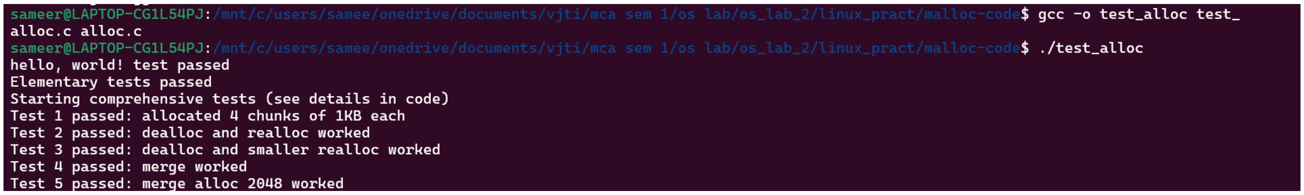
Sameer Gupta
September 25, 2025

LAB 2 : Dynamic Memory Management

Question 1. Build a simple memory manager.

```
The code in alloc.c :

#include "alloc.h"

typedef struct block {
    size_t size;
    int free;
    struct block *next;
} block_t;

#define BLOCK_SIZE sizeof(block_t)

static void *memory = NULL;
static size_t total_size = 0;
static block_t *free_list = NULL;

int init_alloc() {
    total_size = PAGESIZE * 4;
    memory = mmap(NULL, total_size, PROT_READ | PROT_WRITE,
                  MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);
    if (memory == MAP_FAILED)
        return 1;

    free_list = (block_t *)memory;
    free_list->size = total_size;
    free_list->free = 1;
    free_list->next = NULL;
    return 0;
}

int cleanup() {
    if (memory == NULL)
        return 0;
    if (munmap(memory, total_size) != 0)
        return 1;
    memory = NULL;
    free_list = NULL;
    total_size = 0;
    return 0;
}

static void split_block(block_t *block, size_t size) {
    if (block->size >= size + BLOCK_SIZE + MINALLOC) {
        block_t *new_block = (block_t *)((char *)block + size);
        new_block->size = block->size - size;
        new_block->free = 1;
        new_block->next = block->next;

        block->size = size;
        block->free = 0;
        block->next = new_block;
    } else {
        block->free = 0;
    }
}

char *alloc(int requested_size) {
    if (requested_size <= 0 || free_list == NULL)
        return NULL;

    size_t aligned_size = ((requested_size + MINALLOC - 1) / MINALLOC) * MINALLOC;
    size_t total_alloc_size = aligned_size + BLOCK_SIZE;

    block_t *curr = free_list;

    while (curr != NULL) {
        if (curr->free && curr->size >= total_alloc_size) {
            split_block(curr, total_alloc_size);
            return (char *)curr + BLOCK_SIZE;
        }
        curr = curr->next;
    }

    return NULL;
}

static void merge_blocks() {
    block_t *curr = free_list;

    while (curr != NULL && curr->next != NULL) {
        if (curr->free && curr->next->free) {
            curr->size += curr->next->size;
            curr->next = curr->next->next;
        } else {
            curr = curr->next;
        }
    }
}

void dealloc(char *ptr) {
    if (ptr == NULL)
        return;

    if (ptr < (char *)memory || ptr >= (char *)memory + total_size)
        return;

    block_t *block = (block_t *)(ptr - BLOCK_SIZE);
    if ((char *)block < (char *)memory || (char *)block >= (char *)memory + total_size)
        return;

    block->free = 1;
    merge_blocks();
}
```



