Proyecto #2:
Llamadas al sistema para manejo de memoria
versión corregida el 3 de noviembre de 2016

1. Objetivos

- Conocer el uso de las llamadas al sistema para manejo de memoria del kernel de Linux.
- Implementar los algoritmos de ajuste de memoria vistos en clase.
- Implementar un mecanismo de manejo de memoria con lista de bloques libres.

2. Proyecto

El objetivo general de este proyecto es reimplmentar las funciones de manejo de memoria de la biblioteca estándar del lenguaje de programación C, utilizando una lista simplemente enlazada para el manejo de los bloques libres.

Se provee junto a este enunciado un archivo llamado `custom_malloc.h`, el cual contiene las redefiniciones de las llamadas de manejo de memoria de C. En este archivo es donde se debe realizar la implementación del proyecto. Adicionalmente se provee un archivo llamado `main.c`, el cual pueden modificar a conveniencia para poder ejecutar y probar su proyecto. El proyecto debe compilarse con el archivo `Makefile` incluido, el cual puede ser modificado a conveniencia.

La evaluación del proyecto será realizada sustituyendo el archivo `main.c` incluido por otro archivo del mismo nombre provisto por el grupo docente.

3. Requerimientos

Su proyecto debe cumplir con los siguientes requerimientos:

1. Debe reimplementar completamente el funcionamiento de las funciones malloc, calloc y free, como se detalla más adelante.
2. Debe reimplementar el funcionamiento de la función realloc como se detalla más adelante.
3. Debe manejar la memoria libre disponible usando un arreglo estático de 10000 posiciones.
4. Debe implementar los algoritmos de primer, mejor, peor y siguiente ajuste para manejar los bloques libres y realizar la asignación de memoria.
5. Debe utilizar las herramientas de compilación condicional de C para escoger cual algoritmo utilizar durante la compilación de su proyecto.
6. Debe implementar una función que imprima el contenido del arreglo de bloques libres.

A continuación se detallan cada uno de los requerimientos.

3.1. Memoria disponible para asignar

En el archivo adjunto `custom_malloc.h` se define una función llamada `set_initial_memory()`, la cual se encarga de solicitar un bloque de memoria de 300 Megabytes de tamaño, utilizando la función `sbrk` definida por el kernel de Linux.

Este bloque queda definido por las variables `mem_start` y `mem_end` las cuales representan la dirección inicial y final de dicho bloque (la dirección final se exluye del bloque, es decir, las direcciones válidas van de `mem_start` a `mem_end` -1). Las funciones malloc, calloc y realloc deberán entonces retornar direcciones que se encuentren dentro de este bloque de memoria.

3.2. Lista de bloques libres

Todas las funciones de manejo de memoria se fundamentan en un arreglo estático de 10000 posiciones que funciona como la lista de bloques libres. Inicialmente debe existir un solo bloque libre que representa la totalidad de la memoria disponible.

A medida que se utilice el bloque de memoria se deberá actualizar este arreglo, agregando o colapsando bloques libres a medida que se asigne o libere la memoria. Si dos bloques del arreglo representan bloques libres contiguos (la dirección final de un bloque es inmediatamente adyacente a la dirección inicial del siguiente), se deberá colapsar ambos bloques en uno solo que represente el bloque libre completo.

3.3. Reimplementación de malloc

La función `malloc` debe recibir una cantidad de bytes a asignar y debe retornar un apuntador al inicio de un bloque de memoria del tamaño solicitado. El bloque a retornar debe ser asignado utilizando alguno de los bloques libres disponibles. El bloque libre a utilizar debe ser escogido mediante alguno de los algoritmos de ajuste de memoria (primer, mejor, peor o siguiente ajuste). La elección del algoritmo de ajuste a utilizar será realizada en tiempo de compilación como se describe más adelante.

3.3.1. Casos borde

Ejecutar la función `malloc` con un tamaño igual a cero (0) debe retornar el valor `NULL` sin afectar la lista de bloques libres.

Si el algoritmo de ajuste utilizado no es capaz de encontrar un bloque libre de tamaño suficiente para realizar la asignación de memoria solicitada, entonces `malloc` debe retornar `NULL` sin afectar la lista de bloques libres.

3.4. Reimplementación de calloc

La función `calloc` debe recibir una cantidad de elementos a asignar y el tamaño en bytes de cada elemento, y debe retornar un apuntador al inicio de un bloque de memoria del tamaño solicitado (elementos $\times$ tamaño). El comportamiento general de la función es equivalente al de `malloc` en lo que se refiere a la búsqueda del bloque libre a asignar. Sin embargo, `calloc` debe inicializar en cero (0) el bloque de memoria a retornar, mientras que `malloc` retorna el bloque directamente, sin inicializarlo.

3.4.1. Casos borde

Ejecutar la función `calloc` con cualquier parámetro igual a cero (0) debe retornar el valor `NULL` sin afectar la lista de bloques libres.

Si el algoritmo de ajuste utilizado no es capaz de encontrar un bloque libre de tamaño suficiente para realizar la asignación de memoria solicitada, entonces `calloc` debe retornar `NULL` sin afectar la lista de bloques libres.

3.5. Reimplementación de realloc

La función `realloc` debe recibir un apuntador a un bloque de memoria y un tamaño. Esta función debe entonces modificar el bloque representado por el apuntador obtenido por parámetro de manera que su tamaño final corresponda con el tamaño pasado por parámetro. Se distinguen cuatro casos posibles:

1. El nuevo tamaño del bloque es menor que su tamaño original.

   En este caso se debe liberar espacio al final del bloque. Este espacio liberado debe ser agregado a la lista de bloques libres. Puede que sea necesario colapsar nodos de la lista de bloques libres.

