PA4 Assignment Report

1. int initHeap(int size): In initHeap, I initialized the memory with a single free node of the specified size and a starting index of 0. To make this process atomic during concurrent calls, I used pthread_mutex_lock and pthread_mutex_unlock. Before returning, I printed the initial linked list layout.

2. int myMalloc(int ID, int size): In myMalloc, I applied the first-fit approach, allocating memory from the first free node with enough space. If the node had excess space, I divided it into two nodes. Using mutex locks (pthread_mutex_lock and pthread_mutex_unlock), I ensured the atomicity of this operation, preventing interference from other threads. The function returned the starting index of the allocated node or -1 if no suitable space was found.

3. int myFree(int ID, int index): In myFree, after marking the specified node with the given ID and index as free, I performed coalescing to optimize memory usage. This involved merging the freed node with adjacent free nodes, creating larger free blocks for efficient space management. Mutex locks ensured the safety of these operations in a concurrent environment. The function returned 1 on success and -1 if the specified node was not found.

4. void print():In print, I traversed the linked list, printing information for each node. Since it was a read-only operation, I didn't use mutex locks.

Concurrency using Mutex: To handle concurrency, I used mutex locks (pthread_mutex_t) within critical sections for functions like initHeap, myMalloc, and myFree. This ensured atomicity, preventing race conditions and maintaining data consistency. I acquired the lock before entering the critical section in each method and released it afterward, allowing safe use by multiple threads concurrently.