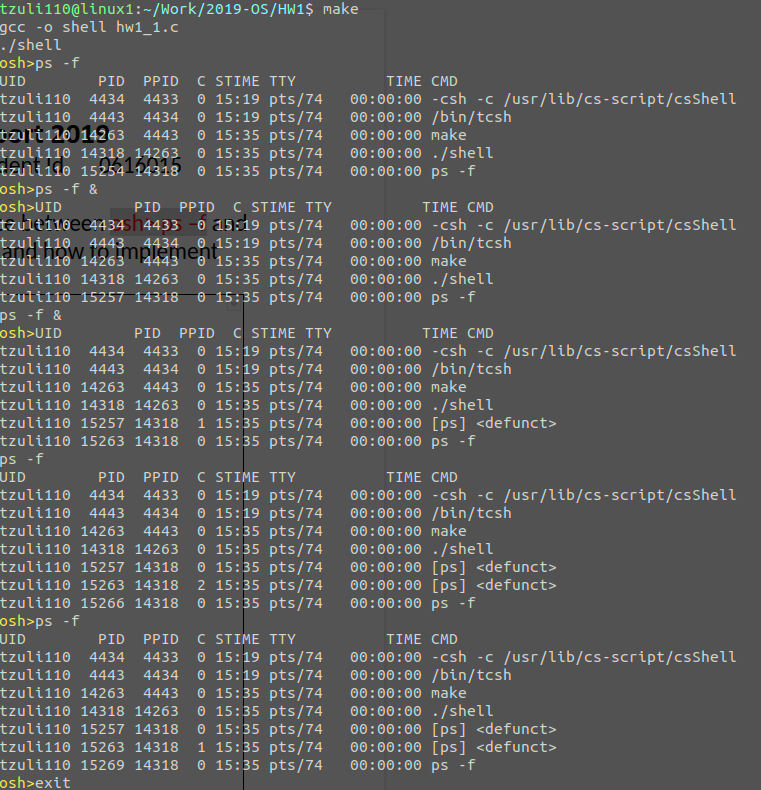
**NCTU OS HW1 report 2019**

|  |  |  |  |
| --- | --- | --- | --- |
| Name | 劉姿利 | Student Id | 0616015 |

**Q1. hw1\_1**

Run your code and show the difference between osh> ps –f and osh>ps –f &. You also need to explain and how to implement “&”.

以下是執行一次 ps -f 兩次 ps -f & 再執行兩次 ps -f 的結果：



Differences

* 沒有下 & 的話下一次的 prompt **osh>** 會在 ps -f 執行完成之後才出現
* 有下 & 的 command 會產生出 defunct process 並顯示在 ps 中

Implementation

建立一個 opt\_wait 變數，指示 shell 是否要 wait for child process。

|  |
| --- |
| **int** opt\_wait = 1; |

在 **while** 迴圈一開始的部份都將這個變數設為 1。

|  |
| --- |
| **while** (should\_run) {  printf("osh>");  fflush(stdout);  opt\_wait = 1;  ... |

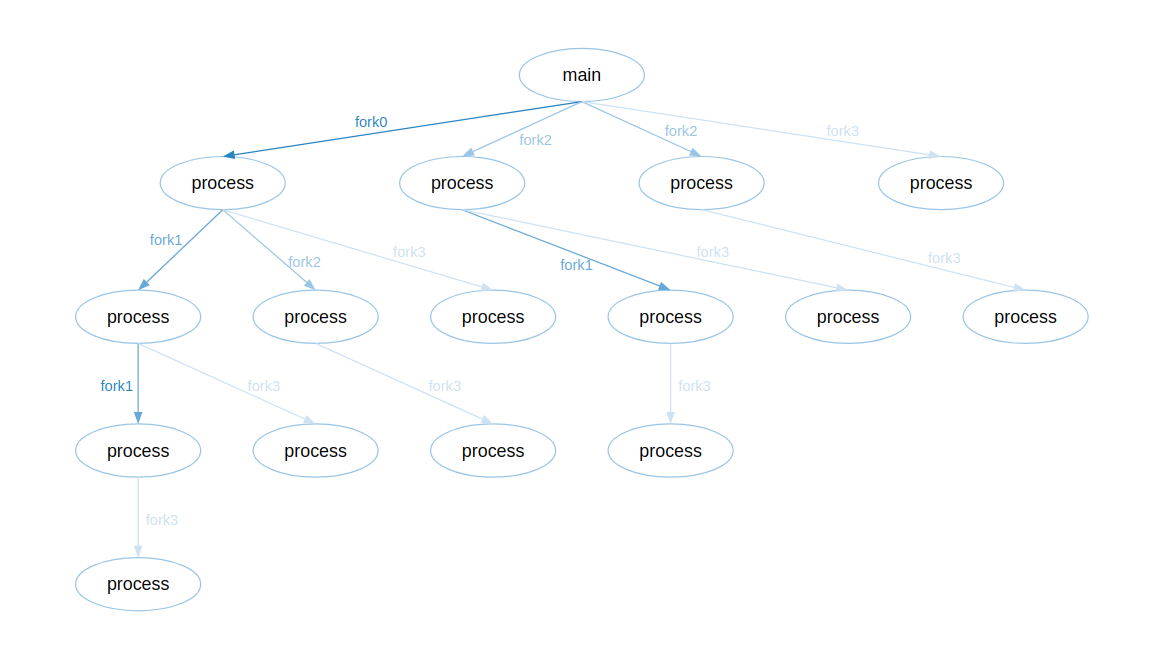
在 fork() 結束後 parent 要繼續執行的部份，以 opt\_wait 決定是否要 wait。

|  |
| --- |
| *// parent* **if** (pid > 0) {  **if** (opt\_wait) {  waitpid(pid, &status, 0);  status = -1;  }  *// clean*  **for** (**int** i = 0; i < arg\_len; i++)  arg[i] = NULL;  **continue**; } |

