Duygu Tümer

24842

REPORT PA4

In my PA4, I used 2 mutex. One of them is for shared locks for memory allocation and freed memory, and the other one is for printing since preventing deadlock. Because when I allocate a memory with myMalloc(), first, I need lock my shared mutex since the other threads cannot allocate a memory at the same time. After then I need to call print() function to print the result. However, if I use the same lock in the print(), then first I have to lock since I do not want to the others to come. But, since, I have already locked my lock, this cause a deadlock. That is why 2 used 2 mutex for locking and unlocking to provide an atomicity for my code. And here is my pseudo code:

```
pthread_mutex_t sharedLock = PTHREAD_MUTEX_INITIALIZER;

pthread_mutex_t printLock = PTHREAD_MUTEX_INITIALIZER;

void HeapManager::print()

{

   pthread_mutex_lock(&printLock);

   // some codes in here

   pthread_mutex_unlock(&printLock);

}

int HeapManager::myMalloc(int ID, int size)

{

       pthread_mutex_lock(&sharedLock);

       //some codes

       if(node == NULL)

      {

          cout << "Can not allocate, requested size " << endl;

          pthread_mutex_unlock(&sharedLock);

          return -1;

      }

   cout << "Allocated for thread " << ID << endl;

    // some codes

   print();

   pthread_mutex_unlock(&sharedLock);

   return node->index; //return the start index of the newly allocated node
```

```
}



int HeapManager::myFree(int ID, int index)
{
   pthread_mutex_lock(&sharedLock);
   // some codes
   if(node == NULL || node->id != ID || node->index != index)
   {
      pthread_mutex_unlock(&sharedLock);
      return -1;
   }
   print();
   pthread_mutex_unlock(&sharedLock);
   return 1;
}
```