Report

Memory management systems

Group 56

Yashica Patodia - 19CS10067

Ishan Goel - 19CS30052

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

The page table is an array of the type int* in which the index signifies the local address of a variable and the value at that index is a pointer that points to the variable in the physical memory space (250 MB). It is a mapping from logical address of a page to the physical address frame. The size of the array is statically allocated in the beginning.

Justification:

We have use a simple array implementation to store the page because of the following reasons:

- It is easy to implement
- O(1) access of each variable,hence it is fast and more effective

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

The following data structures were used in our library

S.n	Name	Data Type	Use	Justification
1.	int* physical_memory	int*	This points to the starting of the allocated physical memory space of the variable	• Incrementing the physical memory pointer by 1 increments the logical address by 4. • O(1) access of the physical memory address • Easy to implement and fast access.

2.	Stack globalStack	Stack	It is the main stack which stores the local variables.Its data type is a pointer of a class variable(Stack).Whenever a variable is defined its pointer is added to stack and whenever a new function is called a NULL function is called, While returning from a function are popped from a stack until a null pointer is recieved. This allows us to mark and sweep the varibale that are out of scope.	• This implementation of stack is similar to the stack implementation done by the kernel. • The local variables needs to be pushed into a data structure when it is defined in a function and needs to be popped out when it goes out of scope. Hence Stack is a perfect data structure which works in the principle of FIFO.
3.	Variable var_array[MAX_VAR]	Variable	This is the array in which all the objects of the class variable are stored and can be accessed using its local adress.	• All the information about the variables in the MMU needs to be stored. • Hence a class called Variable is created which encapsulates name,datatype,length and local address.
4.	bool* freespace	bool*	This is a bitmap for each "**block**" of the allocated memory space. 4 bytes= 1 block=1 bool	• O(1) access of freespace. (1) indicates it is free and 0 indicates it is occupied. • Fast access

Additional functions and used are as follows:

S.N	Name	Data Type	Use	Justification
1.	medium_int	Class	It contains a char array of size 3.Bit manipulation is used to find the last 3 bytes of int and store in the medium_int variable.	We need a medium_int class to store 3 bytes of memory as this data type is not-predefined in c/c++ libraries.
2.	Variable	Class	It is a class which stores the basic information name,datatype, length,local address	All the information about the variables in the MMU needs to be stored.Hence a class Variable is defined.
3.	Stack	Class	A class called stack is implemented to store the main stack in memory for book keeping to store the local variables of a function.	A class of Stack is implemented and not library is used because we are unaware abt the memory footprints of the stl library stack.

We have used First Fit Algorithm for finding a valid free segment Its advantage is that it is the fastest search as it searches only the first block i.e. enough to assign a process.
As all the data type have 4 byte size we are allocation blocks of chunk size 4 bytes best fit would not make it more efficient.

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

We implement compact in Garbage Collection as follows ,we iterate throught the bitmap array and once a free space is found ahead of it in the physical memory that variable/array is shifted to that empty location freeing up the address initially occupying.

This happens because sometimes when the files are removed even though we have allocated the blocks in increasing order some holes are created so we will have gaps in the memory space. In the compacted memory space all the blocks have been moved to lower indices and there are no interleaved gaps in the memory.

Once the memory is compacted all the book keeping structures (page table) are updated accordingly. Their update happens in constant O(1) time.Hence the total time complexity is is Linear O(n).

This function is called in the following scenarios

- Periodically : In the scenario where the user may freeElem inside a function.
- Before return of a function : Just before a function returns a value the space occupied by the variables are freed because they are no longer used.
- When Memory allocation fails: This may happen when even though space is there for the allocation of a variable we are unable to allocate a variable because it is not contiguous.

