

第3章 进程管理

——进程退出和等待

进程退出

正常终止有5种，它们是

1. 从 main 返回（return或隐含）
2. 调用 exit
3. 调用 _exit
4. 最后一个线程从其启动例程返回
5. 最后一个线程调用 pthread_exit

异常终止有3种，它们是

6. 调用 abort
7. 接到一个信号并终止
8. 最后一个线程对取消请求做出响应

return将控制权交给主调函数
exit、_exit将控制权交给系统

_exit完成的操作

_exit	
功能	正常终止一个程序
头文件	#include<stdlib.h>
函数原型	void exit(int status);
参数	status：程序退出的状态。 0或EXIT_SUCCESS：正常退出， 非0或EXIT_FAILURE：异常退出。
返回值	无

_exit完成的操作

1. 关闭进程打开的所有文件描述符、目录描述符；
2. 清除进程使用的内存空间；

关闭与进程相关内核资源

3. 将该进程的子进程的PPID置为init进程的PID；
4. 向父进程发送SIGCHLD信号；
5. 如果父进程调用wait或waitpid来等待子进程结束，唤醒父进程，取得终止进程的status。

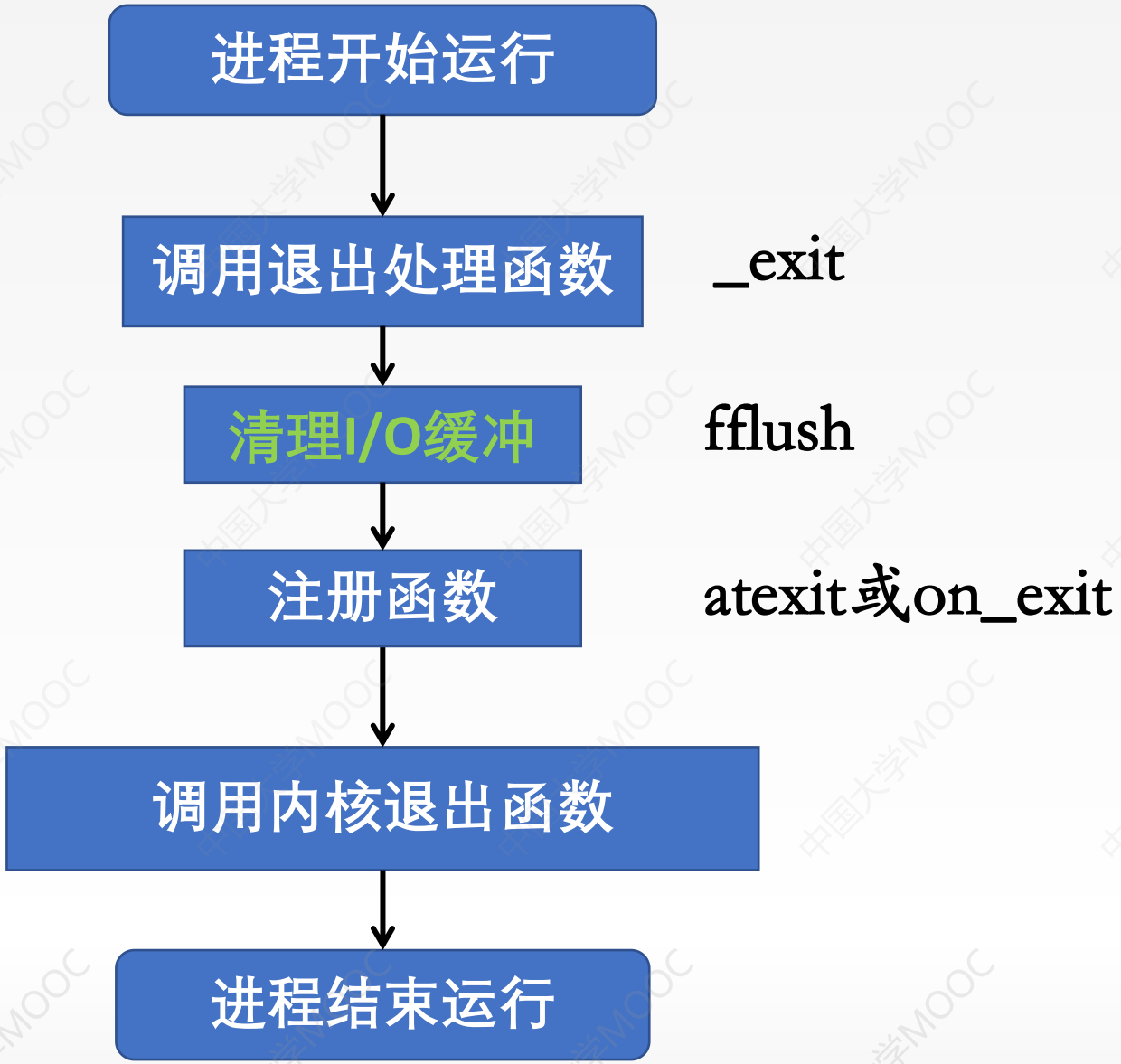
善后处理

6. 结束进程

终止进程

exit和_exit的差别

exit刷新所有的流，随后调用由atexit或on_exit注册的函数，执行当前系统定义的其他与exit相关的操作。最后调用内核退出函数。



例1：对比exit和_exit

```
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>

main()     {
    pid_t pid;
    pid=fork();
    if(pid==-1) { perror("fork");  exit(0);     }
    else if(pid==0)      {
        printf( “This is  _exit:\n");   printf("now is child"); _  exit(0); }
    else {
        printf( “This is exit:\n");     printf("now is parent");
        exit(0);  }
}
```

输出设备是行缓冲设备
_exit未冲洗缓冲区
exit冲洗了缓冲区（应用层的）

```
[huangru@xiyoulinux
chap3]$ ./exp_exit
This is exit:
now is parent This is  _exit:
```

atexit	
功能	注册一个进程正常终止前的处理函数
头文件	/usr/include/stdlib.h
函数原型	int atexit(void (*func)(int status, void *arg), void *arg);
参数	func　退出前要先调用的函数
	arg　传给func中arg的参数指针
返回值	0　成功
	-1　失败

