CS4375-13948 Fall 2023 Homework Report 6

Javier Venegas Submitted on 12/8/2023

[Javenegas8@miners.utep.edu](mailto:Javenegas8@miners.utep.edu)

https://github.com/javenegas8/OperatingSMoore

**UG HW6: Semaphores for xv6**

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

Implementing sem\_wait() and sem\_post() involves ensuring proper synchronization and coordination between processes using semaphores. Begin by acquiring the lock associated with the semaphore to ensure exclusive access when updating the semaphore's state. Check the value of the semaphore. If it is greater than zero, decrement it and proceed. If it is zero, the process needs to sleep until another process increments the semaphore.

* The **sleep()** function is used to put a process to sleep until it is explicitly woken up by another process.
* The **wakeup()** function is used to wake up a sleeping process, allowing it to resume execution.

Ensuring that the implemented functions work correctly under various scenarios. Comprehensive testing, including cases with multiple producers and consumers, to verify the correctness and robustness of the semaphore implementation. By addressing these challenges and implementing the approaches, it creates reliable and effective sem\_wait() and sem\_post() functions in the context of xv6.

**Task 4. Test cases.**

**A computer screen shot of a computer code

Description automatically generatedA screenshot of a computer program

Description automatically generated**

**A screenshot of a computer

Description automatically generatedA screenshot of a computer screen

Description automatically generated**

The code includes the initialization of a semaphore table (semtab) and functions (seminit(), semalloc(), and sedealloc()) for managing semaphores in the xv6 operating system. To ensure the correctness of this implementation, it's important to design comprehensive test cases that cover various scenarios. By using the new implementations of system calls, the test cases are valued into simple numbers in prodcons files to ensure the correct output.

**Kernel bug with our implementation.**

Failure to deallocate semaphores allocated in the kernel semaphore table by a user program could lead to a significant problem. If the operating system neglects to handle these deallocations after program procedures, system failure becomes a possibility. The accumulation of unreleased entries in the semaphore table may result in resource scarcity when more processes require availability, leading to allocation failures. This situation can also hinder the allocation of new entries, potentially causing unexpected outputs and inaccuracies. To address this issue, implementing a process to deallocate unused semaphore entries would be a viable solution.

**Summary:**

In summary, this lab provides a hands-on opportunity for students to deepen their understanding of operating systems, concurrency, and system call implementation, all of which are crucial aspects of systems programming. I learn how to extend an operating system by implementing new system calls. This includes modifying user and kernel-level code to introduce functionality that can be utilized by user programs.