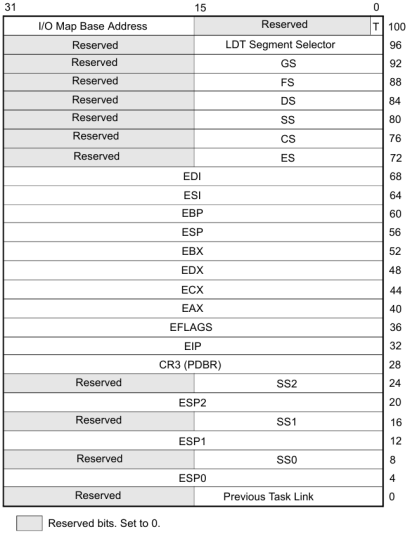
Intentaremos abordar el diseño de un scheduler mas o menos sencillo comenzando por la parte basal: El Context Switch.
Esta es la parte dependiente del hardware. La capa superior interfaz consistente mediante puede reutilizarse, para lo cual es conveniente escribirla en C, para segurar su portabilidad.

Temario

La arquitectura se basa en 5 elementos

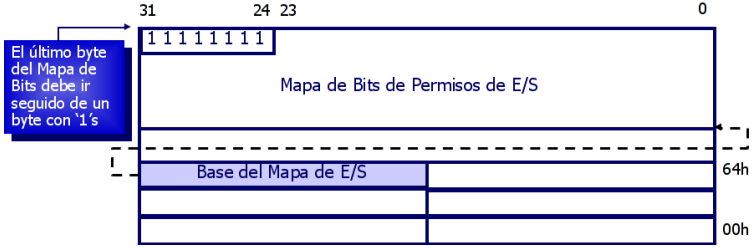
1. Segmento de estado de tarea (TSS).
2. Descriptor de TSS
3. Descriptor de Puerta de Tarea
4. Registro de tarea
5. Flag NT (bit 14 de EFLAGS)

El TSS implementa el Espacio de Contexto



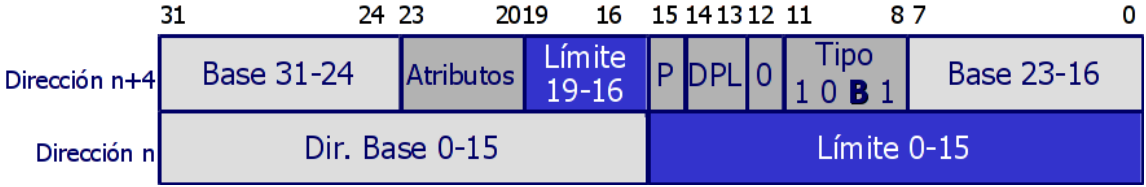
1. Es el lugar de memoria previsto en los procesadores IA-32 como espacio de contexto de cada tarea.
2. El tamaño mínimo de este segmento es 67h.
3. Se guardan los valores de SS y ESP para los stacks de nivel 2, 1, y 0. El del nivel 3 eventualmente estará en los registros SS:ESP.
4. El Flag T genera 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



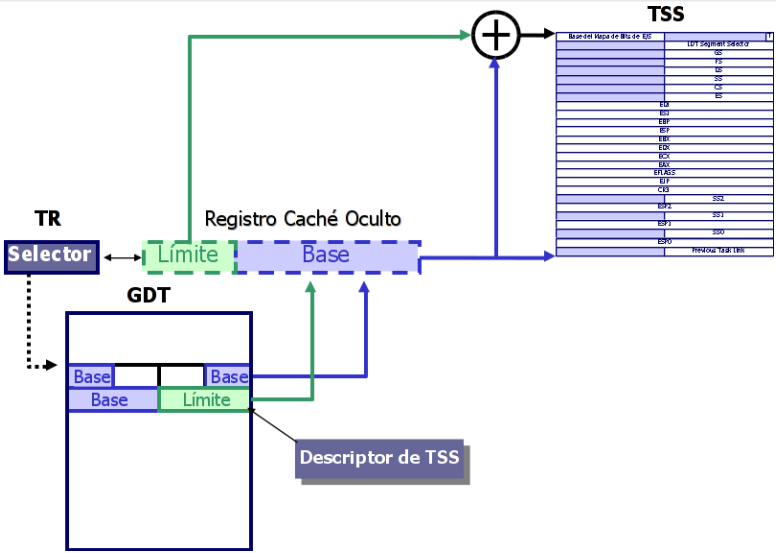
- En Modo Protegido, por default una tarea que ejecuta con **CPL=11** no puede ejecutar instrucciones de acceso a E/S, es decir IN, OUT, INS, y OUTS.
- El procesador utiliza el par de bits **IOPL** del registro **EFLAGS**, para modificar este comportamiento default, de manera selectiva para cada tarea.
- 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 bits 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).

