1. Class Process - This class represents an individual process. All the variables are private, and thus, have their appropriate getter/setter functions.

1.1 Member Variables of Process Class

PID, with type *unsigned int*, represents the Process ID for each process.

Addr_physical, with type int, represents the start address in the memory array for the process.

isProcessCreated, with type *boolean*, indicates whether this process has been created or not. This property is helpful for solving hash collisions using the double hashing method.

1.2 Member Functions of Process Class

Since all member variables are private, their values can be accessed through public getter/setter functions.

get_PID, get_addr_physical, get_isProcessCreated return the PID, Addr_physical, and isProcessCreated respectively.

set_PID, set_addr_physical, set_isProcessCreated provide the values for PID, Addr_physical, and isProcessCreated respectively.

Additionally, there is no destructor for the Process class since we never dynamically allocate memory. In the constructor, we initialize isProcessCreated to be false, PID as 0 and Addr_physical as 0.

2. Class OpenAddressedHT - This class represents a hash table, where hash collisions are resolved using Open Addressing technique, or more specifically, double hashing.

1.1 Member Variables of OpenAddressedHT Class

Memory is a pointer to an integer array with size N i.e. memory size inputted by the user. It is dynamically allocated, with all elements initialized to 0.

Process is an array of Process objects, initialized to default values as set by the constructor. The array is dynamic, with a size of N/P i.e memory size/ page size. It represents the hash table with the key being Process IDs.

HT_Size represents the size of the Process array. P represents the page_size inputted by the user, and N represents the memory_size inputted by the user.

Pages_used is a pointer to an integer array with the size of the number of processes that can be stored in memory. It tracks whether a page has been allocated to a certain part of the memory, so other processes don't use it.

Current_pages_used is an integer, representing the number of existing processes.

1.2 Member Functions of OpenAddressedHT Class

The constructor of the class takes in two integer inputs - memory_size and page_size. It initializes all the member variables and prints out success.

The destructor of the class deletes the three dynamically allocated member variables i.e. memory, pages_used, process and sets them to nullptr.

Insert_PID is called with the INSERT command, and takes in an id of type unsigned int. First, it checks to see if the hash table is full and if the PID exists already. If both cases are false, using both the given hash functions, it calculates the appropriate index within the hash table to add the key. The start address and PID is set for the Process object. The value of the current pages used is incremented. Assuming allocation of pages is ignored and there is uniform hashing, the insertion is in constant time. Note, since we assume uniform hashing, I ignore the time complexity from checking to see if the PID already exists or not.

Search_PID is called with the SEARCH command, and takes in an id of type unsigned int. Since, in open addressing, when multiple keys map to the same value, they search for the next slot, there is a loop to go through each possible index it could be in. In each index, there is a check to see if the ID we are searching for is the same. The loop iterates m times i.e. the size of the hash table. If no PID is found, it exits the loop and prints failure. Assuming uniform hashing, the search is in constant time.

Write_PID is called with the WRITE command, and takes in an id of type unsigned int, virtual address of type int and the value to be inserted. First, there is a check to see if the virtual address is less than the allocated page size for a process. When the PID is found, the memory address is calculated and the value is set. Assuming uniform hashing, the write is in constant time.

Read_PID is called with the READ command, and takes in an id of type unsigned int and virtual address of type int. First, there is a check to see if the virtual address is less than the allocated page size for a process. When the PID is found, the memory address is calculated and the value is read. Assuming uniform hashing, the write is in constant time.

Delete_PID is called with the DELETE command, and takes in an id of type unsigned int. Similar to previous functions, there is a loop to find the PID in the hashtable, determined by the two hash functions. When the PID is found, all the values are set to their initial values. The total number of available pages is decremented. Assuming uniform hashing, the write is in constant time.

3. Class SeparateChainingHT - This class represents a hash table, where hash collisions are resolved using Separate Chaining technique.

1.1 Member Variables of SeparateChainingHT Class

Similar to OpenAddressedHT Class, SeparateChainingHT also has the following member variables with the exact same purpose and implementation: Memory, HT_Size, N, P, Pages_used, Current_pages_used. The major difference is Process, which is a dynamic array that stores a vector of pointers to Process objects.

1.2 Member Functions of OpenAddressedHT Class

The constructor of the class takes in two integer inputs - memory_size and page_size. It initializes all the member variables and prints out success. The destructor of the class iterates through the process array, freeing each pointer to the process object and also, deallocates memory for the other two dynamically allocated member variables i.e. memory, and pages_used.

Insert_PID is called with the INSERT command, and takes in an id of type unsigned int. First, it checks to see if the hash table is full and if the PID exists already. As it iterates to see if the PID exists in the vector, it also looks for the appropriate index to add the element such that it will be in descending order. After setting the value for the Process object, like its PID and start address, it's appended in the appropriate place of the vector.

Search_PID is called with the SEARCH command, and takes in an id of type unsigned int. It calculates the index from the hash function, goes to the vector and loops through it until it finds the right PID. This makes the runtime O(n) where n is the length of the vector.

Write_PID is called with the WRITE command, and takes in an id of type unsigned int, virtual address of type int and the value to be inserted. It checks to see if the virtual address is within the range of the page size. It calculates the index from the hash function, goes to the vector and loops through it until it finds the right PID. It calculates the memory address and writes the value. Read_PID is similar, except it reads the memory address. Both functions have a runtime O(n) where n is the length of the vector.

Delete_PID is called with the DELETE command, and takes in an id of type unsigned int. It finds the index using the hash function, and loops through the vector till it finds the PID it's looking for. It deallocates memory for the array and erases it from the vector. It has a runtime O(n) where n is the length of the vector.

Finally, Print_PID is called with the PRINT command. First, there's a check to see if there is a vector at that index in the hash table and if not, prints out that the chain is empty. If it finds a vector with more than 0 elements, it loops through the vector and prints out the PID as it goes. Since the insert adds the elements in an orderly fashion, there is no sorting here. It has a runtime O(n) where n is the length of the vector.