

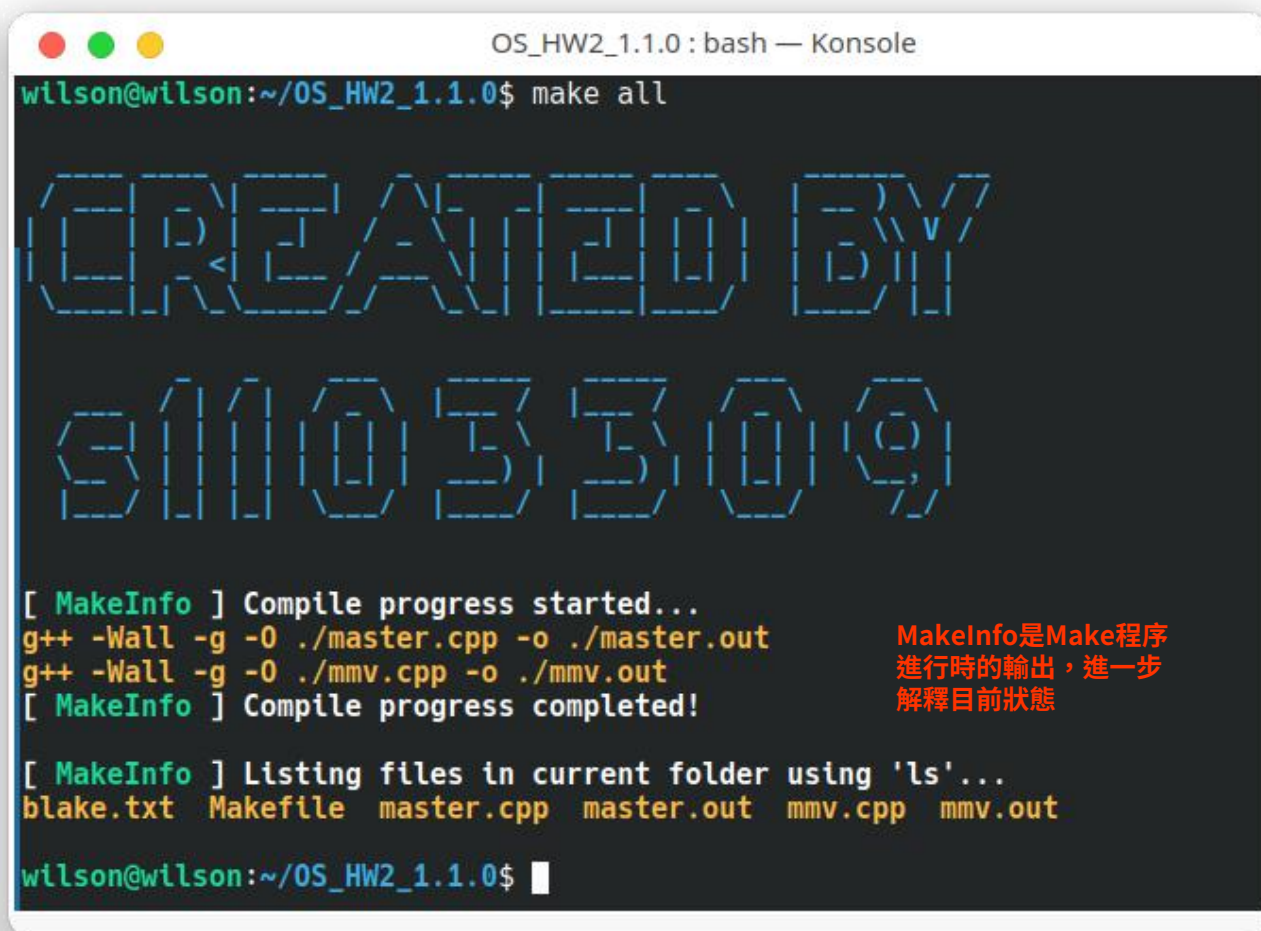
## 一、測試建置環境

- Distro : [KDE Neon 5.26](#) based on [Ubuntu 20.04](#)
- Toolchain : [GNU make 4.2.1](#) 、 [g++ 9.4.0](#)
- Shell : [bash 5.0.17](#) with [KDE Konsole](#)

