**Instructions** This lab assignment explores the data shared problem and process synchronization using Peterson’s solution.

Objectives of this assignment:

* to work on a Unix based system
* to “*dust off*” your programming skills in C
* to understand the fork() function to create a”child” process
* to understand the relationship (or lack of) between parent and child process
* to experience the ***data shared*** problem
* to deploy the **Peterson’s solution** to address the data shared problem

**IMPORTANT:**

1. *Your code will be tested and graded* ***REMOTELY*** *on the Engineering Unix (Tux) machines. If the code does not work on those machines, you will not get any credit even if your code works on any other machine.*
2. *A late submission will get a 50% penalty if submitted right after the deadline. The next day, you cannot submit the lab.*
3. *One submission per group.*
4. *Writing and presentation of your report are considered to grade your lab (30%). Your conclusions* ***must be supported*** *by the data/measurements you collect.*
5. *The quality of your code will be evaluated (****80%****).*
6. ***Questions about this lab must be posted on Piazza if you need a timely answer benefiting all students****.*

**Use this file to answer the questions. Highlight your answers and do NOT remove anything from this file. Just Insert your answers.**

**Part I: Programming on Tux machines**

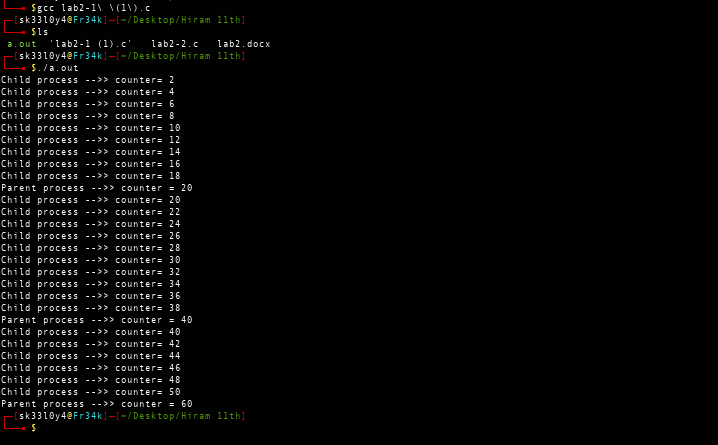
**(10 points) Program Exercise 1**:

# Exercise 1: Download the program *lab2-1.c*. Compile it and execute it. Observe the code and observe the output. This program has a parent and child processes *sharing* a variable. This program is *intended* to increment the shared (common) variable counter *\*countptr*. The parent process is *supposed* to increment *\*countptr* by increments of 20 while the child increments by 2s. A satisfactory execution of this program may be: the child increments the counter *\*countptr* twice (reaching 4), then the parent increments the counter *\*countptr* thrice to reach finally 64. Answer the following questions:

1. Does the program really execute as supposed (or intended)? Describe/Justify/Explain

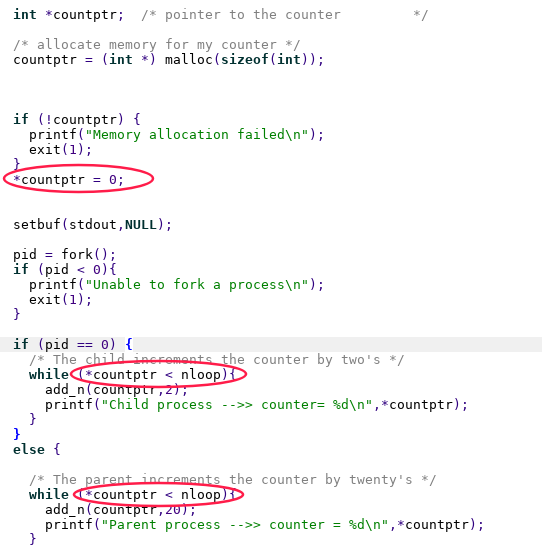
No. The program does not execute as intended.

The concept described within this format above explains that the program executes while making the parent process to have thrice the child process while the programs increments only twice thus making the output only twice the output. Logically this is correct but the description in the process is wrong.