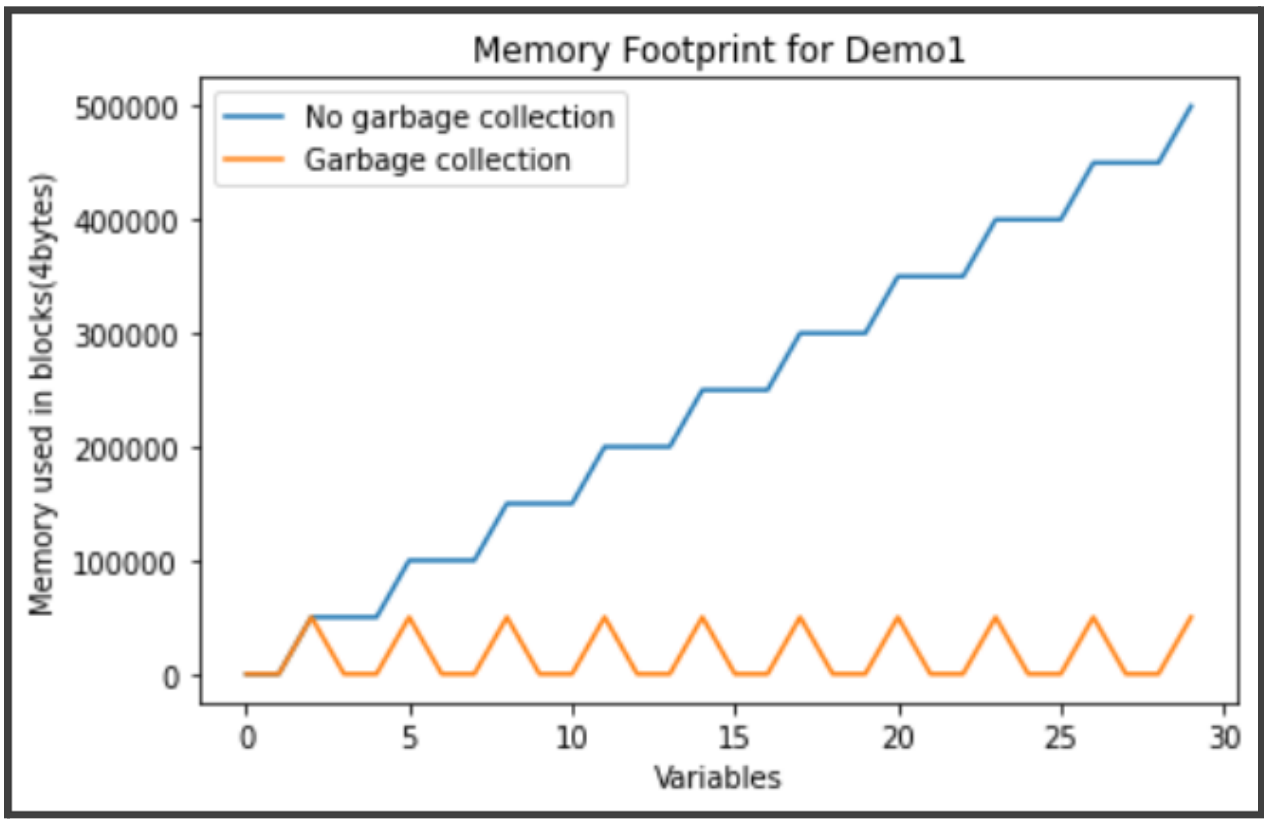
We need compaction to avoid external fragmentation.This happens when total memory space is enough to satisfy a request or to reside a process in it, but it is not contiguous, so it cannot be used. This step is important during garbage collection because during garbage collection(using mark-sweep) we are freeing the memory which are no longer used and are out of scope.Hence we need to perform a compaction to reclaim space lost due to external fragmentation.

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.

All the objects which are created dynamically are allocated memory in the heap. If we go on creating objects we might get Out Of Memory error since it is not possible to allocate heap memory to objects. So we need to clear heap memory by releasing memory for all those objects which are no longer referenced by the program (or the unreachable objects) so that the space is made available for subsequent new objects. Here garbage collection comes to our rescue, and it automatically releases the heap memory for all the unreferenced objects.

DEMO1

In demo1 where we are allocating 10 arrays one by one,garbage collection helps to free the space and enable better and more efficient memory allocation.



