Programación de Sistemas
Proyecto Parcial

1. Objetivo

El objetivo de este proyecto es practicar el uso de C, versionamiento de código, administración de memoria dinámica y uso punteros.

2. Descripción

La administración de memoria es una de las partes más importantes del software de sistema (por ejemplo, en hipervisores, sistemas operativos, motores de base de datos, interpretes, etc). De esto depende el rendimiento y eficiencia del software y de las aplicaciones que corren sobre dicho software.

En este curso hemos aprendido el uso las funciones malloc/calloc/realloc. El problema es que estas funciones no son rápidas, ya que implican llamadas al sistema. Si un programa hace muchas llamadas a estas funciones, es posible que el rendimiento se vea afectado. Para evitar esto, se inventó el **slab allocator**.

En un slab allocator, se pre-asignan una cantidad determinada de un tipo de objeto (por ejemplo, pre-asignamos 100 objetos cada uno de 1500 bytes). A cada uno de estos objetos los llamamos **slabs.** De esta forma, cuando el programa requiera memoria para ese tipo de objeto, simplemente retornamos un puntero a uno de los slabs, en vez de llamar a malloc() por cada objeto que queramos crear. **Un slab allocator devuelve objetos del mismo tipo.** Si necesitamos objetos de otro tipo, necesitamos crear un nuevo allocator.

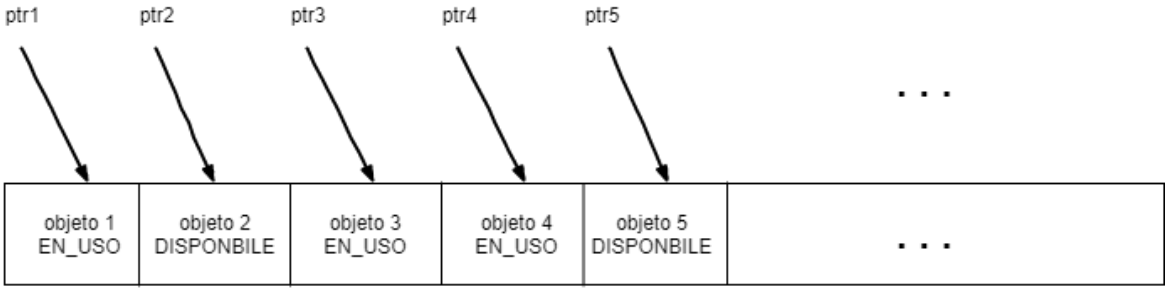

Ilustración 1 Slab Allocator

Por ejemplo, con el slab allocator mostrado arriba, si programa desea crear un objeto, el allocator le devolvería el puntero ptr2. Note que todos los elementos ya fueron creados anteriormente, se usen o no. El slab allocator es usado mucho en el kernel de Linux. **En este proyecto implementarán un slab allocator para espacio de usuario**.

Un slab allocator debe saber algunas cosas antes de poder funcionar. Estas son:

a. Nombre: el nombre del allocator
b. Cantidad de objetos: la cantidad de objetos a ser creados cuando se inicialice el allocator
c. Tamaño del objeto: el tamaño de cada objeto
d. Qué hacer cuando se crea el objeto (función de inicialización)
e. Qué hacer cuando ya se van a eliminar el allocator (función de destrucción).
f. Región de memoria donde se guardan los objetos. El tamaño de la región de memoria a reservar es **tamaño_objeto * cantidad_objetos**
g. Cantidad de slabs en uso.
h. Para cada slab, se debe saber la dirección donde está el objeto del usuario, y si está en uso o no.

El allocator tiene unas pocas funciones básicas, mostradas abajo.

a. crear_cache: inicializa el slab allocator, y asigna la memoria para los objetos.
b. obtener_cache: devuelve un objeto DISPONIBLE del cache. Esto hará que el objeto se marque como EN_USO. Si todos los objetos pre-creados ya están en uso, y se envia la bandera crecer, el allocator duplicara la cantidad de objetos.
c. devolver_cache: devuelve el objeto al cache. Esto hará que el slab correspondiente se marque como DISNPONIBLE.
d. destruir_cache: libera toda la memoria asignada por el allocator.

En este proyecto Ud. implementará un slab allocator que pueda ser configurado con cualquier tipo de objeto.

3. Implementación

Para este proyecto, a Ud. se le provee un repositorio y le hará **fork.** Su repo debe llamarse **proyecto_parcial**:

https:// bitbucket.org/ProgramacionSistemasEM/proyecto_parcial_2019_1.git

En el repo, encontrará dos archivos cabecera: slaballoc.h y objetos.h. En el archivo **slaballoc.h** están las estructuras que vamos a usar. Estas estructuras son:

a. SlabAlloc → contiene la información del allocator
b. Slab → contiene la información de cada slab.

