CS4375-13948 Fall 2023 Homework Report
Denisse Loya Submitted on 12-08-2023
dloyavilla@miners.utep.edu

UG HW6: Semaphores for xv6

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

1. *Setup and Declarations:*

   - Added system call declarations for sem_init(), sem_wait(), sem_post(), and sem_destroy() to user/user.h.
   - Added a definition for sem_t type and updated user/usys.pl, kernel/syscall.h, kernel/syscall.c, and kernel/types.h as needed.
   - Included prodcons-sem.c in UPROGS in the Makefile and added it to the user directory.

   By typing "make qemu," compilation errors were fixed.

2. *Semaphores Data Structure Definitions:*

   - Added definitions for the semaphore and semtab data structures to spinlock.h.
   - A defined structural semaphore featuring a validity, count, and spinlock indication.
   - A spinlock and an array of semaphores were included in a struct semtab that was created to represent the semaphore table.
   - To set the maximum number of open semaphores per system, add #define NSEM 100 to kernel/param.h.
   - In semaphore.c, implemented seminit() to initialize the semaphore table.

3. *Semaphore Table Management:*

   - Used semalloc() to invalidate an entry and semalloc() to locate an unused location in the semaphore table.
   - To handle possible race situations in the semaphore table, concurrency control with spinlocks was employed.

4. *Sys_sem_init(), sys_sem_destroy(), sys_sem_wait(), and sys_sem_post() implementation:*

   - Consulted the XV6 textbook's sections 7.5 and 7.6 for instructions on how to implement semaphores.
   - Sys_sem_init() was implemented. It returned the index of the semaphore after initializing it with the specified count.
   - Sys_sem_destroy() was implemented: The specified semaphore entry was deallocated.

- Sys_sem_wait() was implemented. This function decremented the number of semaphores and blocked if needed until the count stopped being negative.
- Sys_sem_post() was implemented. This increased the number of semaphores and awoke any processes that were awaiting a semaphores.

5. sem_wait() and sem_post():

- Process blocking and waking were handled using the sleep() and wakeup() kernel functions.
- If the semaphore count was zero in sys_sem_wait(), indicating that a wait was necessary, then sleep() was called.
- To wake up any processes that were waiting on the semaphore, wakeup() was called in sys_sem_post().

6. Difficulties:

- Spinlocks had to be used carefully to manage concurrency in the semaphore table in order to prevent race situations. Effective use of the acquire(), release(), and initlock() functions helped to lessen this.

- It was essential to access the user's sem_t value by appropriately utilizing copyout() in sys_sem_init() and copyin() in sys_sem_wait(), sys_sem_post(), and sys_sem_destroy(). supervised appropriate validation and error management throughout data transfers.

- It took great care to use spinlocks and the sleep() function in sys_sem_wait() to ensure correct synchronization and prevent race conditions. This was accomplished by using a methodical locking and unlocking procedure.

7. Summary:

- The xv6 system calls for sem_init(), sem_wait(), sem_post(), and sem_destroy() have been implemented successfully.

- Used spinlocks to manage concurrency in the semaphore table. Employed sleep() and wakeup() to ensure effective synchronization of processes.

- Overcame difficulties with synchronization, concurrency, and user-kernel data transfer when implementing the xv6 operating system's semaphore functionality.

Task 4. Test cases.

Unexpected technical difficulties with the code hosted on an old version of GitHub made it difficult to create the intended test cases for the semaphore implementation of the xv6

operating system. Despite these setbacks, there was a plan in place, and the following were the intended testing scenarios:

- Positive Test Cases: Verify that in typical scenarios, sem_init(), sem_wait(), sem_post(), and sem_destroy() function as intended.
- Boundary Test Cases: Test using the most semaphores that the system is capable of allowing.
- Error Handling Test Cases: Test error conditions, like invalid counts or semaphore indices.
- Concurrency Test Cases: Model situations in which several processes communicate with semaphores at the same time.

Kernel bug with our implementation.

A memory leak may happen if a user program doesn't call sem_destroy() to deallocate semaphores because the OS wouldn't release them appropriately.
Could be fixed by providing a kernel mechanism to deallocate a process's associated semaphores automatically when the process ends. This could be completed in a termination procedure similar to the exit() function.

