WES 237A: Introduction to Embedded System Design (Winter 2026)
Lab 3: Serial and CPU
Due: 2/1/2026 11:59pm

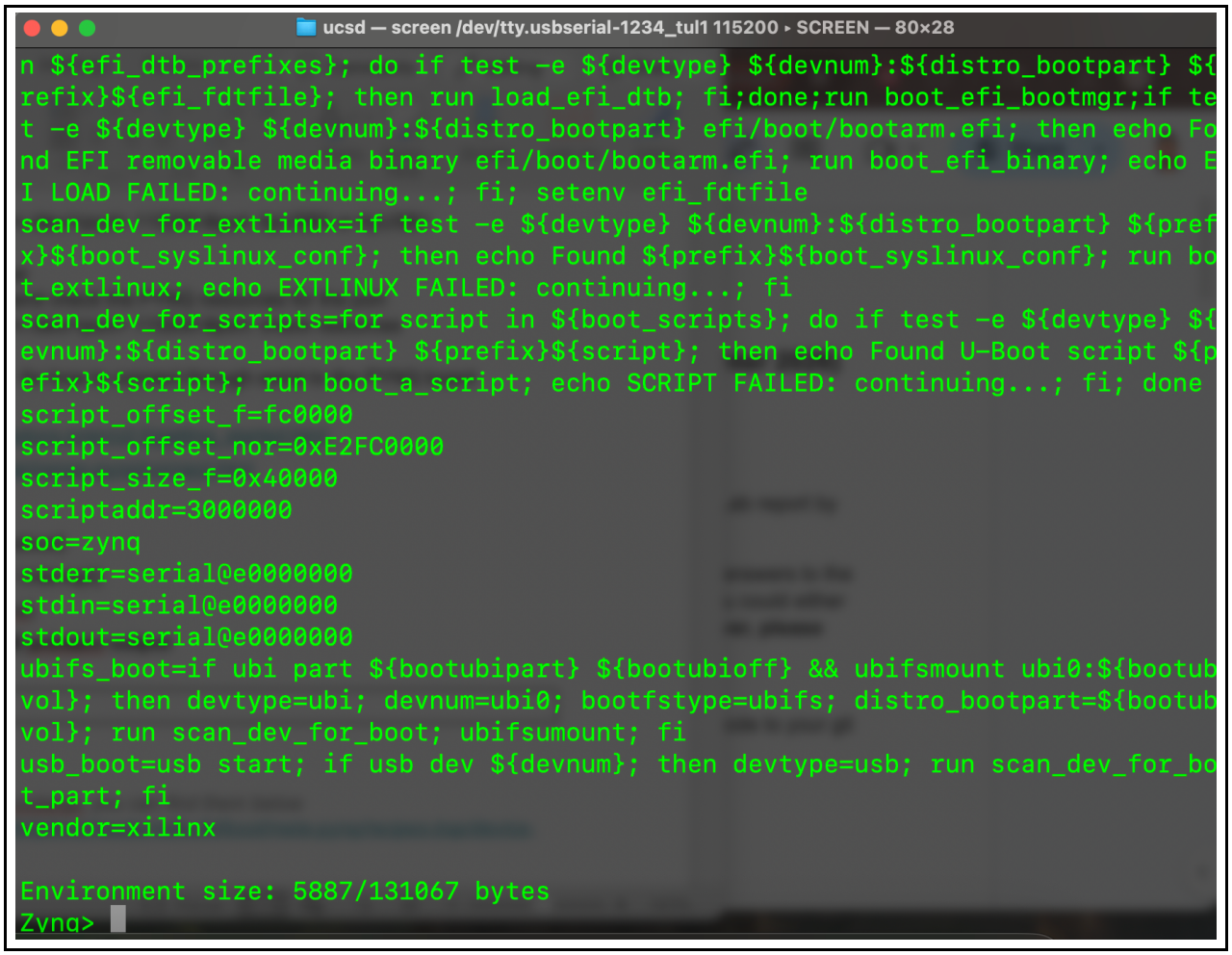
In order to report and reflect on your WES 237A labs, please complete this Post-Lab report by the end of the weekend by submitting the following 2 parts:

- Upload your lab 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*
  - *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
    - *$ 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?**

isolcpus — Isolate CPUs from the kernel scheduler. Specifically, isolate cpu 1 which is the second cpu.

