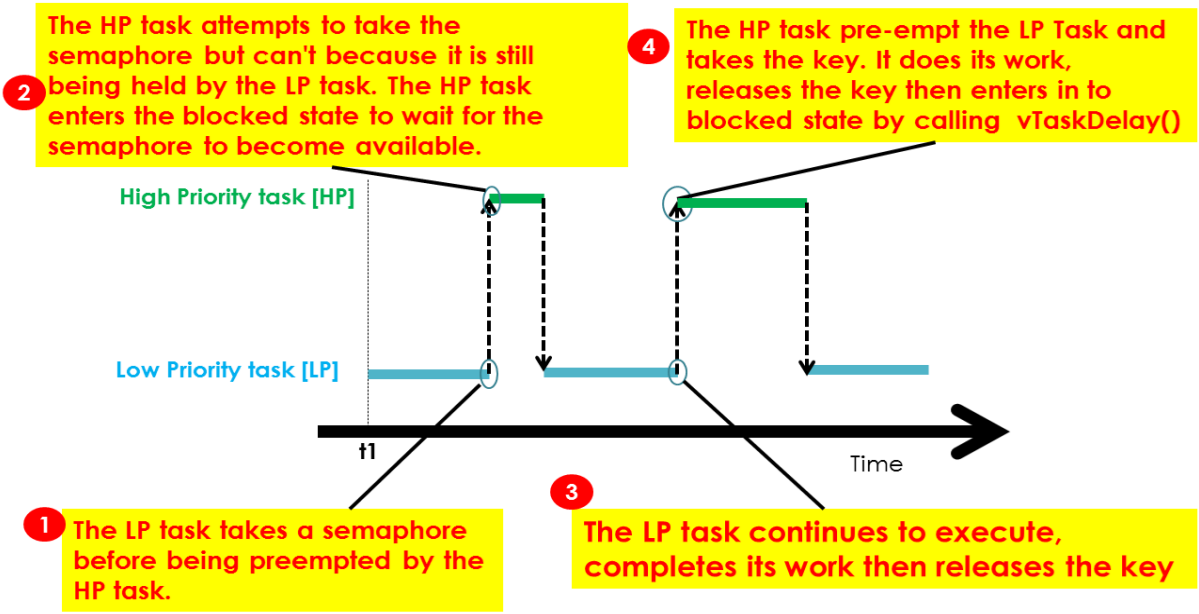
Issue with Binary Semaphore when used for Mutual Exclusion.

In the previous lectures we wrote a code to implement the mutual exclusion using binary semaphore and we tested on our Arduino hardware.

As you know we actually used 2 tasks, **Task-1** and **Task-2**, both are trying to write some strings of data on to the output screen. you can say, that is standard output. The standard output is actually in our case is serial port.

We also saw that there is no corruption in the data being outputted by both the tasks.

Let's analyse execution sequence of the previous code exercise with the below diagram.



Initially there are 2 tasks which are in ready state.

The **Task-1's** priority is 1 and **Task-2's** priority is 2.

That means **Task-2** is having highest priority than **Task-1**.

When you start the scheduler, **Task-2** will be moved to running state.

That means it will be allowed to run first.

What **Task-2** does is, it simply acquires the semaphore, prints the string and then releases the semaphore, then it will call **vTaskDelay()** function and gets blocked for some amount of time.

Since the **Task-2** is blocked, now we have only one task in the ready state that is **Task-1**, so immediately **Task-1** will be yielded to run on the CPU.

When the **Task-1** is running, it will acquire the semaphore and before printing the string, it may be pre-empted by the **Task-2**, because **Task-2** blocking period may be over.

So, **Task-2** comes in to picture, it tries to get the semaphore, but since **Task-1** is already holding the semaphore, so **Task-2** will be blocked until the semaphore is available.

That again makes **Task-1** to continue its execution, so it will print the string, releases the semaphore, and once the shampooer is released, **Task-2** will come out of its blocking state immediately and it will again kick the task out of the ring to claim the CPU.

So, **Task-2** continues, it again acquires the key and prints the data, and releases THE key and again gets blocked for some time due to **vTaskDelay()** function.

That's fine !!

No what's the problem with this?

Why people say do not use binary semaphore for mutual exclusion. ??

Let's see,

If you carefully look at this timing diagram, there is a chance for priority inversion!

Here, **Task-2** which is higher priority task gets blocked due to non-availability of semaphore and it is waiting for **Task-1** to release the semaphore. Right?

That means, by this design, you made a higher priority task dependent on the lower Priority task. That is a bad.

