CS4375-13948 Fall 2023 Homework Report 5

<Javier Venegas> Submitted <12/08/2023>

[javenegas8@miners.utep.edu](mailto:javenegas8@miners.utep.edu)

https://github.com/javenegas8/OperatingSMoore/tree/hw5

**UG HW5: Anonymous Memory Mappings for xv6**

Task 1.

A computer screen with white text

Description automatically generated

1. The errors encountered in the private.c file are related to the implicit declaration of the mmap and munmap functions. These errors occur when the compiler encounters a function call for which it has not seen a prior declaration. Additionally, there is a warning regarding the cast to a pointer from an integer of different size.
2. Ensure that successfully compiled the modified operating system code, including the changes made to trap.c and any other relevant files. Make sure that the "private" program is properly written and included in the build. Ensure it exercises the specific functionality modified or added. Boot up the operating system using your preferred method, whether it's using an emulator like QEMU or running on real hardware. Once the operating system is running, execute the "private" program and observe its behavior. Check for any error messages, crashes, or unexpected behavior.
3. It's crucial to properly free the resources associated with mapped memory regions during the process termination to ensure the correct functioning and stability of the system. The addition of the code you provided in freeproc() helps prevent these issues by releasing the resources associated with mapped regions when a process is being terminated.

Task 2.

a. Uvmcopy: creates independent copies of physical memory for each process, resulting in separate memory spaces.

Uvmcopyshared: shares the same physical memory between processes, allowing changes made by one process to be immediately visible to others.

The choice between these functions depends on the desired memory-sharing behavior between parent and child processes in your operating system.

b. The provided code ensures that private and shared mapped regions are appropriately handled during the fork operation, with separate memory for private regions and shared physical memory for shared regions, along with proper management of the process family for shared regions.

1. prodcons1: Since processes share the same memory mapping, modifications made by the producer process (writing to the buffer) will be visible to the consumer process. This can result in race conditions, and the final total might not be accurate.

prodcons2: Each process has its private copy of the memory. Changes made by one process are not visible to others. This prevents race conditions, and the final total will be accurate, representing the sum of consumed values.

Explanation

The primary difference is in how the programs handle memory sharing. prodcons1 uses shared memory, which can lead to race conditions, while prodcons2 uses private (copy-on-write) memory, preventing race conditions. The final total for prodcons1 might vary due to race conditions, whereas the total for prodcons2 should be accurate.

Task 3.

A black and purple background

Description automatically generated with medium confidence

In this specific case, the program is producing incorrect results because it's using a shared buffer without proper synchronization mechanisms (like mutexes or semaphores) to control access to shared resources. The producer and consumer processes are concurrently modifying the shared variables (nextin, nextout, num\_produced, num\_consumed, total), leading to race conditions.

Accessing pointers and structures within the kernel requires careful attention to type compatibility and correct dereferencing. Ensure that you use the appropriate types and dereference pointers correctly. Compiler warnings and type-checking can help catch potential issues.

Allocating and deallocating memory within the kernel needs to be handled with care to avoid memory leaks or corruption. Use kernel-specific memory allocation functions (kalloc, kfree) and pay attention to freeing memory at the right time to prevent leaks.

**Summary:**

Kernel-level programming is intricate and involves dealing with low-level details such as memory management, process synchronization, and hardware interrupts. The assignment involved acquiring and releasing locks (acquire and release). Proper use of locks is crucial in a multi-process or multi-threaded environment to prevent race conditions and ensure data consistency. The error messages encountered, such as "incompatible pointer types," highlight the importance of understanding and maintaining consistency in data structures. Mixing different types of pointers can lead to compilation errors.