CS307 PA4 REPORT

Firstly I created a struct for the linked list's node which is called allocNode. Includes the ID, index, size as mentioned in the document and also includes another allocNode which will indicate the next allocNode in the linked list.

I created the class with one private member which is the head of the linked list.

In public members there is a constructor for the linked list initializing head to NULL since the list will be initially empty. The four functions of the PA are also public members.

I've used mutex in order to deal with concurrency for this homework. Here are the pseudo codes of four functions:

Print():

Create a node that is equal to head of the list.

Traverse the list and print the desired output.

myMalloc(int ID, int size):

Lock the mutex.

Initialize two nodes (currNode indicating the current node, prevNode indicating the previous node).

Start traversing the list.

Check if the current node is free and it has >= size to allocate.

If so create another node called allocateNode which will be the node to be allocated.

Give it the ID, size and index of the current node.

If the allocateNode's size is bigger than the requested size we will have to divide it into two nodes therefore: create another node, since it will represent the free space set its id to -1, set its size to the remaining size from the allocateNode's total size, set its index and next node. Set this node as the allocateNode's next node. Delete the current node we have (the node that has been parsed into 2 nodes). If the list was containing only 1 element which is the head, set AllocateNode as the first node of the list. Otherwise set the previous node's next node as AllocateNode (We deleted the current node so prevNode's next has to be initialized.).

If it is just enough to allocate: set allocateNode's next node as currentNode's next node and delete currentNode (we are replacing the currentNode). If the currentNode was the head node, set allocateNode as the head node, else set it's previous node as allocateNode's previous node.

If the allocation has been done, print the Allocation successfull line, print the linked list and release the mutex and return the index of allocateNode.

If there wasn’t any node eligible to allocate space; print error, print the linked list, release the mutex and return -1.

myFree(int ID, int index);

Lock the mutex.

Start traversing the list.

Check if the required node is found (node’s id and index have to be the same with given id and index).

If it is found: set its id to -1 indicating that it’s a free node

Check if the previous node was also free. If so:

Set previous node’s size to prevNode size + currNode size.

Set the previous node’s next node as current node’s next node (we are putting two nodes into one single node.)

Delete the current node.

Set the current node as previous node.

Print success statement, print the linked list, release the mutex and return 1.

If the previous node wasn’t free check if the next node is free. If so:

Merge the two nodes together.

Print success message, print the linked list, release the mutex and return 1.

If both the previous node and next node is not free it means that the current node is the only free one. Therefore:

Print success message, print the linked list, release the mutex and return 1. (We don’t have to do anything else since we already set the current node’s id to -1 at the start.)

If there were no such nodes found:

Print error message, print the linked list, release the mutex and return -1.

initHeap(int size):

We don’t need any mutex here since the heap is initialized before any threads are created.

In this function we are initializing a node as the head node, setting its attributes, printing the linked list and returning 1.

URAL SARP SİPAHİ 28093