Reporte

Autores:

- Joseph Avila
- Ramiro Serrano

Objetivos

- Trabajar con punteros para manipular los datos almacenados en la memoria
- Trabajar con el mecanismo de asignación de memoria en la biblioteca c estándar
- Comprender e implementar los conceptos detrás de un asignador de memoria

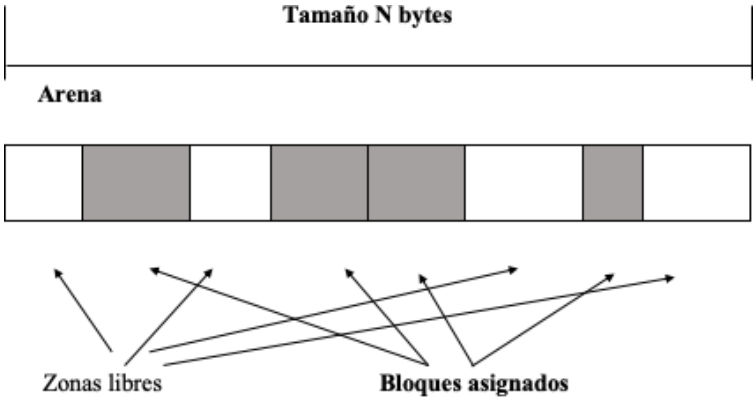
Un asignador de memoria se puede describir, en los términos más simples, de la siguiente manera:

Recibe un bloque grande y compacto (sin "agujeros") de memoria, que tiene que gestionar. Este bloque, en terminología especializada, se llama ***arena***. Por ejemplo, el sistema de asignación malloc()administra heap su programa, que es un segmento especial de memoria reservado específicamente para asignaciones dinámicas.

Los usuarios solicitan porciones más pequeñas del tamaño especificado de este bloque. El asignador debe encontrar en la ***arena*** una porción libre continua (no asignada), mayor o igual al tamaño solicitado por el usuario, que luego marca como ocupada y devuelve al usuario la dirección de inicio del área recién marcada como asignada. El asignador debe tener cuidado de que los bloques asignados no se superpongan (estén disyunto), porque de lo contrario los datos modificados en un bloque alterarán los datos en el otro bloque también.

Los usuarios pueden pedirle al asignador que libere una parte de la memoria asignada previamente para que el nuevo espacio libre esté disponible para otras asignaciones.

En cualquier momento, la *arena* parece una sucesión de bloques libres u ocupados, como se muestra en la figura siguiente.



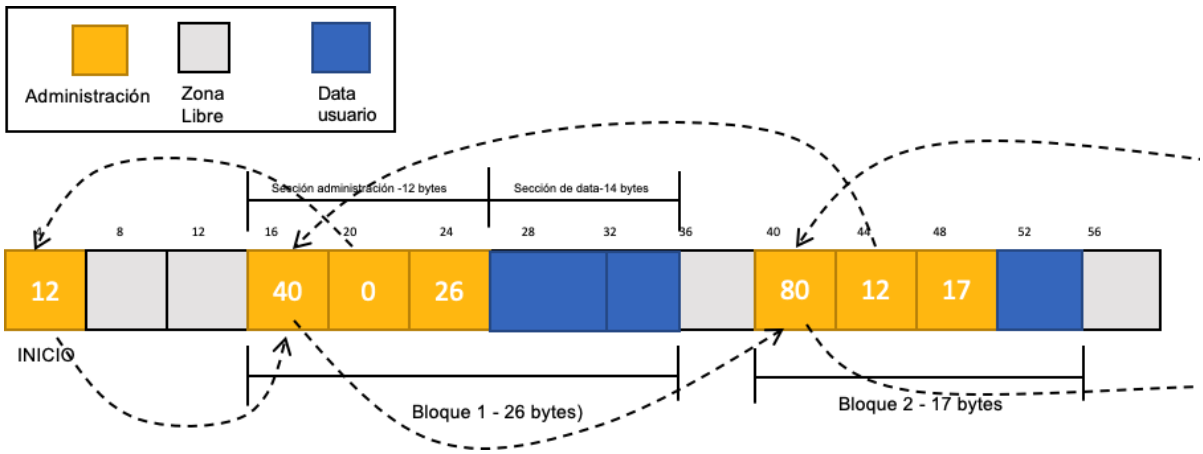
Un problema que tiene cualquier asignador de memoria es cómo realizar un seguimiento de los bloques asignados, las porciones libres y su tamaño. Generalmente, existen dos soluciones para este problema:

- Definición de áreas de memoria de arena separadas que contienen listas de bloques y su descripción. Por lo tanto, la arena contendrá solo datos del usuario, y el asignador utilizará la sección separada para encontrar bloques libres y realizar un seguimiento de los bloques asignados.
- La otra solución, lo que se implemento en este trabajo, usa la arena para almacenar información sobre los bloques asignados. El precio que se paga es que la arena no estará totalmente disponible para los usuarios, porque contendrá, además de datos, información de gestión, pero la ventaja es que no necesita memoria adicional y generalmente es más rápida que la primera opción.

Estructura de la arena

A continuación, consideraremos la arena como una secuencia (vector) de N bytes (tipo de datos char sin signo). Se puede acceder a cada byte a través de su índice 0 a N1. Consideraremos que un índice es un entero de 32 bits con signo (el tipo de datos int en un sistema operativo de 32 bits). Además, a veces necesitaremos considerar 4 bytes sucesivos en la arena como representando el valor de un índice. En esta situación, consideraremos que el índice está representado en formato 'little-endian', por lo que podremos lanzar desde un puntero escriba unsigned char * a uno de tipo int *, para acceder al valor del índice, almacenado en la arena.

