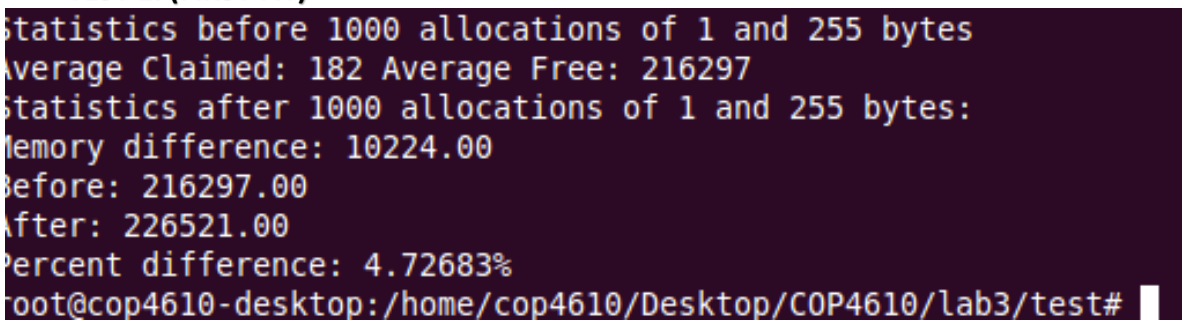
Tiago Moore
Lukas Borges

COP 4610 Lab 3 Report

After looking deeply into the memory allocation architecture of a linux operating system, we have learned extensive amounts on how processes are allocated memory for execution. My main objective was to test the amount of free memory in the operating system after random allocation requests against different memory allocation algorithms. We modified the slob_page_alloc to attempt to allocate a best page. Another system call was implemented to call kmalloc. The funtion that performes this is defined as 100, the function returns the slob with the best fit given a page sp . In addition, a function get_best_block was added to returns the best bloc page, found by our modified find page alloc ,it is subsequently used for comparison. Subsequently, the slob_alloc() function was modified to get us the best fit block using the implemented functions and the logic that saves best page until it is replaced by new best page. We keep track of the variables for the system call when an allocation requested memory is not served(!b).

In my test case for my best fit algorithm – I made 25000 mallocs to force new page creation and memory allocation. These allocations ranged from 16 to 255 bytes. After the loop, we analyzed the that the performance of the best fit algorithm decreased memory fragmentation by increasing the size free memory. The more times we ran the test program with the best-fit implementation, the higher was the free memory. The first-fit implementation also showed an increase In free memory, but it was lower than the best-fit algorithm. To prove this, I calculated the percentage increase of free memory before and after the allocations. With first fit we achieved the following output: Since the amount of memory allocated to the virtual machine was limited, we did not see a significant increase , but if we allocated more memory there would be more memory, resulting in a higher potential difference of free memory.

TEST 1: (FIRST FIT)



TEST 2(BEST FIT):