Nithisha Sathishkumar

CSS 422

Professor Browne

June 8, 2024

Final Project Documentations

Starting with my experience on this project, I found it very challenging and struggled to implement many of the functions, especially starting from the midpoint. Fortunately, since I began working on the project one week after it was assigned, I was able to complete it a week early. The midpoint wasn't too difficult for me because it primarily involved implementing the C part of the heap. However, there was a bug in the _ralloc function where every memory from 1 to 7 output correctly, but at memory 8, the output kept returning 0. I had to pull multiple all-nighters to fix this issue, but thankfully, I resolved it two days before the due date.

For the startup file, _bzero and _strncpy were easier to implement, and I completed them in two days. I referenced the final documentation, slides, and online YouTube resources to complete the midpoint. I was unaware that we were supposed to submit additional screenshots beyond those mentioned in the final documentation for the midpoint, which caused me to lose some points. Overall, despite the challenges, the midpoint phase went relatively smoothly.

Since the midpoint already covered the logic of the heap implementation, working on the assembly part was not as challenging as it was the first time. Thankfully, the professor released the heap implementation key from the midpoint, which I used as a guide to write the assembly for the heap. I had a tough time with the _rfree part because it kept getting stuck in the loop and would not progress further. Like the midpoint, it took me a while to debug the code.

In heap.s, the function heap_init initializes the memory control blocks (MCBs) used for memory allocation tracking. The _kalloc function handles kernel memory allocation by finding a suitable memory chunk in the heap and marking it as allocated. If the requested size is not available, it recursively splits larger chunks to fulfill the request. The _kfree function handles kernel memory deallocation by finding the corresponding MCB for the given memory address and marking the memory as deallocated. It also performs buddy system coalescing to merge contiguous free memory blocks. Lastly, _rfree performs buddy system coalescing during kernel memory deallocation. It recursively merges adjacent free memory blocks using the buddy system algorithm until coalescing is complete or no further merging is possible.

For timer.s, in the initialization function _timer_init, I first initialized the timer by disabling it, setting the reload value, and clearing the current value register. In _timer_start, I started the timer with the specified countdown value, enabled the timer, and cleared the current register. In _timer_update, it decreases the timer, stops the timer when it reaches zero, and calls a

user-defined handler. Lastly, for the signal_handler, I set the user-defined signal handler for the SIGALRM signal and returned the old handler.

In svc.s, the initialization function _syscall_table_init initializes the system call table with function addresses. The _syscall_table_jump function jumps to the appropriate system call function based on the system call number passed in register R7. Lastly, the jump table contains addresses of functions corresponding to each system call number.

In stdlib.s, the first function is _bzero, which sets each byte in a memory block to 0. The second function is _strncpy, which copies characters from the src string to the dest string until either size characters are copied, or a null terminator is encountered in src. The third function is _malloc, which allocates size bytes of memory and returns a pointer to the allocated memory block. The fourth function is _free, which deallocates the memory block pointed to by addr, making it available for future use. The fifth function is _alarm, which schedules an alarm signal to be delivered after seconds and returns the number of seconds remaining until any previously scheduled alarm was due to be delivered. The sixth function is _signal, which sets a signal handler for the signal specified by signum and returns a pointer to the user-level signal handling function that was previously set for the same signal.

For extra credit, I implemented four functions: _strcmp, _strlen, _atoi, and _strcpy. _strcmp compares the characters of both strings until it finds a difference or reaches the end of either string. It has parameters const char* str1 (the first string) and const char* str2 (the second string). It returns 0 if the strings are equal, a negative value if str1 is less than str2, and a positive value if str1 is greater than str2. The results are stored at address 0x2000000. _strlen iterates through the characters of the string until it encounters the null terminator, counting each character encountered. It has the parameter const char* str (a pointer to the string). It returns the length of the string, stored at address 0x2000000. _atoi converts the string str to an integer, assuming it represents a decimal number. It skips leading whitespace characters and stops parsing when it encounters a non-digit character. It has the parameter const char* str (a pointer to the string). It returns the converted integer value, stored at address 0x2000000. _strcpy copies characters from the src string to the dest buffer until it encounters the null terminator in src. It has parameters char* dest (the destination buffer) and const char* src (the source string). The results are stored at address 0x2000000.