“`isolcpus=` *`cpu_number`*`[,` *`cpu_number`*`,...]`

Remove the specified CPUs, as defined by the *`cpu_number`* values, from the general kernel SMP balancing and scheduler algroithms. The only way to move a process onto or off an "isolated" CPU is via the CPU affinity syscalls. *`cpu_number`* begins at `0`, so the maximum value is 1 less than the number of CPUs on the system.” – O’Reilly https://www.oreilly.com/library/view/linux-kernel-in/0596100795/re46.html

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

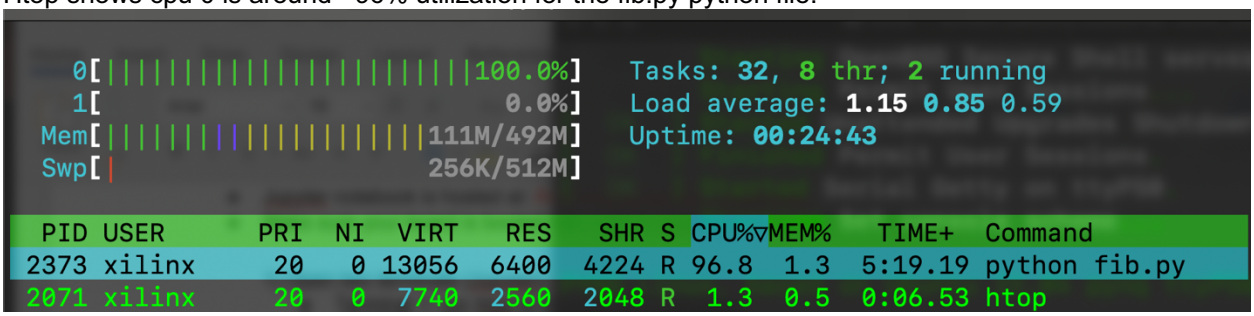
It would isolate the first cpu from the linux scheduler.

Heavy CPU Utilization

- Download *fib.py* from here. This is a recursive implementation for generating Fibonacci sequences. We just do not print the results.
- Jupyter notebook is hosted at: /home/xilinx/jupyter_notebooks
- Make sure your board is booted with custom bootargs above, including *isolcpus=1*

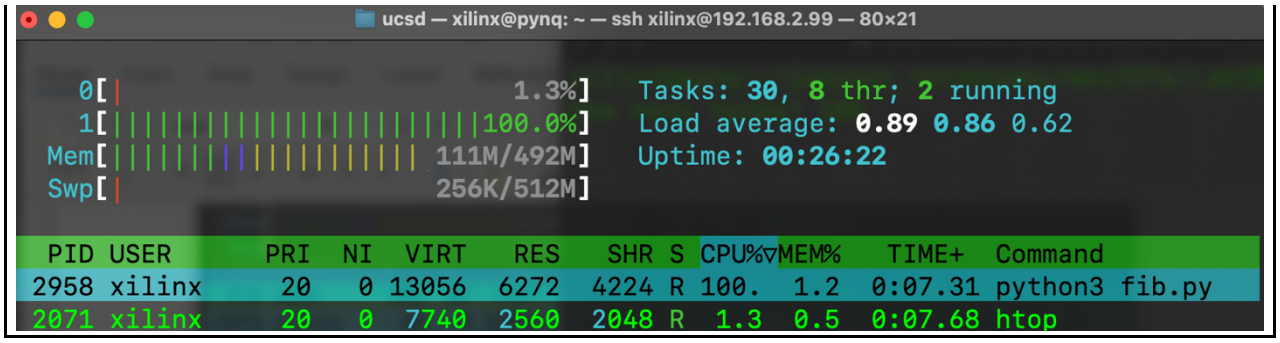
1) Open two terminals (Jupyter):
   - Terminal 1: run *htop* to monitor CPU utilization
   - Terminal 2: run *$ python3 fib.py* and monitor CPU utilization and time spent for running the script (set terms to lower than 40)
   - **Describe the results of *htop*.**

Htop shows cpu 0 is around ~96% utilization for the fib.py python file.