La siguiente figura ilustra la estructura detallada de la arena, durante la ejecución del programa:



La estructura de un bloque

Se puede ver que cada bloque de memoria asignado (marcado con un borde en negrita) consta de dos secciones:

La primera sección, gestión, está representada por 12 bytes (3 * tamaño de (int)) divididos en 3 enteros (de 4 bytes cada uno). Los tres enteros representan lo siguiente:

- El primer número entero representa el índice de inicio del siguiente bloque de memoria en la arena (ubicado inmediatamente "a la derecha" del bloque actual, si miramos la arena como una secuencia de bytes de izquierda a derecha). Se considera que un bloque comienza con la sección de gestión, y todas las pistas sobre los bloques se tratarán como tales. Si el bloque es el último en la arena (más "a la derecha"), entonces el valor del primer número entero en la sección será0.
- El segundo número entero de la sección representa el índice inicial del bloque inmediatamente anterior en la arena. Si el bloque es el primero en la arena, entonces el valor de este número entero será 0.
- El tercer número entero de la sección representa la longitud total del bloque, es decir, la longitud de las dos secciones juntas (no solo los datos asignados al usuario).

La segunda sección contiene los datos reales del usuario. La sección tiene una longitud en bytes igual al tamaño de los datos solicitados por el usuario al llamar a la función de asignación. El índice devuelto por el asignador a una nueva asignación representa el comienzo de esta sección del nuevo bloque y no el comienzo de la primera sección, ya

que la parte de administración de memoria debe ser completamente transparente para el usuario.

Encadenamiento de bloques

Como se puede ver en la figura anterior, al comienzo de la arena están reservados 4 bytes que representan el índice de inicio. Si la arena no contiene ningún bloque (por ejemplo, inmediatamente después de la inicialización), este índice es 0.

El índice de inicio marca el inicio de la cadena de bloques en la arena: desde este índice podemos llegar al inicio del primer bloque, luego usando la sección de administración del primer bloque podemos encontrar el inicio del segundo bloque, y así sucesivamente, hasta llegar al bloque que tiene el índice del siguiente bloque 0 (es el último bloque de la arena). De esta forma podemos cruzar todos los bloques de la arena, y también identificar los espacios libres en la arena, que representan los espacios entre dos bloques sucesivos.

Es de destacar que la cadena se puede atravesar en ambos sentidos: desde una cuadra podemos llegar tanto al vecino de la derecha como al de la izquierda. Además, cuando se asigna un nuevo bloque o se libera uno antiguo, "es necesario cambiar la cadena de bloques". Por lo tanto, al asignar un nuevo bloque de memoria, debe considerar lo siguiente:

▼ El espacio libre en el que se asigna el nuevo bloque está delimitado por dos bloques vecinos como máximo. Las secciones de gestión de estos vecinos deben modificarse de la siguiente manera:

- El índice del siguiente bloque en la estructura de gestión del bloque izquierdo debe apuntar al nuevo bloque. Si el bloque de la izquierda no existe, se cambia el índice de inicio.
- El índice del bloque anterior en la estructura de gestión del bloque derecho debe apuntar al nuevo bloque. Si el bloque de la derecha no existe, no pasa nada.

▼ La sección de administración del nuevo bloque contendrá los índices de los dos vecinos, o 0 en lugar del vecino faltante

Al liberar un bloque, las secciones de administración de vecinos deben cambiarse de manera similar a como al agregar.

Operación del programa

El programa implementa una serie de operaciones en la arena, que se lanzarán siguiendo los comandos que reciba en la entrada. Cada pedido se dará en una línea y los resultados se mostraran en el acto. La siguiente sección presenta la sintaxis de posibles comandos y el formato de visualización de los resultados.

Implementaciones

INICIALIZAR <N>

Este comando se llama primero e inicializa una arena de N bytes. Inicialización significa la asignación dinámica de la memoria requerida para almacenar la arena, la configuración de cada byte en 0 y la inicialización de la cadena de bloques (configurando el índice de inicio en 0).
El comando no muestra ningún resultado.

FINALIZAR

Este comando se llama en último lugar y libera la memoria asignada en la inicialización y finalizar el programa.
El comando no mostrará ningún resultado.

DUMP

Este comando mostrará el mapa de memoria completo, tal como se encuentra actualmente, byte por byte. Se mostrarán 16 bytes por línea de la siguiente manera:

- Al principio de la línea, se mostrará el índice actual, en formato hexadecimal, con 8 dígitos hexadecimales en mayúscula.
- Luego se muestra una TAB ('\ t'), seguida de 16 bytes, separados por un espacio y en formato hexadecimal, con 2 dígitos hexadecimales en mayúscula cada uno. Se mostrará espacio adicional entre el octavo y el noveno bytes.
- En la última línea, independientemente del número de bytes en la arena, se mostrará el índice del último byte en la arena + 1 (prácticamente, el tamaño de la arena), en formato hexadecimal con 8 dígitos hexadecimales en mayúscula.
- Muestra los bytes del mapa en formato hexadecimal;
- Similar a hexdump.

ASIGNAR<TAMAÑO>

- El comando asignará TAMAÑO bytes de memoria de la arena. Necesitará encontrar un área libre suficientemente grande (para ajustarse a TAMÑO bytes + sección de administración) y reservar un bloque 'al principio' del área (no en el medio, no al final). La primera zona franca válida se debe utilizar en una búsqueda de izquierda a derecha.
- El comando mostrará, en formato decimal, el índice de inicio del bloque asignado en

la arena, o 0 si no se ha encontrado una zona libre suficientemente grande en la arena. Se muestra el índice de la sección de datos en el nuevo bloque, no la sección de administración.

- Devuelve la dirección de inicio del área asignada.

