Ruben Carmona

CS4375-13948 Fall 2023
Ruben Carmona
rcarmoname@miners.utep.edu

Homework Report
Submitted on December 7, 2023

UG HW6: Semaphores for xv6

https://github.com/Rcarmonam/myxv6RubenCarmona.git

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

sem_init()

```c
uint64
sys_sem_init(void){
  uint64 sem_addr;
  int init_val, pshared, index;

  if(argaddr(0,&sem_addr)<0){
    return -1;
  }
  if(argint(1,&pshared)<0){
    return -1;
  }
  if(argint(2,&init_val)<0){
    return -1;
  }
  if(pshared == 0){
    return -1;
  }
  index = semalloc();
  semtable.sem[index].count = init_val;
  if(copyout(myproc()->pagetable,sem_addr,(char*)&index,sizeof(index))<0){
    //semdealloc(index);
    return -1;
  }
 return 0;
}
```

sem_wait()

```c
uint64
sys_sem_wait(void){
  uint64 sem_addr;

  if(argaddr(0,&sem_addr)<0){
    return -1;
  }
  int sem_index;

  copyin(myproc()->pagetable,(char*)&sem_index,sem_addr,sizeof(int));

  acquire(&semtable.sem[sem_index].lock);

  if(semtable.sem[sem_index].count > 0){
    semtable.sem[sem_index].count--;
    release(&semtable.sem[sem_index].lock);
    return 0;
  }else{
    while(semtable.sem[sem_index].count == 0){
      sleep((void*)&semtable.sem[sem_index], &semtable.sem[sem_index].lock);
      //release(&semtable.sem[addr].lock);
    }
    semtable.sem[sem_index].count -=1;
    release(&semtable.sem[sem_index].lock);
  }
  return 0;
}
```

sem_post()

```c
uint64
sys_sem_post(void){
  uint64 sem_addr;

  if(argaddr(0,&sem_addr)<0){
    return -1;
  }
  int sem_index;

  copyin(myproc()->pagetable,(char*)&sem_index,sem_addr,sizeof(int));

  acquire(&semtable.sem[sem_index].lock);
  semtable.sem[sem_index].count +=1;
  wakeup((void*)&semtable.sem[sem_index]);
  release(&semtable.sem[sem_index].lock);
  return 0;
}
```

sem_destroy()

```c
uint64
sys_sem_destroy(void){
  uint64 sem_addr;


  if(argaddr(0,&sem_addr)<0){
    return -1;
  }
  int sem_index;
  acquire(&semtable.lock);

  if(copyin(myproc()->pagetable,(char*)&sem_index,sem_addr,sizeof(int))<0 ){
    release(&semtable.lock);
    return -1;
  }
  semdealloc(sem_index);
  release(&semtable.lock);
  return 0;
}
```

- In implementing sys_sem_wait() and sys_sem_post(), the key challenges lies in correctly managing process synchronization via semaphores. sys_sem_wait() decrements the semaphore's count and puts the process to sleep using sleep() if the count is zero, indicating the resource is unavailable. The process remains asleep until another process increments the semaphore count through sys_sem_post(), which then uses wakeup() to awaken any sleeping processes. This mechanism ensures that resources are accessed in an orderly fashion, preventing race conditions. The main difficulty in such implementations is ensuring atomicity and deadlock avoidance, which is addressed through careful use of locking mechanisms and condition checks.

Task 4. Test cases.

The testing strategy for test_prodcon.c aims to validate the functionality, concurrency control, and scalability of the producer-consumer solution using semaphores. The tests range from a basic setup with one producer and consumer to more complex scenarios involving multiple producers and consumers, ensuring the system's robustness under various loads. These tests are crucial for verifying seamless operation without deadlocks, maintaining data integrity in concurrent access, and assessing the system's behavior in balanced and high-load situations. The results will demonstrate the practical viability of semaphore-based synchronization in handling scenarios effectively.

Prodcons-sem 1 1

```
xv6 kernel is booting

hart 1 starting
hart 2 starting
init: starting sh
$ prodcons-sem 1 1
producer 5 producing 1
consumer 4 consuming 1
producer 5 producing 2
consumer 4 consuming 2
producer 5 producing 3
consumer 4 consuming 3
producer 5 producing 4
consumer 4 consuming 4
producer 5 producing 5
producer 5 producing 6
producer 5 producing 7
producer 5 producing 8
producer 5 producing 9
consumer 4 consuming 5
producer 5 producing 10
producer 5 producing 11
consumer 4 consuming 6
consumer 4 consuming 7
producer 5 producing 12
consumer 4 consuming 8
producer 5 producing 13
producer 5 producing 14
producer 5 producing 15
consumer 4 consuming 9
producer 5 producing 16
consumer 4 consuming 10
producer 5 producing 17
consumer 4 consuming 11
producer 5 producing 18
consumer 4 consuming 12
producer 5 producing 19
consumer 4 consuming 13
producer 5 producing 20
consumer 4 consuming 14
consumer 4 consuming 15
consumer 4 consuming 16
consumer 4 consuming 17
consumer 4 consuming 18
consumer 4 consuming 19
consumer 4 consuming 20
total = 210
```

