Operating Systems Laboratory Assignment 5

Group No - 53

19CS30050 - Suryam Arnav Kalra

19CS30025 - Kunal Singh

Ques 1 - What is the structure of your internal page table? Why?

Ans - Our page table is a contiguous block of memory with each page of size 4 bytes. The ith page in the page table maps to a frameAddress on giving the logical address to it. Each frame is of size 4 bytes and each page is also of size 4 bytes therefore each page contains the address of a single frame. This is done in order to be in consistency with word alignment because of this we are accessing memory everywhere in chunks of 4 bytes only. Frames of size 4 bytes store the value of different types of variables and add redundant space when required for word alignment.

Ques 2 - What are additional data structures/functions used in your library. Describe all with justifications.

Ans -

The Data Structures that we used are :

1. varToPage - This structure is used as an element in the vP array. The purpose of this element is to map the variable name to a page number which contains the address of the frame which stores the value of that variable. The structure contains following items:
    - name - character array to store the name of the variable.
    - type - integer to store type of the variable (1 - INT, 2 - CHAR, 3 - MEDINT, 4 - BOOL)
    - localcounter - value of the localcounter at the time of creation of variable, with the help of this we can find the page number associated with the variable.
2. pageToVar - This structure is used as an element in the pV array. The purpose of this element is to map page number to variable names, it is a reverse map for varToPage.
   The ith index of this array contains name of the variable for which the information is in page number i The structure contains following items:
    - name - character array to store name of the variable.
3. arrSz- This structure is used as an element in the aS array. The purpose of this element is to store the size of the array and name of the array. The structure contains following items:
    - name - character array to store name of the array.
    - sz - size of the array.

4. mark- This structure is used as an element in the mk array. The purpose of this element is to store whether the variable is marked for sweeping phase or not. The structure contains following items:
    - name - character array to store name of the variable or array.
    - is_mark - integer to indicate marking (0 - not marked, 1 - marked).
5. freeFrame- This structure is used as a stack. The purpose of this stack is to store the actual value of the variables. The structure contains following items:
    - a - array where each element is bytes and used to store values of variables.
    - cnt - stores the number of frames occupied

Our library file contains the following functions:

1. min
   The function takes the following input:
    - a - first integer value for comparison
    - b - second integer value for comparison

   The function does the following job:
    - returns the minimum of the two integers provided as parameters.
2. push
   The function takes the following input:
    - val - value to push into the stack

   The function does the following job:
    - inserts the value in the stack
    - increments the count value for the stack
3. init_stack
   The function takes the following input:
    - sz - size of the stack

   The function does the following job:
    - allocates memory for the stack
    - initializes the count value of stack to 0
    - pushes (sz/4) free frames to the stack

4. top
   The function does the following job:
    - returns the top element of the stack
5. pop
   The function does the following job:
    - returns the top element of the stack, removes it from the stack and decrease the count of number of elements in the stack

6. search_varToPage
   The function takes the following input:
   - name - name of the variable to search

   The function does the following job:
   - Iterate through the varToPage array and match the name with every entry
   - if the name given to the function is found then return the index
   - else return -1
7. insert_varToPage
   The function takes the following input:
   - name - name of the variable to search
   - type - the datatype of the variable
   - lc - localcounter value

   The function does the following job:
   - Iterate through the varToPage array and check if type = -1 (implies empty in our case) we put the name , type and lc value passed to the function at this index.

8. getFreeFrame
   The function does the following job:
   - returns the top element of the stack, that is the first free frame
9. removeFreeFrame
   The function does the following job:
   - pop one element from the stack, that is remove one free frame from the stack.
10. insert_pageToVar
    The function takes the following input:
    - name - name of the variable to insert in pageToVar
    - lc - index at which we want to insert the name

    The function does the following job:
    - Inserts the name at the given index
11. remove_varToPage
    The function takes the following input:
    - name - name of the variable or array

    The function does the following job:
    - Iterates through the vP array and when the variable or array with the name is found, it's name is set to NULL, type to -1 and localcounter to -1
12. remove_pageToVar
    The function takes the following input:
    - lc - index from which we want to remove the name

    The function does the following job:

- assign NULL to the pV array at the given index

13. search_arrSz

    The function takes the following input:
    - name - name of the array which we want to find

    The function does the following job:
    - Iterates through the aS array and checks if any entry with the same name exists.
    - if found we return the index
    - if not found we return -1.

