Diego Avina-Escobedo

Shirley Moore

CS 4375

HW 6: Semaphore implementation

Task 1a: System Call Declarations:

In the fist task, we extended the xb6 operating system by incorporating system call declarations for semaphore. We added **sem_init(), sem_wait(), sem_destroy(),** and **sem_post()** to user/user.h and introduced a **sem_t type.** Modification were made to user/usys.pl. kernel/syscall.h, and kernel/syscall.c to intergrate these new system calls seamlessly. Additionally, we included the test program prodcons-sem.c in the user directory and updated the UPROGS in Makefile. The compilation was tested using make qemu to identify and rectify any compilation errors.

```
int sem_init(sem_t *sem, int pshared, unsigned int value);
int sem_destroy(sem_t *sem);
int sem_wait(sem_t *sem);
int sem_post(sem_t *sem);
```

```
entry("sem_init");
entry("sem_destroy");
entry("sem_wait");
entry("sem_post");
```

```
#define SYS_sem_init 26
#define SYS_sem_destroy 27
#define SYS_sem_wait 28
#define SYS_sem_post 2
```

Task 1b: data structure definitions for Semaphores:

To support the implementation of semaphores, we defined counting semaphore data structures in spinlock.h. The structures included a semaphore with a spinlock, count, and a validity flag. Additionally, we introduced a semtab structure to serve as table for semaphores. The kernel/param.h file was updated to include #define NSEM 100, representing the maximum open semaphores per system. A new file, semaphore.c was created to manage the semaphores, initializing them and providing functions such as semalloc() and semdealloc() for allocation and deallocation. These changes set the foundation for handling semaphores within the xv6 operating system.

```
struct semaphore{
  struct spinlock lock;
  int count;
  int valid; // 1 if this entry is in use
};

struct semtab {
    struct spinlock lock;
    struct semaphore sem[NSEM];
};

extern struct semtab semtable;
```

```
struct semtab semtable;

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

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

void sedealloc(int index){
    acquire(&semtable.sem[index].lock);
    if(index >= 0 && index < NSEM){
        semtable.sem[index].valid = 0;
    }
    release(&semtable.sem[index].lock);
}
```

Task 2: Implementation of system calls:

The next step involved the actual implementation of system calls related to sempahores. We added code for sys_sem_init(), sys_sem_destroy(), sys_sem_wait(), and sys_sem_post in kernel/sysproc.c. This implementation adhered to guidance provided in sections 7.5 and 7.6 of the xv6 textbook. To interact with user space, copyout() and copyin() were employed. Various user and kernel files were updated to accommodate the new system calls. To ensure the correct integration of these calls, testing was performed and any discrepancies were addressed, ensuring the seamless functioning of semaphore-related system calls within xv6

```
int sys_sem_init(void) {
    uint64 s;
    int index, value, pshared;

    if (argaddr(0, &s) < 0 || argint(1, &pshared) < 0 || argint(2, &value) < 0) {
        return -1;
    }

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

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

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

    return 0;
}
```

```
int sys_sem_destroy(void) {
    uint64 s;
    int addr;

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

    acquire(&semtable.lock);

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

    sedealloc(addr);
    release(&semtable.lock);

    return 0;
}
```

```c
int sys_sem_wait(void) {
    uint64 s;
    int addr;

    if (argaddr(0, &s) < 0 || copyin(myproc()->pagetable, (char*)&addr, s, sizeof(int)) < 0) {
        return -1;
    }

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

    while (semtable.sem[addr].count == 0) {
        sleep((void*)&semtable.sem[addr], &semtable.sem[addr].lock);
    }

    semtable.sem[addr].count--;
    release(&semtable.sem[addr].lock);

    return 0;
}
```

```c
int sys_sem_post(void) {
    uint64 s;
    int addr;

    if (argaddr(0, &s) < 0 || copyin(myproc()->pagetable, (char*)&addr, s, sizeof(int)) < 0) {
        return -1;
    }

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

    semtable.sem[addr].count++;
    wakeup((void*)&semtable.sem[addr]);

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

    return 0;
}
```

Task 3: Test Cases:

To validate the functionality of the semaphores implementation, we devised at leat four distinct tests. These tests were designed to cover different scenarios and usage patterns of semaphores. The goal was to confirm that the implemented semaphores produced correct and expected result under various conditions.

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

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

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

Conclusion:

This lab provided a comprehensive overview of the process of implementing unnamed semaphores in the xv 6 operating system. The systematic approach involved task-wise enhancements, starting with system call declarations, followed by the definition of data structures, implementation of system calls, and validation through diverse test cases.