Operating System

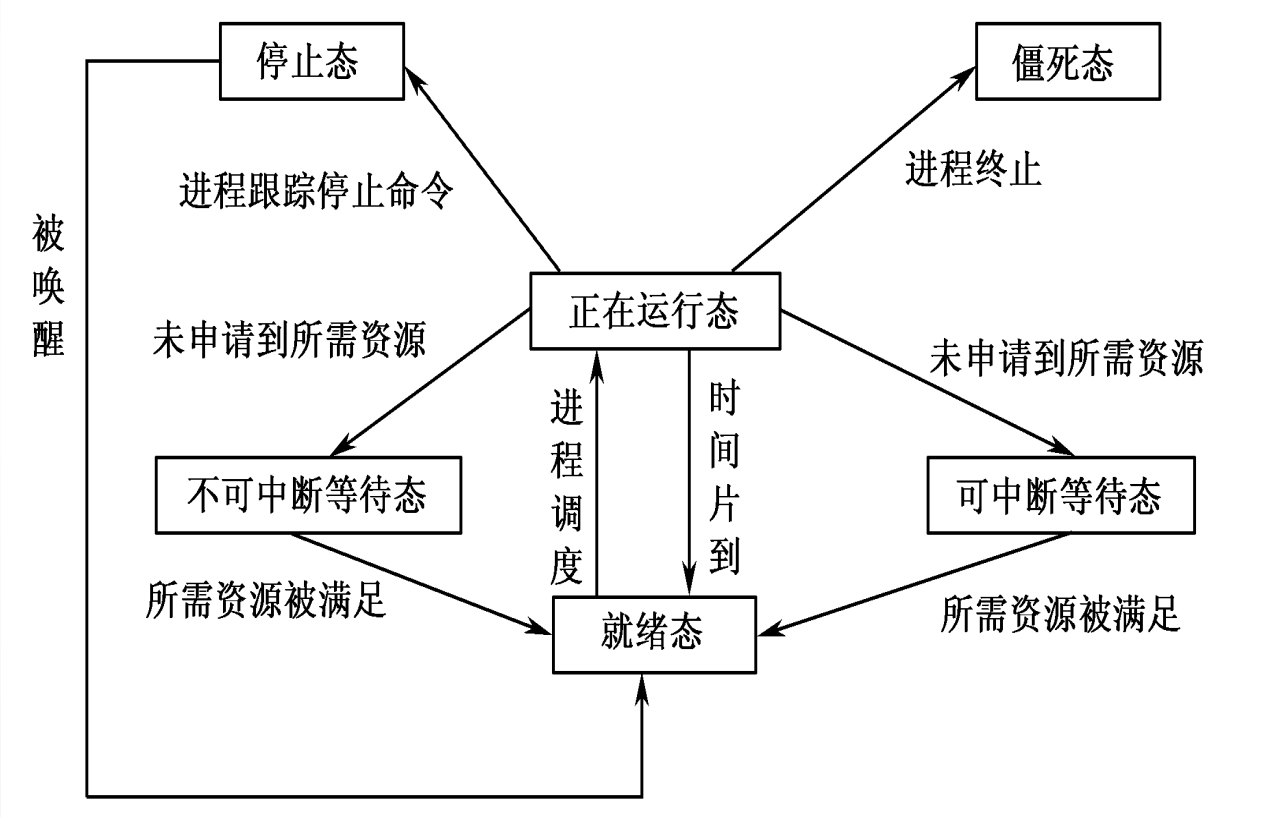
Lab05 Introduction

Linux进程编程1/2

- 什么是进程？
  - 简单说来，**进程就是程序的一次执行过程**。
  - 进程至少要有三种基本状态。这三种基本状态是：运行态、就绪态和封锁态（或等待态）。
  - 进程的状态可依据一定的条件和原因而变化

1、Linux进程编程1/2

- Linux进程状态之间的变化



1、Linux进程编程1/2

- 进程的属性

  基本属性：进程ID、父进程ID、进程组ID、会话和控制终端

- 进程ID（PID）

  函数定义：#include <sys/types.h>

  #include <unistd.h>

  pid_t getpid(void);

  函数说明：每一个进程都有一个非负整型表示的唯一进程ID（PID）。

1、Linux进程编程1/2

- 父进程ID（PPID）

函数定义：#include <sys/types.h>

#include <unistd.h>

pid_t getppid(void);

