Report for Assignment 5

Grp - 16(Vinit Raj, Aryan Agarwal)

Data Structures Involved:

All of these are allocated in the same memory block by pointer manipulation.

- **Memory Block :** This is an array of char that is user allocated and works as the memory for the program.
- **Bitset** for marking the used parts of memory block : This is maintained as one bit per 32 bit word, so it takes 32 times less space than memory allocated.
- **Symbol Table :** This works as the internal Page table, it maps a symbol(identifier of a variable to its logical address in the allocated block memory. It's a simple array of a defined struct that also stores the type and length associated with the variable. This is efficient since it's rare to have many distinct symbols in a program, so the actual size will be much smaller compared to the memory allocated, especially for arrays.
- **Hashmap :** Hashes the string identifier of a variable to its position in the Symbol Table, making access faster.
- **Variable stack :** This is a global stack that keeps track of alive variables(variables in scope). When variables go out of scope it pops them and marks them for deletion(mark phase of garbage collection).
- Only medium int has no native support so it is supported by creating a struct for other types internal implementation is used.

Main Flow:

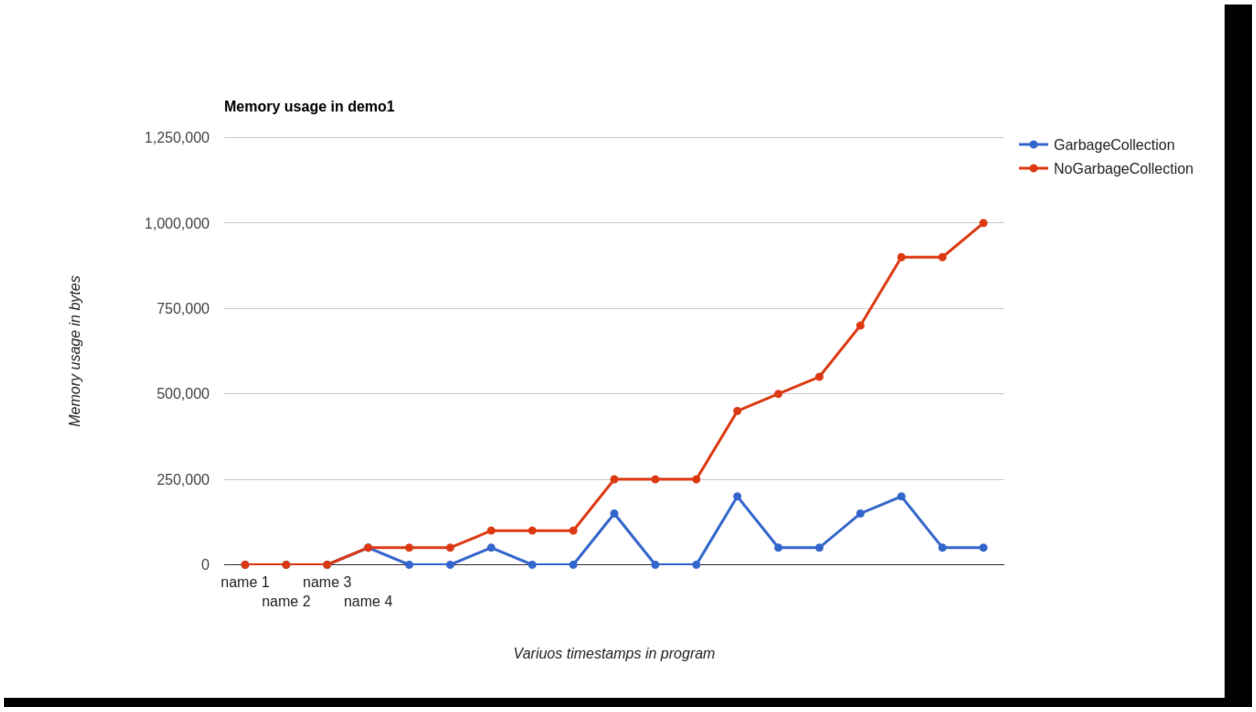
- Variables are identified by their names, this goes to a simple hashing function that maps it to position in the symbol table. This method of access is pretty fast. So during creating a variable its entry is made in the symbol table and then given a position in the physical memory.
- There is a "def" variable created always at the beginning, This works as a placeholder for all arithmetic operations as well as return variable.
- There are mult and add functions for int data type since they are needed in demo2

Pipeline for variable access:

The identifier of the variable is passed to the hashing function which gives the position of its entry in the symbol table. From this struct we get the actual position of this variable in the memory block and then access is performed.

Garbage Collector:

- The mark and sweep algorithm dramatically reduces the memory footprint of the whole program.
- Performance of Garbage collector in demo1 :-



- The extra time overhead for the garbage collection(with compact) in demo1 is around 1.5 seconds.

Compact:

- The compact algorithm uses the two pointer method, one for current empty memory and one for current occupied memory and then copies latter to the former and then advances. This way the holes in the memory are filled and it becomes contiguous.

- The compact is done on two occasion, when we are returning from a function (because some variables just got deleted so holes might be there) and also when array allocation fails(this can happen due to external fragmentation and total empty space might be enough for the array).

Locks:

Yes, We have used locks since there are two threads(main and garbage collector) that are accessing the shared data structures(the bitset used for marking physical memory), Hence to prevent data corruption locks are used.

Demo 3 and Special Consideration

- The extra data structures have some fixed overhead and when allocating small amounts of memory a lot is wasted in the overhead(Evident in demo2) for small values of K.
- Single variables are given the full word length even if it leads to wastage(1 bytes wasted in m_int and 3 in bool/char). But Arrays are much more efficient as they are contiguously allocated minimizing wastage, hence the user is advised to use arrays if multiple variables of the same type are needed.
- Results of calculations (Add and Mult) for integers needed in demo2 are stored in a "def" variable stored at beginning of memory and is never deallocated hence facilitating return statements (acting as return register).