**实验五 实现带参数的简单shell**

实验五利用课本第9页程序1-5的框架，实现允许输入命令带参数的简单shell。原来的实现是不能够带参数的。输入命令所能带的参数个数，只受到系统键盘输入缓冲区长度（以及shell输入缓冲区长度）的限制，该缓冲区的缺省长度是4096个字节。实现时要解决的主要问题有：

1. 正确理解并使用系统调用fork()，execve()和waitpid()，特别是execve()函数。

fork()函数创建一个新的进程。新进程就是所谓的子进程，它是执行fork()函数的进程(父进程)的“克隆”，也就是说，子进程执行的程序与父进程的完全一样。当fork()函数返回值为0时表示处于子进程中；而返回值大于0时表示处于父进程中，此时的返回值是子进程的进程id。因此，fork()的返回值可以用来划分仅仅适合父进程和子进程执行的程序段。fork()函数返回值为-1时表示出错。

如果子进程只是运行与父进程完全一样的程序，那用处是很有限的。要让子进程运行不同于父进程的程序，就必须调用execve函数，它是所有其他exec函数的基础。execve函数把调用它的进程的程序，替换成execve函数的参数所指定的程序。运行execve函数成功后，进程将开始运行新的程序，也就是execve函数的参数所指定的程序。

execve函数原型：int execve(const char \*path, const char \*argv[],const char \*envp[]);

其中：

path：要执行的程序路径名，比如“/bin/ls”，“cd”，“/usr/bin/gcc”等等。

argv：参数表，比如ls命令中可带的命令行参数-l,-a等。注意，argv的第一个元素必须是要执行的程序（命令）的路径名。

envp：环境变量表，供要执行的命令使用。实参数用NULL或系统环境变量environ均可。注意，因为environ由系统提供，属于外部变量，所以说明时必须用“extern”修饰。

例子：

char \*argv[] = {“gcc”, “-g”, “-c”, “hello.c”, NULL};

char \*argv1[] = {“/bin/ls”, “-l”, “-a”, NULL};

execve(“/usr/bin/gcc”, argv, environ); // 编译程序“hello.c”

execve(“/bin/ls”, argv1, NULL); // 执行命令“ls –l –a”

execve(“/usr/ls”, argv1, NULL); // 出错，因为目录/usr/下没有ls程序。

// 注意，在argv1 的第一个字符串“/bin/ls”中，只有ls是有用的。

系统调用waitpid()用于等待子进程结束、获取子进程的运行状态，详细说明在第八章。本实验仅仅用它使父进程等待子进程结束，因此维持程序1-5的用法即可。

2．根据简单shell的输入，构造execve函数的参数。

根据程序1-5，数组buf保存用户的输入，包括命令和参数。由于shell命令的命令名和各参数之间是用空格分开，因此可以用空格作为分界符。通过一个循环可以把buf数组中的命令和各个参数依次分离开来，并赋给数组argv的各元素适当的指针值。argv数组的最后一个指针必须是NULL。接着就可以调用execve(argv[0],argv, environ)来执行用户输入的命令。

提示：argv数组中各指针所指向的字符串，可以直接利用buf的存储空间，不需要另外分配内存。

1. 正确编译程序。

由于书中例子用到了本书第一版中作者自己定义的error.c文件，因此编译时要记得把error.c文件和源代码文件一起编译，error.c文件可以到cs8的/home/luwei/common目录下复制。可以用如下命令编译：

gcc <源文件名> error.c –o<可执行文件名>