The low priority task should never decide when the higher priority task must run.

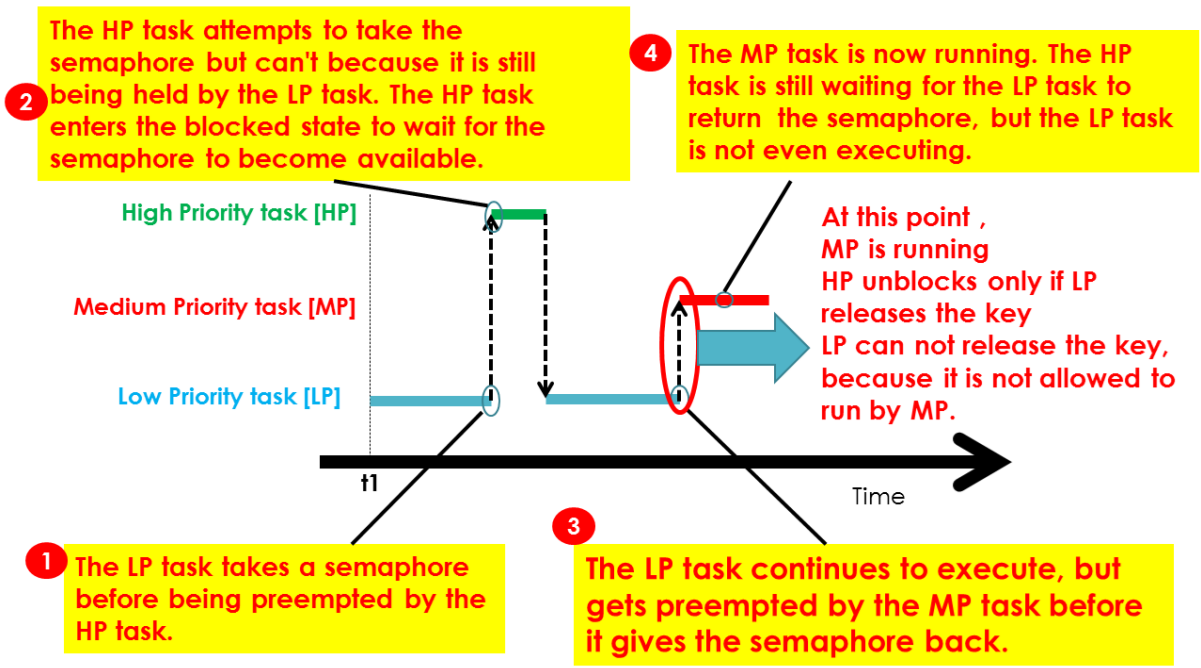
It's like you made a king dependent on a servant to take some action.

So until lower priority task releases the semaphore, the higher priority task never runs. Isn't it?
(Remember this is like your car and ambulance scenario i showed in the very first section of the course. You are blocking the way of ambulance.)

And lower priority task will release the semaphore only when it gets allowed to run isn't it ?

So, what if there is another medium task appears and it occupies the processor which will not allow lower Priority task to run?

Look at this diagram,



Here , higher priority task is blocked and waiting for semaphore to gets released by lower priority task.

But lower priority task is not allowed to run by the medium priority task until it finishes and exits.

As a result your lower Priority task may be delayed indefinitely and the higher priority task will remain in blocked state for indefinite time . **This is exactly the scenario of Priority inversion.**

That means, even though the **Task-2** has the highest priority the situation made it behave like lower Priority.

Remember that, until medium task appears everything is OK, but the moment medium priority task appears, then it may lead to priority inversion. So, as your application contains more tasks of different priority levels, then you have to test you application against priority inversion. There could be chances of higher priority tasks getting delayed unnecessarily.

So, are there any ways to solve or reduce the priority inversion problem ?

Simple answer is don't use binary semaphore for mutual exclusion when your system has many tasks of different Priorities. Use MUTEX instead.

MUTEX has inbuilt intelligence to minimize the priority inversion effect and this is the biggest difference between MUTEX and Binary semaphore.

Yes!! MUTEX is also another kernel service, which is carefully designed to take care of this problem of priority inversion.

Even though there are lots of similarities between mutex and binary semaphore, the mutex has inbuilt intelligence to reduce the problem like priority inversion.

Assignment:
Now you know that MUTEX reduces the priority inversion effect. But how? I request you to go through the MUTEX implementation of freeRTOS and publish you findings in the course discussion board. I will then verify and announce it as a educational announcement on your name.