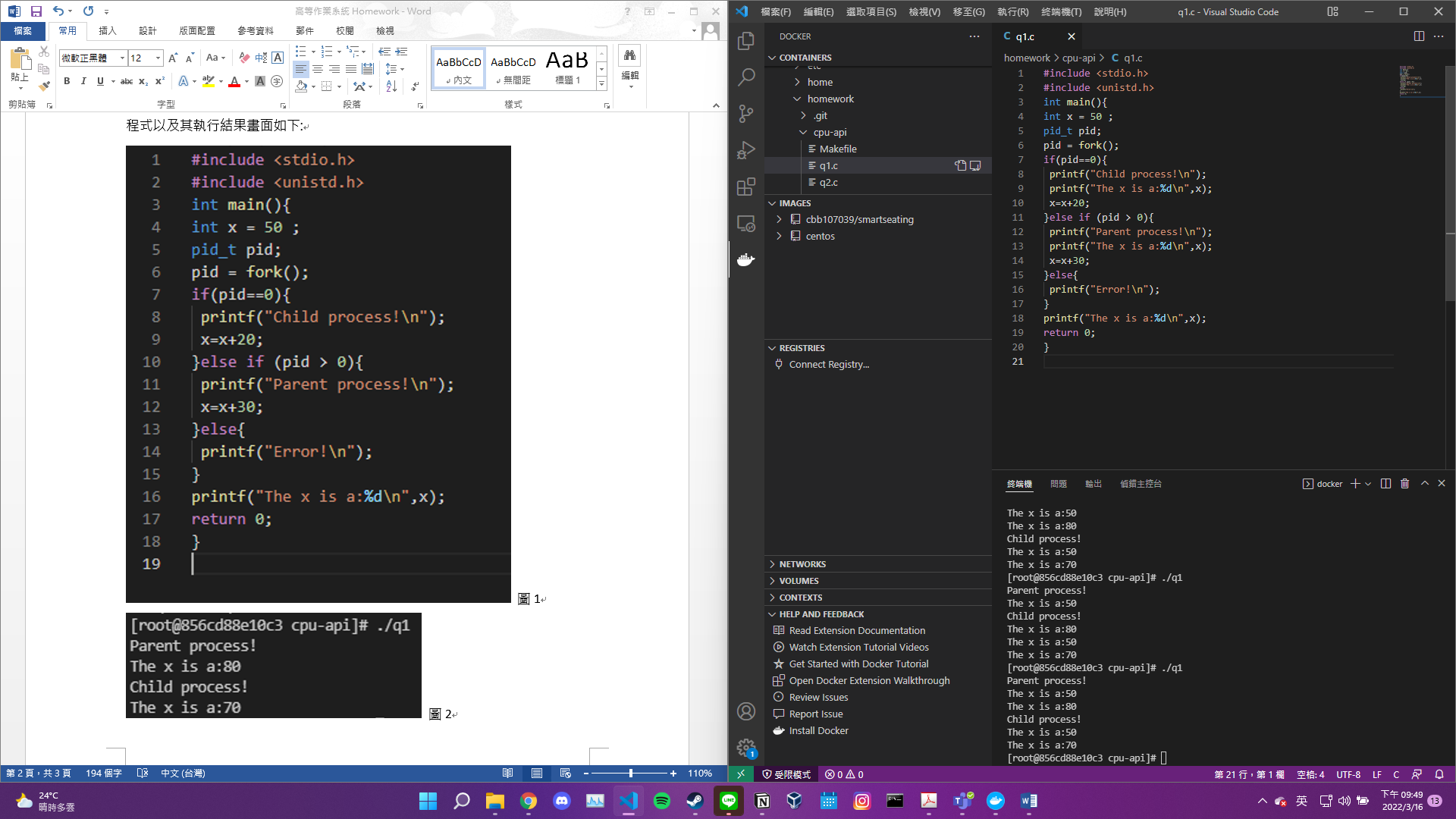
**高等作業系統 Homework(中文版)**

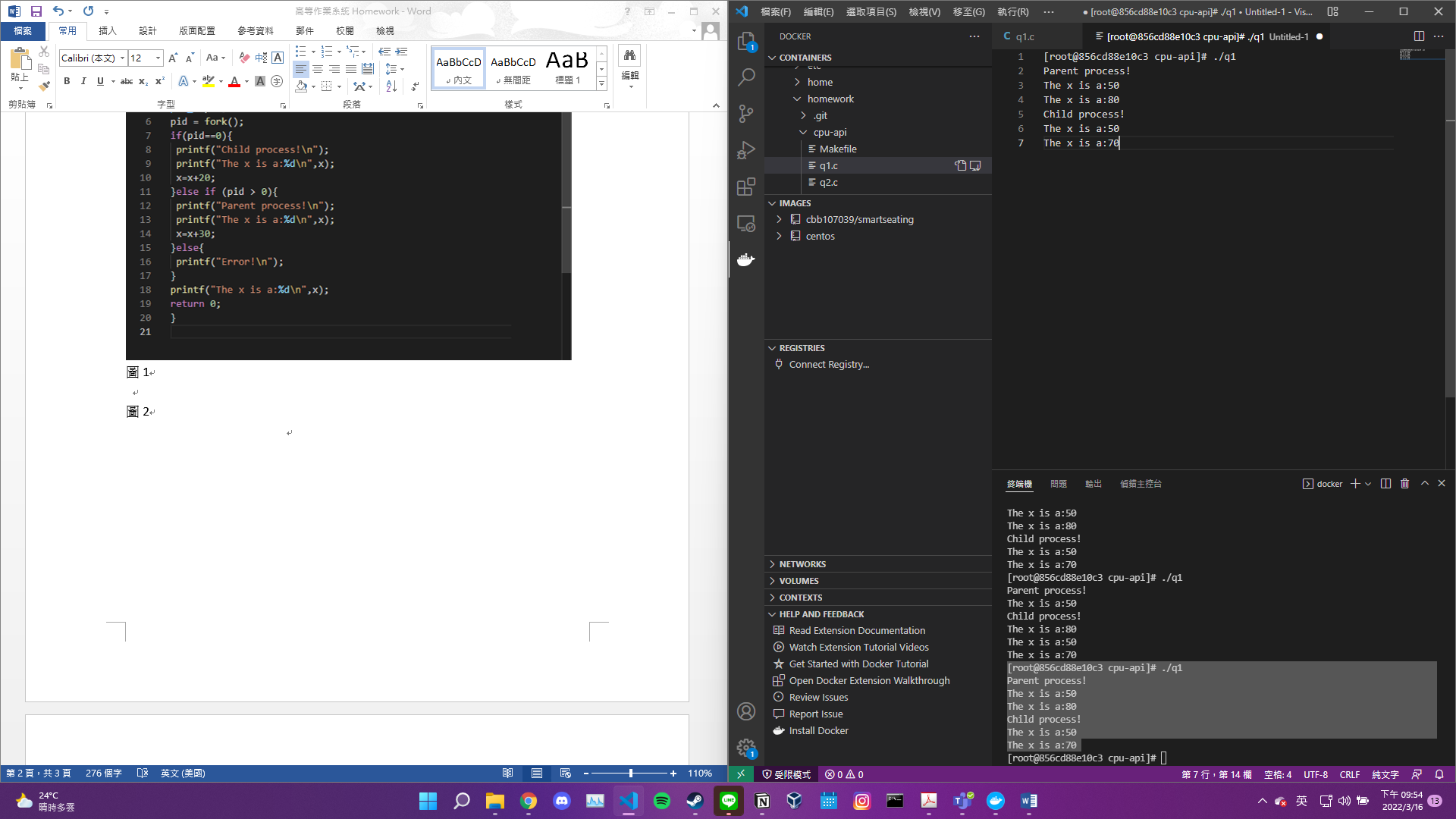
**CBB107039 李尚昇**

**Q1:**

設計一個會呼叫fork()的程式，在呼叫fork()前，建立一個變數x並且設定任意數值，請問在child process x變數為何?而當Child以及Parent對x變數進行數值更動時會發生甚麼事情?

程式以及其執行結果畫面如下:

 圖1

 圖2

**A1:**

圖1程式中，fork被呼叫於第6行，而在第5行中我們先設置一個變數x並設定其為int value為50，在fork被呼叫的同時也會複製第6行之前的所有細節(stack code等等….)，我們分別在9、13、18行都使用printf()印出x的數值在圖2第3、4、6、7行中，而Child的x值可以透過圖2的第6、7行得知。

圖1程式中，Child以及Parent分別在程式第10、14行對x變數進行更動，在圖2第3、4行以及第6、7行中我們可以看到原本x為50因為圖1中第10、14的關係Child中x數值變成:70，Parent中x數值變成80。

**Q2:**

設計program會開啟一個file(使用open() system call)並呼叫fork()建立新的process，請問Child以及Parent執行open會不會遇到問題?而當child以及parent同時對檔案進行寫入會發生甚麼事情?