Summary:

- Setting up and maintaining an operating system's semaphores.
- Managing synchronization and concurrency while using semaphore operations.
The appropriate application of kernel mechanisms and data structures, including spinlocks.
- The significance of validating and handling errors during system calls.

```
31 int sem_init(void*, int, int);
32 int sem_destroy(void*);
33 int sem_wait(void*);
34 int sem_post(void*);
35
```

```
$U/_prodcons1\
$U/_prodcons2\
$U/_prodcons3\
$U/_prodcons-sem\
```

```
#define MAXPATH      128   // maximum file path name
#define MAX_MMR      10    // maximum number of memory
#define NSEM         100 // maximum open semaphores per system
```

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

```
[SYS_munmap]    sys_munmap,
[SYS_sem_init] sys_sem_init,
[SYS_sem_destroy] sys_sem_destroy,
[SYS_sem_wait] sys_sem_wait,
[SYS_sem_post] sys_sem_post,
};
```

```
//  HW 6
void            seminit(void);
int                 semalloc(void);
void            sedealloc(int);
```

```
9
10 typedef uint64 pde_t;
11 typedef int sem_t;
```

```
1 #define NSEM 100
2 // Mutual exclusion lock.
3 struct spinlock
4 {
5   uint locked; // Is the lock held?
6
7   // For debugging:
8   char *name;       // Name of lock.
9   struct cpu *cpu; // The cpu holding the lock.
10 };
11
12 // Counting semaphore
13 struct semaphore {struct spinlock lock;       // semaphore lock
14 int    count;                          // semaphore value
15 int    valid;// 1 if this entry is in use
16 };
17 // OS semaphore tabletype
18 struct semtab {
19   struct spinlock lock;
20   struct semaphore sem[NSEM];
21   };
22   extern struct semtab semtable;
```

```
120 int sys_sem_init(void) {
121     uint64 s;
122     int index, value, pshared;
123
124     if (argaddr(0, &s) < 0 || argint(1, &pshared) < 0 || arg
125         return -1;
126     }
127
128     if (pshared == 0) {
129         return -1;
130     }
131
132     index = semalloc();
133     semtable.sem[index].count = value;
134
135     if (copyout(myproc()->pagetable, s, (char*)&index, sizeof(index)) < 0) {
136         return -1;
137     }
138
139     return 0;
140 }
141
142 int sys_sem_destroy(void) {
143     uint64 s;
144     int addr;
145
146     if (argaddr(0, &s) < 0) {
147         return -1;
148     }
```

```
163 int sys_sem_wait(void) {
164     uint64 s;
165     int addr;
166
167     if (argaddr(0, &s) < 0 || copyin(myproc()->pagetable, (char*)&addr, s, sizeof(int)) < 0) {
168         return -1;
169     }
170
171     acquire(&semtable.sem[addr].lock);
172
173     while (semtable.sem[addr].count == 0) {
174         sleep((void*)&semtable.sem[addr], &semtable.sem[addr].lock);
175     }
176
177     semtable.sem[addr].count--;
178     release(&semtable.sem[addr].lock);
179
180     return 0;
181 }
182
183 int sys_sem_post(void) {
184     uint64 s;
185     int addr;
186
187     if (argaddr(0, &s) < 0 || copyin(myproc()->pagetable, (char*)&addr, s, sizeof(int)) < 0) {
188         return -1;
189     }
190
191     acquire(&semtable.sem[addr].lock);
192
193     semtable.sem[addr].count++;
194     wakeup((void*)&semtable.sem[addr]);
195
196     release(&semtable.sem[addr].lock);
197
198     return 0;
199 }
```

```
22 #define SYS_close  21
23 #define SYS_getprocs  22
24 #define SYS_freepmem 23
25 #define SYS_mmap 24
26 #define SYS_munmap 25
27 #define SYS_sem_init 26
28 #define SYS_sem_destroy 27
29 #define SYS_sem_wait 28
30 #define SYS_sem_post 29
```

```
31    $K/virtio_disk.o \
32    $K/semaphore.o
33
34 # riscv64-unknown-elf- or
```

```c
#include "types.h"
#include "riscv.h"
#include "param.h"
#include "defs.h"
#include "spinlock.h"
#define NSEM 100

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);
}
```