FREE <INDICE>

- El comando liberará el bloque de memoria cuyas secciones de datos comienzan en la posición INDICE en la arena. Básicamente, INDICE será un valor que fue devuelto previamente por un comando 'ASIGNAR'. Tras la ejecución de este comando, el espacio de arena ocupado por el bloque antiguo volverá a estar disponible para asignaciones posteriores.
- El comando no mostrará ningún resultado

LLENAR <INDICE> <TAMAÑO> <VALOR>

- El comando establecerá TAMAÑO bytes consecutivos en la arena, comenzando con el índice INDICE, al valor VALOR, entre 0 y 255 inclusive. Advertencia, este comando también puede cambiar los bytes de administración, no solo los bytes de datos. En este caso, se garantiza que la arena no se corrompe después de una serie de comandos LLENAR consecutivos.
- El comando no mostrará ningún resultado.

SHOW <INFORMACION>

El comando mostrará información estadística sobre el estado de la arena. INFORMACION puede ser uno de los siguientes:

▼ **FREE**

Se mostrará el número de bytes libres en la arena (en formato decimal), junto con la cantidad de regiones libres (zonas continuas) en la arena de la siguiente manera:

> <nblocks> bloques (<nbytes> bytes) libres

▼ **USO**

▼ Se mostrarán, una línea a la vez:

- Número de bytes utilizados en la arena (solo secciones de datos)
- Eficiencia de uso (en porcentaje) igual a la cantidad de bytes usados en relación con la cantidad de bytes reservados (que no son libres)
- Fragmentación (en porcentaje), igual al número de zonas francas - 1, relacionado con el número de bloques asignados. Para una arena sin bloques asignados, la fragmentación se considerará 0.

▼ El formato de visualización es:

> <nblocks> bloques(<nbytes> bytes) usados
> <eff>% Eficiencia
> <fragm>% Fragmentacion

▼ **ASIGNACIONES**

▼ Las áreas libres y asignadas se mostrarán en una línea, en el orden en que se colocan en la arena. Cada línea tendrá la forma:

> {Libre|Ocupado} <N>

▼ Donde {.. | ..} representa que solo se mostrará uno de los valores. N representa el tamaño (distinto de cero), en bytes, de esa área.

ALLOCALIGNED <TAMAÑO> <ALINEACION>

Este comando funcionará como ASIGNAR, excepto que el índice devuelto deberá estar alineado con los bytes ALINEACION, donde ALINEACION es una potencia de 2 (puede ser 1, 2, 4, 8, etc.). Un índice INDICE se alinea con los bytes ALINEACION si INDICE% ALINEACION == 0.

REASIGNAR <INDICE> <TAMAÑO>

- Este comando reasignará un área de memoria devuelta previamente a la dirección INDICE en un nuevo espacio de memoria TAMAÑO y mostrará el índice de la sección de datos del nuevo bloque asignado. También copiará los datos del bloque antiguo al bloque nuevo. Si TAMAÑO es más pequeño que el tamaño original, solo se copiarán los bytes TAMAÑO(se producirá el truncamiento).
- Al encontrar un área de memoria libre, se repite el procedimiento de búsqueda de izquierda a derecha. No es válido comprobar que ya hay espacio para expansión / reducción en la ubicación actual

SHOW MAP <LONGITUD>

El comando mostrará una cadena de caracteres LONGITUD en varias líneas, cada carácter será "*" o ".", Que describirá las áreas libres u ocupadas de la arena. Un carácter es "" si hay al menos un byte reservado en el área que describe, de lo contrario será "." Cada línea mostrará un máximo de 80 caracteres de este tipo. Si el tamaño de la arena es N, entonces un carácter representará x = N / LONGITUD bytes. Si al menos uno de los x bytes está ocupado, se mostrará "*", en caso contrario ".". Atención, x también puede ser una subunidad.

DEFRAGMENTAR

- Pega todas las áreas asignadas a la izquierda, de modo que, después de ejecutar el comando, la fragmentación disminuye a * 0% *;
- Devuelve un vector que hace la conexión entre los índices antiguos de las áreas de memoria y los nuevos.

SAFE_LLENAR INDEX SIZE VALUE

- La función `LLENAR` corre el riesgo de sobrescribir bytes donde no deberían tener acceso;
- La función `SAFE_LLENAR` comprueba la escritura en un `INDICE` válido (asignado) y un número de bytes que no exceda el área donde se realiza la escritura.

Implementaciones

Funciones

```
int32_t ALLOCALIGNED(block_t *block, int32_t size, int32_t align); int32_t
buscarPos(int32_t pos, int32_t align, int32_t size,
                  int32_t prevFinal, int32_t nextFirst);
// la función busca la posición a la que se podría asignar la nueva área  alineada

int32_t realloc_(block_t *block, int32_t pos, int32_t size);
void copiarMemoria(block_t *block, int32_t *currIndex, int newPos, int size);
/ * copiar bytes del área currIndex al área newPos, teniendo en
cuenta las dimensiones de estas áreas * / void defragmentar(block_t
*block, defrag_t *def);
/ * adjuntar todas las áreas asignadas de la arena y regresar un vector con
sus posiciones antiguas y nuevas * /
void FREE(block_t *block, int32_t pos);
void llenar(block_t *block, int32_t pos, int32_t size, int32_t value); void safe_fill(block_t
*block, int32_t pos, int32_t size, int32_t value);
// es un fill(LLENAR) que no sobrescribe áreas inválidas o no asignadas void
mostrar(block_t *block, char *arg);
void mostrar_map(block_t *block, int32_t l);
// mostrar el carácter c (* o.), para Imprimir tiempos, en el formato solicitado por MOSTRAR_MAP void
printMemoria(int32_t toPrint, int32_t *count, char c);
int32_t asignar(block_t *block, int32_t size);
void asignarBtw(block_t *block, int32_t *prev, int32_t *next, int32_t start,
                int32_t size);
// inserta un área entre 2 índices (punteros) en la  arena
void asignarLast(block_t *block, int32_t *prev, int32_t *next, int32_t start,
                 int32_t size);
// agrega un área después de la última asignada previamente void
volcado(block_t *block);
void mostrar_free(block_t *block); void
mostrar_usage(block_t *block); void
mostrar_alloc(block_t *block);
int32_t verificarPos(block_t *block, int32_t pos);
/ * devuelve 0 si pos no es parte de un área disponible  o
posición inicial del área a la que pertenece en caso contrario * / void
inicializar(block_t *block);
void finalizar(block_t *block);
```

