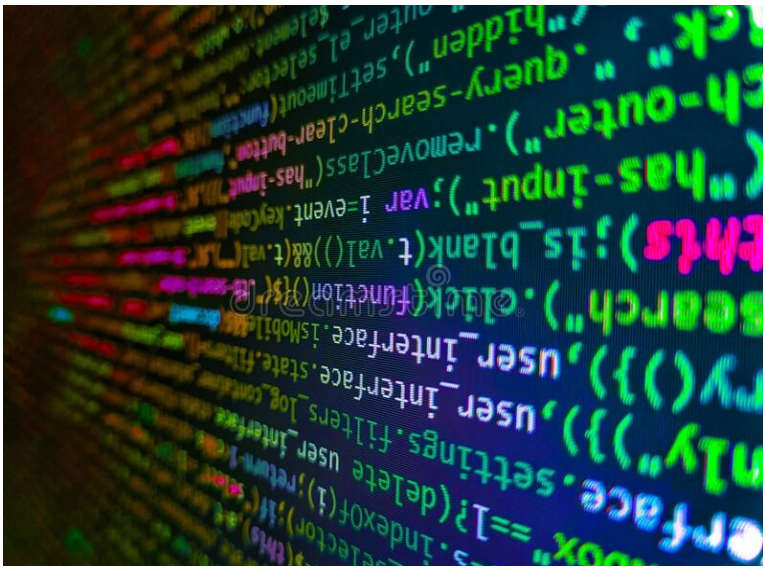
Sistemas Operativos

Práctica 4 – Gestión de memoria dinámica para procesos

Semestre de Primavera 2023



Jose Maria Fernández Paredes
Víctor Sort Rubio

Jose Maria Fernández
Víctor Sort

ÍNDICE

1. INTRODUCCIÓN

En esta última práctica se tenía cómo objetivo la implementación propia de la gestión de memoria dinámica en un proceso a través de la implementación de las funciones de la librería estándar malloc, free, calloc y realloc.

Para ello, se partía de una versión básica dada que se ha ido mejorando poco a poco. Se entregan juntamente con este informe tres códigos. En el primero, se implementan las funciones de manera básica y en el segundo se mejoran los métodos search_available_space para seleccionar el espacio más óptimo y el free para juntar bloques. En el último código, que era opcional, se mejoran el malloc y el realloc para la partición de bloques.

2. IMPLEMENTACIÓN Y PRUEBAS REALIZADAS

2.1 Implementación first_fit

Para implementar el free, incialmente se comprueba que el puntero pasado por parámetro no sea nulo e se inicializa un p_meta_data con la supuesta posición de inicio con la línea:

```
p_meta_data meta_data = (p_meta_data)(ptr - SIZE_META_DATA);
```

Luego, tras comprobar que el valor magic del p_meta_data es correcto, simplemente se pone el valor de available a 1, para indicar que está libre.

Para implementar el calloc, tras comprobar que el número de elementos o su tamaño pasado por parámetro no es menor o igual a 0, simplemente se llama a malloc con el tamaño total. Luego se inicializan todos los datos a 0 con la siguiente línea, dónde ptr es el puntero que apunta al inicio de los datos y tsize el tamaño total, producto de los parámetros pasados por argumento:

```
memset(ptr, 0, tsize);
```

Para implementar el realloc, tras comprobar que el puntero no sea nulo (en caso contrario se llama a malloc y se termina) o el tamaño pasado por argumento no sea menor o igual a 0, se cambia el valor del tamaño para que sea múltiplo de 8 e se inicializa un p_meta_data con la supuesta posición de inicio, comprobando que su valor magic sea correcto.

Si el tamaño pasado por argumento es menor al del p_meta_data no hace falta hacer nada. En caso contrario, se llama a malloc para buscar o generar un nuevo bloque de memoria, y se copian los datos con la siguiente linea, dónde p es un puntero a la nueva posición, ptr un puntero a la posición antigua y before_size_bytes el tamaño del antiguo p_meta_data:

```
memcpy(p, ptr, before_size_bytes);
```

2.2 Implementación best_fit

El método search_available_space recorría los p_meta_data hasta encontrar uno disponible con el suficiente espacio de memoria. Para mejorar-lo, se ha hecho que no se conforme con ese, sino que los acabe de recorrer todos y seleccione el que tiene un espacio mayor o igual al pedido, pero con menos espacio sobrante.

Para mejorar el free, después de poner el atributo available a 1, se mira si el bloque anterior también está disponible i en caso afirmativo, y si el valor magic es correcto, se juntan. Para juntar-

se, lo que se hace es ignorar los datos del p_meta_data acabado de liberar y conectar el anterior y el posterior. En el caso de que no tenga posterior, se marca que ese es el último bloque. También se incrementa el tamaño de bytes del anterior bloque con el tamaño del bloque liberado, teniendo en cuenta el espacio para guardar los atributos del p_meta_data.