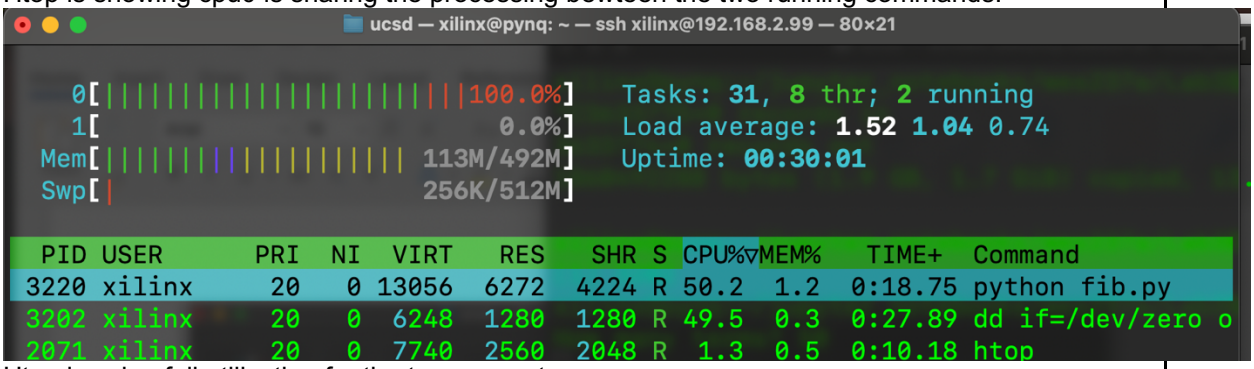
2) Repeat the previous part, but this time use *taskset* to use CPU1:
   - Terminal 2: run *$ taskset -c 1 python3 fib.py* and monitor CPU utilization and time spent for running the script
   - **Describe the results of *htop*. Specifically, what’s different from running it in 1)?**

Htop is showing the cpu is running the fub.py file on cpu 1 as instructed vs avoiding cpu 1 since we told the kernel to avoid unless specified.

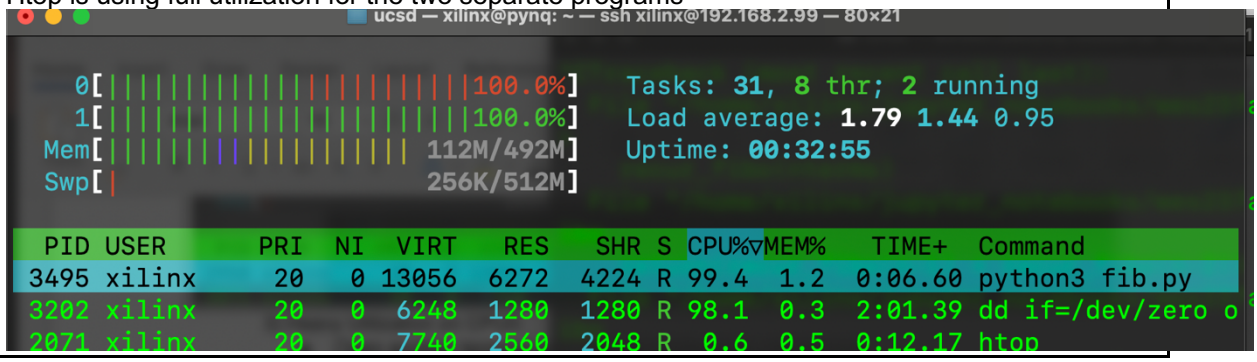


3) Heavy Utilization on CPU0:
   - Open another terminal and run $ *dd if=/dev/zero of=/dev/null*
   - Repeat parts 1 and 2
   - **Describe the results of *htop.***

1) Htop is showing cpu0 is sharing the processing bewteen the two running commands.



2) Htop is using full utilization for the two separate programs



Jupyter Notebook CPU Monitoring

Download CPU_monitor.ipynb from here. 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.**

The python CPU_monitor notebook shows only 100% cpu utilization but doesn't show how many threads all sharing the cpu unlike htop where it would show there are 2 programs running on cpu

ARM Performance Monitoring (C++)

- Download kernel_modules folder
- Read through CPUcntr.c and reference the ARM documentation for the PMU registers here 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");*