Structs

Struct para la defragmentacion

```
typedef struct {
     int32_t oldPos, newPos;
} defrag_t;
```

Scruct para los bloques:

```
typedef struct {
     int32_t len;
     uchar_t *mem;
} block_t;
```

Resultados

Se realiza 12 test :

Test 1

INPUT

```
INICIALIZAR 100
ASIGNAR 20
ASIGNAR 10
FREE 16
ASIGNAR 19
LLENAR 16 19 255
LLENAR 48 10 127 DUMP
MOSTRAR ASIGNACIONES
ASIGNAR 1
MOSTRAR USO
FINALIZAR
```

OUTPUT

```
16
48
16
00000000         04 00 00 00 24 00 00  00       00 00 00 00 1F 00 00  00
00000010         FF FF FF FF FF FF FF  FF       FF FF FF FF FF FF FF  FF
00000020         FF FF FF 00 00 00 00  00       04 00 00 00 16 00 00  00
00000030         7F 7F 7F 7F 7F 7F 7F  7F       7F 7F 00 00 00 00 00  00
00000040         00 00 00 00 00 00 00  00       00 00 00 00 00 00 00  00
00000050         00 00 00 00 00 00 00  00       00 00 00 00 00 00 00  00
00000060     00 00 00 00
00000064
Ocupados   4 bytes
OCupados   31 bytes

Libres1 bytes
OCupados 22 bytes
Libres 42 bytes
```

```
70
3 blocks (30 bytes) usados
42% Eficiencia 33% Fragmentacion
```

Test 2

INPUT

```
INICIALIZAR 100
ASIGNAR 13
ASIGNAR 14
ASIGNAR 15
FREE 41 MOSTRAR
USO LLENAR 0 4 0
MOSTRAR FREE
MOSTRAR ASIGNACIONES
FINALIZAR
```

OUTPUT

```
16
41
67
2 blocks (28 bytes) usados
50% Eficiencia 50% Fragmentacion
1 blocks (96 bytes) free
Ocupados 4 bytes
Libres 96 bytes
```

Test 3

INPUT

```
INICIALIZAR 100
LLENAR 0 1 30
LLENAR 38 1 15 MOSTRAR
ASIGNACIONES LLENAR 42 3
255
DUMP ASIGNAR
20 DUMP
FREE 42
MOSTRAR ASIGNACIONES
FINALIZAR
```

OUTPUT

```
Ocupados 4 bytes Libres26
bytes OCupados 15 bytes
Libres 55 bytes
00000000 1E 00 00 00 00 00 00  00
                                        00 00 00 00 00 00 00  00
00000010 00 00 00 00 00 00 00  00       00 00 00 00 00 00 00  00
```

```
00000020     00  00   00  00   00  00  0F  00    00  00  FF  FF  FF  00  00   00
00000030     00  00   00  00   00  00  00  00    00  00  00  00  00  00  00   00
00000040     00  00   00  00   00  00  00  00    00  00  00  00  00  00  00   00
00000050     00  00   00  00   00  00  00  00    00  00  00  00  00  00  00   00
00000060     00  00   00  00
00000064
57
00000000     1E  00   00  00   00  00  00  00    00  00  00  00  00  00  00   00
00000010     00  00   00  00   00  00  00  00    00  00  00  00  00  00  2D   00
00000020     00  00   00  00   00  00  0F  00    00  00  FF  FF  FF  00  00   00
00000030     00  1E   00  00   00  20  00  00    00  00  00  00  00  00  00   00
00000040     00  00   00  00   00  00  00  00    00  00  00  00  00  00  00   00
00000050     00  00   00  00   00  00  00  00    00  00  00  00  00  00  00   00
00000060     00  00   00  00
00000064

Ocupados 4 bytes
Libres41 bytes
OCupados 32 bytes
Libres 23 bytes
```

Test 4

INPUT

```
INICIALIZAR 100
ASIGNAR 13
LLENAR 16    13  255
DUMP
FREE 16
ASIGNAR 50
ASIGNAR 40
ASIGNAR 30
ASIGNAR 20
LLENAR 78    20  127
DUMP
FREE 16
FREE 78
FINALIZAR
```

OUTPUT

```
16
00000000     04  00   00  00   00  00  00  00    00  00  00  00  19  00  00   00
00000010     FF  FF   FF  FF   FF  FF  FF  FF    FF  FF  FF  FF  FF  00  00   00
00000020     00  00   00  00   00  00  00  00    00  00  00  00  00  00  00   00
00000030     00  00   00  00   00  00  00  00    00  00  00  00  00  00  00   00
00000040     00  00   00  00   00  00  00  00    00  00  00  00  00  00  00   00
00000050     00  00   00  00   00  00  00  00    00  00  00  00  00  00  00   00
00000060     00  00   00  00
00000064
16
0
0
78
00000000     04  00   00  00   42  00  00  00    00  00  00  00  3E  00  00   00
00000010     FF  FF   FF  FF   FF  FF  FF  FF    FF  FF  FF  FF  FF  00  00   00
00000020     00  00   00  00   00  00  00  00    00  00  00  00  00  00  00   00
00000030     00  00   00  00   00  00  00  00    00  00  00  00  00  00  00   00
```

```
00000040  00 00 00 00 00 00 04 00  00 00 20 00 00 00 7F 7F
00000050  7F 7F 7F 7F 7F 7F 7F 7F  7F 7F 7F 7F 7F 7F 7F 7F
00000060 7F 7F 00 00
00000064
```

