**WES 237A: Introduction to Embedded System Design (Winter 2024)**

**Lab 3: Serial and CPU**

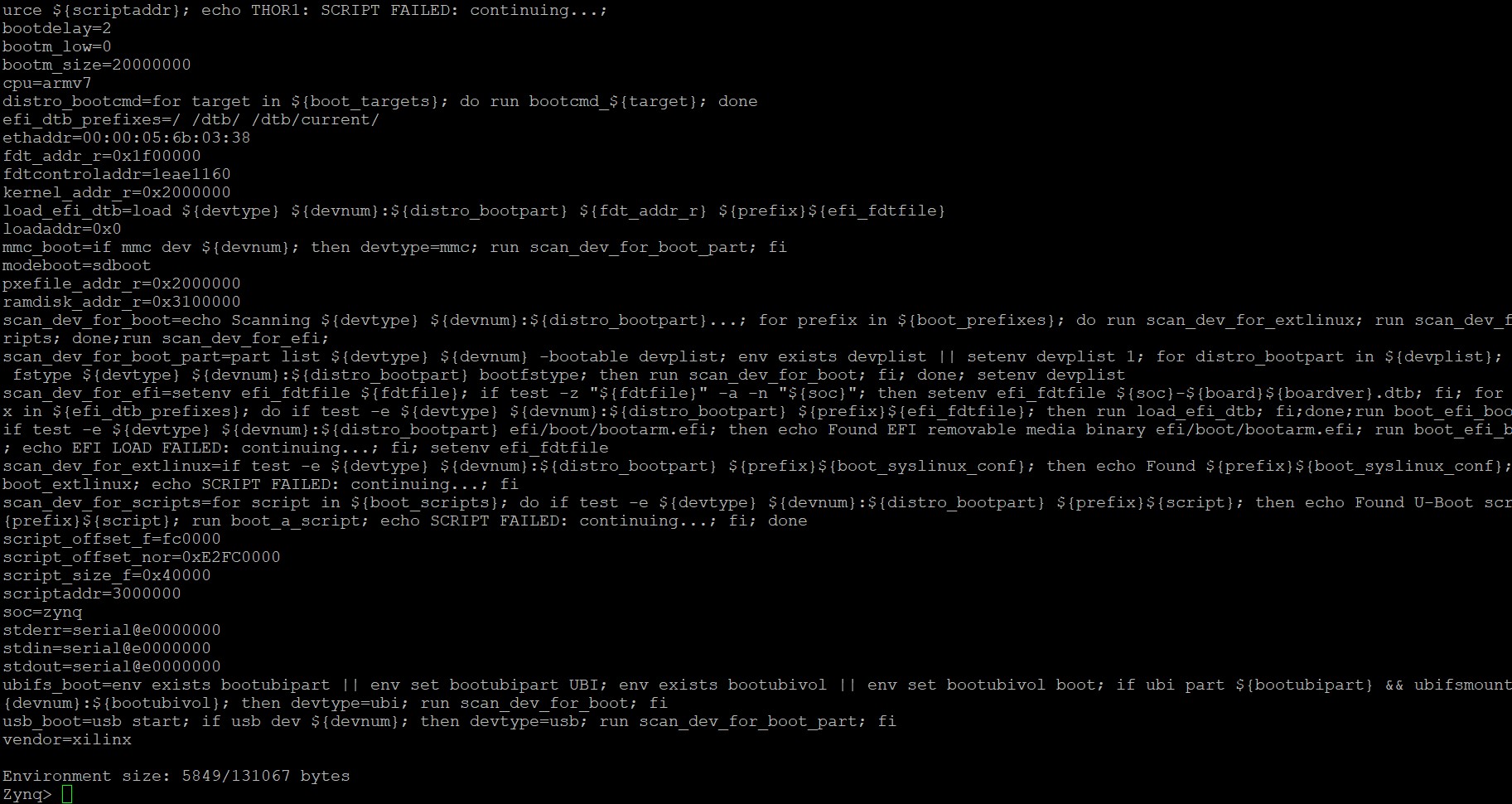
**Due: 2/4/2024 11:59pm**

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 3 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. You could either scan your written copy, or simply type your answer in this Google Doc. **However, please make sure your responses are readable.**
* 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.

