Cache Commandos
12/02/2023
CSS422

FINAL PROJECT DOCUMENTATION/SUMMARY

Driver_Keil.c

- This is the C program that serves as our main driver and links to our assembly functions using the extern command followed by a series of pre-written tests that tests our implementation. The only change made to this document was to comment out the while loop as per the instructions given in the pdf.

Stdlib.s

- _bzero : To implement this function we first stored our destination variable into r3 and assigned r0 to the value 0. Then the code uses a while loop approach that decrements n iteratively until our passed n value reaches 0 which then calls the branch that returns the destination value. During each iteration the code is zero-initializing each spot it goes over until the branch condition is met.
- _strncpy : This function starts out by storing the destination value into r3 and entering a while loop once again. In this loop the size variable that was passed and stored in r2 is decremented while the src value is stored into the destination value. This iterates through until we meet our branch condition which is if size is 0 or if src value is empty. Both of these branch into _strncpy_return which simply returns our destination.
- _malloc
    -
- _free
    -
- _alarm
    -
- _signal
    -

Svc.s

- In this file the system call table is initialized, loaded into memory, and stored to the table to be utilized by different files by the two functions _syscall_table_init and _syscall_table_jump.

Heap.s

- _heap_init : First, the beginning entry of the mcb is initialized to ensure the entire 16KB space is available by loading r1 with MCB_TOP, giving r0 the 16KB max size, and then storing it. Then, after initializing the amount of entries into r2 we use _heap_init_loop to zero-initialize the entire 16KB of memory through a simple loop until we branch and return.
- _ralloc : First the mcb's entire size is calculated by doing right(r2) - left(r1) + mcb_ent_sz(r3). This is then halved and stored in r5 to get the mcb's half size and

midpoint. The half size(r9) and entire size(r8) are then multiplied by 16 to get the actual size. After all these calculations the next step is to use compares to decide which step is next.

  - If the size is greater than or equal to the actual half size then _else is called which takes the beginning value in the range and if it is 0 or is too small returns NULL. If it is 1 then it does a final calculation of (left -mcb_top) * 16 before branching to done and finishing the call.
  - The next path forward is if right = midpoint - mcb_ent_sz then we recursively loop through back to the top of _ralloc. After breaking from that recursion once the heap address is 0 we branch to _rightalloc which simply moves the recursion call to the right side of the current range.

- _kalloc : This function starts out by saving the return location and comparing r0 which holds the current size that was passed through the parameter. If it is greater than or equal to 32 then it branches normally to _alloc which after loading the address location of the range as well as the size of each entry calls to our recursive helper function _ralloc. If the size doesn't meet the branch condition of being at least 32, then the size is set to the minimal size of 32 and returned.
- _kfree : Function starts out by saving addr, HEAP_TOP, and HEAP BOT. It then compares addr to HEAP_TOP and stops the run with a null return if they are not equivalent. The address of the mcb is then calculated through adding MCB_TOP to addr minus HEAP_TOP divided by 16. Now that the address is stored the helper function rfree is called and the finishing value is returned.
- _rfree
  - The first thing that happens is the mcb contents are split into chunks, the used bit is cleared, and the offset is stored so that the buddy memory allocation algorithm can be used. Calculating ((mcb offset / mcb chunk) % 2) determines whether or not the indicator is left or right.
    - If left then the code is branched to _left_free which adds the mcb's address with the chunk and if it is lower than mcb_bot then zero is returned.
    - Once the code reaches right as the answer it branches to right_free which compares (mcb_addr - mcb_chunk) and mcb_top. If it is lower than mcb_top then zero is returned, otherwise the code branches to _rejoin. _rejoin tests the code by saying if mcb_buddy is either 0 or is equivalent to my_size then the code branches to done. Otherwise it will branch to _left_free2 which will put the code back through its recursive loop by calling _rfree and returning the new address.

Timer.s

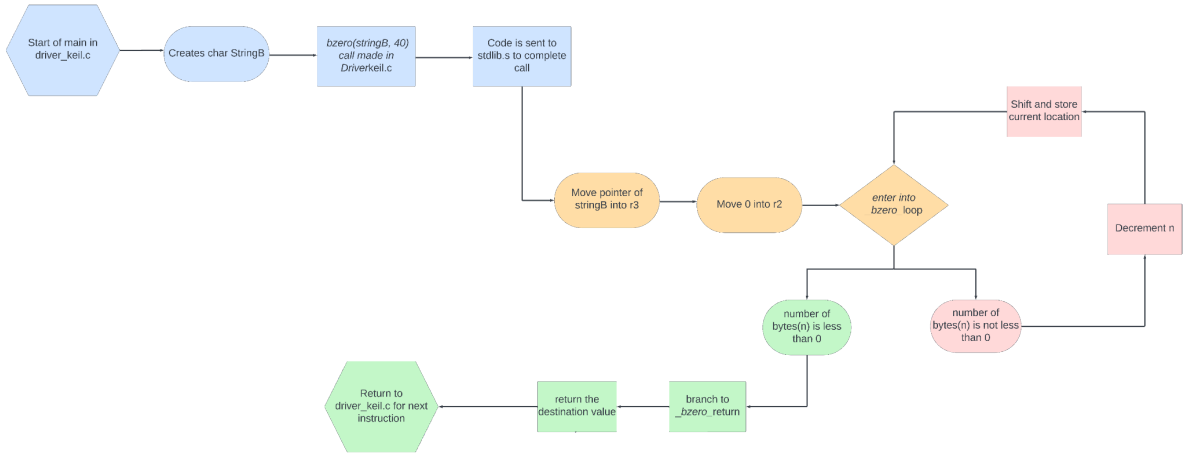
- _timer_init : To initialize the timer in this function the systick is first ensured to be stopped by loading STCTRL_STOP and storing the control/status registers using r0 and r1. The maximum value(1 second) is then loaded into r0 and the Systick reload value

register is stored. Finally the counter, countflag, current value register, and addresses are all cleared so they can be rest for the next call before the code is returned to the reset_handler.

- _timer_start : This function starts out by loading and storing the amount of seconds that are left using SECOND_LEFT, r0, and r1. The control/status registers are also loaded using STCTRL, right before the interrupt and clock enables are set to 1 using STCTRL_GO. Finally the new value is stored and then the program returns to SVC_Handler.
- _timer_update : The amount of seconds that are left on the alarm is grabbed through SECOND_LEFT, r0, and r1. This value is then decremented and enters a branch statement that compares the value to 0. If the value is anything other than 0 then we branch to _timer_update_done that sends the code back to Systick_Handler. If the value is 0 then the timer needs to be stopped by loading the control/status registers through STCTRL and stopping them with STCTRL_STOP.
- _signal_handler : This function loads USR_HANDLER that sends the address of the given signal handler function and returns to the Reset_Handler.

DIAGRAM(S)

This diagram illustrates the _bzero(stringB, 40) call that is made at the beginning of the main in driver_keil.c and how it interacts with stdlib.s.



FINAL SNAPSHOTS FOR EACH TEST