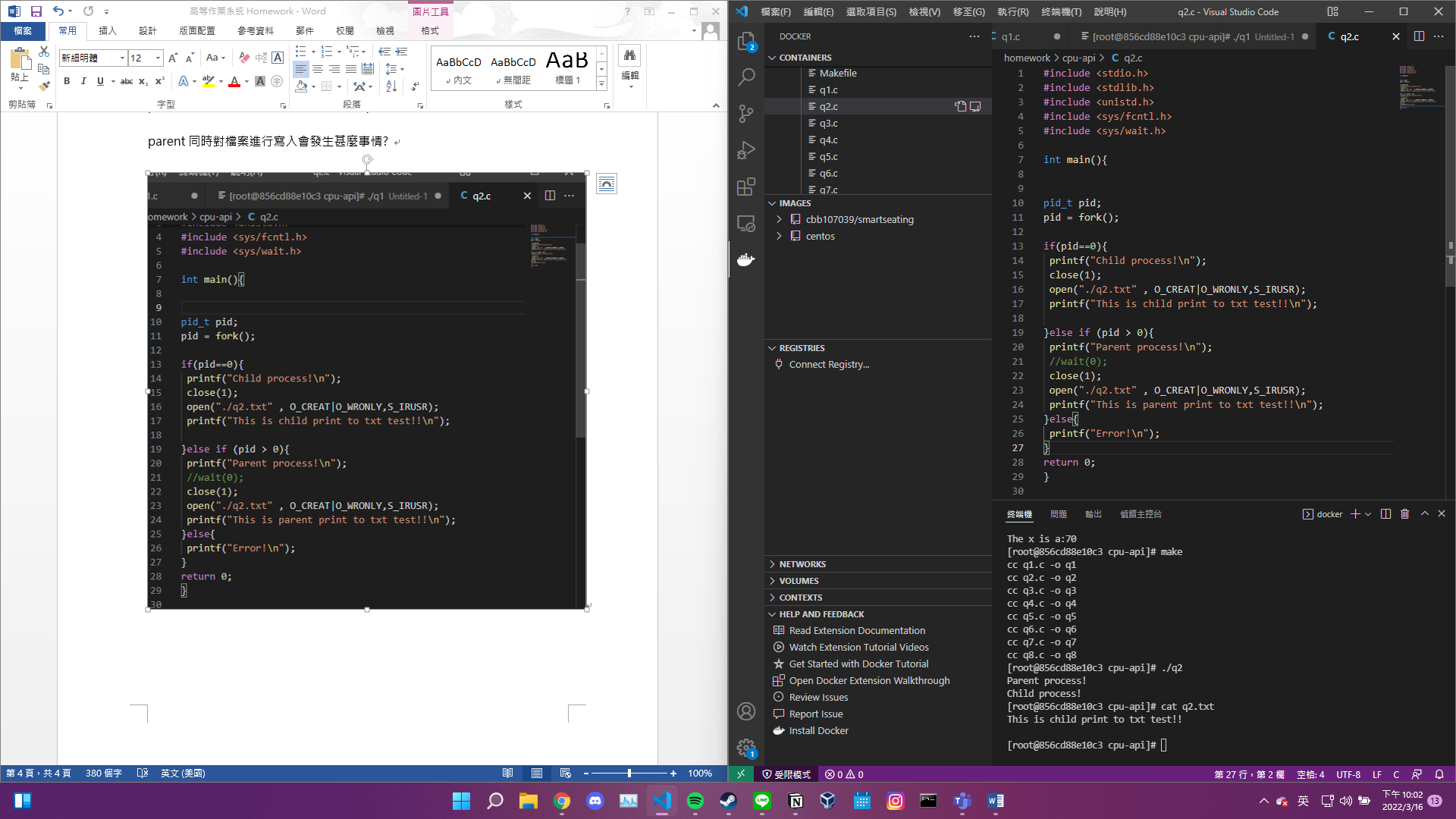


圖3

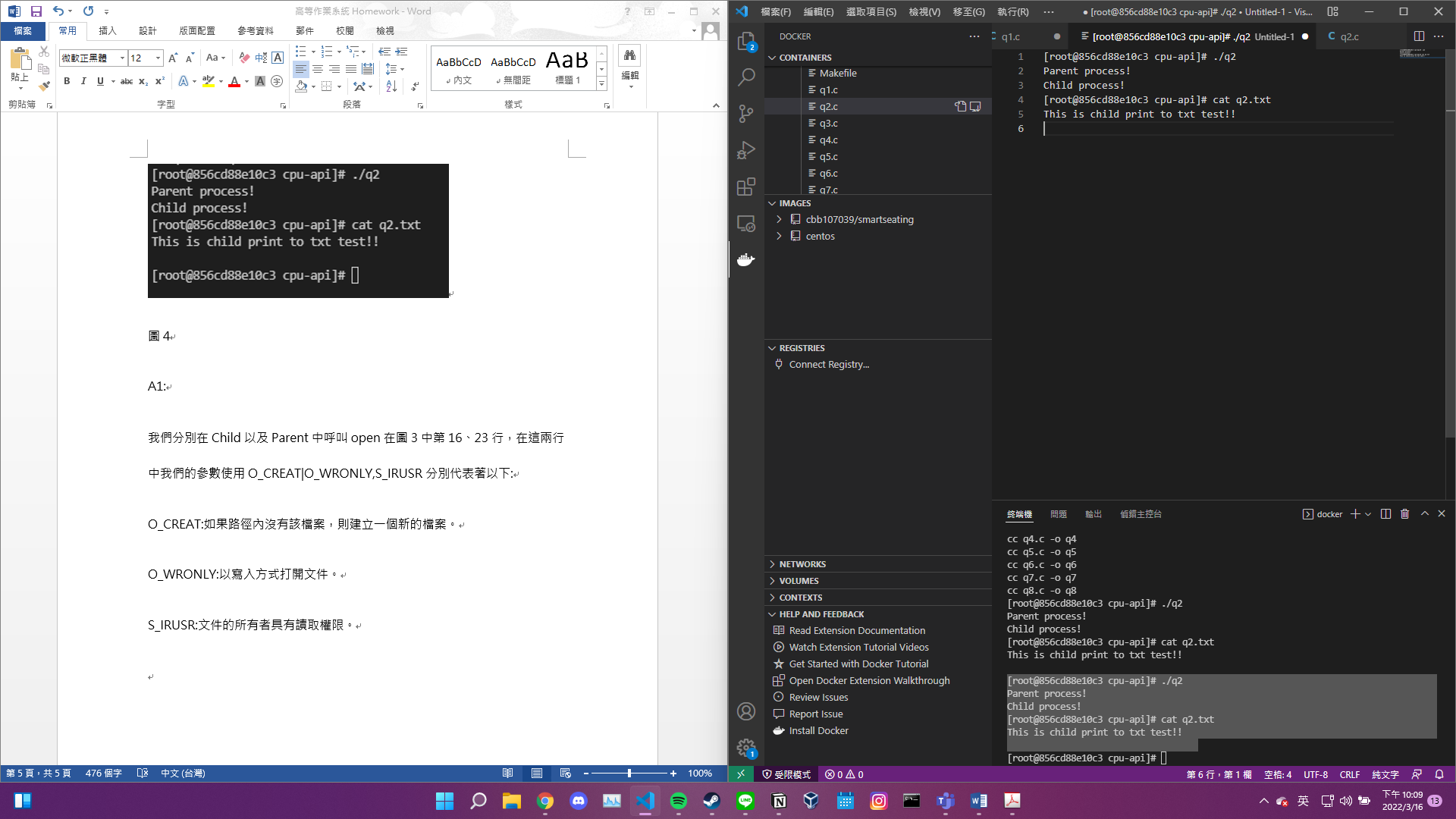


圖4

A1:

我們分別在Child以及Parent中呼叫open在圖3中第16、23行，在這兩行中我們的參數使用O\_CREAT|O\_WRONLY,S\_IRUSR分別代表著以下:

