dup和dup2也是两个非常有用的调用，它们的作用都是用来复制一个文件的描述符。它们经常用来重定向进程的stdin、stdout和stderr。这两个函数的原形如下：

c代码

```
1. #include <unistd.h>
2. int dup( int oldfd );
3. int dup2( int oldfd, int targetfd );
```

dup2()函数

dup2函数跟dup函数相似，但dup2函数允许调用者规定一个有效描述符和目标描述符的id。dup2函数成功返回时，目标描述符（dup2函数的第二个参数）将变成源描述符（dup2函数的第一个参数）的复制品，换句话说，两个文件描述符现在都指向同一个文件，并且是函数第一个参数指向的文件。下面我们用一段代码加以说明：

c代码

```
1. int oldfd;
2. oldfd = open("app_log", (O_RDWR | O_CREATE), 0644 );
3. dup2( oldfd, 1 );
4. close( oldfd );
```

在本例中，我们打开了一个新文件，称为“app_log”，并收到一个文件描述符，该描述符叫做fd1。我们调用dup2函数，参数为oldfd和1，这会导致用我们新打开的文件描述符替换掉由1代表的文件描述符（即stdout，因为标准输出文件的id为1）。任何写到stdout的东西，现在都将改为写入名为“app_log”的文件中。需要注意的是，dup2函数在复制了oldfd之后，会立即将其关闭，但不会关掉新近打开的文件描述符，因为文件描述符1现在也指向它。

例子

下面我们介绍一个更加深入的示例代码。回忆一下命令行管道，我们可以将ls –l命令的标准输出作为标准输入连接到wc –l命令。接下来，我们就用一个C程序来加以说明这个过程的实现。代码如下所示。

c代码

```
1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <unistd.h>
4.
5. int main()
6. {
7.     int pfds[2];
8.
9.     if ( pipe(pfds) == 0 ) {
10.
11.      if ( fork() == 0 ) {
12.
13.          close(1);
14.          dup2( pfds[1], 1 );
15.          close( pfds[0] );
16.          execlp( "ls", "ls", "-l", NULL );
17.
18.      } else {
19.
20.          close(0);
21.          dup2( pfds[0], 0 );
22.          close( pfds[1] );
23.          execlp( "wc", "wc", "-l", NULL );
24.
25.      }
26.
27.     return 0;
28. }
```

在示例代码中，首先在第9行代码中建立一个管道，然后将应用程序分成两个进程：一个子进程（第13–16行）和一个父进程（第20–23行）。接下来，在子进程中首先关闭stdout描述符（第13行），然后提供了ls –l命令功能，不过它不是写到stdout（第13行），而是写到我们建立的管道的输出端，这是通过dup2函数来完成重定向的。在第14行，使用dup2 函数把stdout重定向到管道（pfds[1]）。之后，马上关掉管道的输入端。然后，使用execlp函数把子进程的映像替换为命令ls –l的进程映像，一旦该命令执行，它的任何输出都将发给管道的输入端。

现在来研究一下管道的接收端。从代码中可以看出，管道的接收端是由父进程来担当的。首先关闭stdin描述符（第20行），因为我们不会从机器的键盘等标准设备文件来接收数据的输入，而是从其它程序的输出中接收数据。然后，再一次用到dup2函数（第21行），让管道的输入端作为输入，这是通过让文件描述符0（即常规的stdin）重定向到pfds[0]实现的。关闭管道的stdout端（pfds[1]），因为在这里用不到它。最后，使用execlp函数把父进程的映像替换为命令wc -l的进程映像，命令wc -l把管道的内容作为它的输入（第23行）