Homework 4

Ally Smith (CSCI442)

March 4, 2022

Question 1.

Many current language specifications, such as for C and C++, are inadequate for multi-threaded programs. This can have an impact on compilers and the correctness of code, as this problem illustrates. Consider the following declarations and function definition:

```
int global_positives = 0;
typedef struct list {
    struct list *next;
    double val;
} * list;

void count_positives(list l) {
    list p;
    for (p = l; p; p = p -> next)
    if (p -> val > 0.0)
    ++global_positives;
}
```

Now consider the case in which threads A executes `count_positives(<list containing only negative values>)` while thread B executes `++global_positives`.

(a) What does the function do?

The function loops through the provided linked list, and counts the number of positive elements in the list.

(b) The C language only addresses single-threaded execution. Does the use of two parallel threads create any problems or potential problems?

Once `count_positives` finishes running, `global_positives` may have a value other than the number of positive numbers in the linked list, causing unexpected behavior.

Question 2.

Some existing optimizing compilers (including gcc, which tends to be relatively conservative) will 'optimize' count_positives to something similar to the following:

```
void count_positives(list l) {
    list p;
    register int r;
    r = global_positives;
    for (p = l; p; p = p -> next)
        if (p -> val > 0.0) ++r;
    global_positives = r;
}
```

What problem or potential problems occurs with this compiled version of the program if threads A and B are executed concurrently?

If thread B starts and finishes after thread A, then the `global_positives` will be incremented. However, if thread A starts before thread B, then thread B will not be able to increment since `global_positives` since it will be reassigned from the register.

Question 3.

In the discussion of ULTs versus KLTs, it was pointed out that a disadvantage of ULTs is that when a ULT executes a system call, not only is that thread blocked, but also all of the threads within the process are blocked. Why is that so?

This happens because, a lot of the time, system calls are blocking calls, which means when the OS gets a syscall, it must block the whole process since it doesn't recognize user-level threads.

Question 4.

Consider an environment in which there is a one-to-one mapping between user-level threads and kernel-level threads that allows one or more threads within a process to issue blocking system calls while other threads continue to run. Explain why this model can make multi-threaded programs run faster than their single-threaded counterparts on a uni-processor computer.

This would allow for multi-threaded programs to make blocking syscalls that don't block their entire process, and instead just block that single thread. This would allow for other threads to run as the scheduler sees fit.