Assignment 5

Problem 5.1: readers / writers problem

a) This fails because the if (readcount = 0) and the up(writer) are executed without holding the mutex. This makes the readcount change inbetween the up(mutex) and if clause and writers and readers can interleave incorrectly

b) The problem is that the last reader releases the mutex before signaling the writer so this gap allows new readers to begin and block the down(writer) and get in the way of it which causes the writer to starve.

c) The problem is that the writer tries to acquire mutex after down(writer), this causes a deadlock where if the writer has already done down(writer) and then tries to do down(mutex) both sides wait for each other.

Problem 5.2: good tasting coffee

```
semaphore s1 = 0;

semaphore s2 = 0;

semaphore s3 = 0;


void a(void) {

printf("the ");

up(s1)

down(s2)

printf("tastes ");

up(s3)
```

```
down(s1)
printf("good ");
up(s2)
}

void b(void) {
down(s1)
printf("coffee ");
up(s2)
down(s2)
printf("here");
}

void c(void) {
down(s3)
printf("so ");
up(s1)
}
```

Problem 5.3: bounded buffer with posix semaphores

a) Code in C

b) Using atomic variables like atomic_fetch_add in the produce function ensures that each produced item gets a unique integer ID which produces a thread-safe, unique sequence number without the use of a mutex. Without the use of atomic variables the shared unsigned int cnt can be interleaved causing lost increments and duplicated sequence numbers.

c) The checkerboard is like a marker to validate that the write to the data slot has actually been published to other threads in a safe way. This helps detect subtle reorderings or wrong reads in concurrent scenarios. More concretely, the checkerboard helps detect a situation where a consumer might read a slot's data that hasn't been fully written or that got overwritten in a race.