Cada TSS necesita un descriptor

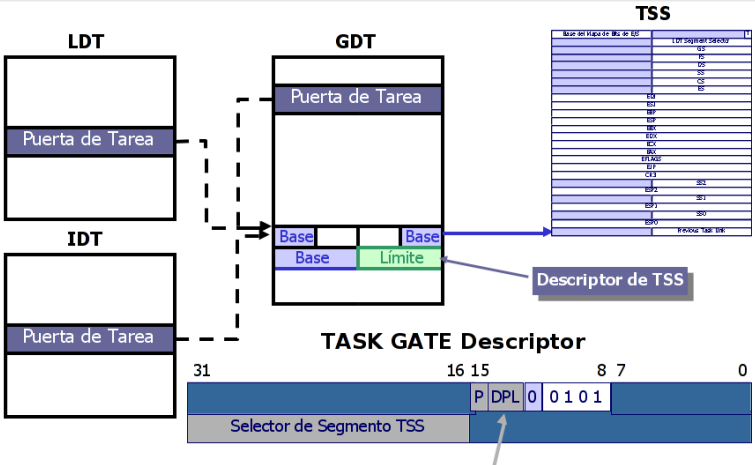


- El Bit **B** (Busy) sirve para evitar recursividad en el anidamiento de tareas. Nos referiremos a él con mas detale cuando analicemos el anidamiento de tareas.
- El Límite debe ser mayor o igual a 67h (mínimo tamaño del segmento es 0x68, o 103_{10}. De otro modo se genera una excepción TSS inválido, tipo 0x0A.

Cada TSS necesita un descriptor



Task Gate: otra forma de acceder a una tarea



Temario

¿Como se cambia a una tarea?

- El procesador puede despachar una tarea de las siguientes formas posibles:
  1. Por medio de una instrucción CALL
  2. Por medio de una instrucción JMP
  3. Mediante una llamada implícita al handler de una interrupción manejado por una tarea.
  4. Mediante una llamada implícita al handler de una excepción manejado por una tarea.
  5. Mediante la ejecución de la instrucción IRET en una tarea cuando el flag NT (bit 14 del registro EFLAGS) es “1” para la tarea actual.

¿Como se cambia a una tarea?

- El procesador puede despachar una tarea de las siguientes formas posibles:
  1. Por medio de una instrucción CALL
  2. Por medio de una instrucción JMP
  3. Mediante una llamada implícita al handler de una interrupción manejado por una tarea.
  4. Mediante una llamada implícita al handler de una excepción manejado por una tarea.
  5. Mediante la ejecución de la instrucción IRET en una tarea cuando el flag NT (bit 14 del registro EFLAGS) es “1” para la tarea actual.
- En cualquier caso se requiere poder identificar a la tarea.

¿Como se cambia a una tarea?

- El procesador puede despachar una tarea de las siguientes formas posibles:
  1. Por medio de una instrucción CALL
  2. Por medio de una instrucción JMP
  3. Mediante una llamada implícita al handler de una interrupción manejado por una tarea.
  4. Mediante una llamada implícita al handler de una excepción manejado por una tarea.
  5. Mediante la ejecución de la instrucción IRET en una tarea cuando el flag NT (bit 14 del registro EFLAGS) es “1” para la tarea actual.
- En cualquier caso se requiere poder identificar a la tarea.
- Se necesita un selector en la GDT que apunte a una puerta de tarea o a un Task State Segment (TSS). Este selector debe estar en la correspondiente posición dentro de la instrucción CALL o JMP.

Proceso de conmutación de tarea

- El procesador analiza el valor que debe colocar en el registro CS como parte de la ejecución de un JMP, CALL, IRET (si NT = 1), o al que obtiene de una Interrupt Gate o Trap Gate.

