CS31202 - OPERATING SYSTEMS LABORATORY

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Assignment 6 : Memory Management

Authors:
Burra Nithish (20CS10018)
Kartik Pontula (20CS10031)
Astitva (20CS30007)
Rahul Mandal (20CS30039)

Date: April 14, 2023

1 Page Table Structure

Our page table is an attribute of the `GoodMallocMemory` class defined as follows:

```
    map<std::string, PTEntry> PT;
```

It is a C++ map that maps the name of each list to a PTEntry struct. Scoping of list names across different function calls is explained later.
The definition of PTEntry is:

```
struct PTEntry
{
    int head;
    int tail;
    int scope;
};
```

- `head:` Frame number of the head of the doubly linked list.
- `tail:` Frame number of the tail of the doubly linked list.
- `scope:` A binary value that indicates whether this list is within scope (1) or not (0).

2 Data Structures used

2.1 Required Structures

A structure for the ”Element” and a ”varStack”

```
struct _Element
{
    long data: 64;
    int prev: 32;
    int next: 32;
} Element;

stack<string> varStack;
```

- `data:` The data to be stored in the ”Element”.
- `prev:` The logical index of the previous ”Element”.
- `next:` The logical index of the next ”Element”.
- `varStack:` The global stack used to keep track of which memory is not needed anymore.

2.2 Additional Structures

Additionally, the following have been declared

```
    string scopeStr;
    int freeFrameHead;
    int freeFrameTail;
    size_t freeFrameCount;
```

The freeFrameHead, freeFrameTail, freeFrameCount all correspond to the free implicit list.

- `scopeStr:` A prefix to scope the list names.
- `freeFrameHead:` Holds the logical index to the beginning of the implicit free list. All the free nodes are linked to each other, so traversal using only the beginning can be done for assigning free nodes.
- `freeFrameTail:` Holds the logical index to the end of the implicit free list.
- `freeFrameCount:` Has the number of free frames in the list.

3 Memory Impact of using `freeElem()`

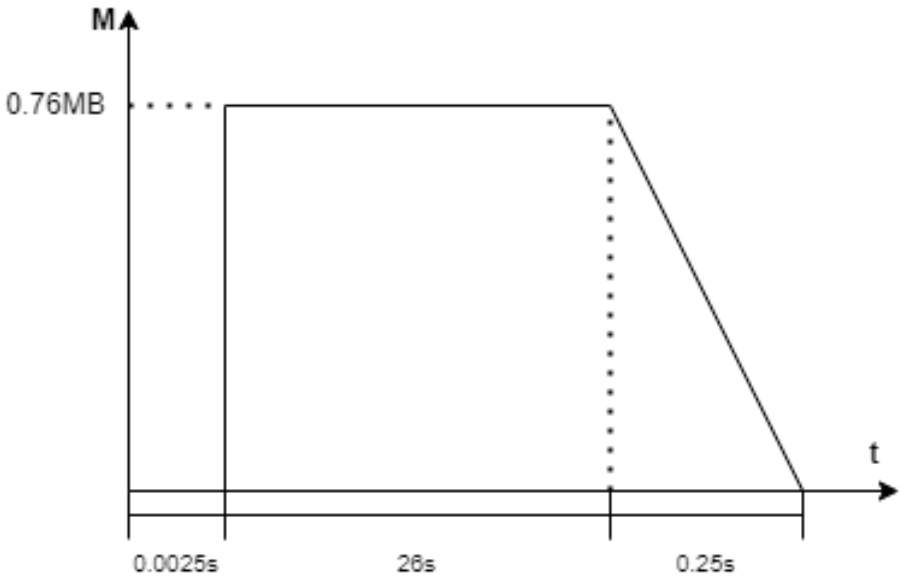

Figure 1: Memory footprint with freeElem()

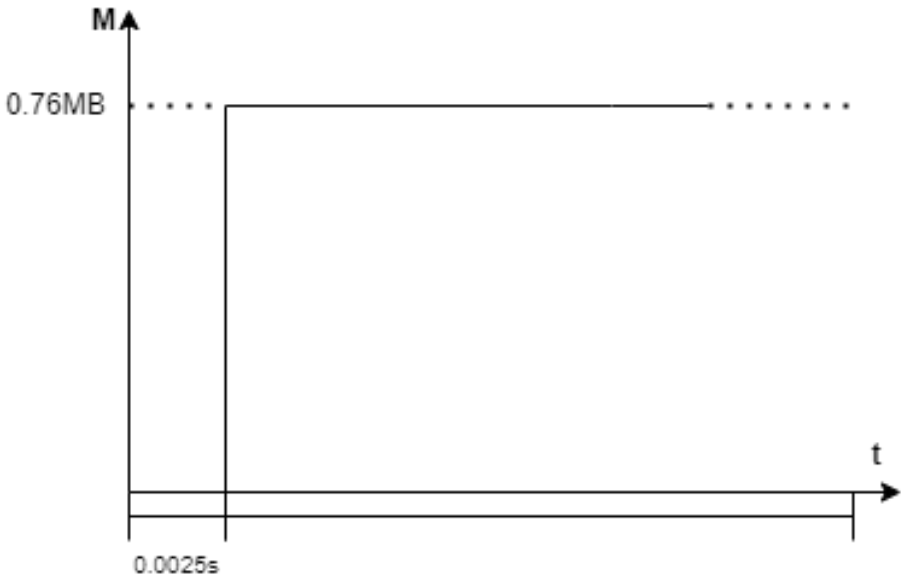

Figure 2: Memory footprint without freeElem()

4 Code Structure Dependencies

In terms of code structure, using object-oriented programming (OOP) principles and design patterns such as the Singleton pattern can help to minimize complexity and improve performance. OOP allows for the encapsulation of data and methods, which can improve maintainability and reduce errors.
Additionally, minimizing unnecessary function calls and avoiding memory leaks through proper use of the createList() and freeElem() functions will be important for maximizing performance.
Overall, the performance of the code will be maximized by efficient memory management and minimizing unnecessary operations, while minimizing performance will occur through inefficient memory management and excessive function calls.

5 Usage of Locks

No locks have been used as there wasn't any requirement with the data structures chosen and the strategy implemented. This is a single-threaded program that eliminated the possibility of any race conditions in any scenario which eliminates the usage of locks.