<https://my.oschina.net/renhc/blog/35116>

标准I/O函数库提供了popen函数，它启动另外一个进程去执行一个shell命令行。

这里我们称调用popen的进程为父进程，由popen启动的进程称为子进程。

popen函数还创建一个管道用于父子进程间通信。父进程要么从管道读信息，要么向管道写信息，至于是读还是写取决于父进程调用popen时传递的参数。下在给出popen、pclose的定义：

#include <stdio.h>

/\*

函数功能：popen（）会调用fork（）产生子进程，然后从子进程中调用/bin/sh -c来执行参数command的指令。

参数type可使用“r”代表读取，“w”代表写入。

依照此type值，popen（）会建立管道连到子进程的标准输出设备或标准输入设备，然后返回一个文件指针。

随后进程便可利用此文件指针来读取子进程的输出设备或是写入到子进程的标准输入设备中

返回值：若成功则返回文件指针，否则返回NULL，错误原因存于errno中

\*/

FILE \* popen( const char \* command,const char \* type);

/\*

函数功能：pclose（）用来关闭由popen所建立的管道及文件指针。参数stream为先前由popen（）所返回的文件指针

返回值：若成功返回shell的终止状态(也即子进程的终止状态)，若出错返回-1，错误原因存于errno中

\*/

int pclose(FILE \* stream);

下面通过例子看下popen的使用：

假如我们想取得当前目录下的文件个数，在shell下我们可以使用：

ls | wc -l

我们可以在程序中这样写：

/\*取得当前目录下的文件个数\*/

#include <stdio.h>

#include <stdlib.h>

#include <errno.h>

#include <sys/wait.h>

#define MAXLINE 1024

int main()

{

char result\_buf[MAXLINE], command[MAXLINE];

int rc = 0; // 用于接收命令返回值

FILE \*fp;

/\*将要执行的命令写入buf\*/

snprintf(command, sizeof(command), "ls ./ | wc -l");

/\*执行预先设定的命令，并读出该命令的标准输出\*/

fp = popen(command, "r");

if(NULL == fp)

{

perror("popen执行失败！");

exit(1);

}

while(fgets(result\_buf, sizeof(result\_buf), fp) != NULL)

{

/\*为了下面输出好看些，把命令返回的换行符去掉\*/

if('\n' == result\_buf[strlen(result\_buf)-1])

{

result\_buf[strlen(result\_buf)-1] = '\0';

}

printf("命令【%s】 输出【%s】\r\n", command, result\_buf);

}

/\*等待命令执行完毕并关闭管道及文件指针\*/

rc = pclose(fp);

if(-1 == rc)

{

perror("关闭文件指针失败");

exit(1);

}

else

{

printf("命令【%s】子进程结束状态【%d】命令返回值【%d】\r\n", command, rc, WEXITSTATUS(rc));

}

return 0;

}

编译并执行：

$ gcc popen.c

$ ./a.out

*命令【ls ./ | wc -l】 输出【2】*

*命令【ls ./ | wc -l】子进程结束状态【0】命令返回值【0】*

上面popen只捕获了command的标准输出，如果command执行失败，子进程会把错误信息打印到标准错误输出，父进程就无法获取。比如，command命令为“ls nofile.txt” ，事实上我们根本没有nofile.txt这个文件，这时shell会输出“ls: nofile.txt: No such file or directory”。这个输出是在标准错误输出上的。通过上面的程序并无法获取。

注：如果你把上面程序中的command设成“ls nofile.txt”,编译执行程序你会看到如下结果：

$ gcc popen.c

$ ./a.out

*ls: nofile.txt: No such file or directory*

*命令【ls nofile.txt】子进程结束状态【256】命令返回值【1】*

 需要注意的是第一行输出并不是父进程的输出，而是子进程的标准错误输出。

有时子进程的错误信息是很有用的，那么父进程怎么才能获取子进程的错误信息呢？

**这里我们可以重定向子进程的错误输出，让错误输出重定向到标准输出（2>&1），这样父进程就可以捕获子进程的错误信息了。**例如command为“ls nofile.txt 2>&1”,输出如下：

命令【ls nofile.txt 2>&1】 输出【ls: nofile.txt: No such file or directory】

命令【ls nofile.txt 2>&1】子进程结束状态【256】命令返回值【1】

**附：子进程的终止状态判断涉及到的宏，设进程终止状态为status.**

WIFEXITED(status)如果子进程正常结束则为非0值。

WEXITSTATUS(status)取得子进程exit()返回的结束代码，一般会先用WIFEXITED 来判断是否正常结束才能使用此宏。

WIFSIGNALED(status)如果子进程是因为信号而结束则此宏值为真。

WTERMSIG(status)取得子进程因信号而中止的信号代码，一般会先用WIFSIGNALED 来判断后才使用此宏。

WIFSTOPPED(status)如果子进程处于暂停执行情况则此宏值为真。一般只有使用WUNTRACED 时才会有此情况。

WSTOPSIG(status)取得引发子进程暂停的信号代码，一般会先用WIFSTOPPED 来判断后才使用此宏。

2011-11-12 任洪彩 qdurenhongcai@163.com