Emre Kaan Usta 30728

---------- PA 4 ----------

Psuedocode

Class HeapManager

Private:

Struct to keep track of index, id and size

Mutex to keep atomicity

List of struct that is defined already

Public:

HeapManagementLibrary(){

Mutex initialization}

İnit Heap (int size){

Create a new node with the input size, id of -1 and index of 0

Push the node to the list

Print the current list

Return 1

}

İnt myMalloc(int threadID, int size){

Lock the function

Iterate through the list

İf current Node's size is bigger than input size and id is -1

Take the size and index of the current node

Create a node with the index of current node and inputs of the function

İnsert newly created node to list

İf current Node size is smaller than or equal to 0

Erase current node

Else

Current node index = current index + size

Currnode size = current size + size

Print the list and message

İf no node is found

Print the regarding output and last list

Return -1

}

Int myFree (int ID, int index) {

Acquire lock

İtreate through list

İf there is a node to satisfy the id and index

Get passed node

İf passed node id == -1 meaning free

Update the size with the node that is going to be deleted from the list

Delete currNode

currNode –

}

Now if there is not a node satifies, mark this curr Node as free id == -1

Get the upcoming node

If node exists and free

Update currNode size and delete upcoming node

Print the message saying correctly freed

Unlock the mutex

Return -1}

Void print(){

Lock the mutex

Using auto iterate through the list

İf node exists and not the last node

Print with curly brackets

Cout additional “---“

Else if node is the last node

Print with curly brackets and do not put --- at the end.

Release the lock

}

In this assignment, I used C++ programming language since it is what I am most used to. I have a struct containing id, size and index. Additionally I have a list to keep these struct objects.

Int Heap() is used to initalize the list with a single node of id -1. I pushbacked this node to the list and printed the inputs. And returned -1.

Int myMalloc() is used to iterate through the list and find a free space with the size given as input. I put a lock at the beginning of the function to keep the atomicity since we are updating struct informations. While iterating, if id is -1 -meaning free- and size is avaliable, I filled update the size of this node accordingly. Then printed allocated message and returns the newly created index. If we cannot find a proper node to store the size, functions prints “cannot allocate” message, unlocks the lock and returns -1.

Int myFree() is again used to iterate through the list, update and delete the nodes. I again used a lock to prevent race condition. Function iterates throıgh the list to find a free node and if it is free, it transfers the sizes and removes the current node. Then sets the ID of the current node to -1 which means it is free. Then function checks if the upcoming node is free and valid, so that it updates the sizes again, prints the message and returns 1. If no node is found accordingly, it returns -1 and prints error message.

Void print() this function should be called after every other function to keep track of the current table. It prints the data of the nodes prints them accordingly. Again protected with a lock so that it does not get intervened by another thread.

I chose coarse grained lock system for this assignment since there are lots of updating inside the functions and it is easier to lock the whole body this way. Which allowed me to print the regarding informations safely.