Final Project Report

Team Name: Assemblers

Members: Artem Grichanichenko and Kirill Grichanichenko

Final Project Design Information

startup_TM4C129.s

This file sets up the startup routine for the TM4C129 microcontroller. We modified it to initialize system-level components like heap, timer, and system call table. Also, we configured the stack pointer and switched the processor to unprivileged thread mode using the Process Stack Pointer (PSP), and implemented handlers for SVC, and SysTick interrupts to manage system calls and timing updates.

timer.s

This file manages the system timer used to trigger alarm-based signals. We implemented the _timer_start, _timer_update, and _signal_handler functions to control countdown timing, call user-defined signal handlers, and register new handlers using memory-mapped constants. The timer is driven by the SysTick interrupt, decrementing the countdown each second and using the appropriate handler when the countdown reaches zero.

_timer_init

- Stops the SysTick timer, sets the reload value to 1 second, and clears any current count to prepare for use.

_timer_start

- Saves the previous alarm duration, sets the new countdown value, starts the SysTick timer, and clears the current counter.

_timer_update

- Decrements the seconds-left counter on each interrupt. If the counter reaches zero, stop the timer and call the user-registered signal handler if available.

_signal_handler

- If the signal is SIGALRM, replace the current signal handler with the provided one and return the previous handler.

svc.s

This file handles system call routing by setting up and using a system call jump table. We implemented _syscall_table_init to map specific system call numbers to their corresponding kernel routine, and _syscall_table_jump to dynamically call these functions based on the value in register R7. This design allows system calls like malloc, free, signal, and alarm to be handled through a separate file.

_syscall_table_init

- Initializes the system call table by storing function addresses for _timer_start, _signal_handler, _kalloc, and _kfree into specific memory slots based on the system call number.

_syscall_table_jump

- Uses the system call number in R7 to look up the corresponding function address in the system call table, then branches to that function and returns its result.

stdlib.s

This file implements C standard library functions in Thumb-2 assembly, which includes _bzero and _strncpy entirely in user mode. The system call functions like malloc, free, alarm, and signal are implemented as wrappers that set the appropriate system call number and trigger a supervisor call (SVC). These functions follow the ARM Procedure Call Standard and rely on the SVC_Handler to send the actual operations.

_bzero

- Clears a block of memory by writing zeros to n bytes starting from the given address using a loop.

_strncpy

- Copies characters from a source string to a destination buffer, padding with null bytes if the source is shorter than the specified length.

_malloc

- Prepares a system call with identifier 4 and uses SVC to allocate memory through the kernel.

_free

- Uses a system call with identifier 5 to deallocate a previously allocated memory block.

_alarm

- Triggers a system call with identifier 1 to start a countdown timer and returns the previous alarm duration.

_signal

- Registers a signal handler by issuing a system call with identifier 2 and returns the previous handler if one was set.

heap.s

This file implements the buddy memory allocation system in the ARM Thumb-2 assembly. We developed functions to initialize the memory control block (_heap_init), allocate memory recursively using _kalloc and _ralloc, and free memory using _kfree and _rfree, including a recursive combination of adjacent blocks. The system manages heap memory in 32-byte aligned segments that track availability and size with a memory control block of 512 entries.

_heap_init

- Initializes the memory control block (MCB) by marking the entire heap space as available and clearing the remaining entries.

_kalloc

- Entry point for memory allocation. Checks for minimum size, then calls _ralloc recursively to find and allocate a suitable memory block.

_ralloc

- Recursively searches for a fitting memory block by dividing memory segments. Marks blocks as used and returns the corresponding heap address.

_kfree

- Converts a given pointer back into its MCB entry and calls _rfree to mark the block as free, which validates boundaries in the process.

_rfree

- Frees a block and attempts to recursively merge it with its buddy if both are free and the same size, supporting combination in the buddy allocation system.

