READ ME

OS ASSIGNMENT 3

SHAMIK SINHA (2022468)

VANSH YADAV (2022559)

OVERVIEW

The code represents the implementation of a custom memory management system (MeMS) in the C programming language. The MeMS system utilizes the system calls “mmap” and “munmap” for memory allocation and deallocation, respectively, in compliance with specific constraints and requirements. The code defines two primary data structures, namely “MainChainNode” and “SubChainNode”, which are used to manage the free list and track allocated and deallocated memory segments.

FUNCTIONS AND APPROACH

1. **void mems_init():** Initializes all the required parameters for the MeMS system. The main parameters to be initialized are the head of the free list, called “mainHead”. The starting virtual address, 1000, is also initialized as per the given example test cases.

2. **void mems_finish():** This function deallocates the memory, which was allotted to make the freelist data structure by using the “munmap” system call to unmap the allocated memory. It works by iterating over the main chain of the freelist, for every node of the main chain, it iterates over the nodes of the corresponding sub chain. It thus deallocates memory allocated to every node one by one. After all the nodes of a sub chain have been deallocated, it finally deallocates the memory of the corresponding main chain and continues the process for the next main chain.

3. **void* mems_malloc(size_t size):** This function is responsible for allocating memory of the specified size within the MeMS system. The function first attempts to find available memory segments within the existing sub-chains, combining adjacent HOLE segments if

possible using the combine function at the beginning of the code to do so. If no suitable memory is found, the function requests additional memory from the OS using the `mmap` system call and updates the free list accordingly. The function returns the MeMS virtual address corresponding to the allocated memory. Key features of this function include dynamic memory allocation, sub-chain management, and virtual-to-physical address mapping. The `combine` function is used within `mems_malloc` to merge adjacent HOLE segments and reduce memory fragmentation within the free list.

4. **void mems_free(void* ptr):** This function is used to free the memory pointed to by the provided MeMS virtual address (`v_ptr`). It marks the corresponding sub-chain node in the free list as a HOLE. The function also includes logic to combine adjacent HOLE segments to prevent memory fragmentation.

5. **void mems_print_stats():** This function provides an overview of the current statistics for the MeMS system. It prints information about the main and sub-chains, including the memory segments marked as HOLE or PROCESS. Additionally, it displays the total number of pages used and the amount of unused space in the system. The function also outputs the lengths of the main and sub-chains, offering insights into the current state of memory usage within the MeMS system.

6. **void *mems_get(void*v_ptr):**This function retrieves the MeMS physical address mapped to the provided MeMS virtual address. It traverses the main chain and sub-chain to locate the corresponding physical address based on the virtual address input. If a matching virtual address is found, the function calculates the offset and returns the corresponding physical address. If the virtual address is not found, the function returns NULL. This function is crucial for mapping MeMS virtual addresses to their corresponding physical addresses, allowing seamless memory access within the MeMS system.

HELPER FUNCTIONS AND APPROACH

Struct `SubChainNode`

- `size`: Size of the memory segment

- `isHole`: Boolean indicating if the segment is a hole

- `next`: Pointer to the next sub-chain node

- `prev`: Pointer to the previous sub-chain node

- `lowerBoundAddress`: Lower bound address of the memory segment

- `upperBoundAddress`: Upper bound address of the memory segment

- `ending_virtual`: Ending virtual address of the memory segment

- `starting_virtual`: Starting virtual address of the memory segment

- `data`: Additional data associated with the segment

Struct `MainChainNode`

- `SubHead`: Pointer to the head of the sub-chain

- `memSize`: Total memory size

- `next`: Pointer to the next main chain node

- `prev`: Pointer to the previous main chain node

- `lowerBoundAddress`: Lower bound address of the memory segment

- `upperBoundAddress`: Upper bound address of the memory segment

- `data`: Additional data associated with the segment

- `ending_virtual`: Ending virtual address of the memory segment

- `starting_virtual`: Starting virtual address of the memory segment

1. **combine():** This function merges adjacent HOLE segments within the sub-chain to minimize memory fragmentation and optimize memory usage.

2. **lengthOfMainList** and **lengthOfSubList:** These functions compute the lengths of the main and sub-chains, aiding the management and manipulation of memory segments.