项目四：实现简单的 Shell

1　学时

- 2 学时

2　实验目的

- 理解、掌握、应用进程相关的 fork,exec,exit,wait 等函数的使用。

3　实验内容

- 编写程序，实现简单的 Shell，该 Shell 能完成根据用户输入调用外部命令的功能，调用的外部命令可以加参数。

4　实验原理

4.1　程序和进程概念

程序是为了完成某项任务编排的语句序列，它告诉计算机如何执行，因此程序是需要运行的。程序运行过程中需要占有计算机的各种资源才能运行下去。

进程是程序在数据集合上的一次运行过程，是拥有资源和分派资源的基本单位。它是操作系统动态执行的基本单元，在传统的操作系统中，进程既是基本的分配单元，也是基本的执行单元。进程映像主要由 PCB、用户数据段、用户数据段、堆栈段组成。

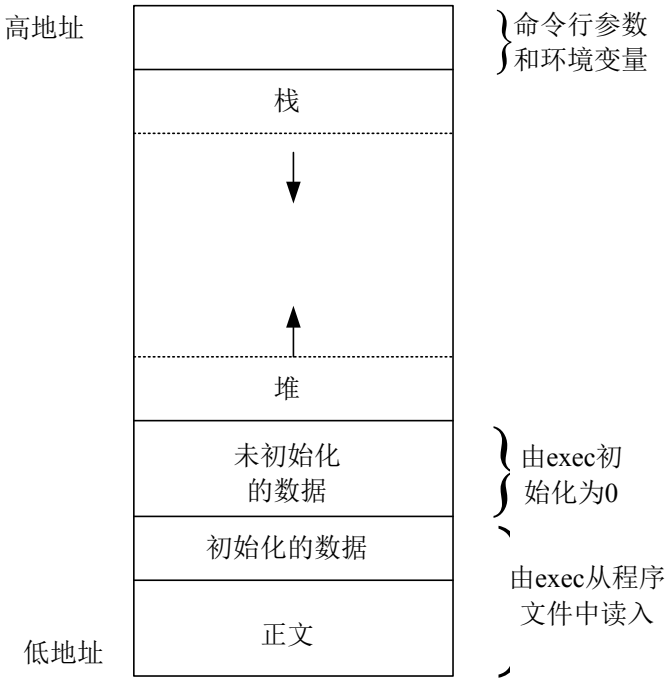


图 1　典型的存储器安排

程序是静态概念，本身可以作为一种软件资源保存；而进程是程序的一次执行过程，是动态概念，它有一定的生命期，是动态地产生和消亡的。

进程是一个能独立运行的单位，能与其他进程并发执行，进程是作为自愿申请和调度单位存在的；而通常的程序不能作为一个独立运行的单位。

程序与进程无一一对应关系，一方面一个程序可由多个进程共用;另一方面一个进程只能对应一个程序。进程和程序的关系犹如演出和剧本的关系。

在一个计算机系统中，CPU 可以交替执行多个程序。尽管 CPU 一直在运行，但就某个正在执行的程序而言，它并不是一直占用 CPU：可能因为超时让出 CPU，也可能因为等待 I/O 事件让出 CPU。于是就有了图 2 所示的进程（执行中的程序）的状态模型。这里执行状态是指进程占用 CPU 的状态，就绪状态是指因为时间超时而让出 CPU 的状态，阻塞状态是指因为等待 I/O 事件而让出 CPU 的状态。

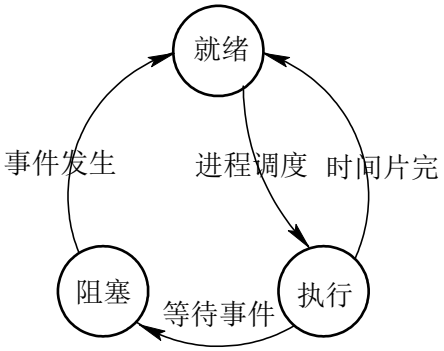


图 2 进程三状态转换

4.2 fork

一个进程调用了 fork 以后,系统会创建一个子进程。这个子进程和父进程不同的地方只有他的进程 ID 和父进程 ID，其他的都一样。

当一个程序中调用 fork 函数后，内核会完成如下工作：

内核系统分配新的内存块和内核数据结构

复制原来的进程到新的进程

向运行进程集添加新的进程

将进程返回给两个进程

设原来的进程为父进程，调用 fork 生成的新进程为子进程，则子进程会执行父进程中 fork 函数后的代码。

fork 系统调用使用格式：

头文件：#include <sys/types.h> /* 提供类型 pid_t 的定义 */

#include <unistd.h>

函数原形：pid_t fork(void);