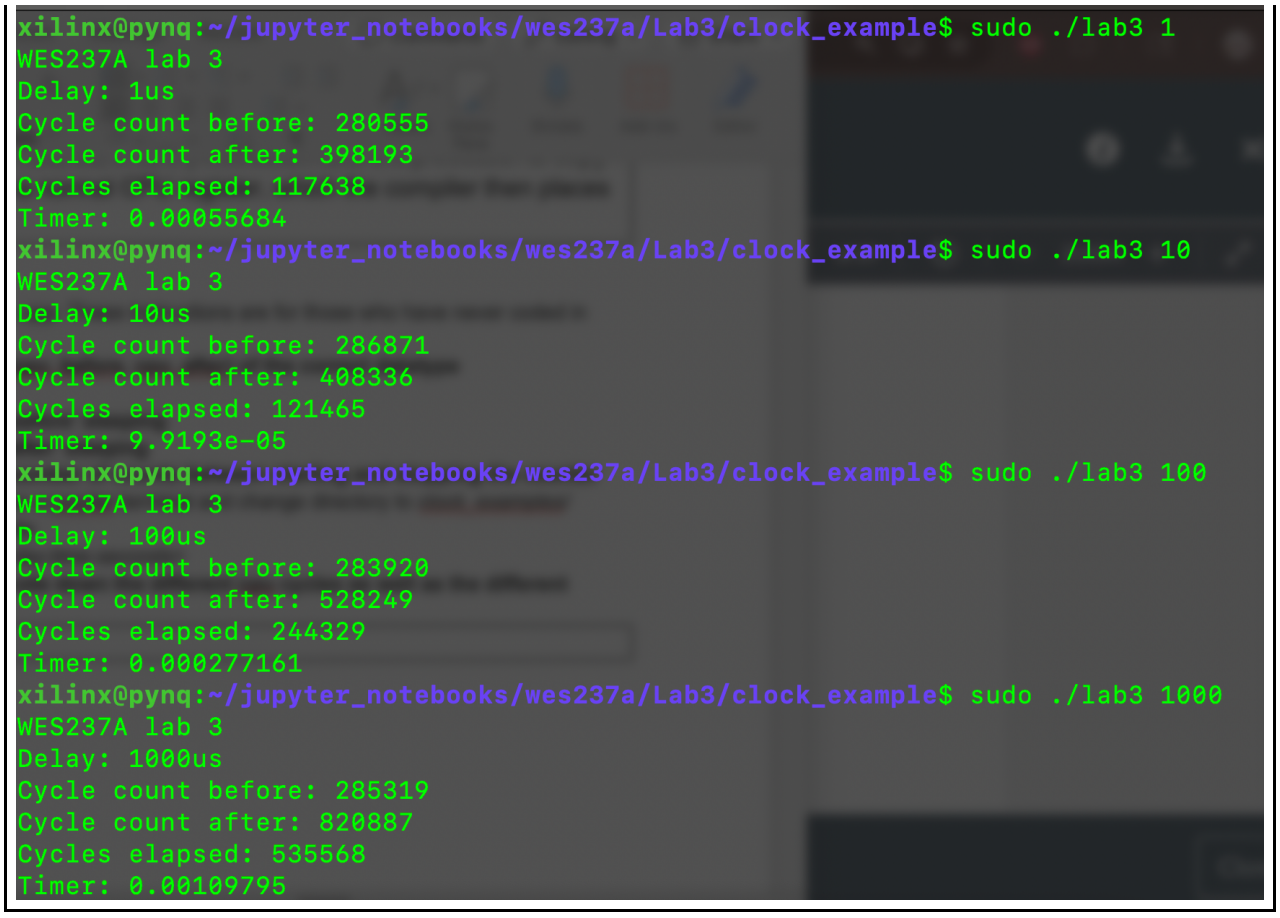
The instruction asm("MCR p15, 0, 1, c9, c14, 0") takes the immediate value 1 and writes it into PMUSERENR, which in ARM's performance monitor register map enables user-level access to the PMU counters defined in that register summary page.

  - Compile and insert the kernel module following the instructions from the README file.
- Download clock_example folder
- 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));*

This line reads a hardware performance counter from the ARM CPU and stores it in a C variable**.** The asm("MRC p15, 0, %0, c9, c13, 0") instruction tells the processor to copy the value of a PMU register into a normal CPU register, which the compiler then places into the variable value.

