pipe & fifo

pipe的簡單用法

使用命令列（使用「|」）

```
1. shiwulo@vm:~/sp/ch11$ less fifo1.c | wc -l
2. 22
3. less fifo1.c | grep "#include"
4. #include <fcntl.h>
5. #include <sys/stat.h>
6. #include <sys/types.h>
7. #include <unistd.h>
8. #include <stdio.h>
9. #include <string.h>
```

列出目前使用者開啟的FIFO

```
shiwulo@vm:~$ lsof | grep FIFO |grep shiwulo | more
COMMAND     PID   TID             USER   FD      TYPE             DEVICE SIZE/OFF       NODE NAME
dropbox     741                 shiwulo   86r     FIFO               0,10      0t0     345673 pipe
dropbox     741                 shiwulo   87w     FIFO               0,10      0t0     345673 pipe
dropbox     741                 shiwulo   89r     FIFO               0,10      0t0     345085 pipe
dropbox     741                 shiwulo   90w     FIFO               0,10      0t0     345085 pipe
dropbox     741                 shiwulo  129r     FIFO               0,10      0t0     347670 pipe
dropbox     741                 shiwulo  130w     FIFO               0,10      0t0     347670 pipe
dropbox     741   748           shiwulo   86r     FIFO               0,10      0t0     345673 pipe
dropbox     741   748           shiwulo   87w     FIFO               0,10      0t0     345673 pipe
dropbox     741   748           shiwulo   89r     FIFO               0,10      0t0     345085 pipe
dropbox     741   748           shiwulo   90w     FIFO               0,10      0t0     345085 pipe
dropbox     741   748           shiwulo  129r     FIFO               0,10      0t0     347670 pipe
dropbox     741   748           shiwulo  130w     FIFO               0,10      0t0     347670 pipe
dropbox     741   751           shiwulo   86r     FIFO               0,10      0t0     345673 pipe
dropbox     741   751           shiwulo   87w     FIFO               0,10      0t0     345673 pipe
dropbox     741   751           shiwulo   89r     FIFO               0,10      0t0     345085 pipe
...
```

pipe()

- `#include <unistd.h>`
- `int pipe(int pipefd[2]);`

- 建立一個溝通的管道，`pipefd[0]`為讀取，`pipefd[1]`為寫入，如果發生錯誤，回傳值為`-1`，否則為`0`

使用pipe的簡單程式（pipe1.c）

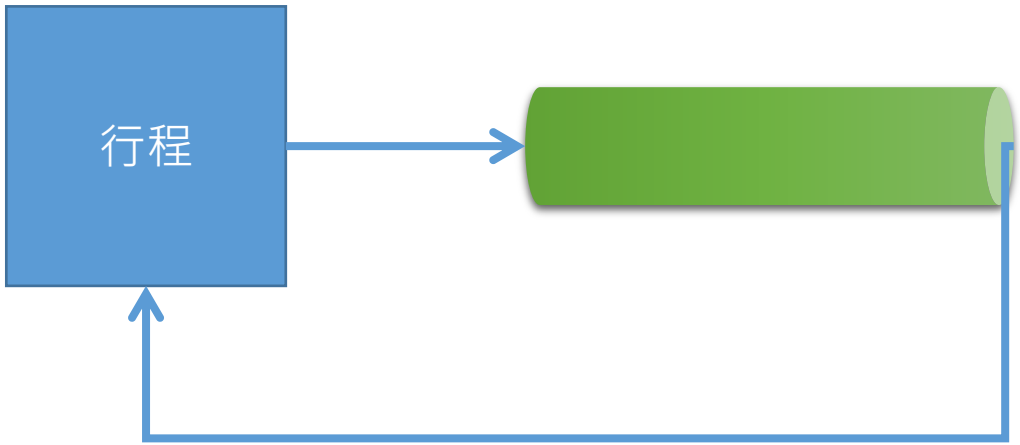
```
1.  #include <unistd.h>
2.  #include <stdio.h>

3.  int main(int argc, char** argv) {
4.      int pipefd[2];
5.      char *str = "hello\n\0";
6.      char buf[200];
7.      pipe(pipefd);
8.      write(pipefd[1], str, strlen(str)+1));
9.      read(pipefd[0], buf, 200);
10.     printf("%s", buf);
11.     return 0;
12. }
```

執行結果

```
shiwulo@vm:~/sp/ch11$ ./pipe1
hello
```

示意圖



程式說明

- 宣告pipefd[2]，代表一個型態為int的一維陣列，該陣列的大小為2
- pipefd[0]是讀取端
- pipefd[1]是寫入端
- 從寫端寫入的資料可以從讀取端讀取