1. Is the variable \****countptr*** really a shared (common) variable? In other words, are the changes made to \**countptr* by the child visible by the parent, and *vice versa*?  Describe/Justify/Explain.

Yes. countptr is a shared common variable within the structure of the program. Both the parent and child process depends on it for functionality and therefore making it a logical variable and common. Whenever the child process changes during execution, the parent process depends on it for functionality and therefore each version of the output are dependant on the variable making it a common variable.

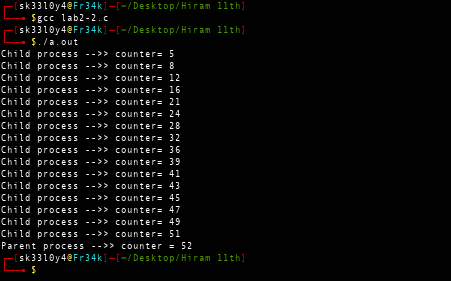


**(90 points) Program Exercise 2**:

The program ***lab2****-****2.c*** creates a genuine **shared** variable \**countptr*. Download, compile, and execute this program.

1. Based on the execution, show that \**countptr* is now a genuine shared variable (*countptr* points to a zone shared by the parent and the child). Now, are the changes to \**countptr* made by the child visible by the parent?

The program does not execute as required. It does not increment the child process by 2 and the parent by 20. Each process adds a single value to its state and generally it just auto increments.



Each process is just auto incremented. Logically the values of the parent and child are not correct but the countptr is relocatable since it gives both the child and parent process an equal and genuine visibility. Since they are all connected.

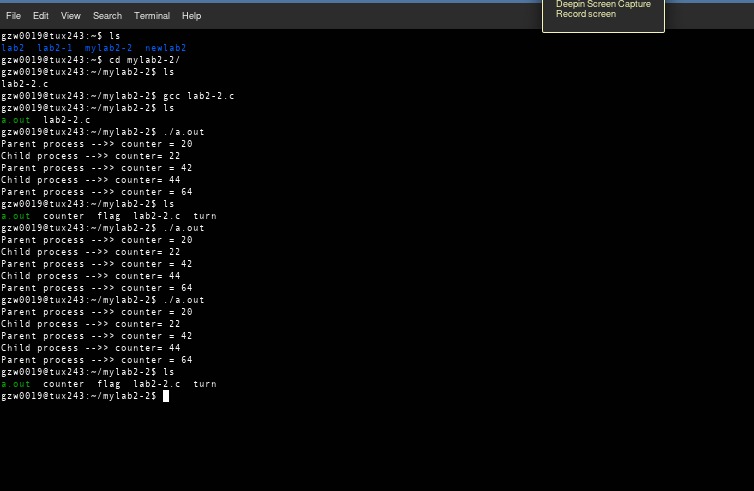
1. Does the program really execute as supposed (or intended), i.e, the counter increases exclusively in increments of 2 or 20? Explain what is happening.

**the values of the program are incorrect and correct in the execution. All implementation are based on the actual independent features that are stipulated by the conterptr function. It symbolically analyzes the implementation of the features engraved in the design and implementation**

3)       **Without modifying** the routine *add\_n()*, use the *Peterson’s* *solution* to correct the program ***lab2-2.c***. to execute as intended: the variable should increase by 2’s or twenty’s

The concept behind the Peterson solution is to enable the child and parent meet at the same process. Whenever all the two processes meet to execute the methodology can be used to entail the various characteristics by adding a 0 and 1 level which represents that if they all at the same turn, one is turned into false and the other into high.

***Hint***: Besides the pointer ***countptr*** used to point to the shared memory zone, you need to map three other integers Interested[2] and Turn (Peterson’s variables); These variables may be shared exactly the way that the zone pointed by *countptr* is shared.



**What to turn in?**

**Electronic copy**

Turn in separate files:

1. THIS file with INSERTED answers
2. Program ***lab2-2.***.c (corrected)

**A penalty of 10 points will be applied if these instructions are not followed.**

1. Your report must:
   1. state whether your code works. If is does work, state any issues you are aware of.
   2. Good writing and presentation are expected.