pipe & fifo

中正大學，作業系統實驗室
羅習五 陽春副教授
shiwulo@gmail.com

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）

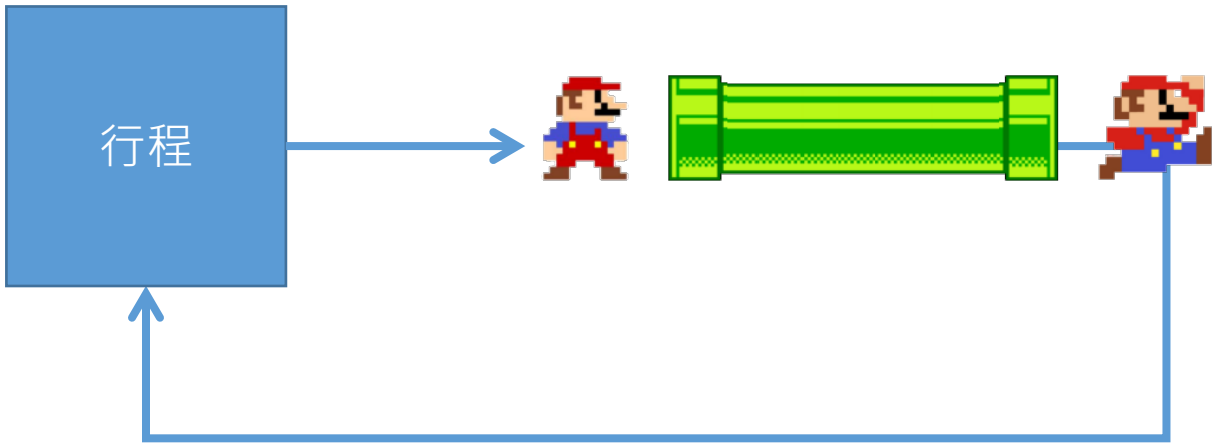
```
1.  #include <unistd.h>
2.  #include <stdio.h>

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）（打開的檔案基本上都會讓子行程繼承）
- 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）

```
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 *str = "hello\n";
7.      char buf[200];
8.
9.      int ret;
10.     pipe(pipefd);
11.     ret = fork();
12.     assert(ret>=0);
13.     if (ret==0) {
14.         close(pipefd[0]);
15.         write(pipefd[1], str,
16.           strlen(str)+1);
17.     } else {
18.         close(pipefd[1]);
19.         read(pipefd[0], buf, 200);
20.         printf(”childL: %s", buf);
21.     }
22.     return 0;
23. }
```

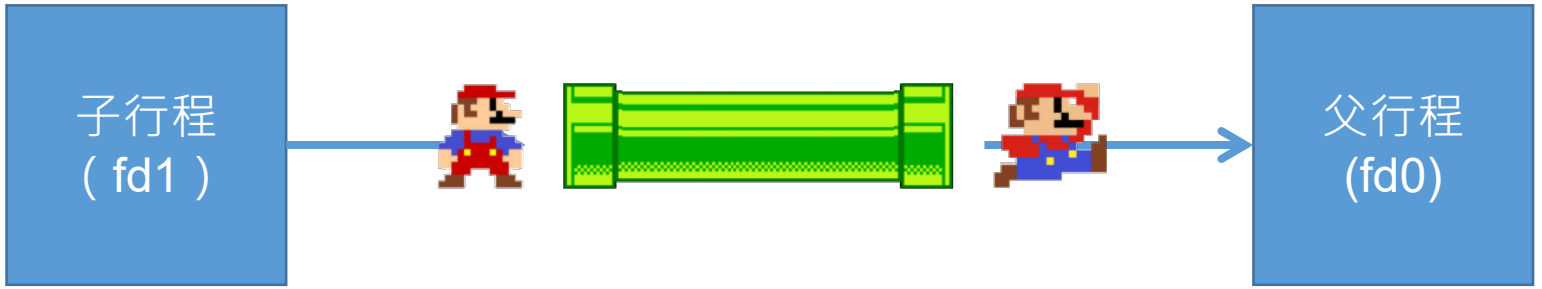
執行結果

```
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.
```

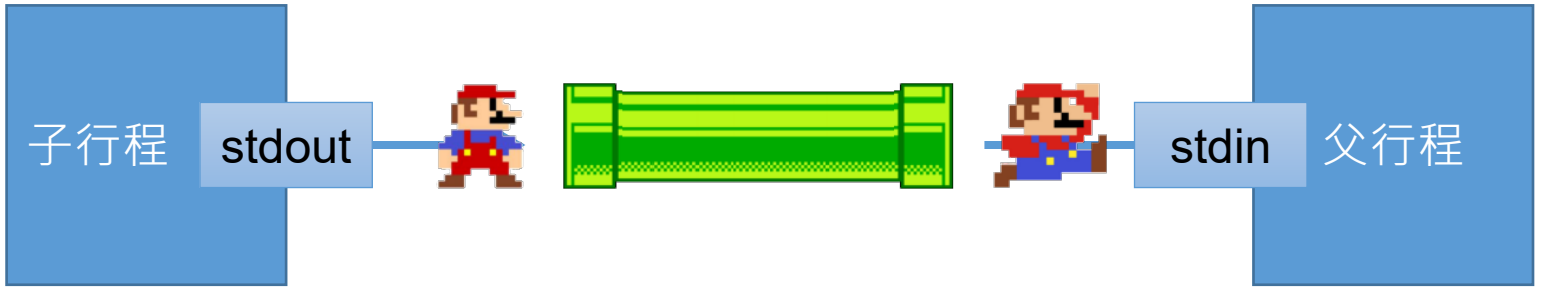
常見的寫法二，改成標準輸出入（pipe3.c）

```
14.         close(1);        /*關閉stdout*/
15.         dup(pipefd[1]);    /*將pipefd複製到stdout*/
16.         close(pipefd[1]);
17.         close(pipefd[0]);
18.         printf("hello");   /*印出 “hello” 到stdout*/
19.       } else {
20.         close(0);        /*關閉stdin*/
21.         dup(pipefd[0]);    /*將pipefd複製到stdin*/
22.         close(pipefd[0]);
23.         close(pipefd[1]);
24.         scanf("%s", buf);  /*從stdin讀入資料*/
25.         printf("parent: %s\n", buf);
26.       }
27.       return 0;
28.   }
```

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);

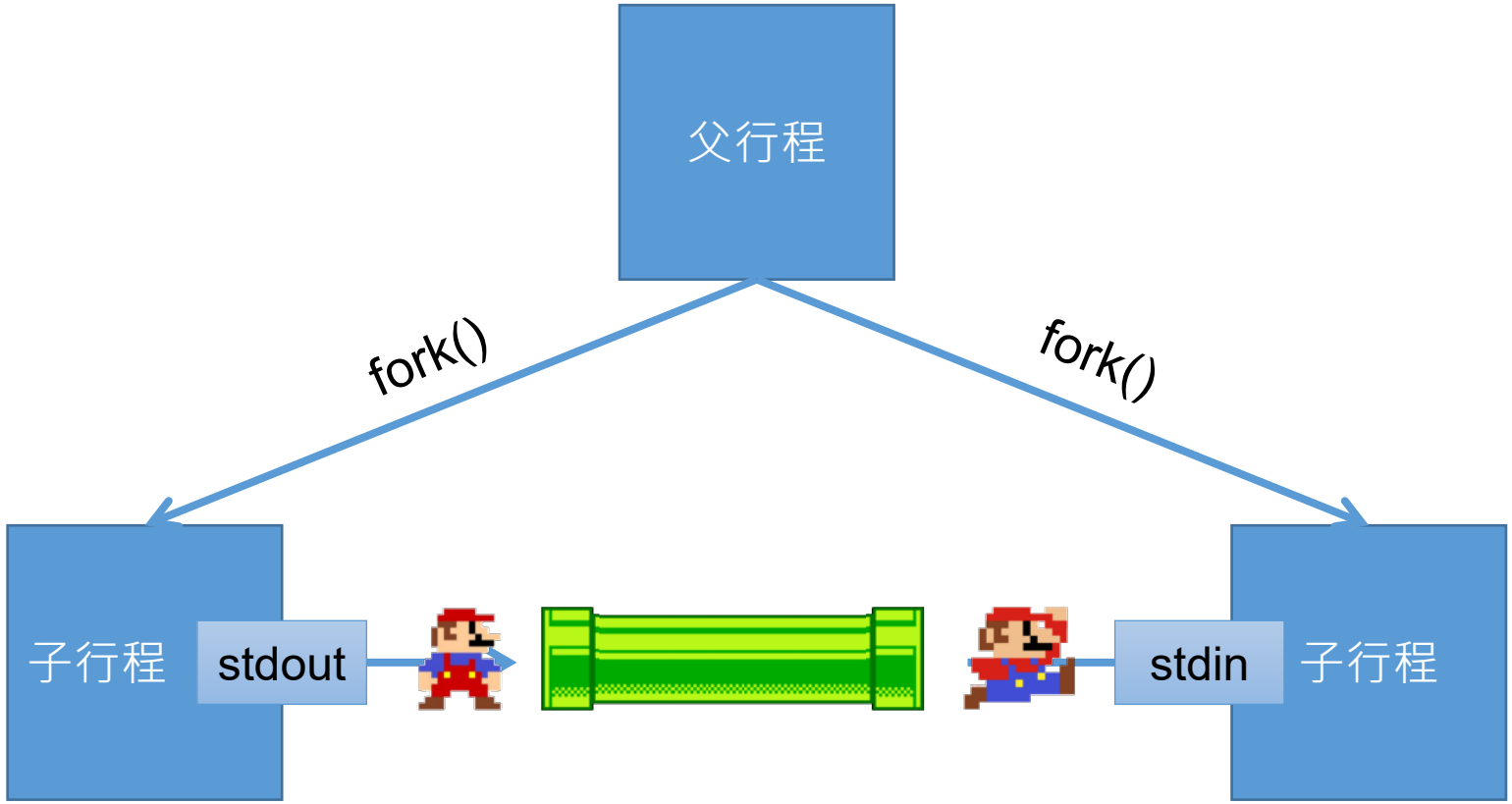
- The popen() function opens a process by creating a pipe, forking, and invoking the shell. The type argument may specify only reading or writing, not both.
- The pclose() function waits for the associated process to terminate and returns the exit status of the command.

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]);
22.             close(pipefd[0]);
23.             in_stream=fdopen(0, "r");
24.             /*用fgets取代gets*/
25.             fgets(buf, 200, in_stream);
26.             printf("child 2: %s\n", buf);
27. }   }   }
```

