

UG HW6: Semaphores for xv6

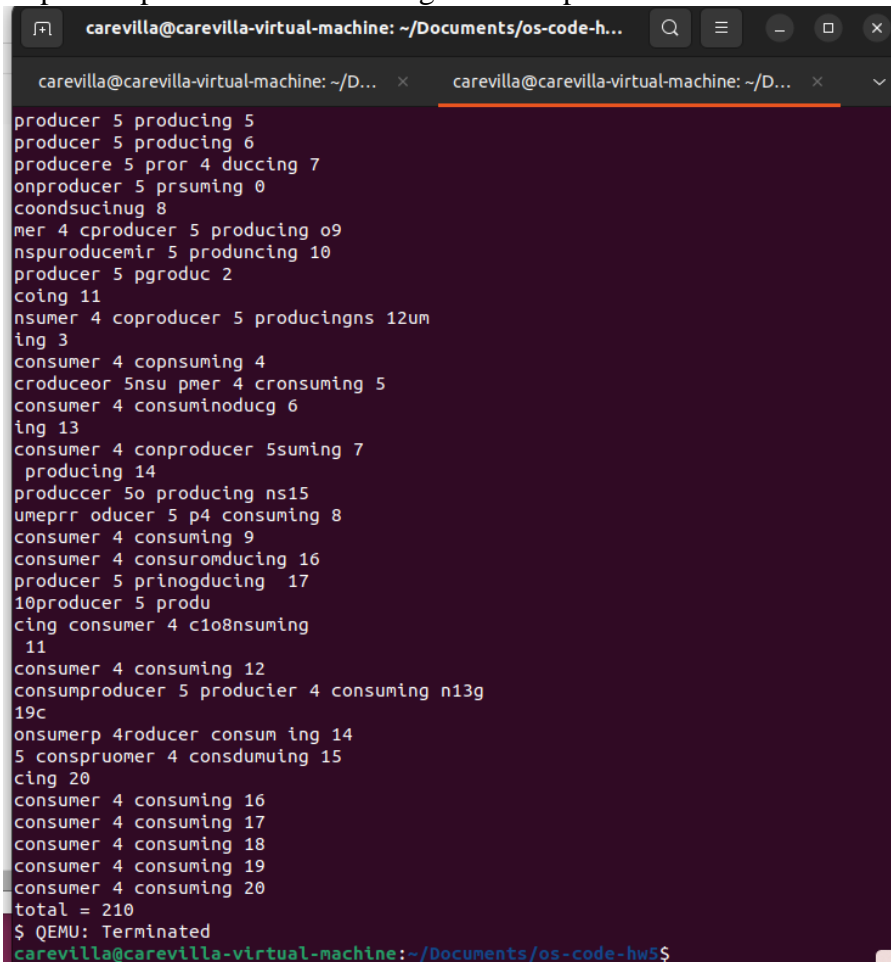
Task 3: Implementation of `sem_init()`, `sem_wait()`, `sem_post()`, and `sem_destroy()`.

1. Following the report template, we were asked to describe the implementation of `sem_wait()` and `sem_post()`
 - a. `sem_wait()` - For this function I first start with checking that register 0 has the memory address that is being passed, which is stored in `temp`. During this check I call `copyin()` to attempt to copy the memory into `addr`, if any of these checks fail, we return `-1`. Once the checks have been verified, we acquire the lock at the `semtables` with `addr`. Utilizing the `sleep` function, we can wait until the count of the semaphore is greater than zero. Once that happens, we break out of the while loop and decrease the count of the semaphore leading to finally releasing the lock associated with the semaphore
 - b. `Sem_post()` - similarly to `sem_wait()` function we begin checking that register 0 has the memory address that is being passed and we check if we can successfully use `copyin()` to copy the memory into `addr`. From here we acquire the lock associated with the semaphore identified by the `addr` variable. Main difference here is that we increment the semaphore count variable to signal that a resource or condition is available to be used or accessed. We then utilize the `wakeup` function which wakes up processes/threads associated with the semaphore given to us is now available. Finally, we release the lock to avoid issues down the line and return a successful 0.
 - c. Overall implementing these system calls posed not many difficulties. The textbook does a good job explaining how to implement them.

Task 4: Test Cases

1. For the test cases portion for this homework assignment, I was not able to successfully implement this portion. After I complied my system calls from task 3 and ran the

respective prodcons-sem.c file I got this output.



```
carevilla@carevilla-virtual-machine: ~/Documents/os-code-h...
carevilla@carevilla-virtual-machine: ~/D... x carevilla@carevilla-virtual-machine: ~/D... x
producer 5 producing 5
producer 5 producing 6
producere 5 pror 4 ducing 7
onproducer 5 prsuming 0
coondsucinug 8
mer 4 cproducer 5 producing o9
nspuroducemir 5 producing 10
producer 5 pproduc 2
coing 11
nsumer 4 coproducer 5 producingns 12um
ing 3
consumer 4 copsuming 4
croduceor 5nsu pmer 4 consuming 5
consumer 4 consuminodug 6
ing 13
consumer 4 conproducer 5suming 7
producing 14
producer 5o producing ns15
umeprrr oducer 5 p4 consuming 8
consumer 4 consuming 9
consumer 4 consuomducing 16
producer 5 prinogducing 17
10producer 5 produ
cing consumer 4 cio8nsuming
11
consumer 4 consuming 12
consumproducer 5 producier 4 consuming n13g
19c
onsumerp 4roducer consum ing 14
5 conspruomer 4 consdumung 15
cing 20
consumer 4 consuming 16
consumer 4 consuming 17
consumer 4 consuming 18
consumer 4 consuming 19
consumer 4 consuming 20
total = 210
$ QEMU: Terminated
carevilla@carevilla-virtual-machine:~/Documents/os-code-hw5$
```

2. As we can see I still do not get smooth print statements and from the expected output that was provided my numbers are slightly off from the consumer and consuming output, however I get the correct total of 210 at the end of the code execution. I noticed on the assignment instructions that if we had not successfully implemented the extra credit from the previous homework, we would have to limit our test cases to producer-consumer scenarios that you are able to test. This was as far as I got in this homework assignment.

Kernel Bug with Implementation:

1. If the user does not properly deallocate semaphores properly it can cause resource leak and resource exhaustion. By the user failing to call `sem_destroy()` the semaphore remains in the kernel's semaphore table without properly being removed leading to many issues. A proposed solution that could be implemented is maybe adding some sort of timing mechanism where if a semaphore is not destroyed within a time period since last used the system automatically will call `sem_destroy()` to deallocate the semaphore.

Summary:

After completion of this homework assignment, I feel I have gained a greater understanding of how operating systems utilize semaphores. The importance of proper code maintenance and management of semaphore system calls and how to properly allocate and deallocate memory in specific semaphores we are utilizing.