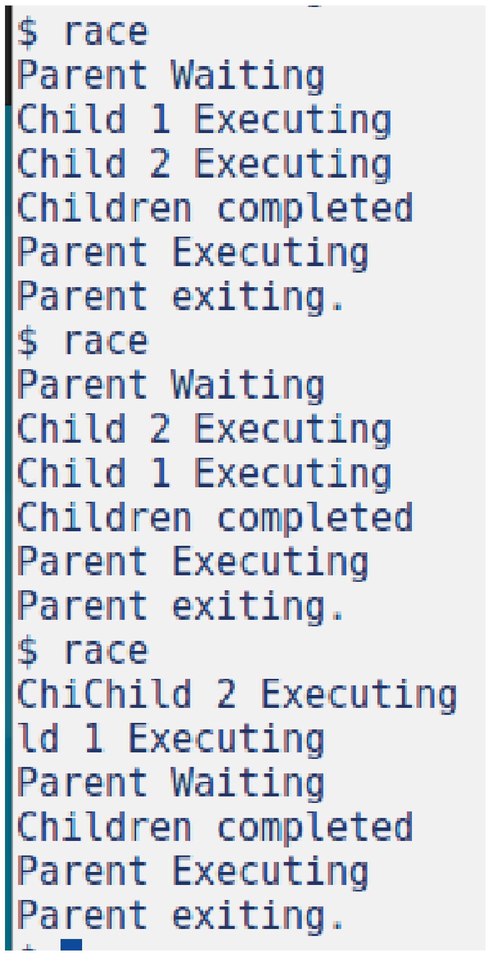
CS350 Project 2 FRQ
Qilin Ye
https://github.com/YQL-Skorpion/yeqilin_CSCI350_P2_FRQ/tree/master

Part 1

- What is the effect of the parent process calling wait() two times? (1 pt)
    - Ans: this is to ensure that the parent process will execute only after BOTH child processes are done.
- Do you always get the same order of execution? (1 pt) Does Child 1 always execute (print Child 1 Executing before Child 2? (1 pt)
    - Ans: no to both questions. Specifically, the child processes' order of execution is random. Sometimes we are even able to see interweaving text. The screenshot below is worth all the description.



```
$ race
Parent Waiting
Child 1 Executing
Child 2 Executing
Children completed
Parent Executing
Parent exiting.
$ race
Parent Waiting
Child 2 Executing
Child 1 Executing
Children completed
Parent Executing
Parent exiting.
$ race
ChiChild 2 Executing
ld 1 Executing
Parent Waiting
Children completed
Parent Executing
Parent exiting.
```

- Add a sleep(5); line before the line where Child 1 prints "Child Executing". What do you notice? (1 pt) Can we guarantee that Child 1 always executes before Child 2? (1 pt)
    - Ans: by running a handful of times it seems like child 2 is always executing before child 1 with the addition of sleep(5). However, in theory, while sleep(5) makes child 2 a lot more likely to be executed first, this is still not guaranteed, as the exact order also depends on how fast the scheduling takes place.

Part 3

After seeing what the two system calls do, why do you think we had to add system calls for the operations on condition variables? Why not just have these operations as functions in ulib.c as we did for the spinlock? (4 pts)

- Ans: By implementing CV operations as system calls rather than library functions, we ensure atomicity during context switches (which is a primary reason for us to use CVs). As mentioned many times in lectures/discussions, this also provides better security.

Part 4

Does Child 1 always execute before Child 2? (2 pts) — Yes.

Part 5

Is it always the case that Child 1 executes before Child 2? (2 pts) Do you observe deadlocks? (2 pts)

Empirical results: yes child 1 always executes first; no, no deadlock has been observed. In theory, child 1 executes first, since child 2 is forced to wait due to the CV. Also, there should not be a deadlock since there is no circular waiting – we addressed this by explicitly putting cv_wait() inside a loop as well as making cv_wait() an atomic syscall.