CSS422 Final Project Documentation:

Thumb-2 Implementation for Memory/Time-Related C Standard Library Functions

Date: March 8, 2024

Members: Mandeep Masoun, Harmanjit Sandhu, Erik Lopez

1. Introduction

This document provides a detailed overview of the final project, explaining the concepts involved, project goals, and the overall execution sequence.

This project implements a core set of functionalities for an embedded system kernel or library. It provides essential operations for memory management, string manipulation, system calls, and timer/alarm functionalities.

2. Objective

This project aims to solidify our understanding of the following concepts through implementing memory/time-related C standard library functions in Thumb-2 assembly language for an ARM processor:

- **CPU Operating Modes:** User mode and supervisor mode.
- **System Call and Interrupt Handling Procedures:** How system calls and interrupts are triggered and handled.
- **C to Assembler Argument Passing:** Understanding ARM Procedure Call Standard (APCS) for passing arguments between C and assembly functions.
- **Stack Operations for Recursion:** Implementing recursion using stack operations in assembly language.
- **Buddy Memory Allocation:** Implementing buddy memory allocation for memory management.

3. What we implemented

We implemented several C standard library functions from the **stdlib.h** library using Thumb-2 assembly language. These functions will be called from a C program named **driver.c.**

Implemented Functions:

- **bzero:** Fills a block of memory with zeros.
- **strncpy:** Copies a specified number of characters from one string to another.
- **malloc:** Allocates a requested size of memory and returns a pointer to the allocated block.
- **free:** Deallocates previously allocated memory using a pointer.
- **signal:** Registers a function to handle a specific signal (implemented for SIGALRM only in this project).
- **alarm:** Sets a timer to deliver a signal (SIGALRM) after a specified number of seconds.

Function Implementations:

User Mode (Implemented entirely within stdlib.s):

- **Functions bzero and strncpy:**
    - Implemented directly within the **stdlib.s** assembly file.
    - Can access user-mode memory and registers.

- Do not require switching to supervisor mode.
- Run under the control of the user program (**driver.c**).

Supervisor Mode (Implemented as SVC and handled by the SVC_Handler):

- **Functions malloc, free, signal, and alarm:**
  - Implemented as supervisor calls (SVC).
  - Triggers a switch to supervisor mode when called from **stdlib.s**.
  - Handled by the **SVC_Handler** routine in **startup_TM4C129.s**.
  - Can access all memory and registers (including privileged ones).
  - May interact with system resources or require operating system intervention.

4. What did not work implemented

We've made significant strides in implementing the malloc function, albeit partially. Presently, our approach involves allocating memory in a reverse order, which, although functional, deviates from the standard first-in-first-out order and the memory skips every other slot. Still, it is all there, just spaced out. Despite this, all other aspects of our code are operating smoothly . However, during our efforts to align the allocation order with the standard, we encountered a setback – it seems to be backward.

Additionally, while our free function is generally effective, we've encountered puzzling instances where certain memory segments fail to operate as expected.

On a more positive note, we're enthusiastic about exploring the potential of an extra functionality we've been developing called set_zero_registers. This side feature has been an enjoyable diversion, wherein we clear registers to zero. This capability promises to maintain cleaner register states, facilitating smoother debugging processes.

5. Conclusion

This project provided valuable experience in implementing essential memory and time-related functionalities using the Thumb-2 assembly language. We gained a deeper understanding of ARM operating modes, system call handling, and memory management techniques like buddy allocation. By successfully testing the implemented functions, we demonstrated the ability to bridge the gap between C standard library concepts and their low-level assembly language equivalents. Future improvements could involve expanding the implemented function set, exploring alternative memory allocation algorithms, and integrating with a more

6. Diagram