Lesson 10: Mutexes

- ☑ 10.1 Introduction
- ☑ 10.2 Commands
- ☑ 10.3 Kconfig
- ☐ 10.4 Exercise

What is a mutex?

A mutex is a kernel object that implements a traditional reentrant mutex. A mutex allows multiple threads to safely share an associated hardware or software resource by ensuring mutually exclusive access to the resource.

Any number of mutexes can be defined (limited only by available RAM). Each mutex is referenced by its memory address.

A mutex has the following key properties:

- A lock count that indicates the number of times the mutex has been locked by the thread that has locked it. A count of zero indicates that the mutex is unlocked.
- An owning thread that identifies the thread that has locked the mutex when it is locked.

A mutex must be initialized before it can be used. This sets its lock count to zero.

A thread that needs to use a shared resource must first gain exclusive rights to access it by locking the associated mutex. If the mutex is already locked by another thread, the requesting thread may choose to wait for the mutex to be unlocked.

After locking a mutex, the thread may safely use the associated resource for as long as needed; however, it is considered good practice to hold the lock for as short a time as possible to avoid negatively impacting other threads that want to use the resource. When the thread no longer needs the resource it must unlock the mutex to allow other threads to use the resource.

Any number of threads may wait on a locked mutex simultaneously. When the mutex becomes unlocked it is then locked by the highest-priority thread that has waited the longest.

Note: Mutex objects are not designed to be used by ISRs.

Reentrant locking

A thread is permitted to lock a mutex it has already locked. This allows the thread to access the associated resource at a point in its execution when the mutex may or may not already be locked.

A mutex that is repeatedly locked by a thread must be unlocked an equal number of times before the mutex becomes fully unlocked so it can be claimed by another thread.

Priority inheritance

The thread that has locked a mutex is eligible for priority inheritance. This means the kernel will temporarily elevate the thread's priority if a higher priority thread begins waiting on the mutex. This allows the owning thread to complete its work and release the mutex more rapidly by executing at the same priority as the waiting thread. Once the mutex has been unlocked, the unlocking thread resets its priority to the level it has before locking that mutex.

Note: The CONFIG_PRIORITY_CEILING configuration option limits how high the kernel can raise a thread's priority due to priority inheritance. The default value of 0 permits unlimited elevation.

When tow or more threads wait on a mutex held by a lower priority thread, the kernel adjusts the owning thread's priority each time a thread begins waiting (or gives up waiting). When the mutex is eventually unlocked; the unlocking thread's priority correctly reverts to its original non-elevated priority.

The kernel does not fully support priority inheritance when a thread holds two or more mutexes simultaneously. This situation can result in the thread's priority not reverting to its original non-elevated priority when all mutexes have been released. It is recommended that a thread lock only a single mutex at a time

How to define a mutex?

A mutex is defined using a variable of type k_mutex. It must then be initialized by calling k_mutex_init().

The following code defines and initializes a mutex.

```
struct k_mutex my_mutex;

k_mutex_init(&my_mutex)
```

Alternatively, a mutex can be defined and initialized at compile time by calling K_MUTEX_DEFINE.

The following code has the same effect as the code segment above.

```
K_MUTEX_DEFINE(my_mutex);
```

How to use a mutex?

A mutex is locked by calling k_mutex_lock().

The following code builds on the example above, and waits indefinitely for the mutex to become available if it is already locked by another thread.

```
k_mutex_lock(&my_mutex, K_FOREVER);
```

The following code waits up to 100 milliseconds for the mutex to become available, and gives a warning if the mutex does not become available.

```
if (k_mutex_lock(&my_mutex, K_MSEC(100)) == 0)
```

```
{
    /* mutex succesfully locked */
}
else
{
    printf("Cannot lock XYZ display\n");
}
```

A mutex is unlocked by calling k_mutex_unlock().

The following code builds on the example above, and unlocks the mutex that was previously locked by the thread.

```
k_mutex_unlock(&my_mutex);
```

When to use?

Use a mutex to provide exclusive access to a resource, such as a physical device.

Command	Description
K_MUTEX_DEFINE	Statically define and initialize a mutex.
k_mutex_init	Initialize a mutex.
k_mutex_lock	Lock a mutex. This routine locks mutex. If the mutex is locked by another thread, the calling thread waits until the mutex becomes available or until a timeout occurs.
k_mutex_unlock	Unlock a mutex. This routine unlocks mutex. The mutex must already be locked by the calling thread.

Kconfig	Description
CONFIG_PRIORITY_CEILING	Priority inheritance ceiling

2 preemptive threads

1 integer value, we want to in one thread add 1 and in the other subtract one. If we don't use mutex then unpredictable results If we do it should stay at same value even though 2 different priorities threads (adn they are preemptive).

-------------------------------------THANK YOU--------------------------------