Proceso de conmutación de tarea

- El procesador analiza el valor que debe colocar en el registro CS como parte de la ejecución de un JMP, CALL, IRET (si NT = 1), o al que obtiene de una Interrupt Gate o Trap Gate.

- Busca el descriptor en la GDT.

Proceso de conmutación de tarea

- El procesador analiza el valor que debe colocar en el registro CS como parte de la ejecución de un JMP, CALL, IRET (si NT = 1), o al que obtiene de una Interrupt Gate o Trap Gate.
- Busca el descriptor en la GDT.
- Si es un Task Gate, Vuelve a buscar en la GDT donde deberá encontrar exclusivamente un descriptor de TSS.

Proceso de conmutación de tarea

- El procesador analiza el valor que debe colocar en el registro CS como parte de la ejecución de un JMP, CALL, IRET (si NT = 1), o al que obtiene de una Interrupt Gate o Trap Gate.
- Busca el descriptor en la GDT.
- Si es un Task Gate, Vuelve a buscar en la GDT donde deberá encontrar exclusivamente un descriptor de TSS.
- Si es un selector de TSS lo que intenta cargarse en CS, busca en la GDT el descriptor de TSS

Proceso de conmutación de tarea

- Reserva el descriptor de TSS y el selector en registros temporarios ya que en TR está el selector del TSS actual, cuya dirección base atributos y límite están en el registro cache hidden asociado.

Proceso de conmutación de tarea

- Reserva el descriptor de TSS y el selector en registros temporarios ya que en TR está el selector del TSS actual, cuya dirección base atributos y límite están en el registro cache hidden asociado.

- Emplea el TSS actual para almacenar el estado del procesador.

Proceso de conmutación de tarea

- Reserva el descriptor de TSS y el selector en registros temporarios ya que en TR está el selector del TSS actual, cuya dirección base atributos y límite están en el registro cache hidden asociado.
- Emplea el TSS actual para almacenar el estado del procesador.
- Setea el bit CRO.TS (Task Switch).

Proceso de conmutación de tarea

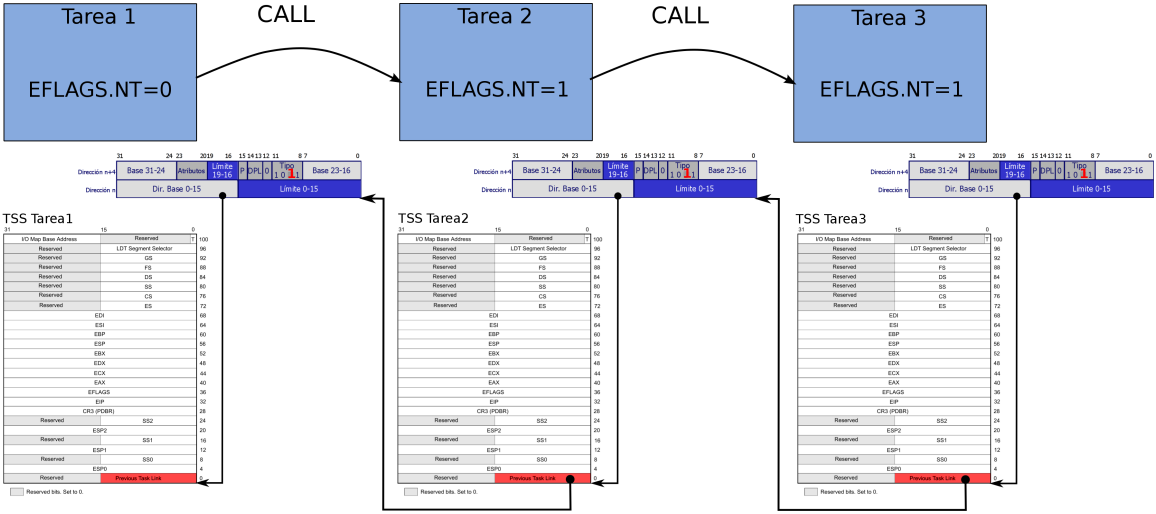
- Reserva el descriptor de TSS y el selector en registros temporarios ya que en TR está el selector del TSS actual, cuya dirección base atributos y límite están en el registro cache hidden asociado.
- Emplea el TSS actual para almacenar el estado del procesador.
- Setea el bit CRO.TS (Task Switch).
- Carga nuevo TR y descriptor.

Proceso de conmutación de tarea

- Reserva el descriptor de TSS y el selector en registros temporarios ya que en TR está el selector del TSS actual, cuya dirección base atributos y límite están en el registro cache hidden asociado.
- Emplea el TSS actual para almacenar el estado del procesador.
- Setea el bit CRO.TS (Task Switch).
- Carga nuevo TR y descriptor.
- Copia a los registros de la CPU el contexto almacenado en el nuevo TSS.

Temario

Funcionamiento de tareas anidadas



Funcionamiento de tareas anidadas

- Cuando se conmuta a una tarea mediante un CALL, una interrupción, o una excepción, el procesador copia el TR de la tarea actual en el campo “Previous Task Link” del TSS de la nueva tarea, y luego de completar el cambio de contexto, setea el bit NT del registro EFLAGS (bit 14).

Funcionamiento de tareas anidadas

- Cuando se conmuta a una tarea mediante un CALL, una interrupción, o una excepción, el procesador copia el TR de la tarea actual en el campo “Previous Task Link” del TSS de la nueva tarea, y luego de completar el cambio de contexto, setea el bit NT del registro EFLAGS (bit 14).
- De este modo si la nueva tarea ejecuta en algún punto la instrucción IRET y el bit NT es ‘1’, el procesador conmuta a la tarea anterior ya que tiene el selector de TSS de la tarea previa almacenado en el campo Previous Task Link del TSS de la tarea en ejecución.

Funcionamiento de tareas anidadas

- Cuando se conmuta a una tarea mediante un CALL, una interrupción, o una excepción, el procesador copia el TR de la tarea actual en el campo “Previous Task Link” del TSS de la nueva tarea, y luego de completar el cambio de contexto, setea el bit NT del registro EFLAGS (bit 14).
- De este modo si la nueva tarea ejecuta en algún punto la instrucción IRET y el bit NT es ‘1’, el procesador conmuta a la tarea anterior ya que tiene el selector de TSS de la tarea previa almacenado en el campo Previous Task Link del TSS de la tarea en ejecución.
- En cambio si la conmutación de tarea se efectúe con un JMP no se afecta el flag NT ni se completa el campo “Previous Task Link”.

Funcionamiento de tareas anidadas

- El procesador utiliza el Bit Busy de un descriptor de TSS para prevenir la reentrancia en una tarea (esto ocasionaría la pérdida de los datos del contexto de ejecución).

Funcionamiento de tareas anidadas

- El procesador utiliza el Bit Busy de un descriptor de TSS para prevenir la reentrancia en una tarea (esto ocasionaría la pérdida de los datos del contexto de ejecución).
- Cuando se avanza en anidamiento de tareas mediante un CALL o una interrupción, este bit debe permanecer seteado, en el descriptor de TSS de la tarea previa.

Funcionamiento de tareas anidadas

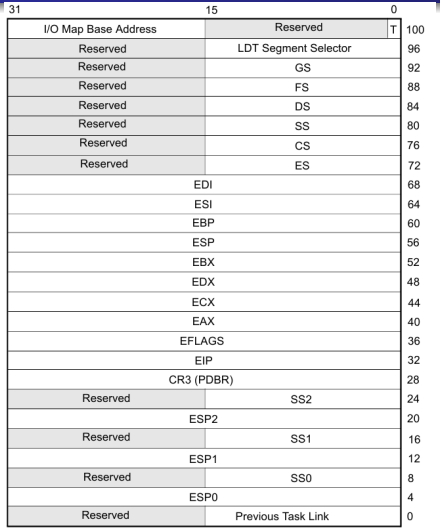
- El procesador utiliza el Bit Busy de un descriptor de TSS para prevenir la reentrancia en una tarea (esto ocasionaría la pérdida de los datos del contexto de ejecución).
- Cuando se avanza en anidamiento de tareas mediante un CALL o una interrupción, este bit debe permanecer seteado, en el descriptor de TSS de la tarea previa.
- El procesador generará una Excepción de Protección General #GP, si se intenta despachar mediante un CALL o una Interrupción una tarea en cuyo TSS está seteado el bit Busy. Esto no ocurre si la la tarea en cuestión se despacha con un IRET ya que es de esperar en esta condición que el bit Busy esté seteado.