示意圖

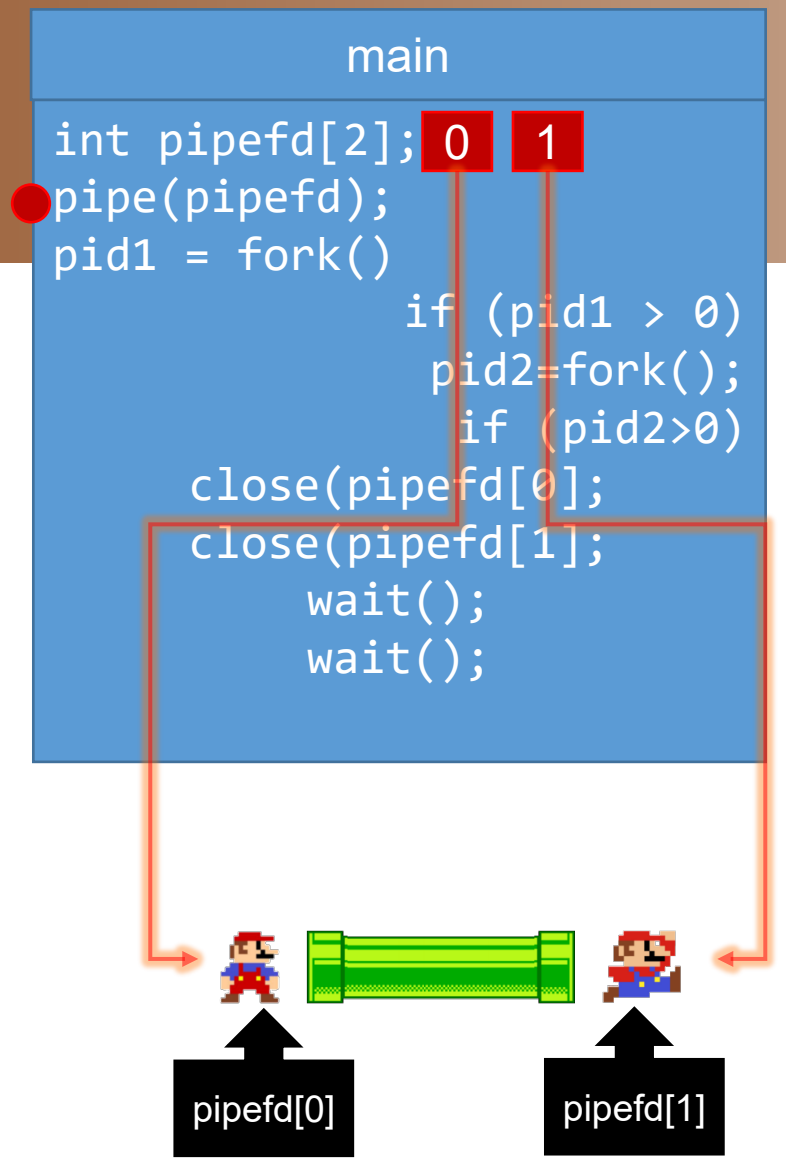


結果

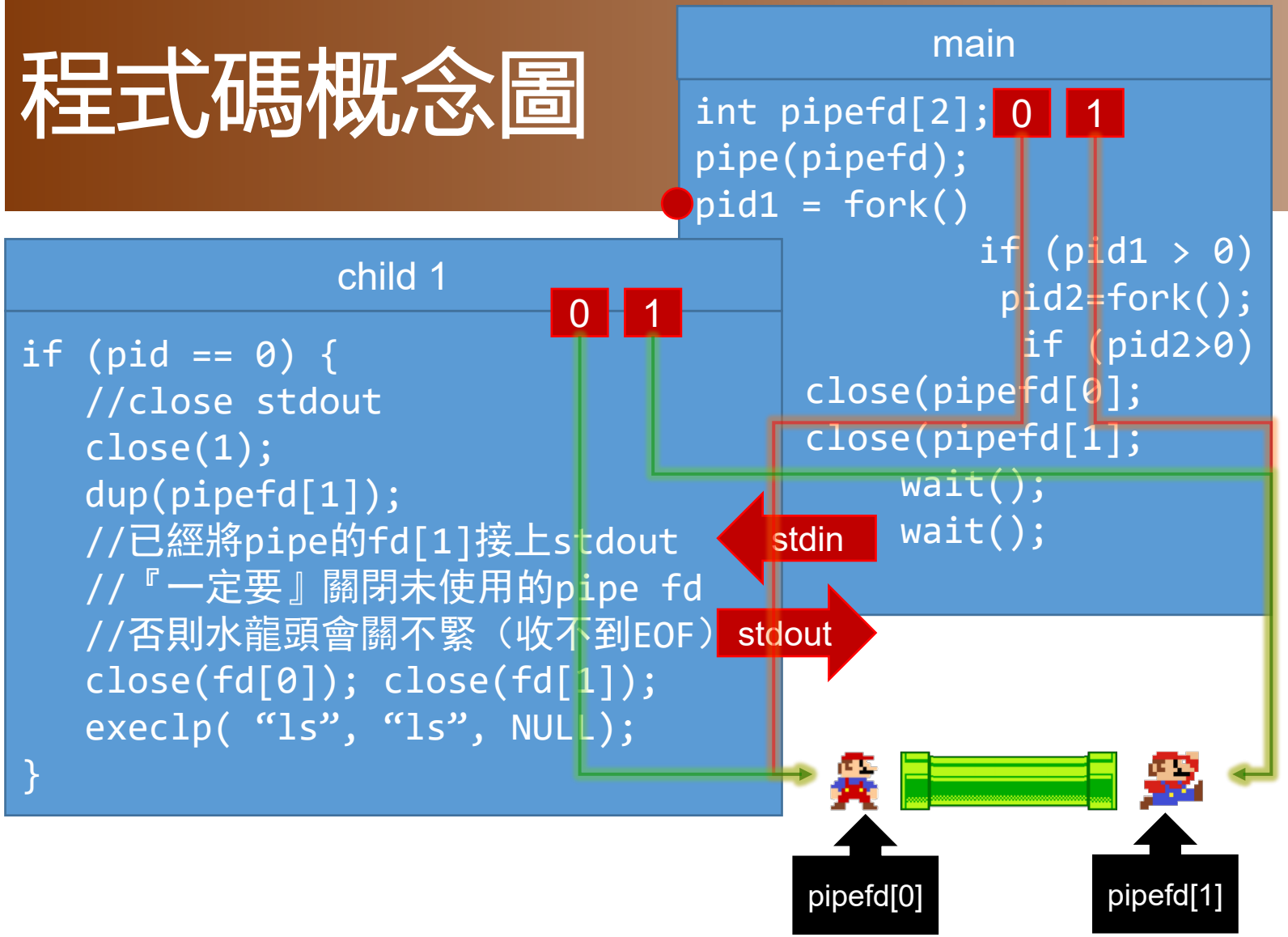
```
$ ./pipe4
I am child 1
child 2: send by pipe
```

