Anushka Chougule
CSS422: Final Project
Prof: Munehiro Fukuda
March 11, 2025

Final Project Documentation

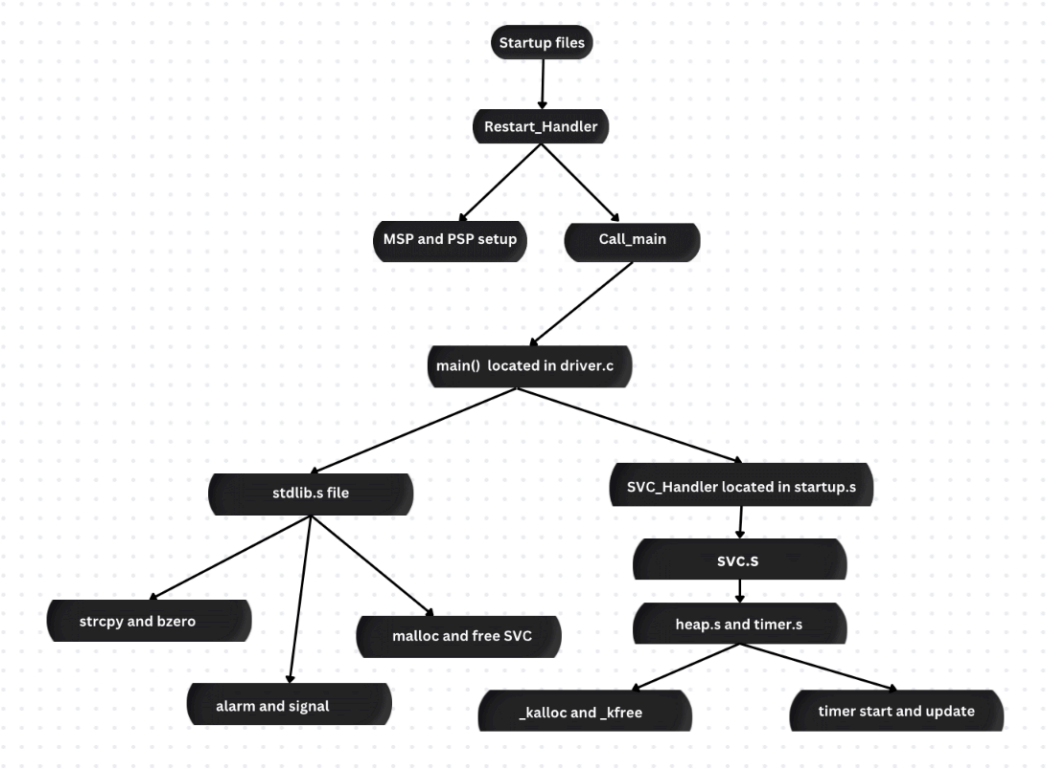

Diagram description: The diagram above shows a simple outline of the final project, highlighting which files are being used and the order they are called in.

What has been implemented:

In this final project, I implemented several C standard library functions using Thumb-2 assembly, which are tested through the driver.c file. The file ensures that the implemented functions behave as expected by running the driver_keil.c file. The primary functions implemented in the stdlib.s file are bzero, strncpy, malloc, free, signal, and alarm.

The bzero function fills a memory block with zeros, ensuring that memory is properly initialized. The strncpy function copies a specified number of characters from one string to another, ensuring no overflow occurs in the destination string. The malloc function allocates memory of a requested size and returns a pointer to the allocated memory, which is essential for

dynamic memory management. The free function deallocates memory that was allocated by malloc, helping to prevent memory leaks. The signal function handles specific signals. Lastly, the alarm function sets a timer, which signals after a defined time period.

The heap.s file manages dynamic memory allocation and deallocation, which is crucial for low-level systems programming. It includes the implementation of malloc for memory allocation, free for memory deallocation, and memory merging using the buddy system, which optimizes memory management by reducing fragmentation.

The svc.s file initializes the system call table with specific addresses for system calls. This table maps system calls to function addresses, and the jump routine executes the correct function based on the system call number. This allows for an efficient handling of system-level operations like I/O or memory management.

The timer.s file is responsible for managing the timer functionality. It initializes the timer by setting registers and clearing the current time. The start function begins the countdown with a specified value, while the update function decrements the timer on each call. A custom signal handler has been set up for when the timer expires and allows flexibility for time-dependent tasks.

Overall, the core functionality is set within stdlib.s, where C standard library functions are implemented in assembly. These functions are called through the driver_keil.c file, which serves as a test to verify the correct behavior of each function. The assembly files manage memory, system calls, and timers, providing an environment similar to C libraries in programming.

What was missing + Narrative:

Before starting the final project, I reviewed the .s and .c files. At first it felt overwhelming, but I got a lot of help from the QSC tutors, who guided me through the program design and helped me understand what was missing. They walked me through the key concepts, which gave me a clearer idea of how to approach the implementation. My starting point was the startup.s and stdlib.s files, where many functions were declared but needed proper data processing and memory allocation. Implementing these functions early on made the process smoother and allowed me to focus on debugging and improving efficiency later. To prepare I also watched YouTube tutorials on running Keil and understanding the basics of ARM assembly before I started working on the actual code.

By the midpoint, I had a solid foundation since I had already worked with a tutor for guidance. I implemented the heap.c file and startup_tm4c129.s before working on stdlib.s. At this stage, I felt more confident in navigating the code and understanding how different parts connected. I performed well but lost a few points on the midpoint due to incorrect output and

screenshot errors. However, Professor Fukuta noted in the rubric that my logic was correct, which made me realize that I might have been executing the code incorrectly due to the wrong ARM settings rather than an actual logic error.

For the final stage, I made significant progress. Compared to my midpoint output, my implementation was much more refined. Initially, I had included too many comments, which made the code harder to navigate. To improve readability, I cleaned it up and kept only brief, relevant comments. This helped me work more efficiently and made it easier to debug.

Overall, this project required a lot of hands-on work and logical thinking. Despite the challenges, I gained a solid understanding of how assembly interacts with C. The process helped me improve my problem-solving skills, and by the end, I felt much more comfortable working with assembly language.