Funcionamiento de tareas anidadas

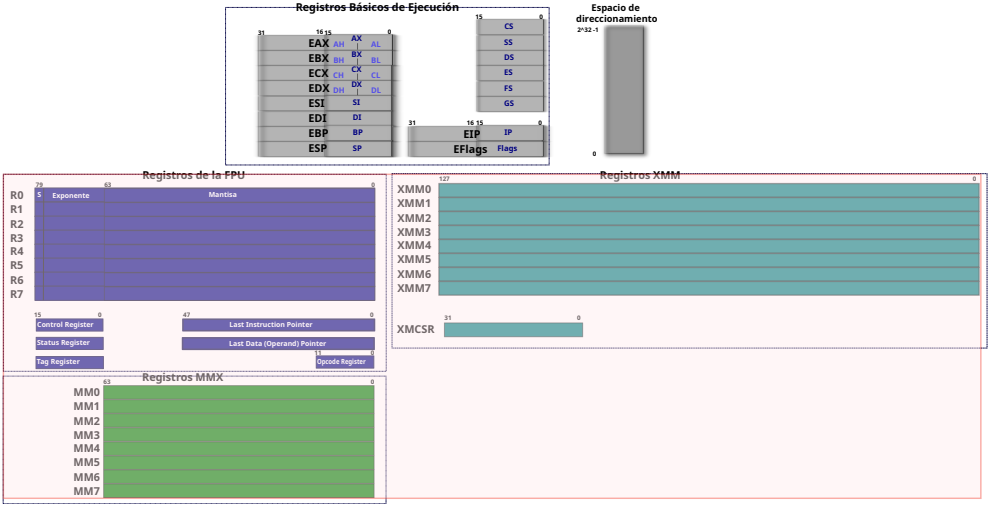
- El procesador utiliza el Bit Busy de un descriptor de TSS para prevenir la reentrancia en una tarea (esto ocasionaría la pérdida de los datos del contexto de ejecución).
- Cuando se avanza en anidamiento de tareas mediante un CALL o una interrupción, este bit debe permanecer seteado, en el descriptor de TSS de la tarea previa.
- El procesador generará una Excepción de Protección General #GP, si se intenta despachar mediante un CALL o una Interrupción una tarea en cuyo TSS está seteado el bit Busy. Esto no ocurre si la la tarea en cuestión se despacha con un IRET ya que es de esperar en esta condición que el bit Busy esté seteado.
- Cuando la tarea ejecuta IRET o bien mediante un JMP despacha una nueva tarea, el procesador asume que la tarea actual finalizará y se debe limpiar el bit Busy en su descriptor de TSS

Temario

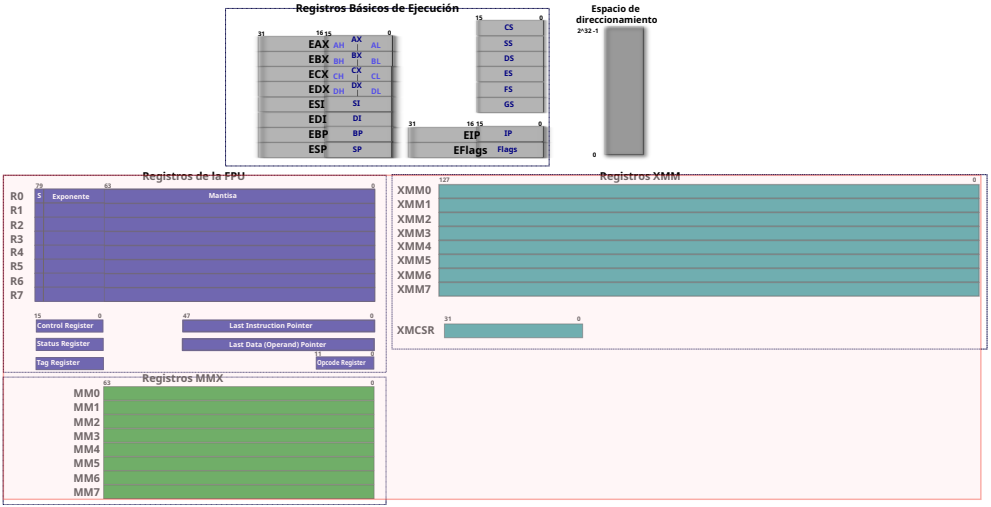
¿Que falta en el TSS?