函数说明：不论什么进程（除init进程）都是由另一个进程创建。该进程称为被创建进程的父进程，被创建的进程称为子进程。

1、Linux进程编程1/2

- 进程组ID（PGID）

函数定义：#include <unistd.h>

int setpgid(pid_t pid, pid_t pgid);//设置进程组号

pid_t getpgid(pid_t pid);//获取进程组号

函数说明：每个进程都属于一个进程组。进程组是一个或多个进程的集合，通常它们与一组作业相关联，可以接受来自同一终端的各种信号。每个进程组都有唯一的进程组ID(整数，也可以存放在pid_t类型中)。

1、Linux进程编程1/2

- ps u命令：以用户为主的格式来显示程序状况

```
zhenz@zhenz-virtual-machine:~$ ps u
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
zhenz      2352  0.0  0.4   8652  4348 pts/4    Ss   14:32   0:00 bash
zhenz      3188  0.0  0.3   9492  3160 pts/4    R+   16:09   0:00 ps u
```

- ps aux命令：查看当前所有进程信息

1、Linux进程编程1/2

创建进程——fork

函数原型　pid_t fork(void);

返回值：

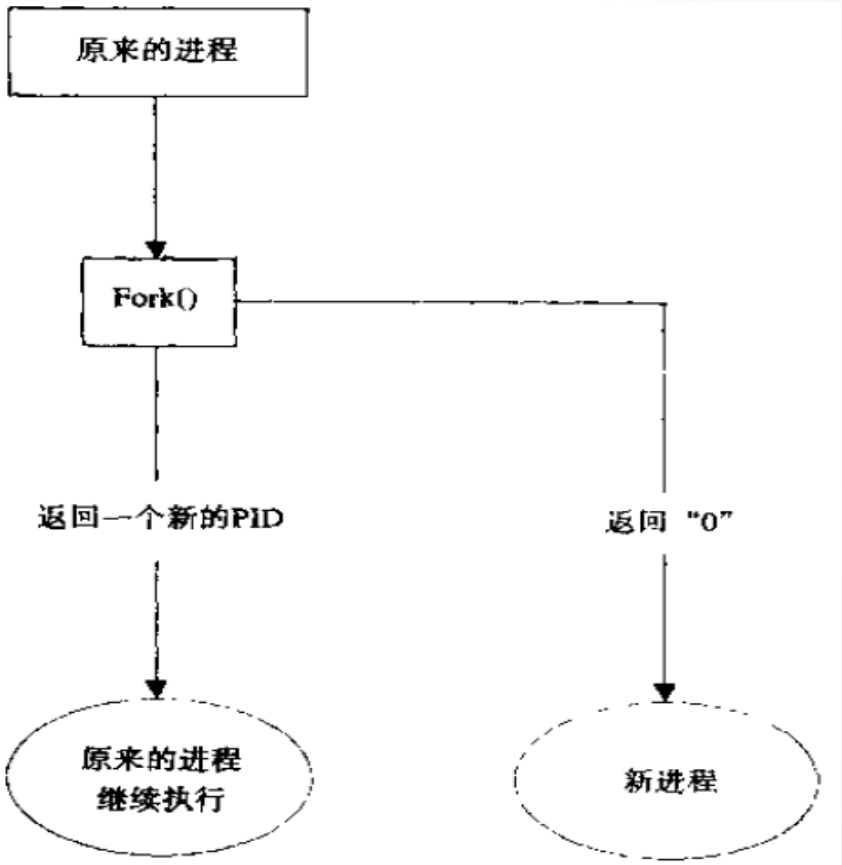
- 如果调用成功，父进程调用返回新子进程ID，新进程继续执行；在子进程返回0。
- 如果调用失败，在父进程返回-1，错误原因存于errno中。

说明：这个系统调用对父进程进行复制，在进程表里创建出一个项目，子进程与父进程几乎一样，执行的是相同的代码，只是新进程有自己的数据空间、环境和文件描述符，而对文件描述符关联的内核文件表项，采用共享方式。

1、Linux进程编程1/2

- 典型用法

```
pid_t  new_pid;
new_pid = fork();
switch(new_pid)
{
        case -1:break;
        case 0:break;
        default:break;
}
```



