CS 232: Operating Systems

Adding Context Switching to CalOS

The purpose of this homework is to explore what needs to be done to add concurrency support (i.e., "time-slicing" or "preemptive multitasking") to CalOS.

Current State of the OS

Accept this assignment. Then go to the web page for the repo, click on Code, then SSH, and get the repo-id. Then use git clone <repo-id> to get your copy of the code.

You should have 5 python files: **main.py, cpu.py, calos.py, ram.py,** and **devices.py**; and 2 assembly language files: **mult.asm** and **fib.asm**.

NOTE: this is python3, so make sure you use python3 to run the code, not python2.

Part Ein

Create a file **answers.txt** and answer these questions about the code.

1. Where is the PCB class defined?
2. What information does a PCB object contain?
3. When is a PCB created? Which line in which file creates a PCB?
4. What process states have been defined?
5. Regarding the variable current_proc: Where has it been defined and what is its significance?
6. In the TimerController class, what does code that reads

```
with self._mutex:
    code here...
```

   do? I.e., what does `with` mean in python and why does the code use it?
7. In the TimerController class, explain what the code does when the countdown goes to 0 (i.e., the time expires).

Part Ni

Note that the code I have given you will not execute now because crucial parts have been removed.

NOTE: the variable `current_proc` is a class variable -- it is not stored in each CalOS **object**, but in the **class** itself. So, to refer to it, you should write `CalOS.current_proc`.

You need to add code to **calos.py**:

- Implement the `timer_isr(self)` method. Here is the algorithm:
  - If the ready queue is not empty:
    - Call `context_switch()`
  - Reset the timer
- Implement the `context_switch(self)` method. Here is the algorithm:
  - Get the new process's PCB off the front of the ready queue.
  - Save the CPU's registers into the currently-running process's PCB.
  - Load the CPU's registers with the registers stored in the new process's PCB.
  - Put the old process on the end of the ready queue.
  - Set the new process to RUNNING state.
  - Set `current_proc` to the new process.
- Implement the `run()` method.
  - while the ready queue is not empty:
    - Remove the PCB from the front of the ready queue and set `current_proc` to the result.
    - Call `reset_timer` to set the timer controller's countdown.
    - Load the CPU's registers with the registers stored in the `current_proc`.
    - Call CPU's `run_process()` method.
    - Set the `current_proc`'s state to DONE.

