Benediction Bora
WES 237A: Introduction to Embedded System Design (Winter 2026)
Lab 2: Process and Thread
Due: 1/19/2026 11:59pm

In order to report and reflect on your WES 237A labs, please complete this Post-Lab report by the end of the weekend by submitting the following 2 parts:

- Upload your lab 2 report, composed by a single PDF that includes your in-lab answers to the bolded questions in the Google Doc Lab and your Jupyter Notebook code.
- Answer two short essay-like questions on your Lab experience.

All responses should be submitted to Canvas. Please also be sure to push your code to your git repo as well.

Create Lab2 Folder

1. Create a new folder on your PYNQ jupyter home and rename it ‘Lab2’

Shared C++ Library

1. In ‘Lab2’, create a new text file (New -> Text File) and rename it to ‘main.c’
2. Add the following code to ‘main.c’:

```
#include <unistd.h>

int myAdd(int a, int b){
     sleep(1);
     return a+b;
}
```

3. **Following the function above, write another function to multiply two integers together. Copy your code below.**

```
int multiply(int a, int b)
{
   sleep(1);
   return a*b;
}
```

4. Save main.c
5. In Jupyter, open a terminal window (New -> Terminal) and *change directories* (cd) to ‘Lab2’ directory.

```
$ cd Lab2
```

6. Compile your ‘main.c’ code as a shared library.

```
$ gcc -c -Wall -Werror -fpic main.c
$ gcc -shared -o libMyLib.so main.o
```

7. Download ‘ctypes_example.ipynb’ from here and upload it to the Lab2 directory.
8. Go through each of the code cells to understand how we interface between Python and our C code
9. **Write another Python function to wrap your multiplication function written above in step 3. Copy your code below.**

```
def multiply(a, b):

    return _libInc.multiply(a,b)
```

To summarize, we created a C shared library and then called the C function from Python

Multiprocessing

1. Download ‘multiprocess_example.ipynb’ from here and upload it to your ‘Lab2’ directory.
2. Go through the documentation (and comments) and answer the following question
    a. **Why does the ‘Process-#’ keep incrementing as you run the code cell over and over?**

Its keeping count for the number of processes being created on CPU each time

    b. **Which line assigns the processes to run on a specific CPU?**

os.system("taskset -p -c {} {}".format(0, p1.pid))

3. In ‘main.c’, change the ‘sleep()’ command and recompile the library with the commands above. Also, reload the Jupyter notebook with the ⟳ symbol and re-run all cells. Play around with different sleep times for both functions.
    a. **Explain the difference between the results of the ‘Add’ and ‘Multiply’ functions and when the processes are finished.**

The differences in time between add and multiply can be used to schedule execution of each function. We can delay or quicken each function for a specific amount of delay

4. Continue to the lab work section. Here we are going to do the following
    a. Create a multiprocessing array object with 2 entries of integer type.
    b. Launch 1 process to compute addition and 1 process to compute multiplication.
    c. Assign the results to separate positions in the array.
        i. Process 1 (add) is stored in index 0 of the array (array[0])
        ii. Process 2 (mult) is stored in index 1 of the array (array[1])
    d. Print the results from the array.
    e. **There are 4 TODO comments that must be completed**
5. Answer the following question
    a. **Explain, in your own words, what shared memory is in relation to the code in this exercise.**

Shared memory is like a bank account with multiple people with bank cards to access the account.

Threading

1. Download ‘threading_example.ipynb’ from here and upload it into your ‘Lab2’ directory.
2. Go through the documentation and code for ‘Two threads, single resource’ and answer the following questions
    a. **What line launches a thread and what function is the thread executing?**

t = threading.Thread(target=worker_t, args=(fork, i)), the thread is executing worker_t()

    b. **What line defines a mutual resource? How is it accessed by the thread function?**

fork = threading.Lock(), its accessed through Thread, as an argument args=(fork, i)

3. Answer the following question about the ‘Two threads, two resources’ section.
    a. **Explain how this code enters a deadlock.**

The threads enter a deadlock because they are being locked because resource1 is being used before assigned and resource 0 is being released before execution. So both resources will never run

4. Complete the code using the non-blocking acquire function.
    a. **What is the difference between ‘blocking’ and ‘non-blocking’ functions?**

A blocking function will not exit or return until the thread is complete and a non-blocking function returns when it can’t complete the thread

5. BONUS:
   Can you explain why this is used in the ‘Two threads, two resources’ section:

```
if using_resource0:
    _l0.release()
if using_resource1:
    _l1.release()
```

The above code snippet is used to check whether a resource is being used, release the lock.