串接二個child

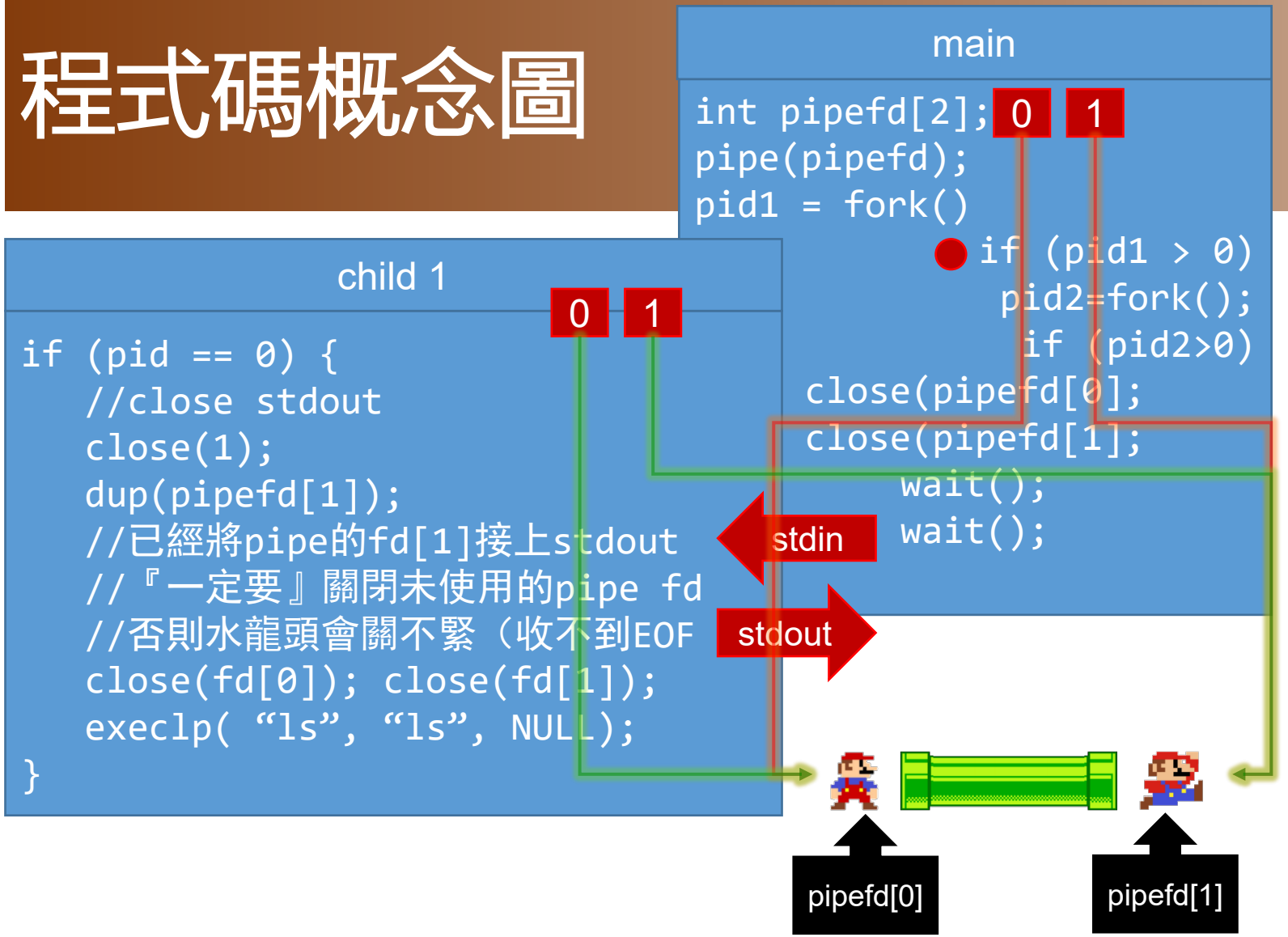
程式碼概念圖



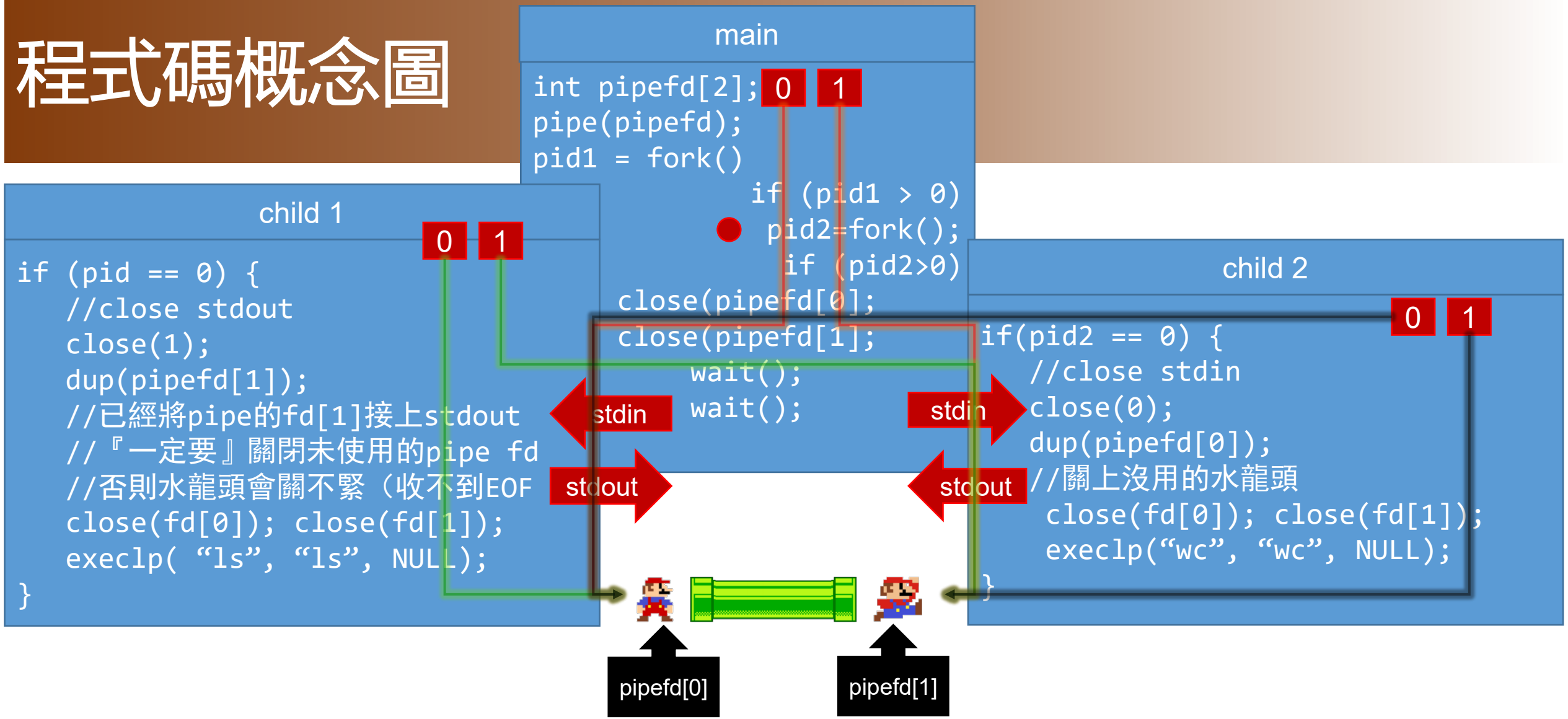
程式碼概念圖



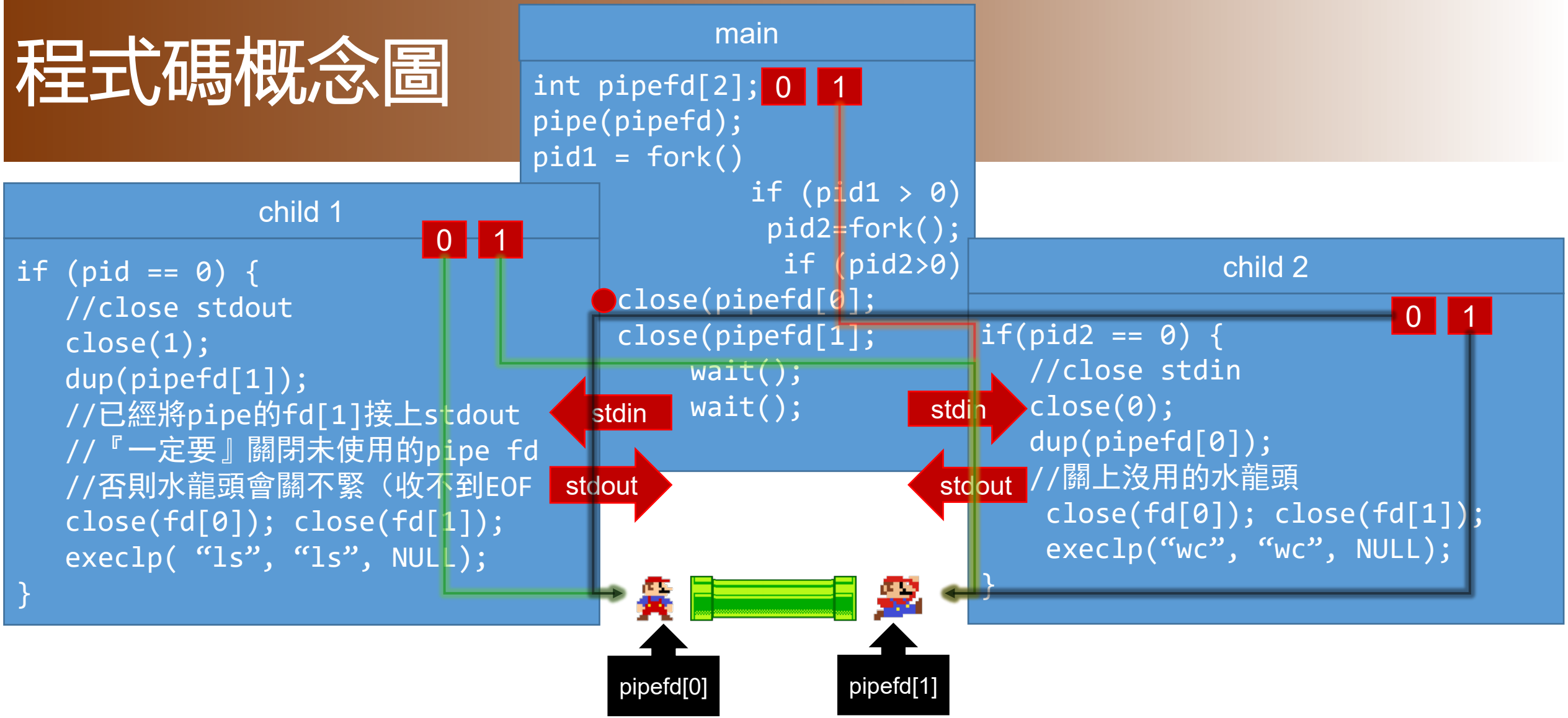
程式碼概念圖



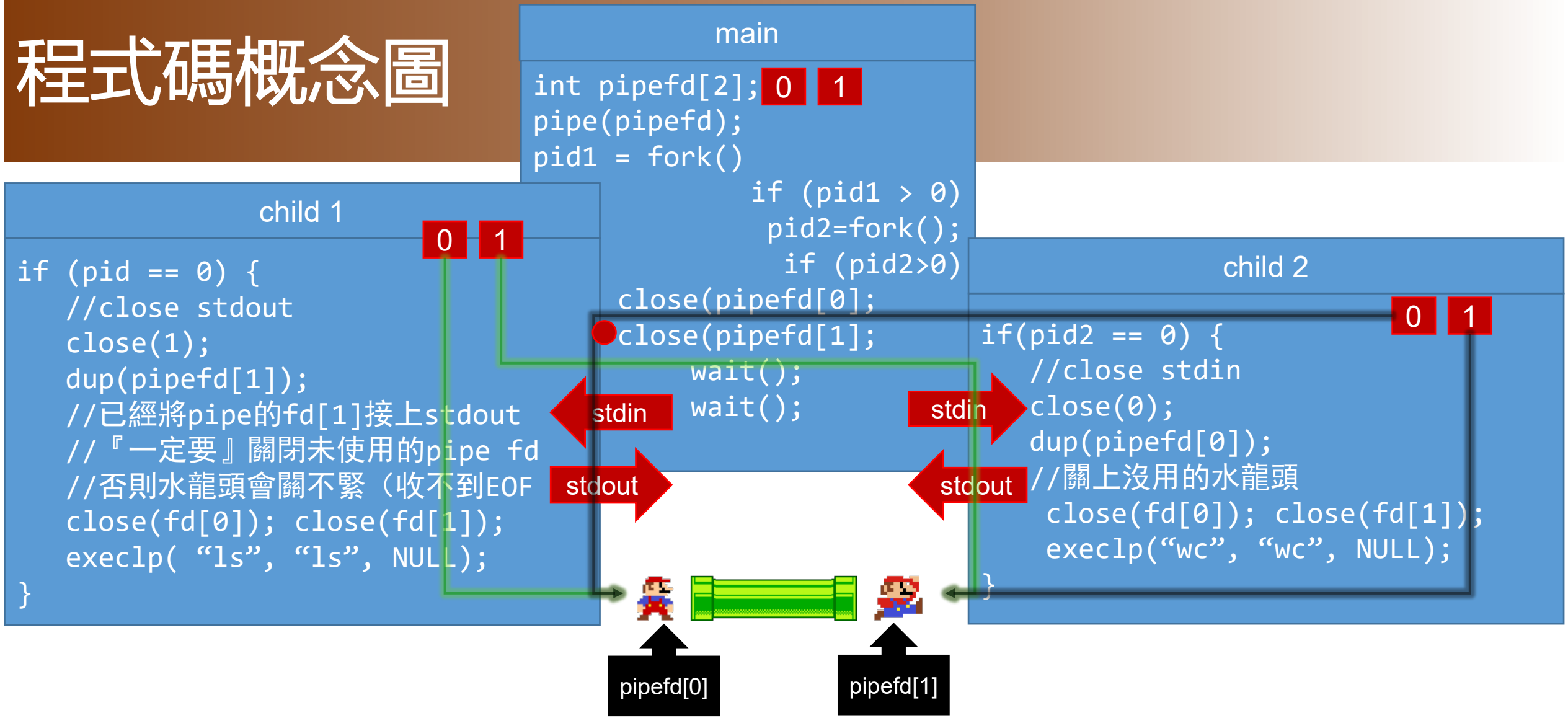
程式碼概念圖



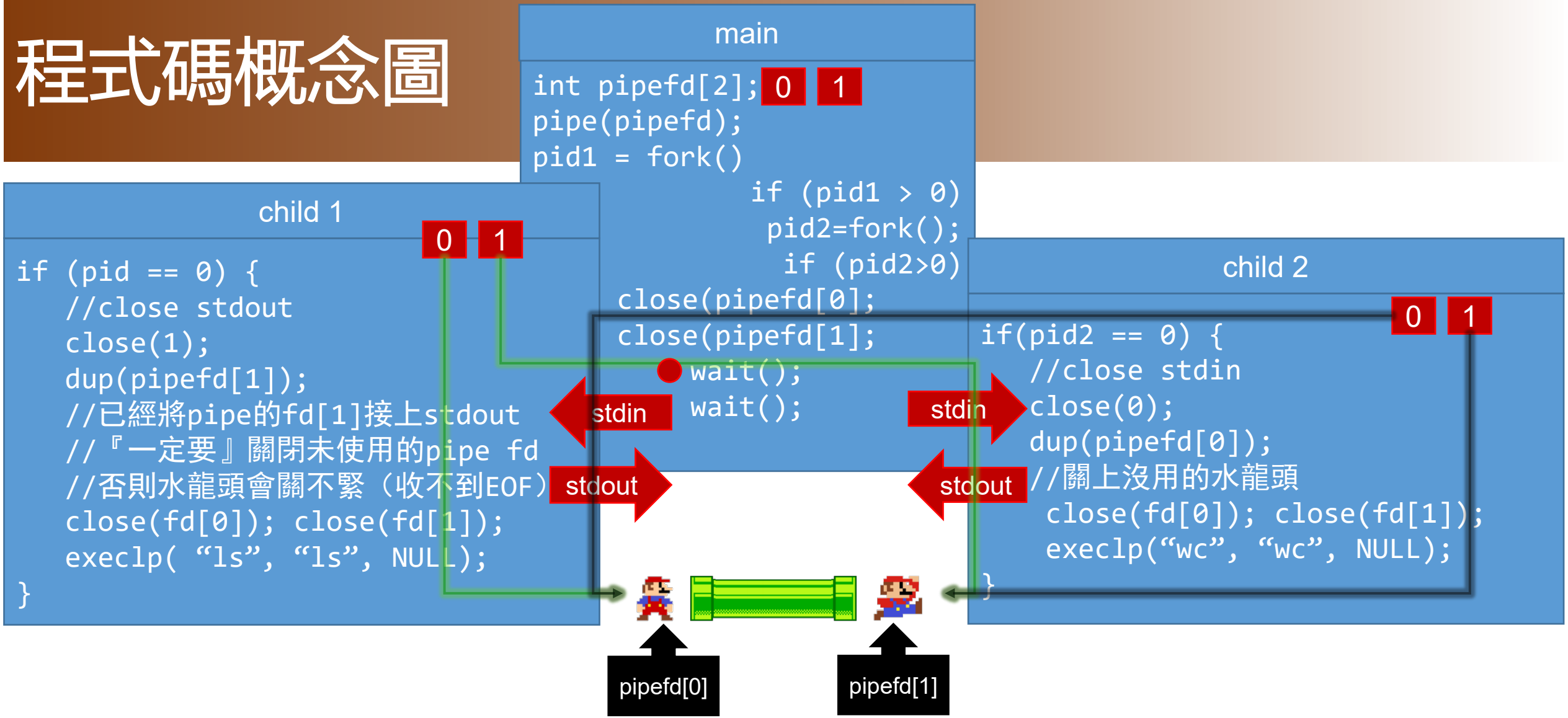
程式碼概念圖



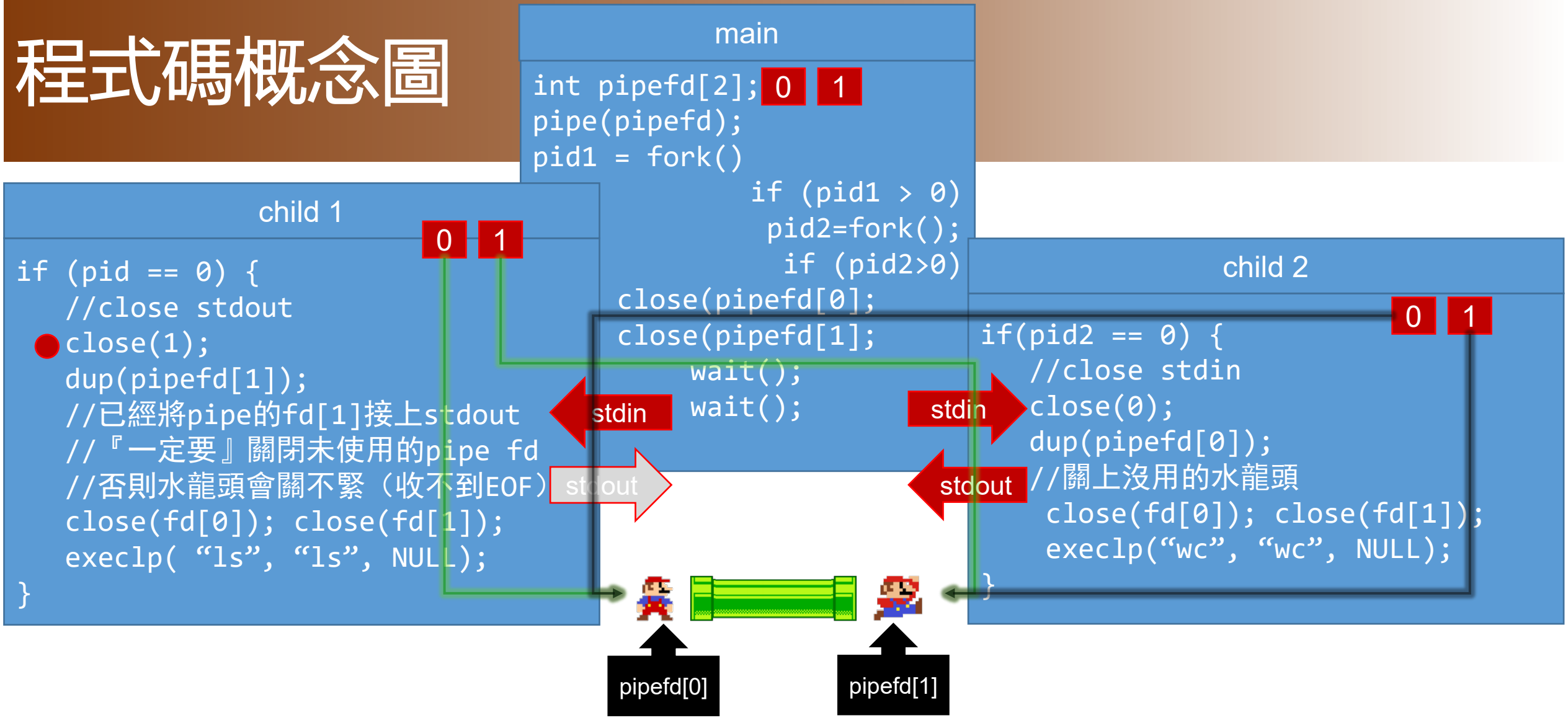
程式碼概念圖



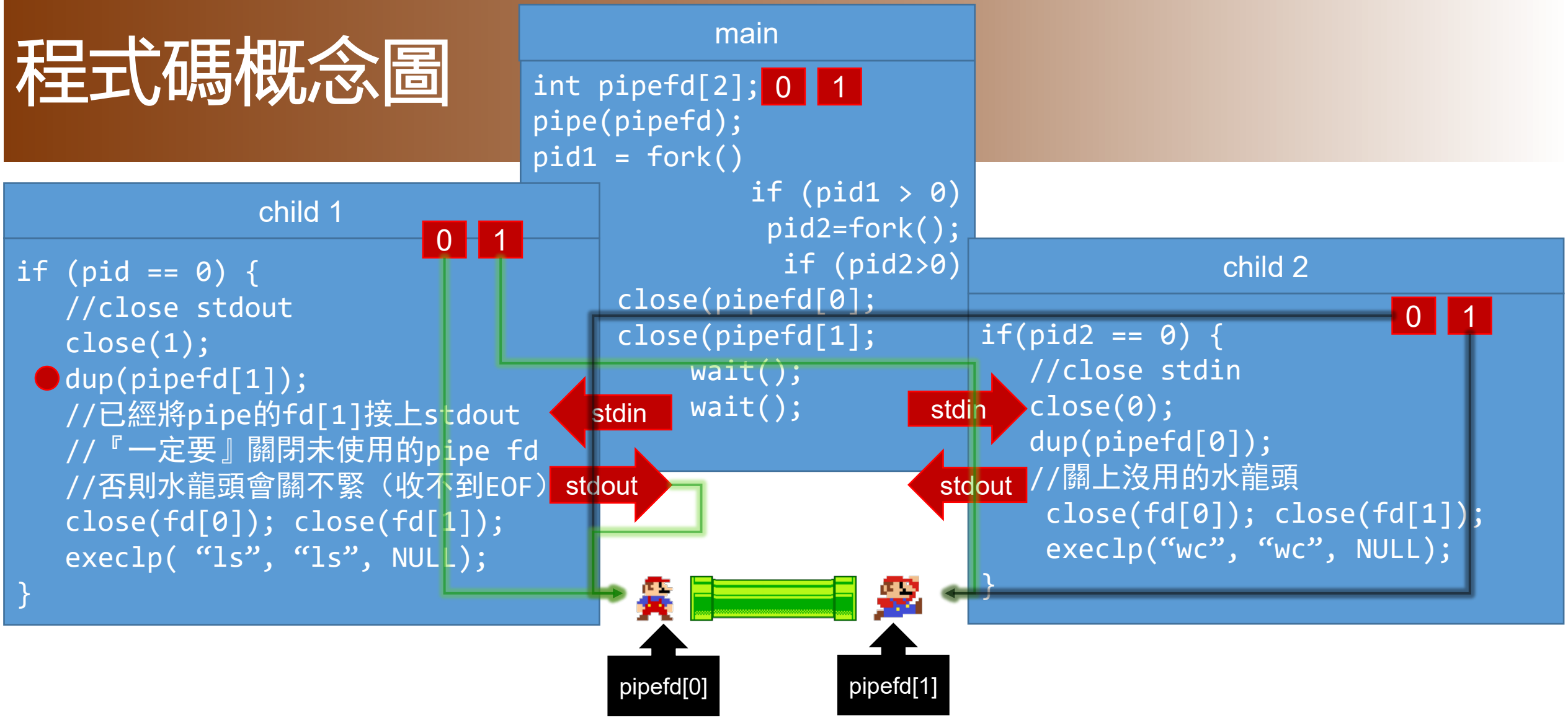
程式碼概念圖



程式碼概念圖



程式碼概念圖



程式碼概念圖

main

```
int pipefd[2];
pipe(pipefd);
pid1 = fork()
        if (pid1 > 0)
          pid2=fork();
          if (pid2>0)
close(pipefd[0];
close(pipefd[1];
      wait();
      wait();
```

child 1

```
if (pid == 0) {
   //close stdout
   close(1);
   dup(pipefd[1]);
   //已經將pipe的fd[1]接上stdout
   //『一定要』關閉未使用的pipe fd
   //否則水龍頭會關不緊（收不到EOF）
   close(fd[0]); close(fd[1]);
   execlp( “ls”, “ls”, NULL);
}
```

child 2

```
if(pid2 == 0) {
   //close stdin
   close(0);
   dup(pipefd[0]);
   //關上沒用的水龍頭
   close(fd[0]); close(fd[1]);
   execlp(“wc”, “wc”, NULL);
}
```

0 1

0 1

0 1

stdin

stdout

stdin

stdout

pipefd[0]

pipefd[1]

程式碼概念圖

main

```
int pipefd[2];
pipe(pipefd);
pid1 = fork()
          if (pid1 > 0)
           pid2=fork();
            if (pid2>0)
close(pipefd[0];
close(pipefd[1];
     wait();
     wait();
```

child 1

0 1

```
if (pid == 0) {
   //close stdout
   close(1);
   dup(pipefd[1]);
   //已經將pipe的fd[1]接上stdout
   //『一定要』關閉未使用的pipe fd
   //否則水龍頭會關不緊（收不到EOF）
   close(fd[0]); close(fd[1]);
   execlp( “ls”, “ls”, NULL);
}
```

child 2

0 1

```
if(pid2 == 0) {
   //close stdin
   close(0);
   dup(pipefd[0]);
   //關上沒用的水龍頭
    close(fd[0]); close(fd[1]);
    execlp(“wc”, “wc”, NULL);
}
```

stdin

stdout

stdin

stdout

pipefd[0]

pipefd[1]

程式碼概念圖

main

```
int pipefd[2];
pipe(pipefd);
pid1 = fork()
        if (pid1 > 0)
         pid2=fork();
          if (pid2>0)
close(pipefd[0];
close(pipefd[1];
     wait();
     wait();
```

child 1

0 1

```
if (pid == 0) {
   //close stdout
   close(1);
   dup(pipefd[1]);
   //已經將pipe的fd[1]接上stdout
   //『一定要』關閉未使用的pipe fd
   //否則水龍頭會關不緊（收不到EOF）
   close(fd[0]); close(fd[1]);
   execlp( “ls”, “ls”, NULL);
}
```

child 2

0 1

```
if(pid2 == 0) {
   //close stdin
   close(0);
   dup(pipefd[0]);
   //關上沒用的水龍頭
    close(fd[0]); close(fd[1]);
    execlp(“wc”, “wc”, NULL);
}
```

stdin

stdout

pipefd[0]

pipefd[1]

stdin

stdout

ls

程式碼概念圖

main

```
int pipefd[2];
pipe(pipefd);
