International Institute of Information Technology
Bangalore

Microprocessors and Micro-controllers
ECE 302

Design Document

Mohd. Rizwan Shaikh
(IMT2019513)

September 5, 2021

1 Overview:

The whole implementation is based on building blocks of memory defined by *struct block*. We start by creating a big block of memory and split it into two parts as per the need. The details are mentioned below.

2 Memory Implementation

We create a global array '*memory*' and a '*memory_list*' which points to the memory array. We define memory blocks as follows:

```
//Struct that defines blocks of memory
typedef struct block
{
    size_t size;
    int free;
    struct block *next;
} block;
```

This struct creates a linked list of memory blocks. The size of each block is defined by user while using *my_malloc* and *my_calloc*. We also have a variable '*free*' which is set to one if the block is not used. Else it is set to zero.

We have three memory management functions which are helper functions for *my_malloc*, *my_calloc* and *my_free*.

The first memory management function that we have is *void initialize()*. We call this function in the start to create a big block of memory to use.

The next memory management function is '*void split_big_space(block *big_space, size_t size)*'. The function takes a big block of memory and partitions it into two blocks. The size of first block is equal to the input parameter 'size'. This is the block that is to be used and hence we make '*free*' as 0. The size of the other block is the remaining size. This block is the last block of the memory_list. This is the free block of memory and can be used later to allocate memory.

The last memory management function is '*void merge_free_blocks()*'. This function is used while freeing memory. What it does is it checks the neighbouring block of memory and if the consecutive blocks are free, then it merges them to one.

3 My_Malloc

In this function, we first initialize the memory if it is not initialized. Then we iterate the linked list of memory and find the first block of memory which is unoccupied and has size greater than or equal to the desired size. If the size of the free block is equal to the required size, we occupy the block. If the size of free block is greater than the required size, we split it into two blocks. One of those blocks is occupied and other is the free block for using later.

4 My_Calloc

This is a fairly straightforward function in which we first create blocks of memory with the help of '*my_malloc*'. Then we use '*memset*' to initialize the whole space with zeros.

5 My_heap_free_space

In this function, we iterate through the memory linked list and find the total size of the occupied blocks. We subtract this value from the size of the block and return it.

6 My_Free

In this function, if a valid pointer is provided, it sets the free variable to 1 indicating that this block of memory is free to use. We call '*merge_free_blocks()*' function to combine consecutive free blocks of memory if there are any.