

Semaphores for xv6

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

For my `sem_init()` implementation I started by making sure the new semaphore pointer and initial value are user-spaces addresses that are available to use. I used the `semtable` to make sure that I'm keeping track of the allocations and finally I would return the initialized semaphore index.

In the `sys_sem_wait()` function I started by getting the user space address that contains the semaphore index and copying it to our kernel space. Depending on the semaphore count we would decrement the count if it was greater than 0 and releases the lock. If the semaphore count is 0 it meant it couldn't be used and here is where I used the `sleeps()` kernel function to create a sleep and wait loop until there is an available semaphore and decrements count and releases lock.

The `sem_post()` function was implemented by getting the user space address which contains the semaphore index then I copied it to the kernel space and got the semaphore lock. This function increases the semaphore count every time there's a signal indicating availability and it's able to wakeup processes that are waiting with the `wakeup()` function.

For the last function `sys_sem_destroy()` I started by getting the user space address that contained the semaphore objects and saving the index with the `copyin()` function. With that saved index we deallocated the semaphore with the `semdealloc()` function.

A difficulty I ran into while implementing these functions was not really understanding the purpose of each, reading the book sections on semaphores really helped understand the purpose of each function. Another difficulty that I ran into was debugging the functions after getting some panic errors, this was being caused by lock releases and I had to research more about them to fix those errors.

Task 4. Test cases.

For the testcases I used the provided `prodcons-sem` code that was provided for this assignment and made four separate file each testing different numbers, the first three are based on the sample output file provided to us to check that my code was working correctly. This command uses the all four of the functions that we have implemented `sem_init()`, `sem_wait()`, `sem_post()`, and `sem_destroy()`. We were also provided with sample outputs which I used for my first three testcases to make sure that the implementations of those functions were correct. The final output of the program is the sum of all items consumed by the consumer processes indicating the synchronizations made by using semaphores.

```
vboxuser@sam: ~/Myxv6.git/Myxv6
hart 2 starting
init: starting sh
$ testcase1
This is test case 1 prodcons-sem 1 1
producer 5 producing 1
producer 5 producing 2
producer 5 producing 3
producer 5 producing 4
producer 5 producing 5
producer 5 producing 6
producer 5 producing 7
producer 5 producing 8
producer 5 producing 9
producer 5 producing 10
consumer 4 consuming 1
consumer 4 consuming 2
consumer 4 consuming 3
consumer 4 consuming 4
consumer 4 consuming 5
consumer 4 consuming 6
consumer 4 consuming 7
consumer 4 consuming 8
consumer 4 consuming 9
consumer 4 consuming 10
producer 5 producing 11
producer 5 producing 12
producer 5 producing 13
producer 5 producing 14
producer 5 producing 15
producer 5 producing 16
producer 5 producing 17
producer 5 producing 18
producer 5 producing 19
producer 5 producing 20
consumer 4 consuming 11
consumer 4 consuming 12
consumer 4 consuming 13
consumer 4 consuming 14
consumer 4 consuming 15
consumer 4 consuming 16
```

Figure 1: Part (1/2) First testcase

```
consumer 4 consuming 17
consumer 4 consuming 18
consumer 4 consuming 19
consumer 4 consuming 20
total = 210
```

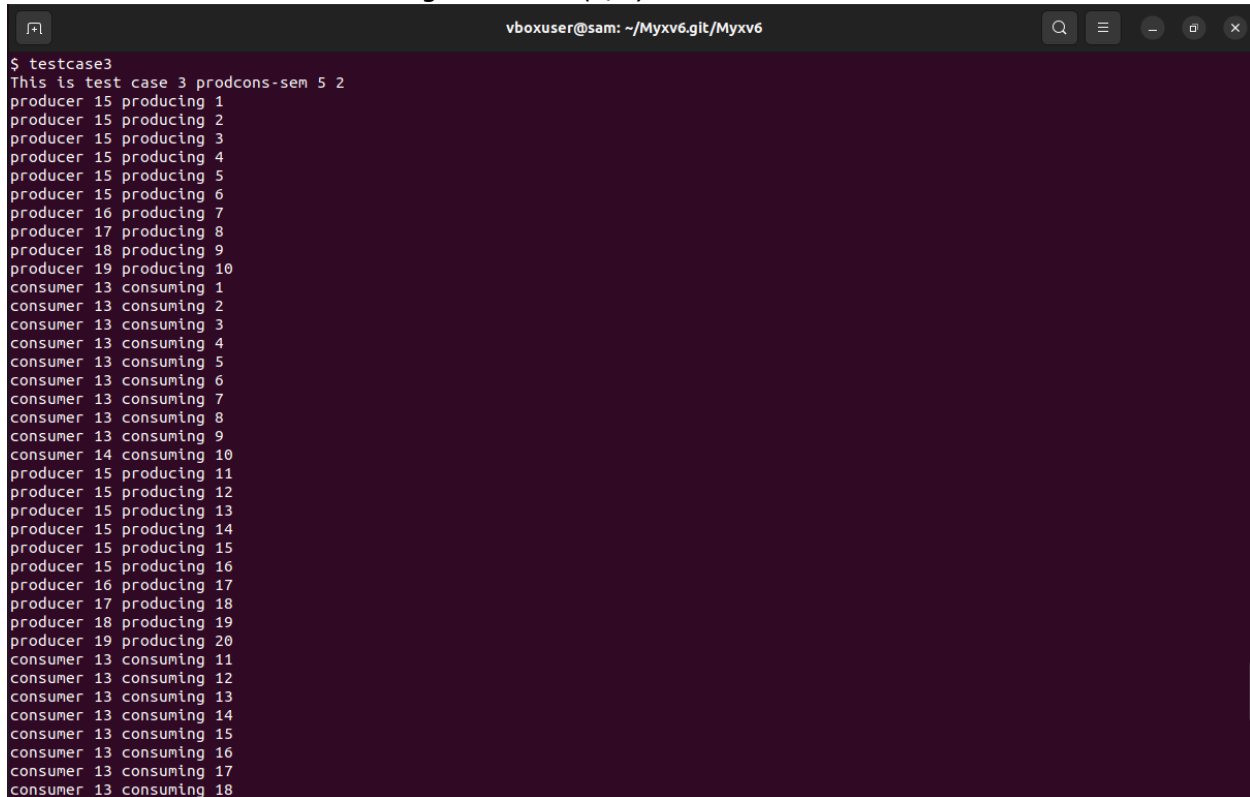
Figure 2: Part (2/2) First testcase

```
$ testcase2
This is test case 2 prodcons-sem 2 3
producer 10 producing 1
producer 10 producing 2
producer 10 producing 3
producer 10 producing 4
producer 10 producing 5
producer 10 producing 6
producer 10 producing 7
producer 10 producing 8
producer 10 producing 9
producer 11 producing 10
consumer 7 consuming 1
consumer 7 consuming 2
consumer 7 consuming 3
consumer 7 consuming 4
consumer 7 consuming 5
consumer 7 consuming 6
consumer 7 consuming 7
consumer 7 consuming 8
consumer 8 consuming 9
consumer 9 consuming 10
producer 10 producing 11
producer 10 producing 12
producer 10 producing 13
producer 10 producing 14
producer 10 producing 15
producer 10 producing 16
producer 10 producing 17
producer 10 producing 18
producer 10 producing 19
producer 11 producing 20
consumer 7 consuming 11
consumer 7 consuming 12
consumer 7 consuming 13
```