pid1 = fork()
        if (pid1 > 0)
          pid2=fork();
           if (pid2>0)
close(pipefd[0];
close(pipefd[1];
     wait();
     wait();
```

child 1

0 1

```
if (pid == 0) {
    //close stdout
    close(1);
    dup(pipefd[1]);
    //已經將pipe的fd[1]接上stdout
    //『一定要』關閉未使用的pipe fd
    //否則水龍頭會關不緊（收不到EOF）
    close(fd[0]); close(fd[1]);
    execlp( “ls”, “ls”, NULL);
}
```

child 2

0 1

```
if(pid2 == 0) {
    //close stdin
    close(0);
    dup(pipefd[0]);
    //關上沒用的水龍頭
     close(fd[0]); close(fd[1]);
     execlp(“wc”, “wc”, NULL);
}
```

stdin

stdout

stdin

pipefd[0]

pipefd[1]

stdout

ls

程式碼概念圖

main

```
int pipefd[2];
pipe(pipefd);
pid1 = fork()
          if (pid1 > 0)
           pid2=fork();
            if (pid2>0)
close(pipefd[0];
close(pipefd[1];
     wait();
     wait();
```

child 1

0 1

```
if (pid == 0) {
    //close stdout
    close(1);
    dup(pipefd[1]);
    //已經將pipe的fd[1]接上stdout
    //『一定要』關閉未使用的pipe fd
    //否則水龍頭會關不緊（收不到EOF）
    close(fd[0]); close(fd[1]);
    execlp( “ls”, “ls”, NULL);
}
```

child 2

0 1

```
if(pid2 == 0) {
    //close stdin
    close(0);
    dup(pipefd[0]);
    //關上沒用的水龍頭
     close(fd[0]); close(fd[1]);
     execlp(“wc”, “wc”, NULL);
}
```

stdin

stdout

stdin

pipefd[0]

pipefd[1]

stdout

ls

程式碼概念圖

main

```
int pipefd[2];
pipe(pipefd);
pid1 = fork()
        if (pid1 > 0)
          pid2=fork();
           if (pid2>0)
close(pipefd[0];
close(pipefd[1];
     wait();
     wait();
```

child 1

0 1

```
if (pid == 0) {
    //close stdout
    close(1);
    dup(pipefd[1]);
    //已經將pipe的fd[1]接上stdout
    //『一定要』關閉未使用的pipe fd
    //否則水龍頭會關不緊（收不到EOF）
    close(fd[0]); close(fd[1]);
    execlp( “ls”, “ls”, NULL);
}
```

child 2

0 1

```
if(pid2 == 0) {
    //close stdin
    close(0);
    dup(pipefd[0]);
    //關上沒用的水龍頭
    close(fd[0]); close(fd[1]);
    execlp(“wc”, “wc”, NULL);
}
```

stdin

stdout

stdin

stdout

pipefd[0]

pipefd[1]

ls

程式碼概念圖

main

```
int pipefd[2];
pipe(pipefd);
pid1 = fork()
          if (pid1 > 0)
           pid2=fork();
            if (pid2>0)
close(pipefd[0];
close(pipefd[1];
     wait();
     wait();
```

child 1

0 1

```
if (pid == 0) {
   //close stdout
   close(1);
   dup(pipefd[1]);
   //已經將pipe的fd[1]接上stdout
   //『一定要』關閉未使用的pipe fd
   //否則水龍頭會關不緊（收不到EOF）
   close(fd[0]); close(fd[1]);
   execlp( “ls”, “ls”, NULL);
}
```

