Assignment 2: Dining Philosophers

The goals for this assignment are as follows

1. Familiarize yourself with the Python threading library.
    - Launching multiple threads
    - Sharing locks between threads
2. Implement LED blinking capabilities
3. Use button interrupts for killing threads

Problem

Five philosophers are dining together at a table with five forks. Each philosopher shares their forks with neighboring philosophers and needs both forks (left and right) to eat. When a philosopher is done eating, it relinquishes the forks and takes a nap. Finally, when the philosopher is finished with the nap, it will wait, starving, for the two pairs of forks (left and right) to be freed to eat.

Thus, there are 3 possible states for each philosopher

1. EATING: the philosopher has both forks (left and right)
2. NAPPING: the philosopher is finished eating
3. STARVING: the philosopher is waiting to have both forks (left and right)

Part A2.1:

- Write code for the dining philosophers problem. Use five LEDs, one for each philosopher, and five locks for forks. The five LEDs will be the four on-board green LEDs above the buttons and one of the on-board RGB LEDs that we saw in Lab1 (make it green to match the other LEDs).
    - The first hurdle I hit with this was finding what the RGB LED was figuring out how it was being referenced in base. I totally missed that we used it in Lab 1 and ended up figuring out that LD4 on the board can be accessed by base.rgbleds[4]. However since we are only using one of the RGB LEDs, I went back and updated it to just reference the one similar to how Lab 1 did it

```
import pynq.lib.rgbled as rgbled

led4 = rgbled.RGBLED(4)

# Previously base.rgbleds[4]
```

- I also reused the blink code from the lab. I added logic to incorporate LED4. One thing that was interesting was that the original blink code toggles the LED, but there is no toggle function for the RGB LED. So for the RGB LED I used t/2 in the for loop to imitate the toggle

```
def blink(t, d, n):
    '''
    Function to blink the LEDs
    Params:
      t: number of times to blink the LED
      d: duration (in seconds) for the LED to be on/off
      n: index of the LED (0 to 4)
    '''
    if n == 4:
        for i in range(int(t/2)):
            led4.on(2)
            time.sleep(d)
        led4.off()
        time.sleep(d)
    else:
      for i in range(t):
        base.leds[n].toggle()
        time.sleep(d)
      base.leds[n].off()
```

- Find appropriate durations for the philosophers to be eating and napping. Consider choices such that your threads do not go into a constant starvation. (i.e., should napping time be greater than or less than eating time?)
  - Napping time should be greater than eating time. I came to this conclusion by thinking about the thread behavior. If napping time is less than eating time, one philosopher could eat, nap quickly, and then steal the resources to start eating before another philosopher is able to grab the resources. This could result in starvation, where one philosopher is constantly eating due to the short nap time.
  - As for duration, it's more about the ratio of the times vs the time. However I will just plan to set the eating time to 1s and adjust the other times to that
- When one of the philosophers is eating, both forks are used by that philosopher, and the LED should blink at a higher rate to indicate "eating".

```
blink(50, 0.02, idx)
```

- When a philosopher is napping, the LED should blink at a lower rate to indicate "napping".

```
blink(15,0.1, idx)
```

- When a philosopher is waiting for forks, its LED should be off to indicate "starving".
  - Pretty straight forward, basically do nothing. The other states will handle turning the LED off so this state will just wait (Unless there is some edge case, where then this state will just turn off the LED once)
- The code must run forever. To terminate the program, you have to use push buttons.
  - For terminating the program, we can think of the buttons as the interrupt. There is an Event class in the threading package that does just that. The idea is to create an Event and map it to any button press. The philosopher will then keep eating, thinking, napping in a while loop that monitors that event. When the philosopher sees the Event, it will exit, that thread will join, and once all threads join the program will exit.

```
def button_monitor_t():
   while not stop_event.is_set():
       if btns.read() != 0:
           stop_event.set()
```

Developing Notes:

- I used a for loop to create the locks and to assign the locks to each philosopher. This just keeps the code cleaner. For assigning the locks to the philosophers, my logic was to have everyone except for one person take from the left side first, with the last person taking from the right side first

```
threads = []

locks = []
for i in range(5):
   locks.append(threading.Lock())

for i in range(4):
   t = threading.Thread(target=philosopher_t, args=(locks[i], locks[i+1], i))
   threads.append(t)
   t.start()

t = threading.Thread(target=philosopher_t, args=(locks[0], locks[4], 4))
threads.append(t)
t.start()
```

- Besides that set up, I just added the logic in the philosophers thread.
- The logic is to grab one resource, then try to grab the other resource. Once both are acquired, eat for a period of time, then release both of the resources. Once done eating, resources are released, then the philosopher naps. Once the nap is complete, the philosopher starves until it is able to grab both resources again.
- I tested with both nap being longer and nap being shorter and eating, and both resulted in fair share between all philosophers.

```
def philosopher_t(_l0, _l1, num):
   while not stop_event.is_set():
       has_l0 = False
       has_l1 = False

       if has_l1:
           _l1.release()
           has_l1 = False

       has_l0 = _l0.acquire(True)
       print("Philosopher {} has one chopstick".format(num))

       has_l1 = _l1.acquire(True)
       print("Philosopher {} is eating".format(num))

       blink(50, 0.02, num)

       if has_l0:
        _l0.release()
        has_l0 = False
    if has_l1:
        _l1.release()
        has_l1 = False

    print("Philosopher {} is napping".format(num))

    blink(5,0.1, num)
```

