Discussion 3/1/19

The Problem

- A closed lane and a flag person is directing traffic.
- One lane closed of a two-lane road, with traffic coming from the North and South. Each car takes 2 seconds to go through the construction area.
- Traffic on the road comes in bursts. When a car arrives, there is an 80% chance another car is following it. No car means a 20 second delay.
- During the times when no cars are at either end, the flag person will fall asleep, requiring the first car that arrives to blow their horn.
- A car arrives at either end, the flag person allows traffic from that side to flow. No more cars or 10 cars queued on the opposing side means switch flow of traffic.

The Problem

- Reconstruct the situation in a program! (Using fork()!)

The Problem

- Treat the road as two queues and have a producer for each direction putting cars into the queues at the appropriately.
- Have a consumer (flagperson) that allows cars from one direction to pass through the work area as described.
- To get an 80% chance of something, you can generate a random number modulo 10, and see if it is less than 8. It's like flipping an unfair coin.
- Use the syscall nanosleep() or sleep() to pause your processes
- Make sure that your output shows all of the necessary events. You can sequentially number each car and say the direction it is heading. Display when the flagperson goes to sleep and is woken back up.

Sleeping

- You can get the current process's task_struct by accessing the global variable current.

```
// To sleep process
set_current_state(TASK_INTERRUPTIBLE);
schedule();

// Wake up
struct task_struct * sleeping_task = ..;
wake_up_process(sleeping_task);
```

Shared Memory

- We can ask for N bytes of RAM from the OS directly by using mmap():

```
void *ptr = mmap(NULL, N, PROT_READ|PROT_WRITE,
MAP_SHARED|MAP_ANONYMOUS, 0, 0);
```

- MMapped region works in children process from fork()

Semaphore

- There are two halves of implementation, the syscalls themselves, and the trafficsim program.
- kmalloc() and kfree() for allocating memory

```
Struct csc452_semv {
    int value;
    // queue
}

asmlinkage long sys_csc452_down(struct csc452_sem *sem)
asmlinkage long sys_csc452_up(struct csc452_sem *sem)
```

Atomicity - Mutex for Locking in Semaphore

```
// obtain lock

DEFINE_MUTEX(sem_lock);



mutex_lock(&sem_lock);
mutex_unlock(&sem_lock);
```

Environment

- In this project, you will have your normal OS (from the last assignment) and one with a modified kernel.

asmlinkage long sys_csc452_down(struct csc452_sem *sem)
asmlinkage long sys_csc452_up(struct csc452_sem *sem)

Adding System Calls

- In this project, you will have your normal OS (from the last assignment) and one with a modified kernel. To call in trafficsim, use syscall(). (Make a wrapper!)

Adding System Calls

- Modify the following:

1. **linux-4.20.10/kernel/sys.c** (System call definitions! Use SYSCALL_DEFINEn())
2. **include/linux/syscalls.h** (Prototypes and struct definition)
3. **include/uapi/asm-generic/unistd.h** (Syscall #'s for generic system, use 294 and 295)
4. **arch/x86/entry/syscalls/syscall_64.tbl** (Syscall #'s for 64bit x86 system)
   a. `335 common csc452_down sys_csc452_down`
   b. `336 common csc452_up sys_csc452_up`
5. **arch/x86/entry/syscalls/syscall_32.tbl** (Syscall #'s for 32bit x86)
   a. `387 i386 csc452_down sys_csc452_down`
   b. **388 i386 csc452_up sys_csc452_up**

Adding System Calls

- Once you are done modifying, rebuild the kernel. This may take awhile because you changed header files.
- In the future when you only modify sys.c, it will compile much faster.