child 2

0 1

```
if(pid2 == 0) {
   //close stdin
   close(0);
   dup(pipefd[0]);
   //關上沒用的水龍頭
   close(fd[0]); close(fd[1]);
   execlp(“wc”, “wc”, NULL);
}
```

stdin

stdout

stdin

pipefd[0]

pipefd[1]

stdout

ls

程式碼概念圖

main

```
int pipefd[2]; 0 1
pipe(pipefd);
pid1 = fork()
            if (pid1 > 0)
             pid2=fork();
              if (pid2>0)
close(pipefd[0];
close(pipefd[1];
     wait();
     wait();
```

child 1

0 1

```
if (pid == 0) {
   //close stdout
   close(1);
   dup(pipefd[1]);
   //已經將pipe的fd[1]接上stdout
   //『一定要』關閉未使用的pipe fd
   //否則水龍頭會關不緊（收不到EOF）
   close(fd[0]); close(fd[1]);
   execlp( “ls”, “ls”, NULL);
}
```

child 2

0 1

```
if(pid2 == 0) {
   //close stdin
   close(0);
   dup(pipefd[0]);
   //關上沒用的水龍頭
    close(fd[0]); close(fd[1]);
    execlp(“wc”, “wc”, NULL);
}
```

stdin

stdout

stdin

pipefd[0]

pipefd[1]

stdout

ls

程式碼概念圖

main

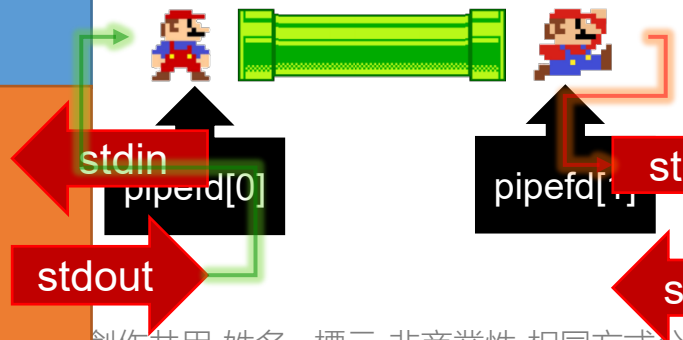
```
int pipefd[2]; 0 1
pipe(pipefd);
pid1 = fork()
            if (pid1 > 0)
             pid2=fork();
              if (pid2>0)
close(pipefd[0];
close(pipefd[1];
     wait();
     wait();
```

child 1

0 1

```
if (pid == 0) {
    //close stdout
    close(1);
    dup(pipefd[1]);
    //已經將pipe的fd[1]接上stdout
    //『一定要』關閉未使用的pipe fd
    //否則水龍頭會關不緊（收不到EOF）
    close(fd[0]); close(fd[1]);
    execlp( “ls”, “ls”, NULL);
}
```

ls



child 2

0 1

```
if(pid2 == 0) {
    //close stdin
    close(0);
    dup(pipefd[0]);
    //關上沒用的水龍頭
     close(fd[0]); close(fd[1]);
    execlp(“wc”, “wc”, NULL);
}
```

WC

程式碼概念圖

main

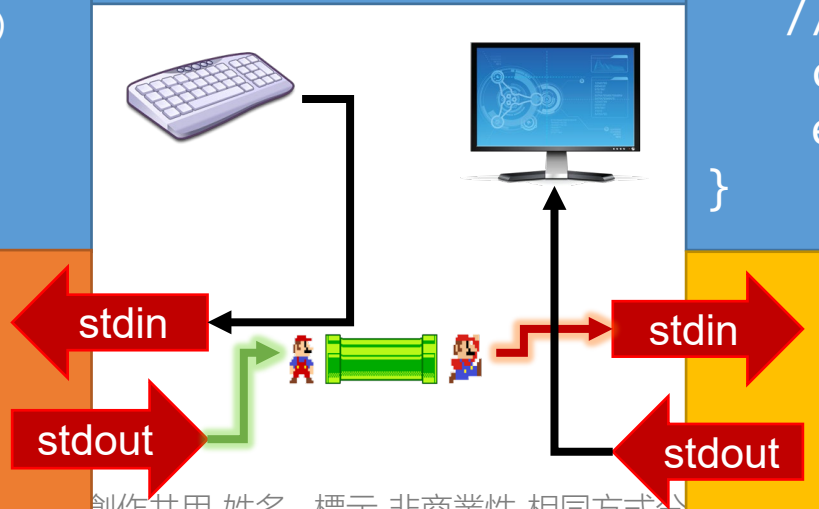
```
int pipefd[2]; 0 1
pipe(pipefd);
pid1 = fork()
            if (pid1 > 0)
             pid2=fork();
              if (pid2>0)
close(pipefd[0];
close(pipefd[1];
     wait();
     wait();
```

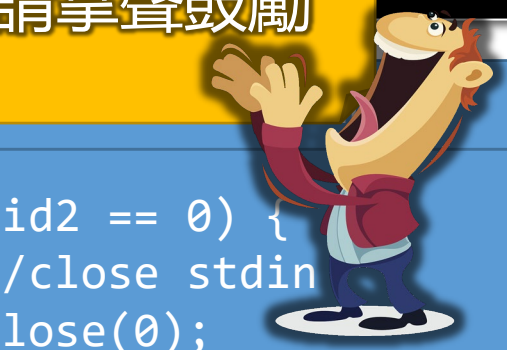
child 1

0 1

```
if (pid == 0) {
    //close stdout
    close(1);
    dup(pipefd[1]);
    //已經將pipe的fd[1]接上stdout
    //『一定要』關閉未使用的pipe fd
    //否則水龍頭會關不緊（收不到EOF）
    close(fd[0]); close(fd[1]);
    execlp( “ls”, “ls”, NULL);
}
```

ls





child 2

0 1

```
if(pid2 == 0) {
    //close stdin
    close(0);
    dup(pipefd[0]);
    //關上沒用的水龍頭
     close(fd[0]); close(fd[1]);
     execlp(“wc”, “wc”, NULL);
}
```

wc

程式碼概念圖

main

```
int pipefd[2]; 0 1
pipe(pipefd);
pid1 = fork()
            if (pid1 > 0)
             pid2=fork();
              if (pid2>0)
close(pipefd[0];
close(pipefd[1];
     wait();
     wait();
```

child 1

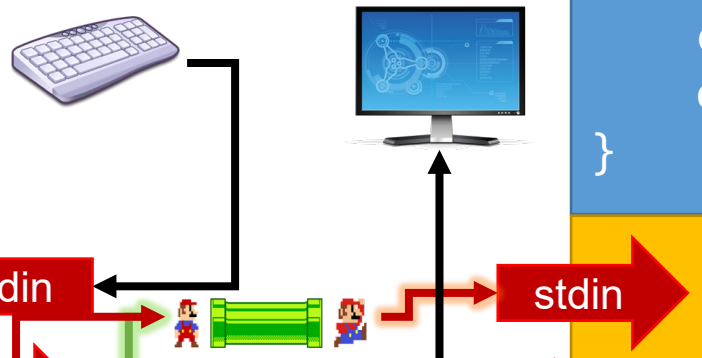
0 1

```
if (pid == 0) {
    //close stdout
    close(1);
    dup(pipefd[1]);
    //已經將pipe的fd[1]接上stdout
    //『一定要』關閉未使用的pipe fd
    //否則水龍頭會關不緊（收不到EOF）
    close(fd[0]); close(fd[1]);
    execlp( “ls”, “ls”, NULL);
}
```

child 2

0 1

```
if(pid2 == 0) {
    //close stdin
    close(0);
    dup(pipefd[0]);
    //關上沒用的水龍頭
     close(fd[0]); close(fd[1]);
     execlp(“wc”, “wc”, NULL);
}
```



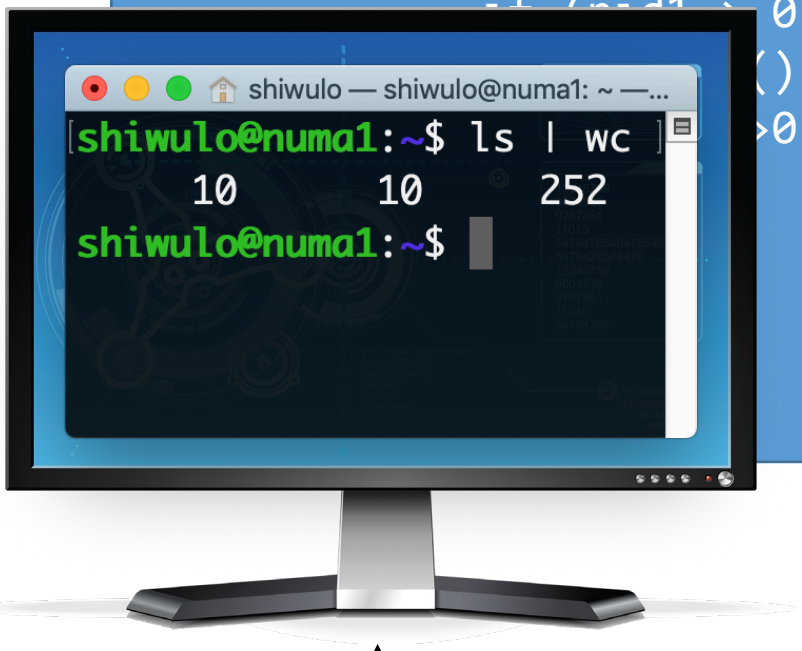
ls

WC

程式碼概念圖

main

```
int pipefd[2]; 0 1
pipe(pipefd);
pid1 = fork()
                    if (pid1 > 0)
                                  ();
                                  >0)
```

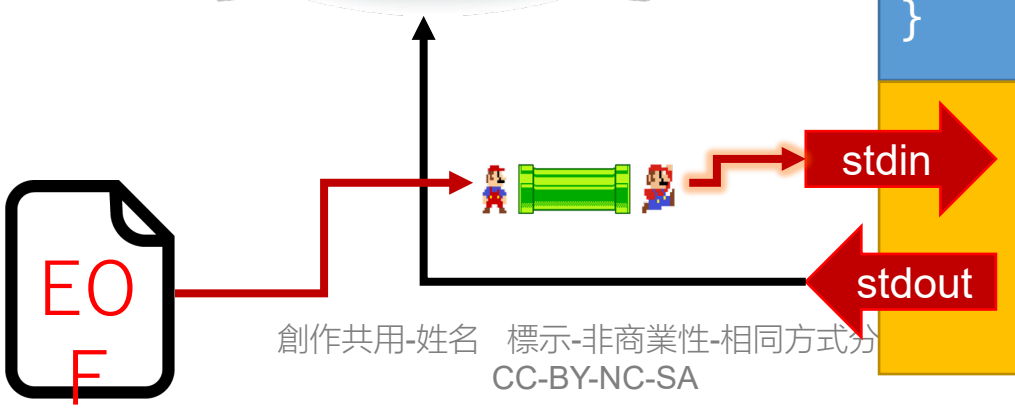


因為收『到EOF』
所以『wc』會印出結果，
然後結束執行

child 2

0 1

```
if(pid2 == 0) {
    //close stdin
    close(0);
    dup(pipefd[0]);
    //關上沒用的水龍頭
     close(fd[0]); close(fd[1]);
     execlp("wc", "wc", NULL);
}
```



程式碼概念圖

main

```
int pipefd[2]; 0 1
pipe(pipefd);
pid1 = fork()
                if (pid1 > 0)
                 pid2=fork();
                  if (pid2>0)
      close(pipefd[0];
      close(pipefd[1];
         wait();
           wait();
```

程式碼概念圖

main

```
int pipefd[2]; 0  1
pipe(pipefd);
pid1 = fork()
                if (pid1 > 0)
                 pid2=fork();
                  if (pid2>0)
      close(pipefd[0];
      close(pipefd[1];
           wait();
          wait();
```

