# Concurrency: Little Book Of Semaphores

## CS 537: Introduction to Operating Systems

Louis Oliphant

University of Wisconsin - Madison

Fall 2024

# Little Book Of Semaphores

It is common to have questions on the concurrency exam where a concurrency problem is presented and you think about possible solutions – might it cause deadlock, does it enforce atomicity, does it address and solve the problem, what does a sample trace through a possible solution look like?

The Little Book of Semaphores contains problems like this, along with hints on how to solve them and then solutions.

# Problem 1 – Rendezvous Problem

Puzzle: Given the code

**Thread A**

```
statement a1
statement a2
```

**Thread B**

```
statement b1
statement b2
```

We want to guarantee that `a1` happens before `b2` and `b1` happens before `a2`.

# Problem 2 – Barrier

A limitation of **Rendezvous** is that it only works with two threads. Generalize the rendezvous solution so that every thread should run the following code

```
rendezvous
criticial point
```

The synchronization requirement is that no thread executes `criticial point` until after all threads have executed `rendezvous`. There are `n` threads. When the first `n-1` threads arrive they should block until the `n`th thread arrives, at which point all the threads may proceed. (quiz problem today)

# Problem 3 – Reusable Barrier

Often a set of cooperating threads will perform a series of steps in a loop and synchronize at a **Barrier** after each step. For this application we need a reusable barrier that locks itself after all the threads have passed through. Rewrite the barrier solution so that after all threads have passed through, the turnstile is locked again so that it can be reused.

# Problem 1 – Rendezvous Possible Solution

Here is a possible solution using mutex locks and condition variables. Does it work? Could deadlock occur? Can you refactor using semaphores? Does it scale to a Barrier or a Reusable Barrier?

```
num_waiting = 0;
pthread_mutex_t lock;
pthread_cond_t cv;

void rendezvous() {
    Pthread_mutex_lock(&lock);
    while (num_waiting == 0) {
        num_waiting += 1;
        Pthread_cond_wait(&cv,&lock);
    }
    Pthread_cond_signal(&cv);
    Pthread_mutex_unlock(&lock);
}
```