OS LAB ASSIGNMENT-6 REPORT

GROUP-2

SARTHAK NIKUMBH SHAM: 20CS30035
AYUSH KUMAR DWIVEDI: 20CS10084
SAPTARSHI DE CHAUDHURY: 20CS10080
RITWIK RANJAN MALLIK: 20CS10049

a. Structure of internal page table:

We have designed our page table to be a map. So, our page table maps the <string name> to <my_struct pointer>, which gives us the starting address of the list “name” in main memory. More importantly, we have maintained a map/ page table scope-wise.
As we know that the complexity for insert, remove and lookup for a map is O(log(N)), therefore this structure of the page table is a great improvement over a normal page table where lookup can take as worse as a linear time of (O(N)).

b. Additional data structures:

The main data structure we have used for implementing all page tables is:

map<scope, map<list_name, my_struct *>>

where, **scope** denotes the scope of page table
list_name denotes the name of the list created
my_struct * is a pointer to my_struct structure, which has following entries:

int **size**: size of list + its header
int **start**: start index which points to the header
int **end**: last element of list

It is to be noted that the first 31 bits of header of a list denote the size of the segment (list+header) and its last bit specifies if this segment is occupied or free. Using this approach helps to avoid storage of a separate variable for storing the availability status of the segment.

Moreover, we have justified our reasoning for using map as appropriate data structure for page table in part (a).

c. Impact of freeElem():

The running time (averaged over 100 runs) of our mergesort.cpp in as follows:

Without freeElem(): **6963.84 ms**
With freeElem(): **7394.57 ms**

The memory footprint has been reported using the memoryFootprint() function in our code. This function is called at the end of merge() function. It reports the current allocated memory size created using createMem() function. Henceforth, the memory footprint is reported in the code output itself.

d. Performance:

The performance will be maximized in code structures where each scope would create as few lists as possible. This is because we have maintained a page table for each scope as a map, and it would be much faster to search for a created list in a map with a lesser number of elements. Our implementation of mergesort follows this code structure.
On the other hand, the performance will be minimized when each scope will have several lists created and freed. It follows from the same logic as above.

e. Locks:

We have used locks for 2 purposes:

- Locks on print statements ensure proper and synchronized printing of library messages.
- Locks are used while accessing created memory and other internal data structures. This is important as a user of this library may spawn a multi-threaded process which may require atomic access of these resources.

The locks used in our program are the following:

- printMutex: mutex for print statements
- memMutex: mutex for locking memory created
- mapMutex: mutex for locking page table structure