OPERATING SYSTEM LAB

Assignment: 6

Group 5

Design and Data Structures Report

Group Members:

Abhibhu Prakash 20CS10002

Anirban Haldar 20CS10008

Shah Dhruv Rajendrabhai 20CS10088

Suhas A M 20CS10066

Vivek Jaiswal 20CS10077

Data Structures:

```
struct node{
   int value;
   node* prev,*next;
};
```

→ This is the node which stores the integer values of list including two pointers pointing to next and previous nodes of the list.

```
struct symbol {
   char name[56];
   node* start;
   symbol* prev,*next;
   int scope;
   int size;
};
```

→ This is for every entry in the Symbol table.

name: List name of maximum 55 characters.

start: Pointing to the list in the memory.

prev: Pointing to previous entry of symbol table.

next: Pointing to next entry of symbol table.

scope: Specifies the scope of the list.

size: Number of nodes in the list.

```
struct symbolTable{
   symbol *head;
   symbol *tail;
   void init();
   void removeScope();
   void add(char*, symbol*);
   symbol* remove(char*);
   symbol* get(char*);
};
```

→Data structure of symbol table.

head: Pointing to the first symbol of the symbol table.

tail: Pointing to the last symbol of the symbol table.

init(): Function to initialize pointers to NULL.

removeScope(): Function removes every list entry in the present scope.

add(char, symbol*)*: Function adds a symbol in the Symbol table.

remove(char)*: Function removes the symbol whose name is given as argument and returns the removed symbol.

get(char)*: Function fetches the symbol of a list from the symbol table.

```
struct free_list{
   // List of node
   node* head;
   node* tail;
   int size;
   void init();
   void insert(node*);
   node* remove();
};
```

→Data Structure for having list of all free nodes in the memory

head: pointer to the first free/unassigned node in the memory

tail: pointer to the last free/unassigned node in the memory

size: current number of free nodes in memory.

Init(): initialize the list.

Insert(): Inserting a free node, that is, when some memory is freed, they can be added in the free list.

remove(): removing a free node, and returning the pointer to it.

DESIGN

The total memory is divided into several segments.

- Firstly, memory for global free list (free_list data structure) is assigned.
- Then, 0.7 times the total size of memory is assigned for the nodes (i.e, the integer and two pointers constitute a node).
- Further memory is assigned to global symbol table and free symbol table data structures.
- And lastly memory for the symbols.

createMem(int size) function does these assignments as mentioned above.

Here, the global free list has the list of all free nodes. Global symbol table gives us the assigned entries (symbols in the assigned memory for symbols) of all the lists currently used by the user. Free symbol table gives us all the free symbols available in the memory assigned to symbols.

createList(int size, char* name) function removes required number of nodes from global free list and a symbol from free symbol table and adds the symbol to the global free list after updating the required fields in the nodes and the symbol.

assignVal(char* name, int offset, int val) function first searches for the list name in the global symbol table and then goes to the required node in that list as given by the offset and updates it and returns the updated integer.

freeElem(char* name) function first searches for the symbol in the global symbol table, removes from it, then adds all the nodes of the list to the global free list (basically clearing the list).

getVal(char* name, int offset, int scope) function first searches for the list in the global symbol table by the given name and scope and then traverse through the list to get to the required node by the offset given, and returns the integer value.

Other functions include init_scope() (initialize scope to 0), get_scope() (returns current scope value), start_scope() (increments scope) and end_scope() (removes all the symbols and lists belonging to the present scope and then decrements the scope).

No locks are used as no threads are used in the management of the memory and hence race conditions do not occur.

With FreeElem():

Memory footprint: Total memory usage in a single run averaged over 100 runs = 1123608 bytes

Total run time averaged over 100 runs = 112.081 seconds.

Without FreeElem():

Total memory usage = 20027136 bytes

Total run time = 997.145s

The searching of an element takes O(n) time as implementation is done by linked list. And hence the performance would be better if the length/ size of lists created are small as reducing the size means, going towards achieving O(1) search time. And hence larger lists infer lesser performance.