# CIS 4480/5480 Recitation 1 - Processes, Valgrind and Style SOLUTION

*Welcome back to recitation!!! 😃*

**<u>Exercise 1</u>: Processes and File Access**

```c
#include <fcntl.h>
#include <stdlib.h>

int main() {
  pid_t child = fork();
  int fd = open("file.txt", O_WRONLY);
  if (fd == -1) {
    exit(EXIT_FAILURE);
  }
  write(fd, "this is parent or child.", 25);
  close(fd);
  return 0;
}
```

Questions to answer:

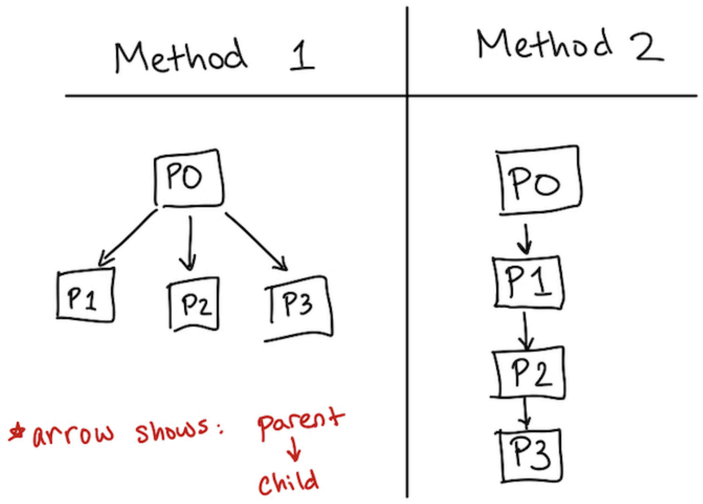
- Which processes have access to `file.txt`?
    a) Parent
    b) Child
    **c) Both**
    d) Neither

- If the parent closes the file, can the child still write to `file.txt`? **Explain your answer.**

Yes, the child can still write to file.txt because when fork() was called, the parent's file descriptor table (the table that lists the files open to the process) is duplicated for the child process. Thus the child has access to all files that the parent did at the time of fork(), but the access is independent of the parent's access. If the parent closes the file, the file descriptor is removed from the parent's file descriptor table but not the child's.

## Exercise 2: The Process Family Tree

Here are two diagrams, where each labeled box represents a process. P0 is the "original process" that forks P1. Arrows show the parent-child relationship. The order of processes spawning from first to last is: P0, P1, P2, P3.



Questions to answer:

- Using either C code, psuedocode, or a written description, describe how you would fork 3 processes to achieve diagram 1 and diagram 2.

| Diagram 1 | Diagram 2 |
|---|---|
| The original process (P0) calls fork. Then, you check if the return value of fork is not zero. If so, you call fork again, ensuring only the parent does this. Check if the most recent return value of fork is not zero, and if it is, call fork a third time. Then, the parent calls wait() three times. | The original process (P0) calls fork. Then, you check if return value of fork is zero. If so, you call fork again (to ensure only the child is calling fork). Check if the most recent return value of fork is zero, and if it is, call fork a third time. Each parent should call wait() after fork returned. |

- Let’s say I have 3 independent tasks: T1, T2, and T3.
    - P1 will exec T1
    - P2 will exec T2
    - P3 will exec T3
    - T1, T2, and T3 all require I/O calls to be made (i.e. reading from or writing to a file)
    - P0 must wait until T1, T2, and T3 have finished.

  Which diagram will result in the faster runtime? Explain your answer.

  The first diagram shows faster runtime, because having one parent fork 3 processes to do 3 independent tasks will make best use of concurrency. For diagram 1, let’s say the CPU is running P3, and comes to a point where P3 has to read a file. An I/O call like reading or writing doesn’t require CPU, so when P3 is waiting to receive data, the CPU can context switch to either P1 or P2 (but not P0, which is blocked). On the other hand, diagram 2 would not allow a context switch from P3 because P0, P1, and P2 are blocked until P3 exits, so the CPU is “wasted.”

## Exercise 3: Waiting

```
int main(void){
  int level_1 = fork();
  if (level_1 == 0) {
    int level_2a = fork();
    if (level_2a == 0) {
      printf("A");
    } else {
      wait(NULL);
      printf("B");
    }
  } else {
    int level_2b = fork();
    if (level_2b == 0) {
      printf("C");
      exit(0);
    }
    printf("D");
  }
  printf("0");
  return (0);
}
```

Questions to Answer:

1. Draw a diagram of all processes and clearly indicate all parent-child relationships. You may model your diagram after the one shown in Exercise 2, if you would like.

$$\begin{tikzcd}
\boxed{\begin{array}{l} \text{level\_1} = 0 \\ \text{level\_2a} \mathrel{!}= 0 \end{array}} \arrow[d] & \boxed{\begin{array}{l} \text{level\_1} \mathrel{!}= 0 \\ \text{level\_2b} \mathrel{!}= 0 \end{array}} \arrow[l] \arrow[d] \\
\boxed{\text{level\_2a} = 0} & \boxed{\text{level\_2b} = 0}
\end{tikzcd}$$

2. Which of the following are possible outputs? Select all that apply:
   a. B0AC0D0
   **b. D0CA0B0**
   **c. D0A0B0C**
   **d. CAD00B0**
   e. ABCD000

**<u>Exercise 4: Good Style</u>**😎

Read through the style guide here:
https://www.seas.upenn.edu/~cis5480/25su/documents/style

Questions to Answer:

- What style guides did you learn from reading the style guide and plan to use before turning in Penn-Shredder?

- Are there any style guidelines that are confusing, or that you think don't actually contribute to good style? If so, explain what you find confusing, or what seems unhelpful.

**Exercise 5: Exit Questions**

From 1-5, answer the following:

How fast is the recitation pacing?

| | | | | | | |
|---|---|---|---|---|---|---|
| (not fast) | 1 | 2 | 3 | 4 | 5 | (fast) |

How helpful is the recitation lecture portion?

| | | | | | | |
|---|---|---|---|---|---|---|
| (not helpful) | 1 | 2 | 3 | 4 | 5 | (helpful) |

How helpful is the recitation worksheet portion?

| | | | | | | |
|---|---|---|---|---|---|---|
| (not helpful) | 1 | 2 | 3 | 4 | 5 | (helpful) |

Would you like to see more practice, or more content-review?

- Content review!
- Practice!
- There’s a good balance of both!

Any feedback?

Mark any topics you would like to see/practice next week

- Wait versus Waitpid
- Masks
- Handlers
- Pipes
- File descriptors
- Debugging (GDB or Valgrind)
- Other: ________________________