14. insert_freeFrame

    The function takes the following input:
    - lc - index of the frame to push

    The function does the following job:
    - pushes into the freeframes stack with value lc

15. remove_arrSz

    The function takes the following input:
    - name - name of the variable or array

    The function does the following job:
    - Iterates through the aS array and when the variable or array with the name is found, it's name is set to NULL and size to -1

16. remove_mark

    The function takes the following input:
    - name - name of the variable which we want to unmark.

    The function does the following job:
    - Iterate through the mk array check the name of the variable and set is_mark = 0 for that variable at the found place and return.

17. sweepPhase

    The function takes the following input:
    - name - name of the variable to sweep

    The function does the following job:
    - find the page number , data type and logicalAddress of the variable passed as parameter from the varToPage map(array of struct in our case).
    - removes the mapping from varToPage map and also from pageToVar map, because the variable is to be deleted.
    - check if the variable name is of array or not.
    - if name is not of array then frees the frame allocated for this variable and also empties the page for this variable.
    - if the name is an array then we first find the size of that array.
    - free all the frames and empties all the pages given to that array.

- at the end unmarks the variable because now it is sweeped from our memory.

18. compactPhase

    The function does the following job:

    - iterates over the page table and tries to find an empty hole.
    - marks the found position.
    - For any page after the found position tries to move it as up in the page table as possible in order to fill all the holes.
    - If the variable that we encounter during this iteration is an array then we will move all the elements of that variable up by the number of holes that we have in the page table from the start.
    - At the end we decrease the localcounter by 4*number of holes encountered during compaction, this way all the holes are aggregated at the end and then discarded.

19. clear_mark

    The function does the following job:

    - Iterates through every entry in mk array and makes the name of the variable as NULL and the is_mark flag as 0.

20. gc_run

    The function does the following job:

    - first of all applies mutex lock
    - for every variable that is marked for sweeping during freeElem, the function calls sweepPhase with that variable name.
    - since the whole marked array is now traversed we can clear the mark array
    - call compactPhase for compaction because now every variable that is of no use is sweeped.
    - applies mutex unlock

21. gc_help

    This is the function of the garbage collector thread. The function does the following job:

    - Runs in a infinite while loop
    - Inside the while loop sleeps for 0.001 seconds and then calls gc_run for garbage collection.

22. gc_initialize

    The function does the following job:

    - Initializes mutex locks
    - creates garbage collector thread

- if creation fails, terminate the program otherwise return.

23. initialize_pageTable

    The function takes the following input:
    - sz - Integer for the size of the page table

    The function does the following job:
    - Allocates memory to the page table.
    - For every page in the page table marks it as a free page and returns

24. initialize_varToPage

    The function does the following job:
    - Allocates memory to the vP array.
    - For every entry in the vP array mark it as a free entry with NULL name, -1 type and -1 localcounter.

25. initialize_pageToVar

    The function does the following job:
    - Allocates memory to the pV array.
    - For every entry in the pV array mark it as a free entry with a NULL name.

26. initialize_arrSz

    The function does the following job:
    - Allocates memory to the aS array.
    - For every entry in the aS array mark it as a free entry with a NULL name and -1 size.

27. initialize_mark

    The function does the following job:
    - Allocates memory to the mk array.
    - For every entry in the mk array mark it as a free entry with a NULL name and is_mark = 0.

28. getVar

    The function takes the following input:
    - name - name of the variable for which user wants the value.

    The function does the following job:
    - First of all check in the varToPage map if a variable with the same name exists, if not then return after printing an error message.
    - Extract the data type from varToPage and check it with the user provided data type if it doesn't match then print the error and return.
    - Extract the logical address of the variable from varToPage map.
    - Extract the frame address by giving a logical address to the page table.

- store the value from the frame address to a temporary variable and return the value.

29. insert_arrSz

    The function takes the following input:
    - name - name of the array.
    - sz - size of the array

    The function does the following job:
    - Iterate through the aS array and insert the array name with size in the first found place in aS array.

30. getArrElem

    The function takes the following input:
    - name - name of the array for which user wants the value.
    - offset - position of the element in the array

    The function does the following job:
    - First of all check in the varToPage map if a variable with the same name exists, if not then return after printing an error message.
    - Extract the data type from varToPage and check it with the user provided data type if it doesn't match then print the error and return.
    - Extract the logical address of the variable from varToPage map using the name of array and the offset.
    - Extract the frame address by giving a logical address to the page table.
    - store the value from the frame address to a temporary variable and return the value.

