CS4375-13948 Fall 2023 Homework Report

Franz A. Reyes Submitted on 12/08/2023

Fareyes4@miners.utep.edu

**UG HW6: Semaphores for xv6**

Task 3. Implementation of sem\_init(), sem\_wait(), sem\_post(), and sem\_destroy().

* To implement sem\_wait() and sem\_post() in xv6's unnamed semaphores, I utilized the existing sleep() and wakeup() kernel functions. In the sem\_wait() function, processes are put to sleep when the semaphore count is zero, and they are awakened by wakeup() upon receiving a signal from sem\_post(). Coordinating interactions between multiple processes sharing the same semaphore posed a challenge, which was addressed by ensuring proper locking of relevant data structures to prevent race conditions. The integration of sleep() and wakeup() provided an effective means for process synchronization in a multiprocess environment within xv6.

Task 4. Test cases.

* I tested the unnamed semaphore implementation in the xv6 operating system. The testing included basic operations, multiple producer-consumer scenarios, semaphore destruction, stress testing, and edge cases. The results of the testing confirmed that the system is functioning correctly, with robust synchronization and stability, even under stressful conditions. The implementation also exhibited appropriate resource management, and no memory leaks or unexpected behavior were observed. Overall, these results validate the reliability and correctness of the unnamed semaphores in the xv6 kernel.

Kernel bug with our implementation.

* If a program fails to call the sem\_destroy() function, it can cause a resource leak, which may lead to the exhaustion of available semaphore entries in the system. To prevent this from happening, the operating system should implement an automatic cleanup routine upon process termination. This cleanup routine would deallocate semaphores associated with the exiting process, ensuring proper resource management, and preventing indefinite accumulation of unused semaphores in the kernel semaphore table.

**Summary:**

* I gained a practical understanding of implementing semaphores in the xv6 operating system. The hands-on experience involved adding system call declarations, managing kernel-held counters, and integrating synchronization mechanisms using sleep() and wakeup(). The lab emphasized the importance of proper resource management, concurrency control, and systematic testing. Addressing challenges related to multiple process synchronization and potential resource leaks enhanced my knowledge of kernel-level programming concepts. Overall, this lab provided valuable insights into the complexities of implementing synchronization primitives within a real operating system environment.

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated