CSC 452/552 Operations Systems
Project 3 Threads
Name: Waylon Walsh
Bronco ID: 114086243
Date: 11/25/2024

1. Project Overview

   For this project I implemented a memory manager using the buddy algorithm. it’s based on Donald Knuth's design from "The Art of Computer Programming." The project implements a dynamic memory allocator that manages memory blocks in powers of two, making it efficient for certain types of allocation patterns. The implementation uses an array of free lists (avail array) where each index k contains blocks of size 2^k. When memory is requested, the system finds the smallest available block that can accommodate the request. If the block is too large, it's split into two "buddy" blocks, with one being used for the allocation and the other being added back to the free list. Conversely, when memory is freed, the system checks if the block's "buddy" is also free. If so, they are merged back into a larger block, continuing this process recursively until either a buddy is found to be in use or the maximum block size is reached.

   After a successful implementation of the buddy algorithm I needed to create a comprehensive set of unit tests to get as close to 100% coverage as possible.

2. Project Management Plan
   a) Task 3:

      For this task to implement a memory manager using the buddy system algorithm as described by Knuth. The implementation required creating several key functions: buddy_init for initializing the memory pool using mmap, btok for calculating block sizes in powers of two, buddy_calc

for finding buddy blocks using XOR operations, buddy_malloc for memory allocation with block splitting, buddy_free for memory deallocation with buddy coalescing, and buddy_realloc for resizing allocated memory blocks. A critical constraint was avoiding the use of any math library functions, instead relying on bit shifting and masking for calculations. The implementation had to handle edge cases, maintain proper memory alignment, and provide comprehensive error handling. The system manages memory in power-of-two sized blocks, splitting larger blocks when necessary to satisfy allocation requests and merging buddy blocks during deallocation to prevent fragmentation.

b) Task 4:

Task 4 focused on achieving comprehensive test coverage of the buddy system implementation through unit testing. The goal was to get as close to 100% code coverage as possible, which required testing both normal operation paths and edge cases. The testing suite was expanded beyond the initial professor-provided tests to include scenarios such as memory content verification during reallocation, edge cases for all core functions (init, destroy, malloc, free, realloc), boundary conditions in block size calculations, buddy block calculations, and error handling paths like mmap failures and NULL pointer cases. The test suite used the Unity testing framework and included helper functions to verify pool states (full and empty). Coverage analysis was implemented using gcov with custom Makefile targets for coverage building and reporting. The final test suite achieved 100% line coverage, 100% branch execution, 93.75% of branches taken, and 100% function coverage, demonstrating robust verification of the implementation's correctness and error handling capabilities.

3. Project Deliveries
    a) How to compile and use my code?

        Steps to configure, build, run, and test the project.

        ## Building

        make

        ## Testing

        make check

        ./test-lab

        ## Clean

        make clean

    b) Any self-modification?

        Coverage analysis was implemented using gcov with custom Makefile targets for coverage building and reporting. but choose not to include in submission for ease of compilation.

    c) Summary of Results.

        I believe the project successfully achieved its primary objectives with satisfactory test coverage metrics. The final implementation of the buddy system memory manager demonstrated 100% line coverage, indicating every line of code was executed during testing, 100% of branches were executed, and 93.75% of all possible branch paths were

taken, with 100% of function calls covered. The memory manager correctly handled all test scenarios, including basic allocations and deallocations, block splitting and coalescing, reallocation with content preservation, and various edge cases such as NULL pointer handling, maximum size allocations, and error conditions. All core functionality was verified through comprehensive unit tests, showing proper memory pattern preservation during reallocation, accurate buddy block calculations, and correct error handling for cases like mmap failures. The implementation successfully avoided using math library functions, instead utilizing bit operations for calculations, and maintained proper memory alignment throughout all operations.

4. Self-Reflection of Project

   Initially it was pretty confusing understanding the implementation of the algorithm. I was carefully following Knuth's algorithm while adhering to the requirement of using bit operations instead of math library functions. I still had difficulty after my implementation getting the included test to pass. also compiler warnings arose about format specifiers, I eventually identified and fixed these issues. When working on achieving comprehensive test coverage, where I iteratively added test cases and analyzed coverage reports to identify untested code paths. I systematically added tests for edge cases, error conditions, and specific scenarios like reallocation content preservation, gradually improving coverage from 95.61% to 100% line coverage and increasing branch coverage significantly.

   I feel after completing this project I have a practical understanding of how applications can directly manage their own memory allocation and deallocation without relying on the system's default memory manager. I also gained more hands-on experience with operating system interactions by using low-level system calls. I also have a better understanding of how to break down a project into solvable components.