**Serial Connection**

* Using a micro USB cable, connect your board to your laptop
* Connect to board using the serial connection
  + Linux
    - Open a new terminal
    - Run the command
      * *sudo screen /dev/<port> 115200* #port: ttyUSB0 or ttyUSB1
  + MAC
    - Open a new terminal
    - Run the command and check the PYNQ resources for the port
      * *sudo screen /dev/<port> 115200* #port: check resources
  + Windows
    - Check the resource for how to connect through serial to the PYNQ board
  + Resources:
    - <https://pynq.readthedocs.io/en/v2.0/getting_started.html>
    - <https://www.nengo.ai/nengo-pynq/connect.html>
* After connecting
  + Restart the board (*$ sudo reboot*)
  + Interrupt the boot (keyboard interrupt)
  + List current settings (*printenv*)
  + **Put a screenshot of your *$ printenv* output**
* 

**Change Bootargs**

* *If you need to return to the default bootargs, you can find them below*
  + [*https://github.com/Xilinx/PYNQ/blob/master/sdbuild/boot/meta-pynq/recipes-bsp/device-tree/files/pynq\_bootargs.dtsi*](https://github.com/Xilinx/PYNQ/blob/master/sdbuild/boot/meta-pynq/recipes-bsp/device-tree/files/pynq_bootargs.dtsi)
  + *bootargs = ‘root=/dev/mmcblk0p2 rw earlyprintk rootfstype=ext4 rootwait devtmpfs.mount=1 uio\_pdrv\_genirq.of\_id="generic-uio" clk\_ignore\_unused’*
* To edit bootargs:
  + Interrupt the boot
  + Edit boot arguments:
    - *$ editenv bootargs*
    - Insert arguments included the quotations all in one line:
      * Bootargs (default and more) are at [here](https://drive.google.com/file/d/1dv4AMO5DWOriQH_njx6iMYb0uMK61-hC/view?usp=sharing)
    - *$ boot*
* Change bootargs to the following
  + *bootargs = 'console=ttyPS0,115200 root=/dev/mmcblk0p2 rw earlyprintk rootfstype=ext4 rootwait devtmpfs.mount=1 uio\_pdrv\_genirq.of\_id="generic-uio" clk\_ignore\_unused* ***isolcpus=1*** *&& bootz 0x03000000 - 0x02A00000'*
  + **What does isolcpus=1 do?**

it’s a Kernal command to isolate the CPU Core 1. This way most tasks from this CPU are removed. So user tasks are removed from this CPU Core.

* + **What would isolcpus=0 do?**

This does same as above but for CPU Core 0.

