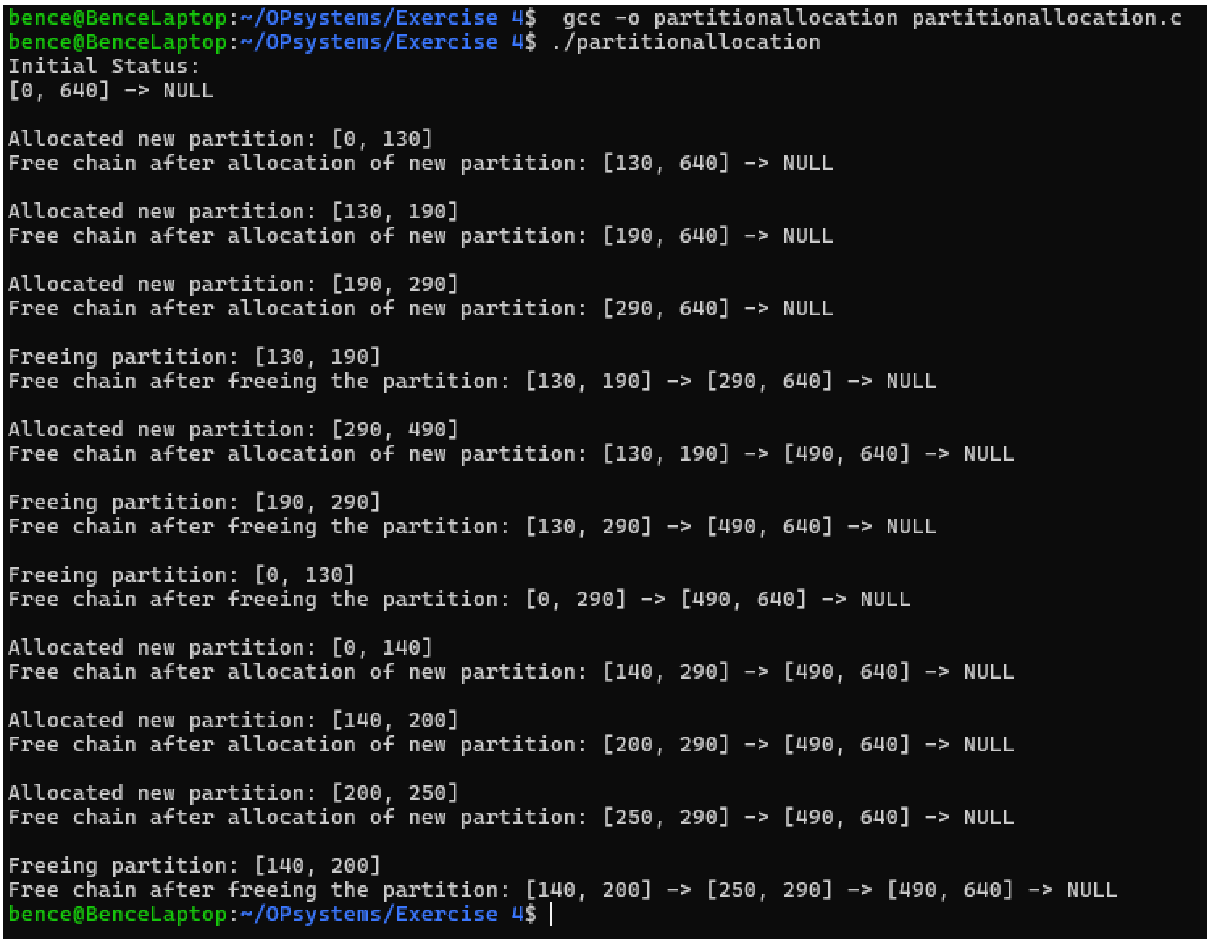
Operating Systems Project - Exercise 4



```
bence@BenceLaptop:~/OPsystems/Exercise 4$  gcc -o partitionallocation partitionallocation.c
bence@BenceLaptop:~/OPsystems/Exercise 4$ ./partitionallocation
Initial Status:
[0, 640] -> NULL

Allocated new partition: [0, 130]
Free chain after allocation of new partition: [130, 640] -> NULL

Allocated new partition: [130, 190]
Free chain after allocation of new partition: [190, 640] -> NULL

Allocated new partition: [190, 290]
Free chain after allocation of new partition: [290, 640] -> NULL

Freeing partition: [130, 190]
Free chain after freeing the partition: [130, 190] -> [290, 640] -> NULL

Allocated new partition: [290, 490]
Free chain after allocation of new partition: [130, 190] -> [490, 640] -> NULL

Freeing partition: [190, 290]
Free chain after freeing the partition: [130, 290] -> [490, 640] -> NULL

Freeing partition: [0, 130]
Free chain after freeing the partition: [0, 290] -> [490, 640] -> NULL

Allocated new partition: [0, 140]
Free chain after allocation of new partition: [140, 290] -> [490, 640] -> NULL

Allocated new partition: [140, 200]
Free chain after allocation of new partition: [200, 290] -> [490, 640] -> NULL

Allocated new partition: [200, 250]
Free chain after allocation of new partition: [250, 290] -> [490, 640] -> NULL

Freeing partition: [140, 200]
Free chain after freeing the partition: [140, 200] -> [250, 290] -> [490, 640] -> NULL
bence@BenceLaptop:~/OPsystems/Exercise 4$ |
```

The main logic behind the program is the list of free memory being stored in a linked list and the allocated partitions being stored as separate entities. The partition structure has a size, a start addresses and a pointer to the next partition. Both free and allocated partitions are of this same type. When adding a new partition, we iterate through the list of free partitions and allocating the first space big enough to store it. We remove or resize the corresponding free memory and allocate space for it. When freeing memory, we iterate through the free list until we find the corresponding empty space in the empty list and then free up the memory of the partition and we will try to merge the free partition to other free partitions from both sides. Both operations use a common function to print out the list of free memory after their operations.

Explanation of code running operation by operation.:

We start with an empty partition the size of 640KB. Job1 applies for 130KB, so the first 130KB are assigned to it. Job2 gets 130-190KB. Job3 gets 190-290KB. We still have a continuous block of free memory. After Job2 is released, there are 2 free partitions. Job4 applies for 200KB, which does not fit into the first free partition, so it gets memory from 290-490KB. Job1 and Job3-s memories are released, the resulting neighbouring free memories are joined, so we still have 2 free memories. Job 5-7 are assigned memories, which all fit into the first empty memory space, after which job6 is released, which leaves us with 3 empty partitions, which we can see in the last line of print.