O\_CREAT:如果路徑內沒有該檔案，則建立一個新的檔案。

O\_WRONLY:以寫入方式打開文件。

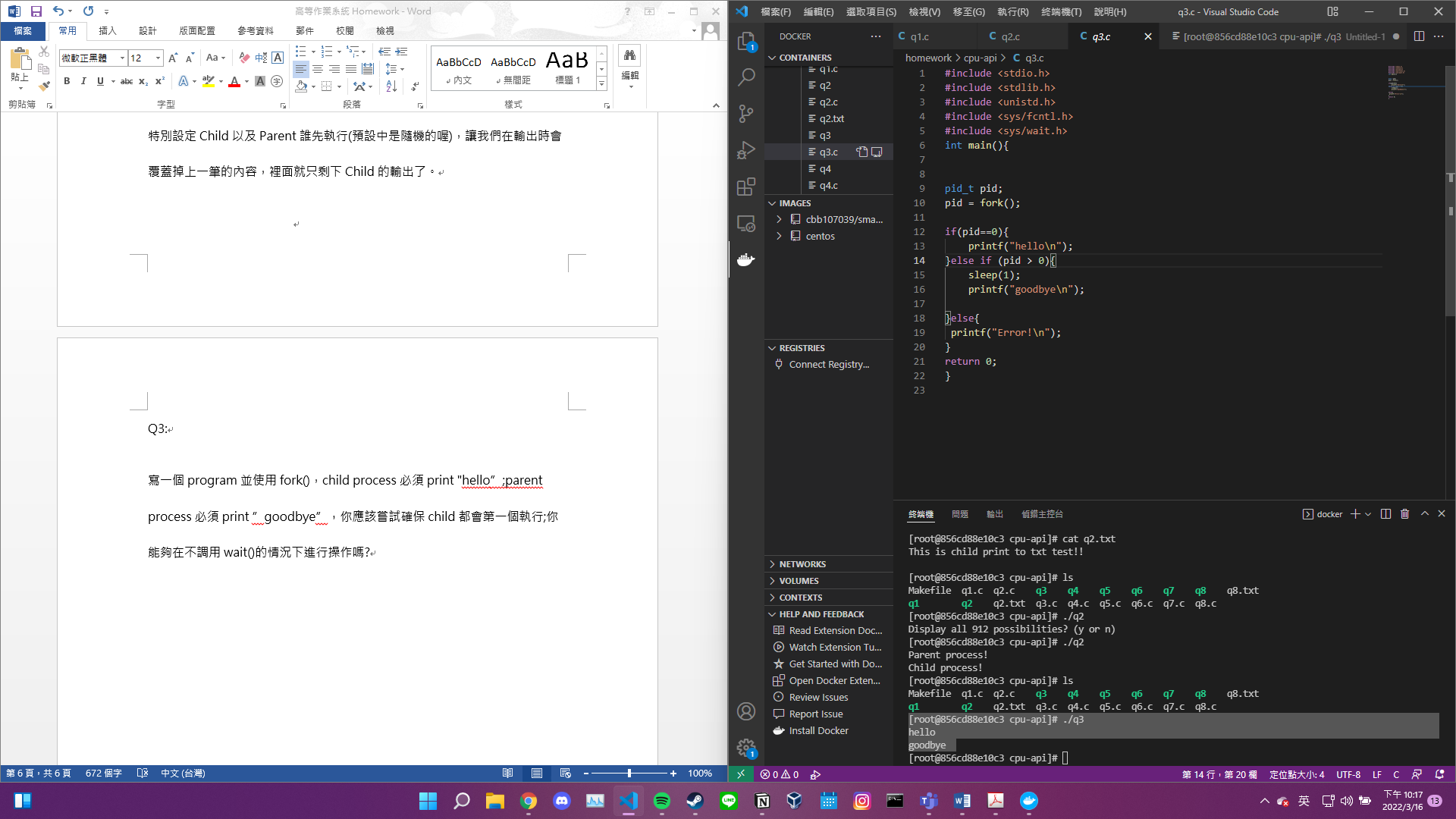
S\_IRUSR:文件的所有者具有讀取權限。

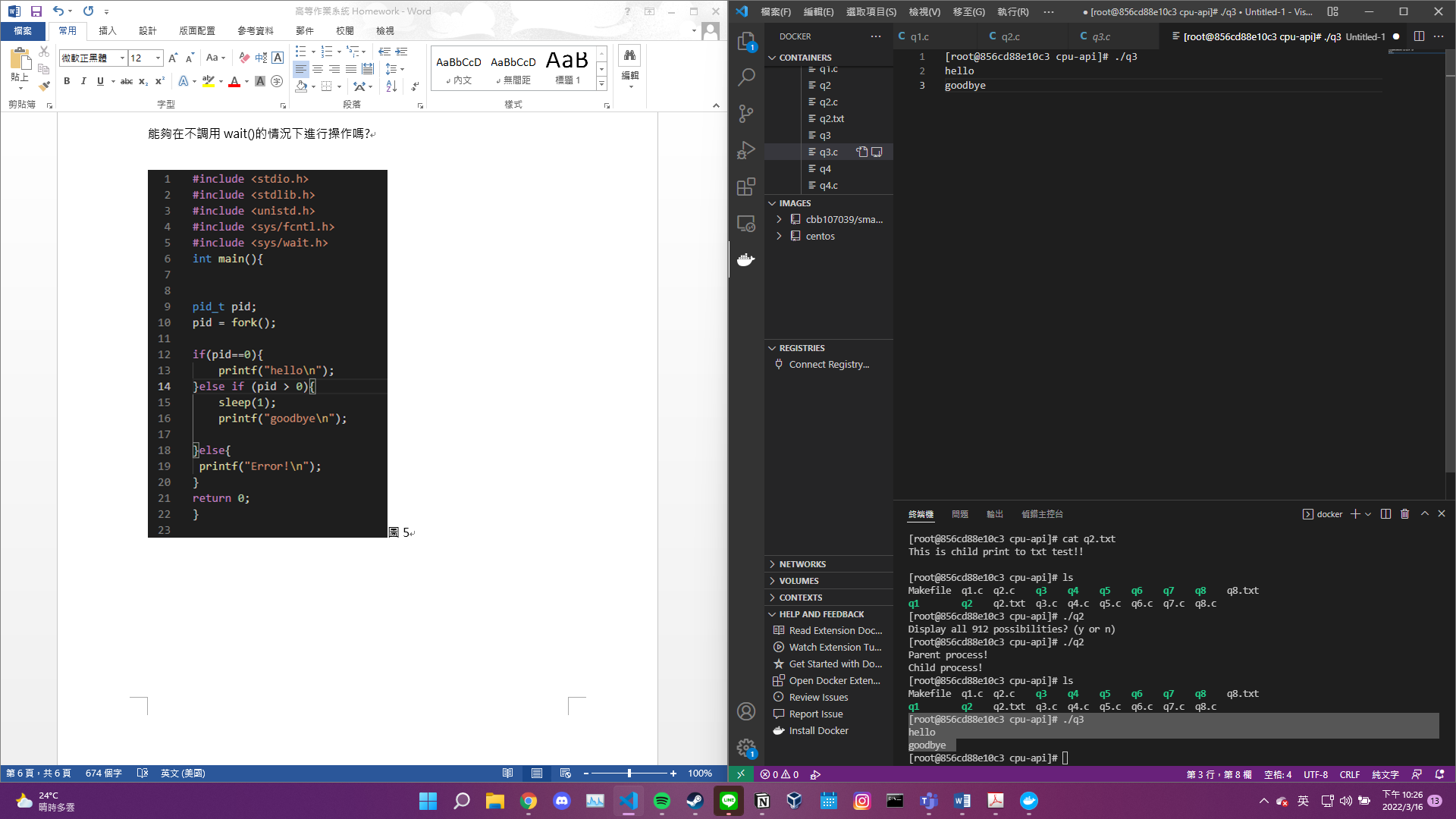
在圖4第2、3行中顯示，Child以及Parent都正常執行完畢。

而當我們open寫入東西後當然要測試一下寫入的結果了，圖4第4行我們使用cat q2.txt對檔案進行擷取，而第5行為執行結果，你會發現由於我們沒有特別設定Child以及Parent誰先執行(預設中是隨機的喔)，讓我們在輸出時會覆蓋掉上一筆的內容，裡面就只剩下Child的輸出了。

Q3:

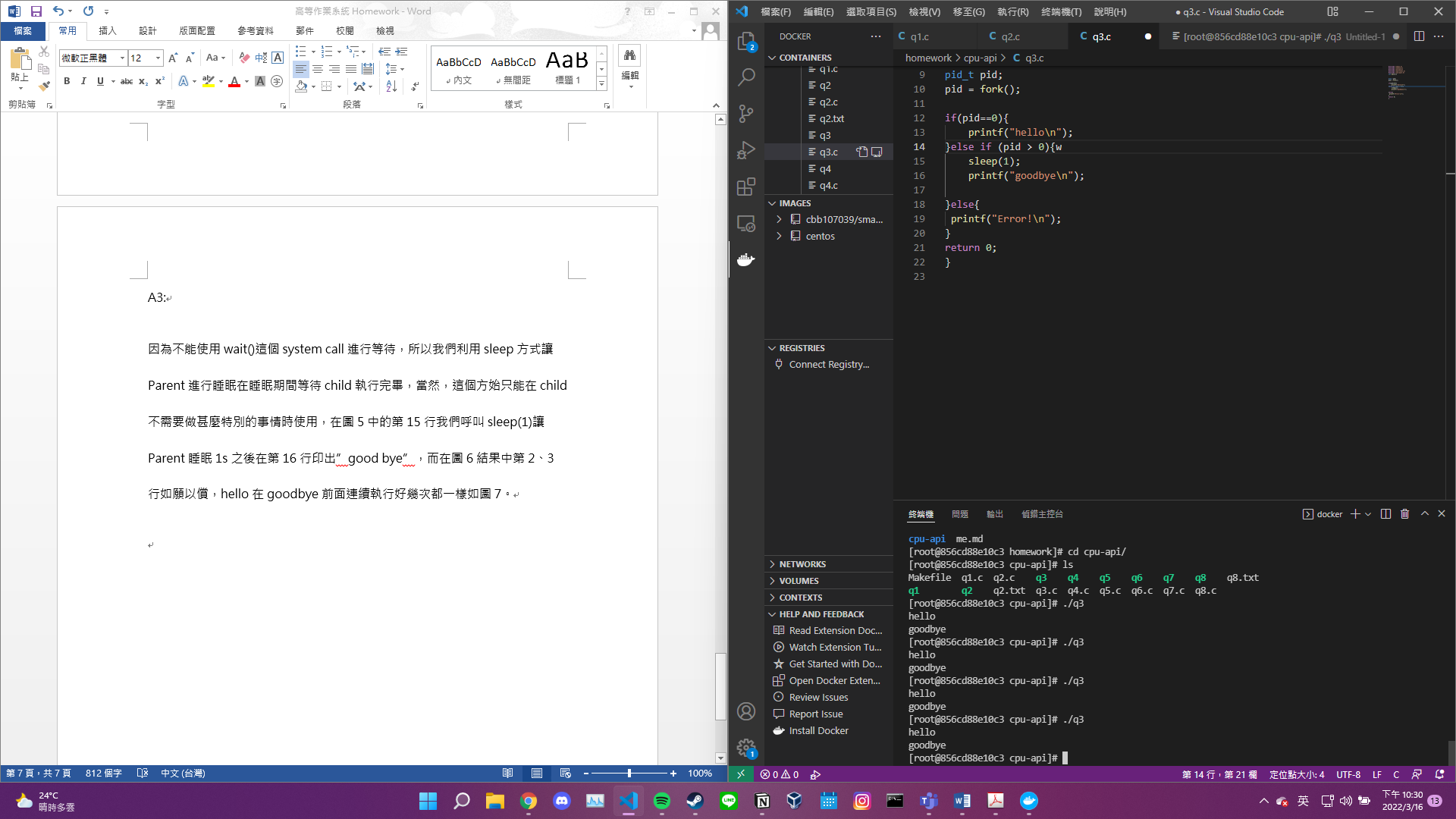
寫一個program並使用fork()，child process必須print "hello”;parent process必須print ”goodbye”，你應該嘗試確保child都會第一個執行;你能夠在不調用wait()的情況下進行操作嗎?

圖5

圖6

A3:

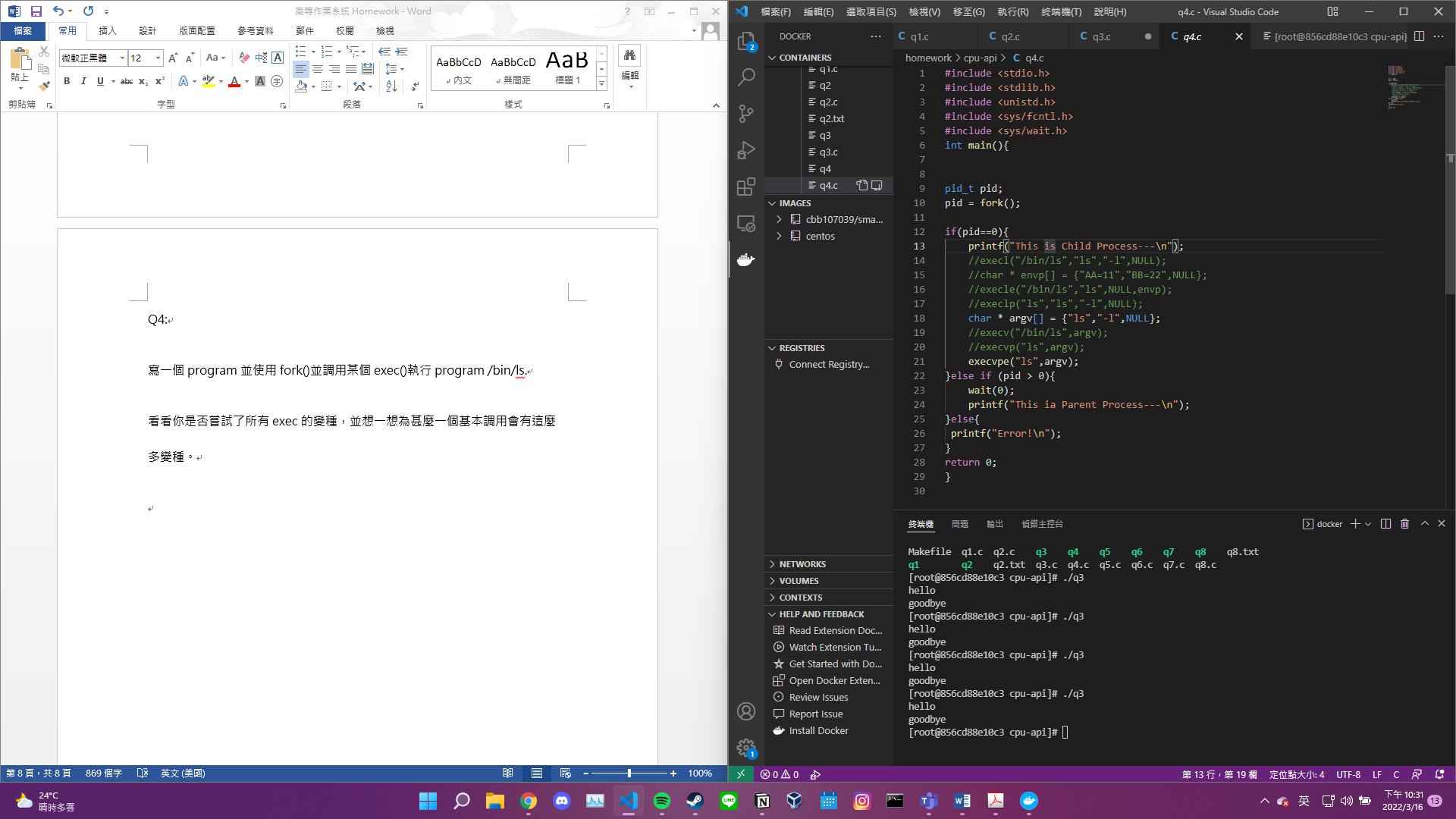
因為不能使用wait()這個system call進行等待，所以我們利用sleep方式讓Parent進行睡眠在睡眠期間等待child執行完畢，當然，這個方始只能在child不需要做甚麼特別的事情時使用，在圖5中的第15行我們呼叫sleep(1)讓Parent睡眠1s之後在第16行印出”good bye”，而在圖6結果中第2、3行如願以償，hello在goodbye前面連續執行好幾次都一樣如圖7。

圖7

Q4:

寫一個program並使用fork()並調用某個exec()執行program /bin/ls.