Overall, I am thankful that I have no missing components in my code, and the screenshots match those provided in the final project document. Each method worked correctly, outputting the expected values into the specified memory addresses. They all connected to each other by passing arguments correctly through the APCS-based registers and returning the correct values. Memory allocation and deallocation were executed as planned, and the timer-related events functioned correctly. Additionally, my extra credit implementations of `_strcmp, _strlen, _atoi, and _strcpy` worked as expected, as shown in the screenshots below. This course has taught me that I never want to code in ARM assembly again, but I am extremely glad that I

followed the instructions thoroughly and everything is fully implemented and functioning as expected.

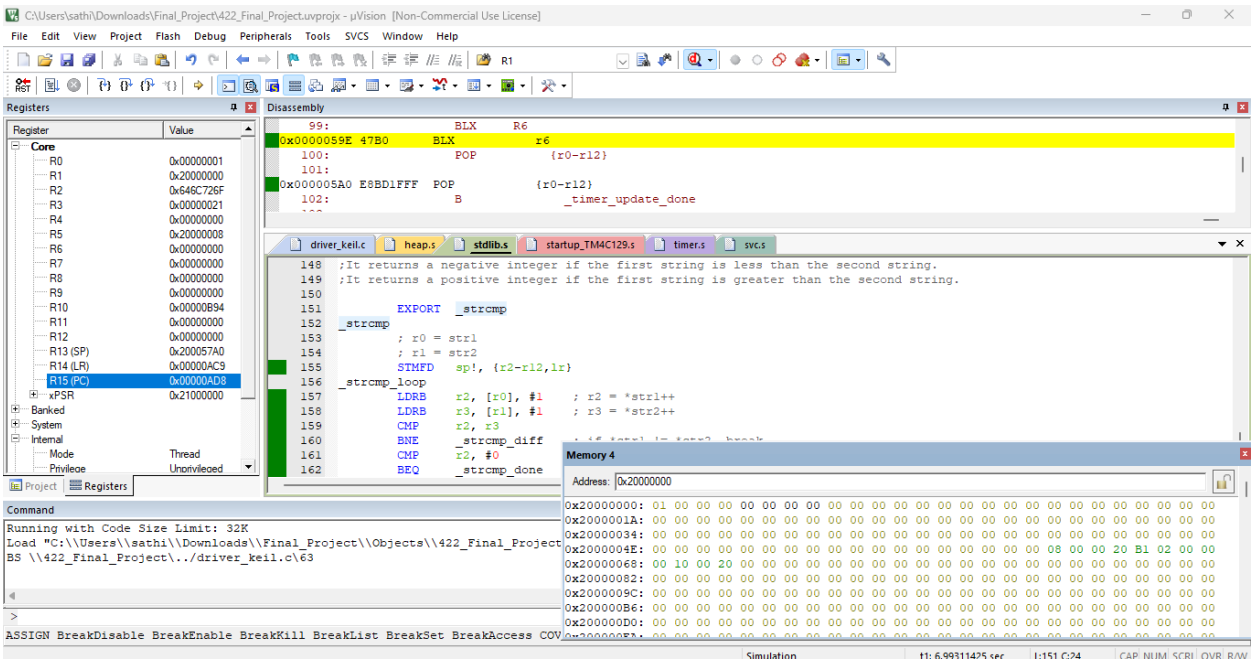
READ THIS

The driver_keil includes the test case to run the extras credit functions with are commented out because it affects the content of the signal_handler. The Startup.s is in the Final Project-> RTE->devices->TM4C129XNCZAD->startup file.s. I wanted to make sure you all know that. The diagram is in the documentation.

