Title: WES_237A_assignment2_report

PID: A69028485

Name: William Wei

GitHub username: William Lien Chin Wei

Email (registered with GitHub): williams19834812@gmail.com

Video demonstration of your working code on Youtube: https://youtu.be/KiP4Fp8HPQA (https://youtu.be/KiP4Fp8HPQA)

Assignments Rubric checklist

(**Done**) Report Submitted?
(**Done**) Video Uploaded?
(**Done**) Code Pushed to Github?
(**Yes**) Does the video demonstration show correct execution?
(**Yes**) Is the submitted code correct?
(**Well-done**) How well does the report outlines the design of the code?
(**Yes in youtube video**)How well does the report describe the results?
(**Yes**) Does the Report detail the student's grasp on the goals/objectives of the assignment?

The goals for this assignment are as follows

1. Familiarlize 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

There are five philosophers dining together at 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 in order 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)

In [1]:

```
import threading
import time
import asyncio
from pynq.overlays.base import BaseOverlay
import pynq.lib.rgbled as rgbled

base = BaseOverlay("base.bit")
btns = base.btns_gpio
LED4 = rgbled.RGBLED(4)
```

In [2]:

```
## Varialbes definition
nP = 5  # number of philosophers
f_e, f_n = 1/0.05, 1/0.2 # eating frequency, napping frequency, Hz
d_e, d_n = 1/f_e,  1/f_n # eating duration, napping duration, second
t_e, t_n, t_w = 3, 2, 1  # eating, napping, waiting time, sec
```

Part A2.1:

• Write code for dining philosophers problem. Use five LEDs, one for each philosopher and five locks for forks. The five LEDs will the 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 other other LEDs**).

In [3]:

```
# blink the LEDs
def blink(t, d, n):
    '''
      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 nP)
    '''
    if n in range(0, nP-1): #four on-board green LEDs above the butto
        for i in range(t):
            base.leds[n].toggle()
            time.sleep(d)

    elif n == nP-1:         #one of the on-board  RGB LEDs
        for i in range(int(t*0.5)):
            LED4.write(0x2)             #0x2, green
            time.sleep(d)
            LED4.write(0x0)   #0x0, black
            time.sleep(d)
    setLEDoff(n)

# turn off a LED
def setLEDoff(n):
    if n in range(0, nP-1):
        base.leds[n].off()
    elif n == nP-1:
        LED4.write(0x0) #0x0, black
```

In [4]:

```
## Testing blinking LEDs (philosophers)
for i in range(nP):
    blink(int(t_e*f_e), d_e, i)

print("Test is done!")
```

```
Test is done!
```

• **Find appropriate durations** for the philosophers to be eating and napping. Consider choices such that your **threads do not go to a constant starvation.** (i.e. should napping time be greater than or less than eating time?)
Given that that Napping time is like a transition after Eating, Napping time should be shorter than the Eating time so that another Waiting (Starving) thread can quickly hold the available resources, i.e., forks, to do Eating.

The timings for EATING, NAPPING, and STARVING I chose in order to avoid a deadlock are 3 seconds eating, 2 seconds napping, 1 seconds waiting time in Part A2.2. As conceptually, eating is the main task which is assigned the longest time, waiting is actually starving time of 2 seconds, preparing each philosopher to take two forks. For napping, it looks like a transition time after eating and releasing forks.

In [5]:

```
## Test appropriate duration
print("eating...")
for i in range(nP): # 3 sec for each LED
    blink(int(t_e*f_e), d_e, i)
print("done with eating...")

print("napping")
for i in range(nP): # 2 sec for each LED
    blink(int(t_n*f_n), d_n, i)
print("done with napping")
```

```
eating...
done with eating...
napping
done with napping
```

• When one of the philosophers is eating, **both forks** is used by that philosopher and its **LED should blink with a higher rate** to indicate “eating”.

• When a philosopher is napping, its **LED should blink with a lower rate** to indicate “napping”.

• When a philosopher is **waiting for forks**, its LED should be **off** to indicate “starving”.
I implemented five Locks as five forks along with each thread, philosopher. Below is a snippet of demonstration of running three runs.