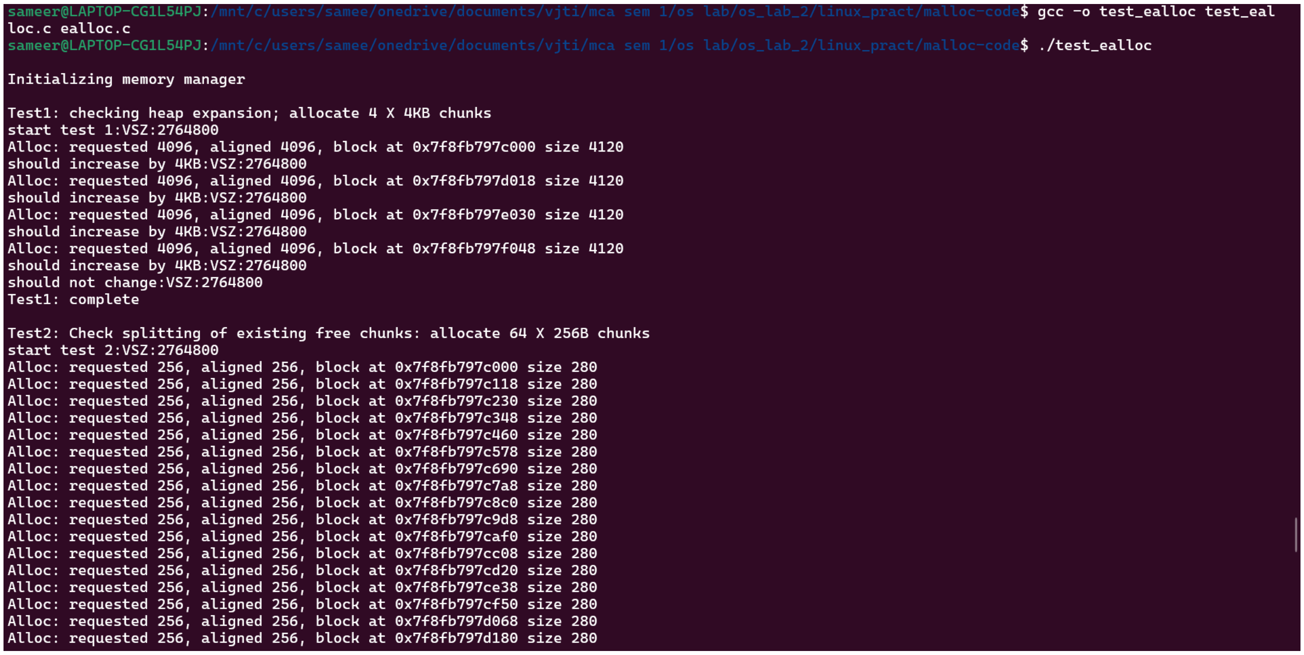
```
sameer@LAPTOP-CG1L54PJ:/mnt/c/users/samee/onedrive/documents/vjti/mca sem 1/os lab/os_lab_2/linux_pract/malloc-code$ gcc -o test_alloc test_
alloc.c alloc.c
sameer@LAPTOP-CG1L54PJ:/mnt/c/users/samee/onedrive/documents/vjti/mca sem 1/os lab/os_lab_2/linux_pract/malloc-code$ ./test_alloc
hello, world! test passed
Elementary tests passed
Starting comprehensive tests (see details in code)
Test 1 passed: allocated 4 chunks of 1KB each
Test 2 passed: dealloc and realloc worked
Test 3 passed: dealloc and smaller realloc worked
Test 4 passed: merge worked
Test 5 passed: merge alloc 2048 worked
```

