CS518 Design of Operating System
Fall 2022

Report

A1: Umalloc()

VISHWAS GOWDIHALLI MAHALINGAPPA
vg421

VIKRAM SAHAI SAXENA
vs799

Design and Implementation of user level malloc() and free() using a 10 MB contiguous character array.

In this project, user-level malloc() and free() are performed on a continuous character array of 10 MB. Upon a user request of malloc (), the number of bytes requested by the user and metadata is stored in a memory block of the 10 MB array. The metadata contains information on the memory block's status (allocated or free), the metadata's size, and the memory block's size. The number of user allocations is maximized by taking different metadata sizes to represent user byte requests, keeping the metadata as small as possible malloc () allocates data in a first-fit manner with the help of metadata information. On a free() call, metadata will indicate whether a memory block is freed. Also, the free() method combines any adjacent free memory blocks into a single free block with the help of metadata information.

umalloc()

The umalloc() function takes three parameters, request_bytes to allocate memory for requested byte size, file_ptr: pointer to the file that calls umalloc(), and line number where umalloc() is called. First, it checks the init variable. If init is the default value, memory gets initialized. If not, memory has already been initialized. It then returns a pointer to a region of memory at least as large as what was requested. If that is not possible, then it will return a NULL pointer.

ufree()

The ufree() function takes three parameters, ptr: pointer to the memory location that we want to free, file_ptr: pointer to the file that calls ufree(), and line number where ufree() is called. The free function will iterate through the entire memory (char array). While iterating, if the pointer to the location of the memory and the pointer provided as input to ufree() match, then the metadata flag in the memory block is set to 0, which makes it available for future memory allocation. In order to avoid fragmentation, free will merge the previous adjacent free blocks (if available) while iterating. This function will not return anything.

Metadata Implementaion:

Metadata size will depend on the number of bytes the user requests during malloc. The size chosen for metadata will be 1 byte for the type unsigned char, 2 bytes for the type unsigned short, and 4 bytes for the type unsigned int. The least significant bit (LSB) will denote whether the memory block is allocated or free (0 – Allocated, 1 – Free). The following two bits to the left of the LSB will denote the metadata size (00 will represent 1 byte, 01 will represent 2 bytes, 10 will represent 4 bytes). The remaining bits in the metadata will contain the memory block size (user-requested bytes + metadata size).

Since we are using 3 bits for storing memory block allocation status and metadata size information, we are left with 5 bits in unsigned char metadata, 13 bits in unsigned short metadata, and 29 bits in unsigned int metadata to store the memory block size. So, we can store a maximum block size of 31 bytes (2^5 – 1) using unsigned char metadata, 8191 bytes

(2^13 – 1) using unsigned short metadata, and 536870811 bytes (2^29 – 1) will using unsigned int metadata.

This information will be used while traversing from one memory block to the next memory block.

Memgrind Results

The umalloc and ufree function calls are evaluated using the following test case scenarios:

1: Consistency:

Testcase:

- *Allocate a small block (1 to 10B), cast it to a type write to it then free it.*
- *Allocate a block of the same size, cast it to the same type, then check to see if the address of the pointers are the same.*

Results:

- *"Consistency test passed" with both the pointers having the same address.*

2: Maximization:

Testcase:

- *Allocate 1B, if the result is not NULL, free it, double the size and try again.*
- *On NULL, halve the size and try again.*
- *On success after a NULL, stop, free it.*

Results:

- *While executing this testcase there will be a point where malloc returns null because of max memory allocation limit, then the size is reduced to half and malloc returns a value which is not null. That means "Maximization test passed".*
- *Our implementation of umalloc() will be able to allocate the maximum memory value of around 8.3MB (i.e., to be exact 8388608 bytes).*

3: Basic Coalescence:

Testcase:

- *Allocate one half of maximal allocation.*
- *Allocate one quarter of maximal allocation.*
- *Free the first pointer.*
- *Free the second pointer.*
- *Attempt to allocate maximal allocation.*
- *On success free it.*

Results:

- *After we free the half and quarter maximum allocation, if we try to allocate the maximal allocation it will succeed. That means "BasicCoalescence test passed".*

4: Saturation:

Testcase:

- *Do 9K 1KB allocations (i.e., do 9216 1024-byte allocations).*
- *Switch to 1B allocations until malloc responds with NULL, that is the saturation of space.*

Results:

- *This test case is used to get the saturation point in our memory.*
- *This saturation point will be used by following test cases to test other scenarios.*

5: Time Overhead:

Testcase:

- *Saturate the memory (obtained from above test case).*
- *Free the last 1B block.*
- *Get the current time.*
- *Allocate 1B.*
- *Get the current time, compute the elapsed time, that will be the max time overhead.*

Results:

- *Using iLAB1: Max time overhead in microseconds = 4386.*
- *Using JetBrains tool: Max time overhead in microseconds = 2050.*

6: Intermediate Coalescence:

Testcase:

- *Saturate the memory (obtained from saturation test case).*
- *free each allocation, one by one.*
- *attempt to allocate your maximal allocation.*
- *On success free all memory.*

Results:

- *After we free all the allocated memory one by one, our umalloc will be able to allocate the maximal allocation. That means "Intermediate Coalescence test passed".*