CODE TIME TECHNOLOGIES

Abassi RTOS

Library Reentrance Protection

Disclaimer

Table of Contents

List of Figures

List of Tables

1 Introduction

This document explains the issues encountered with compiler libraries when used in a multi-thread / multi-core application and how to eliminate them when using Code Time RTOS. Although this document's title is about Abassi, it also covers the multi-core mAbassi and µAbassi RTOS.

2 Multi-thread & Multi-core

This section describes the problems that can be exhibited with the compiler libraries when used in a multi-thread application.

2.1 Multi-threading

The key issues with compiler libraries used in a multi-thread application are dual. The first issue is related to non-reentrant functions. An example is the standard dynamic memory management from the "C" library, i.e. `malloc()` / `free()`. If these functions aren't used with exclusive access protection, then there is a likelihood of misbehavior in a multi-thread environment. This protection is what is called "re-entrant safe" in this document.

The second issue is due to the fact all libraries use static and global variables. One such example is the `errno` variable in the "C" libraries. This also applies to library functions that to rely on static variables, alike the `stdio` devices access. In the case of static variables, protecting the function with a mutex is sufficient, but this is not the case for the global variables. Let's use `errno` as the example gain, if two tasks use concurrently (multi-core) functions that set `errno`, or a preemption happens, `errno` will be set to the value of the task that was last to perform the write. This can deliver weird result: e.g. performing a `malloc`() that would return `ENOMEM` (out of heap) and getting `EBADF` instead, indicating a bad file descriptor. The way proper protection for global variables is achieved is to assign individual global variables to each task. This protection is what is called "multi-thread safe" in this document.

2.1.1 Re-entrant safe

The library non-reentrance problem is solved by relying on mutex(es) to provide exclusive access to the non-reentrant functions. All currently compiler library supported by all Abassi ports use "hooks" to attach mutex(es) to the non-reentrant function. Some libraries use the mutex protection on

2.1.2 Multi-thread safe

This problem is properly and supported by some libraries, partially by other and not by others. The following section describes what is, and what's not supported by each one of the libraries.

3 How to

Abassi library protection is enabled through one of the following build options:

- TI's Code Composer: `OS_CCS_REENT`
- GCC Newlib: `OS_NEWLIB_REENT`
- IAR Dlib `OS_IAR_MTHREAD`
- Keil ARMCC: `OS_KEIL_REENT`

Abassi defines a token to indicate if library re-entrance protection is enabled or not. The token `OX_LIB_REENT_PROTECT` is set to zero if the library is not protected, set to non-zero if protected (either full or partial multithreading)

3.1 Set-up

The re-entrance protection is enabled, by defining and setting the build option associated to the compiler used, to a non-zero value. If the build option is set to a zero value, re-entrance protection is not used. The build option is for the compiler, except GCC Newlib that also needs to assembly support function to have the build option `OS_NEWLIB_REENT` defined and set to the same value as the one for the compiler.

When re-entrance protection is enabled, the mutex-based protection for non-reentrant functions is used on all tasks in the application. In the case of the multi-thread protection, it is possible to enable the protection for all tasks (the build option is set to a positive value) or to set the protection on a per task basis (the build option is set to a negative value). The later option is provided because the memory required to provide unique global variables for each task could be quite large (Newlib requires over 1KB per task). When the build option is set to a negative value, all newly created task don't have global variable protection; to provide protection, the protection must be enabled on the desired tasks:

Table 3-1 Setting a task to use re-entrant library

```
#inclde "Abassi.h"

TSK_t *Tsk_1                                    /* Descriptor of the task to protect */

…                                               /* First the task must be created    */
                                                /* in the suspended state            */
Tsk_1 = TSKcreate("TaskName", TskPrio, StackSize, TaskFct, 0);

TSKmultThrd(Tsk_1);                             /* Initialization of the lib context */

TSKresume(Tsk_1);                               /* The task may now be resumed       */
```

The memory needed to hold all the global variables is always allocated through Abassi's `OSalloc()` component.

The most well known modules that need to be under multithreading protection in the libraries are:

- The time structure `tm`
- `atexit()`
- stdio
- File I/O for stdin, stdout, stderr
- Rand / Rand48
- errno
- Signals
- Locale
- And a few more

If a task uses none of the above modules, then the task does not need to access the library in a multithread-safe manner, so there is no need to reserve the memory block (over 1KB for Newlib) of data memory. If a task uses one or more these modules, but it is the only task using this/these module(s), there is still no need to make the library multithread-safe for that task. Only when two or more tasks use the same modules of the library do these tasks need to access the library in a multithread-safe manner.

3.2 CCS (TI's Code Composer)

TI's Code Composer libraries provide support to make the libraries re-entrant safe, but do not have the capability to provide to make then re-entrant safe. So neither the full, nor partial multi-threading protection is available, only re-entrance protection.

3.3 GCC (RedHats' NewLiB)

Code Sourcery, Yagarto, Atollic, Altera and Xilinx's SDK all use Red Hat's Newlib `libc` library and this library can be set to be fully reentrant-safe and multithread-safe. Newlib uses two techniques to provide the multi-threading protection and this may be an issue depending on how the library has been built (with or without dynamic re-entrance protection). Most provided libraries are built without dynamic re-entrance.

The basic multi-threading internal protection mechanism of NewLib is based on the global variable `_impure_ptr`, which points to a data structure holding all the non-reentrant data (global AND function static). During each context switch, this pointer is updated to indicate the task specific non-reentrant data. This method works perfectly for applications running on a single core, but it becomes useless when an application is operating on a multi-core platform because all cores have to share the same unique pointer. When the library is built with dynamic re-entrance, the multi-thread protection become valid on a multi-core target.

