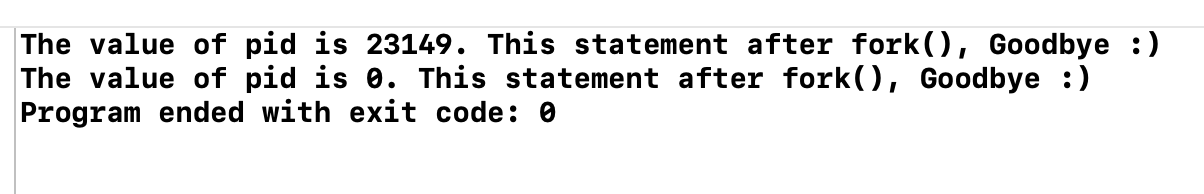
**Q1**

Variable pid will not be consistent, because it’s the process ID of children. Each time execute the code, there will be new execution of this code, hence new child process will be made, and resulting in different PID.

Text

Description automatically generated



**Q2**

The result of the variable myVariable in the output is 300 from parent, 200 from child. This is because fork() will return 0 to rc variable if it’s a child process, return PID of child process (not 0) to rc variable if it’s a parent process.

In this case, for child process, rc will have 0, hence in the if-else statement, it will change myVariable into 200. Where for parent process, it will go to else statement, and myVariable into 300.

Graphical user interface, text

Description automatically generated

**Q3**

No matter how, there will be 2 “level1” printed, 3”level2”, 3”level3”, 3”level4” printed.

So in the code, there are 3 process ID, namely pid1, pid2, pid3

Pid1 is fork(), therefore it will have parent and child process, either one will be first executed.

Let’s say PID 1’s parent process get executed first:

Pid1 is not 0, therefore it will go into else statement and fork() pid3, therefore will have pid3’s parent process & child process, either one will be first executed.

Let’s say PID3’s parent process get executed first:

pid 1 is not 0, pid3 is not 0, therefore, it will go into next if-else statement.

Where reach condition (pid1 !=0) print “level2” and condition (pid3!=0) print”level4”

PID3’s child process get executed:

Pid1 is not 0, pid3 is 0, in this case, it will execute pid2 fork(), then pid2 will have pid2’s parent process and child process, either one will be first executed.

Let’s say PID2’s parent process get executed first :

Pid1 is not 0, pid3 is 0, pid2 is not 0

Therefore, print “level2”, print ”level3”

PID2’s child process get executed:

Pid1 is not 0, pid3 is 0, pid2 is 0

Therefore, print “level2” .

PID1’s child process get executed:

Pid1 is 0, therefore will fork() pid2, have pid2’s parent & child process.

Let’s say PID2’s parent get executed first:

PID1 is 0, PID2 is not 0

Now it execute fork() pid3, same for pid 3, have parent and process

Let’s say PID3’s parent get executed first :

PID1 is 0, PID2 is not 0, PID3 is not 0

So print “level3”, “level4”

PID3’s child et executed :

PID1 is 0, PID2 is not 0, PID3 is 0

Print “level3”, “level4”

PID2’s child get executed :

PID1 is 0, PID2 is 0

Now fork() pid3, have parent and child

Lets say PID3’s parent first :

PID1 is 0, PID2 is 0, PID3 is not 0,

Print “level1”

Print”level4”

PID3’s child executed:

PID1 is 0, PID2 is 0, PID3 is 0

Print “level1”

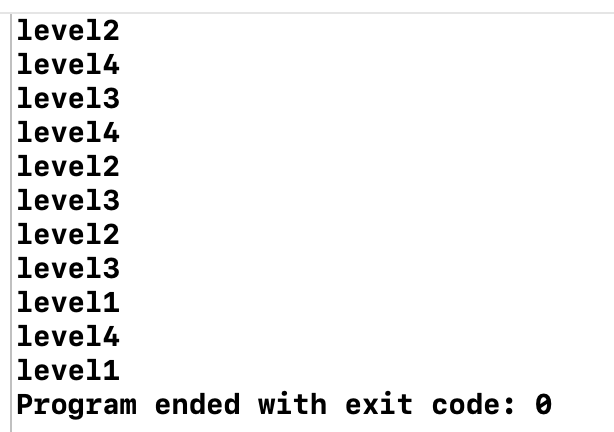
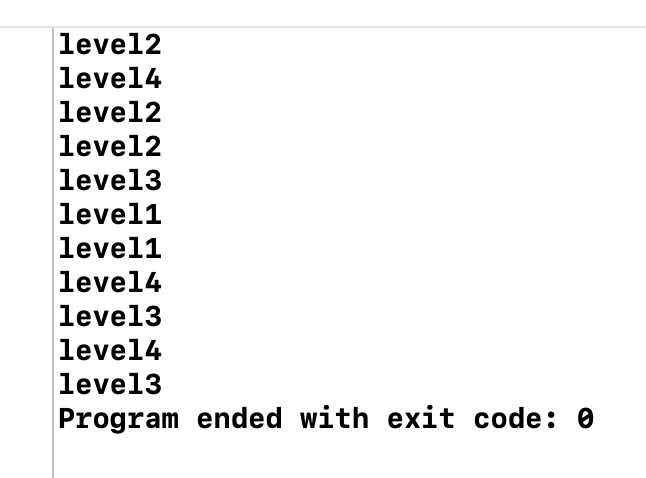
So I can count for frequency :

“level1” : 2

“level2” : 3

“level3” : 3

“level4” : 3



**Q4**

In this case, the &status is the address of status of child process. If child process if finish, then the memory will be released, in such case, the address of child process will be nothing. Hence, will return -1.

