Sistemas Operativos I Informe Práctica 4

28 de mayo de 2020

Autores y NIUB:
Maria Isabel González Sánchez 20221913
Oriol Saguillo González 20150502

Grupo: C00

Índice

Aclaración: las pruebas del malloc_simple contienen *printfs* de la forma: *"Malloc 1024 bytes"* o *"Free at ba4db248: 120 bytes"*. En las pruebas del malloc_split_and_merge, estarán comentados para ver mejor los resultados de *ls*, *cp* y *find*.

1. malloc_simple: *find*

En esta prueba, en la carpeta que contiene el programa, buscaremos archivos .c. Para ello, después de hacer *make*, introducimos la comanda *find* tal que:

maribel@maribel-UX430UAR: /Documentos/So1/Lab/Practica4/codi/mallocSimple$ find -type f -name "*.c"

Malloc 552 bytes

Malloc 120 bytes

...

./malloc_simple.c

Free at b64d7a80: 280 bytes

Malloc 16 bytes

...

Free at b64cf4e8: 32816 bytes

./exemple.c

Free at b64d7cf0: 280 bytes

Además podemos observar que se realizan tanto *callocs* como *reallocs*, gracias a las gestiones del código y a los *printfs* aclaratorios:, como:

Fem un calloc de 1 bytes 128 vegades

Malloc 128 bytes

Fem un realloc com a un malloc.

Malloc 4352 bytes

2. malloc_simple: *grep*

Para el comando *grep*, hemos introducido el fichero1.txt, que contiene la frase "En busca de la felicidad - Will Smith". Veamos su funcionamiento:

maribel@maribel-UX430UAR: /Documentos/So1/Lab/Practica4/codi/mallocSimple$ grep -w felicidad fichero1.txt

...

Free at e63cb490: 8 bytes

En busca de la **felicidad** - Will Smith

Free at e63c93b0: 1024 bytes

3. malloc_split_and_merge: *ls*

Cambiamos de *Makefile* y tras compilar el programa e introducir la comanda *ls*, observaremos nuevas funciones respecto al malloc_simple. Estas serán *connectBlocks* (en color azul) y *splitBlocks* (en color rojo), que juntan y dividen bloques como su nombre indica. Se puede ver su uso a través de los mensajes siguientes:

...

Split completed successfully (not last block).

fichero1.txt fichero2.txt malloc.so malloc_split_and_merge.c struct.h

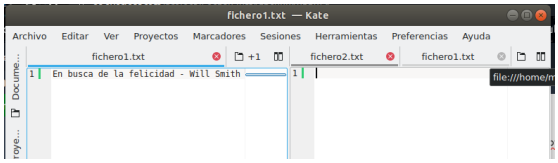
Connecting blocks, next is not last-element.

Block connected with next successfully. ...

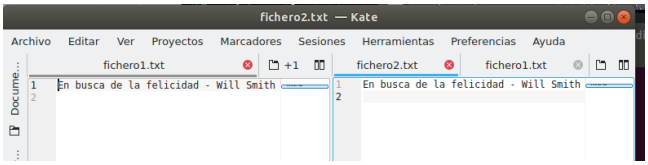
4. malloc_split_and_merge: *cp*

Para la comanda *cp*, usaremos los archivos: *fichero1.txt*, donde tenemos la misma frase que usamos en el *grep* de malloc_simple y *fichero2.txt*, que está vacío. Después de hacer el *copy*, veremos que *fichero2.txt* también tiene la frase:

maribel@maribel-UX430UAR: /Documentos/So1/Lab/Practica4/codi/mallocSplit$ cp fichero1.txt fichero2.txt



(a) Antes de hacer cp



(b) Después de hacer cp

5. malloc_split_and_merge: *find*

Por último, el *find*. Es muy similar al *find* del malloc_simple en salida por pantalla, pero implícitamente sí presenta una diferencia. Para verla, compararemos el tiempo de ejecución entre los 2 programas para notar el cambio entre ambos. Primero miraremos la salida sin el *time* del *find* en malloc_split_and_merge. Solo hay un único archivo .c en este directorio:

...

Block connected with next successfully.

./malloc_split_and_merge.c

Connecting blocks, previous will not be last-element. ...

Ahora compararemos el tiempo. Para ello, introduciremos en la terminal para ambos programas *time find -type f -name "*.c"* :

TIME Y FIND DEL MALLOC_SIMPLE

real 0m0,003s

user 0m0,000s

sys 0m0,003s

TIME Y FIND DEL MALLOC_SPLIT_AND_MERGE

real 0m0,004s

user 0m0,000s

sys 0m0,004s

Como podemos observar, el *find* del malloc_simple funciona mejor al ser muy pocos archivos a encontrar (2 en ese caso). Aún así, si incrementamos el número de archivos, el *find* del 2º programa se mantiene a un tiempo menor y más estable, mientras que en el 1º programa se dispara.

6. Errores y problemas a comentar

Además, nos hemos fijado que al realizar el *find* con el malloc_split_and_merge en directorios que a su vez contienen subdirectorios, nos da un error del core: *"Aborted (dumped core)"*. Creemos que hay problemas en la gestión de memoria a la hora de introducirse en los nuevos subdirectorios, ya que si es un directorio sin subdirectorio lo hace perfecto y en un tiempo óptimo.