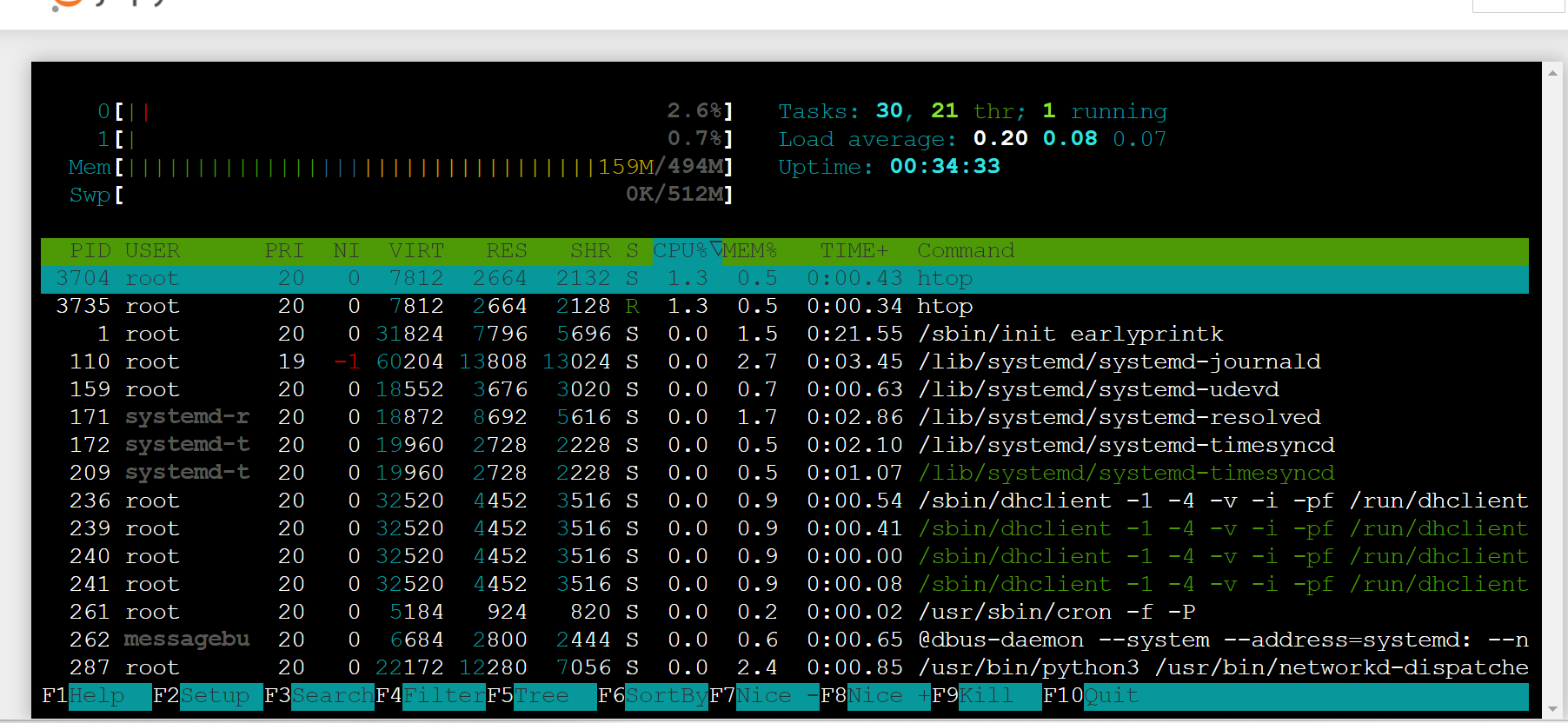
**Heavy CPU Utilization**

* Download *fib.py* from [here](https://drive.google.com/file/d/183Nbe01d34iH481xRHyO4F2KMEgDRxlR/view?usp=drive_link). This is a recursive implementation for generating Fibonacci sequences. We just do not print the results.
* Make sure your board is booted with custom bootargs above including *isolcpus=1*

1) Open two terminals (Jupyter):

* Terminal 1: run *htop* to monitor CPU utilization
* Terminal 2: run *$ python3 fib.py* and monitor CPU utilization and time spent for running the script (set terms to lower than 40)
* **Describe the results of *htop*.**

This describes the information about the CPU utilization.



2) Repeat the previous part, but this time use *taskset* to use CPU1:

* Terminal 2: run *$ taskset -c 1 python3 fib.py* and monitor CPU utilization and time spent for running the script
* **Describe the results of *htop.* Specifically, what’s different from running it in 1)?**

CPU 0 is not taking the load and CPU 1 is taking load.

3) Heavy Utilization on CPU0:

* Open another terminal and run $ *dd if=/dev/zero of=/dev/null*
* Repeat parts 1 and 2
* **Describe the results of *htop.***

CPU is 100% utilizing since its doing th null task in infinite loop.

**ARM Performance Monitoring (C++)**

* Download [kernel\_modules folder](https://drive.google.com/drive/folders/16pC3hYfLeJFQXMaJ9Zk8Do8NKPKm5nMA?usp=drive_link)
* Read through CPUcntr.c and reference the ARM documentation for the PMU registers [here](https://developer.arm.com/docs/100511/latest/performance-monitoring-unit/pmu-register-summary) to answer the following question.
  + **According to the ARM docs, what does the following line do? Are they written in assembly code, python, C, or C++?**
    - *asm(“MCR p15, 0, 1, c9, c14, 0\n\t”);*

Its assembly code. It means move to coprocessor from ARM register. P15 op1,op2 operations of register C9,C14.   
It enabled User mode to access Performance Mode register.

* + Compile and insert the kernel module following the instructions from the README file.
* Download [clock\_example folder](https://drive.google.com/drive/folders/1eXrLrpO5fSgJxwccfl1-axhWO59ByfoZ?usp=share_link)
* Read through *include/cycletime.h* and take note of the functions to initialize the counters and get the cyclecount (what datatype do they return, what parameters do they take)
  + **What does the following line do?**
    - *asm volatile ("MRC p15, 0, %0, c9, c13, 0\n\t" : "=r"(value));*

volatile makes sure that compiler doesnt optimize the code.   
And this code reads value from PMU cycle counter register.

* Complete the code in *src/main.cpp.* These instructions are for those who have never coded in C++
  + **Declare 2 variables (cpu\_before, cpu\_after) of the correct datatype**
  + **Initialize the counter**
  + **Get the cyclecount ‘before’ sleeping**
  + **Get the cyclecount ‘after’ sleeping**
  + **Print the difference number of counts between starting and stopping the counter**
* After completing the code, open a jupyter terminal and change directory to *clock\_examples/*
* Run *$ make* to compile the code
* Run the code with *$ ./lab3 <delay-time-seconds>*
* **Change the delay time and note down the different cpu cycles as well as the different timers.**

**Jupyter Notebook CPU Monitoring (OPTIONAL)**

* Download CPU\_monitor.ipynb from [here](https://drive.google.com/file/d/1_qpu_8fDwiqHNYXQyCQM2iKsUvZii259/view?usp=share_link). This is an interactive implementation for plotting in a loop. Running this notebook is a computationally heavy task for your CPU, therefore you do not need to run any additional process to utilize your CPU0.
* Create a Jupyter notebook
  + Use the *os* library to create a python program that accepts a number from user input (0 or 1) and runs *fib.py* on a specific core (0 or 1).
  + *Hint: look at the os.system() call and remember the ‘taskset’ function we’ve used previously.*
* You should have two notebooks running: 1) CPU\_monitor, 2) CPU\_select
  + **Compare your observations between using Jupyter notebook CPU\_monitor and linux command *htop* for monitoring CPU utilization.**