University of Victoria

# University of Victoria
# Midterm 1 PRACTICE QUESTIONS
# Fall 2025

| | |
|---|---|
| Course Name & No. | CSC 360: Operating Systems |
| Section (CRN) | |
| Instructor | Wenjun Yang |
| Duration | 50 Minutes |

- This series of practice questions reflects the grade distribution and structure of our actual exam. You may expect the exam to have the same structure and distribution of concepts, albeit with different questions.
- The marks included may serve to help guide your understanding of work distribution within the exam, as you may assume the actual midterm to have similarly weighted questions.

# Part 1: Single Choice Questions (30 Marks)

No practice is provided here, however you may assume that these questions will be constructed from the concepts covered in the study guide.
Distribution: 10x3 pts.

- This part omitted from practice exam.

# Part 2: Written Answer Questions (70 Marks)

The following questions require written responses. Point-form should be used where possible, providing they accurately convey the idea behind your response.
Distribution: 30+40 pts.
**Make sure to state any assumptions made.**

## Question 11 (20+10 Marks)

Consider the syntactically correct C code below:

```c
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/wait.h>

int counter = 10;

int main(){
    pid_t pid;
     pid = fork();
    if (pid == 0) {
        counter *= 3;
        printf("CHILD: counter = %d\n", counter);
        return 0;
    }
    else if (pid > 0) {
        wait(NULL);
        printf("PARENT: counter = %d\n", counter);
        return 0;
    }
}
```

## 11.a (20 Marks)

After running the program, what will be the complete output from both lines 13 and 18?

**CHILD: counter = 30**
**PARENT: counter = 10**

**Explanation**

This problem becomes incredible simple with one key observation: Processes do not share memory.

When `fork()` is called, the program splits off from the original, creating a clone of all previously instantiated data with the instructions following the call to `fork()`. Hence changing a variable within one of these will not impact the other.

Because we have two separate processes after the `fork()` on line 10 and they share the same memory, we can check the `pid` value to see which process we are in. If we are in the child `pid` will be 0. If we are in the parent, `pid` will resolve to the process id of the child process (which is inherently >0). Using these `if` statements to verify checking if we are in parent or child lets us create instructions that execute in only one of the two.

Here, we multiply the child's copy of `counter` by 3, which has no bearing on the value of `counter` within the parent's scope.

## 11.b (10 Marks)

In no more than 1-2 sentences, explain what would happen if we moved the wait(NULL) call from line 17 to after the `printf()` statement on line 18?

**Explanation**

Moving the wait(NULL) call to after the printf statement would make the output order non-deterministic, as the parent would print immediately without waiting for the child to complete first. Both processes would still terminate properly, but sometimes the parent's output might appear before the child's output.

## Question 12 (40 Marks)

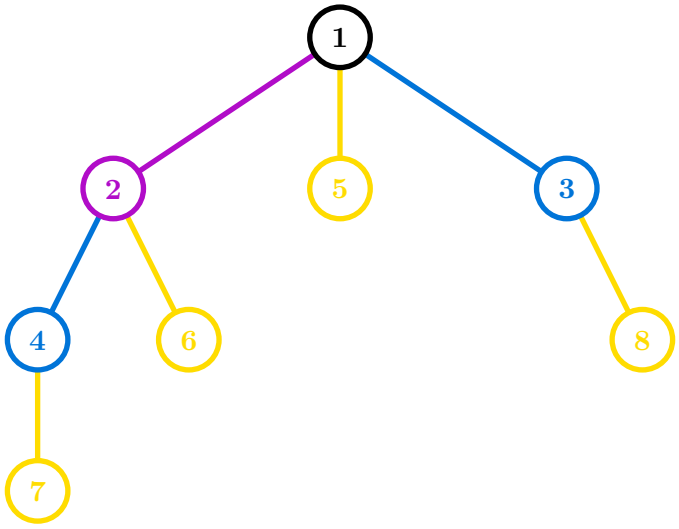
Consider the syntactically correct C code below:

```c
#include <unistd.h>

int main(){
  fork();
  fork();
  fork();
  return 0;
}
```

Including the initial parent process, how many process are created by the program shown above? You may use a tree-like structure to help visualize this.

Partially correct answers with a clear visualization may receive partial marks.




$\therefore$ There are 8 processes.