Name: Ian Ruvuto Kampwanyi

Assignment: 03

Problem 1a)

```
a) void reader()
   {
       down(&mutex);
       readcount = readcount + 1;
       if (readcount == 1) down(&writer);
       up(&mutex);
       read_shared_object(&data);
       down(&mutex);
       readcount = readcount - 1;
       up(&mutex);
       if (readcount == 0) up(&writer);
   }
```

```
void writer()
{
    down(&writer);
    write_shared_object(&data);
    up(&writer);
}
```

- Here the problem is identified in the selected lines of code
- For this reader / writer problem, if one reader is left and desires to finally leave.
- They will down the mutex signaling that no other reader can change the variable readcount.
- The reader will now decrement the variable readcount to 0 showing no reader is reading the content
- The reader will now up(&mutex) giving the rights to the variable readcount.
- if a new reader comes (since they can concurrently) before the uncritical section line { if (readcount == 0) up(&writer); } They will down the mutex update readcount to 1;
- Since readcount it now 1 the new reader may down(&mutex) however, this reader will not proceed since the previous reader had not up(&writer) and will not be able to since the shared readcount is now 1.
- So the last line of code will not run because of the semaphore being 1 and not 0 in order to up(&mutex) blocking the entire process of the new reader. In other words, a deadlock will happen, and no process can run.
- A solution to this would be exchanging the selected lines of code. The 9^{th} line of code should be at the 8^{th} line of code and the 8^{th} line of code should be at the 9^{th} line of code.

1b)

```
b) void reader()
   {
       down(&mutex);
       readcount = readcount + 1;
       if (readcount == 1) down(&writer);
       up(&mutex);
       read_shared_object(&data);
       down(&mutex);
       readcount = readcount - 1;
       if (readcount == 0) {
           up(&mutex);
           up(&writer);
       } else {
           up(&mutex);
       }
   }
```

```
void writer()
{
    down(&writer);
    write_shared_object(&data);
    up(&writer);
}
```

- Here I believe there is no problem, however switching up(mutex) and down(writer) would reduce starvation. Since readers would keep reading even though the writer was que ued for a long time.

1c)

```
c) void reader()
   {
       down(&mutex);
       readcount = readcount + 1;
       if (readcount == 1) down(&writer);
       up(&mutex);
       read_shared_object(&data);
       down(&mutex);
       readcount = readcount - 1;
       if (readcount == 0) up(&writer);
       up(&mutex);
   }
```

```
void writer()
{
    down(&writer);
    down(&mutex);
    write_shared_object(&data);
    up(&mutex);
    up(&writer);
}
```

- Here the problem is identified in the highlighted lines of code.
- Since Readers and a writer will try to access data concurrently, assume the writer calls down(&writer) line 1 and at the same time the 1st reader gets in and calls down(&mutex) and updates readcount to 1.
- In this situation the writer will not proceed and will get blocked by calling down(&mutex).
- Now the reader performs the if condition and call down(&writer). However, the reader is blocked too since the down(&mutex) was already called by a writer.
- This will leave both the reader and writer to wait indefinitely for the shared variable mutex and writer leaving their state to be a Deadlock.