UNIVERSITY OF WATERLOO
FACULTY OF ENGINEERING
Department of Electrical & Computer Engineering

ECE 250 – Project 2
VMM Design Document
Hossein Molavi : hmolavi

Overview of Classes

Time Complexity

VirtualMachineManager (VMM) Class

std::string insertPID(unsigned int) O(1)*

Called to insert a PID, works for both Chaining and Open-Addressing. Will take constant time for Open- Addressing assuming first hash function works most of the time. Same for Chaining operation, however values are inserted in descending order.

void printPID(unsigned int) O(1)*

Searches for and prints the index to a PID. Works for both Chaining and Open-Addressing.

void printPosition(unsigned int) O(1)*

Exclusively for Chaining, function prints all PID keys at specified position on the Vector List.

void printMem(unsigned int, unsigned int) O(1)*

Called with PID key and an index within its memory page to print.

std::string writeMem(unsigned int , unsigned int , int) O(1)*

Given PID key and page address, function will write an integer value to that memory address

std::string deletePID(unsigned int) O(1)*

Searches through the table for PID given its key, and **ONLY** resets its key value back to zero.

private : unsigned int hash_function1(unsigned int) O(1)

Would return Primary hash function: $h1(k) = k \ mod \ m$ given key

private : unsigned int hash_function2(unsigned int) O(1)

Would return $h2(k) = \lfloor k/m \rfloor \ mod \ m$ given key

private : unsigned int double_hashing_function(unsigned int , unsigned int , unsigned int) O(1)

Would return $h(k, i) = (h1(k) + i * h2(k)) \ mod \ m$, where i is an integer from 0 to m-1.

Class members: RandomAccessMemory* RAM ; unsigned int pageSize ; ProcessID* p_PIDArray ; std::vector<ProcessID>* p_PIDVector ; unsigned int maxPIDCapacity ; unsigned int curPIDCount; int COLLISION_RESOLVE_MODE

RandomAccessMemory Class (RAM) : friend class VirtualMachineManager

Class has no functions.

Class members: int* p_Physical_Mem

ProcessID Class : friend class VirtualMachineManager

Class has no functions.

Class members: int* p_mem_index ; unsigned int key ; bool isDeleted

*Constant time under the assumption only the first hash function is needed most of the time (uniform hashing), completely Ignoring the problem of allocating/deallocating the pages themselves

Constructors and Destructors

The VMM and ProcessID Class both have a default constructor. VMM along with RAM classes have a parameterized constructor, for VMM it will set the m and p values, and for RAM, it will set the m. Important to note that the RAM parameterized constructor will be called from VMM. Similarly, the destructor for RAM is called from VMM's destructor. ProcessID class does not have a destructor.

Why Not multiple VMM classes

I believe there is a fine balance between simplicity and practicality. It would be very possible to implement two VMM classes for Open addressing and chaining which inherit from a parent VMM class. However, due to the small size of the project the classes would be very similar and not easy to understand at first glance. I chose one inclusive VMM class which handles all the details based on the mode it is in, makes for more practical use in comparison to three different VMM classes.

Why Not separate Chaining - findNode

For the sake of the same argument as above, creating one function for two uses is more practical specially when only one will be used per run time.

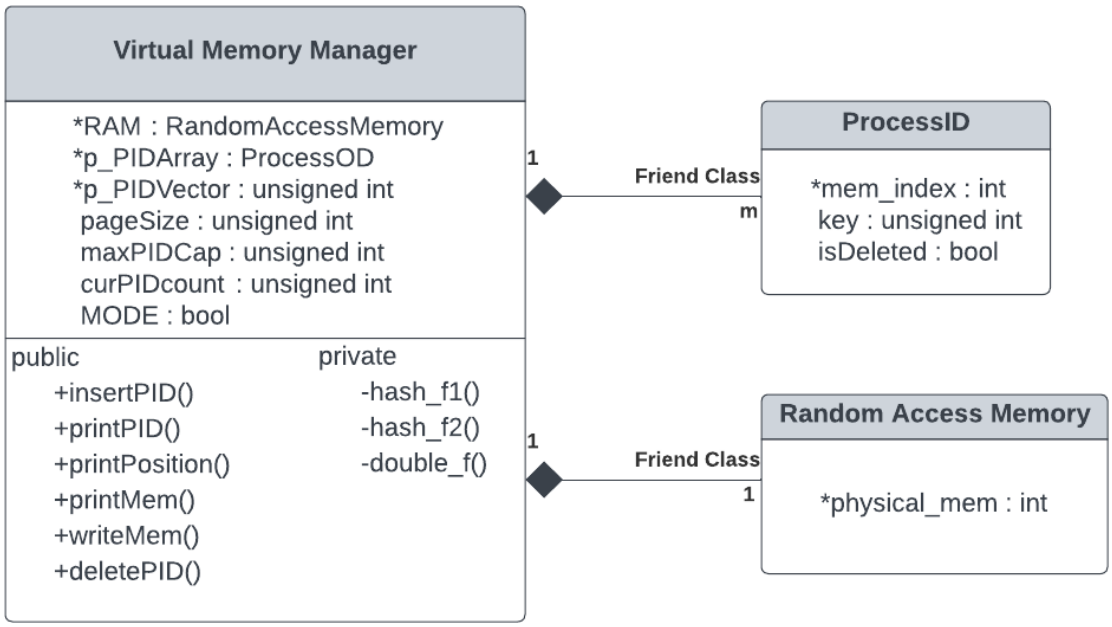
Why capacity AND size - (capacity is m, size is number of current PIDs)

Having a size parameter makes insertPID conditions more efficient, allowing for quick check of full capacity or empty state, specifically at the beginning of insertPID and deletePID.

Why private hash functions in VMM

Organizing the hashfunctions makes for more readable code as they are used in every function. This design also makes for easier changes in the future to the hash functions.

Structure - UML Diagram



Program Sources:

https://ece.uwaterloo.ca/~ece150/Lecture_materials/

*Constant time under the assumption only the first hash function is needed most of the time (uniform hashing), completely Ignoring the problem of allocating/deallocating the pages themselves