CS307 – PA4

AHMET ALPEREN GÜNGÖR - 28847

The provided C++ code is an implementation of a simple (also not really heap, just a simulation) heap manager with a linked list data structure for managing memory allocations and deallocations by multiple threads. Pthread is used for multi-threading.

Linked list implementation has nodes, where it has id, size and start values as well as next/prev pointers.

In order to ensure atomicity, a single mutex lock is used for member functions. Also, the print function that is requested have a lock, but another print function called printNonLock with the same functionality as print but without lock is used, since member functions are already implemented using lock – another lock inside print function causes problems.

In addition to public methods that are requested in the pdf file, some helper function methods are implemented too (explained in code as comment).

Member Functions (requested in pdf):

initHeap(int size): Initializes the heap with a memory block of the specified size.

myMalloc(int threadId, int size): Allocates memory for a thread with the given thread ID and size.

myFree(int threadId, int start): Frees memory allocated for a thread with the specified thread ID and start index.

print(): Prints the contents of the linked list using a lock.

printNonLocked(): Prints the contents of the linked list without using a lock.

Below ist he pseudocode for each public and private method.

Private Methods:

```
Node findFreeNode(int size):
    Node curr = head
    while curr is not NULL:
        if curr.tid == -1 and curr.size >= size:
            return curr
        curr = curr.next
    return NULL

Node findNode(int threadId, int start):
    Node currentNode = head
    while currentNode is not NULL:
        if currentNode.tid == threadId and currentNode.start == start:
            return currentNode
        currentNode = currentNode.next
    return NULL

void removeNode(Node targetNode):
    if targetNode is head:
        head = head.next
    else:
        Node current = head
        while current.next is not targetNode:
            current = current.next
        current.next = targetNode.next
    delete targetNode

void prevMerge(Node currentNode):
    if currentNode is not head:
        Node previousNode = head
        while previousNode.next is not currentNode:
            previousNode = previousNode.next
        if previousNode.tid == -1:
            previousNode.size += currentNode.size
            removeNode(currentNode)
```

Public Methods:

```
    void initHeap(int size):
        head = new Node(-1, size, 0)
        lock = Mutex()
        printNonLocked()
        unlock(lock)

    int myMalloc(int threadId, int size):
        lock(lock)
        Node curr = findFreeNode(size)
        if curr is not NULL:
            printNonLocked()
            unlock(lock)
            return curr.start
        else:
            printNonLocked()
            unlock(lock)
            return -1

    int myFree(int threadId, int start):
        lock(lock)
        Node curr = findNode(threadId, start)
        if curr is not NULL:
            prevMerge(curr)
            curr.tid = -1
            Node nextNode = curr.next
            if nextNode is not NULL and nextNode.tid == -1:
```

```
            curr.size += nextNode.size
            removeNode(nextNode)
        printNonLocked()
        unlock(lock)
        return 1
    else:
        printNonLocked()
        unlock(lock)
        return -1

void print():
    lock(lock)
    Node current = head
    while current is not NULL:
        print("[", current.tid, "][", current.size, "][", current.start, "]---")
        current = current.next
    print()
    unlock(lock)

void printNonLocked():
    Node current = head
    while current is not NULL:
        print("[", current.tid, "][", current.size, "][", current.start, "]---")
        current = current.next
    print()
```