Hemanth Chiluka, Andrew Dos Reis, Benjamin Yang
hkc33, ad1005, bty10
Ilab Used: adapter
03/26/2018

Operating Systems Project 2

Usage

- Use make to compile the my_pthread library.
- NOTE: We added all our my_malloc.c code to our my_pthread.c file (as well as everything required for my_malloc.h to my_pthread.h)
- Make clean removes the compiled my_pthread library
- We ran our test cases under the my_pthread library's directory

Global Data Structures

- We used a global page table to keep track of pages and what frames they were currently using
- The size of the page table is 2048 in RAM pages plus 4096 pages in the swap file.
- In RAM OS pages and User pages are split 1024/ 1024

Other Data Structures

- **page_entry**
    - The actual page in the page entry
    - Attributes
        - currFrame-frame where page's memory is currently stored
        - realFrame- frame where the memory should be when pointer references it
        - Access counter- used for nth chance page replacement algorithm
        - Total access counter - used to keep track of how many times page is referenced
- **frame**
    - Frame in memory
    - Attributes
        - Start - starting meta block in list of nodes
    - An id for the mutex
- **meta**
    - Block of memory in frame
    - Attributes
        - Next
        - Prev

        - Data-pointer that gets given to user

Design

- myallocate
    - Initializes a wrapperstruct for the function pointer and arguments being passed in
    - Traverses curr thread's set of pages through the first page's linked list onward
    - If a page is not in right location, call handler to either swap it back in place from another portion of RAM or from the swap file
    - Continue until free node with greater size than request is seen
- mydeallocate
    - Toggles is free on the meta struct corresponding to the pointer
    - Merges adjacent free blocks
- shalloc
    - Returns a pointer from the shared memory region
    - A region in the last 4 frames of RAM are where this pointer will point to

Testing

- To compile each test case:
    - Gcc -g [file name] -o [executable name]
- Test1.c
    - Creates a thread that runs func2
        - Func2 attempts to force the use of the swapfile
    - This test case runs properly
    - Confirmation that our swapfile methods work
- Test2.c
    - Creates a thread that runs func2
    - Main mallocs 1024 pages of size 4056
    - Pthread_create is called, which runs func2 and attempts another malloc
    - This test case runs properly
    - Confirms again that our swapfile methods work, if the page limit is reached or if myMemory is filled up
- Test3.c
    - Tests to see if malloc returns null, and if so, frees more memory so that when malloc is called again, it returns an actual pointer
    - Runs properly
    - Confirms that our free method and malloc methods work properly
- Test4.c
    - Tests the methods in test3.c on 10 different threads

    - Runs properly
    - Confirms that our free and malloc methods also work on a varying amount of my_pthreads
- Test5.c
    - Tests the shalloc method, and see if the pointer returned can be called by different threads
    - Runs properly
    - Confirms that our shalloc method returns no errors
- Test6.c
    - Tests the shalloc method on more threads, instead of just two as seen in test5.c
    - Runs properly
    - Confirms that our shalloc method runs on any number of threads sharing memory
- testMulti.c
    - Creates six threads that run func1
        - In func1, [XXX] pointers are malloced and the freed
    - Main waits on each thread spawned
    - If [XXX] < 300 ~
        - The program will run properly without any segfaults
    - If [XXX] > 300 ~
        - The program will segfault after the first thread ends
        - This happens because the stack size that was defined for Project 1 is exceeded if more than around 300 pointers are allocated
        - Thus, this is not a concerning error for our program
- Some other tests that are not documented:
    - Pageswapper.c
    - TestMultiType.c
    - TestMultiType2.c
    - All of these files try to stress test our malloc and pthread library
        - They essentially do the methods in tests one through six many times
    - None of these tests fail

<u>Swapping</u>

- Swapping occurs on demand through the use of a signal handler when the memory a ptr is pointing to is mprotected.
- The handler finds the current page corresponding to the faulted frame, and swaps the frame with the frame of the page that is currently supposed to be there. If the page supposed to be there is in the swap file, evict the page currently at the faulted frame to the swap file and put the target in its spot.

EXTRA CREDIT:

- In Memory Swapping
    - The access count of a page is kept track of by incrementing it when it is faulted on
    - When a page fault occurs, the handler first checks if the page currently at the faulting address is the most used in RAM. If it is, the handler grabs another page in the free list and evicts that to the swap file. This is done through an iterative search of the free list to find a page that has a higher frequency than it.
    - The handler first swaps the new victim with the page currently at the faulting address. Next, it swaps the victim with the page in the swap file
- Second Chance Free List
    - Kept track of currently usable pages in RAM with a linked list
    - Implemented a second chance algorithm on the free list
        - Each page has a use bit that is set to 1 each time the page is swapped in
        - To get a new page to swap to, traverse the linked list for a page who is set to 0 while decrementing those already set to 1.
        - Once a new page is found, swap the target page out with it and into its correct location
- Other Page Replacement Algorithm: Nth Chance
    - Another page replacement algorithm we implemented was nth chance, which is extendable from second chance
    - To test the other nth chance algorithms, the max value for the use variable will be n instead of 1.
    - Over n iterations of a full traversal of the linked list, each page entry's use variable is decremented until one of them equals 0. Return that page entry.
    - Please see the graph below for our profiling the running time of our page replacement algorithms:
    -

Nth Chance Time Profiling