NOTE: On ARM target processors that supports both Thumb and A32 instructions, it is not possible to mix the 2 instruction sets in an application when enabling / using a Newlib built with dynamic re-entrance protection.

3.3.1 OS_NEWLIB_SHARE_STDIO

A build option can be used to control the sharing of `stdin`, `stdout` and `stderr`. Depending on how a multi-thread safe Newlib has been built, the standard I/O may or may not be shared amongst all tasks. The build option `OS_NEWLIB_SHARE_STDIO` controls if new tasks are created sharing the same standard I/O or are left to the default of the Newlib set-up. When the build option `OS_NEWLIB_SHARE_STDIO` is set to a non-zero value, the standard I/Os are shared amongst all tasks. When set to a value a zero, the standard I/Os are not set-up so the sharing is what the Newlib has been built for. There is no provision to force a non-sharing of the standard I/O. If such functionality is desired, then a task standard I/O must have `stdin`, `stdout` and `stderr` opened and after the task is created, the following must be performed:

Table 3-2 Individual standard I/O

```
#include "mAbassi.h"

TSK_t *MyTask;                                  /* Task to assign unique standard I/O */
FILE  *MyStdin;                                 /* Dedicated standard I/O             */
FILE  *MyStdout;
FILE  *MyStderr;

…

   MyStdin  = fopen(…);                         /* Open the dedicated standard I/O    */
   MyStdout = fopen(…);
   MyStderr = fopen(…);

   MyTask   = TSKcreate("My Task", MY_PRIO, MY_STACK_SIZE, MyFct, 0);

   ((struct reent *)(MyTask->XtraData[0]))->_stdin     = MyStdin;
   ((struct reent *)(MyTask->XtraData[0]))->_stdout    = MyStdout;
   ((struct reent *)(MyTask->XtraData[0]))->_stderr    = MyStderr;
   ((struct reent *)(MyTask->XtraData[0]))->__sdidinit = 1;

   TSKresume(MyTask);

…
```

3.3.2 Non dynamic reentrant library

Newlib built without dynamic reentrance protection is set-up as described in section 3.1. Define the build option `OS_NEWLIB_REENT` (for both the compiler and assembler) and set it to a non-zero value to enable the protection.

If a non dynamic reentrant library is used on a multi-core target, although the library itself is not capable of performing proper multi-thread safe protection, Abassi offers a stopgap alternative. If the built option `OS_NEWLIB_REDEF` is defined and set to a non-zero value, then when the number of core is greater or equal to 2, the dynamic re-entrance protection is "emulated" by redefining all (to the best of our knowledge) functions that require dynamic re-entrance protection. This approach has one caveat because when the application is built with C++, these re-definitions could be in conflict with C++ classes.

If turning on the build option `OS_NEWLIB_REDEF` is unsatisfactory, there is another way around. All functions in the library that need to be multi-thread safe are always internally re-mapped to their equivalent multi-thread safe (the function name is post-pended with `_r`), which calls `__getreent()`. When the library is not built with dynamic re-entrance protection `__getreent()` is internally defined as `_impure_ptr`. Because these equivalent functions are always available, they can also be used directly. So for example, instead of using `printf()` directly:

```
printf("Hello World\n");
```

The equivalent function, `printf_r()` should be used:

```
printf_r(__getreent(), "Hello World\n");
```

Please refer to the NewLib user manual [R3] for further details.

3.3.3 Library with dynamic reentrance

When the Newlib is built with dynamic re-entrance protection, a build option can be used to enhance the protection. The build otion `OS_NEWLIB_MUTEX_SINGLE`, which selects what mutex are provided to the `generic_gxx_...()` functions internally used by Newlib. These mutex calls provide protections on other types of library modules than the ones listed in section 3.1; for example, file I/O become protected against concurrent access. The build option `OS_NEWLIB_MUTEX_SINGLE` can be set to three different approaches for the mutex protection. When set to a negative value, the mutex used by all calls to `generic_gxx_...()` functions is the global mAbassi `G_OSmutex` mutex. When set to a positive value, then a dedicated mutex, named `G_OSmtxGCC` is used instead of `G_OSmutex`. When set to a value of zero, they are created/destroyed upon request. As mAbassi does not support the real destruction of services (for safety), "destroyed" mutexes are stored in a *parking lot* and when a mutex creation is requested, the parking lot is emptied before creating a new mutex with `MTXopen()`. All mutex created/destroyed are un-named mutexes. By default, the build option `OS_NEWLIB_MUTEX_SINGLE` is set to a positive value; i.e. the dedicated mutex `G_OSmtxGCC` is used.

3.4 IAR

The IAR DLib library supports both the reentrance safe and multi-thread safe protection. Contrary to the other libraries, Abassi provides re-entrance protection through the file `Abassi_IAR_MTX_IF.c`. Al there is to do is defined `OS_IAR_MTHREAD` to a non-zero value and add the file `Abassi_IAR_MTX_IF.c` in application.

3.5 Keil (ARMCC)

The regular Keil's ARMCC library supports both the reentrance safe and multi-thread safe protection. Microlib library supports none of the two. Therefore, when MicroLib is used, any build options related to the library protection are ignored.

4 References

[R1] mAbassi RTOS – User Guide, available at http://www.code-time.com
[R2] Abassi Port – Cortex-A9, available at http://www.code-time.com
[R3] NewLib documentation, available at http://sourceware.org/newlib