COM S 352 Homework 5

Alec Meyer

March 12, 2021

Question 1

Two bidders could call the bid function concurrently, which results in the highestBid variable to be shared. The race condition is that that two users could be on two separate threads and bid the values 10 and 20. It is possible for the 10 bid to be used if highestBid is shared between the two.

```
void bid(double amount) {
    acquire(highestbid)
    if (amount > highestBid)
    highestBid = amount;
}
release();
```

Question 2

```
typedef struct {
    int available;
} lock;

void acquire(lock *mutex) {
    while (mutex->available == 1);
        mutex->available = 1;
}

void release(lock *mutex) {
    mutex->available = 0;
}

void initialization_code(lock *mutex) {
    mutex->available = 0;
}
```

Question 3

1: A spinlock would be a better locking mechanism because it doesn't require a context switch and can avoid busy waiting. This makes them more preferable for short durations.
2: A mutex lock would be preferable for a long duration as it allows other processes to run while the lock waits.
3: A mutex lock would be prefered for this scenario as it doesn't need to be spinning while waitng for the thread to wake up.

Question 4

a. The race condition appears in the variable *number_of_processes* as the method *allocate_process()* and *release_process()* modify its value.

b.

```
#define MAX_PROCESSES 255
int number_of_processes = 0;
/* the implementation of fork() calls this function */
int allocate_process() {
    int new_pid;
    acquire(mutex);/////////////////////////////
    if (number_of_processes == MAX_PROCESSES) {
        release(mutex);/////////////////////////
        return -1;
    } else {
        /* allocate necessary process resources */
        ++number_of_processes;
        release(mutex);/////////////////////////
        return new_pid;
     }
}
/* the implementation of exit() calls this function */
void release_process() {
    /* release process resources */
    acquire(mutex);/////////////////////////////
    --number_of_processes;
    release(mutex);/////////////////////////////
}
```

Question 5

For monitors, a *signal()* call will resume only one process if there is a waiting process. However, when there is not a waiting process the call to *signal()* will not be saved.