# CSE231: Operating Systems

## Assignment 2, Question 1

**Name:** Mihir Chaturvedi
**Roll no.:** 2019061
**Section:** B
**Branch:** CSE

---

The code for both variants process (part 1) and threads (part 2) follows the following structure:

- Declare global `num` variable and initialize it to 0.
- In the `main` function
    - Fork the process/create a new thread
        - Inside the new process/thread, decrease `num` until it reaches the value -90.
        - Print the current value of `num`.
    - In the parent process/thread, increase `num` until it reaches the value 100.
    - Wait for the child process/thread to return.
    - Print the current value of `num`.

A difference is observed when running the two programs separately.
Output of part 1 (processes):

```
-90
100
```

Output of part 2 (threads):

```
-90
-90
```

This difference is caused due to the concurrent nature of two kernel threads running simultaneously. Processes are not run concurrently - they are scheduled by the scheduler to run one-after-the-other, but not parallely.

Moreover, when a process is forked, the child process creates a copy of parent memory space, and thus parent and child processes operate on completely different, separate memory. Changes to a global variable (in our case, `num`) by the child process, does not bring change, or affect the global variable of the parent process because they don't point to the same memory address.

This does not follow in threads. Multiple threads of the same process share the memory space of their parent, and so changes brought on by some thread will be accessible to the different threads, or the parent process. This brings in scope for ambiguity in the final values stored in memory addresses if multiple threads are accessing it simultaneously.

In part 2, two threads are simultaneously attempting to increase or decrease the value of `num`. The value of `num` from the parent thread is -90, as we wait for the child thread.

When testing with values of much greater magnitude, say -90000 and 10000, we start to see much more different outputs in the threads' program. There we can truly appreciate the ambiguity brought about by the concurrency - the print statement will attempt to read `num` after it has been decreased to -90000, but until then, the other thread increases starts incrementing it (to make `num` reach 100000), and so the printed value of the child thread is a bit larger than -90000.