CS4375-13948 Fall 2023
Sonam S Tshering
sstshering@miners.utep.edu

Homework Report
Submitted on 12/16/2023

UG HW6: Semaphores for xv6

Task 1. Implementation of sem_init(), sem_wait(), sem_post(), and sem_destroy().

Task 3. Added sys_sem_init(), sys_sem_destroy(), sys_sem_wait(), and sys_sem_post() to sysproc.c in kernel.

```
uint64
sys_sem_init(void){

  struct proc *p = myproc();

  uint64 addr;
  int start;
  int pshared;
  int index;

  if(argaddr(0, &addr) < 0){
    return -1;
  }

  if(argint(1, &pshared) < 0){
    return -1;
  }

  if(pshared == 0){
    return -1;
  }

  if(argint(2, &start) < 0){
    return -1;
  }

  index = semalloc();
  semtable.sem[index].count = start;

  if(copyout(p->pagetable,addr,(char*) &index, sizeof(index)) < 0){
    return -1;
  }

  return 0;
```

```
uint64
sys_sem_destroy(void){

  struct proc *p = myproc();

  uint64 addr;
  int index;

  if(argaddr(0,&addr) < 0){
    return -1;
  }

  acquire(&semtable.lock);

  if(copyin(p->pagetable,(char*) &index, addr, sizeof(int)) < 0){
    release(&semtable.lock);
    return 0;
  }

  semadealloc(index);
  release(&semtable.lock);
  return 0;

}
```

```
 uint64
 sys_sem_wait(void){

   struct proc *p = myproc();
   uint64 addr;
   int index;

   if(argaddr(0,&addr) < 0){
     return -1;
   }

   copyin(p->pagetable,(char*) &index, addr, sizeof(int));
   acquire(&semtable.sem[index].lock);

   if(semtable.sem[index].count > 0){

     semtable.sem[index].count--;
     release(&semtable.sem[index].lock);
     return 0;
   }else{

     while(semtable.sem[index].count == 0){

       sleep((void*)&semtable.sem[index], &semtable.sem[index].lock);
     }

     semtable.sem[index].count -= 1;
     release(&semtable.sem[index].lock);
   }

   return 0;

 }
```

```
uint64
sys_sem_post(void){

  struct proc *p = myproc();
  uint64 addr;
  int index;

  if(argaddr(0, &addr) < 0){
    return -1;
  }

  copyin(p->pagetable,(char *)&index, addr, sizeof(int));
  acquire(&semtable.sem[index].lock);

  semtable.sem[index].count += 1;
  wakeup((void*)&semtable.sem[index]);

  release(&semtable.sem[index].lock);

  return 0;

}
```

Task 2. Update Spinlock.h and add semalloc() in semaphore.c

```
You, yesterday | 1 author (You)
struct semaphore {
  struct spinlock lock;
  int count;
  int valid;
};

// OS semaphore table type
You, yesterday | 1 author (You)
struct semtab {
  struct spinlock lock;
  struct semaphore sem[NSEM];
};

extern struct semtab semtable;
```

```c
struct semtab semtable;

void
seminit(void){
    initlock(&semtable.lock, "semtable");
    for(int i = 0; i < NSEM; i++){
        initlock(&semtable.sem[i].lock, "sem");
    }
}

int      You, 5 days ago • hw6
semalloc(void){

    acquire(&semtable.lock);
    for (int i = 0; i < NSEM; i++){
        if(semtable.sem[i].valid == 0){
            semtable.sem[i].valid = 1;
            release(&semtable.lock);
            return i;
        }
    }
    release(&semtable.lock);
    return -1;
}

void
semadealloc(sem_t index){

    semtable.sem[index].valid = 0;
}
```

Task 4. Test cases.

For my test program, hw6test.c, I implemented the buffer code provided in prodcons_sem.c and created a function to go over all four tests at the same time producing the results above. Although, I wasn't able to run the testcases, due to a kernel error.

From the assignment, I gained insight into the significance of semaphores, which are like traffic controllers for processes in an OS. They help manage multiple processes smoothly. I also learned more about the importance of using locks, to control the flow of activities happening simultaneously.