Part A2.2:

- In this part, you use the random library to generate random numbers for the eating and napping states. By using random.randint(a, b) you can get a random number between a and b.
- You have to set the boundaries for your random number (a, b) such that napping is not longer than eating, and therefore your threads do not go to a constant starvation.
- I chose eating to be in the range of 0.8-1.4s and napping to be in the range 1.4-2s

```
eat_time = random.randint(40,70)
nap_time = random.randint(14,20)

blink(eat_time, 0.02, num)
blink(nap_time, 0.1, num)
```

Video: https://youtu.be/H1NXhjz7i5w

GitHub: https://github.com/cxu4426/WES237A-hw

Deliverables

Each student must submit the following **individually**

1. A PDF report detailing your workflow and the relevant Jupyter notebook cells relating to your progress.
    - Your report should detail your workflow throughout the assignment. Be sure to discuss **any** difficulties or troubles you encountered and your troubleshooting procedure. Also, detail your thought process which led to the design and implementation of the code. For example, describe your top-down design methodology (i.e., how did you split the large task into smaller, more incremental jobs? How were you able to test each of these smaller parts?)
    - Please also include a brief discussion on the timings for EATING, NAPPING, and STARVING you chose to avoid a deadlock.
2. Your complete Jupyter notebook, downloaded as a PDF **attached at the very end of your report**
    - You can do this by selecting 'File -> Print Preview', then printing to PDF from the browser.
    - Use a PDF stitching tool like pdfjoiner to join your Report and Jupyter Notebook into a single PDF file
3. Video demonstration of your code working on the PYNQ board. Please limit videos to 60 seconds and upload them to a video-sharing site, and include the link in your PDF report.

Each team should submit the following: **one per team**

1. All relevant code (.ipynb, .py, .cpp, .c, etc. files) pushed to your team's git repo.

Part 1

In [1]:
```
import threading
import time
from pynq.overlays.base import BaseOverlay
base = BaseOverlay("base.bit")
import pynq.lib.rgbled as rgbled
import random

led4 = rgbled.RGBLED(4)
btns = base.btns_gpio
```

In [2]:
```
stop_event = threading.Event()

def blink(t, d, n):
    '''
    Function to blink the LEDs
    Params:
      t: number of times to blink the LED
      d: duration (in seconds) for the LED to be on/off
      n: index of the LED (0 to 4)
    '''
    if n == 4:
        for i in range(int(t/2)):
            led4.on(2)
            time.sleep(d)
            led4.off()
            time.sleep(d)
    else:
        for i in range(t):
            base.leds[n].toggle()
            time.sleep(d)
        base.leds[n].off()

def button_monitor_t():
    while not stop_event.is_set():
        if btns.read() != 0:
            stop_event.set()

def philosopher_t(_l0, _l1, num):
    while not stop_event.is_set():
        has_l0 = False
        has_l1 = False

        if has_l1:
            _l1.release()
            has_l1 = False

        has_l0 = _l0.acquire(True)
        print("Philosopher {} has one chopstick".format(num))

        has_l1 = _l1.acquire(True)
        print("Philosopher {} is eating".format(num))
        blink(50, 0.02, num)

        if has_l0:
            _l0.release()
            has_l0 = False
        if has_l1:
            _l1.release()
            has_l1 = False

        print("Philosopher {} is napping".format(num))

        blink(15,0.1, num)


# Initialize and launch the threads
threads = []

locks = []
for i in range(5):
    locks.append(threading.Lock())

for i in range(4):
    t = threading.Thread(target=philosopher_t, args=(locks[i], locks[i+1], i))
    threads.append(t)
    t.start()

t = threading.Thread(target=philosopher_t, args=(locks[0], locks[4], 4))
threads.append(t)
t.start()

button_thread = threading.Thread(target=button_monitor_t)
button_thread.start()


for t in threads:
    name = t.name
    t.join()
    print('{} joined'.format(name))

button_thread.join()
print("button thread joined")

# clean up code
```

```
Philosopher 0 has one chopstick
Philosopher 0 is eating
Philosopher 2 has one chopstick
Philosopher 2 is eating
Philosopher 0 is nappingPhilosopher 1 has one chopstick

Philosopher 4 has one chopstick
Philosopher 4 is eating
Philosopher 2 is napping
Philosopher 1 is eating
Philosopher 3 has one chopstick
Philosopher 4 is nappingPhilosopher 3 is eating

Philosopher 1 is napping
Philosopher 0 has one chopstick
Philosopher 0 is eating
Philosopher 2 has one chopstick
Philosopher 3 is napping
Philosopher 2 is eating
Philosopher 0 is napping
Philosopher 4 has one chopstick
Philosopher 4 is eating
Philosopher 1 has one chopstick
Philosopher 2 is napping
Philosopher 1 is eating
Philosopher 4 is napping
Philosopher 3 has one chopstick
Philosopher 3 is eating
Philosopher 0 has one chopstick
Philosopher 1 is napping
Philosopher 0 is eating
Philosopher 3 is napping
Philosopher 2 has one chopstick
Philosopher 2 is eating
Philosopher 0 is napping
Philosopher 4 has one chopstick
Philosopher 4 is eating
Philosopher 2 is napping
Philosopher 1 has one chopstick
Philosopher 1 is eating
Philosopher 3 has one chopstick
Philosopher 4 is napping
Philosopher 3 is eating
Philosopher 1 is napping
Philosopher 0 has one chopstick
Philosopher 0 is eating
Philosopher 2 has one chopstick
Philosopher 3 is napping
Philosopher 2 is eating
Philosopher 0 is napping
Philosopher 4 has one chopstick
Philosopher 4 is eating
Philosopher 1 has one chopstick
Philosopher 2 is napping
Philosopher 1 is eating
Philosopher 4 is napping
Thread-5 (philosopher_t) joined
Philosopher 1 is napping
Thread-6 (philosopher_t) joined
Thread-7 (philosopher_t) joined
```

```
Thread-8 (philosopher_t) joined
Thread-9 (philosopher_t) joined
button thread joined
```

