Lab 8 Pre-Lab Submission

Michael Brodskiy

Professor: S. Shazli

March 30, 2023

1. Existing Functions:

   (a) `char *Initialize(int *fd)`

   First and foremost, `Initialize()` points the `fd` pointer at the location of a physical device in memory, and gives it read, write, and synchronization access. Next, an `if` statement checks for the possibility of errors; that is, if the pointer `fd` is equal to -1, then an error is printed, and the program exits with exit code 1 (error). The physical device is then mapped to a virtual device, and given a virtual memory address, using the `mmap()` function. Another `if` statement then checks whether the memory mapping was successful or not; if yes, the virtual memory location of the device is returned. Otherwise, an error is printed, the `fd` pointer connection to memory is closed, and exit code 1 is returned.

   (b) `void Finalize(char *pBase, int fd)`

   The `Finalize()` function checks whether the device attached to the memory address pointer `pBase` is successfully unmapped from memory. If successful, it closes the connection to the device using address `fd`. Otherwise, an error is printed, and exit code 1 is returned.

   (c) `int RegisterRead(char *pBase, unsigned int reg_offset)`

   By combining the base address pointer `pBase` and the device mapping offset value (`reg_offset`), the value that is read from the device at the memory address `pBase` is returned by `RegisterRead()`.

   (d) `void RegisterWrite(char *pBase, unsigned int reg_offset, int value)`

   Similar to `RegisterRead()`, `RegisterWrite` combines the base address pointer `pBase` and the device mapping offset, `reg_offset`, to find the device, and then assigns a specified value to that address.

2. Writing a switch read function:

Listing 1: Switch Reading Code

```
/**Reads the value of a switch
 * -Uses base address of I/O
 * @param pBaseBase address returned by'mmap'
 * @param switchNumSwitch number (0 to 9)
 * @returnSwitch value read
 */
int Read1Switch(char * pBase, int switchNum) {

    // Read the switch register
    int switchRegisterValue = RegisterRead(pBase, SW_BASE);

    // Mask the value to extract the specified switch bit
    int switchBitMask = 1 << switchNum;

    // if the result is non-zero, then the switch is on, off
       otherwise
    int switchValue;
    //use the bitwise AND operator and compare result against
       the bit mask
    if (switchRegisterValue & switchBitMask == switchBitMask)
       {
        switchValue = 1;
    } else {
        switchValue = 0;
    }

    return switchValue;

}
```

3. Writing a switch write function:

Listing 2: Switch Writing Code

```
/** Changes the state of an LED (ON or OFF)
 *  @param pBase Base address returned by 'mmap'
 *  @param ledNum LED number (0 to 9)
 *  @param state State to change to (ON or OFF)
 */
void Write1Led(char *pBase,int ledNum, int state) {

    if (ledNum < 0 || ledNum > 9) {
        cout << "ERROR: Invalid LED number. Only LED numbers 0
            to 9 are valid." << endl;
        return;

    }

    // Read the LED redister
    int ledRegisterValue = RegisterRead(pBase, LEDR_BASE);

    // Set the state of the specified LED based on the state
       parameter
    if (state == 1) {
        ledRegisterValue = (1 << ledNum);
    } else {
        ledRegisterValue &= ~(1 << ledNum);
    }

    // Write the new LED register value
    RegisterWrite(pBase, LEDR_BASE, ledRegisterValue);

    // Read the LED register again and print out the value to
       double check (this can be commented out)
    int UpdatedRegisterValue = RegisterRead(pBase, LEDR_BASE);
    cout << "LED Register Updated Value: " <<
       UpdatedRegisterValue << endl << endl;

}
```