返回值：pid_t

对于父进程，fork 函数返回了子程序的进程号，而对于子程序，fork 函数则返回零。这样，对于程序，只要判断 fork 函数的返回值，就知道自己是处于父

进程还是子进程中。如果调用不成功，则返回-1。

4.3 exec 族函数

exec 函数族的作用是根据指定的文件名找到可执行文件，并用它来取代调用进程的内容，换句话说，就是在调用进程内部执行一个可执行文件。这里的可执行文件既可以是二进制文件，也可以是任何 Linux 下可执行的脚本文件。

exec 系统调用有六种不同的使用格式，但在核心中只对应一个调用入口。它们有不同的调用格式和调用参数。这六种调用格式分别为：

```
#include <unistd.h>
int execl (const char *path, const char *arg0, ..., const char*argn, (char *)0);
int execv (const char *path, char *const *argv);
int execle (const char *path, const char *arg0, ..., const char*argn,(char *0), const char *envp[]);
int execve (const char *path, char *const *argv, char *const *envp);
int execlp (const char *file, const char *arg0, ..., const char*argn, (char *)0);
int execvp (const char *file, char *const *argv);
```

说明：参数 path 指出一个可执行目标文件的路径名；参数 file 指出可执行目标文件的文件名。arg0 作为约定同 path 一样指出目标文件的路径名；参数 arg1 到 argn 分别是该目标文件执行时所带的命令行参数；参数 argv 是一个字符串指针数组，由它指出该目标程序使用的命令行参数表，按约定第一个字符指针指向与 path 或 file 相同的字符串；最后一个指针指向一个空字符串，其余的指向该程序执行时所带的命令行参数；参数 envp 同 argv 一样也是一个字符指针数组，由它指出该目标程序执行时的进程环境，它也以一个空指针结束。

exec 的六种格式在以下三点上有所不同：

path 是一个目标文件的完整路径名，而 file 是目标文件名，它是可以通过环境变量 PATH 来搜索的；

由 path 或 file 指定的目标文件的命令行参数是完整的参数列表或是通过一指针数组 argv 来给出的；

环境变量是系统自动传递或者通过 envp 来给出的。

下表说明了 exec 系统调用的六种不同格式对以上三点的支持。

表 1 exec 系统调用的六种不同格式

系统调用	参数形式	环境传送	路径搜索
execl	全部列表	自动	否
execv	指针数组	自动	否
execle	全部列表	不自动	否
execve	指针数组	不自动	否

execlp	全部列表	自动	是
execvp	指针数组	自动	是

与一般情况不同，exec 函数族的函数执行成功后不会返回，因为调用进程的实体，包括代码段，数据段和堆栈等都已经被新的内容取代，只留下进程 ID 等一些表面上的信息仍保持原样，颇有些神似"三十六计"中的"金蝉脱壳"。看上去还是旧的躯壳，却已经注入了新的灵魂。只有调用失败了，它们才会返回一个-1，从原程序的调用点接着往下执行。

Linux 下是如何执行新程序的？每当有进程认为自己不能为系统和拥护做出任何贡献了，他就可以发挥最后一点余热，调用任何一个 exec，让自己以新的面貌重生；或者，更普遍的情况是，如果一个进程想执行另一个程序，它就可以 fork 出一个新进程，然后调用任何一个 exec，这样看起来就好像通过执行应用程序而产生了一个新进程一样。

事实上第二种情况被应用得如此普遍，以至于 Linux 专门为其作了优化，我们已经知道，fork 会将调用进程的所有内容原封不动的拷贝到新产生的子进程中去，这些拷贝的动作很消耗时间，而如果 fork 完之后我们马上就调用 exec，这些辛辛苦苦拷贝来的东西又会被立刻抹掉，这看起来非常不划算，于是人们设计了一种"写时拷贝（copy-on-write）"技术，使得 fork 结束后并不立刻复制父进程的内容，而是到了真正实用的时候才复制，这样如果下一条语句是 exec，它就不会白白作无用功了，也就提高了效率。

4.4 wait

wait 可以完成父进程和子进程的同步。进程一旦调用了 wait，就立即阻塞自己，由 wait 自动分析是否当前进程的某个子进程已经退出，如果让它找到了这样一个已经变成僵尸的子进程，wait 就会收集这个子进程的信息，并把它彻底销毁后返回；如果没有找到这样一个子进程，wait 就会一直阻塞在这里，直到有一个出现为止。

wait 系统调用使用格式：

```
#include <sys/types.h>     /*  提供类型 pid_t 的定义  */
#include <sys/wait.h>
pid_t wait(int *status)
```

