<u>Explanation of Each Function Used</u>:

Github link: https://github.com/sameer-singh-godara/Assignment-3-OS
There are 2 linked lists with 2 types of struct (MainChainNode and SubChainNode) which will contain particular attributes.
There are 4 Global Variables, of which 3 are used for maintaining virtual addresses of process/hole nodes and 1 for maintaining the free list.
Virtual Address is starting from 1000.

1) **mems_init()**: It is a function which is initialising the head of the free list. All the attributes of the head will be pointing NULL (or zero), i.e. it does not contain any data, hence pointing to another node that will contain data. We cannot use any other function until this function is called. It will also give an error when the mmap system call is failed.
   The head of the free list is stored in `free_list_head,` which is a global variable that will contain the address of the Node of MainChainNode type; hence in every function to access the free list we will be using this variable.

2) **mems_malloc()**: It is a function that allocates a process of a particular size in the free list (in its pages). It can also generate more pages when required. Pages and Processes in pages are allocated memory by using the mmap system call.

   While generating the pages; they are allocated memory as a node of MainChainNode. And its 1 attributes contains the head of the Node of type SubChainNode in which processes are allocated.

   In this function while allocating the process, it will check 3 conditions:
   a) If mems_offset is less than equal to the virtual address of the current node while iterating. This is because if mems_offset is more than the virtual address of the current node then this means the process has enough space to fit in but it will change the virtual address of that node. But it is not possible, else the virtual address of the current node will change. If mems_offset is less than or equal to the virtual address of the current node then new node will be created using the below 2 conditions where the virtual address of the new node will be the mems_offset.
   b) What is the type of the node, whether it is hole node or process node.
   c) Whether the size of the hole is greater than or equal to the size demanded for allocating the process. In case where size of the hole is equal to the size demanded for allocating the process then it will just change the type of the node else it will split the hole. Splitting of hole is done by creating a new hole of size with the difference of size of the previous hole and the size demanded, after that, we change the size of the previous hole (size variable) to the size demanded and change its type to the process node. The data is set accordingly. A new memory pointer is mapped by mmap system call of size which was demanded. And added to the attribute of the process node. And mems_offset is increased by the size demanded.

   If while iterating the hole is not found then new page is created. And process is allocated on that page. And at last the virtual address of the node where process is allocated is returned.

   3). **mems_print_stats()**: This function is used to print stats related to the MeMS system. It first checks if any memory has been allocated using the MeMS system by checking the by flag. If no memory has been allocated, it prints a message indicating that memory needs to be allocated first.Otherwise, it calculates and prints the following statistics:

i). Pages used: The total number of pages used in the MeMS system, considering both allocated pages and holes.
ii). Space unused: The total amount of unused space (holes) in the MeMS system.
iii). Main Chain Length: The length of the main chain, which represents the number of main chain nodes in the MeMS system.
iv). Sub-chain Length Array: An array containing the lengths of each sub-chain in the main chain. The array elements correspond to the number of pages used in each sub-chain.

This function starts by initializing variables to store the statistics. It then iterates over the main chain, counting the main chain length and calculating the number of pages used and unused space in each sub-chain. The printMainChain() function is called to print the details of each sub-chain node according to example_output.

4). **mems_get(void* v_ptr)**: This function takes a virtual address v_ptr as input and returns the corresponding memory pointer. The function traverses the MeMS system by iterating over the main chain and sub-chain nodes, searching for a sub-chain node that contains the specified virtual address function follows these steps:

i). Start from the free_list_head, which represents the beginning of the main chain.
ii). Iterate over the main chain nodes until a match is found or the end of the main chain is reached.
iii). Within each main chain node, iterate over the sub-chain nodes until a match is found or the end of the sub-chain is reached.
iv).For each sub-chain node, check if the virtual address of the node and the next node in the sub-chain encompass the target virtual address v_ptr.
a). If a match is found, calculate the offset diff between the target virtual address and the start of the sub-chain node.
b). Add the offset diff to the memory pointer stored in the sub-chain node (sub_chain_node->mem_ptr) and return the resulting memory pointer.
c). If no match is found, continue iterating over the remaining sub-chain nodes.

v). If the end of the sub-chain is reached without finding a match, move to the next main chain node and repeat the process.
vi). If the end of the main chain is reached without finding a match, it means that the specified virtual address does not exist in the MeMS system. In this case, the function returns NULL to indicate the address was not found.

5). **mems_free function():** The purpose of this function is to deallocate a memory block associated with a specific virtual address in the MeMS system. The mems_free function takes a virtual address v_ptr as input and performs the following steps:

i). Start from the free_list_head, which represents the beginning of the main chain.

ii). Iterate over the main chain nodes until a match is found or the end of the main chain is reached.
iii). Within each main chain node, iterate over the sub-chain nodes until a match is found or the end of the sub-chain is reached.
iv). Check if the virtual address of the current sub-chain node matches the target virtual address v_ptr and if the next node in the sub-chain is a hole node (type 0).
v). If both conditions are met, it merges the current sub-chain node with the next hole node:
a). Retrieve references to the left node (leftNode), the hole node (holeNode), and calculate the combined size of the two nodes (sizeNode).
b). Update the virtual address of the hole node to match the target virtual address v_ptr.
c). Update the size of the hole node to reflect the combined size of the current node and the hole node.
d). Update the right pointer of the left node to point to the hole node.
e). Use the munmap function to release the memory associated with the current sub-chain node using the size parameter.
vi). If only the first condition is met (no adjacent hole node), it removes the current sub-chain node:
a). Retrieve a reference to the left node (leftNode).
b). Update the right pointer of the left node to skip over the current sub-chain node.
c). Use the munmap function to release the memory associated with the current sub-chain node using the size parameter.
d). Return from the function after successful deallocation.
vii). If the end of the sub-chain is reached without finding a match, move to the next main chain node and repeat the process.
viii). If the end of the main chain is reached without finding a match, it means that there is no node with the specified virtual address in the MeMS system. In this case, the function prints a message indicating that no node with such an address was found.

6). **void mems_finish()**: This function in the MeMS (Memory Management System) library. The purpose of this function is to finalize and clean up the MeMS system, releasing all allocated memory and associated data structures.
i). Initialize a pointer to the current main chain node as current_main and set a flag finish_used to indicate that the mems_finish function has been called.
ii). Iterate through the main chain nodes:
a). For each main chain node, iterate through its sub-chain nodes.
b). Check if the sub-chain node is of type 1 (indicating a PROCESS segment).
c). If it is a PROCESS segment, unmap the allocated memory associated with it using the munmap function.
d). Unmap the sub-chain node itself after releasing its memory.
iii). Unmap the MeMS heap start, which is presumably the starting point of the MeMS memory management.
iv). Unmap the free_list_head to release its associated memory.

HOW TO RUN by make file:
1). Open file then right click open in terminal.
2). Then write make press enter

3). Write ./example enter.

Via vs code:

Directly run it.