Real-Time Embedded Systems, 7.5 hp
Assignment 2

Before you start the Assignment

Check the "Practical Assignments and Project Seminar" learning module on the course page on Blackboard. It includes general information about the assignments and the Lab Kit.

After you complete Assignment

Once you have completed the tasks for Assignment 2, one student in the group will submit the results via the course page on Blackboard.

The assignment submission form is found within the learning module "**Submissions**".

Grading and Deadline

- **Grading Session**: Check the date and time in Time edit for "Grading Ass. 2".
- **Deadline**: By the day before the grading session at **11PM**.

Objectives

- Enhance your Raspberry Pi skills, particularly in input and output operations.
- Experiment with concurrency, observing potential challenges without underlying support.

Content

Assignment 2 includes three parts:

- Part 1 involves getting familiar with a C library for PiFace Control and Display.
- Part 2 involves implementing Taylor Expansion and displaying the results on the PiFace Display.
- Part 3 involves developing a manual interleaving mechanism for managing two tasks.

Q&A

Supervision sessions will address questions, and students are encouraged to share relevant queries or comments on the Discussions on the course page on Blackboard. **Note that posting source code is not allowed.**

Required Equipment and Software:

- Lab kit, including the PiFace Control and Display, along with a breadboard circuit for the LED.
- Raspberry Pi OS.
- For debugging purposes, consider using a serial console software like PuTTY, which works on Windows, macOS, and Linux.

References

- Valvers Bare Metal Programming in C[1]
- Cambridge Baking Pi[2]
- C reference[3] (Recommended by Wagner)
- HD44780U LCD's specification[4]
- Taylor series[5]

[1] http://www.valvers.com/open-software/raspberry-pi/step01-bare-metal-programming-in-cpt1
[2] https://www.cl.cam.ac.uk/projects/raspberrypi/tutorials/os/index.html
[3] https://en.cppreference.com/w/c
[4] https://www.sparkfun.com/datasheets/LCD/HD44780.pdf
[5] https://en.wikipedia.org/wiki/Taylor_series

PART 1 - PiFace Control and Display

The PiFace Control and Display is a display unit capable of showing 16 characters across 2 lines. For more detailed information, please refer to the PiFace Control and Display website[6].

The objective of Part 1 is to get familiar with a C library implemented for the PiFace Control and Display.

Assignment Details

1. Start by downloading ´a2p1.zip´file and extracting its contents.
2. Familiarize yourself with the PiFace C Library, i.e. piface.* files.
   - The PiFace Control and Display utilizes the HD44780 LCD[7] to display alphanumeric characters.
     - This LCD is built on top of an MCP23S17 general-purpose I/O expander, operating in sequential mode for communication with the Raspberry Pi's SPI interface.
     - It functions in 4-bit mode and features two communication channels. One channel is designated for transmitting commands, while the other is used for sending data.
   - Displaying characters on the PiFace's screen involves writing the corresponding values into the expander's data register. This can be accomplished using the provided functions:
     - `static void lcd_write_cmd(uint8_t cmd)`
     - `static void lcd_write_data(uint8_t data)`
   - For example, `lcd_write_cmd` is used to implement commands for clearing the display, moving the cursor and writing characters in the PiFace display. Some of these functions are:
     - `void piface_putc(char c)`
     - `void piface_puts(char s[])`
     - `void piface_clear(void)`
3. Testing PiFace C Library
   - Connect the PiFace Control and Display to the GPIO pins on the Raspberry Pi.
   - Consider `p2p1.c`. It creates a program that continually displays arbitrary text on the LCD and clears it. Boot the `a2p1.expected.img` kernel to visualize it.
   - Compile `p2p1.c` and boot the Raspberry Pi using the newly generated kernel, i.e., ´`a2p1.img`´.

Deliverables

For Assignment 2 Part 1, there are no deliverables, i.e., nothing to submit.

[6] http://www.piface.org.uk/products/piface_control_and_display_2
[7] https://www.sparkfun.com/datasheets/LCD/HD44780.pdf

PART 2 - Taylor expansion of e^x

The goal of this part is to implement the Taylor series expansion for the exponential function e^x. The exponential function can be expressed as:

In practice, the series is truncated after a finite number of terms to approximate e^x within a specified accuracy.

The algorithm uses an iterative approach to compute each term based on the previous one, avoiding redundant calculations of powers and factorials. This results in a linear time complexity of O(n), where n is the number of terms used. The execution time grows proportionally with the number of terms required for the approximation. For example, approximating e^{10} requires significantly more terms than e^1, making it slower to compute.

The implementation developed in this part will be reused in Part 3 of the assignment.