程式碼概念圖

main

```
int pipefd[2];
pipe(pipefd);
pid1 = fork()
            if (pid1 > 0)
             pid2=fork();
              if (pid2>0)
close(pipefd[0];
close(pipefd[1];
     wait();
     wait();
```

child 1

```
if (pid == 0) {
   //close stdout
   close(1);
   dup(pipefd[1]);
   //已經將pipe的fd[1]接上stdout
   //『一定要』關閉未使用的pipe fd
   //否則水龍頭會關不緊（收不到EOF）
   close(fd[0]); close(fd[1]);
   execlp( “ls”, “ls”, NULL);
}
```

ls

child 2

```
if(pid2 == 0) {
   //close stdin
   close(0);
   dup(pipefd[0]);
   //關上沒用的水龍頭
    close(fd[0]); close(fd[1]);
    execlp(“wc”, “wc”, NULL);
}
```

wc

常見寫法三-2，子行程間通訊（pipe4-2.c）

```
1.  int main(int argc, char **argv) {
2.      int pipefd[2];
3.      int ret, wstat, pid1, pid2;
4.      //char **param={"EXENAME", NULL};
5.      pipe(pipefd);
6.      pid1 = fork();  //產生第一個child
7.      if (pid1==0) {
8.          close(1);  //關閉stdout
9.          dup(pipefd[1]); //將pipefd[1]複製到stdout
10.         close(pipefd[1]);  //將沒用到的關閉
11.         close(pipefd[0]);  //將沒用到的關閉
12.         execlp("ls", "ls", NULL);  //執行ls，ls會將東西藉由stdout輸出到pipefd[1]
13.     } else printf("1st child's pid = %d\n", pid1);
14.     if (pid1>0) {
```

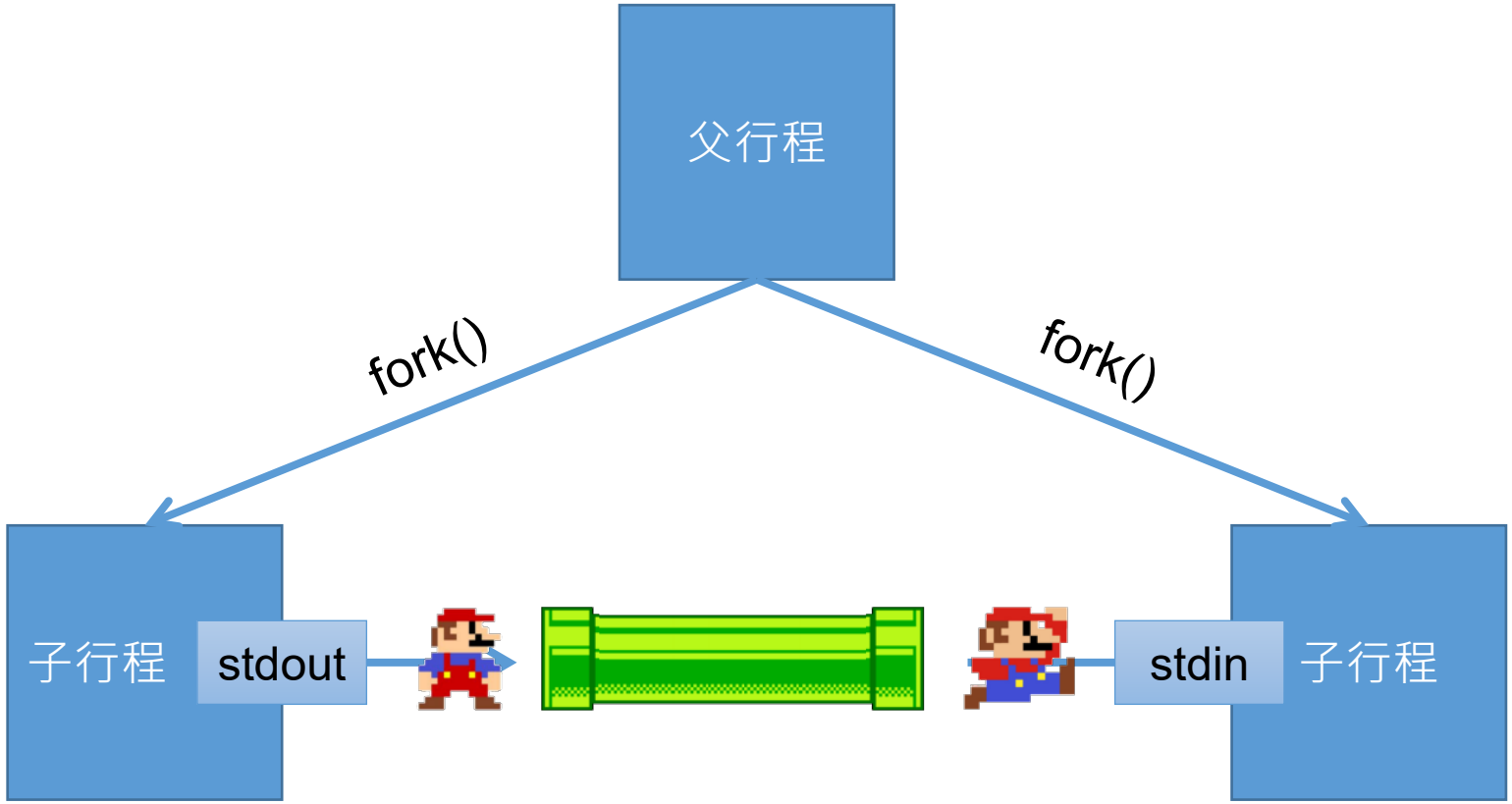
常見寫法三-2，子行程間通訊（pipe4-2.c）

```
15.       pid2 = fork();//產生第二個child
16.       if (pid2==0) {
17.          close(0);  //關閉stdin
18.          dup(pipefd[0]); //將pipefd[0]複製到stdin
19.          close(pipefd[1]);  //將沒用到的關閉
20.          close(pipefd[0]);  //將沒用到的關閉
21.          execlp("wc","wc", NULL);  //執行wc，wc將透過stdin從pipefd[0]讀入資料
22.       } else printf("2nd child's pid = %d\n", pid2);
23.    }
24.    //parent一定要記得關掉pipe不然wc不會結束（因為沒有接到EOF）
25.    close(pipefd[0]); close(pipefd[1]);
26.    printf("child %d\n",wait(&wstat));
27.    printf("child %d\n",wait(&wstat));
28. }
```

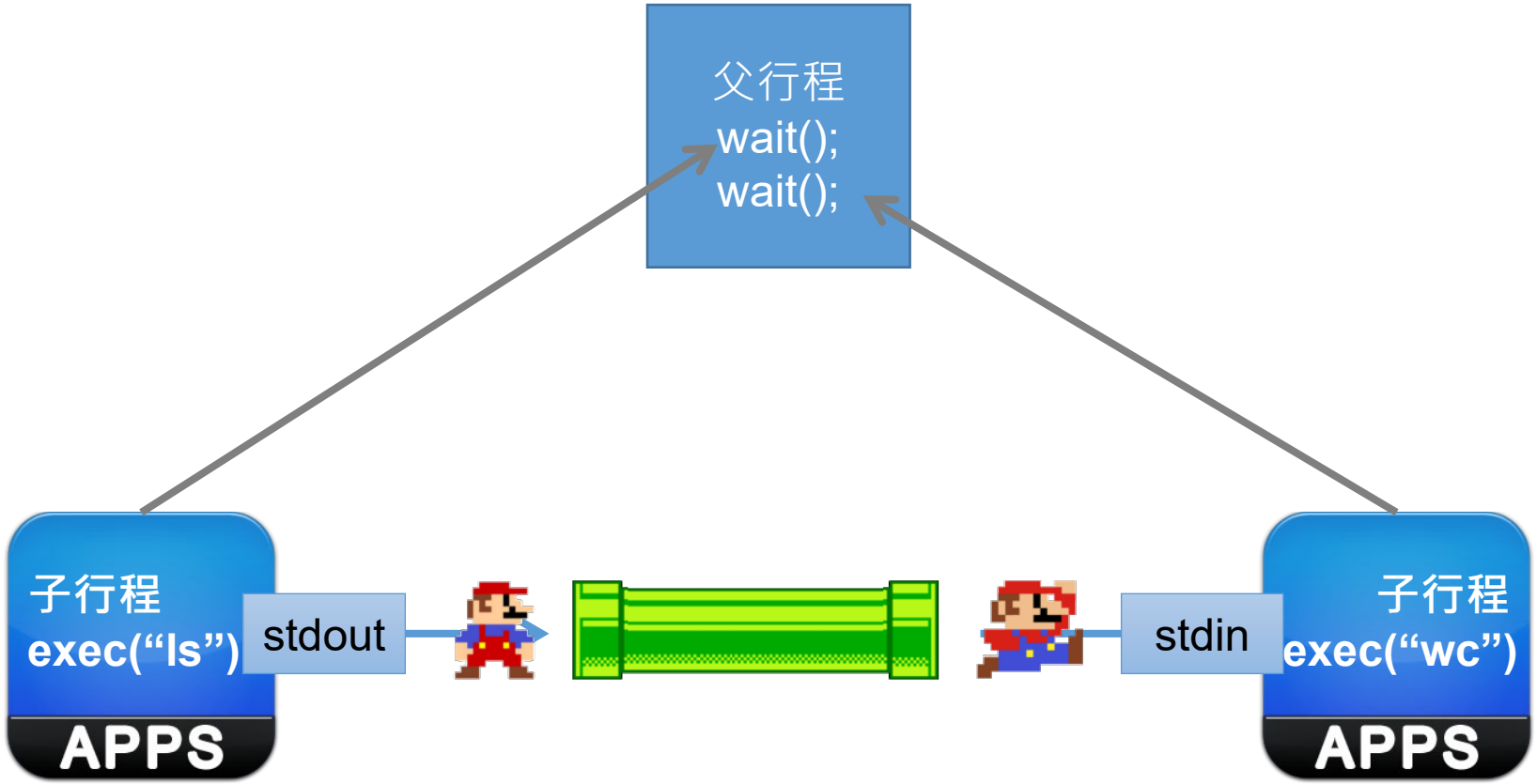
程式說明

- 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

示意圖



示意圖



結果

```
$ ./pipe4-2
1st child's pid = 27705
2nd child's pid = 27706
     18      18     139
child 27705
child 27706
$ ls | wc
     18      18     139
```

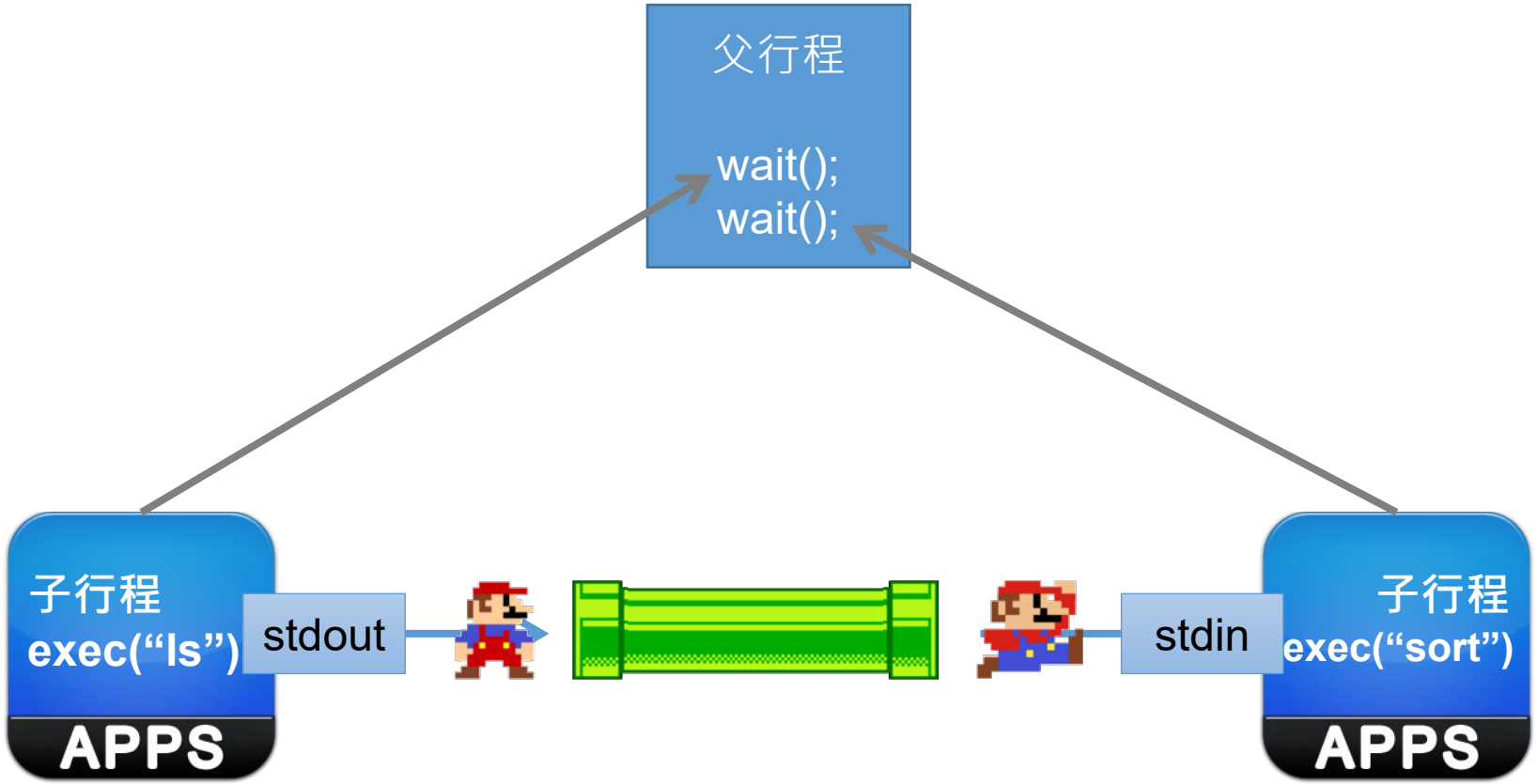
pipe & process group

```
1.  void signal_ctr_c(int signum) {
2.      kill(-1*pid1, signum);    //殺掉process group
3.      _exit(0);
4.  }
5.  int main(int argc, char **argv) {
6.      pipe(pipefd);
7.      pid1 = fork();   //產生第一個child
8.      if (pid1==0) {
9.          setpgid(0, 0);  //將第一個child設定為新的group
10.         close(1);   //關閉stdout
11.         dup(pipefd[1]); //將pipefd[1]複製到stdout
12.         close(pipefd[1]);   //將沒用到的關閉
13.         close(pipefd[0]);   //將沒用到的關閉
14.         execlp("ls", "ls", "-R", "/", NULL);//執行ls，ls會將東西藉由stdout輸出到pipefd[1]
15.     } else printf("1st child's pid = %d\n", pid1);
```