Las estructuras se muestras a continuación:

```
typedef struct slab{
   void *ptr_data;
   void *ptr_data_old;
   unsigned int status;
} Slab;


typedef struct slaballoc{
    char *nombre;
    void *mem_ptr;
    unsigned int tamano_cache;
    unsigned int cantidad_en_uso;
    size_t tamano_objeto;
    constructor_fn constructor;
    destructor_fn destructor;
    Slab *slab_ptr;
} SlabAlloc;
```

La relación entre estas estructuras se muestra en la ilustración 2. El objeto SlabAlloc tiene un puntero (mem_ptr) a la región de memoria dinámica donde se crean los objetos. A su vez, SlabAlloc tiene un puntero a un arreglo de objetos de tipo Slab, cada uno de los cuales tiene dos campos: un puntero a la dirección (dentro de la gran región apuntada por mem_ptr) donde esta guardado el i objeto, y una bandera si dicho objeto está siendo usado por alguien o no.

La función **crear_cache** debería crear todo lo visto en el gráfico, e inicializarlo correctamente. Cuando se crear los objetos (objeto 1, objeto 2, etc.) la función debería llamara la función constructor(). La función recibe el tamaño del objeto que podrá proveer el allocator, así como las funciones constructora y destructora del objeto.

La función **obtener_cache** devuelve el puntero ptr_data del slab correspondiente, y marca el slab como EN_USO. Si todos los slabs están en uso, y la bandera crecer es igual

a 1, su allocator debería incrementar el número de slabs al doble del tamaño anterior. La función **devolver_cache** busca el slab asociado al objeto, y lo marca como DISPONIBLE. La función **destruir_cache** debe liberar toda la memoria asignada (SlabAlloc, Slabs y el bloque donde están los objetos). Antes de liberar el bloque donde están los objetos, debe llamar a la función destructor, sobre cada uno de los objetos.

En el archivo **objetos.h**, Uds. pueden declarar los objetos así como sus constructores y destructores que podremos usar con el allocator. Para este proyecto, declaren tres tipos de objetos distintitos. Se provee un objeto de ejemplo.

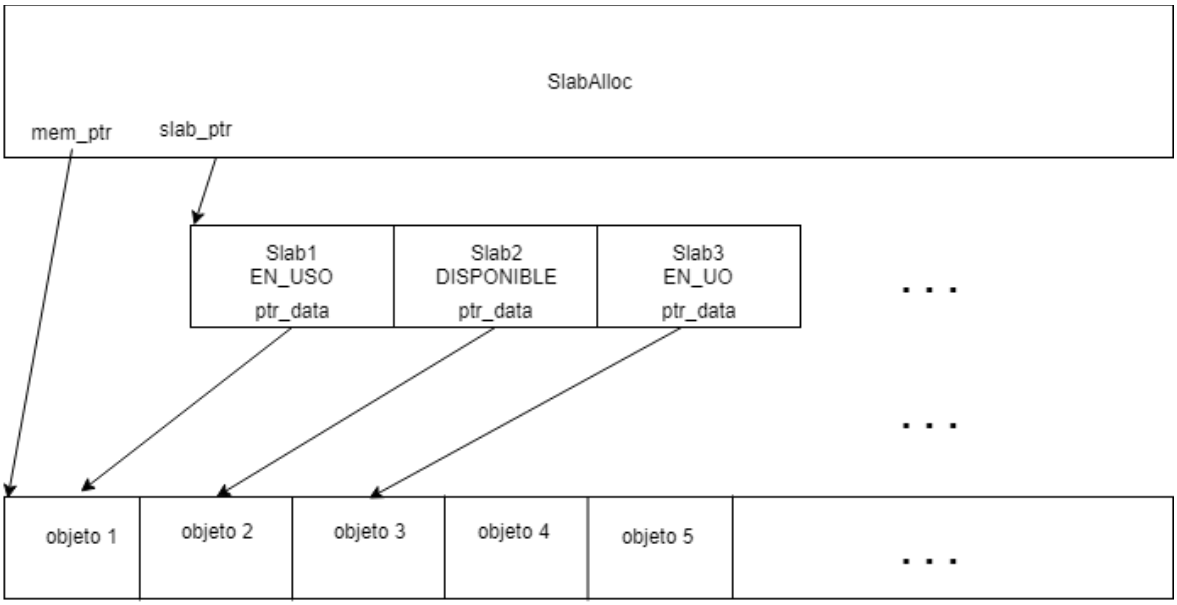

Ilustración 2 SlabAlloc, Slab, objetos de usuario.