Figure 3: Part (1/2) Second testcase

```
consumer 7 consuming 14
consumer 7 consuming 15
consumer 7 consuming 16
consumer 7 consuming 17
consumer 7 consuming 18
consumer 8 consuming 19
consumer 9 consuming 20
total = 210
```

Figure 4: Part (2/2) Second testcase



```
$ testcase3
This is test case 3 prodcons-sem 5 2
producer 15 producing 1
producer 15 producing 2
producer 15 producing 3
producer 15 producing 4
producer 15 producing 5
producer 15 producing 6
producer 16 producing 7
producer 17 producing 8
producer 18 producing 9
producer 19 producing 10
consumer 13 consuming 1
consumer 13 consuming 2
consumer 13 consuming 3
consumer 13 consuming 4
consumer 13 consuming 5
consumer 13 consuming 6
consumer 13 consuming 7
consumer 13 consuming 8
consumer 13 consuming 9
consumer 14 consuming 10
producer 15 producing 11
producer 15 producing 12
producer 15 producing 13
producer 15 producing 14
producer 15 producing 15
producer 15 producing 16
producer 16 producing 17
producer 17 producing 18
producer 18 producing 19
producer 19 producing 20
consumer 13 consuming 11
consumer 13 consuming 12
consumer 13 consuming 13
consumer 13 consuming 14
consumer 13 consuming 15
consumer 13 consuming 16
consumer 13 consuming 17
consumer 13 consuming 18
```

Figure 5: Part (1/2) Third testcase

```
consumer 13 consuming 19
consumer 14 consuming 20
total = 210
```

Figure 5: Part (2/2) Third testcase

```

$ testcase4
This is test case 4 prodcons-sem 4 7
producer 22 producing 1
producer 22 producing 2
producer 22 producing 3
producer 22 producing 4
producer 22 producing 5
producer 22 producing 6
producer 22 producing 7
producer 22 producing 8
producer 22 producing 9
producer 22 producing 10
consumer 21 consuming 1
consumer 21 consuming 2
consumer 21 consuming 3
consumer 21 consuming 4
consumer 21 consuming 5
consumer 21 consuming 6
consumer 21 consuming 7
consumer 21 consuming 8
consumer 21 consuming 9
consumer 21 consuming 10
producer 22 producing 11
producer 22 producing 12
producer 22 producing 13
producer 22 producing 14
producer 22 producing 15
producer 22 producing 16
producer 22 producing 17
producer 22 producing 18
producer 22 producing 19
producer 22 producing 20
consumer 21 consuming 11
consumer 21 consuming 12
consumer 21 consuming 13
consumer 21 consuming 14
consumer 21 consuming 15

```

Figure 6: Part (1/2) Fourth testcase

```

consumer 21 consuming 16
consumer 21 consuming 17
consumer 21 consuming 18
consumer 21 consuming 19
consumer 21 consuming 20
total = 210
$ █

```

Figure 7: Part (2/2) Fourth testcase

Kernel bug with our implementation.

If our user program doesn't call on our `sem_destroy()` function in order to deallocate semaphores it would probably start a resource leak in which the program would keep using kernel resources and reach the limit we set for NSEM. A proposed solution for this would be to clean up the semaphores that are associated with the process when it's exiting, by deallocating the semaphores that are not being used.

Summary:

This exercise helped me really understand what semaphores are, how they work and how to implement them to xv6. I learned how to create implementations of `sem_init()`, `sem_wait()`, `sem_post()`, and `sem_destroy()` and the semaphore file. I learned why locks are important and how they affect the program, and I kept learning how to be better at debugging my errors.