More design information available directly in the project comments

Missing Work

Here is a list of some of the missing work in the final project:

- We did not have time to implement any optimizations to the project
- We did not have time to implement SYS_MEMCPY

Final Project Visual Element

Figure 1 : Final Project Class Diagram

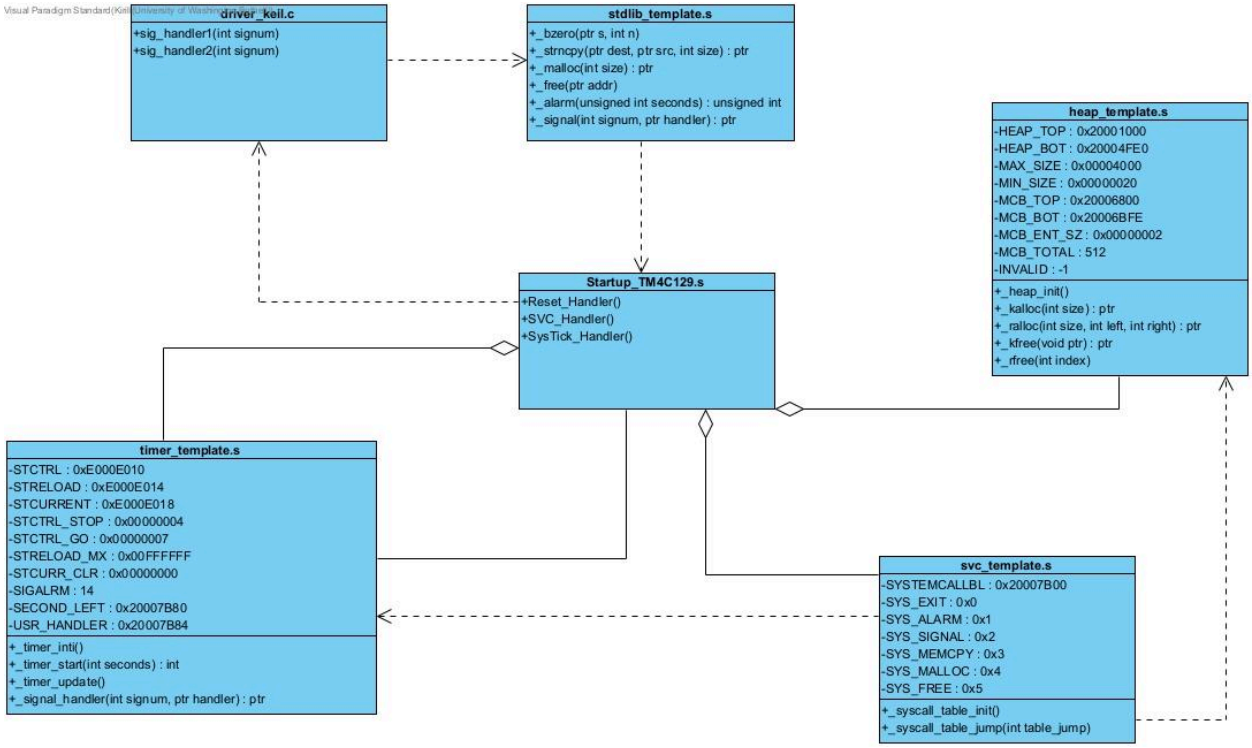

Figure 1 Explanation

This class diagram represents the six major files within the final project. The six files include:

- StartUp_TM4C129.s
- heap_template.s
- driver_keil.c
- timer_template.s
- stdlib_template.s
- svc_template.s

This diagram shows the functionality with each file and the relationships that the files share with each other. We can see that svc_template.s, timer_template.s, and heap_template.s all share an aggregation relationship with StartUp_TM4C129.s. The StartUp_TM4C129.s file is responsible

for initializing those three files and communicating between the stdlib_template.s and svc_template.s.

Furthermore, we can see the dependency between the StartUp_TM4C129.s and driver_keil.c files where StartUp_TM4C129.s branches to the driver_keil.c after setting up the three aggregated files. Following the setup into the driver_keil.c file we can see the dependency with the stdlib_template.s file where the driver depends on the libraries custom functions. The stdlib_template.s has a dependency with the startup because of the SVC_Handler() which leads stdlib_template.s' privileged functions to the svc_template.s file. The svc_template.s file is then responsible for directing the correct table jump into timer_template.s and/or heap_template.s where they share a dependency with each other.

Execution Screenshots

All execution screenshots are located in the Additional_Folder->Screenshots folder.