Figure 1. Alloc

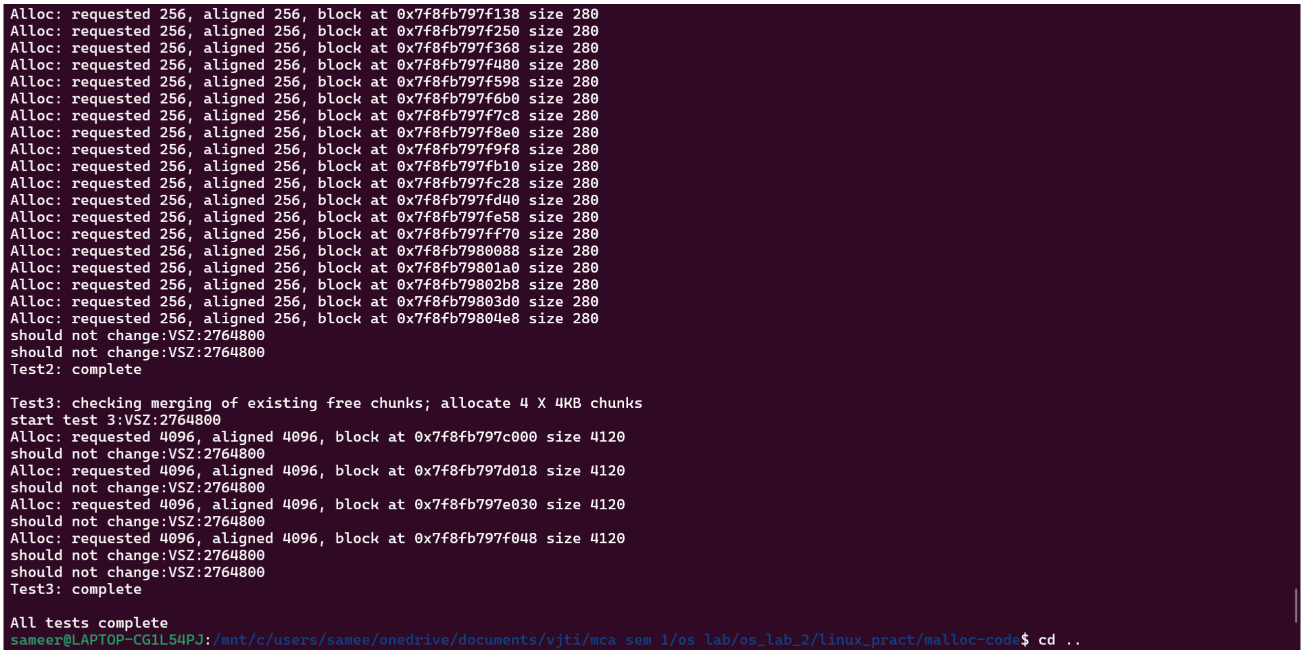
The code in ealloc.c is as follows :

```
#include "ealloc.h"
#include <sys/mman.h>
#include <stdio.h>
#include <stdlib.h>

typedef struct block {
    size_t size;        // size of block including header
    int free;           // free or not
    struct block *next; // next block in free list
} block_t;

#define BLOCK_SIZE sizeof(block_t)

static void *memory = NULL;
static size_t total_size = 0;
static block_t *free_list = NULL;

void init_alloc(void) {
    total_size = PAGESIZE * 5;  // 5 pages (20 KB) to fit overhead
    memory = mmap(NULL, total_size, PROT_READ | PROT_WRITE,
                  MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);
    if (memory == MAP_FAILED) {
        perror("mmap failed");
        exit(1);
    }

    free_list = (block_t *)memory;
    free_list->size = total_size;
    free_list->free = 1;
    free_list->next = NULL;
}
```

```
void cleanup(void) {
    if (memory != NULL) {
        if (munmap(memory, total_size) != 0) {
            perror("munmap failed");
            exit(1);
        }
        memory = NULL;
        free_list = NULL;
        total_size = 0;
    }
}

static void split_block(block_t *block, size_t size) {
    if (block->size >= size + BLOCK_SIZE + MINALLOC) {
        block_t *new_block = (block_t *)((char *)block + size);
        new_block->size = block->size - size;
        new_block->free = 1;
        new_block->next = block->next;

        block->size = size;
        block->free = 0;
        block->next = new_block;
    } else {
        block->free = 0;
    }
}

char *alloc(int requested_size) {
    if (requested_size <= 0 || free_list == NULL)
        return NULL;

    size_t aligned_size = ((requested_size + MINALLOC - 1) / MINALLOC) * MINALLOC;
    size_t total_alloc_size = aligned_size + BLOCK_SIZE;

    block_t *curr = free_list;

    while (curr != NULL) {
        if (curr->free && curr->size >= total_alloc_size) {
            split_block(curr, total_alloc_size);
            printf("Alloc: requested %d, aligned %zu, block at %p size %zu\n",
                   requested_size, aligned_size, (void *)curr, curr->size);
            return (char *)curr + BLOCK_SIZE;
        }
        curr = curr->next;
    }

    printf("Alloc failed: no sufficient block for %d bytes\n", requested_size);
    return NULL;
}

static void merge_blocks() {
    block_t *curr = free_list;

    while (curr != NULL && curr->next != NULL) {
        if (curr->free && curr->next->free) {
            curr->size += curr->next->size;
            curr->next = curr->next->next;
        } else {
            curr = curr->next;
        }
    }
}

void dealloc(char *ptr) {
    if (ptr == NULL)
        return;

    if (ptr < (char *)memory || ptr >= (char *)memory + total_size)
        return;

    block_t *block = (block_t *)(ptr - BLOCK_SIZE);

    if ((char *)block < (char *)memory || (char *)block >= (char *)memory + total_size)
        return;

    block->free = 1;
    merge_blocks();
}
```



```
sameer@LAPTOP-CG1L54PJ:/mnt/c/users/samee/onedrive/documents/vjti/mca sem 1/os lab/os_lab_2/linux_pract/malloc-code$ gcc -o test_ealloc test_eal
loc.c ealloc.c
sameer@LAPTOP-CG1L54PJ:/mnt/c/users/samee/onedrive/documents/vjti/mca sem 1/os lab/os_lab_2/linux_pract/malloc-code$ ./test_ealloc

Initializing memory manager

Test1: checking heap expansion; allocate 4 X 4KB chunks
start test 1:VSZ:2764800
Alloc: requested 4096, aligned 4096, block at 0x7f8fb797c000 size 4120
should increase by 4KB:VSZ:2764800
Alloc: requested 4096, aligned 4096, block at 0x7f8fb797d018 size 4120