Therefore, we can do the same thing using for loop as well, because the logic behind this is actually TRUE and FALSE. Let TCpid = &pid of child, if pid of child is not finish, it will have value. Once it’s finish, it will become -1. So with for-loop, we can do the same thing as wait, where if TCpid is not -1, then wait. If it’s -1, then execute remaining code.

Text

Description automatically generated

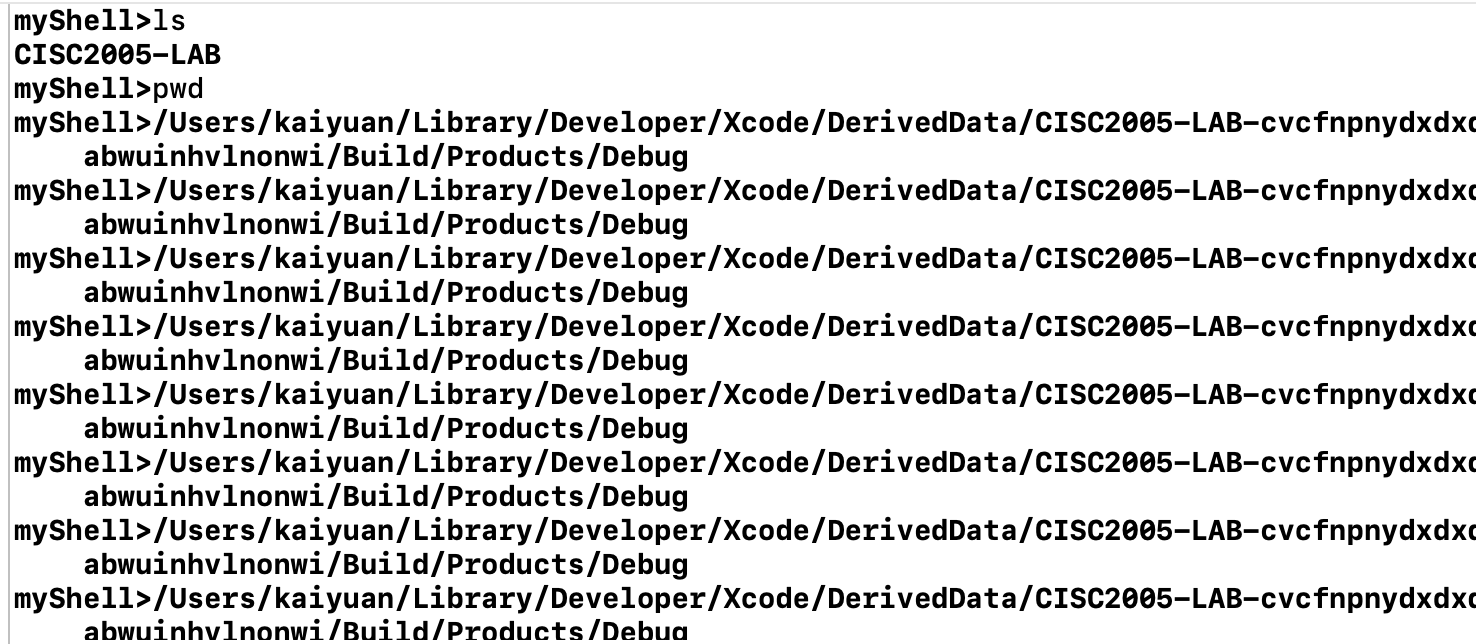
**Q5**

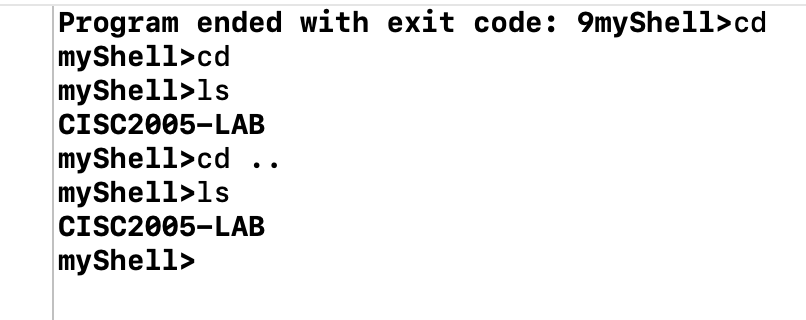


Nope, it’s impossible for the “Child Complete” printed before execution of subprocess.

This is because the wait(null) will block the parent process and wait for the child process to finish. In this case, the “child complete” wont get execute until the subprocess under condition of (pid==0) is finished.

**Q6**





**Q7**

It will not be consistent, because there is no wait() code on parent, so the code will sometimes start with child process, sometime start with parent process. Hence, the content will not be consistent.