Modelo Básico de programación



¿Como se resguardan los registros XMM y FPU?



Mecanismo: Excepción #NM (tipo 7)

- Cada vez que el procesador conmuta una tarea mediante cualquiera de los métodos vistos algunos slides atrás, setea el bit **`CR0.TS`**.

Mecanismo: Excepción #NM (tipo 7)

- Cada vez que el procesador conmuta una tarea mediante cualquiera de los métodos vistos algunos slides atrás, setea el bit **`CR0.TS`**.

- Por su parte en la máquina de estados de ejecución, cada vez que va a ejecutar una instrucción que utilice algún registro XMM, o de la FPU o MMX (recordar que son alias de los de la FPU), el procesador chequea **`CR0.TS`** y si es '1'genera una excepción #NM.

Mecanismo: Excepción #NM (tipo 7)

- Cada vez que el procesador conmuta una tarea mediante cualquiera de los métodos vistos algunos slides atrás, setea el bit **`CR0.TS`**.
- Por su parte en la máquina de estados de ejecución, cada vez que va a ejecutar una instrucción que utilice algún registro XMM, o de la FPU o MMX (recordar que son alias de los de la FPU), el procesador chequea **`CR0.TS`** y si es '1'genera una excepción #NM.
- En ese handler se procede a switchear el banco de regsitros.

Mecanismo: Excepción #NM (tipo 7)

- En el handler, hay que limpiar el bit **`CR0.TS`**.

Mecanismo: Excepción #NM (tipo 7)

- En el handler, hay que limpiar el bit **`CR0.TS`**.
- La instrucción FXSAVE permite salvar en forma automática en un bloque de 512 bytes de memoria el contexto FPU/XMM

Mecanismo: Excepción #NM (tipo 7)

- En el handler, hay que limpiar el bit **`CR0.TS`**.
- La instrucción FXSAVE permite salvar en forma automática en un bloque de 512 bytes de memoria el contexto FPU/XMM
- La instrucción FXRSTR recupera del bloque de 512 bytes el contexto FPU/XMM guardado oportunamente

Temario

Uno de los cambios mayores...

Uno de los cambios mayores...

¿Que cambia en Modo IA-32e?

Se mantienen los conceptos de espacio de tareas, estado de tareas, y se deben construir también las estructuras para almacenar los contextos de cada tarea.
Sin embargo, no se mantiene el mecanismo de soporte a la conmutación de tareas propio del Modo Protegido legacy.

Uno de los cambios mayores...

¿Que cambia en Modo IA-32e?

Se mantienen los conceptos de espacio de tareas, estado de tareas, y se deben construir también las estructuras para almacenar los contextos de cada tarea.
Sin embargo, no se mantiene el mecanismo de soporte a la conmutación de tareas propio del Modo Protegido legacy.

Condiciones que generan una excepción #GP

- Si se transfiere el control a un TSS o a una task gate mediante las instrucciones JMP, CALL, INT, o mediante una interrupción de hardware.
- Si se ejecuta IRET con EFLAGS.NT=1.

¿Que pasa con el TSS?

¿Que pasa con el TSS?

- En Modo 64 bits el procesador mantiene el TSS, pero su función ya no es la de almacenar el contexto de la tarea.

¿Que pasa con el TSS?

- En Modo 64 bits el procesador mantiene el TSS, pero su función ya no es la de almacenar el contexto de la tarea.
- Su función ahora es mantener

¿Que pasa con el TSS?

- En Modo 64 bits el procesador mantiene el TSS, pero su función ya no es la de almacenar el contexto de la tarea.
- Su función ahora es mantener
  1. Los valores de RSP para los Niveles de Privilegio 2, 1, y 0, en formato canónico.

¿Que pasa con el TSS?

- En Modo 64 bits el procesador mantiene el TSS, pero su función ya no es la de almacenar el contexto de la tarea.
- Su función ahora es mantener
  1. Los valores de RSP para los Niveles de Privilegio 2, 1, y 0, en formato canónico.
  2. La Tabla de Punteros a Stacks de Interrupciones (IST), punteros expresados también en su formato canónico.

¿Que pasa con el TSS?