## 二、測試執行擷圖（紅字為解說）

執行的輸出大多與作業1相同，完全相同部份將不再多做講解。  
此次作業著重於過程的不同，將會偏重於實做方面的講解。

- 執行 make all



```
OS_HW2_1.1.0 : bash — Konsole
wilson@wilson:~/OS_HW2_1.1.0$ make all

CREATED BY
s1103309

[ MakeInfo ] Compile progress started...
g++ -Wall -g -O ./master.cpp -o ./master.out
g++ -Wall -g -O ./mmv.cpp -o ./mmv.out
[ MakeInfo ] Compile progress completed!

[ MakeInfo ] Listing files in current folder using 'ls'...
blake.txt Makefile master.cpp master.out mmv.cpp mmv.out

wilson@wilson:~/OS_HW2_1.1.0$
```

MakeInfo是Make程序進行時的輸出，進一步解釋目前狀態

- 執行 make test

```
OS_HW2_1.1.0: bash — Konsole
wilson@wilson:~/OS_HW2_1.1.0$ make test

CREATED BY
91103309

[ MakeInfo ] Test progress started...

[ MakeInfo ] Show Content of 'blake.txt' using 'cat'...
To see a World in a Grain of Sand
And a Heaven in a Wild Flower,
Hold Infinity in the palm of your hand
And Eternity in an hour.

[ MakeInfo ] Listing files in current folder using 'ls'...
blake.txt  Makefile  master.cpp  master.out  mmv.cpp  mmv.out

[ MakeInfo ] Start testing with default output file (happy.tmp)
-----

[ MakeInfo ] Executing program 'master.out'
此處執行指令 ./master

./master.out
[ Success ] (#5690) Successfully connected process to the pipe
[ Success ] (#5690) Parent process is running
[ log ] (#5690) * No argv, use the default files *
[ Process ] (#5690) Trying to open file "./blake.txt"
[ Success ] (#5690) Successfully opened file "./blake.txt"
[ log ] (#5690) Reading file content into buffer
[ Process ] (#5690) Trying to write buffer into the pipe
[ Success ] (#5690) Successfully written data into pipe
[ Success ] (#5691) Child process is created and running
[ Process ] (#5691) Replacing current process image with mmv.out
[ Success ] (#5691) Successfully started mmv.out in process
[ Process ] (#5691) Trying to create file "./happy.tmp"
[ Success ] (#5691) Successfully created file "./happy.tmp"
[ Process ] (#5691) Trying to read data from the existed pipe
[ log ] (#5691) Adding new a line into the buffer string
[ Process ] (#5691) Writing buffer into file
[ Success ] (#5691) Successfully written 167 Bytes into "./happy.tmp"
[ Close ] (#5691) Child process closed with exit code 0
[ Close ] (#5690) Parent process closed
每行開頭顯示 (#PID)
紫色為「父行程PID」、
藍色為「子行程PID」

[ MakeInfo ] Exited program 'master.out'
-----
```

▲ 執行方式符合作業要求 (詳見程式碼)