- pipe會繼承給子行程（fork產生的子行程）
- 通常pipe是父行程與子行程，或子行程間的通訊管道

用pipe作為
父行程與子行程的通訊

常見的寫法一，直接溝通（pipe2.c）

```c
#include <assert.h>
#include <unistd.h>
#include <stdio.h>

int main(int argc, char **argv)
{
    int pipefd[2];
    char *str = "hello\n";
    char buf[200];

    int ret;
    pipe(pipefd);
    ret = fork();
    assert(ret>=0);
    if (ret==0) {
        close(pipefd[0]);
        write(pipefd[1], str, strlen(str)+1);
    } else {
        close(pipefd[1]);
        read(pipefd[0], buf, 200);
        printf(”childL: %s", buf);
    }
    return 0;
}
```

執行結果

```
shiwulo@vm:~/sp/ch11$ ./pipe2
child: hello
```

程式說明

- fork產生父行程與子行程
  - 子行程對pipe寫入
  - 父行程對pipe讀取
- 因此構成行程間（父與子行程）間的通訊

示意圖

常見的寫法二，改成標準輸出入（pipe3.c）

```
1.  #include <assert.h>
2.  #include <unistd.h>
3.  #include <stdio.h>

4.  int main(int argc, char **argv) {
5.      int pipefd[2];
6.      char buf[200];
7.
8.      int ret;
9.      pipe(pipefd);
10.     ret = fork();
11.     assert(ret>=0);
12.     if (ret==0) {   /*child*/
13.
```

常見的寫法二，改成標準輸出入（pipe3.c）

```
14.         close(1);
15.         dup(pipefd[1]);
16.         close(pipefd[0]);
17.         printf("hello");
18.     } else {
19.         close(0);
20.         dup(pipefd[0]);
21.         close(pipefd[1]);
22.         scanf("%s", buf);
23.         printf("parent: %s\n", buf);
24.     }
25.     return 0;
26. }
```

執行結果

```
shiwulo@vm:~/sp/ch11$ ./pipe3
parent: hello
```

示意圖

dup

- 複製一個file descriptor到最低的file descriptor
- 常見的用法是：
  - 關閉stdin或stdout，隨後馬上執行dup，如此可以讓stdin或stdout變成所指定的file descriptor
  - 換言之，對stdin或stdout所有的操作都變成對該file descriptor的操作

dup2

- `int dup2(int oldfd, int newfd);`
- 複製一個「oldfd」file descriptor到「newfd」的file descriptor

程式說明

- 先將子行程的stdout結束掉（close），再呼叫dup(fd[1])，讓stdout接到pipe的輸出端
- 先將父行程的stdin結束掉（close），再呼叫dup(fd[0])，讓stdin接到pipe的輸入端

- 從子行程呼叫printf，會將資料導向父行程，父行程會將該字串加上 “parent:”

system()-like版的FIFO（自行練習）

```
1. #include <stdio.h>
2. FILE *popen(const char *command, const char
   *type);
3. int pclose(FILE *stream);
```

- The popen() function opens a process by creating a pipe, forking, and invoking the shell. The type argument may specify only reading or writing, not both.
- The pclose() function waits for the associated process to terminate and returns the exit status of the command.

