Jaden Williams

SNHU – CS350

6/8/25

Milestone Three

### **1. Why does the loop that processes the LED blinking need to run in a separate thread?**

The LED blinking loop runs in a separate thread so that it can operate independently of the main program flow — particularly the button press detection and other background tasks. Without a separate thread, the main program would be blocked while the message is being transmitted in Morse code, and it wouldn’t be responsive to button presses or other interactions. Multithreading allows the program to **simultaneously send Morse code** and **monitor user input**, such as toggling between "SOS" and "OK".

### **2. What is the purpose of returning to the off state after each completed state action?**

Returning to the off state ensures that:

* The LEDs are turned off between signals (dots and dashes).
* Transitions between Morse elements (dot/dash, letter, word) are clear and well-defined.
* The state machine remains modular and predictable by always moving from off to the next required state.

This design mirrors how Morse code is transmitted: a signal (on) followed by a period of silence (off). It simplifies transitions and avoids keeping the LEDs in a potentially undefined state.

### **3. How could you integrate serial communications to facilitate changing the messages available to the program?**

You could integrate serial communication using Python’s pyserial library. The Raspberry Pi can listen on a serial port (e.g., /dev/ttyS0) for new input. Here’s how this might be used:

* Open a serial port for reading using serial.Serial().
* Wait for incoming strings and parse them as new messages.
* Update the activeMessage variable in the CWMachine with the new content.
* Use a mutex or similar thread-safe method to prevent race conditions when updating the message.

This would allow a user connected via serial terminal (e.g., PuTTY or Arduino Serial Monitor) to dynamically send new messages without modifying the code.

### **4. How could you use the 16x2 display to provide debugging information to the user when they don’t have access to the application console?**

You could display key debug information on the 16x2 LCD, such as:

* The currently active message ("Sending: SOS" or "Sending: OK").
* The current Morse symbol being transmitted (e.g., "Symbol: .").
* The current state (e.g., "State: dot").
* Button press detection ("Button Pressed!").
* Error messages or status updates.

This would involve calling the updateScreen() method of the ManagedDisplay class at key points in the program, replacing or alternating the main message with debug info — potentially using a secondary display mode or scrolling text if space is limited.