- 執行 make test (continued)

```
OS_HW2_1.1.0: bash — Konsole

-----
[ MakeInfo ] Show Content of 'happy.tmp' using 'cat'...
\\ ---- Say Hello to s1103309! ----\\
To see a World in a Grain of Sand
And a Heaven in a Wild Flower,
Hold Infinity in the palm of your hand
And Eternity in an hour.

[ MakeInfo ] Listing files in current folder using 'ls'...
blake.txt  Makefile      master.out  mmv.out
happy.tmp  master.cpp  mmv.cpp

*****

[ MakeInfo ] Start testing with user-defined output file
-----
[ MakeInfo ] Executing program 'master.out' with argv

./master.out ./blake.txt ./happy2.tmp  此處執行指令 ./master ./blake.txt ./happy2.tmp
[ Success ] (#5702) Successfully connected process to the pipe  執行程式的同時
[ Success ] (#5702) Parent process is running                輸出運作順序的紀錄
[ log ] (#5702) * argv detected, use user-defined files *      利於找出bug和錯誤
[ Process ] (#5702) Trying to open file "./blake.txt"
[ Success ] (#5702) Successfully opened file "./blake.txt"
[ log ] (#5702) Reading file content into buffer
[ Process ] (#5702) Trying to write buffer into the pipe
[ Success ] (#5702) Successfully written data into pipe
[ Success ] (#5703) Child process is created and running
[ Process ] (#5703) Replacing current process image with mmv.out
[ Success ] (#5703) Successfully started mmv.out in process
[ Process ] (#5703) Trying to create file "./happy2.tmp"
[ Success ] (#5703) Successfully created file "./happy2.tmp"
[ Process ] (#5703) Trying to read data from the existed pipe
[ log ] (#5703) Adding new a line into the buffer string
[ Process ] (#5703) Writing buffer into file
[ Success ] (#5703) Successfully written 167 Bytes into "./happy2.tmp"
[ Close ] (#5703) Child process closed with exit code 0
[ Close ] (#5702) Parent process closed

每行開頭顯示 (#PID)
紫色為「父行程PID」、
藍色為「子行程PID」

[ MakeInfo ] Exited program 'master.out'  ▲ 執行方式符合作業要求 (詳見程式碼)
-----

[ MakeInfo ] Show Content of 'happy2.tmp' using 'cat'...
\\ ---- Say Hello to s1103309! ----\\
To see a World in a Grain of Sand
And a Heaven in a Wild Flower,
Hold Infinity in the palm of your hand
And Eternity in an hour.

[ MakeInfo ] Test progress completed!

wilson@wilson:~/OS_HW2_1.1.0$
```

- 執行 make clean

```
OS_HW2_1.1.0 : bash — Konsole
wilson@wilson:~/OS_HW2_1.1.0$ make clean

CREATED BY
S1103309

[ MakeInfo ] Clean progress started...

rm -v ./*.tmp ./*.out
removed './happy.tmp'
removed './happy2.tmp'
removed './master.out'
removed './mmv.out'

[ MakeInfo ] Clean progress completed!

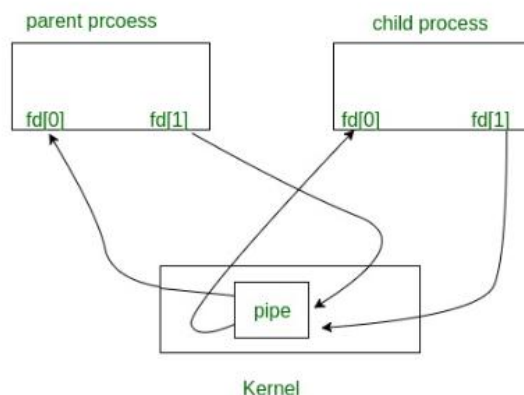
wilson@wilson:~/OS_HW2_1.1.0$
```

### 三、實做方法解說

除了上次作業就有的system calls（fork、open、read write、close等），這次用到了pipe這個功能。此處引用國外知名網站 [geeksforgeeks](https://www.geeksforgeeks.org/pipe-in-c/) 上的圖像來協助解釋pipe的原理：

Parent and child sharing a pipe

When we use `fork` in any process, file descriptors remain open across child process and also parent process. If we call `fork` after creating a pipe, then the parent and child can communicate via the pipe.