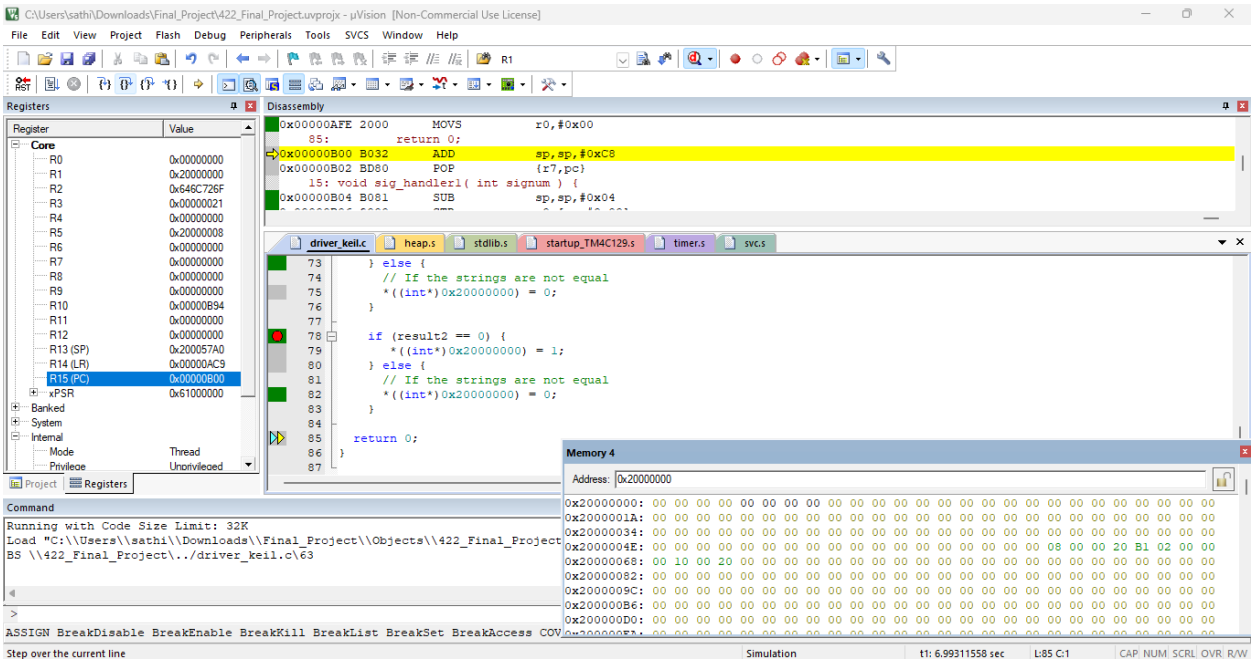
EXTRAS CREDIT

_Strcmp

```
char str1[] = "Hello, World!";
char str2[] = "Hello, World!";
char str3[] = "Hello, ARM!";

int result1 = _strcmp(str1, str2); // result1 should be 0 (equal strings)
int result2 = _strcmp(str1, str3); // result2 should be positive (str1 > str3)


if (result1 == 0) {
        *((int*)0x20000000) = 1;
} else {
// If the strings are not equal
*((int*)0x20000000) = 0;
}
```

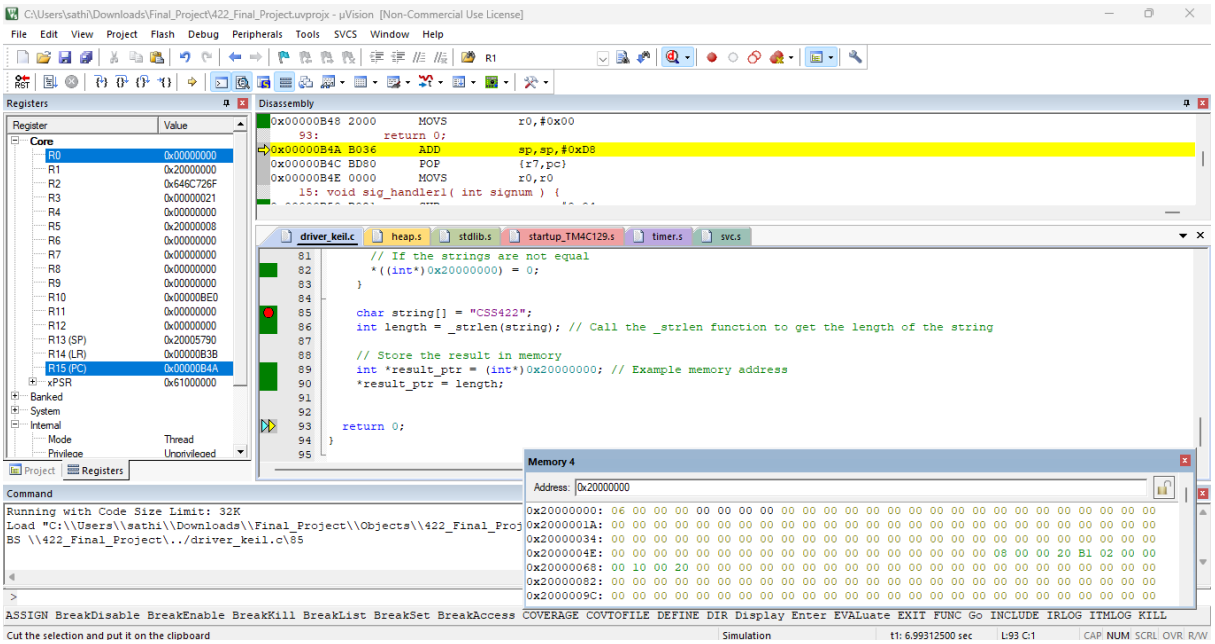



```
if (result2 == 0) {
      *((int*)0x20000000) = 1;
} else {
        // If the strings are not equal
         *((int*)0x20000000) = 0;
}
```