Here is a sample run of the operating system where I

- load mult.asm into locations 20 - 32,
- put data in locations 10 and 11,
- load fib.asm into locations 100 - 125,
- put data in location 50,
- run the OS, which context switches back and forth between the two processes,
- turn off debugging,
- load mult.asm again so that it is in the ready queue again,
- run the OS again

```
Mac52078:~/classes/cs232/assignment-solutions/homework05 vtn2$
python3 main.py
Monitor: enter ? to see options.
MON> !
State of the CPU is:
```

```
CPU 0: pc 0, reg0 0, reg1 0, reg2 0
------------------------------------------------------------------------
MON> l 20 mult.asm
Created PCB for process mult
Main found at location 20
Tape loaded from 20 to 32
PCB(1): mult, state NEW, entrypoint 20
add_to_ready_q: queue is now:
     PCB(1): mult, state READY, entrypoint 20
Num ready processes = 1
State of the CPU is:
CPU 0: pc 0, reg0 0, reg1 0, reg2 0
------------------------------------------------------------------------
-
MON> d 10
Enter value (. to end) [10]> 5
Enter value (. to end) [11]> 4
Enter value (. to end) [12]> .
State of the CPU is:
CPU 0: pc 0, reg0 0, reg1 0, reg2 0
------------------------------------------------------------------------
-
MON> l 100 fib.asm
Created PCB for process fib
Main found at location 100
Tape loaded from 100 to 125
PCB(2): fib, state NEW, entrypoint 100
add_to_ready_q: queue is now:
     PCB(1): mult, state READY, entrypoint 20
     PCB(2): fib, state READY, entrypoint 100
Num ready processes = 2
State of the CPU is:
CPU 0: pc 0, reg0 0, reg1 0, reg2 0
------------------------------------------------------------------------
-
MON> d 50
Enter value (. to end) [50]> 8
Enter value (. to end) [51]> .
State of the CPU is:
CPU 0: pc 0, reg0 0, reg1 0, reg2 0
------------------------------------------------------------------------
-
MON> r
Calos.run() ready processes = 2
Timer: set countdown to 3
Running PCB(1): mult, state READY, entrypoint 20
{'reg2': 0, 'reg0': 0, 'pc': 20, 'reg1': 0}
Executing code at [20]: mov 0 12
```

```
CPU 0: pc 21, reg0 0, reg1 0, reg2 0
{'reg2': 0, 'reg0': 0, 'pc': 21, 'reg1': 0}
Executing code at [21]: mov *10 reg2
CPU 0: pc 22, reg0 0, reg1 0, reg2 5
{'reg2': 5, 'reg0': 0, 'pc': 22, 'reg1': 0}
Executing code at [22]: jez reg2 31
CPU 0: pc 23, reg0 0, reg1 0, reg2 5
{'reg2': 5, 'reg0': 0, 'pc': 23, 'reg1': 0}
Executing code at [23]: mov *11 reg1
CPU 0: pc 24, reg0 0, reg1 4, reg2 5
GOT INTERRUPT
End of quantum!
Switching procs from mult to fib
add_to_ready_q: queue is now:
     PCB(1): mult, state READY, entrypoint 20
Num ready processes = 1
Timer: set countdown to 3
{'reg2': 0, 'reg0': 0, 'pc': 100, 'reg1': 0}
Executing code at [100]: mov 1 150
CPU 0: pc 101, reg0 0, reg1 0, reg2 0
{'reg2': 0, 'reg0': 0, 'pc': 101, 'reg1': 0}
Executing code at [101]: mov 1 151
CPU 0: pc 102, reg0 0, reg1 0, reg2 0
{'reg2': 0, 'reg0': 0, 'pc': 102, 'reg1': 0}
Executing code at [102]: mov 1 500
CPU 0: pc 103, reg0 0, reg1 0, reg2 0
GOT INTERRUPT
End of quantum!
Switching procs from fib to mult
add_to_ready_q: queue is now:
     PCB(2): fib, state READY, entrypoint 100
Num ready processes = 1
Timer: set countdown to 3
{'reg2': 5, 'reg0': 0, 'pc': 24, 'reg1': 4}
Executing code at [24]: jez reg1 31
CPU 0: pc 25, reg0 0, reg1 4, reg2 5
{'reg2': 5, 'reg0': 0, 'pc': 25, 'reg1': 4}
Executing code at [25]: mov reg2 reg0
CPU 0: pc 26, reg0 5, reg1 4, reg2 5
{'reg2': 5, 'reg0': 5, 'pc': 26, 'reg1': 4}
Executing code at [26]: sub 1 reg1
CPU 0: pc 27, reg0 5, reg1 3, reg2 5