- En Modo 64 bits el procesador mantiene el TSS, pero su función ya no es la de almacenar el contexto de la tarea.
- Su función ahora es mantener
  1. Los valores de RSP para los Niveles de Privilegio 2, 1, y 0, en formato canónico.
  2. La Tabla de Punteros a Stacks de Interrupciones (IST), punteros expresados también en su formato canónico.
  3. El Offset al BitMap de E/S.

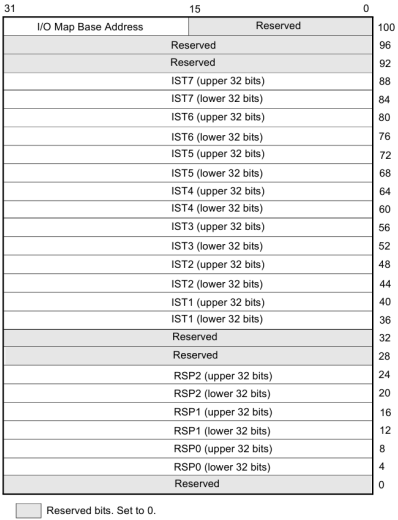
¿Que pasa con el TSS?

- En Modo 64 bits el procesador mantiene el TSS, pero su función ya no es la de almacenar el contexto de la tarea.
- Su función ahora es mantener
  1. Los valores de RSP para los Niveles de Privilegio 2, 1, y 0, en formato canónico.
  2. La Tabla de Punteros a Stacks de Interrupciones (IST), punteros expresados también en su formato canónico.
  3. El Offset al BitMap de E/S.
- El sistema operativo de 64 debe crear al menos un TSS e inicializar el TR con el selector correspondiente a este segmento.

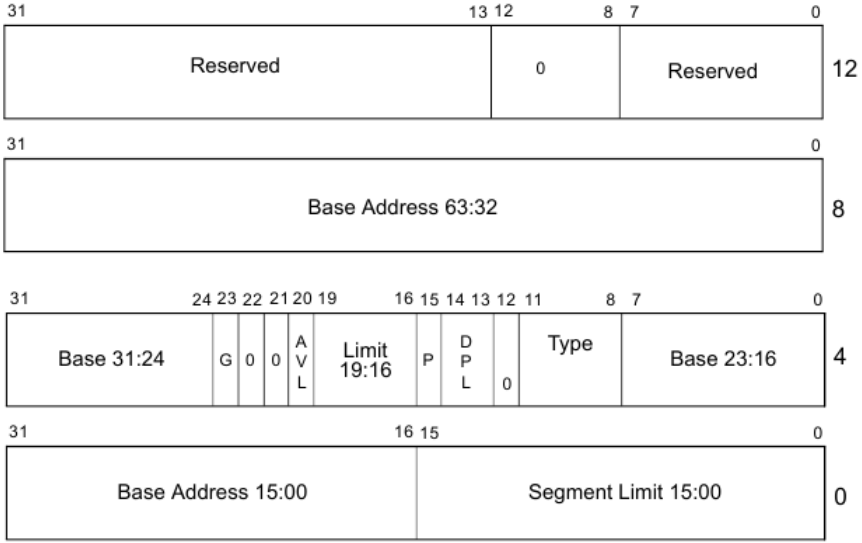
¿Que pasa con el TSS?

- En Modo 64 bits el procesador mantiene el TSS, pero su función ya no es la de almacenar el contexto de la tarea.
- Su función ahora es mantener
  1. Los valores de RSP para los Niveles de Privilegio 2, 1, y 0, en formato canónico.
  2. La Tabla de Punteros a Stacks de Interrupciones (IST), punteros expresados también en su formato canónico.
  3. El Offset al BitMap de E/S.
- El sistema operativo de 64 debe crear al menos un TSS e inicializar el TR con el selector correspondiente a este segmento.
- Este segmento se utilizará tanto para tareas que ejecuten en el sub-modo 64 bits como en el sub-modo compatibilidad.

TSS en Modo IA-32e



Descriptor de TSS en Modo IA-32e



Resumiendo...

En modo 64 Bits la conmutación de tareas no está soportada por el hardware, y por lo tanto debe hacerse por software quedando a cargo del programador de Sistemas....ouch!

Resumiendo...

En modo 64 Bits la conmutación de tareas no está soportada por el hardware, y por lo tanto debe hacerse por software quedando a cargo del programador de Sistemas....ouch!