4. Crecimiento del cache

Cuando nos piden un elemento, y todos los elementos del caché están en uso, debemos hacer crecer el cache. Para esto, duplicamos el tamaño del cache (usamos realloc). Puede ver un ejemplo abajo en la ilustración 3. El allocator empezó con tres slabs, y luego lo hizo crecer a 6 slabs (los slabs y cache nuevos están en rojo). Ud. debe inicializar estos slabs acordemente.

Hay que tener especial cuidado al hacer crecer el cache. Note que en la ilustración 3, el cache empieza en la dirección 1000. Cuando se llama **realloc**, esta función nos devolverá el bloque ampliado, pero NO necesariamente desde la misma dirección de memoria. Entonces, las referencias ptr_data pueden ser invalidas (están apuntando a la dirección vieja). Por ejemplo, asumiendo que cada objeto tenga 100 bytes de tamaño, si **realloc**

nos devuelve el bloque ampliado desde la dirección 5000, tendríamos los mostrado e la ilustración 4. Fíjese que los punteros están apuntando a la dirección incorrecta!

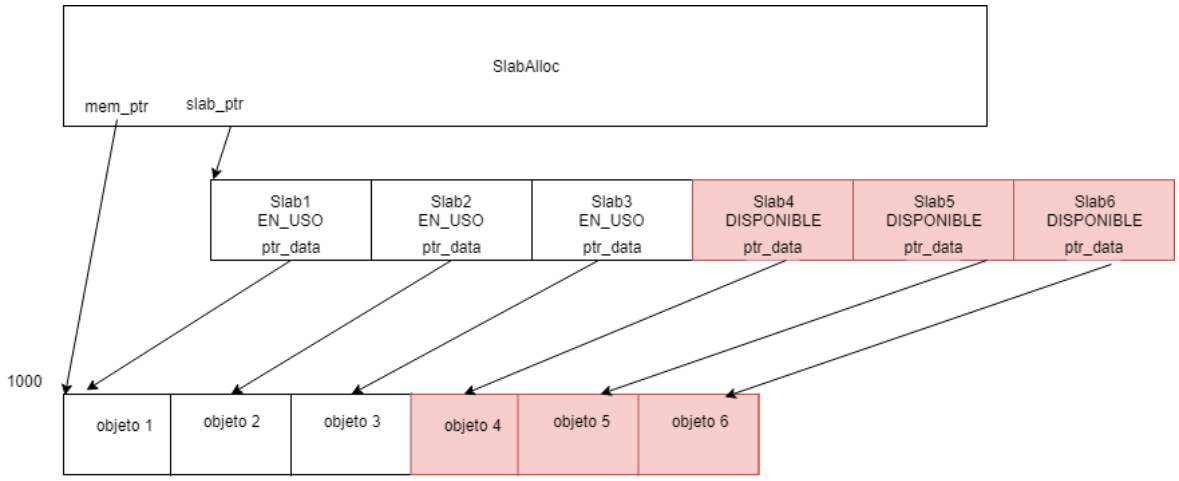

Ilustración 3 Haciendo crecer el allocator.

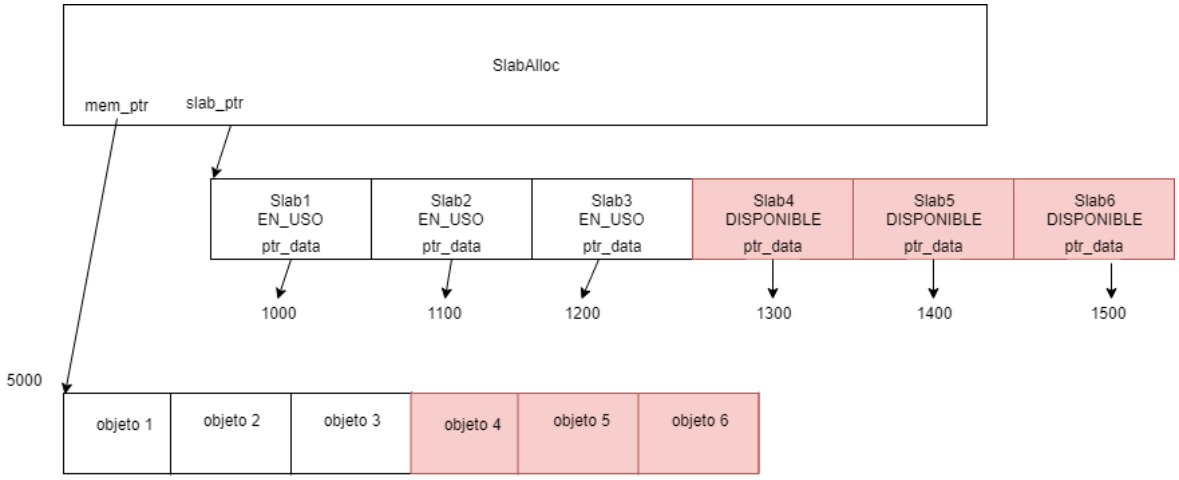

Ilustración 4 Crecimiento incorrecto.