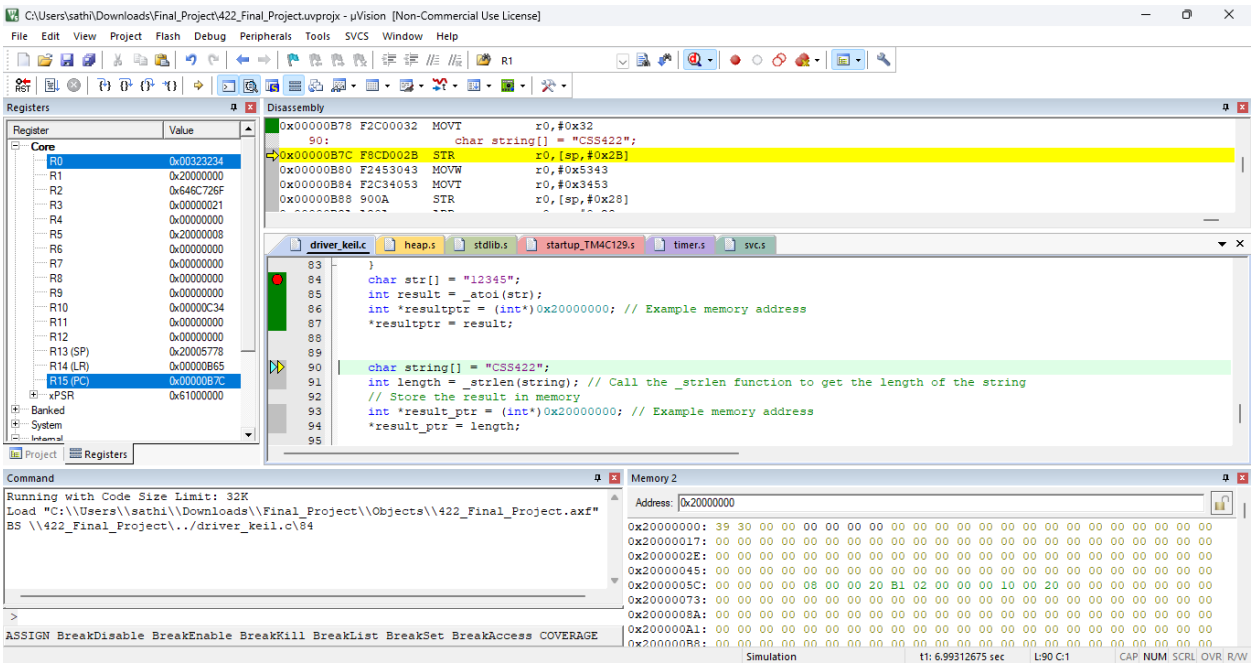
_strlen

```
char string[] = "CSS422";
int length = _strlen(string); // Call the _strlen function to get the length of the string
// Store the result in memory

int *result_ptr = (int*)0x20000000; // Example memory address

*result_ptr = length;
```