should increase by 4KB:VSZ:2764800
Alloc: requested 4096, aligned 4096, block at 0x7f8fb797e030 size 4120
should increase by 4KB:VSZ:2764800
Alloc: requested 4096, aligned 4096, block at 0x7f8fb797f048 size 4120
should increase by 4KB:VSZ:2764800
should not change:VSZ:2764800
Test1: complete

Test2: Check splitting of existing free chunks: allocate 64 X 256B chunks
start test 2:VSZ:2764800
Alloc: requested 256, aligned 256, block at 0x7f8fb797c000 size 280
Alloc: requested 256, aligned 256, block at 0x7f8fb797c118 size 280
Alloc: requested 256, aligned 256, block at 0x7f8fb797c230 size 280
Alloc: requested 256, aligned 256, block at 0x7f8fb797c348 size 280
Alloc: requested 256, aligned 256, block at 0x7f8fb797c460 size 280
Alloc: requested 256, aligned 256, block at 0x7f8fb797c578 size 280
Alloc: requested 256, aligned 256, block at 0x7f8fb797c690 size 280
Alloc: requested 256, aligned 256, block at 0x7f8fb797c7a8 size 280
Alloc: requested 256, aligned 256, block at 0x7f8fb797c8c0 size 280
Alloc: requested 256, aligned 256, block at 0x7f8fb797c9d8 size 280
Alloc: requested 256, aligned 256, block at 0x7f8fb797caf0 size 280
Alloc: requested 256, aligned 256, block at 0x7f8fb797cc08 size 280
Alloc: requested 256, aligned 256, block at 0x7f8fb797cd20 size 280
Alloc: requested 256, aligned 256, block at 0x7f8fb797ce38 size 280
Alloc: requested 256, aligned 256, block at 0x7f8fb797cf50 size 280
Alloc: requested 256, aligned 256, block at 0x7f8fb797d068 size 280
Alloc: requested 256, aligned 256, block at 0x7f8fb797d180 size 280
```



```
Alloc: requested 256, aligned 256, block at 0x7f8fb797f138 size 280
Alloc: requested 256, aligned 256, block at 0x7f8fb797f250 size 280
Alloc: requested 256, aligned 256, block at 0x7f8fb797f368 size 280
Alloc: requested 256, aligned 256, block at 0x7f8fb797f480 size 280
Alloc: requested 256, aligned 256, block at 0x7f8fb797f598 size 280
Alloc: requested 256, aligned 256, block at 0x7f8fb797f6b0 size 280
Alloc: requested 256, aligned 256, block at 0x7f8fb797f7c8 size 280
Alloc: requested 256, aligned 256, block at 0x7f8fb797f8e0 size 280
Alloc: requested 256, aligned 256, block at 0x7f8fb797f9f8 size 280
Alloc: requested 256, aligned 256, block at 0x7f8fb797fb10 size 280
Alloc: requested 256, aligned 256, block at 0x7f8fb797fc28 size 280
Alloc: requested 256, aligned 256, block at 0x7f8fb797fd40 size 280
Alloc: requested 256, aligned 256, block at 0x7f8fb797fe58 size 280
Alloc: requested 256, aligned 256, block at 0x7f8fb797ff70 size 280
Alloc: requested 256, aligned 256, block at 0x7f8fb7980088 size 280
Alloc: requested 256, aligned 256, block at 0x7f8fb79801a0 size 280
Alloc: requested 256, aligned 256, block at 0x7f8fb79802b8 size 280
Alloc: requested 256, aligned 256, block at 0x7f8fb79803d0 size 280
Alloc: requested 256, aligned 256, block at 0x7f8fb79804e8 size 280
should not change:VSZ:2764800
should not change:VSZ:2764800
Test2: complete

Test3: checking merging of existing free chunks; allocate 4 X 4KB chunks
start test 3:VSZ:2764800
Alloc: requested 4096, aligned 4096, block at 0x7f8fb797c000 size 4120
should not change:VSZ:2764800
Alloc: requested 4096, aligned 4096, block at 0x7f8fb797d018 size 4120
should not change:VSZ:2764800
Alloc: requested 4096, aligned 4096, block at 0x7f8fb797e030 size 4120
should not change:VSZ:2764800
Alloc: requested 4096, aligned 4096, block at 0x7f8fb797f048 size 4120
should not change:VSZ:2764800
should not change:VSZ:2764800
Test3: complete

All tests complete
sameer@LAPTOP-CG1L54PJ:/mnt/c/users/samee/onedrive/documents/vjti/mca sem 1/os lab/os_lab_2/linux_pract/malloc-code$ cd ..
```