参数 status 用来保存被收集进程退出时的一些状态，它是一个指向 int 类型的指针。但如果我们对这个子进程是如何死掉的毫不在意，只想把这个僵尸进程消灭掉（事实上绝大多数情况下，我们都会这样想），我们就可以设定这个参数为 NULL，就象这样：pid = wait(NULL);

4.5 exit

exit 系统调用格式如下：

#include <stdlib.h>

void exit(int status);

#include <unistd.h>

void _exit(int status);

说明：exit 的功能是终止进程的执行，并释放该进程所占用的某些系统资源。参数 status 是调用进程终止时传递给其父进程的值。如果调用进程执行 exit 系统调用时，其父进程正在等待子进程暂停或终止（使用 wait 系统调用），则父进程可立刻得到该值；如果此时父进程并不处在等待状态，那么一旦父进程使用 wait 调用，便可立刻得到子进程传过来的 status 值，注意：只有 status 的低八位才传递给其父进程。

系统调用_exit 与 exit 之间的差异是_exit 只做部分的清除，因此建议不要轻易地使用这种调用形式。

每个进程在消亡前都要调用该系统调用，没有显示地使用该系统调用，则生成目标文件的装载程序为该进程隐含地做这一工作。

4.6 Shell

Shell 也是程序，用户登录后便开始运行。通过 Shell 执行程序的过程：

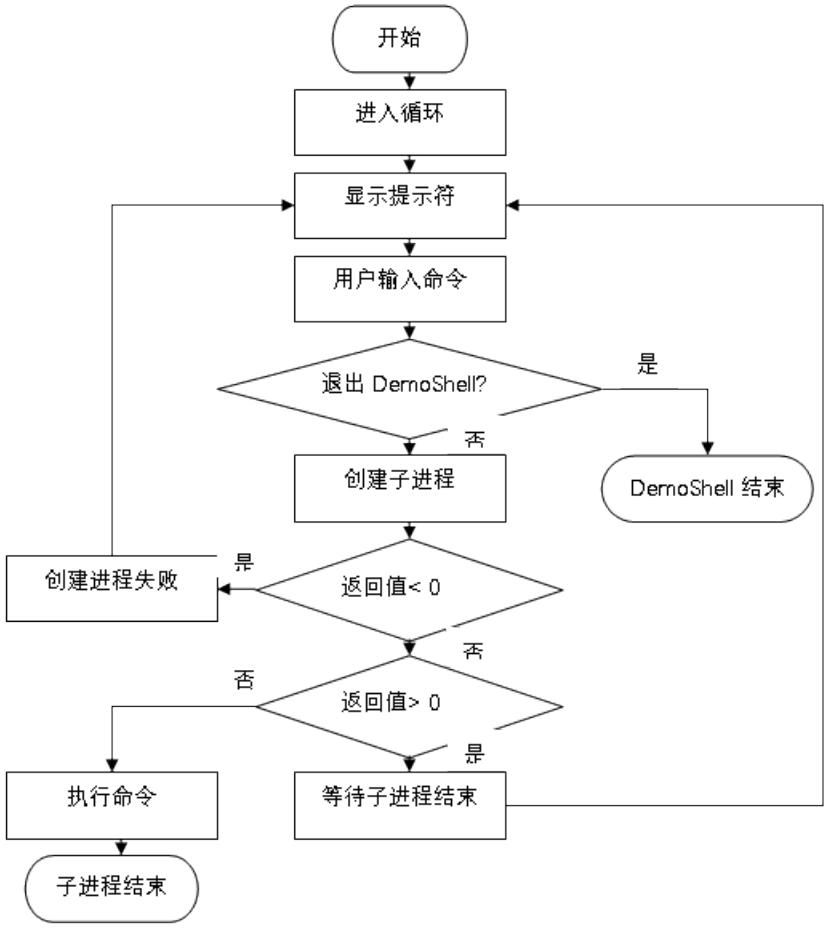
（1）键入可执行文件路径名

（2）Shell 创建一个新的进程来运行这个程序

（3）Shell 将程序从磁盘载入

（4）Shell 等待该程序运行结束

Shell 就是这样一个程序：不断地进行获取命令、执行命令、等待命令结束的循环过程。程序流程如下图所示。



5 预习要求和技术准备工作

- 掌握进程基本概念
- 掌握 fork、exec 族、exit、wait 函数的使用
- 掌握 Shell 执行过程

6 实验环境

- PC 机
- 在 Windows 环境中的 VMware 虚拟机上运行 RedhatLinux9.0 操作系统或者独立的 Redhat Linux 9.0 操作系统
- 基于 Linux 的 vi 编辑器和 gcc 编译器