我們平常使用 `open()` 開啟檔案時，概念上類似於將檔案開啟於Kernel中，並且利用其回傳的 File Descriptor（一個int）來取用檔案（像上圖中想像的樣子，而Kernel裡的process資訊會被掛載於 `/proc`，令這些內容變得可視化（參見 [man7.org](http://man7.org)）。例如：我們用「`ls -l /proc/$$/fd`（註1）」這個指令可以看到目前這個bash process底下開啟的檔案有哪些）。

而當我們建立pipe的時候，pipe也會被想像成在kernel中開啟的虛擬檔案。pipe是一個匿名的FIFO容器，有read端點與write端點，所以我們用兩個file descriptors去連接這兩個端點（圖中fd[0]連接read端、fd[1]連接write端）。如此一來，我們就可以這些file descriptors存取這個pipe，進行讀取與寫入。當我們在fork()前建立pipe，父程序和子程序都是可以用fd去存取pipe的。

這邊我們還要知道的是，當我們執行c program時，0、1、2這幾個 file descriptor 會被保留給 stdin buffer、stdout 和 stderr 的實體。而我們可以想辦法把這幾個 file descriptor 連接到 pipe 的端點（註2），就可以把 pipe 當作 std I/O 的目標，以達成更廣泛的pipe共享。

我在這次程式中，就是用這種方式來實現目的。首先，我在master父程序中將讀出的資料放入pipe中，並且把子程序 stdin 的 file descriptor 指向 pipe 的 read 端點（把 pipe 直接當成 std input buffer），這樣我們在process image替換成mmv以後，就可以直接用 stdin 的 getline 函式將 pipe 的內容抽取出來。

註1：proc是process的縮寫，底下有許多以PID為名的目錄。每個PID目錄中的fd目錄底下有很多link連結檔，每一個連結檔都指向某個實體，用「ls -l」會顯示指向的資訊。而「\$\$」在bash中是一個變數，代表目前bash的process ID。知道這些資訊就可以了解這個指令的意義。

註2：stdin 的 file descriptor 用來讀取 input buffer，連接時要連接到 pipe 的讀取端。相反的，stdout 和 stderr 是將資訊寫入 output，連接時連到寫入端。用dup2()來複製某fd內容到另一個fd（參見 [man7.org](http://man7.org)）。



在編寫這次的程式過程中，我做了一些小實驗。在程式中加上這樣的程式碼：

```
string cmd = string(s: "ls -l /proc/") + to_string(val: getpid()) + "/fd";  
system( command: cmd.c_str());
```

這會讓 shell 去執行「ls -l /proc/[pid]/fd」這個指令，我們將這個動作在父程序、子程序、以及子程序 exec() 替換成 mmv 以後執行，得到以下結果：

```
total 0  
lrwx----- 1 wilson wilson 64 4月 8 11:00 0 -> /dev/pts/1  
lrwx----- 1 wilson wilson 64 4月 8 11:00 1 -> /dev/pts/1  
lrwx----- 1 wilson wilson 64 4月 8 11:00 2 -> /dev/pts/1  
lr-x----- 1 wilson wilson 64 4月 8 11:00 3 -> 'pipe:[854195]'  
l-wx----- 1 wilson wilson 64 4月 8 11:00 4 -> 'pipe:[854195]'
```

在父程序中的輸出

```
total 0  
lrwx----- 1 wilson wilson 64 4月 8 11:00 0 -> /dev/pts/1  
lrwx----- 1 wilson wilson 64 4月 8 11:00 1 -> /dev/pts/1  
lrwx----- 1 wilson wilson 64 4月 8 11:00 2 -> /dev/pts/1  
lr-x----- 1 wilson wilson 64 4月 8 11:00 3 -> 'pipe:[854195]'  
l-wx----- 1 wilson wilson 64 4月 8 11:00 4 -> 'pipe:[854195]'
```

在子程序中 (dup前)

```
total 0  
lr-x----- 1 wilson wilson 64 4月 8 11:00 0 -> 'pipe:[854195]'  
lrwx----- 1 wilson wilson 64 4月 8 11:00 1 -> /dev/pts/1  
lrwx----- 1 wilson wilson 64 4月 8 11:00 2 -> /dev/pts/1  
lr-x----- 1 wilson wilson 64 4月 8 11:00 3 -> 'pipe:[854195]'  
l-wx----- 1 wilson wilson 64 4月 8 11:00 4 -> 'pipe:[854195]'
```

