Ming Tai Ha: mingtha

Harsh Kothari: hk596

Zheng Yu: zy120

Ilab Machine: cd.cs.rutgers.edu

**Assignment 2: Memory Manager**

The assignment is composed of three phases: Directed-Mapped Memory, Virtual Memory, and Swap File. The following description of the implementation of our assignment will be broken down based on the phases.

**Directed-Mapped Memory**

This part of the assignment required that we write our own malloc library for our user-level threads to use, as well as a memory manager in order to maintain the bookkeeping which keeps track of which pages are used by which threads.

The malloc library is based of of dlmalloc, which maintains a set of free lists which keeps track of the free blocks in the page. Each free list is a circularly doubly linked list associated with a certain size of blocks. The first free lists are fixed-sized blocks; so the first 8 free lists are associated with the a block size of 16, 32, 48, 64, 80, 96, 112 and 128, respectively. The next 6 free lists are variable-size blocks; so the next 6 free lists are associated with blocks that are <= 256 Bytes and >128 Bytes, <= 512 Bytes and > 256 Bytes, <=1024 Bytes and > 512 Bytes, <= 2048 Bytes and > 1024 Bytes, <= 4096 Bytes and > 2048 Bytes, and < 4097 Bytes.

Each block is a multiple of 16 bytes, and the size requested by the user always rounded up to the next multiple of 16 Bytes. For a free block, the first 4 bytes and the last 4 bytes denote the size of the block plus 1 (size+1), where the +1 represents that the block is free. After the first 4 bytes used to denote the size and its free/malloc status, the next 8 bytes are used as pointers and store the address of the address of the previous free block and the next free block. Since the free blocks require 8 bytes for size and status and 8 bytes for pointers, the minimum size of a free block is 16 bytes. For malloc'd blocks, a first and last 4 Bytes of a block store the size of the block (+ 0 for malloc'd status); any remaining space is space that can be used by the user.

When the myallocate function is called, it checks whether there is a block of the requested size can be allocated. It first checks the bin associated with the smallest size that can satisfy the request. If a block is found in this list is exactly the rounded size (rounded up by 16), then it is allocated. If a block is found but it is bigger than needed, then myallocate will split the block. Part of the split block is allocated to the user, while the remaining block will be inserted back into the free list associated with the appropriate size. If no bin can satisfy the request, then malloc will return NULL.

When the mydeallocate function is freed, it first checks whether the space to the left and to the right are also free. If at least one of the adjacent blocks are free, then mydeallocate will join the recently freed block with any free adjacent blocks in order to create one contiguous free block. The free block is then inserted back into the appropriate free list for future use. If a user chooses to free a block using a pointer that does not point to the start of any block, mydeallocate will NOT stop the user from freeing the block. Since this behavior is undefined, the user must be careful in using the right pointers to free a block.

The memory manager implemented first memaligns a 8MB static array so that we have one contiguous block in physical memory. Then the memory manager will initialize the first 48 pages in the static array as systems pages reserved for bookkeeping. The system pages are used to store the free list of pages (a circularly doubly linked list of free pages). The reserved pages also store a page structure which contains the thread id to which it is assigned, the current position of the page, the original position of the page, and the pointer to the next page. If the page is assigned to a thread, then the next pointer points to the next page has been assigned to the same thread. If the page is free, then the next page pointer points to the next free page. In the reserved memory, the memory also store thread structures, which stores the thread id and the total heap size of thread (for malloc), a pointer to its stack page, and a list to the heap pages. Finally, we store a two lists: a free list which keeps track of the free pages, and an occupied list which keeps track of the occupied pages. These pages are included for quicker access of pages.

When a scheduler creates a thread, the new thread will call the myallocate function to allocate heap and stack pages for the thread. The additional bookkeeping structures for the pages and thread are maintained. When a thread is finished, the function my\_pthread\_exit will free all of the pages and the associated page and thread structures. The freed pages will be inserted into the free list for later use.

**Virtual Memory**

Since the free lists has the ability to join free blocks together, contiguous blocks of memory can be allocated first by checking whether blocks to the end of the existing memory is free, which is to the left of the newly allocated page. If the end of the existing memory is free, then the newly allocated page block will be merged with the end of the existing memory. In either case, the block is then added to the proper free list associated with the thread.

When the user thread asks for more memory than what its heap currently has, the memory manager will assign a free page to the thread. If the last of the heap of the current thread is occupied by page used by another use, the memory manager will switch in a free page. Now, the heap of the current thread is contiguous. The new page will then be inserted using the method above with joining. If there are no more free pages we return NULL.

The signal handler handles any segmentation faults that occur when a thread tries to access memory but the page which contains the memory has been moved to some other location. If the thread is allowed to access this memory address, then the signal handler finds the appropriate page and swaps it into the location where the thread expects. The pages will have switched locations in physical memory. The pages which are switched out, we use mprotect to protect the memory so that the contents cannot be accessed by another thread. The pages which are switched in are un-mprotected since they will be used by the current thread.

**Swap Management**

Here we have only managed to implement a very basic eviction method: we swap the first page not reserved for bookkeeping into the swap memory (swap file). This occurs when there is not enough memory in the heap of the current thread to satisfy the malloc request AND when there are no free pages in physical memory.