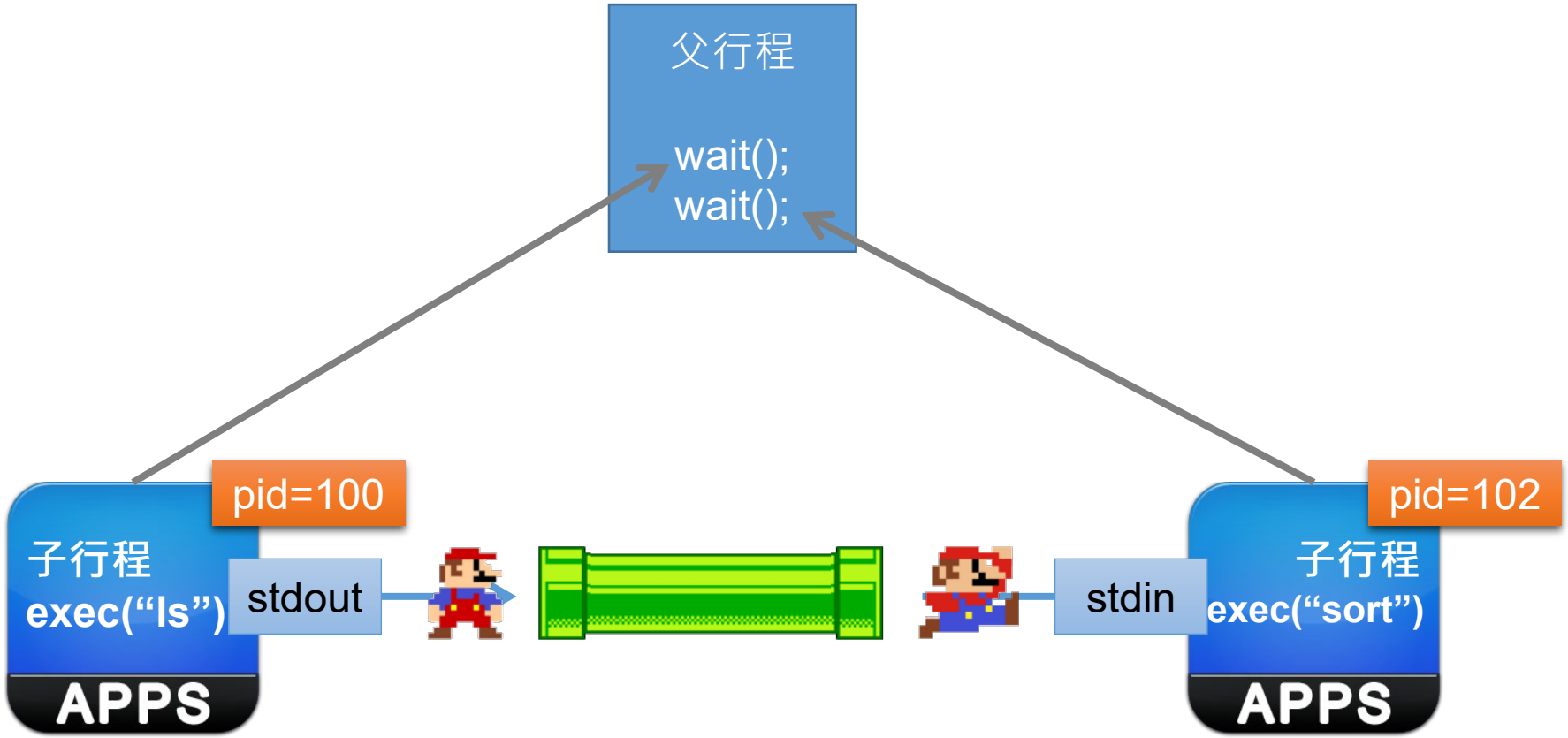
pipe & process group

```
16.         if (pid1>0) {
17.             pid2 = fork();//產生第二個child
18.             if (pid2==0) {
19.                 setpgid(0, pid1);   //第二個child加入第一個child的group
20.                 close(0);   //關閉stdin
21.                 dup(pipefd[0]); //將pipefd[0]複製到stdin
22.                 close(pipefd[1]);   //將沒用到的關閉
23.                 close(pipefd[0]);   //將沒用到的關閉
24.                 execlp("sort","sort", NULL);   //執行wc，wc將透過stdin從pipefd[0]讀入資料
25.             } else printf("2nd child's pid = %d\n", pid2);
26.         }
27.         close(pipefd[0]); close(pipefd[1]);     //parent一定要記得關掉pipe不然wc不會結束（因為沒有接到EOF）
28.         signal(SIGINT, signal_ctr_c);      /*parent註冊signal handler*/
29.         printf("child %d\n",wait(&wstat));
30.         printf("child %d\n",wait(&wstat));
31.     }
```