看看你是否嘗試了所有exec的變種，並想一想為甚麼一個基本調用會有這麼多變種。

圖8

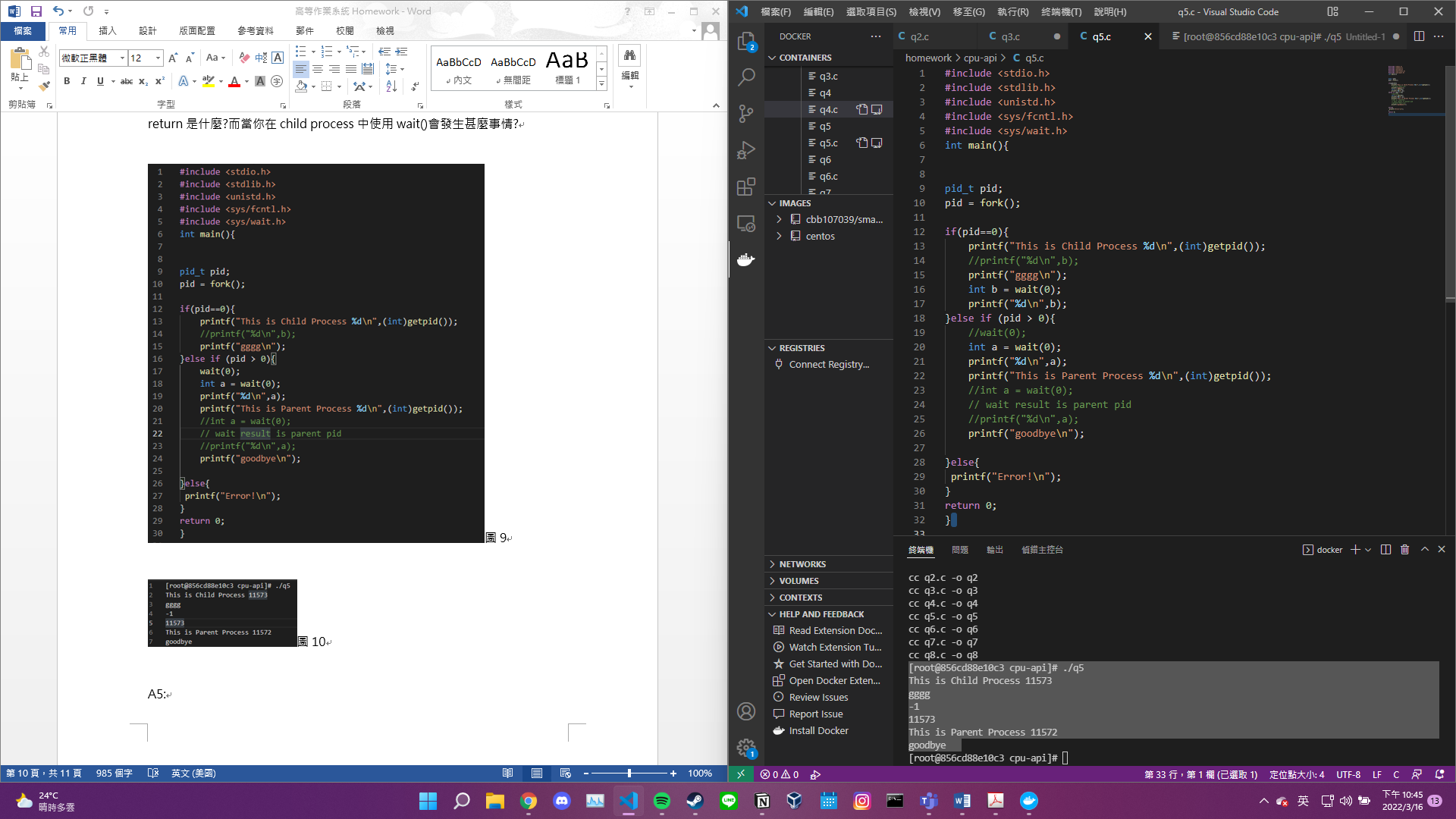
A4:

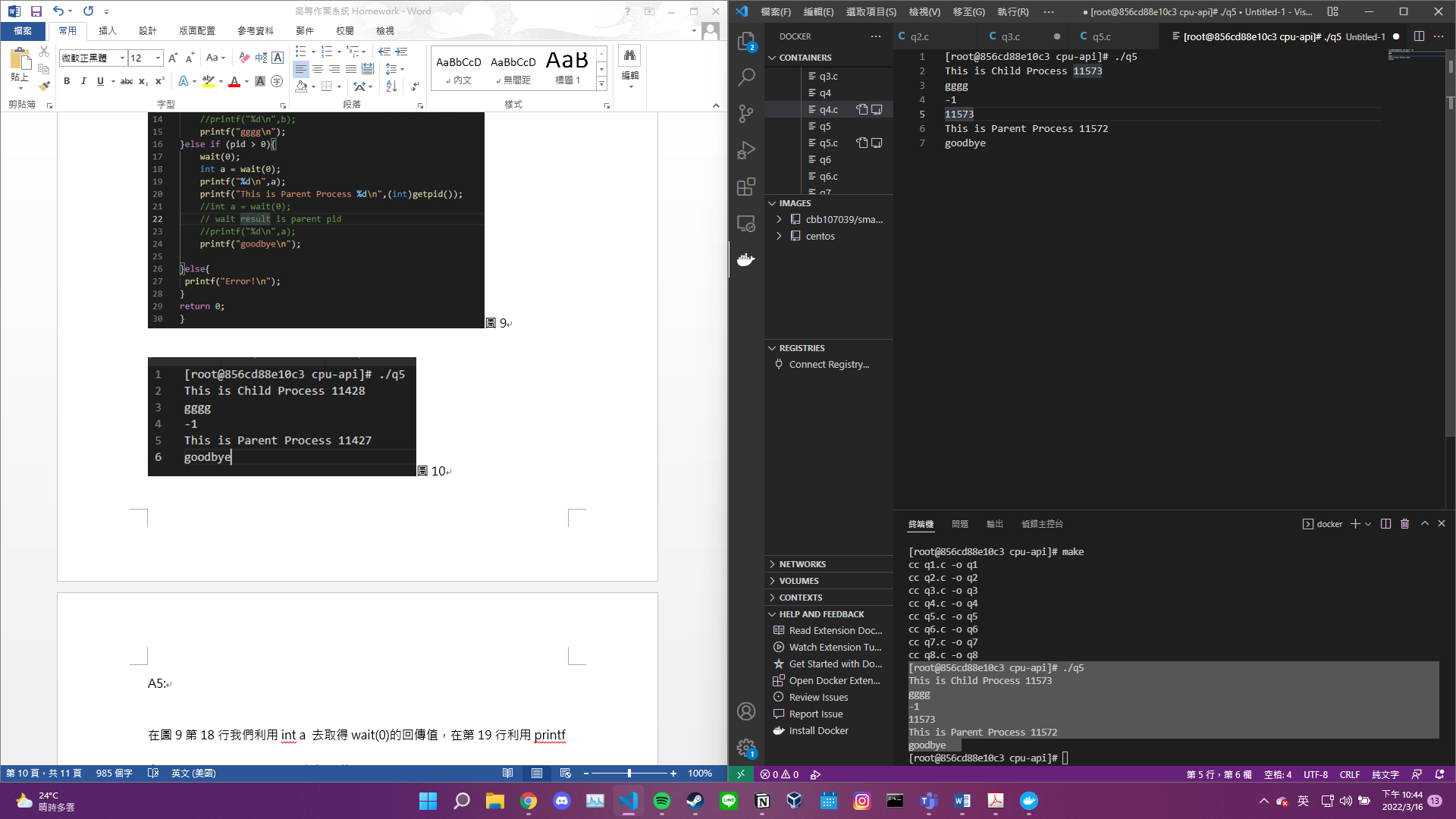
從圖8第14~21行中，我們都在利用exec家族的system call進行ls功能的測試，而為什麼要有這麼多不同的變種呢?

我們可以從每一個call所需要的參數來看看，execl就是把所有的東西都寫在參數裡面而execv可以先將後面的參數利用char \* argv[]的方式先包起來再一起使用，兩種方法各有所好也各有優缺點，在不同的情況下可以選擇不同的exec來進行process的調用並且給予額外的參數進行更多的功能，這是我對於一個basic為什麼會有這麼多變種calls的想法。

Q5:

現在寫一個program並使用wait 讓parent去等待child完成，請問wait()的return是什麼?而當你在child process中使用wait()會發生甚麼事情?