Por lo tanto, cuando hacemos crecer el cache, debemos recorrer todos los Slabs, y hacerlos apuntar a la dirección correcta (en este ejemplo serian las direcciones 5000, 5100, 5200, 5300, 5400, 5500). Para conveniencia del usuario, la estructura Slab tiene un campo llamada **ptr_data_old**, que guardará la dirección donde estaba anteriormente el objeto. Vea la ilustración 5.

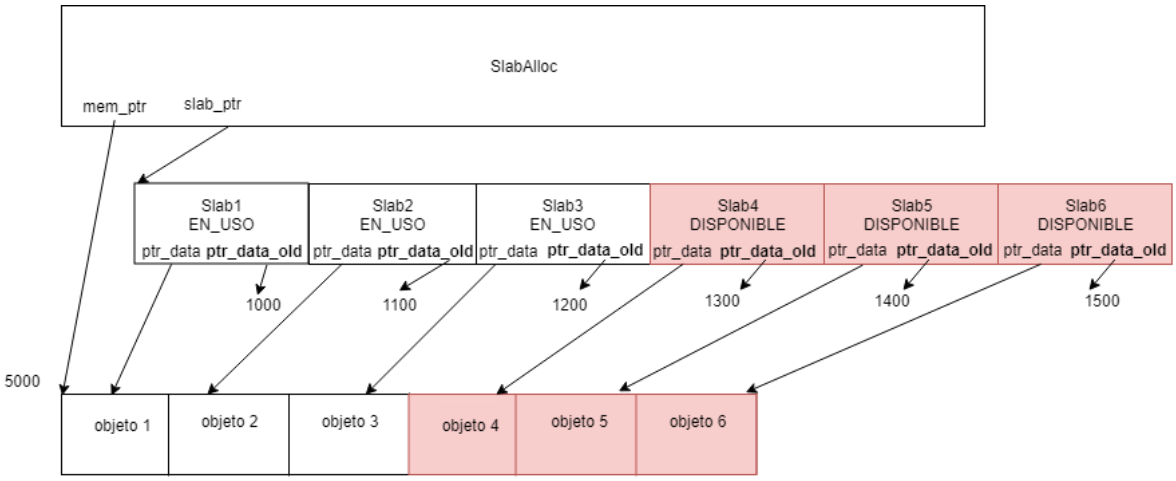

Ilustración 5 Crecimiento correcto.

5. Repositorio

Archivos del repo:

a. bin -> ejecutable (llamado **prueba**)
b. include
    i. slaballoc.h → interfaz del allocator. **NO MODIFICAR**
    ii. objetos.h → sus objetos para el allocator los declaran aquí
a. obj → archivos objeto
b. src -> archivos de código fuente. Los constructores/destructores deben implementarse en un archivo .c con el mismo nombre del objeto.
    i. Pueden escribir su programa de prueba en prueba.c. **Este archivo será reemplazado con el programa de prueba del profesor** al momento de calificar.
c. Makefile → Su makefile debe generar el ejecutable llamado prueba. **TODOS los archivos .c deben compilarse por separado, y luego ser enlazados.**

6. Calificación

Su implementación será probada con un programa de prueba escrito por el profesor.

Si su programa no compila por ERRORES en su código o no tiene un MAKEFILE o el MAKEFILE tiene erros y no genera el ejecutable, no tendrán crédito para el proyecto.

Luego, se verificará lo siguiente (puntos restados)

a. Existe un *segmentation fault*.	Se califica sobre la mitad.
b. Existen *warnings* al compilar el proyecto	-5/warning
c. Los archivos no son compilados separadamente	-20
d. El proyecto no fue organizado por carpetas	-10
e. No existe manera de borrar los archivos generados	-5

7. Entrega

El proyecto deberá ser entregado hasta el **28 de Junio de 2019, 22:00**. Se deberá proveer el repositorio de Git del proyecto con todos los archivos necesarios.

Referencias

Bonwick, J. (1994). The Slab Allocator: An Object-Caching Kernel. *USENIX Summer 1994 Technical Conference*. Boston, Massachusetts, EEUU.

Love, R. (2011). *Linux Systems Programming.* Cambridge: O'Reilly.

Randal E. Bryant, D. R. (2015). *Computer Systems A Programmer's Perspective.* Boston: Prentice Hall.