例2：atexit注册退出前处理函数

可以定义多个善后处理函数，atexit需多次注册，exit时逆序调用。

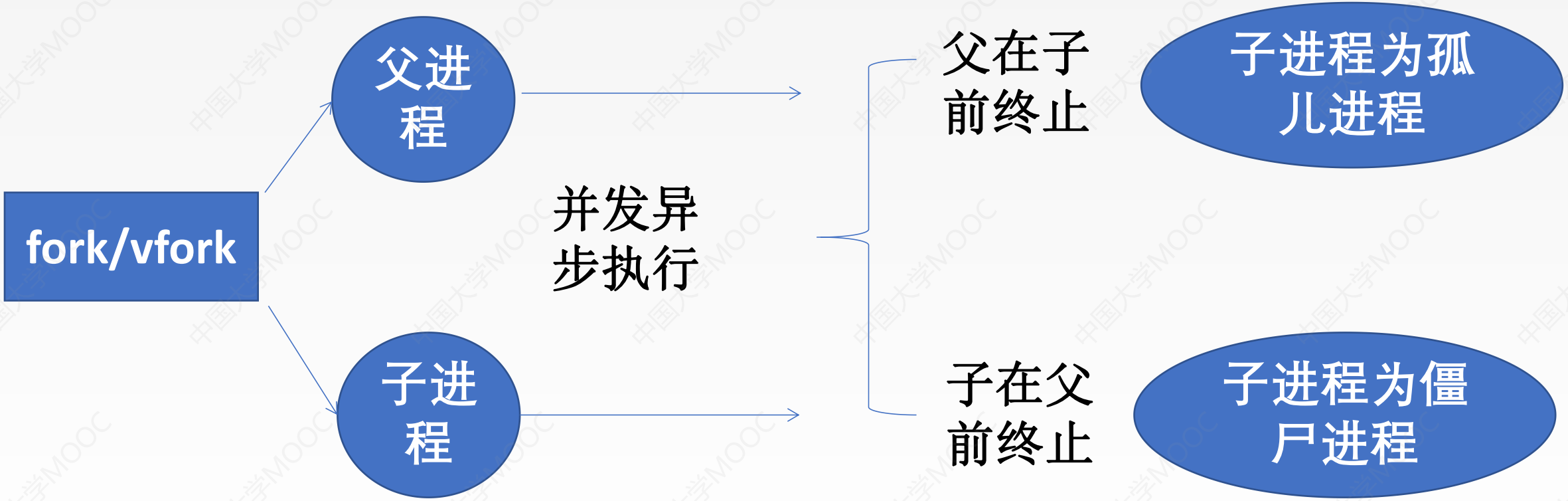
```
#include <stdio.h>
#include <stdlib.h>

void before_exit(int status,void *arg){
    printf("before exit()!\n");
    printf( "exit %d, arg=%s\n",
}
main()      {
    char * str= "on_exit";
    atexit(before_exit,(void *)str);
    exit(1111);
}
```

```
//运行结果
[huangru@xiyoulinux
chap3]$ ./exp_on_exit
before exit()!
exit_status= 1111,arg=on_exit
```

去掉之后运行结果如何?

僵尸进程



僵尸进程

1.为什么会存在？ 需要通知其父进程它是如何终止的

```
[huangru@xiyoulinux archive] ps -u |grep Z
USER   PID %CPU %MEM    VSZ    RSS TTY      STAT START   TIME COMMAND
hr     2860  0.0  0.4   8600   4920 pts/1    Ss   21:08   0:00 bash
hr     2987  0.0  0.0   2028    564 pts/1    S+   21:13   0:00 ./a.out
hr     2988  0.0  0.0      0      0 pts/1    Z+   21:13   0:00 [a.out] <defunct>
```

4.如何妥善处理？

Zombile，僵尸进程，+进程位于前台进程组。
使用命令ps -A -ostat,ppid,pid,cmd |grep -e '^[Zz]'可以定位僵尸进程以及该僵尸进程的父进程。

4. 如何妥善处理？

需要父进程为僵尸进程“收尸”，回收其PCB资源。

① 父进程执行wait或waitpid函数，等待子进程结束，回收僵尸子进程的资源。

② 将父进程中对SIGCHLD信号的处理函数设为SIG_IGN（忽略信号），让内核清理这些子进程；

③ fork两次并杀死一级子进程，令二级子进程成为孤儿进程而被init“收养”、init进程负责清理僵尸进程资源。

wait	
功能	暂停当前进程直至它的一个子进程结束，并取回该子进程结束时的状态
头文件	/usr/include/sys/wait.h
函数原型	pid_t wait(int *statloc);
参数	statloc 存放子进程终止状态的内存地址
返回值	>0 结束的子进程PID号
	-1 出错

例3：wait等待子进程结束


```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

void child()        {
        printf("PID %d is child\n",getp
        sleep(2);
        printf("child is exiting\n");
        exit(0);
}
void parent()       {
        int p_return;
        p_return=wait(NULL);
        printf("PID %d is parent, wait return from
}
```

```
main()        {
        pid_t pid;
        pid=fork();
        if(pid<0) perror("fork");
        else if(pid==0)  child();
        else   parent();
}
```

参数为NULL表示不获取子进程退出状态；如欲获取，则可改为

```
int p_return, stat;
p_return=wait(&stat);
```

waitpid

waitpid	
功能	阻塞调用进程直到特定的子进
头文件	/usr/include/sys/wait.h
函数原型	pid_t waitpid(pid_t pid,int *statloc, int options);
参数	pid 子进程PID号
	statloc 存放子进程终止状态的内存地址
	options 进一步控制waitpid的操作
返回值	>0 子进程
	-1 出错
	0 不阻塞