Figure 2. ealloc

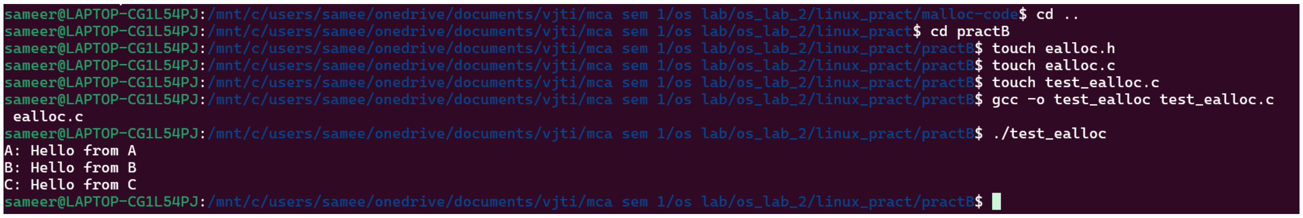
Question 2. Expandable Heap.

The code in our custom memory allocator (ealloc.c) :

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/mman.h>
#include <string.h>
#include "ealloc.h"

#define PAGE_SIZE 4096
#define MAX_PAGES 4

typedef struct Chunk {
    size_t size;
    int is_free;
    struct Chunk *next;
} Chunk;

typedef struct Page {
    void *addr;
    Chunk *free_list;
} Page;

static Page pages[MAX_PAGES];
static int page_count = 0;

void init_alloc() {
    page_count = 0;
    for (int i = 0; i < MAX_PAGES; i++) {
        pages[i].addr = NULL;
        pages[i].free_list = NULL;
    }
}

void cleanup() {
    init_alloc(); // Reset state; no need to munmap
}

char *alloc(int size) {
    if (size <= 0 || size > PAGE_SIZE || size % 256 != 0) return NULL;