Test 5

INPUT

```
INICIALIZAR 100
ASIGNAR 10
ASIGNAR 10
ASIGNAR 10
ASIGNAR 10
ASIGNAR 10
FREE 16
FREE 60
LLENAR 38          10 255
LLENAR 82          10 255
DUMP
FINALIZAR
```

OUTPUT

```
16
38
60
82
0
00000000     1A 00   00  00   1A 00 00 00        00 00 00 00 16 00 00  00
00000010     00 00   00  00   00 00 00 00        00 00 46 00 00 00 00  00
00000020     00 00   16  00   00 00 FF FF        FF FF FF FF FF FF FF  FF
00000030     46 00   00  00   1A 00 00 00        16 00 00 00 00 00 00  00
00000040     00 00   00  00   00 00 00 00        00 00 1A 00 00 00 16  00
00000050     00 00   FF  FF   FF FF FF FF        FF FF FF FF 00 00 00  00
00000060     00 00   00  00
00000064
```

Test 6

INPUT

```
INICIALIZAR 100
ASIGNAR 20
ASIGNAR 20
MOSTRAR FREE
MOSTRAR USO
MOSTRAR ASIGNACIONES FREE
16
MOSTRAR FREE MOSTRA
USO
MOSTRAR ASIGNACIONES
FINALIZAR
```

OUTPUT

```
16
48
1 blocks (32 bytes) free
2 blocks (40 bytes) usados
58% Eficiencia 0% Fragmentacion
Ocupados 4 bytes
OCupados 32 bytes
OCupados 32 bytes
Libres 32 bytes
2 blocks (64 bytes) free [-
]¡Comando desconocido!
Ocupados 4 bytes Libres32
bytes
OCupados 32 bytes
Libres 32 bytes
```

Test 7

INPUT

```
INICIALIZAR 100
ASIGNADOS 10 32 MOSTRAR
ASIGNACIONES LLENAR 32 10
255
MOSTRAR MAP 50
MOSTRAR MAP 31
MOSTRAR MAP 2
MOSTRAR MAP 200
DUMP
DUMP
MOSTRAR MAP 100
FINALIZAR
```

OUTPUT

```
32
Ocupados 4 bytes
Libres16 bytes
OCupados 22 bytes
Libres 58 bytes
**........**********.............................
**....********.................
*.
********.............................****************************************
****..........................................................................
00000000    14 00  00 00  00 00  00 00   00 00  00 00  00 00  00 00  00 00
00000010    00 00  00 00  00 00  00 00   00 00  00 00  16 00  00 00
00000020    FF FF  FF FF  FF FF  FF FF   FF FF  00 00  00 00  00 00  00 00
00000030    00 00  00 00  00 00  00 00   00 00  00 00  00 00  00 00  00 00
00000040    00 00  00 00  00 00  00 00   00 00  00 00  00 00  00 00  00 00
00000050    00 00  00 00  00 00  00 00   00 00  00 00  00 00  00 00  00 00
00000060    00 00  00 00
00000064
00000000    14 00  00 00  00 00  00 00   00 00  00 00  00 00  00 00  00 00
00000010    00 00  00 00  00 00  00 00   00 00  00 00  16 00  00 00
00000020    FF FF  FF FF  FF FF  FF FF   FF FF  00 00  00 00  00 00  00 00
00000030    00 00  00 00  00 00  00 00   00 00  00 00  00 00  00 00  00 00
00000040    00 00  00 00  00 00  00 00   00 00  00 00  00 00  00 00  00 00
```

```
00000050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00000060 00 00 00 00
00000064
****..............**********.....................................
.....................
```

Test 8

INPUT

```
INICIALIZAR 200
  ASIGNADOS 10 128
  ASIGNADOS 10 64
  ASIGNADOS 10 32
  ASIGNADOS 10 16
  ASIGNADOS 10 8
  ASIGNADOS 10 4
  ASIGNADOS 10 2 MOSTRAR
  ASIGNACIONES MOSTRAR MAP
  200
  DUMP
  FINALIZAR
```

OUTPUT

```
 128
64
32
96
152
176
0
Ocupados 4 bytes
Libres16 bytes
OCupados 22 bytes
Libres10 bytes
OCupados 22 bytes
Libres10 bytes
OCupados 22 bytes
Libres10 bytes
OCupados 22 bytes
Libres2 bytes
OCupados 22 bytes
Libres2 bytes
OCupados 22 bytes
Libres 14 bytes
****..............**********.........**********......
....**********.........**********..**********
**..**********.............
00000000      14  00   00  00   00  00   00  00    00  00   00  00   00  00   00  00
00000010      00  00   00  00   34  00   00  00    00  00   00  00   16  00   00  00
00000020      00  00   00  00   00  00   00  00    00  00   00  00   00  00   00  00
00000030      00  00   00  00   54  00   00  00    14  00   00  00   16  00   00  00
00000040      00  00   00  00   00  00   00  00    00  00   00  00   00  00   00  00
00000050      00  00   00  00   74  00   00  00    34  00   00  00   16  00   00  00
00000060      00  00   00  00   00  00   00  00    00  00   00  00   00  00   00  00
00000070      00  00   00  00   8C  00   00  00    54  00   00  00   16  00   00  00
00000080      00  00   00  00   00  00   00  00    00  00   00  00   A4  00   00  00
00000090      74  00   00  00   16  00   00  00    00  00   00  00   00  00   00  00
000000A0      00  00   00  00   00  00   00  00    8C  00   00  00   16  00   00  00
```

```
000000B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
000000C0 00 00 00 00 00 00 00 00
000000C8
```