圖9

圖10

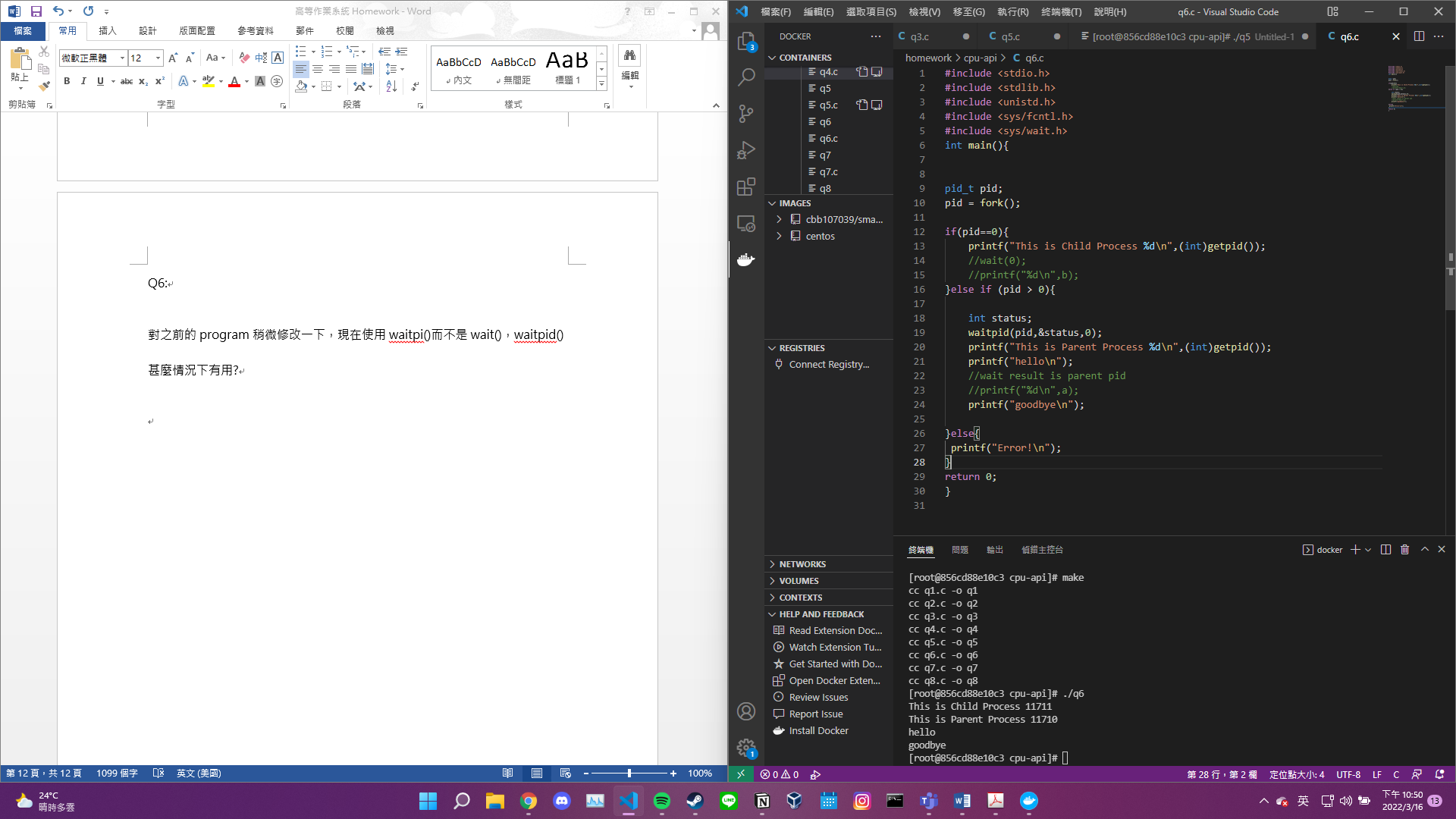
A5:

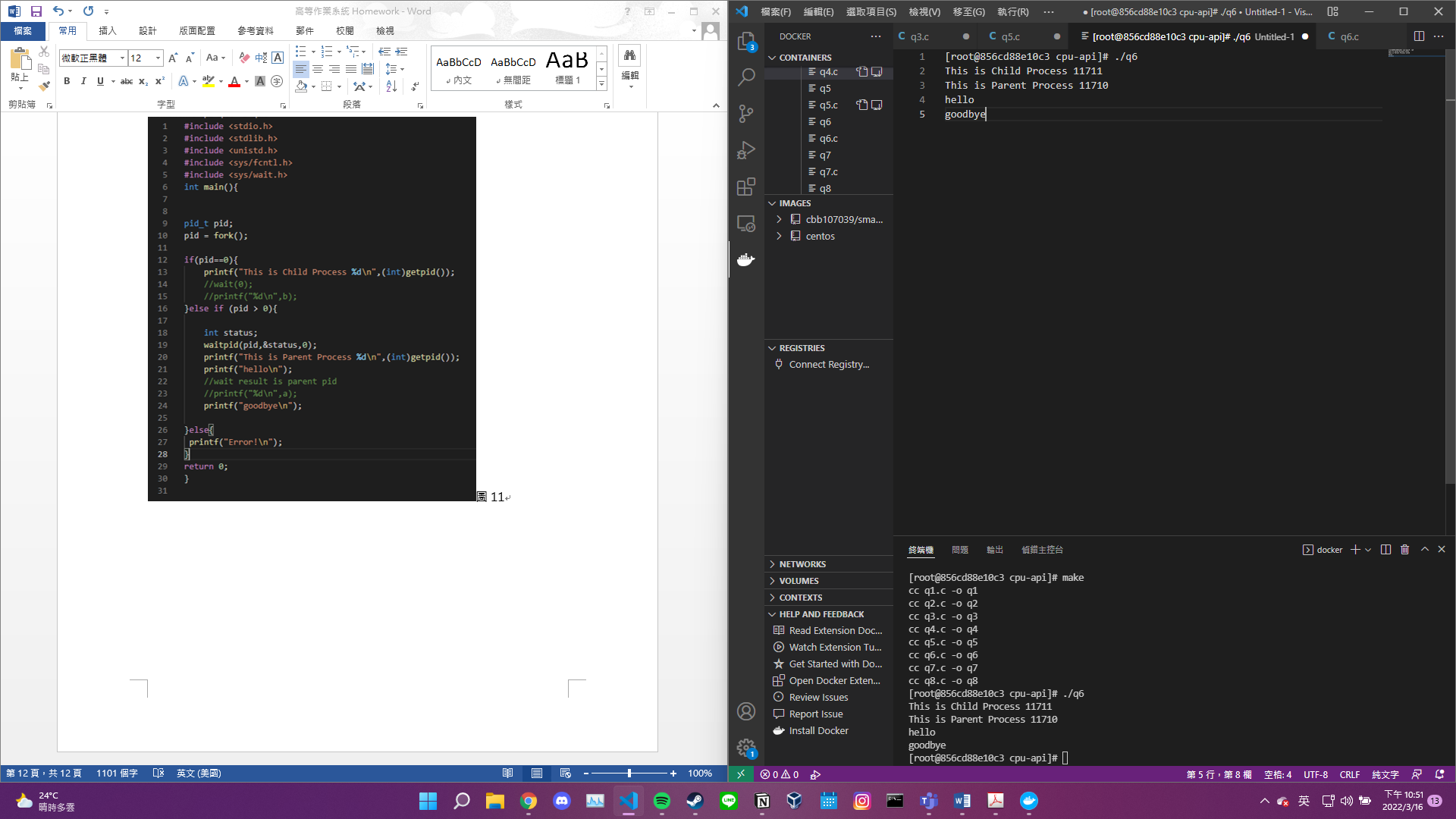
wait()會去尋找有沒有正在執行的Child，如果沒有的話會回傳-1的錯誤值，在圖9第20行我們利用變數a去承接wait(0)的return值並在第21行printf出來，而結果也在圖10中的第5行顯示出來，為Child的PID。

而在圖9第16行我們也同樣在Child裡面使用wait(0)並且利用變數b承接wait的return數值，而結果也如我們預期的出現錯誤在圖10第4行中顯示-1之錯誤值。

Q6:

對之前的program稍微修改一下，現在使用waitpi()而不是wait()，waitpid()甚麼情況下有用?