Assignment Details

1. Start by downloading ´a2p2.zip´file and extracting its contents.
2. Connect the PiFace Control and Display to the RPi GPIO.
3. In the expstruct.h file, write the specification for the function iexp(). The specification should include:
   - A brief description of the function.
   - Pre-conditions (e.g., valid input range).
   - Post-conditions (e.g., expected output format).
   - Use the following function signature:

```
struct expStruct {
    int expInt;
    int expFraction;
};

typedef struct expStruct ExpStruct;

ExpStruct* iexp(int);
```

4. In the expstruct.c file, implement the iexp() function with the following requirements:
   - The function should approximate e^x using the Taylor series expansion.
   - The input parameter x represents the exponent and must satisfy:
     - $0 \leq x \leq 20$ (values outside this range are not required to be supported).
     - Internally, the number of terms used in the series should be:
     - Limited to a maximum of 100 terms to prevent excessive computation.
     - Sufficient to achieve two decimal places of accuracy (≈0.01).
   - The function should return a structure containing:
     - The integer part of the result.
     - The fractional part with two digits after the decimal point.
5. In the `a2p2.c` file, write the code for a program that calculates and displays the values of the exponential function for natural numbers, starting from 1 and increasing up to 20.
   - The program must display these values on the PiFace Display connected to the RPi GPIO.
     - You might want try displaying the values via serial console using the TTL USB serial cable.

- Once your code is complete, compile it and boot the Raspberry Pi using the newly generated kernel, i.e., ´`a2p1.img`´.
- The RPi can calculate e^n rather fast. Thus, to ensure that the results are visually observable, you may need to introduce a delay in each iteration of the sum of terms calculation. See the difference by trying the provided `` `a2p1.nodelay.img` `` and `` `a2p1.withdelay.img` `` kernels.
-

Deliverables

For Assignment 2 Part 2, one student in the group must upload the following individual files:

- `expstruct.c`
- `expstruct.h`
- `a2p2.c`

Note:

- All students in the group are equally responsible for the submitted source code.
- The group ensures that the submitted code does not include cheating and plagiarism issues.

PART 3 - Manual Interleaving

Embedded systems are often considered "event-driven," where their primary function is to respond to various "events." But how does the program become aware of changes in the system's environment and respond to these events? Two common approaches are:

- Status-driven using polling (busy waiting)
- Interrupts driven

In Part 3, students will create kernels that execute two tasks (Task A and Task B) according to an offline scheduler. For example, you will build a kernel to execute Task A (e^x) first, then Task B (toggle LED). Given that computing e^x is faster than computing e^{x+1}, the timing between toggling the LED will differ.

In the context of embedded systems or real-time systems, when precise timing is required for certain operations, certain tasks cannot wait for another task to complete. One approach is employing manual interleaving, which is a method of managing and scheduling tasks or processes in a computer program or embedded system where the programmer explicitly controls the order and timing of task execution. It involves designing the program in such a way that different tasks or operations take turns running in a specific sequence.

The objective of Part 3 is to build a kernel that calculates the result e^x while simultaneously blinking an LED at a constant rate.

Please refer to the attached TheoryA2.pdf for a deeper understanding of why an automatic interleaving approach is needed.

Assignment Details

1. Hardware Setup
   - Remove all components connected to the RPi GPIO.
   - Connect the breadboard circuit assessed in Assignment 1 to GPIO16 (pin 36) and GND pin 34.
2. Download ´a2p3.zip´ and Uncompress It
   - Add the C library files you developed to blink the LED into the \lib directory.
   - Add the C library files you developed to compute Taylor expansion of e^x into the \lib directory.
3. Cyclic Execution in a2p3.c
   - Write a program that execute Task A (e^x) first, then Task B (toggle).

```
int i = 1;

// cyclic execution

while (1) {

    iexp(i++);

    led_toggle();

    if (i >= 20)

        i = 1;

}
```

   - You might need to implement the function `led_toggle()` in lib\led.c.

NOTE: If you want to display the result of iexp in the PiFace Control and Display and the LED blinking, use the GPIO extension header to connect it to the RPi GPIO. Of course, you will need to implement the appropriate code.

- Once your code is complete, compile it and boot the RPi using the newly created kernel, i.e., ´`a2p3.img`´.
- As n increases, the time between LED on and off will also increase. The provided `a2p3.cyclic.img` kernel exemplify this behavior.
- Question: how can you make the LED blink at a constant rate?

4. Manual Interleaving
   - After observing the previous results, you may notice that the LED blinks more slowly as the exponential value of the number grows.
     - **NOTE**: If the previous kernel does not exhibit such behavior, try adding some constant delay in the iexp() function.
   - Design and implement a fair interleaving of the tasks (blinking and exponential) to maintain a constant blinking speed.
   - Once your code is complete, compile it and boot the RPi using the newly created kernel, i.e., `a2p2.img`.

Deliverables

For Assignment 2 Part 3, one student in the group must upload the following individual files:

- `expstruct.c`
- `expstruct.h`
- `led.c`
- `a2p3.c`

Note:

- All students in the group are equally responsible for the submitted source code.
- The group ensures that the submitted code does not include cheating and plagiarism issues.