Atomik Kernel API

kmalloc

Desde el punto de vista del microkernel, a la hora de administrar una porción de memoria libre (entendida como una dirección de comienzo y final) puede ser interesante utilizar un enfoque tipo malloc. Sin embargo, ya que la arquitectura subyacente puede ofrecer la memoria libre de forma discontínua, el propio espacio de direcciones puede definir segmentos con semánticas diversas o incluso puede que queramos especificar qué partes de la memoria administramos así, no podemos confiar en el par malloc/free de la biblioteca estándar de C para que oculte todos estos detalles.

kmalloc ofrece una forma de administrar una región de memoria contigua siguiendo la filosofía de malloc. Dicha API actúa desconociendo la naturaleza de la memoria que va a administrar o cualquier otro detalle de la arquitectura en la que se ejecuta, ofreciendo simplemente funciones para alojar y liberar trozos de memoria dentro de una lista circular alojada en buffer que se le proporciona de antemano.

Tipos, prototipos y macros pueden encontrarse en el fichero de cabecera `<mm/kmalloc.h>`

`struct kas_hdr *kmalloc_init (void *start, void *end)`

Inicializa una porción de memoria para ser administrada por `kmalloc`, comenzando en la dirección `start` y acabando en `end`, siendo `start` la dirección del primer byte de memoria alojable y `end` la del último byte alojable. El resultado es un puntero a una estructura de tipo `struct kas_hdr` conteniendo información sobre la lista circular de trozos de memoria libres y en uso. Este puntero ha de pasársele al resto de funciones de esta API e identifica la región de memoria sobre la que se opera.

Reentrante : sí
Thread-safe : no

`void *kmalloc (size_t size, struct kas_hdr *hdr)`

Intenta alojar un trozo de `size` bytes libres dentro de la región de memoria administrada por `kmalloc` especificada por `hdr`. La función devuelve la dirección de una zona de memoria de `size` bytes lista para ser usada o `NULL` en caso de que no hubiese memoria libre.

La política de selección de trozo es *worst fit chunk* : se busca el trozo libre más grande de toda la lista para evitar una sobrefragmentación de memoria.

Reentrante : sí
Thread-safe : no

`void *kmalloc_fast (size_t size, struct kas_hdr *hdr)`

Intenta alojar un trozo de `size` bytes libres dentro de la región de memoria administrada por `kmalloc` especificada por `hdr`. La función devuelve la dirección de una zona de memoria de `size` bytes lista para ser usada o `NULL` en caso de que no hubiese memoria libre.

La política de selección de trozo es *first fit chunk* : se busca el primer trozo de libre de tamaño mayor o igual que `size` bytes, siendo esta más rápida que `kmalloc`.

Reentrante : sí
Thread-safe : no

`void kfree (void *ptr, struct kas_hdr *hdr)`

Libera la memoria apuntada por `ptr` dentro de la región de memoria administrada por `kmalloc` especificada por `hdr`. Si ptr es `NULL` no se hace nada y la memoria permanece intacta.

Como esta función hace comprobaciones de seguridad e integridad de la memoria liberada, emitiendo potencialmente mensajes de error por la consola del sistema, en general no se puede decir que sea reentrante. En todo caso es condicionalmente reentrante : sólo se asegura que no se modifican variables globales u objetos del sistema si `ptr` es `NULL` o un puntero administrado por `kmalloc` y la memoria no está corrupta (caso ideal).

Reentrante : no (leer explicación)
Thread-safe : no