7 实验设计及操作步骤

7.1 以 root 身份登录系统，在/home 目录中创建目录 exp44

```
cd  /home
mkdir  exp44
```

7.2 进入刚创建的目录

```
cd  exp44
```

7.3 使用 vi 编辑文件，文件名为 Demoshell.c

vi　Demoshell.c

7.4　　编写程序，实现一个简单的 Shell 功能：

★要求：

（1）能够接收用户输入的外部命令（具有独立可执行文件的命令，如 ls，ps 等）并正确执行该命令，输入的命令可以带参数；

（2）命令提示符为：%；

（3）如果输入“exit”或“logout”时退出 Shell；

（4）如果输入的命令不存在则提示“命令名：unknown command”。

★编程思路参考：

假设用户输入的命令保存在 s 字符数组中，将 s 解析后的字符串数组为 cmd。

```
char　s[64];
char *cmd[64];
main()
```

{Shell 本身是一个循环过程，循环次数不限，循环体可包括如下内容：

①输出命令提示符%

②使用 fgets 获取用户输入，假设保存在 s 中，fgets(s,sizeof(s),stdin);

③判断 s 中是否只保存了“\n”，如果是则结束本次循环，继续下次循环。

④用户输入时以“\n”结尾，去掉 s 中的回车，s[strlen(s)-1]=0;

⑤调用自定义函数（如 parse）解析 s 字符串，成为各选项参数独立的字符串数组 cmd，有效选项参数后以 NULL 为结尾，并返回选项参数个数。

⑥判断 cmd［0]如果是“exit”或“logout”，

是：退出循环，结束进程。

否：创建子进程（fork），子进程使用 execvp 执行程序 cmd 对应的程序，如果执行失败则提示“命令名：unknown command”，然后退出子进程。父进程等待子进程结束（wait）。

}

★关于解析 s 字符串函数，以下给出参考程序，建议自行编写。

```
/*参数：buf---用户输入的除去回车的命令字符串
        args---存放解析后各选项参数字符串的字符指针数组
返回值：有效的选项参数个数
*/
int parse (char *buf, char **args)
{
int num=0;
        while (*buf != '\0')
```

```
        {
                //该循环是定位到命令中每个字符串的第一个非空的字符
              while((* buf = =' ')||(* buf = = '\t')||(*buf = = '\n'))
                    *buf++ = '\0';
                //将找到的非空字符串  依次赋值给 args[i]。
                *args++ = buf;
                //选项参数个数加 1
            ++num;
                 //正常字母就往后移动，直至定位到正常字符后第一个空格
              while ((*buf!='\0')&&(* buf!=' ')&&(* buf!= '\t') && (*buf!= '\n'))
                          buf ++;
        }
    //字符指针数组结尾置 NULL
    *args = '\0';
    //返回选项参数个数
    return num;
}
```

7.5 编译 Demoshell.c 为可执行文件 Demoshell，并能正确执行。

```
gcc  Demoshell.c  -o  Demoshell
```

7.6 执行 Demoshell

```
./Demoshell
```

此时，这个程序就是一个简单的 shell，用户在命令提示符%后输入系统命令（外部命令，非 shell 内部命令），将默认执行。参考示例如下：

```
cst@ubuntu:~/linuxsys2021/44/teacher$ ./Demoshell
% ls
Demoshell  Demoshell.c  Demoshell.zip
% ls -l
总用量 28
-rwxrwxr-x 1 cst cst 17112 7月  23 10:57 Demoshell
-rw-rw-r-- 1 cst cst  2090 7月  16 17:04 Demoshell.c
-rwxrw-rw- 1 cst cst  1097 2月  28 16:28 Demoshell.zip
% ps
    PID TTY          TIME CMD
  20544 pts/5    00:00:00 bash
  21295 pts/5    00:00:00 Demoshell
  21298 pts/5    00:00:00 ps
% exit
```

8 实验报告提交要求：

将实验操作每个步骤中的命令、源程序以及截图写入实验报告，实验报告命

名为“学号姓名-实验 44.doc”，交给指定人员。

9 项目思考

1、由于实际 ls 命令的各个参数顺序无所谓，即“-al” “-la” “-a –l” “-l –a”都是同样的功能，怎么实现这种判断？提示：可以使用 getopt 函数完成。

2、这个 shell 可以作为登录用户的 shell，该如何去设置？