31. insert_mark

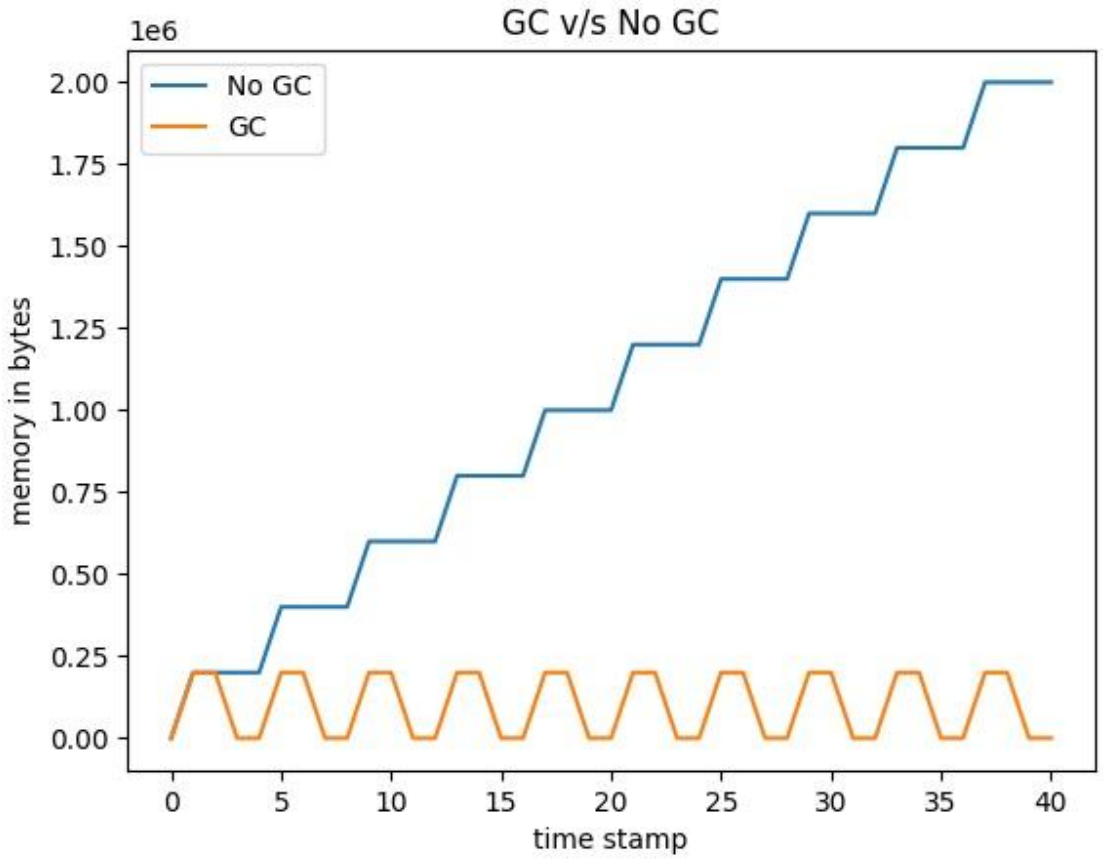
    The function takes the following input:
    - name - name of the variable which we want to mark.

    The function does the following job:
    - Iterate through the mk array insert the name of the variable and set is_mark = 1 for that variable at the first found free place and return.

Ques 3 - What is the impact of mark and sweep garbage collection for demo1 and demo2. Report the memory footprint with and without Garbage collection. Report the time of Garbage collection to run.

Ans -

Comparison for Demo1



Without Garbage Collection

Max = 2000072 bytes, mean = 1073223.219512195 bytes , std dev = 592247.9063142686 bytes.