利用pipe作為
子行程間的通訊

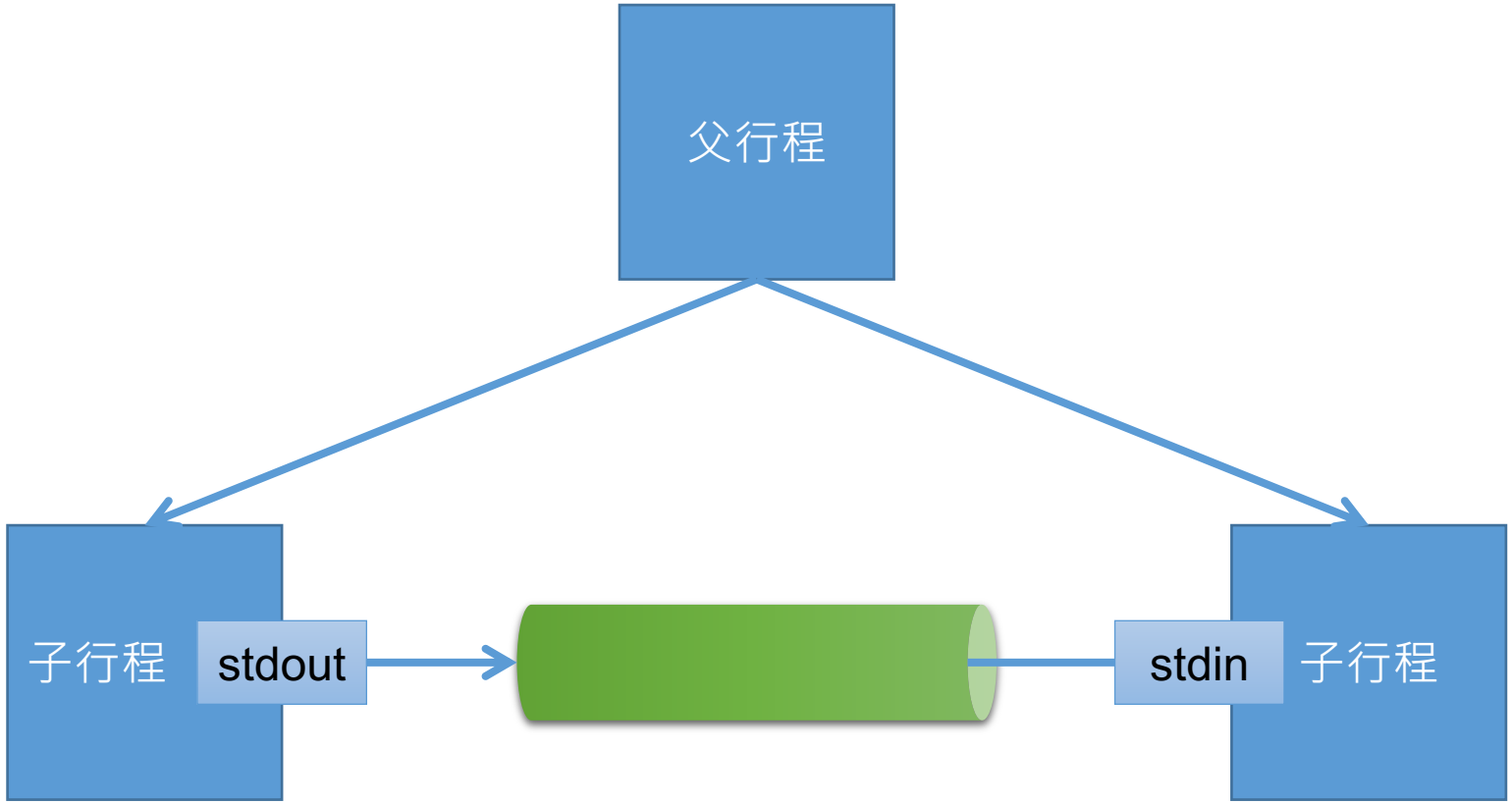
常見寫法三，子行程間通訊（pipe4.c）

```
1. #include <assert.h>
2. #include <unistd.h>
3. #include <stdio.h>
4. int main(int argc, char **argv) {
5.     int pipefd[2];
6.     char buf[200];
7.     FILE *in_stream;
8.     int ret;
9.     pipe(pipefd);
10.    ret = fork();
11.    if (ret==0) { /*child 1*/
12.        printf("I am child 1\n");
13.        close(1); dup(pipefd[1]);
14.        close(pipefd[1]); close(pipefd[0]);
15.        printf("send by pipe");
16.    }
17.    if (ret>0) {
18.        ret = fork();
19.        if (ret==0) { /*child 2*/
20.            close(0); dup(pipefd[0]);
21.            close(pipefd[1]);
   close(pipefd[0]);
22.            in_stream=fdopen(0, "r");
23.            /*用fgets取代gets*/
24.            fgets(buf, 200, in_stream);
25.            printf("child 2: %s\n", buf);
26.}   }   }
```

常見寫法三-2，子行程間通訊（pipe4-2.c）

```
1. #include <assert.h>
2. #include <unistd.h>
3. #include <stdio.h>
4. int main(int argc, char **argv) {
5.     int pipefd[2];
6.     char buf[200];
7.     FILE *in_stream;
8.     int ret;
9.     pipe(pipefd);
10.    ret = fork();
11.    if (ret==0) { /*child 1*/
12.        printf("I am child 1\n");
13.        close(1); dup(pipefd[1]);
14.        close(pipefd[1]); close(pipefd[0]);
15.        system(“ls”);
16.        printf("send by pipe");
17.    }
18.    if (ret>0) {
19.        ret = fork();
20.        if (ret==0) { /*child 2*/
21.            close(0); dup(pipefd[0]);
22.            close(pipefd[1]);
    close(pipefd[0]);
23.            system(“wc”);
24.            in_stream=fdopen(0, "r");
25.            /*用fgets取代gets*/
26.            fgets(buf, 200, in_stream);
27.            printf("child 2: %s\n", buf);
28.}   }   }
```