這裡使用 waitpid() 而不使用 wait()，是為了讓 shell process (parent) 只等待「這次迴圈 fork 出的 child process」，才不會因為上次迴圈 fork 出的 child process exit 而結束這次 wait 的動作。用法及參數配置參考了 [Linux Man Page](http://man7.org/linux/man-pages/man2/waitpid.2.html)。

**Q2. hw1\_2**

Tree format(according OS\_hw1.pdf/p.12):

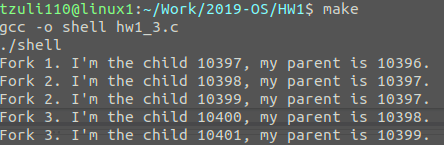


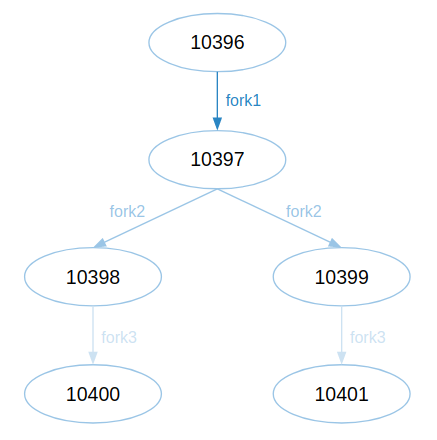
**Q3. hw1-3**

Please put your result(screen shot).

Also write down the problems you met and solutions.

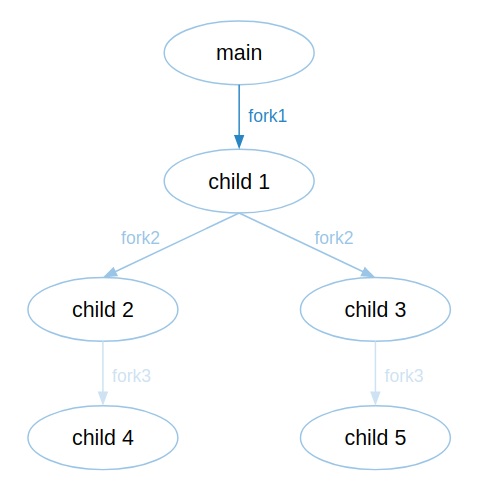
Result





Problems & Solutions

Notations:



Problem 1

如何只讓 child1 產生出兩個 fork2

Solution 1

在第一個 fork() 之後產生了 child1，使用 getpid() 將 child1 的 pid 存下來

|  |
| --- |
| **int** child1\_pid = getpid(); |

接著在需要判斷是否要執行 fork2 的時候加上 getpid() == child1\_pid 的條件

|  |
| --- |
| **int** child1\_pid = getpid(); **for** (**int** i = 0; i < 2; i ++) {  **if** (getpid() == child1\_pid) {  pid = fork(); *// fork 2*  ...  } |

Problem 2

要確保 child2 和 child3 在 print statement 時 child1 還沒結束，不然 child2 和 child3 會變成 defunct、parent pid 會變成 1。

Solution 2

在 fork2 之後 child1 (parent process) 要 wait child2 和 child3

|  |
| --- |
| **for** (**int** i = 0; i < 2; i ++) {  **if** (getpid() == child1\_pid) {  pid = fork(); *// fork 2*  **if** (pid == 0) {  print(2);  }   } } *// child1 wait for child2 or child3* **if** (getpid() == child1\_pid) {  **int** status = -1;  waitpid(pid, &status, 0); } |

這個 Problem 在 fork3 的時候也會有，所以 child2 也要等待 child4、child3 等待 child4

|  |
| --- |
| pid = fork(); *// fork 3* **if** (pid == 0) {  print(3); } **else** {  **int** status = -1;  waitpid(pid, &status, 0); } |

Problem 3

要確保執行 fork3 前 fork2 都執行完了

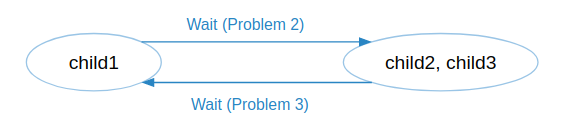
Solution 3

相當於 child1 執行完兩個 fork2 之後 child2 和 child3 才執行 fork3。

*Approach 1: Wait (Failed)*

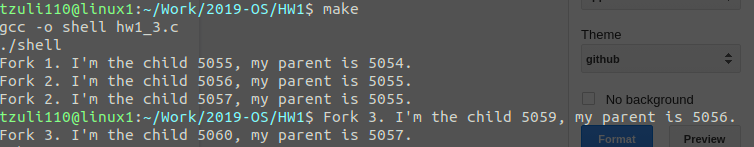
第一個想法是 child2 和 child3 要 wait child1，等待 child1 執行完畢後才執行 fork3。

但考慮到 Problem 2，這樣會形成沒有用的 deadlock。



*Approach 2: Sleep (Failed)*

使用 sleep(1) 讓 child2、child3 在 fork3 前等一秒，相信一秒夠讓 child1 執行完兩個 fork2 了。



然而 sleep(1) 會讓程式暫停而讓 tcsh 的 prompt 跑出來。

*Approach 3: For Loop*

|  |
| --- |
| **for** (**int** i = 0; i < 10000000; i++) {} pid = fork(); *// fork 3* |

在 fork3 前加個耗時較久的 **for** 迴圈總算成功了。