Test 9

INPUT

```
INICIALIZAR 300
ASIGNAR 10
ASIGNAR 20
ASIGNAR 30
ASIGNAR 40
ASIGNAR 50
ASIGNAR 60
LLENAR 226 60 255
DUMP
FREE 16
FREE 70
FREE 164
REASIGNAR 226 50
REASIGNAR 164 30
REASIGNAR 70 10 MOSTRAR
ASIGNACIONES DUMP
MOSTRAR MAP 10
MOSTRAR MAP 50
MOSTRAR MAP 101
MOSTRAR MAP 1000
FINALIZAR
```

OUTPUT

```
16
38
70
112
164
226
00000000     04  00  00  00  1A  00  00  00    00  00  00  00  16  00  00  00
00000010     00  00  00  00  00  00  00  00    00  00  3A  00  00  00  04  00
00000020     00  00  20  00  00  00  00  00    00  00  00  00  00  00  00  00
00000030     00  00  00  00  00  00  00  00    00  00  64  00  00  00  1A  00
00000040     00  00  2A  00  00  00  00  00    00  00  00  00  00  00  00  00
00000050     00  00  00  00  00  00  00  00    00  00  00  00  00  00  00  00
00000060     00  00  00  00  98  00  00  00    3A  00  00  00  34  00  00  00
00000070     00  00  00  00  00  00  00  00    00  00  00  00  00  00  00  00
00000080     00  00  00  00  00  00  00  00    00  00  00  00  00  00  00  00
00000090     00  00  00  00  00  00  00  00    D6  00  00  00  64  00  00  00
000000A0     3E  00  00  00  00  00  00  00    00  00  00  00  00  00  00  00
000000B0     00  00  00  00  00  00  00  00    00  00  00  00  00  00  00  00
000000C0     00  00  00  00  00  00  00  00    00  00  00  00  00  00  00  00
000000D0     00  00  00  00  00  00  00  00    00  00  98  00  00  00  48  00
000000E0     00  00  FF  FF  FF  FF  FF  FF    FF  FF  FF  FF  FF  FF  FF  FF
000000F0     FF  FF  FF  FF  FF  FF  FF  FF    FF  FF  FF  FF  FF  FF  FF  FF
00000100     FF  FF  FF  FF  FF  FF  FF  FF    FF  FF  FF  FF  FF  FF  FF  FF
00000110     FF  FF  FF  FF  FF  FF  FF  FF    FF  FF  FF  FF  FF  FF  00  00
00000120     00  00  00  00  00  00  00  00    00  00  00  00
0000012C
164
```

```
70
16
Ocupados 4 bytes
OCupados 22 bytes
OCupados 32 bytes
Libres42 bytes
OCupados 52 bytes
Libres 148 bytes
00000000    04  00  00  00  1A  00  00  00   00  00  00  00  16  00  00  00
00000010    FF  FF  FF  FF  FF  FF  FF  FF   FF  FF  64  00  00  00  04  00
00000020    00  00  20  00  00  00  00  00   00  00  00  00  00  00  00  00
00000030    00  00  00  00  00  00  00  00   00  00  64  00  00  00  1A  00
00000040    00  00  2A  00  00  00  FF  FF   FF  FF  FF  FF  FF  FF  FF  FF
00000050    FF  FF  FF  FF  FF  FF  FF  FF   FF  FF  FF  FF  FF  FF  FF  FF
00000060    FF  FF  FF  FF  00  00  00  00   1A  00  00  00  34  00  00  00
00000070    00  00  00  00  00  00  00  00   00  00  00  00  00  00  00  00
00000080    00  00  00  00  00  00  00  00   00  00  00  00  00  00  00  00
00000090    00  00  00  00  00  00  00  00   00  00  00  00  64  00  00  00
000000A0    3E  00  00  00  FF  FF  FF  FF   FF  FF  FF  FF  FF  FF  FF  FF
000000B0    FF  FF  FF  FF  FF  FF  FF  FF   FF  FF  FF  FF  FF  FF  FF  FF
000000C0    FF  FF  FF  FF  FF  FF  FF  FF   FF  FF  FF  FF  FF  FF  FF  FF
000000D0    FF  FF  FF  FF  FF  FF  00  00   00  00  64  00  00  00  48  00
000000E0    00  00  FF  FF  FF  FF  FF  FF   FF  FF  FF  FF  FF  FF  FF  FF
000000F0    FF  FF  FF  FF  FF  FF  FF  FF   FF  FF  FF  FF  FF  FF  FF  FF
00000100    FF  FF  FF  FF  FF  FF  FF  FF   FF  FF  FF  FF  FF  FF  FF  FF
00000110    FF  FF  FF  FF  FF  FF  FF  FF   FF  FF  FF  FF  FF  FF  00  00
00000120    00  00  00  00  00  00  00  00   00  00  00  00
0000012C
**.***....
**********......**********.......................
********************.............********************.............................
.....................
********************************************************************************
********************************************************************************
***********************************.............................................
.....................................................................................
.....................................................................................
............************************************************************
********************************************************************************
****************************.............................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................................................................
.....................................
```

Test 10

INPUT

```
INICIALIZAR 100
ASIGNAR 10
ASIGNAR 10
ASIGNAR 10
ASIGNAR 10
ASIGNAR 10
FREE 16
FREE 60
ASIGNAR 30
MOSTRAR ASIGNACIONES
DEFRAGMENTAR
MOSTRAR ASIGNACIONES
```

```
MOSTRAR ASIGNACIONES
FINALIZAR
```

