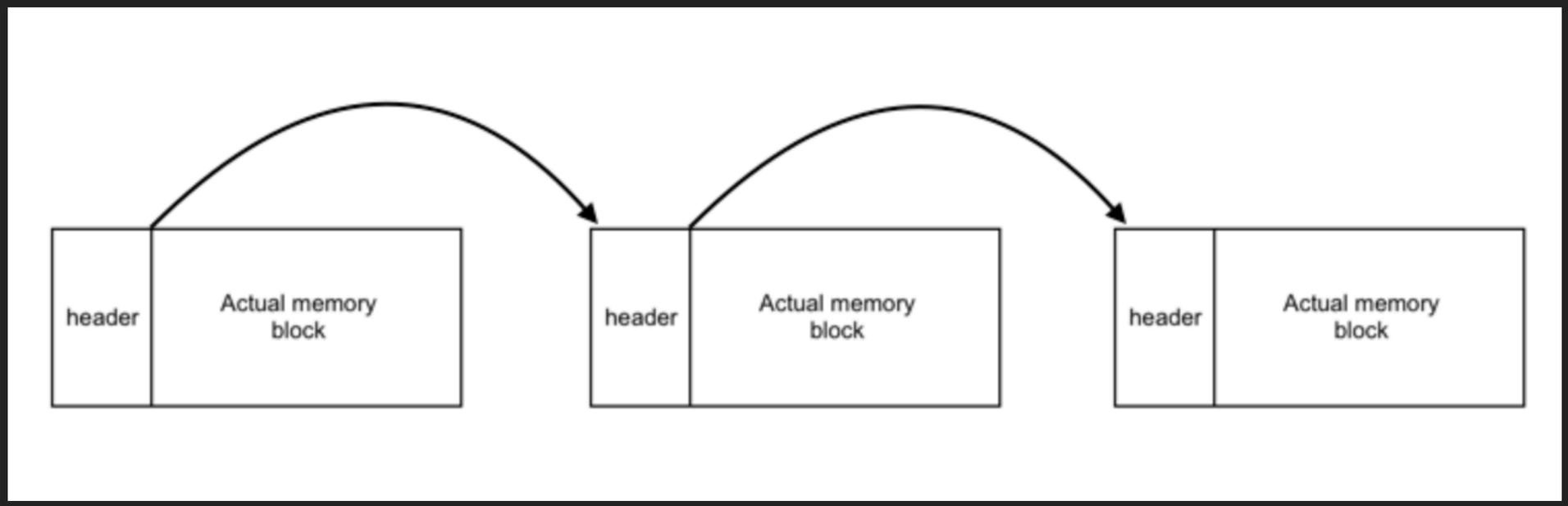
HUSEYIN MERT YENILMEZ

NEW MEMORY ALLOCATOR

LINKED LIST FOR FIRST-FIT APPROACH

```
struct header {
    size_t size;
    unsigned is_free;
    struct header_t *next;
};

struct header *head = NULL, *tail = NULL;
```



- http://arjunsreedharan.org/post/148675821737/write-a-simple-memory-allocator

GETTING FREE BLOCK

```
struct header *get_free_block(size_t size)
{
    struct header_t *curr = head; // current head node
    while(curr) {
        if (curr->is_free && curr->size >= size) // if there's a free block with requested size
            return curr;
        curr = curr->next;
    }
    return NULL;
}
```

TEDU_ALLOC

```
void *TEDU_alloc(size_t size)
{
    size_t total_size;
    void *block;
    struct header *header;
    if (!size) // We check if the requested size is zero. If it is, then we return NULL.
        return NULL;
    header = get_free_block(size);
    if (header) {

        header->is_free = 0;
        return (void*)(header + 1);
    }

    total_size = sizeof(struct header_t) + size;
    block = mmap (0, total_size, PROT_READ | PROT_WRITE,MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);
     //the OS to increment the program break total_size
    return (void*)(header + 1);
}
```

TEDU_FREE

```
void TEDU_free(void *block)
{
    struct header_t *header, tmp*;

    void *programbreak; // program break is the end of the process's data segment

    if (!block)
        return;
    header = (struct header*)block - 1; //we get the header of the block we want to free
                                        //get a pointer that is behind the block

    /* mmap gives the current program break address */
    /* mmap with a negative argument decrements the program break.*/

    programbreak = mmap (0,0, PROT_READ | PROT_WRITE, MAP_PRIVATE | MAP_ANONYMOUS, -1, 0);

    if ((char*)block + header->size == programbreak) { // Check if the block to be freed is the last one in the linked
list.
     //we could decrease the size of the heap and release memory back to OS
     //header->size - sizeof(struct header_t) is the sum of sizes of the header and the acutal block

     mmap (0, 0 - header->size - sizeof(struct header_t), PROT_READ | PROT_WRITE, MAP_PRIVATE | MAP_ANONYMOUS,-1,0);

     }



}
```

PRINT METHOD

```
void TEDU_GetStats()
{
    struct header_t *curr = head;
    printf("beginning, head = %p, next = %p \n", (void*)head, (void*)tail);
    while(curr) {
        printf("current block = %p, size = %zu, is_free=%u, next block =%p\n",
               (void*)curr, curr->size, curr->is_free, (void*)curr->next);
        curr = curr->next;
    }
}
```

MAIN METHOD

```
int main(int argc, char* argv[])
{
    int *a;
    a = TEDU_alloc(10);
    TEDU_free(TEDU_alloc(5));
    TEDU_GetStats();
}
```

- Output:

```
beginning, head = 0x10019a000, next = 0x10019b000
current block = 0x10019a000, size = 10, is_free=0, next block =0x10019b000
current block = 0x10019b000, size = 5, is_free=1, next block =0x0
Program ended with exit code: 0
```

LIBRARY FILE AND COMPILING

LIBRARY FILE

```
gcc -o TEDU_alloc.so -fPIC -shared TEDU_alloc.c
export LD_PRELOAD=$PWD/TEDU_alloc.so
```

COMPILING

```
gcc -o TEDU_alloc TEDU_alloc.c
```