Dalhousie University
Department of Electrical and Computer Engineering

ECED 4402 – Real Time Systems

Assignment 1: Memory Management

1 Objectives

The objective of this assignment is to design, implement, and test a simple memory management system supporting the dynamic allocation and deallocation of memory blocks from an 80 KiB heap in the SRAM memory on an ARM Cortex-M3.[1]

2 Overview

Memory is a shared resource that must be protected from unauthorized access. At a minimum, the operating system should ensure that, unless otherwise instructed, at most one program should have access to a block of memory at any time. Dynamic memory is memory whose size is determined and allocated at runtime as opposed to compile time (as is the case with globals and automatics).[2]

Two utilities are to be written to handle the dynamic allocation and deallocation of memory. The allocation utility is to take a specified number of bytes and return the address of a memory block of *at least* that size. The deallocation utility is to take an address and attempt to return that memory back to the pool of available blocks.

The Cortex SRAM address space allows for a maximum of 512 MiB (0x2000.0000 or 536,870,912 bytes), occupying addresses 0x2000.0000 to 0x3FFF.FFFF. Of this, the Cortex-M3 uses 96 KiB (address range 0x2000.0000 through 0x2001.7FFF or 0x1.8000 bytes); this is allocated to the initial stack, the uninitialized global variables (**.BSS**), the initialized global variables (**.DATA**), and the memory reserved at compile time for the system memory (**.sysmem**). This memory, including **.sysmem**, is contiguous starting at 0x2000.0000.

The total space for the heap to be designed and implemented in this assignment is to be 80 KiB, allocated in blocks as follows:

Number	Size (bytes)
128	0x80
64	0x100
32	0x200
16	0x400
8	0x800

[1] MiB and KiB refer to Mibibytes and Kibibytes (or mega-binary or kilo-binary bytes), respectively, as defined by the International Electrotechnical Commission (or IEC).

[2] Although the physical space for automatics is obtained from the stack at runtime, the required size of a subroutine's stack is determined at compile time.

Each block is to have a fixed length and, for ease of implementation, the location of each should be determined at run time. If insufficient memory exists in the SRAM for the proposed heap, reduce the number of 2 KiB blocks until it fits.

The allocation routine should attempt to return the address of a memory block that is at least as long as the specified size; in the worst case, this means returning a 2 KiB block for a one byte request. If the size request cannot be met (i.e., no memory available or the request is greater than 2 KiB), an address of **NULL** is to be returned. It is up to the application to recognize the **NULL** pointer and terminate its activities; for example:

```
char *ptr;
...
ptr = allocate(512);
if (ptr == NULL)
{
   printf("Allocation failed\n");
   return;
}
```

The deallocation routine expects an address of a memory block; the address must match the location of one of the blocks currently allocated. If the address is invalid or the block is not in use, the deallocation routine should return a value of **FALSE**; otherwise the block is available for reuse and a value of **TRUE** is to be returned; for example:

```
...
if (!deallocate(ptr))
{
        printf("Deallocation failed\n");
        return;
}
```

Finally, there is no memory protection, other than the assumption that applications won't write outside their allocated memory area.

3 Other Issues

Two questions that you should be asking yourself at this stage are, "where does the **.sysmem** start and what is its length?" Not surprisingly, the designers of a **malloc()** routine would face this same question. Fortunately, the linker makes both the starting address of **.sysmem** and its size available for use by a program through the two external variables **_sys_memory** and **_SYSMEM_SIZE**, respectively. A program can access them as follows:

```
extern void* _sys_memory;      // Start of system memory
extern int __SYSMEM_SIZE;      // Size of system memory
```

Pointer (i.e., address) manipulation should be done using 32-bit unsigned integer arithmetic. Pointers can be cast to integers with **uin32_t**; this and other built-in types are stored in **stdint.h** and can be included using **#include <stdint.h>**.

The system-stack and system-heap sizes are specified at link-time using the following steps:

a) Right-click on the currently active project in the C/C++ Projects Window
b) Select (left-click) on *Build Properties…*
c) Select *Tool Settings* tab (default) in *Configuration Settings*
d) Under the *Linker*, select *Basic Options*
e) System stack size and heap size can now be specified; hex (0x prefix) is allowed

4 Marking

This assignment will be marked as follows:

Design
The design description must include a brief introduction of the problem, a detailed analysis of the problem, and a description of the algorithms and data structures associated with each of the three parts of the memory management software (i.e., initialization, allocation, and deallocation).

Total points: 4.

Software
A fully commented, indented, magic-numberless, tidy piece of software that meets the requirements described above and follows the design description.

Total points: 4.

Testing
A set of tests showing that the allocation and deallocation works. The tests should be in two parts: the first describing the test and what the test is to achieve; and the second, describing how the software met the objectives.

Total points: 2.

The assignment must be submitted on paper.

5 Important Dates

Available: 11 September 2015

Due: 28 September 2015 (start of class)

Late assignments will be penalized half-a-point per day or fraction thereof.

6 Miscellaneous

It will be necessary to download a copy of TI's Code Composer Studio (CCS). Details on how this is done can be found on the course website.

If you want to implement a more complex memory allocation and deallocation algorithm (such as buddy memory allocation), you may do so.

The allocation software does not actually allocate any memory; it simply maintains one or more data structures that represent the storage that has been allocated. In other words, it is not necessary for the SRAM used for the heap to be physically accessed in this assignment

(although it should so that you can prove to yourself that the addresses are valid). The software should be tested on the Cortex simulator.

Do *not* discard this work once you've finished with it, since this makes up part of the operating system you will eventually implement.

If you are having *any* difficulty with this assignment or find any problems with it, *please* contact me as soon as possible.

Although **deallocate()** should act like the memory deallocation function **free()** found in C's memory allocation library, it is different in that most, if not all, versions of **free()** require the pointer to have the same address that was supplied by **malloc()**. In other words, the following would probably not work in most versions of **free()**:

```
ptr = malloc(100);
ptr++;
free(ptr);
```

And while on the subject of **malloc()** and **free()**,these functions are not to be used in this or any other assignment in this course. You are to write software that takes control of the Cortex.