示意圖



程式說明

- 父行程先建立fd[2]
- 二個子行程分別關閉stdin和stdout
- 隨後立即使用dup將stdin和stdout轉向到pipe

FIFO (named pipe)

mkfifo()

Linux指令 mkfifo – make FIFOs (named pipes)

- `int mkfifo(const char *pathname, mode_t mode);`

- 第一個參數放入「路徑」及「檔名」
- 第二個參數可以指定讀寫的權限（mode & ~umask）
- 只要知道fifo位置的人，並且有適當的權限，就可以讀寫fifo

使用FIFO的簡單程式（fifo1.c）

```
1. #include <fcntl.h>
2. #include <sys/stat.h>
3. #include <sys/types.h>
4. #include <unistd.h>
5. #include <stdio.h>

6. char buf[200];

7. int main(int argc, char
   **argv) {
8.     int fd;
9.     mkfifo(”/tmp/shiwulo",
   0666);
10.    fd =
   open(”/tmp/shiwulo",O_RDWR);
11.    write(fd, "hello",
   sizeof("hello"));
12.    read(fd, buf, 200);
13.    printf("%s\n", buf);
14.    close(fd);
15.    unlink(”/tmp/shiwulo");
16.    return 0;
17.}
```

執行結果

```
shiwulo@vm:~/sp/ch11$ ./fifo1
hello
```

檔案系統出現pipe

```
shiwulo@vm:~$ ls /tmp/shiwulo -l
prw-rw-r-- 1 shiwulo shiwulo 0  五  17 18:20 /tmp/shiwulo
```

程式說明

- 使用mkfifo建立FIFO檔案，權限是666（所有人都可以讀寫）
- 使用open打開這個檔案，並且指定可以讀寫
- 用read和write對這個FIFO讀寫

- FIFO通常用於「想進行通訊的二個行程」但這二個行程「沒有共同的parent」

使用FIFO進行行程間通訊（fifo2-r）

```
1.  #include <fcntl.h>
2.  #include <sys/stat.h>
3.  #include <sys/types.h>
4.  #include <unistd.h>
5.  #include <stdio.h>
6.  char buf[200];
7.  int main(int argc, char **argv) {
8.      int fd; int ret;
9.      ret = mkfifo("/tmp/shiwulo", 0666);
10.     printf("mkfifo() = %d\n", ret);
11.     close(0);
```

使用FIFO進行行程間通訊（fifo2-r）

```
12.     fd = open("/tmp/shiwulo",O_RDONLY);
13.     scanf("%s", buf);
14.     printf("%s\n", buf);
15.     scanf("%s", buf);
16.     printf("%s\n", buf);
17.     scanf("%s", buf);
18.     printf("%s\n", buf);
19.     close(fd);
20.     unlink("/tmp/shiwulo");
21.     return 0;
22. }
```

程式說明

- 請先執行reader的程式，reader會停留在`scanf()`這一行

使用FIFO進行行程間通訊（ fifo2-w ）

```
1.  #include <fcntl.h>
2.  #include <sys/stat.h>
3.  #include <sys/types.h>
4.  #include <unistd.h>
5.  #include <stdio.h>
6.  #include <string.h>
7.  char buf[200];
8.  int main(int argc, char **argv) {
9.          int fd;
10.         int ret;
```

使用FIFO進行行程間通訊（ fifo2-w ）

```
11.         ret=mkfifo("/tmp/shiwulo", 0666);
12.         printf("mkfifo() = %d\n", ret);
13.         close(1);
14.         fd = open("/tmp/shiwulo",O_WRONLY);
15.         printf("hello\n");
16.         printf("1234\n");
17.         printf("5678\n");
18.         close(fd);
19.         unlink("/tmp/shiwulo");
20.         return 0;
21. }
```

程式說明

- writer先建立FIFO的通訊管道
- 隨後open這個FIFO所代表的檔案
- 用write寫出“hello”後結束執行（reader此時會收到writer的訊息）

作業

- 使用fork、pipe、execv等函數，創造出二個child，第一個child執行「cat filename」，並透過PIPE的方式導入到第二個child。第二個child執行「wc」。並讓第二個child的結果輸出到螢幕上
  - 執行檔名稱為wordcount，該執行檔接受一個參數「filename」
  - wordcount執行結束後，需要在螢幕上印出「filename」的「行數、字數、大小」