- 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.**

The below shows the number of cycles increase as the delay is increased.


xilinx@pynq:~/jupyter_notebooks/wes237a/Lab3/clock_example$ sudo ./lab3 1
WES237A lab 3
Delay: 1us
Cycle count before: 280555
Cycle count after: 398193
Cycles elapsed: 117638
Timer: 0.00055684
xilinx@pynq:~/jupyter_notebooks/wes237a/Lab3/clock_example$ sudo ./lab3 10
WES237A lab 3
Delay: 10us
Cycle count before: 286871
Cycle count after: 408336
Cycles elapsed: 121465
Timer: 9.9193e-05
xilinx@pynq:~/jupyter_notebooks/wes237a/Lab3/clock_example$ sudo ./lab3 100
WES237A lab 3
Delay: 100us
Cycle count before: 283920
Cycle count after: 528249
Cycles elapsed: 244329
Timer: 0.000277161
xilinx@pynq:~/jupyter_notebooks/wes237a/Lab3/clock_example$ sudo ./lab3 1000
WES237A lab 3
Delay: 1000us
Cycle count before: 285319
Cycle count after: 820887
Cycles elapsed: 535568
Timer: 0.00109795

CPU_monitor

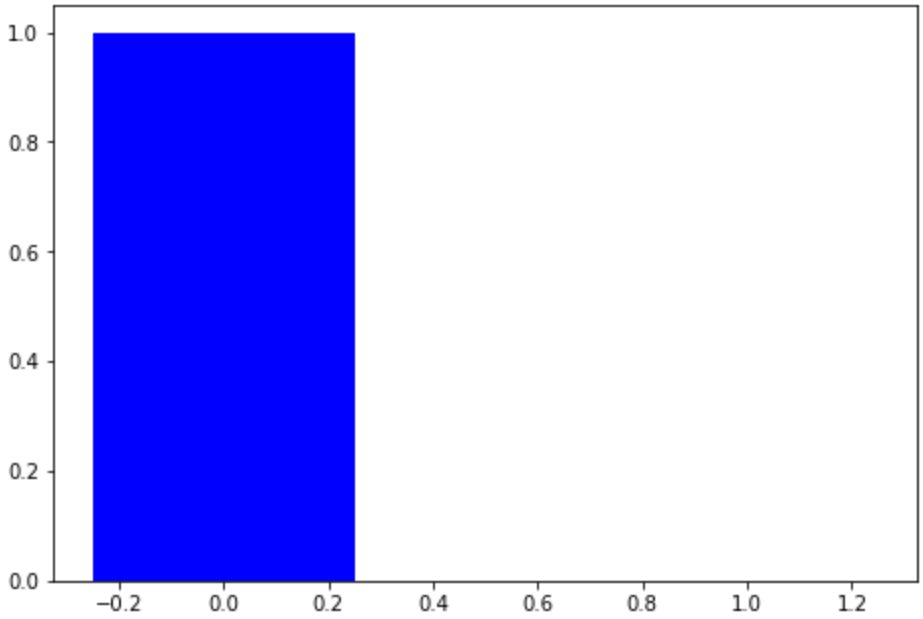
In [1]:
```
import time
import pylab as pl
from IPython import display
import psutil
import matplotlib.pyplot as plt
import numpy as np
```

In [27]:
```
psutil.cpu_percent(percpu=True)
```

Out[27]:
```
[3.7, 0.0]
```

In [35]:
```
%matplotlib inline

X = np.arange(1)
fig = plt.figure()
ax = fig.add_axes([0,0,1,1])
for i in range(10):
    data = psutil.cpu_percent(percpu=True)
    ax.cla()
    ax.bar(X + 0.0, data[0]/100, color = 'b', width = 0.5)
    ax.bar(X + 1.0, data[1]/100, color = 'g', width = 0.5)
    display.clear_output(wait=True)
    display.display(plt.gcf())
plt.clf()
```




```
<Figure size 432x288 with 0 Axes>
```

In [ ]:

CPU_select

In [1]:
```
import subprocess

ncpu = int(input("Which cpu to use? "))
if ncpu<0 or ncpu>1:
    print("0 or 1 only.")
else:
    nterms = input("How many terms? ")

    subprocess.run(
        ["taskset", "-c", str(ncpu), "python3", "fib.py"],
        input=nterms + "\n",
        text=True
    )
```

```
Which cpu to use? 1
How many terms? 10
How many terms? time spent: 0.0003464221954345703
```