Using C++ class function in C

One way which I am going to explain is using pointers! Our task involves interaction between two different kind of sources which can easily be handled by accessing the memory. Some core C++ application developers might not like it but in Embedded domain it is useful.

I have taken Logging functionality as an example to explain it which uses UART to send data to terminal, the class design and interface can be used with different peripherals such as SD Card, Network etc (?).

I designed a Logger class which has a static object like ‘cout’ in C++ named ‘logger’ to log the events occurring in system which uses FreeRTOS utilities for synchronization and peripheral access.

There may be other ways but I have not considered them in this document, this one is easy (in my Knowledge at-least!).

Author: Hemant Sharma

Let’s get to the source and learn the methodology 😊

1. First, we need to declare the public member function which can be accessed without creating a class object.

```
static inline void write( unsigned char *p_data, unsigned short length );
Can remove inline if you do not want the function implementation in header file of class and define it in source file. (declared in <class_header.h>)
```

```
/* Write function which could be called from C file functions */
static inline void write( unsigned char *p_data, unsigned short length );
```

2. Create a private function which is used to write data in class and would be used by our static ‘write’ function as well. (Below functions are declared in <class_header.h> file)

```
virtual void _write( unsigned char *p_data, unsigned short length )
I have put ‘_’ in front to represent it as an internal function as in libraries.
```

This function will be used by two member functions:

a. `virtual Logger__& operator<<( Logger__& end(Logger__&) )` [Class public function which is used to trigger the send operation]

b. `static inline void write( unsigned char *p_data, unsigned short length )` [To be accessed using function pointer]

```
/* Functions for endl functionality,
 * Which is responsible to trigger the send operation
 * */
static Logger__& endl( Logger__& stream )
{
    return stream;
}

/* Function entry to send data to destination,
 * This can be overridden for transferring data to hardware device
 *
 * <i>Imp Note:</i> For every transfer of data buffers and parameters needs to be reset for fresh transfer
 * */
virtual Logger__& operator<<( Logger__& end(Logger__&) )
{
    /* Write data to hw driver */
    _write( (unsigned char *)buffer_, index_ );
    /* Wait for data transfer completion using Semaphore */
    if( xSemaphoreTake( xlog_event_sem, pdMS_TO_TICKS(200) ) != pdPASS )
    {
        /* If data sent confirmation is not there then take it into next cycle of transmission,
         * But if data goes beyond buffer size then buffer will be overwritten.
         * Buffer size is kept according to embedded applications and shall work fine. */
        /* for Debugging only */
        __asm("NOP");/* Oops, Data not reliably sent! */
    }
    else
    {
        memset( (void *)buffer_, '\0', ::sg_logger_buffer_size );
        index_ = 0;
    }
    /* Return object reference */
    return *this;
}
```

```
/* Can move it to source file also! */
void Logger__::write( unsigned char *p_data, unsigned short length )
{
    const void *p_object = &logger;
    ((Logger__*)p_object)->_write( p_data, length );
    /* Wait for Semaphore for this much time at-least and move on!(taken for test only) */
    xSemaphoreTake( xlog_event_sem, pdMS_TO_TICKS(200) );
}
```

```
virtual void _write( unsigned char *p_data, unsigned short length )
{
    BaseType_t mutex_ret = pdFALSE;
    /* Take Mutex */
    mutex_ret = xSemaphoreTake( xlog_tx_mutex, portMAX_DELAY );
    if( pdTRUE == mutex_ret )
    {
        /* Send data to output using HW layer */
        p_fcn_comm_( LOGGER_COMM_CHANNEL(p_channel_), p_data, length );
    }
    if( pdTRUE == mutex_ret )
    {
        /* Release Mutex */
        xSemaphoreGive( xlog_tx_mutex );
    }
}
```

Actual write to peripheral happens in this function

3. Create a function pointer which points to the static function of class object (for this example)

```
void (*p_Logger__) ( unsigned char *p_data, unsigned short length ) = Logger__::write;
```

It is a global function pointer which is compiled within extern "C" block in C++ file <class_source.cpp>.

```
extern "C"
{
    void (*p_Logger__) ( unsigned char *p_data, unsigned short length ) = Logger__::write;
    // or
//  void (*p_Logger__1) ( unsigned char *p_data, unsigned short length ) = logger.write; /* Above is better */
//  logger_fp p_Logger__ = logger.write;
}
```

(Other method to declare and use function pointer is described in source file)

All the above operations ([1,2] & 3) were done in class header and source file (<class_header.h> and <class_source.cpp>) respectively.

Now we need to include the function pointer pointing to C++ Class member function in other C source files to use it.

```
extern void (*p_Logger__)( unsigned char *, unsigned short );
```

We can include this function pointer reference in some header and include that header in source file also.

4. Time to use the C++ class member function in C source file in which function pointer reference has been included (user can choose the way).
   Create a test C source file or main file and include function pointer reference as shown in image below.

```c
extern void (*p_Logger__)( unsigned char *, unsigned short );

void dummy_freeRTOS_task( void *pvParams )
{
    (void) pvParams;
    unsigned char * p_data = (unsigned char *)"Hello World\r\n";

    for( ; ; )
    {
        p_Logger__( p_data, 13 );
        vTaskDelay(200);
    }
}
```

We are done!!

Now we can use the p_Logger__ function pointer in this file anywhere to log events or anything on UART terminal.

END

I hope you found the above data insightful and helping. You can see the source code for your reference.
Hemant Sharma
Enjoy! Happy to Share and Help 😊