圖11

圖12

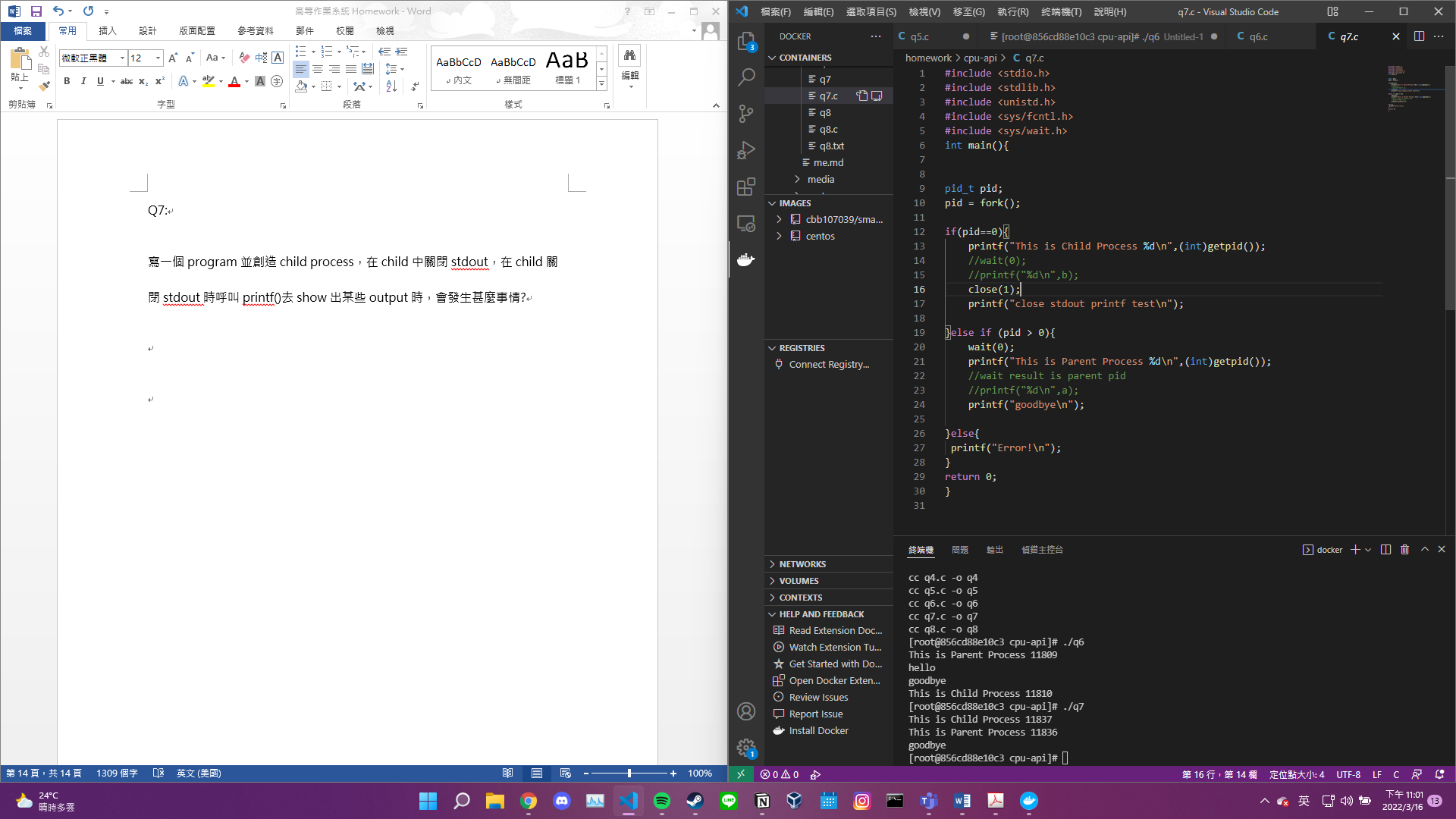
A6:

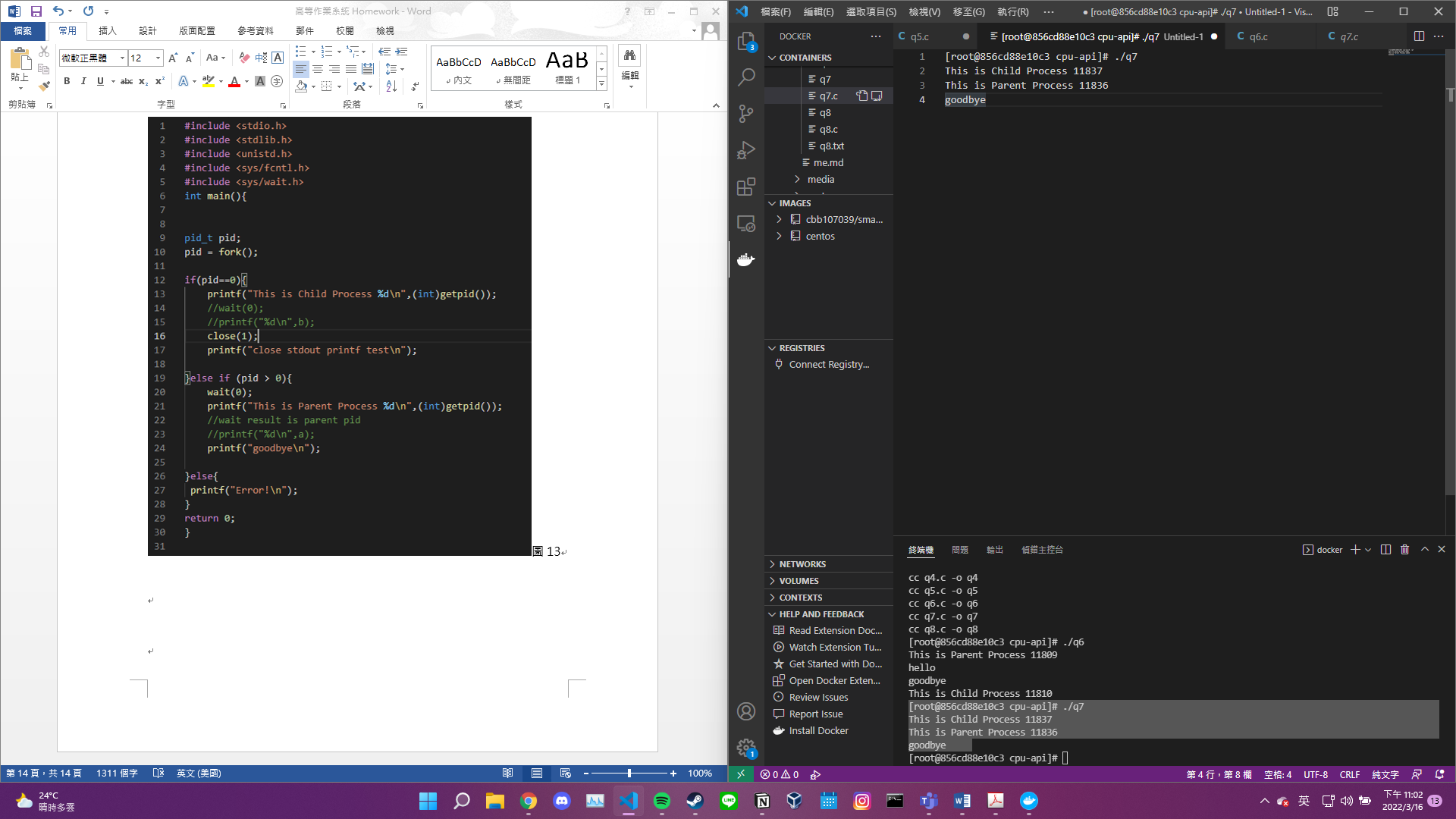
在圖11第19行，我們將原本使用的wait(0)替換成waitpid(pid,&status,0)，在圖12的結果也如我們預期的Child先執行完畢之後才執行Parent的內容，而waitpid跟wait差在哪裡?

Wait會等待子程序執行完畢，顧名思義Parent一定會等到Child執行完之後才執行而waitpid有思考到有些事情Child在做的時候Parent也可以一起做這一就不用浪費時間，使用waitpid時需要給3個參數，pid、狀態、是否繼續執行(Ex.waitpid(pid,&status,1))，第三個參數決定是否繼續執行。

Q7:

寫一個program並創造child process，在child中關閉stdout，在child關閉stdout時呼叫printf()去show出某些output時，會發生甚麼事情?

