Memory System Data Structures Overview

Muhammad Arsalan Hussain

November 19, 2023

Introduction

This document provides a high-level overview of the data structures used in the ‘memorysystem.c‘ program. The program simulates memory allocation and deallocation in a stack and heap environment.

Data Structures

Allocation Structure

`allocation` structure represents an allocated memory block and is defined as follows:

- `name`: An 8-character identifier for the allocation.
- `frameaddress`: The starting address of the allocated frame.
- `next`: Pointer to the next allocation in the linked list.

Frame Status Structure

`framestatus` structure maintains the status of each frame in the stack:

- `number`: Frame number.
- `name`: A function name representing the frame.
- `functionaddress`: Address of the function in the code section.
- `frameaddress`: Starting address of the frame in the stack.
- `used`: A boolean value indicating whether the frame status entry is in use.

Free List Structure

`freelist` structure represents a free region in the memory and is implemented as a linked list:

- `start`: Start address of the free region.
- `size`: Size of the free region.
- `next`: Pointer to the next free region.

Memory System Operations Overview

The program provides functions for creating frames, deleting functions, and allocating/deallocating various data types, including integers, doubles, characters, and buffers. The memory system is simulated with a stack and a heap.

Memory Management Functions

- `create_frame`: Creates a new frame in the stack.
- `delete_function`: Deletes the topmost frame from the stack.
- `expand_frame`: Expands the size of a frame in the stack.
- `create_integer`, `create_double`, `create_character`: Allocates space for integer, double, or character variables in the stack, and copies data to the allocated memory.
- `create_buffer`: Allocates a buffer in the heap, updates the linked list of allocations and the free list accordingly.
- `delete_buffer`: Deallocates a buffer in the heap, updates the linked list of allocations and the free list accordingly.
- `show_memory`: Displays the current state of the memory, including frame statuses, allocations, and the free list.

Memory Simulation

The program simulates stack memory starting at index 394 and heap memory starting at index 0. Frames are managed in the stack, and dynamic memory allocations are managed in the heap.

Linked List Operations

The `freelist` and `allocation` structures are implemented as linked lists. When a new buffer is created ($create_bufferfunction$), the program searches for a free node in the freelist with enough space to accommodate the requested buffer size. If a suitable free node is found, it is used for the allocation. The size and start address of the free node are updated accordingly to reflect the allocated space. If a free node has exactly the required size, it may be entirely consumed by the allocation, in which case it is removed from the linked list. If a free node has more space than needed, a new free node is created with the remaining space, and it is inserted back into the linked list.

When a buffer is allocated ($create_bufferfunction$), a new node is added to the allocation linked list. This node contains information about the allocated buffer, such as its name and starting address in the heap. When a buffer is deallocated ($delete_bufferfunction$), the corresponding node is removed from the linked list. The deallocated memory block in the heap is then added back to the freelist as a new free node, keeping the list updated with available free space.

Memory Traversal and Copying

Memory is traversed and copied in the `create_integer`, `create_double`, and`create_character`$functions, wherethedatai$

Conclusion

The ‘memorysystem.c‘ program implements a simple memory system with stack and heap management. The data structures used provide a foundation for simulating memory allocation and deallocation operations.