In [6]:

```
## test run
# philosopher function for waiting, eating, and napping
def ph_t(_lfk, num):
    #global t_w, t_e, t_n
    '''
    _lfk: threading lock list   (resource)
    i_fk_left: index of a fork on the LHS of a philosopher
    i_fk_right: index of a fork on the RHS of a philosopher
    num: index representing thread number (philosopher).
    '''
    i_fk_left = num
    i_fk_right = (num+1)%nP
    _lL = _lfk[i_fk_left]
    _lR = _lfk[i_fk_right]

    holding_left_fk = False
    holding_right_fk = False

    for i in range(3): # 3 runs
    #while True:
        print("philosophser {} is waiting for forks {} sec...\n".forma
        setLEDoff(num)
        time.sleep(t_w)

        holding_left_fk = _lL.acquire(False)
        holding_right_fk = _lR.acquire(False)

        if holding_right_fk:
            print("philosophser {} is holding right fork...\n".format
        if holding_left_fk:
            print("philosophser {} is holding left fork...\n".format(


        if (holding_right_fk and holding_left_fk): # have both forks
            print("philosophser {} is eating for {} sec...\n".format(
            blink(int(t_e*f_e), d_e, num)

            _lR.release()
            holding_right_fk = False
            _lL.release()
            holding_left_fk = False

            print("philosophser {} is napping... for {} sec\n".format
            blink(int(t_n*f_n), d_n, num)

        if (holding_right_fk):
            print("philosophser {} is releasing right fork...\n".forma
            _lR.release()
        if (holding_left_fk):
            print("philosophser {} is releasing left fork...\n".forma
            _lL.release()

        time.sleep(0) # yeild
    print("philosopher {} is done.\n".format(num))
```

In [7]:
```
# Initialize forks and launch the threads
forks = []    # forks
for i in range(nP):
    forks.append(threading.Lock())

threads = [] # philosophers
for i in range(nP):
    t = threading.Thread(target=ph_t, args=(forks, i,))
    threads.append(t)

for t in threads: # launch threads
    t.start()

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

```
philosophser 0 is waiting for forks 1 sec...

philosophser 1 is waiting for forks 1 sec...

philosophser 2 is waiting for forks 1 sec...

philosophser 3 is waiting for forks 1 sec...

philosophser 4 is waiting for forks 1 sec...

philosophser 0 is holding right fork...

philosophser 0 is holding left fork...

philosophser 0 is eating for 3 sec...

philosophser 1 is holding right fork...

philosophser 1 is releasing right fork...
```

- The code must run forever. To terminate the program, you have to use **push buttons**.

I implemented a separate thread "bT = threading.Thread(target=get_btns, args=())" handling the button detection, get_btns(),

In [8]:

```
status = True

def ph_t_2(_lfk, num):
    global status
    '''
    Worker function to try and acquire resource and blink the LED
    _lfk: threading lock list   (resource)
    i_fk_left: index of a fork on the LHS of a philosopher
    i_fk_right: index of a fork on the RHS of a philosopher
    num: index representing thread number (philosopher).
    '''
    i_fk_left = num
    i_fk_right = (num+1)%nP
    _lL = _lfk[i_fk_left]
    _lR = _lfk[i_fk_right]

    holding_left_fk = False
    holding_right_fk = False

    #for i in range(10): # 10 runs
    while status:
        print("philosophser {} is waiting for forks {} sec...\n".forma
        setLEDoff(num)
        time.sleep(t_w)

        holding_left_fk = _lL.acquire(False)
        holding_right_fk = _lR.acquire(False)

        '''
        if holding_right_fk:
            print("philosophser {} is holding right fork...\n".format
        if holding_left_fk:
            print("philosophser {} is holding left fork...\n".format(n
        '''

        if (holding_right_fk and holding_left_fk): # have both forks
            print("philosophser {} is eating for {} sec...\n".format(n
            blink(int(t_e*f_e), d_e, num)

            _lR.release()
            holding_right_fk = False
            _lL.release()
            holding_left_fk = False

            print("philosophser {} is napping... for {} sec\n".format
            blink(int(t_n*f_n), d_n, num)

        if (holding_right_fk):
            #print("philosophser {} is releasing right fork...\n".form
            _lR.release()
        if (holding_left_fk):
            #print("philosophser {} is releasing left fork...\n".forma
            _lL.release()

        time.sleep(0) # yeild
    print("philosopher {} is done.\n".format(num))
```

```
def get_btns():
    global status
    while status:
        time.sleep(0.01)
        if btns.read() == 1: #terminate the program
            print("pressing BTN0, terminating the program...\n")
            status = False
```

In [9]:

```
# Initialize forks and launch the threads
forks = []    # forks

for i in range(nP):
    forks.append(threading.Lock())

threads = [] # philosophers
for i in range(nP):
    t = threading.Thread(target=ph_t_2, args=(forks, i,))
    threads.append(t)