1、Linux进程编程1/2

创建进程——vfork

- 出现原因
  - 当子进程只是执行exec函数，则使用fork()从父进程复制到子进程的数据空间将不被使用，为了改进这种情况下效率低下的问题，出现了vfork()。
  - exec函数族的作用是将当前进程映像替换成新的程序文件。
- vfork函数特点
  - 与父进程共享数据空间
  - 执行顺序：子先父后

1、Linux进程编程1/2

替换——execl

头文件：#include<unistd.h>

函数原型：

```
int execl(const char *path, const char *arg, ...);
```

返回值：成功则函数不会返回，执行失败则直接返回-1，失败原因存于errno中。

说明：execl()用来执行参数path字符串所代表的文件路径，参数代表执行该文件时传递过去的argv(0)、argv[1]……，最后一个参数必须用空指针(NULL)作结束。

1、Linux进程编程1/2

- 举例：

```
#include<unistd.h>
  int main()
  {
     execl(“/bin/ls”, ”ls”, ”-l”,
           ”/etc/passwd”, (char *)0);
  return 0;
}
```

1、Linux进程编程1/2

- 替换——execv

头文件：#include<unistd.h>

函数原型：

```
int execv(const char *path, char *const argv[]);
```

返回值：成功则函数不会返回，执行失败则直接返回-1，失败原因存于errno中。

说明：execv()用来执行参数path字符串所代表的文件路径，与execl()不同的地方在于execv()只需两个参数，第二个参数利用数组指针来传递给执行文件。

1、Linux进程编程1/2

- 举例：

```
#include<unistd.h>
int main()
{
      char *argv[]={“ls”,”-l”,
                  ”/etc/passwd”,(char*)0 };
      execv(“/bin/ls”,argv);
      return 0;
}
```

1、Linux进程编程1/2

- 替换——execle

头文件：#include<unistd.h>

函数原型

```
int execle(const char *path, const char* arg,
        ..., char *const envp[]);
```

返回值：成功则函数不会返回，执行失败则直接返回-1，失败原因存于errno中。

说明：execle()用来执行参数path字符串所代表的文件路径，并为新程序复制最后一个参数所指示的环境变量。接下来的参数代表执行该文件时传递过去的argv(0)、argv[1]……，最后一个参数必须用空指针(NULL)作结束。

1、Linux进程编程1/2

- 举例：

```
#include<unistd.h>
int main()
{
      char * envp[]={“PATH=/bin”,0};
      execle(“/bin/ls”, ”ls”, ”-l”,
  ”/etc/passwd”, (char *)0, envp);
      return 0;
}
```

1、Linux进程编程1/2

替换——execve

头文件：#include<unistd.h>

函数原型

```
int execve(const char * path, char * const
    argv[],char * const envp[]);
```

返回值：成功则函数不会返回，执行失败则直接返回-1，失败原因存于errno中。

说明：execve()用来执行参数filename字符串所代表的文件路径，第二个参数是利用数组指针来传递给执行文件，并且需要以空指针（NULL）结束。最后一个参数则为传递给执行文件的新环境变量数组。

1、Linux进程编程1/2

- 举例：

```
#include<unistd.h>
int main()
{
        char * argv[]={“ls”,”-l”,
                        ”/etc/passwd”,(char *)0};
        char * envp[]={“PATH=/bin”,0};
        execve(“/bin/ls”,argv,envp);
        return 0;
}
```

1、Linux进程编程1/2

- 替换——execlp

头文件：#include<unistd.h>

函数原型：

```
int execlp(const char * file, const char * arg, ...);
```

返回值：成功则函数不会返回，执行失败则直接返回-1，失败原因存于errno中。

说明：execlp()会从PATH 环境变量所指的目录中查找符合参数file的文件名，找到便执行该文件，然后将第二个以后的参数当做该文件的argv[0]、argv[1]……，最后一个参数必须用空指针(NULL)作结束。

1、Linux进程编程1/2

- 举例：

```
#include<unistd.h>
int main()
{
      execlp(“ls”, ”ls”, ”-l”,
            ”/etc/passwd”, (char *)0);
      return 0;
}
```