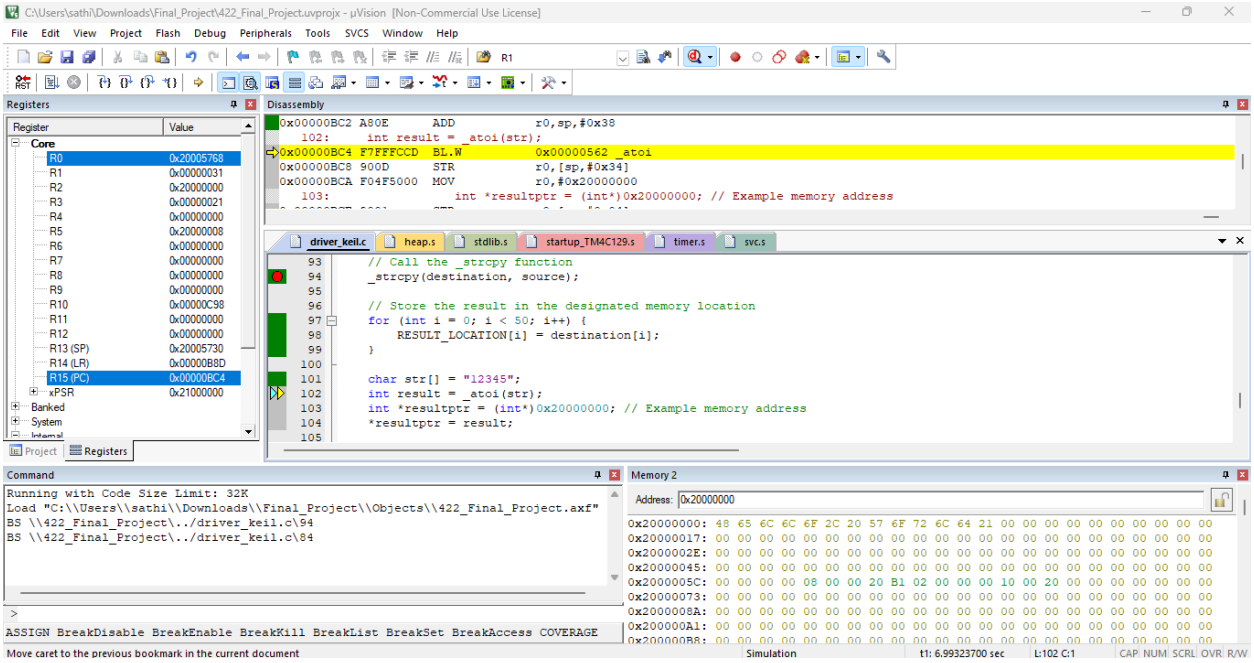
_atoi

```
char str[] = "12345";
int result = _atoi(str);
int *resultptr = (int*)0x20000000; // Example memory address
*resultptr = result;
```