OUTPUT

```
16
38
60
82
0
0
Ocupados 4 bytes
Libres22 bytes
OCupados 22 bytes
Libres22 bytes
OCupados 22 bytes
Libres 8 bytes
Posición antes de desfragmentar: 38 Posición después de la desfragmentación: 16 Posición antes de
desfragmentar: 82 Posición después de la desfragmentación: 38 Ocupados 4 bytes
OCupados 22 bytes
OCupados 22 bytes
Libres 52 bytes
60
Ocupados 4 bytes
OCupados 22 bytes
OCupados 22 bytes
OCupados 42 bytes
Libres 10 bytes
```

Test 11

INPUT

```
INICIALIZAR 100
ASIGNADOS 10 64
ASIGNADOS 10 32 DUMP
SLLENAR 66 3 255
SLLENAR 32 100 255
MOSTRAR ASIGNACIONES
MOSTRAR USO
DUMP
DEFRAGMENTAR
MOSTRAR ASIGNACIONES
MOSTRAR USO
DUMP
ASIGNADOS 10 64
SLLENAR 65 4 255
DUMP DEFRAGMENTAR
DUMP
FINALIZAR
```

OUTPUT

```
64
32
00000000     14 00 00 00 00 00 00  00        00  00  00  00  00  00  00  00
00000010     00 00 00 00 34 00 00  00        00  00  00  00  16  00  00  00
00000020     00 00 00 00 00 00 00  00        00  00  00  00  00  00  00  00
00000030     00 00 00 00 00 00 00  00        14  00  00  00  16  00  00  00
00000040     00 00 00 00 00 00 00  00        00  00  00  00  00  00  00  00
00000050     00 00 00 00 00 00 00  00        00  00  00  00  00  00  00  00
00000060     00 00 00 00
00000064
[+] 3 byte(s) modificados [+] 10
byte(s) modificados Ocupados 4
bytes
Libres16 bytes
OCupados 22 bytes
Libres10 bytes
OCupados 22 bytes
Libres 26 bytes
2 blocks (20 bytes) usados
41% Eficiencia 100% Fragmentacion
00000000     14 00 00 00 00 00 00  00        00  00  00  00  00  00  00  00
00000010     00 00 00 00 34 00 00  00        00  00  00  00  16  00  00  00
00000020     FF FF FF FF FF FF FF  FF        FF  FF  00  00  00  00  00  00
00000030     00 00 00 00 00 00 00  00        14  00  00  00  16  00  00  00
00000040     00 00 FF FF FF 00 00  00        00  00  00  00  00  00  00  00
00000050     00 00 00 00 00 00 00  00        00  00  00  00  00  00  00  00
00000060     00 00 00 00
00000064
Posición antes de desfragmentar: 32 Posición después de la desfragmentación: 16 Posición antes de
desfragmentar: 64 Posición después de la desfragmentación: 38 Ocupados 4 bytes
OCupados 22 bytes
OCupados 22 bytes
Libres 52 bytes
2 blocks (20 bytes) usados
41% Eficiencia 0% Fragmentacion
00000000     04  00 00 00 1A    00  00  00   00  00  00  00  16  00  00  00
00000010     FF  FF FF FF FF    FF  FF  FF   FF  FF  00  00  00  00  04  00
00000020     00  00 16 00 00    00  00  00   FF  FF  FF  00  00  00  00  00
00000030     00  00 00 00 00    00  00  00   04  00  00  00  16  00  00  00
00000040     00  00 FF FF FF    00  00  00   00  00  00  00  00  00  00  00
00000050     00  00 00 00 00    00  00  00   00  00  00  00  00  00  00  00
00000060     00  00 00 00
00000064
64
[+] 4 byte(s) modificados
00000000     04  00 00 00 1A    00  00  00   00  00  00  00  16  00  00  00
00000010     FF  FF FF FF FF    FF  FF  FF   FF  FF  34  00  00  00  04  00
00000020     00  00 16 00 00    00  00  00   FF  FF  FF  00  00  00  00  00
00000030     00  00 00 00 00    00  00  00   1A  00  00  00  16  00  00  00
00000040     00  FF FF FF FF    00  00  00   00  00  00  00  00  00  00  00
00000050     00  00 00 00 00    00  00  00   00  00  00  00  00  00  00  00
00000060     00  00 00 00
00000064
Posición antes de desfragmentar: 64 Posición después de la
00000000     04 00 00 00 1A 00 00  00        00 00 00    00 16 00    00  00
00000010     FF FF FF FF FF FF FF  FF        FF FF 30    00 00 00    04  00
00000020     00 00 16 00 00 00 00  00        FF FF FF    00 00 00    00  00 desfragmentación: 60
00000030     00 00 00 00 1A 00 00  00        16 00 00    00 00 FF    FF  FF
00000040     FF 00 00 00 00 00 00  00        00 00 00    00 00 00    00  00
00000050     00 00 00 00 00 00 00  00        00 00 00    00 00 00    00  00
00000060     00 00 00 00
00000064
```

Test 12

INPUT

```
INICIALIZAR 104
ASIGNAR 8
ASIGNAR 8
ASIGNAR 8
ASIGNAR 8
ASIGNAR 8
DEFRAGMENTAR
FREE 36
FREE 76
SLLENAR 16 7 255
SLLENAR 17 6 15
SLLENAR 56 200 255
SLLENAR 62 1 15
SLLENAR 63 1 1 DUMP
MOSTRAR ASIGNACIONES
MOSTRAR USO
ASIGNAR 20
DEFRAGMENTAR
DUMP
MOSTRAR ASIGNACIONES
SLLENAR 76 23     255
SLLENAR 77 23     254
SLLENAR 78 23     253
SLLENAR 75 23     255
SLLENAR 96 23     255
SLLENAR 105 23 255
DUMP
FREE 56
DEFRAGMENTAR
DUMP
```