Luego se hace un proceso casi idéntico en el caso de que tenga un bloque posterior libre, con la diferencia que el p_meta_data que va ahora a “desaparecer”, o del cual vamos a perder los datos, es del posterior, y no del recién aliberado.

2.3 Implementación best_fit_opcional

Para terminar de perfeccionar el malloc, en el caso de que se esté reusando un p_meta_dato liberado, se comprueba si tiene un espacio de memoria superior al requerido y se crea un nuevo p_meta_data con las siguientes lineas:

```
size_t remaining_size = meta_data->size_bytes - size_bytes – SIZE_META_DATA;

if((int)remaining_size > 0){

p = (void *) meta_data;

p_meta_data meta_data_sobrante = (p_meta_data) (p + size_bytes + SIZE_META_DATA);
```

Luego, de manera similar a como en el punto 2.2, se ha relacionado el nuevo p_meta_dato con el antiguo y el posterior (si es que tiene) y se han actualizado los valores del tamaño.

En el caso del realloc, si se pide modificar el espacio de memoria reservado de un p_meta_data en un tamaño inferior, se hace el mismo proceso. El código en este caso es, excepto en pequeños momentos, totalmente igual al implementado en la mejora del malloc.

2.4 Pruebas realizadas

La justificación de que no se hayan explicado las pruebas realizadas en cada código es porque han sido siempre las mismas en todos los casos:

Inicialmente, se creo un archivo txt llamado text.txt con diferentes palabras, para poder probar el grep, y se ubicó dentro un directorio llamado P4SO en el escritorio.

Tal y como se explico en el guión de la práctica, generamos el ejecutable asociado al archivo que contiene las funciones de la librería estándar, gneramos una librería dinámica asociada a él e indicamos, a través de una variable de entorno, que es necesario cargar esta librería dinámica antes que cualquier otra librería.

Luego, se probaron los comandos grep, find, ls y ps aux, además de la aplicación exemple dada:

```
(base) victor@molsut:~$ cd Escriptori/P4SO

(base) victor@molsut:~/Escriptori/P4SO$ gcc -O -shared -fPIC malloc_first_fit.c
-o malloc_first_fit.so

(base)                  victor@molsut:~/Escriptori/P4SO$                  export
LD_PRELOAD=$PWD/malloc_first_fit.so

(base) victor@molsut:~/Escriptori/P4SO$ ./exemple

Malloc 40 bytes
```

```
0
1
2
3
4
5
6
7
8
9
Free 40 bytes
(base) victor@molsut:~/Escriptori/P4SO$ ls
(base) victor@molsut:~/Escriptori/P4SO$ ps aux
(base) victor@molsut:~/Escriptori/P4SO$ ps aux | grep 'victor'
(base) victor@molsut:~/Escriptori/P4SO$ find . -name "*.so"
 (base) victor@molsut:~/Escriptori/P4SO$ grep 'Hola' text.txt
(base) victor@molsut:~/Escriptori/P4SO$ ./exemple
```

En todos los códigos y casos el funcionamiento es el esperado. No incluyo la salida por pantalla del resto, pues al hacer prints cada vez que se produce un Malloc, Calloc, Realloc o Free la salida es muy extensa, cosa que nos ha hecho ver la gran cantidad de usos que se hace de estas funciones al ejecutar un proceso simple. Además, es muy fácil comprobar por el corrector que los códigos funcionan adecuadamente.

3. CONCLUSIONES Y VALORACIONES PERSONALES

Cómo se ha visto en el punto anterior, hemos sido capaces de implementar y perfeccionar todas las funciones de la librería estándar que se nos pedían y se ha comprobado que funcionan correctamente. Por esta parte pues, estamos muy satisfechos con nuestros resultados.

Además, realmente creemos que la realización de esta práctica nos ha aportado mucho conocimiento referente a todos los conceptos con los cuales hemos trabajado.

De cara a un aspecto a mejorar, en la implementación del best_fit_opcional se necesitaba un cast de size_t a int. Tardamos dos días en encontrar el error y nos consta que no fuimos los únicos, así que si es algo recurrente, estaria bien que se comentara en el guión de la práctica.

Finalmente, creemos que hemos trabajado muy bien en equipo, que nos hemos ayudado mutuamente y esforzado para aprender el temario que tocaba, a la vez que sinceramente nos lo hemos pasado bien programando el código. En resumen, y en nuestra opinión, hemos hecho una muy buena práctica.