CS 1550

Week 4 – Project 1 Discussion

Teaching Assistant

Xiaoyu (Veronica) Liang

CS 1550 – Project 1

- **Due**: Friday, September 27, 2019 @11:59pm
- **Submission**:
  - GradeScope (link is available through courseweb)
  - Unlimited attempts until the deadline. It takes about two minutes to grade your solution.
  - Files:
    - The three well-commented files: **sys.c**, **syscall_table.S** and **unitstd.h**
    - A header file, named **sem.h**, that contains the declaration of your struct **cs1550_sem**.
  - Test using provided test files.

Synchronization

- Each process operates sequentially
- All is fine until processes want to share data
  - Exchange data between multiple processes
  - Allow processes to navigate *critical regions*
  - Maintain proper sequencing of actions in multiple processes
- These issues apply to threads as well
- Semaphores is a protected integer variable that can facilitate and restrict access to shared sources in a multi-processing environment.

Semaphore

- S – Integer (non-negative value at initialization)
- Q – Queue of processes/threads (empty at initialization)
- Two most common kinds of semaphores
  - Counting semaphores
    - Represent multiple resources
  - Binary semaphores
    - Represent two possible states (1 or 0 locked or unlocked)

Semaphore – two basic operations

- down() / wait()
  - Decrements S
  - If S is now negative, the current process is blocked and placed in Q
- up() / signal()
  - Increments S
  - If after the increment, S is still <= 0, that means there is still some blocked process in the queue. One of them should be dequeued and becomes unblocked.

Semaphore – basic mutual exclusion

Shared variables

```
Semaphore mutex;
```

Code for process P_i

```
while (1) {
  down(mutex);
  // critical section
  up(mutex);
  // remainder of code
}
```

Semaphore – pseudo code

```
class Semaphore {
  int value;
  ProcessList pl;
  void down ();
  void up ();
};
```

Semaphore code

```
Semaphore::down ()
{
  value -= 1;
  if (value < 0) {
    // add this process to pl
    Sleep ();
  }
}
Semaphore::up () {
Process P;
  value += 1;
  if (value <= 0) {
    // remove a process P
    // from pl
    Wakeup (P);
  }
}
```

Project 1 – Discussion

- Declare a simple struct that contains an integer value and a queue of processes:

```
struct cs1550_sem
{
    int value;
    //Some queue of your devising
};
```

- Make two new system calls that each has the following signatures:

```
asmlinkage long sys_cs1550_down(struct cs1550_sem *sem)
asmlinkage long sys_cs1550_up(struct cs1550_sem *sem)
```

Project 1 - Discussion

```
asmlinkage long sys_cs1550_down(struct cs1550_sem *sem)
```

- Here the process can sleep.
- Mark the task as not ready (but can be awoken by signals)
- set the current stat as “TASK_INTERRUPTIBLE”
  set_current_state(TASK_INTERRUPTIBLE);
- Invoke the scheduler to pick a ready task.
  schedule();

```
asmlinkage long sys_cs1550_up(struct cs1550_sem *sem)
```

- wake_up_process(**sleeping_task**);

Struct that represents a process put to sleep by the **down() method**

Project 1 - Discussion

- The semaphores need to be implemented as part of the kernel
- We need to do our increment or decrement and the following check on it **atomically**
  - We can use spin locks for that

- Create a spinlock with a provided macro:
  DEFINE_SPINLOCK(sem_lock);

- We can then surround our critical regions with the following:
  spin_lock(&sem_lock);
  // critical region
  spin_unlock(&sem_lock);

Project 1 - Tips

- Using **kmalloc** to allocate memory
  - Synopsis: void * kmalloc (size_t size, gfp_t flags);
  - https://www.kernel.org/doc/htmldocs/kernel-api/API-kmalloc.html

- printk(), you may want to use for printing out debugging messages from the kernel.

- In general, you can use some library standard C functions, but not all. If they do an OS call, they may not work.

Project 1 – Building and running test programs

```
gcc -m32 -o trafficsim -I /u/OSLab/USERNAME/linux-2.6.23.1/include/ trafficsim.c
```

Tell gcc to look for the new include files

Cannot run our test program on thoth.cs.pitt.edu

Test the program under QEMU

- Installed the modified kernel
- Copy the test program to QEMU
- Then just run it

Project 1 – Files for submission

- Syscalls you will modify the files
  - Actual implementation
    - linux-2.6.23.1/kernel/**sys.c**
  - Syscall Number map
    - linux-2.6.23.1/arch/i386/kernel/**syscall_table.S**
  - Exposes syscall number to C programs
    - linux-2.6.23.1/include/asm/**unistd.h**
- A header file named **sem.h**
  - All required declarations into the file.
  - Should be in the same folder as the test case file when compiling.

CS 1550

Week 4 – Project 1 Discussion

Teaching Assistant

Xiaoyu (Veronica) Liang