Part 2: random sleep times

In [3]:
```
stop_event = threading.Event()

def blink(t, d, n):
    '''
    Function to blink the LEDs
    Params:
      t: number of times to blink the LED
      d: duration (in seconds) for the LED to be on/off
      n: index of the LED (0 to 4)
    '''
    if n == 4:
        for i in range(int(t/2)):
            led4.on(2)
            time.sleep(d)
            led4.off()
            time.sleep(d)
    else:
        for i in range(t):
            base.leds[n].toggle()
            time.sleep(d)
        base.leds[n].off()

def button_monitor_t():
    while not stop_event.is_set():
        if btns.read() != 0:
            stop_event.set()

def philosopher_t(_l0, _l1, num):
    while not stop_event.is_set():
        has_l0 = False
        has_l1 = False

        eat_time = random.randint(40,70)
        nap_time = random.randint(14,20)

        if has_l1:
            _l1.release()
            has_l1 = False

        has_l0 = _l0.acquire(True)
        print("Philosopher {} has one chopstick".format(num))

        has_l1 = _l1.acquire(True)
        print("Philosopher {} is eating".format(num))

        blink(eat_time, 0.02, num)

        if has_l0:
            _l0.release()
            has_l0 = False
        if has_l1:
            _l1.release()
            has_l1 = False

        print("Philosopher {} is napping".format(num))

        blink(nap_time, 0.1, num)


# Initialize and launch the threads
threads = []

locks = []
for i in range(5):
    locks.append(threading.Lock())

for i in range(4):
    t = threading.Thread(target=philosopher_t, args=(locks[i], locks[i+1], i))
    threads.append(t)
    t.start()

t = threading.Thread(target=philosopher_t, args=(locks[0], locks[4], 4))
threads.append(t)
t.start()

button_thread = threading.Thread(target=button_monitor_t)
button_thread.start()


for t in threads:
    name = t.name
    t.join()
    print('{} joined'.format(name))

button_thread.join()
print("button thread joined")

# clean up code
```

```
Philosopher 0 has one chopstick
Philosopher 0 is eating
Philosopher 2 has one chopstick
Philosopher 2 is eating
Philosopher 0 is nappingPhilosopher 1 has one chopstick
Philosopher 4 has one chopstick
Philosopher 4 is eating

Philosopher 2 is napping
Philosopher 1 is eating
Philosopher 3 has one chopstick
Philosopher 4 is napping
Philosopher 3 is eating
Philosopher 1 is napping
Philosopher 0 has one chopstick
Philosopher 0 is eating
Philosopher 2 has one chopstick
Philosopher 3 is napping
Philosopher 2 is eating
Philosopher 0 is nappingPhilosopher 4 has one chopstick
Philosopher 4 is eating

Philosopher 1 has one chopstick
Philosopher 4 is napping
Philosopher 2 is napping
Philosopher 1 is eating
Philosopher 3 has one chopstick
Philosopher 3 is eating
Philosopher 0 has one chopstick
Philosopher 3 is napping
Philosopher 1 is napping
Philosopher 0 is eating
Philosopher 2 has one chopstick
Philosopher 2 is eating
Philosopher 0 is napping
Philosopher 2 is napping
Philosopher 4 has one chopstick
Philosopher 4 is eating
Philosopher 3 has one chopstick
Philosopher 1 has one chopstick
Philosopher 1 is eating
Philosopher 4 is napping
Philosopher 3 is eating
Thread-11 (philosopher_t) joined
Philosopher 1 is napping
Philosopher 3 is napping
Thread-12 (philosopher_t) joined
Thread-13 (philosopher_t) joined
Thread-14 (philosopher_t) joined
Thread-15 (philosopher_t) joined
button thread joined
```