prodcons-sem 2 3

```
xv6 kernel is booting

hart 1 starting
hart 2 starting
init: starting sh
$ prodcons-sem 2 3
producer 7 producing 1
producer 8 producing 2
consumer 5 consuming 1
producer 7 producing 3
consumer 6 consuming 2
consumer 5 consuming 3
producer 8 producing 4
producer 8 producing 5
consumer 4 consuming 4
producer 7 producing 6
consumer 4 consuming 5
consumer 5 consuming 6
producer 7 producing 7
producer 7 producing 8
consumer 5 consuming 7
consumer 4 consuming 8
producer 8 producing 9
producer 7 producing 10
consumer 5 consuming 9
producer 7 producing 11
producer 7 producing 12
producer 7 producing 13
consumer 5 consuming 10
consumer 6 consuming 11
consumer 5 consuming 12
consumer 4 consuming 13
producer 8 producing 14
consumer 4 consuming 14
producer 7 producing 15
consumer 4 consuming 15
producer 7 producing 16
producer 8 producing 17
consumer 5 consuming 16
consumer 4 consuming 17
producer 8 producing 18
consumer 5 consuming 18
producer 8 producing 19
producer 7 producing 20
consumer 4 consuming 19
consumer 6 consuming 20
total = 210
```

prodcons-sem 5 2

```
xv6 kernel is booting

hart 1 starting
hart 2 starting
init: starting sh
$ prodcons-sem 5 2
producer 6 producing 1
producer 7 producing 2
consumer 4 consuming 1
consumer 5 consuming 2
producer 6 producing 3
consumer 5 consuming 3
producer 6 producing 4
producer 6 producing 5
producer 6 producing 6
consumer 4 consuming 4
producer 6 producing 7
consumer 5 consuming 5
consumer 4 consuming 6
consumer 5 consuming 7
producer 6 producing 8
producer 7 producing 9
consumer 4 consuming 8
consumer 5 consuming 9
producer 6 producing 10
consumer 4 consuming 10
producer 7 producing 11
consumer 5 consuming 11
producer 6 producing 12
consumer 4 consuming 12
producer 6 producing 13
consumer 4 consuming 13
producer 6 producing 14
consumer 4 consuming 14
producer 7 producing 15
consumer 4 consuming 15
producer 6 producing 16
consumer 4 consuming 16
producer 6 producing 17
consumer 4 consuming 17
producer 6 producing 18
producer 7 producing 19
consumer 4 consuming 18
consumer 5 consuming 19
producer 6 producing 20
consumer 5 consuming 20
total = 210
```

test_prodcon.c 4, M × defs.h usys.pl

user > test_prodcon.c > main(int, char * [])

```c
void run_test(int nproducers, int nconsumers) {
    int i;
    for (i = 0; i < nproducers; i++)
        if (!fork()) {
            producer();
            exit(0);
        }
    for (i = 0; i < nconsumers; i++)
        if (!fork()) {
            consumer();
            exit(0);
        }
    for (i = 0; i < nproducers + nconsumers; i++)
        wait(0);
}

int main(int argc, char *argv[]) {
    if (argc != 1) {
        printf("usage: %s\n", argv[0]);
        exit(1);
    }

    // Test 1:
    printf("Test 1:\n");
    run_test(1, 1);

    // Test 2:
    printf("Test 2:\n");
    run_test(3, 3);

    // Test 3:
    printf("Test 3:\n");
    run_test(2, 2);

    // Test 4:
    printf("Test 4:\n");
    run_test(4, 4);

    exit(0);
}
```

Kernel bug with our implementation.

To address this issue the operating system should implement a mechanism to clean up semaphores created by a user program when the program exits. By implementing such a mechanism, the OS can ensure that semaphores are properly cleaned up and resources are released when user programs exit, preventing resource leaks and resource exhaustion. This helps maintain the system's stability and performance in multi-programming environments.

Summary:

Through this lab, I gained a deeper understanding of process synchronization and semaphore mechanisms in operating systems. Implementing functions like sem_init(), sem_wait(), sem_post(), and sem_destroy() highlighted the intricacies of managing process synchronization, particularly in ensuring atomicity and preventing deadlocks while using semaphores. Testing various producer-consumer scenarios with different numbers of processes underscored the importance of robust concurrency control and scalability in applications. Overall, this lab was instrumental in enhancing my practical knowledge of operating systems concepts and their application in complex, multi-process environments.