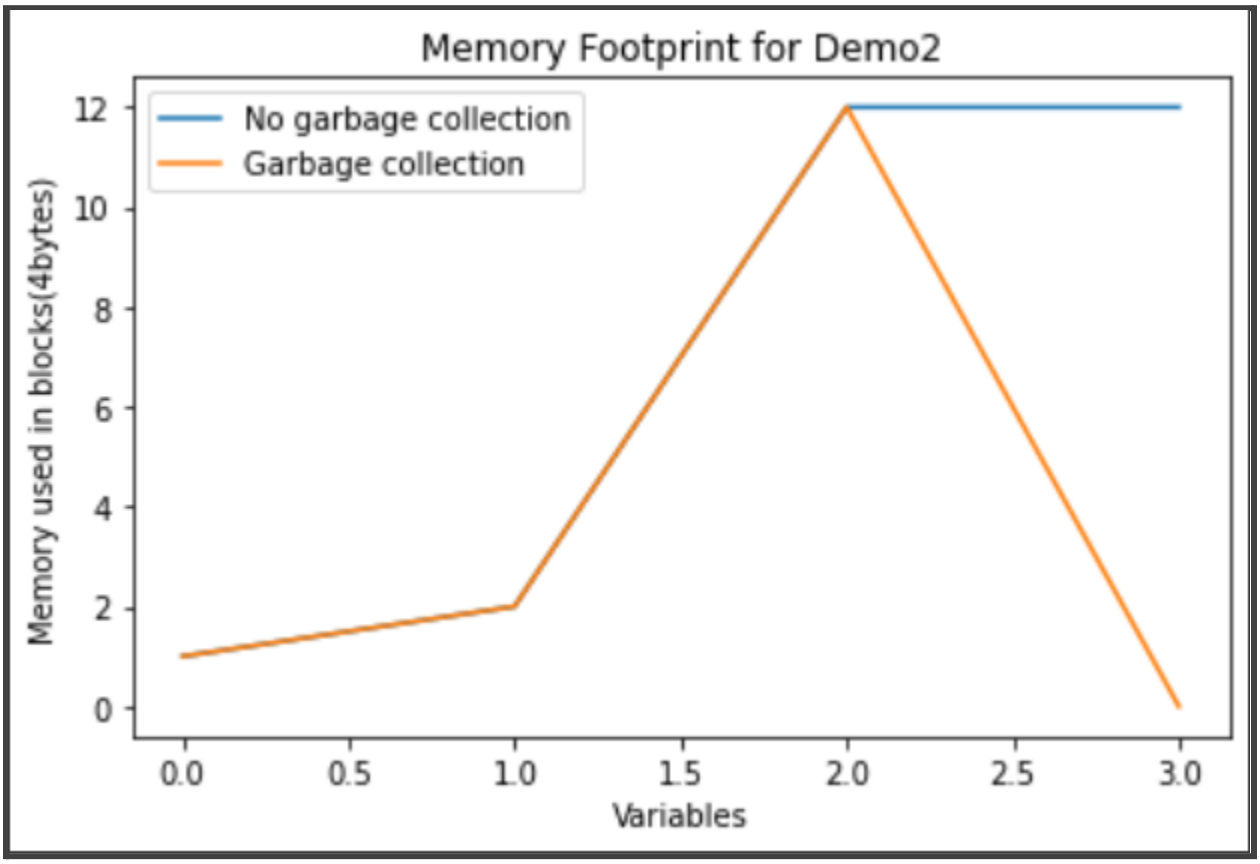
In the above diagram the blue line indicates memory allocation without garbage collection whereas the orange line indicates with garbage collection.

	Maximum	Average	Standard Deviation
Without GC	500020	241677.34	145541.12
With GC	50020	16677.34	23570.46

As we can see with garbage collection the the average memory footprint is **14.5 times** less . Hence garbage collection highly increase the program efficiency in terms of memory storage.

Avg Time for garbage collection for all 10 functions is **191 ms** and for the main function it is **380ms.**

DEMO2



In demo2 we found the product of k fibonacci elements.

In the above diagram the blue line indicates memory allocation without garbage collection whereas the orange line indicates with garbage collection.
As we can see with garbage collection the the average memory footprint is **1.8 times** less .
Hence garbage collection highly increase the program efficiency in terms of memory storage.

	Maximum	Average	Standard Deviation
Without GC	12	6.75	5.2
With GC	12	3.75	4.8

Avg Time for garbage collection forthe function fibonacciProduct is **172** ms and for the main function it is **357**ms.

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

Yes we have used locks in our Memory Management Library.
Justification: The process has two threads. First main thread which adds variables in the stack and allocates space for the variable in the physical memory. Second thread is for garbage collection runs periodically (as well as just before returning from a function) to check the global stack (mark phase) and then go to your memory space to free up the unused memory (sweep phase) after consulting the page table. As there is a separate thread for garbage collection and compaction, since compaction changes the memory address of a variable directly in the physical space , hence accessing before update would give us the wrong memory address (old memory address), which might happen in case of context switches. Hence locks were necessary to be applied, We have used a main memory lock which guarantees that during memory access only one thread can access at a time.

Important Design Decisions

- We have used First Fit Algorithm for finding a valid free segment Its advantage is that it is the fastest search as it searches only the first block i.e. enough to assign a process.As all the data type have 4 byte size we are allocation blocks of chunk size 4 bytes best fit would not make it more efficient.
- Design Decision
    - createMem: Allocating in* pointer of the specified memory size using malloc function.
    - createVar/createArr: A class object Variable and stored in the global array, that variable is also mapped to pagr table with itsphysical address. The pointer to the variable object is pushed to the global stack. All these operations are done in O(1) time.
    - assignVar/assignArr : The physical address of the variable is found through the page table and then the assignment is made for a variable in O(1) time and for an array in O(n) time.
    - freeElem: The garbage collecter calls freeElem on each of the out of scope variables

        - Just before returning from a function
        - If the user specifically declares freeElem
        - The freeElem is an O(1) operation for a variable and for an array it is order O(n) as entry for the page table is deleted and the occupied space is marked as available/free.
    - Word Alighnment: The physical memory is of type int* so every read/write opertion is done in chunks of 4 bytes.

How to Run the code

- To run demo1
    - make demo1
    - ./demo1

  Similary for demo2 and demo3.