001

Calculator

Operating Systems

Lab001 Calculator

Building a Bare-Metal ARM Application with UART Communication

Objective
The primary objective of this laboratory is to familiarize students with the fundamentals of bare-metal programming on ARM architectures. By dissecting and understanding the provided code, students will learn how to structure applications across different abstraction layers, implement basic input/output functionalities, and interface directly with ARM hardware components using UART communication.

Overview
In this lab, you will work with a simple ARM bare-metal application designed to add two integers received via UART and display the result. The provided code is organized into multiple files, each serving a specific purpose within the application's architecture. Your task is to analyze, understand, and restructure the code into three distinct layers:

1. **User Level**
2. **Language Library Level**
3. **OS Level (Interface to ARM Hardware)**

Additionally, you will enhance the application's input/output capabilities to support formatted strings akin to printf and scanf, handling data types such as strings, integers, and floats.

Project Structure
The project consists of the following files:

1. **main.c**: Contains the main application logic for adding two numbers.
2. **root.s**: Assembly startup file responsible for initializing the stack and invoking the main function.
3. **string.c & string.h**: Minimal implementation of string manipulation functions.
4. **build_and_run.sh**: Shell script to automate the build and execution process using QEMU within a Docker container.
5. **linker.ld**: Linker script defining the memory layout and entry points.
6. **stdio.h & stdio.c** (to be created): To implement PRINT and READ functionalities similar to printf and scanf.

Layer Breakdown
Understanding the separation of concerns is crucial for developing scalable and maintainable software. The provided code can be logically divided into three layers:

1. User Level
Description:
This is the highest abstraction layer where the application logic resides. It interacts with the language library to perform high-level operations without delving into hardware specifics.
Components:

- **main.c**: Implements the core functionality of the application. It handles user interactions, processes input, performs calculations, and displays output.

Responsibilities:

- Prompting the user for input.
- Parsing and validating input data.
- Performing arithmetic operations.
- Displaying results to the user.

2. Language Library Level
Description:
This intermediate layer provides utility functions that facilitate higher-level operations. It abstracts common tasks such as string manipulation and data conversion, making them reusable across different parts of the application.
Components:

- **string.c & string.h**: Provide a minimal implementation of the strncpy function (my_strncpy), enabling safe string copying without relying on the standard C library.
- **stdio.c & stdio.h** (to be created): Will implement PRINT and READ functions analogous to printf and scanf, handling formatted input and output.

Responsibilities:

- Implementing string manipulation functions.
- Handling formatted input and output.
- Converting data types (e.g., string to integer and vice versa).

3. OS Level (Interface to ARM Hardware)
Description:
This lowest layer interfaces directly with the ARM hardware, managing communication protocols and hardware-specific operations. It abstracts the complexities of hardware interactions, providing simple functions that higher layers can utilize.
Components:

- **os.c & os.h**: Manage UART communication by implementing functions like uart_putc, uart_getc, uart_puts, and uart_gets_input. They also handle data conversions with functions like uart_atoi and uart_itoa. The exposed API will be WRITE and READ methods.
- **root.s**: Assembly code responsible for initializing the stack pointer and invoking the main function.

Responsibilities:

- Initializing hardware components (e.g., UART).
- Sending and receiving data via UART.
- Converting data between different formats.
- Setting up the execution environment (stack initialization).

Assignment Tasks

1. **Analyze the Provided Code:**
    - Understand the purpose and functionality of each file.
    - Identify how the layers interact with each other.

2. **Implement the Language Library Layer (stdio.c & stdio.h):**
    - Develop PRINT and READ functions that mimic the behavior of printf and scanf.
    - Ensure that these functions can handle formatted strings with specifiers for strings (%s), integers (%d), and floats (%f).

3. **Enhance the User Level (main.c):**
    - Modify main.c to utilize the newly implemented PRINT and READ functions for user interactions.
    - Ensure that the application can:
        - Prompt the user to enter two numbers.
        - Read the input numbers.
        - Calculate their sum.
        - Display the result in a formatted manner.

4. **Validate UART Communication:**
    - Ensure that messages sent via PRINT appear correctly in the terminal.
    - Test receiving input via READ and verify accurate data processing.

5. **Document the Layered Architecture:**
    - Provide clear explanations of how each layer interacts.
    - Highlight the flow of data from user input to hardware communication and back.

6. **Build and Execute the Application:**
    - Use the provided build_and_run.sh script to compile and run the application in QEMU.
    - Observe the UART communication to ensure correct functionality.

Deliverables

1. **New and Updated Source Files**
   - Implementing PRINT and READ.
   - main.c: Utilizing the language library for user interactions.

2. **Documentation:**
   - A brief report explaining the separation of layers.
   - Descriptions of how each function operates within its respective layer.

3. **Demonstration:**
   - Screenshots or terminal logs showing successful input and output operations.
   - Evidence of correct arithmetic operations and formatted output.

Expected Outcomes

Upon successful completion of this lab, students will:

- **Understand Layered Architecture:**
  - Grasp the importance of separating concerns across different abstraction layers.
  - Recognize how high-level application logic interacts with hardware interfaces.

- **Implement Basic I/O Functions:**
  - Develop functions analogous to printf and scanf without relying on the standard C library.
  - Handle various data types and ensure safe data transmission over UART.

- **Interface with ARM Hardware:**
  - Manage UART communication effectively, ensuring reliable data exchange.
  - Understand low-level hardware operations and their role in application functionality.

- **Build and Run Bare-Metal Applications:**
  - Utilize toolchains and build scripts to compile and execute bare-metal applications.
  - Employ emulation tools like QEMU to test and debug ARM applications.

Additional Resources

- **ARM Documentation:**
    - ARM Architecture Reference Manual
- **QEMU Documentation:**
    - QEMU ARM Emulation

Tips for Success

- **Start Simple:**
  Begin by ensuring that the PRINT function works correctly with static strings before adding complexity.

- **Incremental Testing:**
  Test each layer individually to isolate and identify issues effectively.

- **Use Debugging Tools:**
  Leverage QEMU's debugging capabilities and consider integrating GDB for step-by-step execution analysis.

- **Collaborate and Discuss:**
  Engage with peers and instructors to clarify doubts and share insights.

- **Stay Organized:**
  Keep your codebase clean and maintain clear documentation of your changes and observations.