PA4 REPORT

Buse Gündoğar 27931

In the solution, I firstly created a struct for node called "**myNode**". In this node I have:

- ID, size, index and next (which shows the next node).
- Constructor for the myNode.

Then I defined the class "**HeapManager**", which the name was given in the test cases. This class has the followings in its public part:

- Constructor
- myAlloc
- myFree
- print
- initHeap

In its private part, it has the head node, which keeps the linked list. I also defined the mutex here.

Here is the implementation of the rest of the functions in allocator.cpp as a pseudocode:

Define **HeapManager** constructor (makes the head NULL and initializes the mutex)
Define **initHeap**(int size)
    Mutex lock (even though in the test cases we don't need a mutex for initHeap, I put it there just for it may be needed)
    Creating the node with the given size, ID = -1, index=0 and next=NULL.
    Mutex unlock

Define **myMalloc**(int ID, int size)
    Mutex lock
    Initialize "search" node as the "head" node
    Initialize "prev" node as NULL
    WHILE search != NULL
        IF search's ID != -1 or search's size < size:
            Assign search node to prev
            Assign search->next to search
        ELSE
            Break

    IF search is NULL:
        Print allocating error message
        Print linked list
        Mutex unlock
        Return -1

    ELSE
        IF size == search->size:

Assign search->ID to ID given as parameter
Print allocation message and linked list
Unlock mutex
Return search->index

ELSE

Define a new myNode with the given parameters ID, size, search ->index and “search” as next node.

IF prev is NULL:
Assign newNode to head
ELSE:
Assign newNode to prev->next

Increment search->index with size
Decrement search->size with size
Print allocation message and linked list
Mutex unlock
Return newNode->index

Define **myFree**(int ID, int index)
Mutex lock
Initialize “search” node as the “head” node
Initialize “prev” node as NULL
WHILE search != NULL:
IF search->ID equal to ID and search->index equal to index:
Make search->ID = -1

IF prev != NULL and prev->ID == -1:
Add search->size to prev->size
Make prev’s next equal to search->next
delete “search”
Assign prev to search

IF search->next != NULL and search->next->ID == -1:
Add search->next->size to search->size
Define a new node “newNode” and assign it to search->next
Make search’s next equal to nextNode’s next
Delete nextNode

Print freeing message and linked list
Unlock mutex
Return 1
ELSE
Update prev as search
Update search as search->next

```
        Print freeing error message and print linked list
        Unlock mutex
        Return -1

Define print()
        Create printPtr equal to the head node
        WHILE printPtr is not equal to NULL:
                Print node's ID, size, index
                IF printPtr's next is not NULL:
                        Print "---"
                Update printPtr as printPtr->next
        Print new line
```

For the locking mechanism, I used mutex. I initialized it in the class constructor and used the lock in the critical sections (such as myFree and myAlloc, so that their operations are not interrupted). I did not use it in the print function, since that function is used inside the other functions, and their atomicity were guaranteed with the mutex lock. The mutex is released at the end of these critical sections.

The atomicity is guaranteed by locking the sections in which shared data (such as linked list) were updated. By doing that, race conditions are prevented.