Bank monitor invariants

Monitor name	*Invariant*
Bank1	balance >= 0
Bank2	balance >= 0 && proper_dequeuing*

*imply that the any withdrawals remaining are queued in the same order

GCD Argumentation

I would like to reason that the program does, given a weakly fair scheduler, always terminate. Considering how this particular implementation concurrently checks if x>y or x<y before doing any writing to variables, it easy to see that there will never be any conflicts because the conditions are already mutually exclusive: Either one can true, but it's logically invalid for both of them to be true at the same time(e.g it is obviously a fallacy to say 3>2 and 2>3). Further I noted that since both of the concurrent statements contains only one critical reference, the program as a whole never violates the AMO property. Combined the mutually exclusive running, this should guarantee write-safety for the duration of the programs lifespan.
The loop variant, which is given as {while x != y}, will inevitably become true as both x and y converge to g=gcd(x,y) and thus terminating the program. Also it is worth mentioning that that the precondition x > 0 && y > 0 is necessary to guarantee inevitable termination.

Question 2.3

Annotations

```
(n > 0, m>0)
x=n, y=m
I : {gcd(x,y) = gcd(n,m)}
B : (n!=m)
!B: (n==m)
int gcd(int n, int m) {
        {I} while(B) {
                CO(
                        {I} if (x > y) {x = x - y;} {I}
                        {I} if (x < y) {y = y - x;} {I}
                OC
        }
        {I ∧ !B)
        return n;
} {I}
```

Reasoning for consecutive statements with pre and post-conditions

Since the invariant {I} needs to hold throughout the loop, we can conclude that the invariant must be valid before each statement inside the loop, otherwise the invariant is in fact not preserved and not truly an invariant. As such it must also be true after every statement in the loop to persist. Therefore every consecutive statement needs to have the loop invariant as its pre and post-condition, and by extension every pair of statements must also oblige.

Reasoning for hoare triples

Given the same values for {I}, B and !B as above:

1.

{I} while (B) S; {I ∧ !B), written out this becomes:

{gcd(x,y) == gcd(n,m)} while (n!=m) S; {gcd(x,y) = gcd(n,m) ∧ n==m)

Since n == m we have the following:

gcd(n,m) == gcd(n,n) == gcd(x,x) = x

This makes the postcondition look like this:

gcd(x,x) =x ∧ x == y, which is the desired outcome as specified by the assignment

2.

{I} if (n > m) {n = n - m;} {I}, written out this becomes:

{gcd(x,y) == gcd(n,m)} if (n > m) {n = n-m;} {gcd(x,y) == gcd(n,m)}

We note that since gcd(x,y) == gcd(n,m) and x > y, we have:

gcd(x,y) == gcd(x-y, y)

We then apply the assignment axiom to get:

{gcd(x,y) == gcd(x-y, y)} x = x -y; {gcd(x,y) == gcd(n,m)}

Finally, by the rule of consequence, we have

{gcd(x,y) == gcd(x-y, y)} x = x -y; {gcd(x,y) == gcd(x-x, y)}

3.

{I} if (m > n {m = m - n;} {I}

Same as the above but with x and y opposite