    for (int i = 0; i < page_count; i++) {
        Chunk *curr = pages[i].free_list;
        while (curr) {
            if (curr->is_free && curr->size >= size) {
                if (curr->size >= size + sizeof(Chunk) + 256) {
                    Chunk *new_chunk = (Chunk *)((char *)curr + sizeof(Chunk) + size);
                    new_chunk->size = curr->size - size - sizeof(Chunk);
                    new_chunk->is_free = 1;
                    new_chunk->next = curr->next;
                    curr->next = new_chunk;
                    curr->size = size;
                }
                curr->is_free = 0;
                return (char *)curr + sizeof(Chunk);
            }
            curr = curr->next;
        }
    }

    if (page_count >= MAX_PAGES) return NULL;

    void *addr = mmap(NULL, PAGE_SIZE, PROT_READ | PROT_WRITE,
                      MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);
    if (addr == MAP_FAILED) return NULL;

    Chunk *chunk = (Chunk *)addr;
    chunk->size = size;
    chunk->is_free = 0;
    chunk->next = NULL;

    pages[page_count].addr = addr;
    pages[page_count].free_list = chunk;

    if (size + sizeof(Chunk) < PAGE_SIZE) {
        Chunk *free_chunk = (Chunk *)((char *)addr + sizeof(Chunk) + size);
        free_chunk->size = PAGE_SIZE - size - sizeof(Chunk);
        free_chunk->is_free = 1;
        free_chunk->next = NULL;
        chunk->next = free_chunk;
    }

    page_count++;
    return (char *)chunk + sizeof(Chunk);
}

void dealloc(char *ptr) {
    if (!ptr) return;

    Chunk *chunk = (Chunk *)(ptr - sizeof(Chunk));
    chunk->is_free = 1;

    if (chunk->next && chunk->next->is_free) {
        chunk->size += sizeof(Chunk) + chunk->next->size;
        chunk->next = chunk->next->next;
    }
}
```

The code in test_ealloc.c :

```
#include <stdio.h>
#include <string.h>
#include "ealloc.h"

int main() {
    init_alloc();

    char *a = alloc(1024);
    char *b = alloc(512);
    char *c = alloc(256);

    if (a && b && c) {
        strcpy(a, "Hello from A");
        strcpy(b, "Hello from B");
        strcpy(c, "Hello from C");

        printf("A: %s\n", a);
        printf("B: %s\n", b);
        printf("C: %s\n", c);
    }

    dealloc(b);
    char *d = alloc(512);
    if (d) {
        strcpy(d, "Hello from D");
        printf("D: %s\n", d);
    }
    cleanup();
    return 0;
}
```



```
sameer@LAPTOP-CG1L54PJ:/mnt/c/users/samee/onedrive/documents/vjti/mca sem 1/os lab/os_lab_2/linux_pract/malloc-code$ cd ..
sameer@LAPTOP-CG1L54PJ:/mnt/c/users/samee/onedrive/documents/vjti/mca sem 1/os lab/os_lab_2/linux_pract$ cd practB
sameer@LAPTOP-CG1L54PJ:/mnt/c/users/samee/onedrive/documents/vjti/mca sem 1/os lab/os_lab_2/linux_pract/practB$ touch ealloc.h
sameer@LAPTOP-CG1L54PJ:/mnt/c/users/samee/onedrive/documents/vjti/mca sem 1/os lab/os_lab_2/linux_pract/practB$ touch ealloc.c
sameer@LAPTOP-CG1L54PJ:/mnt/c/users/samee/onedrive/documents/vjti/mca sem 1/os lab/os_lab_2/linux_pract/practB$ touch test_ealloc.c
sameer@LAPTOP-CG1L54PJ:/mnt/c/users/samee/onedrive/documents/vjti/mca sem 1/os lab/os_lab_2/linux_pract/practB$ gcc -o test_ealloc test_ealloc.c
 ealloc.c
sameer@LAPTOP-CG1L54PJ:/mnt/c/users/samee/onedrive/documents/vjti/mca sem 1/os lab/os_lab_2/linux_pract/practB$ ./test_ealloc
A: Hello from A
B: Hello from B
C: Hello from C
sameer@LAPTOP-CG1L54PJ:/mnt/c/users/samee/onedrive/documents/vjti/mca sem 1/os lab/os_lab_2/linux_pract/practB$
```

Figure 3. Output after testing