圖13

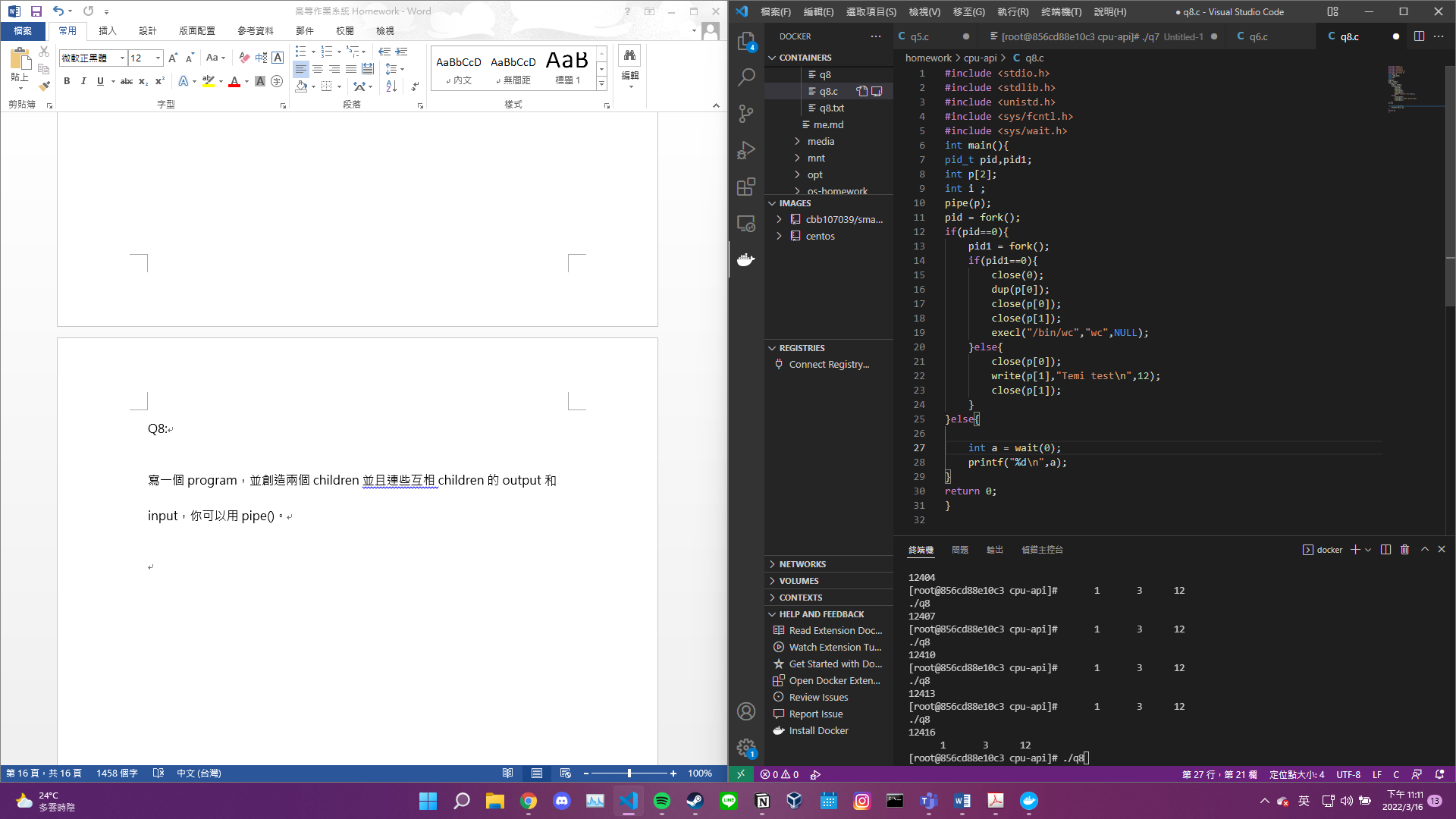
圖14

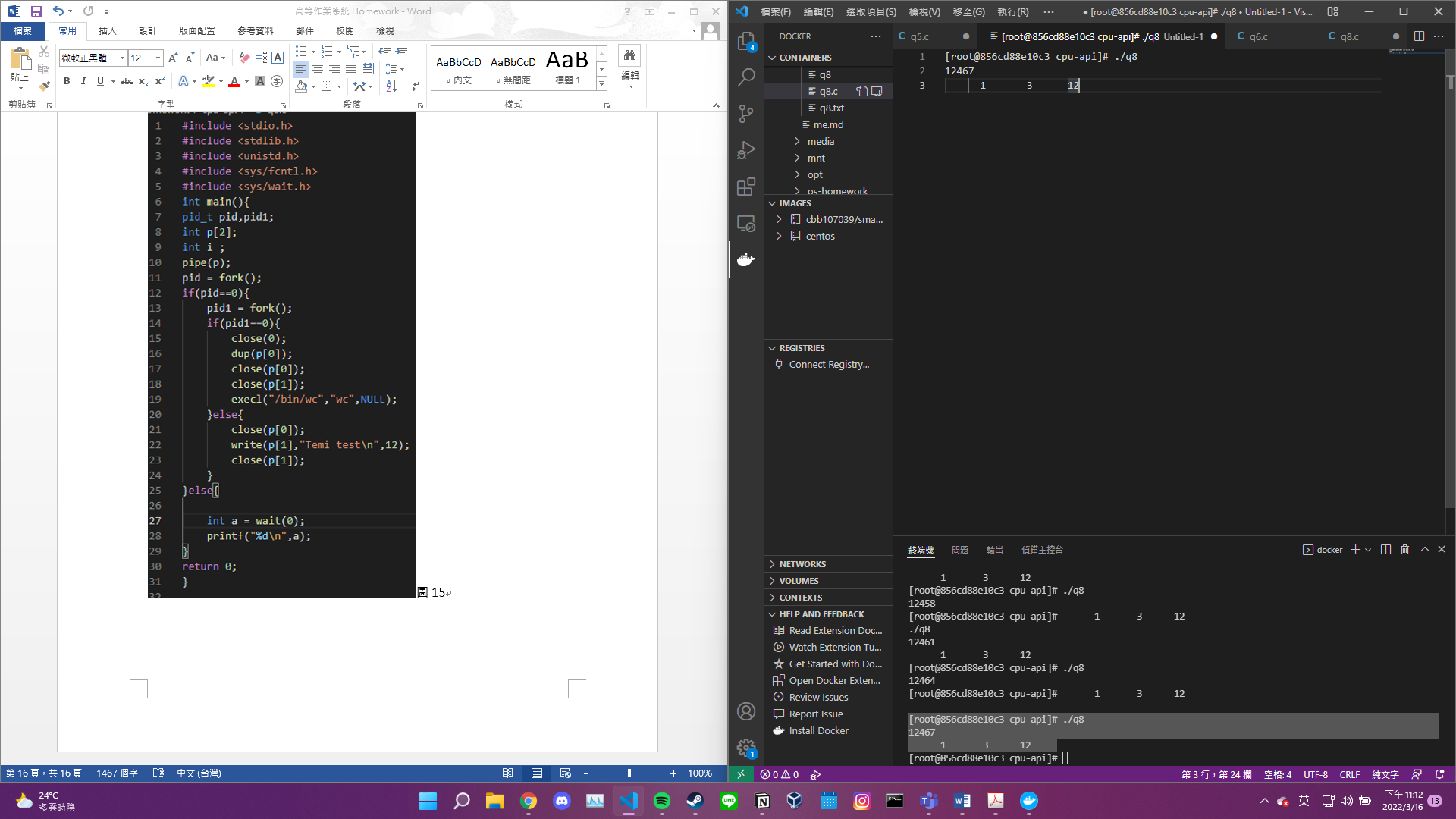
A7:

在圖13第16行中，使用close(1)關閉stdout，並且在17行利用printf印出一段話查看會有甚麼變化，輸出結果在圖14，結果發現Child在執行時只會印出close前的句子，而一般stdout會顯示在command line上面，因為我們關閉stdout的關係讓輸出沒有地方可以顯使導致圖13第17行的結果顯示不出來。

Q8:

寫一個program，並創造兩個children連結其中一邊的輸入到另外一邊的輸入中，你可以用pipe()。

圖15

圖16

A8:

我們分別在圖15中第11、13行建立了兩個fork達到建立兩個子程序，而為了要將兩個子程序其中一端輸出接到另外一端輸入我們在第8行中建立一個空陣列p大小為2，在第10行中利用pipe()對p陣列進行設定。

在圖15第21~23行中，我們先將child的pipe輸入關閉，對pipe輸出寫入”Temi test\n”的文字之後關閉pipe輸出。

在14~19行我們依序執行，關閉child2中的stdin，並且利用dup將空白的std table指向p[0]的位置也就是把stdin指向p[0]，而因為不會再用到p陣列的東西所以關閉p[0]以及p[1]，因為第22行中已經寫入一個字串到這邊的輸入端了我們從第19行執行execl時就會自動帶入剛剛寫入的數值，就有點像echo Temi test | wc，也就是pipe功能了喔!!，那圖16輸出結果也如我們預期的一樣。