DESIGN DOC

Group 32
Project Contributors:
Aditya Pandiri
Neeraj Boddeda
Mohan Kishore Chilukuri
Gitanjali Gupta

---

a. What is the structure of your internal page table? Why?

Each page has a size of PAGE_SIZE (2048 bytes) and can store a fixed number of "element" structures (170 elements per page) as the element size is 12. Our program allocates memory in whole pages and keeps track of which pages are allocated in the "page_table" data structure .“page_table” data structure we made is a simple vector that would depict the availability of each of the pages. Since initially, all are available, we set all of them as 1. Once that page is given to a list, we set its value to 0. This is done to maintain the list of all free and available pages so that we can use the pages effectively. When a new linked list is created, the program finds enough free pages to hold the list and marks those pages as allocated in the page table. Our program also tracks which pages are allocated to which linked lists using the pages_allocated vector in the entry structure.

b. What are additional data structures/functions used in your library? Describe all with justifications.

“allocation_stack” and “global_stack” are 2 vectors of the entry element. The prior one is used to maintain the stack of all the lists created through the main or any other function and the latter one stores the global lists only. “entry” element consists of a string name representing the list and then a vector of all the pages allocated to it. We maintain the scope of the list by using another function called “initScope” which inserts a dummy entry element into the stack to signify the scope of the entries being pushed in. This dummy variable helps us free memory of the local variables since we pop until we encounter that dummy entry(when the list name is not explicitly mentioned as an argument when freeElem is called).

c. What is the impact of freeElem() for merge sort. Report the memory footprint and running time with and without freeElem (average over 100 runs).

freeElem() helps to clear up the local variables once their scope ends. This helps in the more effective use of memory space, since when the local variable's scope gets over, and if we don't free that memory, we might simply waste those occupied spaces leading to less effective memory usage.

Memory footprint and running time with and without freeElem():

Maximum Number of pages in our memory block that are allocated with freeElem() = 295

Maximum Number of pages in our memory block that are allocated without freeElem() = 102215

The difference is = 102215-295= 101920

Therefore memory footprint in MB= (101920)*(2048)/1024*1024=199.0625MB

NOTE: All the above values are averaged over 100 runs

d. In what type of code structure will this performance be maximised, where will it be minimised? Why?

If a code is recursively making use of the creation of local variables, then this design will give us a better memory usage plan.
Whereas in codes where there are multiple global variables or many lists in the same scope, this code structure will have the least performance in this case.

e. Did you use locks in your library? Why or why not?

No locks are used in our library since we are not creating any thread explicitly for any of the processes. Hence no process synchronisation is required.