OUTPUT

```
16
36
56
76
96
[+] 7 byte(s) modificados [+] 6
byte(s)   modificados   [+]   8
byte(s)   modificados   [+]   1
byte(s)   modificados   [+]   1
byte(s) modificados
00000000     04  00 00 00 2C 00 00  00           00 00 00 00 14 00 00  00
00000010     FF  0F 0F 0F 0F 0F 0F  00           2C 00 00 00 04 00 00  00
00000020     14  00 00 00 00 00 00  00           00 00 00 00 54 00 00  00
00000030     04  00 00 00 14 00 00  00           FF FF FF FF FF FF 0F  01
00000040     54  00 00 00 2C 00 00  00           14 00 00 00 00 00 00  00
00000050     00  00 00 00 00 00 00  00           2C 00 00 00 14 00 00  00
00000060     00  00 00 00 00 00 00  00
00000068
Ocupados 4 bytes
OCupados 20 bytes
Libres20 bytes
```

```
OCupados 20 bytes Libres20
bytes OCupados 20 bytes
3 blocks (24 bytes) usados
37% Eficiencia 33% Fragmentacion
0
Posición antes de desfragmentar: 56 Posición después de la desfragmentación: 36 Posición antes de
desfragmentar: 96 Posición después de la desfragmentación: 56

00000000          04 00 00 00 18 00 00  00   00  00  00  00  14  00  00  00
00000010          FF 0F 0F 0F 0F 0F 0F  00   2C  00  00  00  04  00  00  00
00000020          14 00 00 00 FF FF FF  FF   FF  FF  0F  01  00  00  00  00
00000030          18 00 00 00 14 00 00  00   00  00  00  00  00  00  00  00
00000040          54 00 00 00 2C 00 00  00   14  00  00  00  00  00  00  00
00000050          00 00 00 00 00 00 00  00   18  00  00  00  14  00  00  00
00000060          00 00 00 00 00 00 00  00
00000068
Ocupados   4 bytes
OCupados   20 bytes
OCupados   20 bytes
OCupados   20 bytes
Libres 40 bytes
76
[+]  20  byte(s)  modificados  [+]
19  byte(s)  modificados  [+]  18
byte(s) modificados
[-]¡Acceso  no  válido  a  la  memoria!  [-
]¡Acceso no válido a la memoria!
00000000     04 00 00 00    18  00 00 00    00  00  00  00  14  00  00  00
00000010     FF 0F 0F 0F    0F  0F 0F 00    2C  00  00  00  04  00  00  00
00000020     14 00 00 00    FF  FF FF FF    FF  FF  0F  01  40  00  00  00
00000030     18 00 00 00    14  00 00 00    00  00  00  00  00  00  00  00
00000040     00 00 00 00    2C  00 00 00    20  00  00  00  FF  FE  FD  FD
00000050     FD FD FD FD    FD  FD FD FD    FD  FD  FD  FD  FD  FD  FD  FD
00000060     00 00 00 00    00  00 00 00
00000068
Posición   antes de desfragmentar: 76 Posición después           de  la  desfragmentación: 56
00000000     04 00 00 00 18 00 00 00 00 00 00 00 14 00            00  00
00000010     FF 0F 0F 0F 0F 0F 0F 00 2C 00 00 00 04 00            00  00
00000020     14 00 00 00 FF FF FF FF FF FF 0F 01 00 00            00  00
00000030     18 00 00 00 20 00 00 00 FF FE FD FD FD FD            FD  FD
00000040     FD FD FD FD FD FD FD FD FD FD FD FD FF FE            FD  FD
00000050     FD FD FD FD FD FD FD FD FD FD FD FD FD FD            FD  FD
00000060     00 00 00 00 00 00 00  00
00000068
```

Test 13

Input

```
INICIALIZAR 300
ASIGNAR 10
ASIGNAR 20
ASIGNAR 30
ASIGNAR 40
ASIGNAR 50
ASIGNAR 60
FREE 16
FREE 70
```

```
FREE 164 MOSTRAR
FREE FINALIZAR
```

Output

```
16
38
70
112
164
226
4 blocks (140 bytes) free
3 Liberaciones Totales
```

Pro y contras de la implementación

Pros

- Facilidad de uso, se muestra de manera sencilla el uso de todos los comandos que existen el administrador de memoria
- Muestra un mapa de los bytes que están asignados y están libres
- Se mostrara alertas si existe accesos no validos en la memoria
- La desfragmentación es rápida
- Se puede ver que se han rechazado demasiadas solicitudes de asignación, mostrando 0 (que es un índice de datos no válido, porque el índice de inicio está en la posición 0 y no se puede asignar memoria en esa área)
- Todas las memorias asignadas que se van almacenando automáticamente de izquierda a derecha por la estructura
- Al reasignar un área de memoria devuelta previamente a la dirección INDICE en un nuevo espacio de memoria TAMAÑO y mostrará el índice de la sección de datos del nuevo bloque asignado. También copiará los datos del bloque antiguo al bloque nuevo

Contras

- Si está trabajando en 64 bits, es posible que no tenga todas las bibliotecas necesarias para ejecutar ejecutables compilados de 32 bits.

SOLUCION

```
sudo apt-get update
sudo apt-get install multiarch-support
```

Referencias

[1]"Memory Allocators 101 – Write a simple memory allocator", Arjun Sreedharan, 2020. [Online]. Available: https://arjunsreedharan.org/post/148675821737/memory-allocators-101-write-a-simple-memory. [Accessed: 20 Aug- 2020].

[2] S. Kanev, S. Xi, G. Wei and D. Brooks, "Mallacc", ACM SIGPLAN Notices, vol. 52, no. 4, pp. 33-45, 2017. Available: 10.1145/3093336.3037736.