CSSE 132 – Introduction to Systems Programming
Rose-Hulman Institute of Technology

Worksheet 26: Race Conditions and Mutual Exclusion

Name (Print):_____ RHIT Username:_____

1. Read the following program before running it.

```c
#include <stdio.h>
#include <pthread.h>

#define COUNT 1000000

int counter = 0;

void *increment(void *arg) {
    for (int i = 0; i < COUNT; i++) {
        counter++;
    }
    return NULL;
}

int main(void) {
    pthread_t t1, t2;

    pthread_create(&t1, NULL, increment, NULL);
    pthread_create(&t2, NULL, increment, NULL);

    pthread_join(t1, NULL);
    pthread_join(t2, NULL);

    printf("Final counter: %d\n", counter);
    return 0;
}
```

Before running the program: What final counter value do you predict?

Predicted value: _____

Observed value: _____

2. A **race condition** occurs when the behavior of a program depends on the _____ or _____ of operations performed by multiple threads.

   In the counter example, two threads may read the same old value and _____ each other's updates.

3. A **critical section** is a part of a program that accesses _____ data or resources and must be protected from _____ access by other threads.

4. An event has only **one ticket left**. Two threads run `buy_ticket()` at about the same time. The code below has *no synchronization yet*.

```
int tickets_left = 1;

void buy_ticket(void) {
    if (tickets_left > 0) {
        tickets_left--;
        printf("Ticket reserved!\n");
    } else {
        printf("Sold out!\n");
    }
}
```

(a) Circle the lines that should be protected together as a critical section.

(b) What could go wrong if both threads run those lines without synchronization?

5. Two threads record purchases for the same store. Both call `record_sale()` to update the shared total. Fill in the **three blanks**. Use a mutex named `lock`.

```
#include <pthread.h>

int total_sales = 0;
pthread_mutex_t __________ = PTHREAD_MUTEX_INITIALIZER;

void record_sale(int dollars) {
    pthread_mutex_lock(__________);
    total_sales += dollars;
    pthread_mutex_unlock(__________);
}
```

(a) Circle the critical section in this code.

(b) Why does `pthread_mutex_lock()` need the `&` symbol before the mutex name?

6. The following program uses a mutex, but has a mistake. Read the code, predict which messages will print, and then explain what happens.

```c
#include <stdio.h>
#include <pthread.h>

pthread_mutex_t lock = PTHREAD_MUTEX_INITIALIZER;

int main(void) {
    printf("Step 1: Trying to acquire lock...\n");
    pthread_mutex_lock(&lock);
    printf("Step 2: Lock acquired!\n");

    printf("Step 3: Trying to acquire lock AGAIN...\n");
    pthread_mutex_lock(&lock);

    printf("Step 4: Made it past the second lock!\n");
    pthread_mutex_unlock(&lock);
    pthread_mutex_unlock(&lock);
    return 0;
}
```

(a) Which numbered messages do you expect to see when this program runs on our Linux VM? Will the program finish normally?

(b) Circle the line that causes the problem. Why does the program get stuck? Use the term *deadlock* in your explanation.

(c) How could you fix the code so it finishes normally?

Optional live demo: Compile with `gcc -O0 -pthread deadlock.c -o deadlock`, run `./deadlock`, and press `Ctrl+C` to stop it if it hangs.