Memory Management System Documentation

BT21CSE015 Prajwal Sam

January 23, 2024

Overview

The provided C++ code implements a simple memory management system, allowing dynamic allocation and deallocation of memory blocks. The memory manager maintains information about used and free memory blocks, facilitating efficient allocation and deallocation operations. Additionally, the system supports memory compaction to optimize the use of available memory.

Assumptions

1. The memory manager is designed for educational purposes and may not be suitable for production environments.
2. The maximum memory size is predefined using the `MEMORY_SIZE_MB` macro (default is 64 MB).
3. The system assumes non-negative sizes for memory allocations.
4. The input file format is assumed to follow a specific pattern (see Sample Input section).

Memory Block Structure

The `MemoryBlock` struct contains information about a memory block, including the variable name, start address, size, and reference count.

```
struct MemoryBlock
{
    std::string var_name;
    int start_address;
    int size;
    int reference_count;
};
```

Memory Manager Structure

The `MemoryManager` struct manages both used and free memory blocks.

```
struct MemoryManager
{
    std::vector<MemoryBlock> used_blocks;
    std::vector<MemoryBlock> free_blocks;
};
```

Functions

1. `create_memory_manager()`

This function initializes a memory manager with one free block representing the entire available memory.

```
MemoryManager create_memory_manager();
```

2. `allocate_memory(MemoryManager &manager, int size, std::string name)`

Allocates a memory block of the specified size and associates it with the given variable name.

```
MemoryBlock *allocate_memory(MemoryManager &manager, int size, std::string name);
```

3. `deallocate_memory(MemoryManager &manager, std::string name)`

Deallocates the memory block associated with the given variable name.

```
void deallocate_memory(MemoryManager &manager, std::string name);
```

4. `print_memory_status(const MemoryManager &manager)`

Prints the current status of used and free memory blocks.

```
void print_memory_status(const MemoryManager &manager);
```

5. `compact_memory(MemoryManager &manager)`

Compacts the memory by sorting used blocks and updating their start addresses, then recreates free blocks.

```
void compact_memory(MemoryManager &manager);
```

6. `assign_memory(MemoryManager &manager, std::string a, std::string b)`

Assigns memory for variable 'a' as a pointer to the memory block associated with variable 'b'.

```
void assign_memory(MemoryManager &manager, std::string a, std::string b);
```

7. `main()`

The main function reads input commands from a file and performs memory allocation, deallocation, and assignment operations.

Sample Input and Output

Sample Input (`input.txt`)

```
var1 allocate 100
var2 = var1
var3 allocate 50
var4 allocate 30
var5 = var4
var3 free
var2 = var5
```

Sample Output

```
... (Output truncated for brevity)
```