2. El nuevo tamaño es igual al tamaño original.

   En este caso `realloc` retorna sin realizar otro procesamiento.

3. El nuevo tamaño es mayor al tamaño original.

   En este caso pueden presentarse dos escenarios:

   a) Si hay un bloque libre inmediatamente adyacente al bloque de datos, entonces se deben tomar parte de ese espacio libre para anexarlo al bloque de datos original, de forma que este último tenga como tamaño final el tamaño solicitado a la función.

   b) Si no hay un bloque libre inmediatamente adyacente, o si hay un bloque libre, pero este no tiene tamaño suficiente para extender el bloque de datos, entonces se debe buscar un bloque libre utilizando el agloritmo de ajuste que tenga espacio suficiente para contener el tamaño final del bloque de datos. Luego, el contenido del bloque de datos original debe ser copiado al nuevo bloque de datos (el segmento del bloque nuevo que fue extendido al bloque original no tiene que ser inicializado). Finalmente, el bloque de datos original debe ser agregado a la lista de bloques libres.

4. El nuevo tamaño es cero (0).

   Este caso es equivalente a realizar una llamada a la función `free`.

La función `realloc` retorna la dirección inicial del bloque de datos, la cual es el mismo apuntador pasado como parámetro para los primeros dos casos anteriores, o la dirección inicial del bloque movido en el tercer caso. `realloc` debe retornar el valor `NULL` en el cuarto caso mencionado.

La implementación de `realloc` implica llevar un seguimiento de alguna manera a los bloques que han sido asignados, en particular la dirección inicial y el tamaño asignado al mismo. Cualquier bloque asignado con `malloc`, `calloc` o `realloc` puede ser modificado con `realloc`.

3.5.1. Casos borde

Ejecutar la función `realloc` con un apuntador `NULL` y un tamaño mayor a cero (0) es equivalente a realizar una llamada a `malloc` para el mismo tamaño. Por otra parte, si el apuntador es `NULL` y el tamaño es cero (0), entonces `realloc` debe retornar `NULL` sin realizar procesamiento adicional.

3.6. Reimplementación de free

La función `free` recibe un apuntador asignado con `malloc`, `calloc` o `realloc`, y se encarga de agregar el bloque de datos señalado por ese apuntador a la lista de bloques libres.

3.7. Consideraciones sobre la memoria y las llamadas realloc y free

Cualquier apuntador que se encuentre dentro del segmento de memoria definido en la Sección 3.1 puede ser utilizado como parámetro de las funciones `realloc` y `free`. Estas funciones deben entonces verificar que el apuntador que han recibido se encunentre efectivamente dentro de un bloque de datos válido, es decir, que se encuentre dentro de los límites de la memoria utilizable, y que además no pertenezca a un bloque libre. Si cualquiera de estas funciones recibe un apuntador no válido como parámetro, entonces debe lanzar una señal SIGSEGV al mismo proceso (utilize las llamadas al sistema `kill` y `getpid`) para esto).

3.8. Compilación condicional de los algoritmos de ajuste

Su proyecto deberá implementar los cuatro algoritmos de ajuste vistos en clase para hacer uso de la lista de bloques libres. Sin embargo, al compilar su proyecto este debe utilizar uno solo de estos algoritmos, siendo posible determinar cual algoritmo utilizar en tiempo de compilación.

Para lograr esto, los algoritmos deben estar implementados dentro de las directivas del preprocesador de C `#ifdef` y `#endif`. Para decidir cual algoritmo utilizar se debe pasar la opción `-D` (menos d mayúsculas) al compilador seguida del nombre del algoritmo a utilizar (`FIRST_FIT`, `BEST_FIT`, `WORST_FIT` ó `NEXT_FIT`) en la linea de compilación. Por ejemplo:

```
gcc -o proyecto2 -D BEST_FIT main.c
```

Su proyecto debe verificar que se indique solamente un algoritmo en la linea de compilación. Si no se especificó un algoritmo, o se especificó más de uno, entonces debe usar la directiva `#error` del preprocesador de C para indicar el correspondiente error.

4. Entregables

Deberá incluir lo siguiente al entregar su proyecto:

- Todos los códigos fuente necesarios para compilar los ejecutables de su proyecto, debidamente comentados e identificados.
- Un *Makefile* que permita compilar su proyecto. Puede utilizar el *Makefile* incluido como punto de partida.
- Un documento de texto que contenga los nombres completos, cédulas de identidad, correos electrónicos y secciones de teoría de los desarrolladores del proyecto.

5. Documentación

El manual del sistema operativo Linux contiene entradas para todas las funciones mencionadas. Utilice los siguientes comandos en una consola para obtener más información:

- `man malloc`
- `man calloc`
- `man realloc`
- `man free`
- `man sbrk`
- `man -s2 kill`
- `man -s7 signal`
- `man getpid`

Consideraciones

1. El proyecto **DEBE** ser entregado el día jueves 10/11/2016 antes de la medianoche. Enviar el proyecto a la dirección de correo sistoperucv@gmail.com .
2. La revisión del proyecto será llevada a cabo el día viernes 11/11/2016 en el Laboratorio ICARO.
3. El proyecto deberá ser realizado en el lenguaje de programación C, en sistemas GNU/Linux y únicamente con el compilador GCC.
4. El proyecto puede ser realizado de manera individual o en grupos de máximo dos (2) personas.
5. Las copias serán penalizadas con la nota mínima para todos los involucrados, además de otras sanciones que contemple la ley de universidades y el reglamento de evaluaciones de la Facultad de Ciencias.