1、Linux进程编程1/2

- 替换——execvp

头文件：#include<unistd.h>

函数原型

```
int execvp(const char *file , char * const argv[]);
```

返回值：成功则函数不会返回，执行失败则直接返回-1，失败原因存于errno中。

说明：execvp()会从PATH 环境变量所指的目录中查找符合参数file 的文件名，找到后便执行该文件，然后将第二个参数argv传给该欲执行的文件。

1、Linux进程编程1/2

- 举例：

```
#include<unistd.h>
int main()
{
        char * argv[]={“ls”, ”-l”,
                ”/etc/passwd”,0};
        execvp(“ls”,argv);
        return 0;
}
```

1、Linux进程编程1/2

- 上述六种函数比较说明
  - 查找方式：上表中前四个函数的查找方式都是完整的文件目录路径(即绝对路径)，而最后两个函数(也就是以p结尾的两个函数)可以只给出文件名，系统就会自动从环境变量“$PATH”所指出的路径中进行查找。
  - 参数传递方式：有两种方式，一种是逐个列举的方式，另一种是将所有参数整体构造成一个指针数组进行传递。（在这里，字母“l”表示逐个列举的方式，字母“v”表示将所有参数整体构造成指针数组进行传递，然后将该数组的首地址当做参数传递给它，数组中的最后一个指针要求时NULL）
  - 环境变量：exec函数族使用了系统默认的环境变量，也可以传入指定的环境变量。这里以“e”结尾的两个函数就可以在envp[]中指定当前进程所使用的环境变量替换掉该进程继承的所有环境变量。

1、Linux进程编程1/2

- 等待子进程中断或结束——wait

头文件 #include<sys/types.h>
#include<sys/wait.h>

函数原型 pid_t wait(int * status);

返回值：如果执行成功则返回子进程识别码(PID)，如果有错误发生则返回-1。失败原因存于errno中。

说明：函数会暂时停止目前进程的执行，直到有信号来到或子进程结束。如果在调用wait()时子进程已经结束，则wait()会立即返回子进程结束状态值。子进程的结束状态值会由参数status 返回，而子进程的进程识别码也会同时返回。如果不在意结束状态值，则参数status可以设成NULL。

1、Linux进程编程1/2

- 正常结束当前调用函数——on_exit

头文件：#include<stdlib.h>

函数原型：int on_exit(void (* function)(int, void*),void *arg);

返回值：如果执行成功则返回0，否则返回-1，失败原因存于errno中。

说明：on_exit（）用来设置一个程序正常结束前调用的函数。当程序通过调用exit（）或从main中返回时，参数function所指定的函数会先被调用，然后才真正由exit（）结束程序。参数arg 指针会传给参数function函数

1、Linux进程编程1/2

- 正常结束进程——exit

头文件：#include<stdlib.h>

函数原型：void exit(int status)

返回值：无

说明：正常终结目前进程的执行，并把参数status返回给父进程，而进程所有的缓冲区数据会自动写回并关闭未关闭的文件。status为0表示正常退出，不为0则表示异常退出。

1、Linux进程编程1/2

- 正常结束进程——_exit

头文件：#include<unistd.h>

函数原型：void _exit(int status)

返回值：无

说明：正常终结目前进程的执行，并把参数status返回给父进程，而进程所有的缓冲区数据不会自动写回。

THE END…

THANK YOU~

实验

```
CC = gcc
OPTIONS = -g -o
OBJECTS = main.o input.o compute.o
SOURCES = main.c input.c compute.c
HEADERS = main.h input.h compute.h
#问题一：以上部分有什么意义
power:main.c $(OBJECTS)
    $(CC) $(OPTIONS) power $(OBJECTS) -lm
#问题二：上一句命令有什么意义
```

使用math.h中声明的库函数还有一点特殊之处，gcc命令行必须加-lm选项，因为数学函数位于libm.so库文件中（这些库文件通常位于/lib目录下），-lm选项告诉编译器，我们程序中用到的数学函数要到这个库文件里找。

1、Linux进程编程1/2

```
tar -cvf $(SOURCES) $(headers) makefile > all.tar
```

#问题三：上一句命令有什么意义

- **-c**：建立压缩档案
- **-x**：解压
- **-t**：查看内容
- **-r**：向压缩归档文件末尾追加文件
- **-u**：更新原压缩包中的文件

- **-z**：有**gzip**属性的
- **-j**：有**bz2**属性的
- **-Z**：有**compress**属性的
- **-v**：显示所有过程
- **-O**：将文件解开到标准输出

-f: 使用档案名字