Assignment 2 – Memory Manager

iLab Machine: adapter

Group Members: Raghav Pandya – rp835

Mayank – um45

Krishna Anantha Padmanabhan – ka478

Implementation Details:

The assignment was implemented in a series of phases as follows:

Phase A:

We initially created a character array of size 8 MB. The system page size was found to be 4096 bits (that is, 2000 pages) and hence we created a structure called **block_meta** for management of pages.

We now created a function called **myallocate()** that takes the size requested from malloc as an argument. We check if the size is within the block limit (otherwise called page limit, that is 4096 bits) and then call the **myallocate_self()** function. It determines the first empty page from the character array and assigns it to the thread. If a thread makes more malloc requests then we allocate it from its assigned page. When the memory is full in the page, we return NULL.

Phase B:

We made changes to the **myallocate()** function in this phase, to support adding more pages to a thread. We check if the size is within the block limit and either call the **myallocate_self()** function or **myallocate_thread()** function based on the type requested (type is to determine if the request is coming from the thread or the library). If the **myallocate_self()** function is called, it determines the first empty page from the character array and assigns it to the thread. If the **myallocate_thread()** function is called (that is, thread's assigned page is full), then we find the next free block from our character array and assign it to the thread. If there is no more free memory in the character array, then we pass NULL.

In order to maintain book-keeping, we are using a global linked list that helps to keep track of the pages that have been used in our system memory (8MB).

Phase C:

In this phase we created another memory region of 16MB that acts as a swap file. If a thread requires more space and the 8 MB memory is full, then we call the **swapout()** function inside the **myallocate()** function. The **swapout()** function looks inside the swap file for an empty space of the required size and swaps out a page from the 8MB space into the swap file and deallocates it's pages from the 8MB space. Now when we call the **myallocate()** function again we will be able to find the freed memory that can be used to grant memory to the thread. When the 8MB space and the 16 MB swap space are both filled up, we pass NULL to indicate that no more memory requests can be taken. Our scheduler and memory manager make sure that any page that has been swapped out is swapped back in so that the relevant thread runs to completion. This is taken care of by using a **swapin()** function that finds a page to be swapped out so that another page can take its space in the 8MB memory. When a certain thread looks for its pages in the 8MB memory and if that page has been swapped out to make space for another thread's page, then it cannot access this page as it is not its own page.

Hence, we have successfully implemented all the required phases and have a memory manager that works in conjunction with our scheduler.