OS Assignment 2

Derek Mao mm2180, Kevin Pei ksp98, Quzhi Li ql88
Tested on ls.cs.rutgers.edu

**Introduction**: This is a program meant to simulate the pthread, mutex, and malloc libraries. The functions allow you to create new threads, yield, exit, and join them, and create mutexes which you can lock and unlock. In addition, the program simulates main memory with an array and allows you to malloc to this main memory array, free from it, and malloc to shared space.

**Mymalloc**: Our malloc function contains memory within a static char array of 8 MB. This 8 MB is split into 4 KB pages and metadata for each page. Within each page is further metadata to keep track of every block of memory that is being used for malloc. Thus, our program consists of both PageData, which keeps track of which thread is using a page, and MemoryData, which keeps track of what memory inside a page is being used and how much by a single malloc call.

PageData contains the address in main memory where the page begins, the pid of the thread using it, its own individual ID, whether it's part of a longer continuous page, and the next page in the sequence if it is continuous. MemoryData itself contains the size of memory it represents, whether that memory is free, and the MemoryData before and after it.

PageData is all stored at the beginning of main memory because if a user mallocs 10 pages, those pages must be contiguous in memory. This means that there can't be metadata keeping track of those pages in or between the pages, because otherwise they would be overwritten by the user who malloc'ed 10 pages. Thus, PageData is stored at the beginning and contains pointers to the pages in memory they refer to. MemoryData itself is stored within the pages because the metadata can be overwritten more freely; you just need to change the size of the previous MemoryData metadata.

Memory is used in conjunction with the scheduler. Thus, depending on which page is running, different pages are swapped in and out of memory. All pages of a given thread are swapped to the beginning of memory when a thread runs. A thread can have multiple pages, which must be put adjacent in memory in order for the malloc call to return a block of memory that is large enough and contiguous.

The last 4 pages of memory are shared memory, which can be allocated using shalloc. These have pid's of -2 to distinguish them from regular empty pages, which have pid's of -1. The shared memory space acts as 4 contiguous pages which will always be at the end of memory.

**Functions**: The functions we implemented were as follows:

My_allocate: It mallocs the given amount of space and returns a pointer to the memory that was malloc'ed. If there is no more space in memory or the swap file, it returns NULL.

my_deallocate: It frees the memory pointed to by the ptr that's passed in as the argument. If the ptr that's passed in does not correlate to a memory block allocated by the currently running thread, then it returns NULL.

Shalloc: It allocates the given amount of space in the shared pages and returns a pointer to the memory that was shalloc'ed. If there is no more space in the shared memory space, it returns NULL.

In addition, there are helper functions to help facilitate the above, such as functions to get the page corresponding to the address given, and functions to swap the pages around so that they would be contiguous in memory.