GOT INTERRUPT
End of quantum!
Switching procs from mult to fib
add_to_ready_q: queue is now:
     PCB(1): mult, state READY, entrypoint 20
Num ready processes = 1
```

```
Timer: set countdown to 3
{'reg2': 0, 'reg0': 0, 'pc': 103, 'reg1': 0}
Executing code at [103]: mov 1 501
CPU 0: pc 104, reg0 0, reg1 0, reg2 0
{'reg2': 0, 'reg0': 0, 'pc': 104, 'reg1': 0}
Executing code at [104]: mov 502 152
CPU 0: pc 105, reg0 0, reg1 0, reg2 0
{'reg2': 0, 'reg0': 0, 'pc': 105, 'reg1': 0}
Executing code at [105]: mov *50 reg0
CPU 0: pc 106, reg0 8, reg1 0, reg2 0
GOT INTERRUPT
End of quantum!
Switching procs from fib to mult
add_to_ready_q: queue is now:
     PCB(2): fib, state READY, entrypoint 100
Num ready processes = 1
Timer: set countdown to 3
{'reg2': 5, 'reg0': 5, 'pc': 27, 'reg1': 3}
Executing code at [27]: jez reg1 30
CPU 0: pc 28, reg0 5, reg1 3, reg2 5
{'reg2': 5, 'reg0': 5, 'pc': 28, 'reg1': 3}
Executing code at [28]: add reg0 reg2
CPU 0: pc 29, reg0 5, reg1 3, reg2 10
{'reg2': 10, 'reg0': 5, 'pc': 29, 'reg1': 3}
Executing code at [29]: jmp 26
CPU 0: pc 26, reg0 5, reg1 3, reg2 10
GOT INTERRUPT
End of quantum!
Switching procs from mult to fib
add_to_ready_q: queue is now:
     PCB(1): mult, state READY, entrypoint 20
Num ready processes = 1
... etc ...
Executing code at [30]: mov reg2 12
CPU 0: pc 31, reg0 5, reg1 0, reg2 20
{'reg2': 20, 'reg0': 5, 'pc': 31, 'reg1': 0}
Executing code at [31]: end
Done running PCB(1): mult, state DONE, entrypoint 20, num
ready_processes now 1
Timer: set countdown to 3
Running PCB(2): fib, state READY, entrypoint 100
{'reg2': 2, 'reg0': 502, 'pc': 118, 'reg1': 1}
Executing code at [118]: add 1 reg0
CPU 0: pc 119, reg0 503, reg1 1, reg2 2
GOT INTERRUPT
End of quantum!
Timer: set countdown to 3
{'reg2': 2, 'reg0': 503, 'pc': 119, 'reg1': 1}
```

```
Executing code at [119]: mov reg0 152
CPU 0: pc 120, reg0 503, reg1 1, reg2 2

... etc ...

CPU 0: pc 108, reg0 508, reg1 13, reg2 0
{'reg2': 0, 'reg0': 508, 'pc': 108, 'reg1': 13}
Executing code at [108]: mov *153 reg0
CPU 0: pc 109, reg0 0, reg1 13, reg2 0
{'reg2': 0, 'reg0': 0, 'pc': 109, 'reg1': 13}
Executing code at [109]: jez reg0 124
CPU 0: pc 124, reg0 0, reg1 13, reg2 0
GOT INTERRUPT
End of quantum!
Timer: set countdown to 3
{'reg2': 0, 'reg0': 0, 'pc': 124, 'reg1': 13}
Executing code at [124]: end
Done running PCB(2): fib, state DONE, entrypoint 100, num
ready_processes now 0
State of the CPU is:
CPU 0: pc 124, reg0 0, reg1 13, reg2 0
------------------------------------------------------------------
-
MON> s 10 14
[0010] 5
[0011] 4
[0012] 20
[0013] 0
[0014] 0
State of the CPU is:
CPU 0: pc 124, reg0 0, reg1 13, reg2 0
------------------------------------------------------------------
-
MON> s 500 510
[0500] 1
[0501] 1
[0502] 2
[0503] 3
[0504] 5
[0505] 8
[0506] 13
[0507] 21
[0508] 0
[0509] 0
State of the CPU is:
CPU 0: pc 124, reg0 0, reg1 13, reg2 0
------------------------------------------------------------------
-
```

```
MON> !
MON> l 20 mult.asm
Tape loaded from 20 to 32
MON> r
MON>
```

Notice (directly above here) that NO output appears when debugging is turned off. Your debugging messages can be different from mine, **but nothing should be printed when debugging is off.**

Checking In

Submit to your github repo all the code and supporting files for this homework. (Don't forget to submit your **answers.txt** file.)

We will grade this exercise according to the following criteria: (15 pts total)

- 7 pts: **answers.txt**
- 8 pts: code executes correctly.