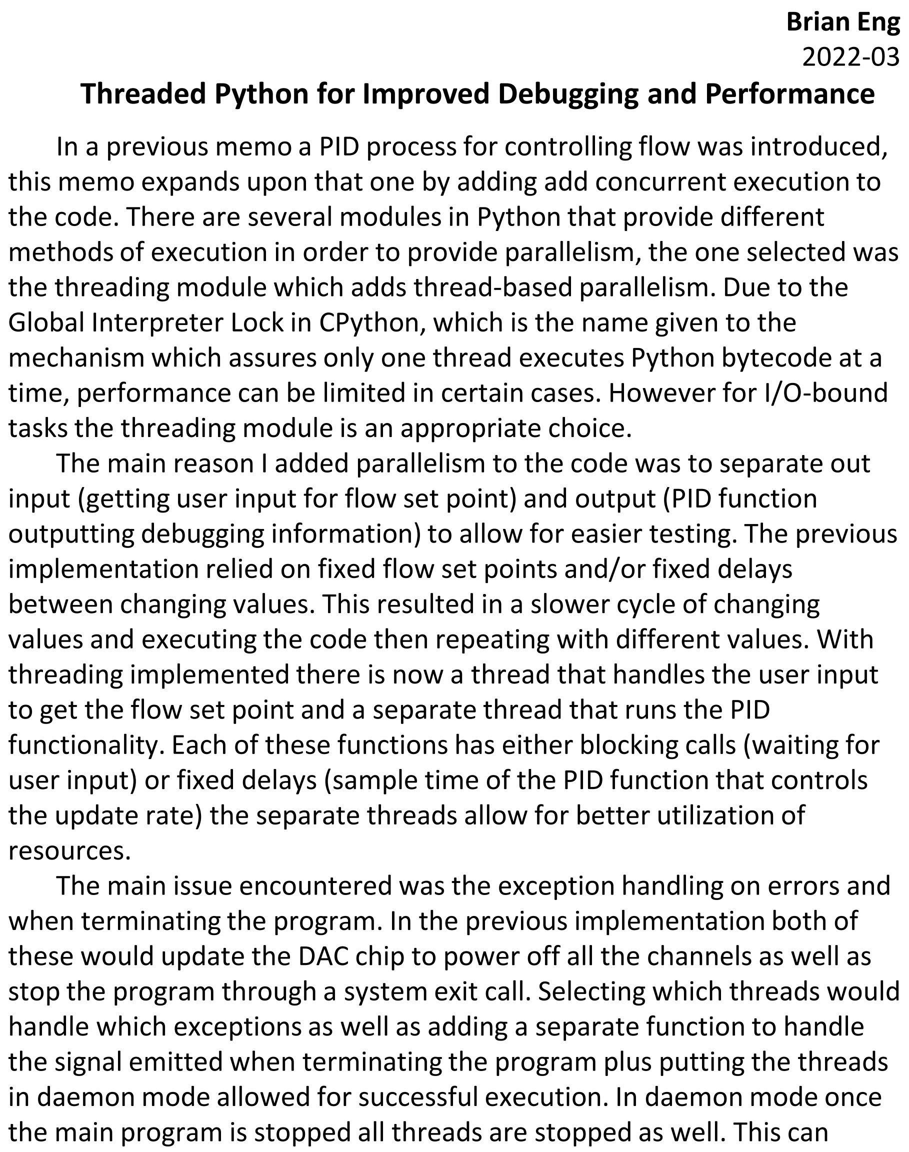
Threaded Python



Brian Eng
2022-03

Threaded Python for Improved Debugging and Performance

In a previous memo a PID process for controlling flow was introduced, this memo expands upon that one by adding add concurrent execution to the code. There are several modules in Python that provide different methods of execution in order to provide parallelism, the one selected was the threading module which adds thread-based parallelism. Due to the Global Interpreter Lock in CPython, which is the name given to the mechanism which assures only one thread executes Python bytecode at a time, performance can be limited in certain cases. However for I/O-bound tasks the threading module is an appropriate choice.

The main reason I added parallelism to the code was to separate out input (getting user input for flow set point) and output (PID function outputting debugging information) to allow for easier testing. The previous implementation relied on fixed flow set points and/or fixed delays between changing values. This resulted in a slower cycle of changing values and executing the code then repeating with different values. With threading implemented there is now a thread that handles the user input to get the flow set point and a separate thread that runs the PID functionality. Each of these functions has either blocking calls (waiting for user input) or fixed delays (sample time of the PID function that controls the update rate) the separate threads allow for better utilization of resources.

The main issue encountered was the exception handling on errors and when terminating the program. In the previous implementation both of these would update the DAC chip to power off all the channels as well as stop the program through a system exit call. Selecting which threads would handle which exceptions as well as adding a separate function to handle the signal emitted when terminating the program plus putting the threads in daemon mode allowed for successful execution. In daemon mode once the main program is stopped all threads are stopped as well. This can

- **Added threading module to PID flow control for concurrent execution**
- **Allows separate threads for handling user input and PID functions**

Threaded Python

potentially cause issues with resources not being released properly however this functionality is performed in the signal handling function.

The addition of the threading module to allow for concurrent execution will enable for faster debugging of the PID functionality when testing different flow change scenarios via manipulating the flow set point. This also paves the way for improved performance if additional valves are added as currently only a single valve is being tested as proof-of-principle before expanding to a full system.