# separate thread handling the button detection, causing program termi
bT = threading.Thread(target=get_btns, args=())
bT.start()

for t in threads: # launch threads
    t.start()

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

bT.join()
print('{} for button detection joined\n'.format(bT.getName()))
```

```
philosophser 0 is waiting for forks 1 sec...

philosophser 1 is waiting for forks 1 sec...

philosophser 2 is waiting for forks 1 sec...

philosophser 3 is waiting for forks 1 sec...
philosophser 4 is waiting for forks 1 sec...


philosophser 0 is eating for 3 sec...

philosophser 1 is waiting for forks 1 sec...

philosophser 2 is eating for 3 sec...

philosophser 4 is waiting for forks 1 sec...

philosophser 3 is waiting for forks 1 sec...
```

Part A2.2:

• In this part, you use **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. The implementation using random function with defined boundaries is shown below.

In [10]:

```
import random

tLower, tMiddle, tUpper = 1, 4, 6 # boundary, seconds
t_n = random.randint(tLower, tMiddle) # eating time
t_e = random.randint(tMiddle, tUpper) # napping time
print("The napping and eating times are {} and {} seconds, respective
```

```
The napping and eating times are 2 and 4 seconds, respectively.
```

In [11]:

```
## Varialbes definition
nP = 5  # number of philosophers
f_e, f_n = 1/0.05, 1/0.2 # eating frequency, napping frequency, Hz
d_e, d_n = 1/f_e,  1/f_n # eating duration, napping duration, second
t_w = 1                  # waiting time, sec
```

In [12]:

```
status = True

def ph_t_2(_lfk, num):
    global status
    '''
    Worker function to try and acquire resource and blink the LED
    _lfk: threading lock list   (resource)
    i_fk_left: index of a fork on the LHS of a philosopher
    i_fk_right: index of a fork on the RHS of a philosopher
    num: index representing thread number (philosopher).
    '''
    i_fk_left = num
    i_fk_right = (num+1)%nP
    _lL = _lfk[i_fk_left]
    _lR = _lfk[i_fk_right]

    holding_left_fk = False
    holding_right_fk = False

    #for i in range(10): # 10 runs
    while status:
        print("philosophser {} is waiting for forks {} sec...\n".forma
        setLEDoff(num)
        time.sleep(t_w)

        holding_left_fk = _lL.acquire(False)
        holding_right_fk = _lR.acquire(False)

        '''
        if holding_right_fk:
            print("philosophser {} is holding right fork...\n".format
        if holding_left_fk:
            print("philosophser {} is holding left fork...\n".format(
        '''

        if (holding_right_fk and holding_left_fk): # have both forks
            print("philosophser {} is eating for {} sec...\n".format(
            blink(int(t_e*f_e), d_e, num)

            _lR.release()
            holding_right_fk = False
            _lL.release()
            holding_left_fk = False

            print("philosophser {} is napping... for {} sec\n".format
            blink(int(t_n*f_n), d_n, num)

        if (holding_right_fk):
            #print("philosophser {} is releasing right fork...\n".for
            _lR.release()
        if (holding_left_fk):
            #print("philosophser {} is releasing left fork...\n".forma
            _lL.release()

        time.sleep(0) # yeild
    print("philosopher {} is done.\n".format(num))
```

```python
def get_btns():
    global status
    while status:
        time.sleep(0.01)
        if btns.read() == 1: #terminate the program
            print("pressing BTN0, terminating the program...\n")
            status = False
```

In [13]:

```python
# Initialize forks and launch the threads
forks = []    # forks

for i in range(nP):
    forks.append(threading.Lock())

threads = [] # philosophers
for i in range(nP):
    t = threading.Thread(target=ph_t_2, args=(forks, i,))
    threads.append(t)

# separate thread handling the button detection, causing program termi
bT = threading.Thread(target=get_btns, args=())
bT.start()

for t in threads: # launch threads
    t.start()

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

bT.join()
print('{} for button detection joined\n'.format(bT.getName()))
```

```
philosophser 0 is waiting for forks 1 sec...
philosophser 1 is waiting for forks 1 sec...


philosophser 2 is waiting for forks 1 sec...

philosophser 3 is waiting for forks 1 sec...

philosophser 4 is waiting for forks 1 sec...

philosophser 1 is eating for 4 sec...

philosophser 2 is waiting for forks 1 sec...

philosophser 0 is waiting for forks 1 sec...

philosophser 4 is eating for 4 sec...

philosophser 3 is waiting for forks 1 sec...
```

```
In [ ]:
```