Time to run the code - 2.250 sec

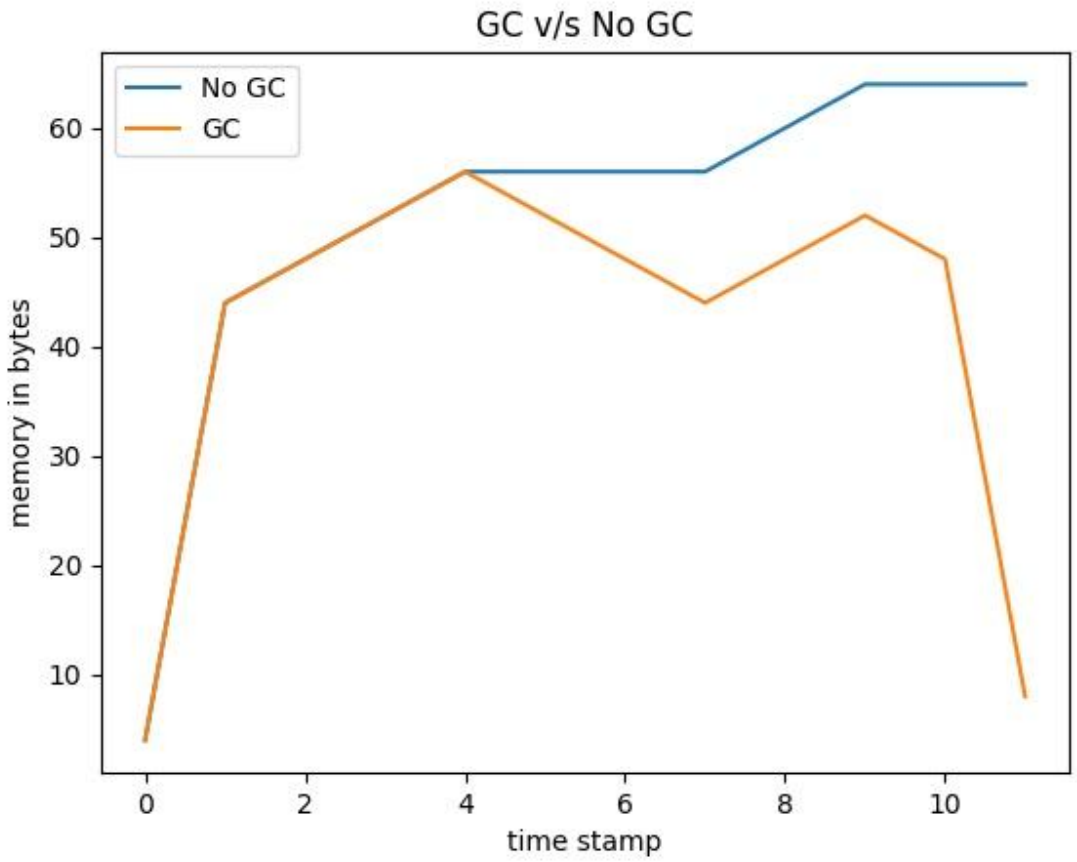
With Garbage Collection

Max = 200036 bytes, mean = 97594.9268292683 bytes, std dev = 99970.29900007992 bytes.

Time to run the code - 2.834 sec

Time to run Garbage collector = 0.584 sec

Comparison for Demo2



Without Garbage Collection

Max = 64, mean = 52.0, std dev = 15.663120165960974

Time to run the code - 0.011 sec

With Garbage Collection

Max = 56, mean = 42.0, std dev = 16.451950239004088

Time to run the code - 0.013 sec

Time to run Garbage collector = 0.002 sec

We can see that without using our garbage collector the memory footprint linearly increases with each time stamp but if we use our garbage collector the memory footprint decreases which shows garbage collection being done and memory being freed.

Ques 4 - What is your logic for running compact in Garbage collection, why?

Ans - Our algorithm for compaction runs in linear time complexity. The whole compaction algorithm goes as follows. Firstly whenever the freeElem function is called with a variable or array name we remove that variable or array from our memory using the sweepPhase function, this will create holes in our page table. Periodically we are calling our garbage collector which will do compaction. In compaction we are trying to fill those holes created, and aggregating all the holes at the end. At first we are traversing the page table and marking the first found hole, after that for every page in the page table we shift the page by hole size so as to fill the hole. If we see a hole in the page table we increase the shift size by that hole size and whenever we see a page we shift the page by shift size. At the end of our iteration we will find all the holes aggregated at the end. We chose this algorithm for our memory management because it does compaction in linear time without using any extra space.

Ques 5 - Did you use locks in your library? Why or why not?

Ans - yes, we have used locks in our library mainly because our garbage collection is done by another thread so to protect the critical data structures of our memory management by simultaneous updates from our main process and garbage collection thread we needed mutex locks