-1 等待任一子进程结束
>0 等待PID号为pid的子进程结束
==0 等待与调用进程同组的子进程
<-1 等待与 PID为-pid的进程同组的子进程结束

WCONTINUED: 支持作业控制
WUNTRACED: 支持作业控制
WNOHANG: 若pid所指定的子进程不是立即可用的，则不阻塞调用进程。

waitpid的说明

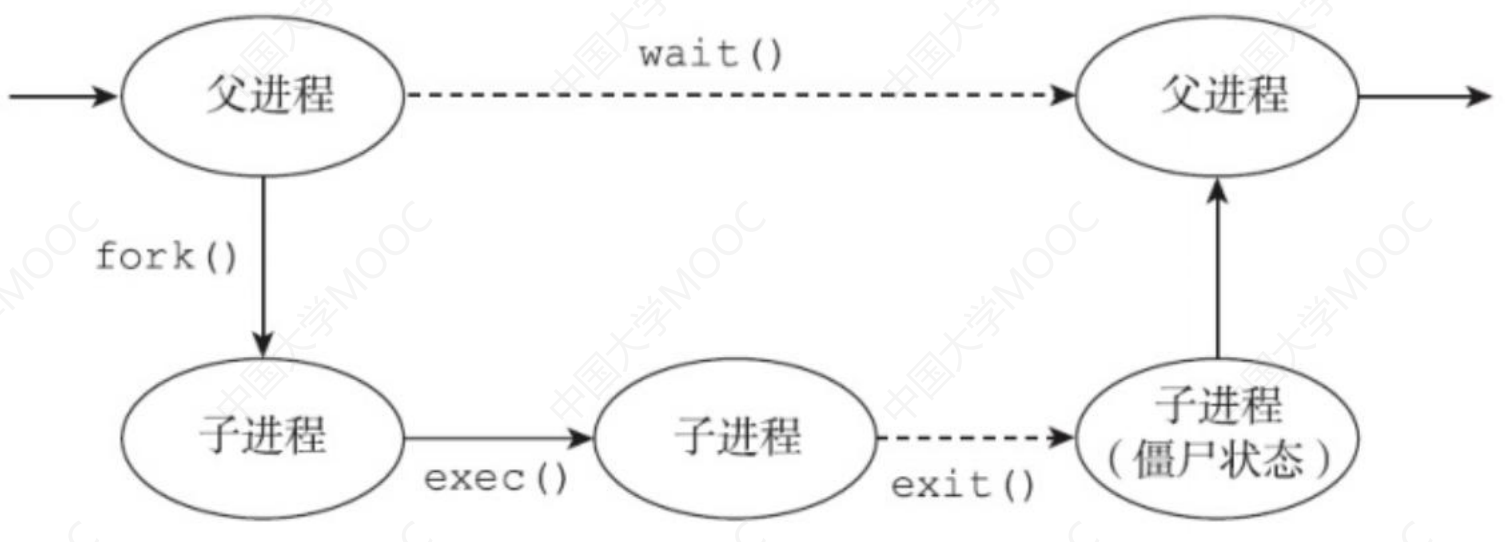
- waitpid的功能和用法与wait类似，但提供了wait函数没有的三个功能：

  1.waitpid可等待一个特定的子进程

  2.可以不阻塞调用进程

  3.支持作业控制

进程的一生



总结

- 进程退出：正常退出、异常退出
- exit或_exit：正常终止一个进程
- atexit：注册一个进程正常终止前的处理函数
- 僵尸进程：子进程已终止，部分系统资源未回收
- wait和waitpid：暂停当前进程直至一个子进程结束，可以回收僵尸进程资源。
- 进程的一生：fork、eXec、exit、wait

谢谢大家！