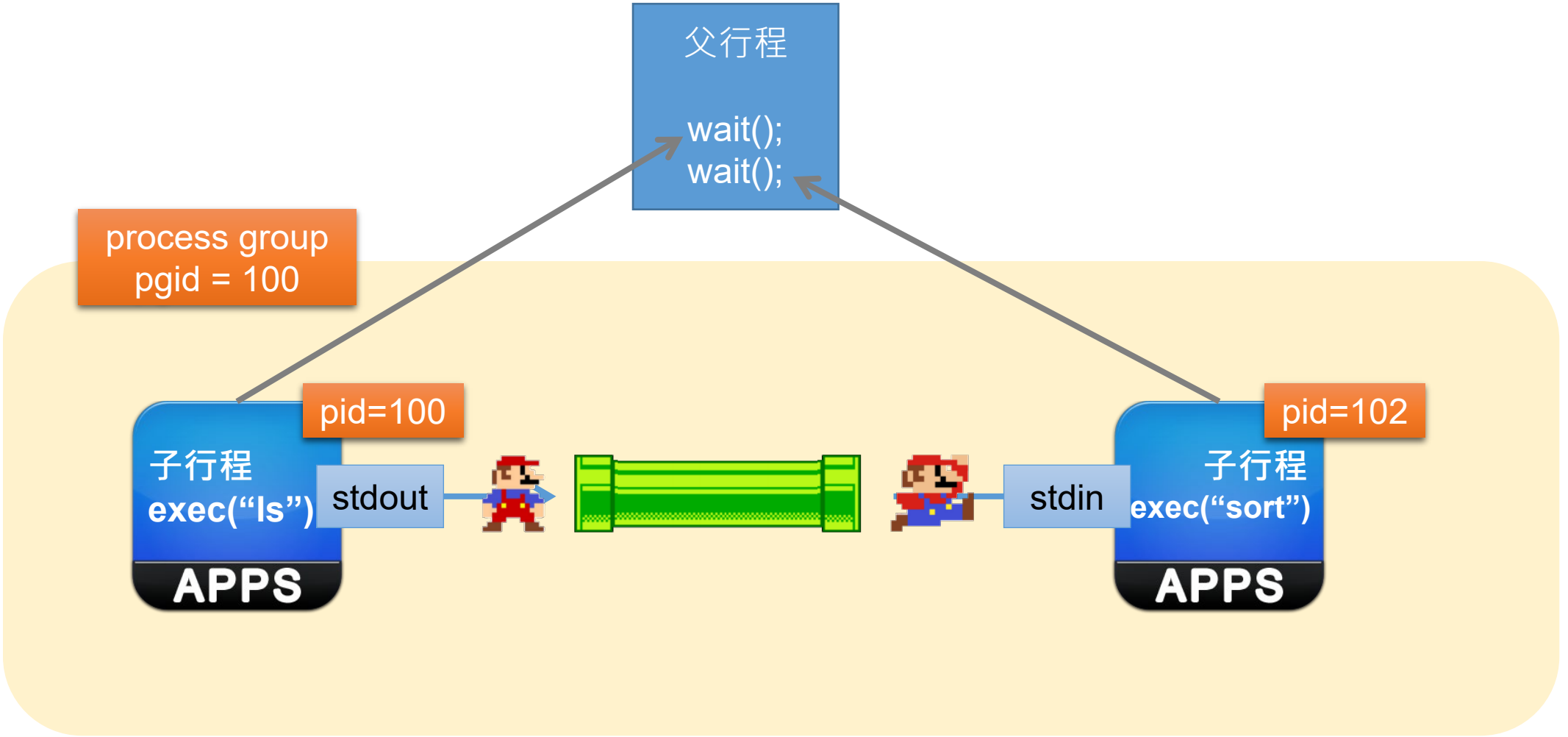
示意圖



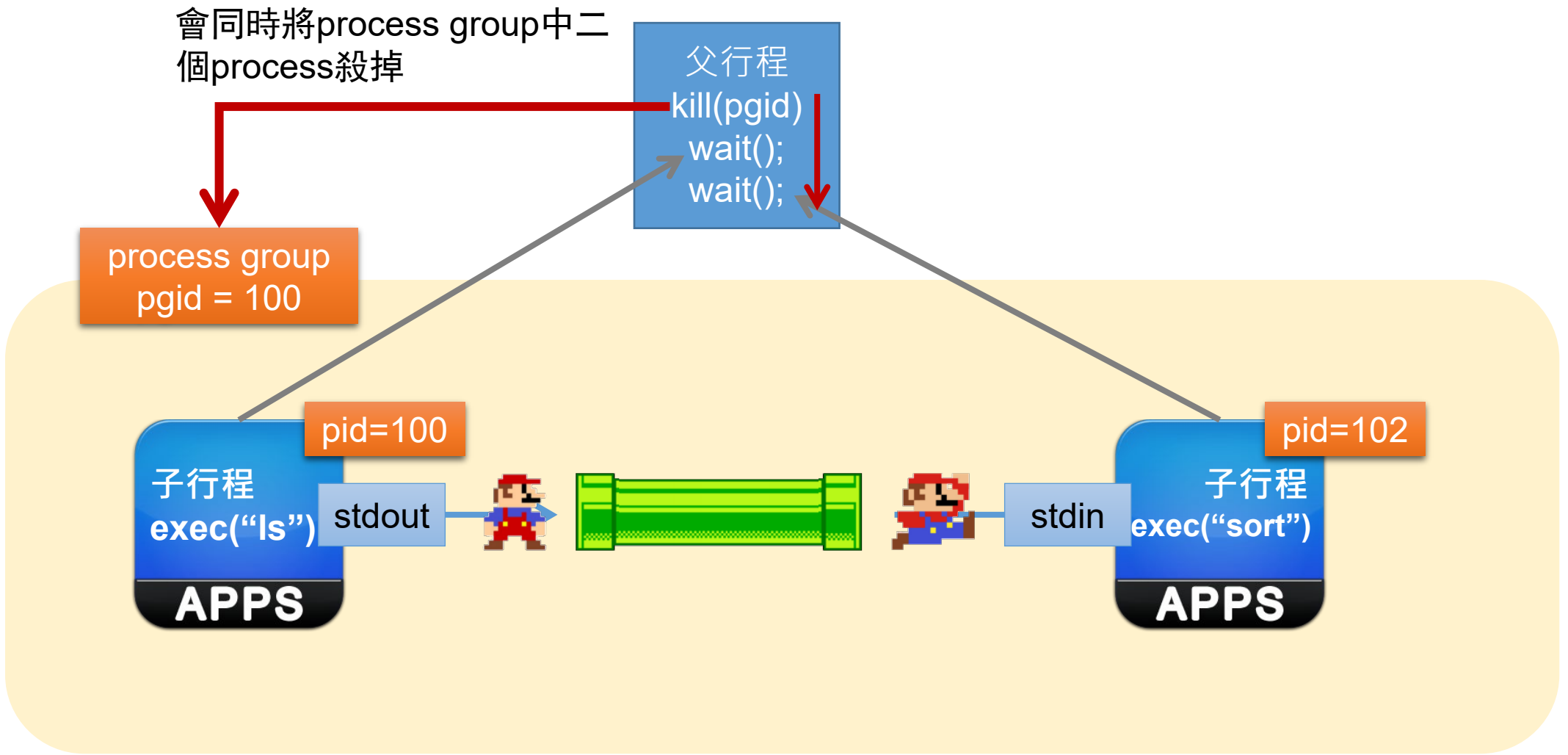
示意圖



示意圖



示意圖



setpgid()

- #include <sys/types.h>
- #include <unistd.h>
- int setpgid(pid_t pid, pid_t pgid);
- pid_t getpgid(pid_t pid);

- 設定process control group
- 常見用法：setpgid(0,0)，把目前這個process設為新的process group的「頭」，該 process group id為這個process的pid
- 常見用法：setpgid(pid, pgid)，將process “pid”加入到process group “pgid”中

執行結果

Kill process

```
$ ./pipe4-3 killproc
gpm-primary-060.svg
gpm-primary-060.svg
gpm-primary-060.svg
gpm-primary-060.svg
gpm-primary-060.svg
gpm-primary-080-charging.svg
gpm-primary-080-charging.svg
gpm-primary-080-charging.svg
gpm-primary-080-charging.svg
gpm-primary-080-charging.svg
^C
/*sort在ls死掉以後，雖然pipe斷掉，但是還是將pipe內的資料繼續排序*/
```

Kill group

```
$ ./pipe4-3 killgrp
Translation-he.xz
^CTranslation-he.xz
Translation-he.xz
Translation-he.xz
Translation-he.xz
Translation-he.xz
Translation-he.xz
Translation-he.xz
Translation-he.xz
Translation-he.xz
kill process group -13599
Translation-he.xz
Translation-he.xz
/*sort馬上死掉*/
```

pipe_perf.c，測試pipe的效率及學習alarm

```
1.  int cont = 1;
2.  void sig_alarm(int signNo) {
3.      cont = 0;                   //讓main跳出loop
4.  }
5.  int main(int argc, char **argv) {
6.      signal(SIGALRM, sig_alarm); //註冊alarm
7.      signal(SIGPIPE, SIG_IGN);   //避免有人離開造成pipe中斷，同學們可以註解掉這段程式碼看看
8.      pipe(pipefd);
9.      child_pid = fork();
10.     if (child_pid==0) {     //child;
11.         close(pipefd[0]);   /*for read*/
12.     //於"alarmSec"sec以後發出SIGALRM這個訊號 fork並不會繼承alarm，因此要對parent和child各設定一次
13.         alarm(alarmSec);
14.         while(cont) {       //一直執行，直到程式收到alarm這個signal
15.             write(pipefd[1], buf, nsize);
16.             nloop++;
17.         }
18.         close(pipefd[1]);
19.         printf("child write: throughput\t%.2fMB\n", (((float)nsize* nloop)/ alarmSec/(1024*1024)));
```

```
20.         } else {            //parent
21.             close(pipefd[1]);
22.             alarm(alarmSec);
23.             while(cont) {
24.                 read(pipefd[0], buf, nsize);
25.                 nloop++;
26.             }
27.             close(pipefd[0]);  //要關掉，不然child收不到EOF
28.             wait(&wstatus);
29.             printf("parent read: throughput\t%.2fMB\n", (((float)nsize* nloop)/ alarmSec/(1024*1024)));
30.             //測試單純記憶體複製的速度
31.             cont = 1;
32.             nloop = 0;
33.             alarm(alarmSec);
34.             while(cont) {
35.                 memcpy(dest, buf, nsize);
36.                 nloop++;
37.             }
38.             printf("speed of memcpy is \t%.2fMB/s\n", (((float)nsize* nloop)/ alarmSec/(1024*1024)));
39.             return 0;
40.         }
41.     }
```

關於alarm

- #include <unistd.h>
- unsigned int alarm(unsigned int seconds);

- 在seconds秒以後，OS發出SIGALRM這個signal給程式
- 如果seconds設定為0，則代表要取消alarm
- 一個程式最多只有一個alarm，如果設定多次的alarm，則以最後一次的設定為主

執行結果

```
shiwulo@NUC:~/sp/ch11$ ./pipe-perf 100 5
child write: throughput 183.89MB
parent read: throughput 183.89MB
speed of memcpy is 33005.52MB/s
shiwulo@NUC:~/sp/ch11$ ./pipe-perf 1000 5
child write: throughput 1759.77MB
parent read: throughput 1759.72MB
speed of memcpy is 87379.50MB/s
shiwulo@NUC:~/sp/ch11$ ./pipe-perf 10000 5
child write: throughput 5225.91MB
parent read: throughput 5225.42MB
speed of memcpy is 89776.95MB/s
shiwulo@NUC:~/sp/ch11$ ./pipe-perf 100000 5
child write: throughput 4503.59MB
parent read: throughput 4502.85MB
speed of memcpy is 35511.84MB/s
```

比alarm更強大 setitimer()

- #include <sys/time.h>
- int getitimer(int which, struct itimerval *curr_value);
- int setitimer(int which, const struct itimerval *new_value, struct itimerval *old_value);

- which的值可以是： ITIMER_REAL、 ITIMER_VIRTUAL、 ITIMER_PROF，分別會產生下列signal： SIGALRM、 SIGVTALRM、 ITIMER_VIRTUAL，代表的意義分別是：真實經過的時間、process花費在user space的時間、 process花費在user space和kernel space的時間
- itimerval資料結構請參考man 3 setitiemr
  - itimerval.it_interval：觸發signal的週期是多少
  - itimerval.it_value： 下次觸發的時間是何時（或者是：第一次觸發的時間是多少）

pipe_perf2.c，練習setitime()

```
1.  void sig_alarm(int signNo) {
2.      cont = 0;   /*讓main跳出loop*/ }
3.  int main(int argc, char **argv) {
4.      struct itimerval iTime={0}; iTime.it_value.tv_sec = alarmSec;
5.      struct itimerval virTime={0}; virTime.it_value.tv_sec = 100;  //很長，不會發生的時間
6.      struct itimerval profTime={0}; profTime.it_value.tv_sec = 100;  //很長，不會發生的時間
7.      signal(SIGALRM, sig_alarm); //註冊alarm
8.      signal(SIGPIPE, SIG_IGN);   //避免有人離開造成pipe中斷，同學們可以註解掉這段程式碼看看
9.      pipe(pipefd);
10.     child_pid = fork();
11.     if (child_pid==0) {     //child;
12.         close(pipefd[0]);    /*for read*/
13.         setitimer(ITIMER_REAL, &iTime, NULL); setitimer(ITIMER_VIRTUAL, &virTime, NULL);
14.         setitimer(ITIMER_PROF, &profTime, NULL);
15.         while(cont) {
16.             write(pipefd[1], buf, nsize); nloop++; }
17.         close(pipefd[1]);
18.         printf("\nchild write: throughput\t%.2fMB/s\n", (((float)nsize* nloop)/ alarmSec/MB));
19.         getitimer(ITIMER_VIRTUAL,&virTime);  getitimer(ITIMER_PROF,&profTime);
20.         printf("在user space花費\t= %fsec\n", 100.0-timeval2sec(virTime.it_value));
21.         printf("整個時間花費\t\t= %fsec\n", 100.0-timeval2sec(profTime.it_value));
```

```
22.     } else {            //parent
23.         close(pipefd[1]);
24.         setitimer(ITIMER_REAL, &iTime, NULL); setitimer(ITIMER_VIRTUAL, &virTime, NULL);
25.         setitimer(ITIMER_PROF, &profTime, NULL);
26.         while(cont) {   //一直執行，直到程式收到alarm這個signal
27.             read(pipefd[0], buf, nsize); nloop++; }
28.         close(pipefd[0]);   //要關掉，不然child收不到EOF
29.         printf("\nparent read: throughput\t%.2fMB/s\n", (((float)nsize* nloop)/ alarmSec/MB));
30.         getitimer(ITIMER_VIRTUAL,&virTime); getitimer(ITIMER_PROF,&profTime);
31.         printf("在user space花費\t= %fsec\n", 100.0-timeval2sec(virTime.it_value));
32.         printf("整個時間花費\t\t= %fsec\n", 100.0-timeval2sec(profTime.it_value));
33.         wait(&wstatus);
34.         return 0;
35.     }
36. }
```

執行結果

```
shiwulo@NUC:~/sp/ch11$ ./pipe-perf2 1000 5
parent read: throughput 1621.98MB/s
在user space花費 =          0.832000sec
整個時間花費 =              4.604000sec
child write: throughput 1622.00MB/s
在user space花費 =          0.852000sec
整個時間花費 =              4.996000sec
shiwulo@NUC:~/sp/ch11$ ./pipe-perf2 10000 5
parent read: throughput 4940.47MB/s
在user space花費 =          0.324000sec
整個時間花費 =              4.964000sec
child write: throughput 4940.91MB/s
在user space花費 =          0.300000sec
整個時間花費 =              4.920000sec
```

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.  char buf[200];
2.  int main(int argc, char **argv) {
3.      int fd;
4.      mkfifo(”/tmp/shiwulo", 0666);
5.      fd = open(”/tmp/shiwulo",O_RDWR);
6.      write(fd, "hello", sizeof("hello"));
7.      read(fd, buf, 200);
8.      printf("%s\n", buf);
9.      close(fd);
10.     unlink(”/tmp/shiwulo");
11.     return 0;
12. }
```

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(fd);
```

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(fd);
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writer寫出“hello”後結束執行（reader此時會收到writer的訊息）

執行結果

先執行fifo2-r

```
$ ./fifo2-r
mkfifo() = 0
hello
1234
5678
```

再執行fifo2-w

```
$ ./fifo2-w
mkfifo() = -1
```

小結

- pipe()可以建立二個fd，這二個fd分別可以進行讀取和寫入
- pipe()配合close()和dup()可以改變stdio及stdout的行為，對stdio/stdout的讀寫會變成對pipe的讀寫
- 大部分的fd（包含pipe）可以繼承給child，就算child執行execve後也繼承這些fd（ps:設定close-on-exec flag的fd不會繼承）
- 學到新的signal：alarm(sec)，“sec”秒以後系統會送出SIGALRM這個signal
- FIFO的用法與pipe差不多，他是有名字的pipe，他的名字就是一個pathName。在電腦領域FIFO又稱為named pipe

作業

1. 修改ch10的「shell_sigfd」或上個章節的作業「shell_sigaction」，新的shell名為「shell_pipe」，shell_pipe可以使用pipe串接數個程式，例如：

   ☘`ls -R --color / | sort | more`

2. 將使用pipe串接的程式設定為同一個process group，使得使用者按下ctr-c時可以同時中斷所有程式。

   ☘以「`ls -R --color / | sort | more`」為例，必須讓ls、sort、more成為同一個process group，當使用者按下ctr-c時必須送出ctr-c給這個process group