CS4375-13948 Fall 2023
Omar Hoang
oahoang@miners.utep.edu

Homework Report
Submitted on 12/8/2023

UG HW6: Semaphores for xv6

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

```
217  uint64
218  sys_sem_wait(void){
219
220    struct proc *p = myproc();
221    uint64 addr;
222    int index;
223
224    if(argaddr(0,&addr) < 0){
225      return -1;
226    }
227
228    copyin(p->pagetable,(char*) &index, addr, sizeof(int));
229    acquire(&semtable.sem[index].lock);
230
231    if(semtable.sem[index].count > 0){
232
233      semtable.sem[index].count--;
234      release(&semtable.sem[index].lock);
235      return 0;
236    }else{
237
238      while(semtable.sem[index].count == 0){
239
240        sleep((void*)&semtable.sem[index], &semtable.sem[index].lock);
241      }
242
243      semtable.sem[index].count -= 1;
244      release(&semtable.sem[index].lock);
245    }
246
247    return 0;
248
249  }
250
```

For sem_wait() the way I used sleep() if the count of the semaphore at the time equals 0 then there is nothing for the current process, either the producer or consumer, to do so the process would be put to sleep.

```
251  uint64
252  sys_sem_post(void){
253
254    struct proc *p = myproc();
255    uint64 addr;
256    int index;
257
258    if(argaddr(0, &addr) < 0){
259      return -1;
260    }
261
262    copyin(p->pagetable,(char *)&index, addr, sizeof(int));
263    acquire(&semtable.sem[index].lock);
264
265    semtable.sem[index].count += 1;
266    wakeup((void*)&semtable.sem[index]);
267
268    release(&semtable.sem[index].lock);
269
270    return 0;
271
272  }
```

For sem_post() the way I used wakeup() was after the semaphore's count was incremented. The process, either the producer or consumer, will then be woken up and continue running.

The main difficulty I had with this task was using copyingin() and copyout(), because I kept passing the wrong values to them. I did some research on both functions and asked the TA for guidance which helped me overcome this obstacle.

Task 4. Test cases.

```
$ $ hw6test
test 1:
producer 5 producing 1
producer 6 producing 2
producer 5 producing 3
consumer 10 consuming 1
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 6 producing 10
producer 7 producing 11
consumer 8 consuming 2
producer 6 producing 12
consumer 8 consuming 3
producer 5 producing 13
consumer 8 consuming 4
consumer 8 consuming 5
consumer 8 consuming 6
consumer 8 consuming 7
consumer 8 consuming 8
consumer 8 consuming 9
consumer 8 consuming 10
producer 5 producing 14
producer 5 producing 15
producer 5 producing 16
producer 5 producing 17
producer 5 producing 18
producer 6 producing 19
producer 7 producing 20
consumer 8 consuming 11
consumer 8 consuming 12
consumer 8 consuming 13
consumer 8 consuming 14
consumer 8 consuming 15
consumer 8 consuming 16
consumer 8 consuming 17
consumer 8 consuming 18
consumer 9 consuming 19
consumer 8 consuming 20
```

```
total = 210
test 2:
producer 11 producing 1
producer 11 producing 2
producer 11 producing 3
consumer 17 consuming 1
producer 11 producing 4
producer 12 producing 5
producer 11 producing 6
producer 11 producing 7
producer 11 producing 8
producer 13 producing 9
producer 14 producing 10
consumer 15 consuming 2
consumer 15 consuming 3
consumer 15 consuming 4
consumer 15 consuming 5
consumer 15 consuming 6
consumer 15 consuming 7
consumer 16 consuming 8
consumer 17 consuming 9
producer 11 producing 11
producer 12 producing 12
consumer 16 consuming 10
producer 12 producing 13
consumer 16 consuming 11
producer 11 producing 14
consumer 16 consuming 12
producer 11 producing 15
producer 12 producing 16
consumer 17 consuming 13
producer 11 producing 17
producer 11 producing 18
producer 11 producing 19
producer 11 producing 20
consumer 15 consuming 14
consumer 15 consuming 15
consumer 15 consuming 16
consumer 15 consuming 17
consumer 15 consuming 18
consumer 15 consuming 19
consumer 15 consuming 20
```

```
test 3:
producer 19 producing 1
producer 20 producing 2
producer 19 producing 3
producer 19 producing 4
producer 19 producing 5
producer 20 producing 6
producer 19 producing 7
producer 21 producing 8
producer 22 producing 9
producer 23 producing 10
consumer 24 consuming 1
consumer 25 consuming 2
producer 20 producing 11
producer 19 producing 12
consumer 24 consuming 3
producer 19 producing 13
consumer 24 consuming 4
consumer 26 consuming 5
producer 19 producing 14
consumer 24 consuming 6
producer 19 producing 15
consumer 24 consuming 7
consumer 24 consuming 8
producer 20 producing 16
producer 21 producing 17
consumer 24 consuming 9
producer 20 producing 18
consumer 24 consuming 10
consumer 24 consuming 11
consumer 24 consuming 12
consumer 24 consuming 13
producer 19 producing 19
producer 20 producing 20
consumer 24 consuming 14
consumer 24 consuming 15
consumer 24 consuming 16
consumer 25 consuming 17
consumer 25 consuming 18
consumer 25 consuming 19
consumer 25 consuming 20
total = 210
```

```
test 4:
producer 29 producing 1
producer 29 producing 2
producer 29 producing 3
producer 29 producing 4
consumer 38 consuming 1
producer 29 producing 5
producer 30 producing 6
consumer 39 consuming 2
producer 29 producing 7
producer 30 producing 8
producer 31 producing 9
producer 32 producing 10
producer 33 producing 11
consumer 35 consuming 3
producer 34 producing 12
producer 29 producing 13
consumer 35 consuming 4
producer 29 producing 14
consumer 36 consuming 5
producer 29 producing 15
consumer 35 consuming 6
consumer 35 consuming 7
consumer 35 consuming 8
consumer 35 consuming 9
producer 32 producing 16
producer 29 producing 17
producer 30 producing 18
producer 31 producing 19
consumer 35 consuming 10
producer 29 producing 20
consumer 35 consuming 11
consumer 36 consuming 12
consumer 35 consuming 13
consumer 36 consuming 14
consumer 35 consuming 15
consumer 35 consuming 16
consumer 36 consuming 17
consumer 36 consuming 18
consumer 37 consuming 19
consumer 35 consuming 20
total = 210
```

For my test program, hw6test.c, I used the same buffer implementation as the on provided in prodcons-sem_sem.c and created a function to go over all four tests at the same time producing the results above.

Kernel bug with our implementation.

If the program does not call sem_destroy() the OS can be vulnerable to resource leaks or can cause the system to deadlock after exhausting the memory being used by the semaphores stored.

Summary:
I learned how semaphores can be a useful resource when multiple processes are being handled by the OS. The assignment also gave me a better understanding on how acquiring and releasing locks are important when it comes to concurrency control.