Mert Kılıçaslan

Heap Management (PA – 4) Report

CS307 – Operating Systems

Pseudo Code of the member functions lock synchronization mechanisms were used:

```
Global pthread_mutex_init lockPrint, lockMalloc, lockFree

void print(){
	mutexLock(lockPrint)
	for(iter = list.begin; iter != list.end; iter++)
		cout << iterID << iterSize << iterIndex << endl
	mutexUnlock(lockPrint)
}

int myMalloc(int ID, int size){
	mutexLock(lockMalloc)

	for(iter = list.begin; iter != list.end; iter++)
		if(iterSize >= size && iterID == -1){
			list.insert newNode(id, size, iterIndex)

			iterSize -= size
			iterIndex += size

			mutexUnlock(lockMalloc)
			return newNodeIndex
		}

	mutexUnlock(lockMalloc)
	return -1
}

int myFree(int ID, int index){
	mutexLock(lockFree)

	for(iter = list.begin; iter != list.end; iter++)
		if(iterID == ID && iterIndex == index)
			iterID = -1

	for(iter = list.begin; iter != list.end; iter++){
		Iterator temp = iter
		if(iterID == -1 && iterID == (++temp)ID){
			iterSize += tempSize
			list.erase temp
			it--
		}
	}
	mutexUnlock(lockFree)
	return nodeFound? 1 : -1
}
```

Description of the Synchronization Mechanisms:

In order to prevent data race and to ensure atomicity of the operations under concurrency, I used mutex lock mechanism in the member functions (print, myMalloc, myFree) of HeapManager class. I did not need to use it in the initHeap member function since the heap is initialized by the main thread only.

Mechanisms are simple, all the mutex locks are set at the beginning of the three functions to lock the function. The locks are released before functions return. In order to prevent deadlock, all the locks must be released before the return statements. For the functions that have more than one return statement (due to if/else conditions), more than one mutex unlocks are used such as myMalloc and myFree.

This locking and unlocking mutex mechanism simply works when one thread starts executing in the critical section, and the other thread waits until the first one finishes and releases the lock. If the synchronization technique is not applied, it causes a race condition where the values of variables are non-deterministic due to the timings of the context switches. As in this assignment, this method should be used when more than one thread is used.