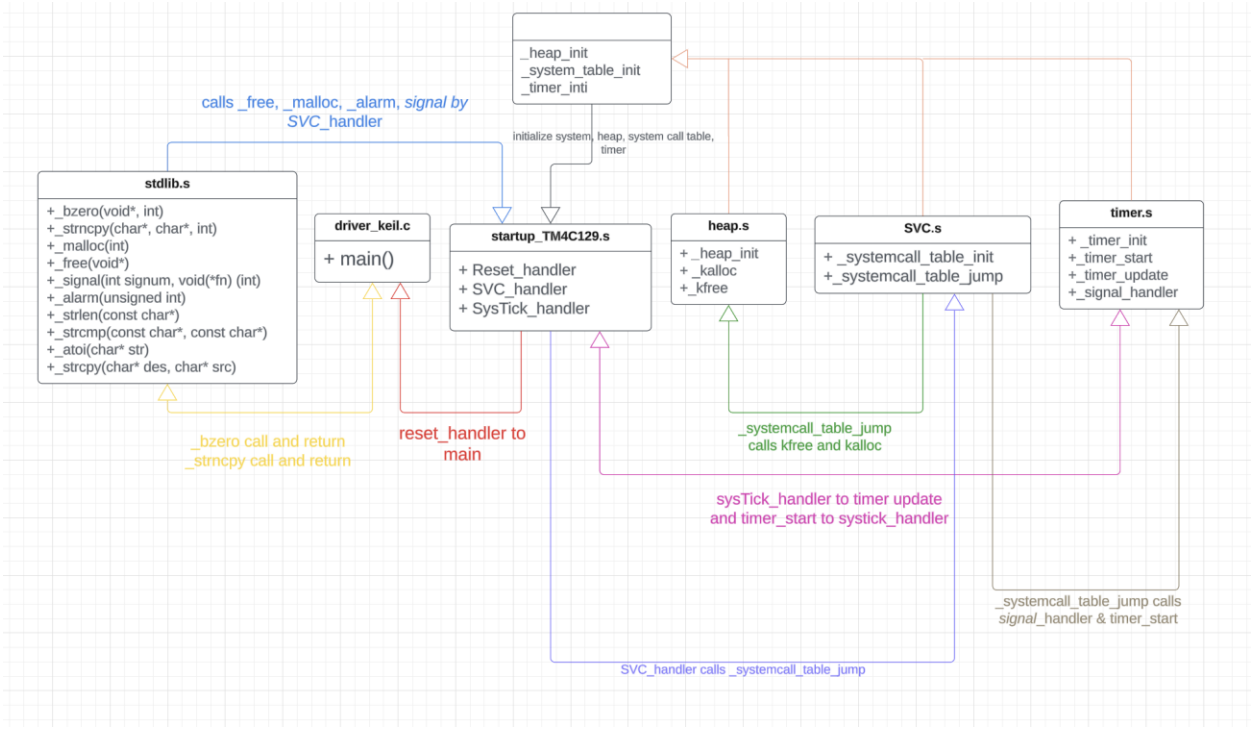


_strcpy

```
// Define source string
char source[] = "Hello, World!";

// Define destination string
char destination[50]; // Make sure it is large enough to hold the source string


_strcpy(destination, source); // Call the _strcpy function

for (int i = 0; i < 50; i++) { // Store the result in the designated memory location

    RESULT_LOCATION[i] = destination[i];
}
```


C:\Users\sathi\Downloads\Final_Project\422_Final_Project.uvprojx - µVision [Non-Commercial Use License]
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Registers
Register Value
Core
R0 0x20005768
R1 0x00000031
R2 0x20000000
R3 0x00000021
R4 0x00000000
R5 0x20000008
R6 0x00000000
R7 0x00000000
R8 0x00000000
R9 0x00000000
R10 0x00000C98
R11 0x00000000
R12 0x00000000
R13 (SP) 0x20005730
R14 (LR) 0x00000B8D
R15 (PC) 0x00000BC4
xPSR 0x21000000
Banked
System
Project Registers
Disassembly
0x00000BC2 A80E ADD r0,sp,#0x38
102: int result = _atoi(str);
0x00000BC4 F7FFFCCD BL.W 0x00000562 _atoi
0x00000BC8 900D STR r0,[sp,#0x34]
0x00000BCA F04F5000 MOV r0,#0x20000000
103: int *resultptr = (int*)0x20000000; // Example memory address
driver_keil.c heap.s stdlib.s startup_TM4C129.s timer.s svc.s
93 // Call the _strcpy function
94 _strcpy(destination, source);
95
96 // Store the result in the designated memory location
97 for (int i = 0; i < 50; i++) {
98 RESULT_LOCATION[i] = destination[i];
99 }
100
101 char str[] = "12345";
102 int result = _atoi(str);
103 int *resultptr = (int*)0x20000000; // Example memory address
104 *resultptr = result;
105
Command
Running with Code Size Limit: 32K
Load "C:\\Users\\sathi\\Downloads\\Final_Project\\Objects\\422_Final_Project.axf"
BS \\422_Final_Project\../driver_keil.c\94
BS \\422_Final_Project\../driver_keil.c\84
ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess COVERAGE
Memory 2
Address: 0x20000000
0x20000000: 48 65 6C 6C 6F 2C 20 57 6F 72 6C 64 21 00 00 00 00 00 00 00 00 00
0x20000017: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x2000002E: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x20000045: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x2000005C: 00 00 00 00 08 00 00 20 B1 02 00 00 00 10 00 20 00 00 00 00 00 00
0x20000073: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x2000008A: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
0x200000A1: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
Move caret to the previous bookmark in the current document
Simulation
t1: 6.99323700 sec
L:102 C:1
CAP NUM SCRL OVR R/W

Diagram


_heap_init
_system_table_init
_timer_inti
calls _free, _malloc, _alarm, signal by SVC_handler
initialize system, heap, system call table, timer
stdlib.s
+_bzero(void*, int)
+_strncpy(char*, char*, int)
+_malloc(int)
+_free(void*)
+_signal(int signum, void(*fn) (int)
+_alarm(unsigned int)
+_strlen(const char*)
+_strcmp(const char*, const char*)
+_atoi(char* str)
+_strcpy(char* des, char* src)
driver_keil.c
+ main()
startup_TM4C129.s
+ Reset_handler
+ SVC_handler
+ SysTick_handler
heap.s
+ _heap_init
+ _kalloc
+ _kfree
SVC.s
+ _systemcall_table_init
+ _systemcall_table_jump
timer.s
+ _timer_init
+ _timer_start
+ _timer_update
+_signal_handler
_bzero call and return
_strncpy call and return
reset_handler to main
_systemcall_table_jump calls kfree and kalloc
sysTick_handler to timer update and timer_start to systick_handler
_systemcall_table_jump calls signal_handler & timer_start
SVC_handler calls _systemcall_table_jump