在 mmv.out 中 (dup後)

可以看到父程序和子程序共用同一個pipe<sup>(註3)</sup> (fd 3 & fd 4 指向同一個 pipe，但是讀寫權限不同，就被視為FIFO的讀取與寫入端)，而進行 dup() 以後，原本指向 stdin 的 fd 也指向了 pipe 的讀取讀取端點了。所以在mmv中使用 std input 相關的函式時，就會從 pipe 讀取資料

註3：前提是要在 fork() 進行前就先建立 pipe，父子程序才能共享 pipe。

其實知道以上這些特性以後，換個方位思考，我其實想到這題的另外一個解法，可以不用打破 stdin 原本的規則。我們可以自己定義一個特別的數字，並用 `dup2()` 產生一個新的 fd，然後將指向 pipe 讀取端的 fd 複製過來。這樣我們就可以用這個專用的數字在 mmv 中輕鬆存取已經存在的這個 pipe 了。（會需要這麼做是因為 `exec()` 後不會繼承原本的變數內容，我們無法知道在 `exec()` 之前指向 pipe 的 fd 的 int 是多少，所以直接指定一個固定的數字是最方便存取的方法）。請見以下範例：

- 在 master.cpp 中

```
dup2( fd: pfd[0], fd2: 3309);
```

- 在 mmv.cpp 中

```
int rd = read( fd: 3309, buf, nbytes: sizeof(buf));
```

在範例中，我們指定把 3309 這個數字指向 pipe 讀取端，就可以在 mmv 中用 3309 這個 fd 直接進行讀取。但是這樣可能有風險，因為依照 `dup()` 的規則（參見 [man7.org](http://man7.org) 或下圖），當我們指定一個數字為複製對象時，如果該數字已經被使用，`dup()` 會直接將原本指向的 file 關閉，這樣有可能會造成錯誤（雖然在這次作業中發生的機率不高）。

If the file descriptor `newfd` was previously open, it is closed before being reused; the close is performed silently (i.e., any errors during the close are not reported by `dup2()`).

所以另外我想到一個更不會出錯的方法：直接將 `exec()` 前 pipe 讀取端 fd 的編號以 argv 的方式傳入 mmv 中，在 mmv 中就直接用這個編號存取 pipe。經過嘗試，這個方法也是可以的，但是相對來說麻煩了點（因為要將 fd 在 int 與 char[] 之間轉換）。

以上就是學生對於執行原理的一些理解與解釋，收穫許多。後面兩個方法是在編寫與爬文過程中意外想到，但是與作業要求不符，所以僅供補充參考。

## 四、其他解說

- 程式碼中有更多詳細的步驟註解，就不附上冗長的程式截圖。以上講解若有任何不足之處，請見master.cpp與mmv.cpp兩個檔案的註解。
- Makefile中也有許多細節，但是內容較為單純，所以沒有附上截圖和多加解釋。然而如果需要進一步了解 make 執行過程，請見隨附的Makefile
- 在cpp檔及Makefile中可以看到許多[Shell Color Codes](#)，讓 log 的輸出可以有種顏色，更為一目了然狀態。
- Makefile中有許多以「@」開頭的指令，主要用來隱藏不必要顯示的指令 (例如：echo 出 log的指令)，讓畫面更乾淨。

## 五、心得感想

在這兩次作業的過程中，有許多同學來問我一些相關的問題，我發現很多人都依查到的資料直接照寫，不理解執行的意義和過程。但我認為學習OS的有趣之處就是好好的理解這些動作在底層是怎麼實行的。雖然很難真的完全知曉所有運作過程的真面目，但是更進一步了解一些概念，會發現作業系統真的是一個很神奇的一個東西。

一直是Linux忠實粉絲和想要推廣Linux的我，在這樣的作業中真的找到很大的樂趣，也希望像今天這樣的講解可以讓更多人了解 Linux。