**实验二 进程的控制**

# 姓 名 陈正江 学 号 19001531 成绩

实验时间 2023.10.26 指导教师(签名)

**（诚信声明：本实验报告内容，均由本人亲自上机完成。 签名：\_\_\_\_\_\_\_\_\_\_\_\_\_）**

一．实验目的

1.熟悉和理解进程和进程树的概念，掌握有关进程的管理机制

2.通过进程的创建、撤销和运行加深对进程并发执行的理解

3.明确进程与程序、并行与串行执行的区别

4.掌握用C 程序实现进程控制的方法

二．实验工具与设备

已安装 Linux 操作系统的计算机。

1. 实验内容

## 1. 了解系统调用 fork( ), exec 系列函数, exit( ), waitpid( )的功能和实现过程

（1）fork()函数

调用 fork 时，系统将创建一个与当前进程相同的新进程，其与原有进程具有相同的数据、连接关系和在程序同一处执行的连续性。原进程称为父进程，新生进程称为子进程。子进程是父进程的一个拷贝，子进程获得同父进程相同的数据，但是同父进程使用不同的数据段和堆栈段。子进程被创建以后，处于可运行状态，与父进程以及系统中的其他进程平等地参与系统调度。

fork 调用将执行两次返回，即分别从父进程和子进程分别返回，即子进程一旦创建就绪，就与父进程一样被平等地调度执行。因此，从 fork 返回以后，不能确切知道执行哪一个进程。

从父进程返回时，返回值>0，为子进程的进程标识号 PID；

从子进程返回时，返回值= 0，并且返回都将执行 fork 之后的语句；

调用出错时返回值= -1，并将 errno 置为相应值。

（2）exec 系列函数

exec系统调用用新程序覆盖调用它的进程的地址空间。exec把一个新的程序装入调用进程的内存空间，来改变调用进程的执行代码。此时，系统把代码段替换成新程序的代码，废弃原有的数据段和堆栈段，并为新程序分配新的数据段和堆栈段，唯一留下的就是进程号。对系统而言，还是同一个进程，只不过运行另一个可执行程序。所以，fork()/exec()组合是典型的Linux新进程产生模式，通常先用fork()创建新进程，然后新进程通过调用exec()系列执行自己的任务。

·exec加后缀，可有多种格式：

int execl(const char \*path,const char \*arg0,const char \*arg1,…,(const char \*)0);

int execlp(const char \*file,const char \*arg0,const char \*arg1,…,(const char \*)0);

int execle(const char \*path, const char \*arg0, const char \*arg1, ... ,(const char \*)0, char \*const envp[]);

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

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

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

其中系统调用名称中的l代表长格式，v代表利用argv传参，e代表从envp传递环境变量，p代表从PATH指定路径搜索文件。

使用以上函数，需要包含头文件unistd.h（#include<unistd.h>）

（3）exit()函数

exit系统调用带有一个整数类型的参数status，可以利用这个参数传递进程结束时的状态，比如该进程是正常结束的，还是出现某种意外而结束的。

一般来说，0表示没有意外的正常结束；其他的数值表示出现了错误，进程非正常结束。在实际编程时，可以用wait系统调用接收子进程的返回值，从而针对不同的情况进行不同的处理。

（4）waitpid()

fork()创建的子进程与父进程的执行顺序是无法控制的。如果想控制必须由父进程使用函数wait()或waitpid()。

·pid\_t waitpid(pid\_t pid,int \* status,int options);

waitpid()比wait()多出两个可由用户控制的参数pid和options，为编程提供更灵活的方式。

waitpid()等待特定的子进程，waitpid()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waitpid()时子进程已经结束，则waitpid()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一块返回。如果不在意结束状态值，则参数status可以设成NULL。

·参数pid为欲等待的子进程识别码，其数值意义如下：

pid< -1 等待进程组识别码为pid绝对值的任何子进程。

pid= -1 等待任何子进程，相当于wait()。例如：pid=waitpid(-1,NULL,0);

pid= 0 等待进程组识别码与目前进程相同的任何子进程。

pid> 0 等待任何子进程识别码为pid的子进程。

·参数option可以为0 或下面的OR 组合：

WNOHANG 如果没有任何已经结束的子进程则马上返回，不予以等待。

WUNTRACED 如果子进程进入暂停执行情况则马上返回，但结束状态不予以理会。

## 2．程序设计

* 1. 编写一段程序，使用系统调用 fork( )创建两个子进程。当此程序运行时，在系统中有一个父进程和两个子进程活动。让每一个进程在屏幕上显示一个字符：父进程显示字符’a’；子进程分别显示字符’b’和’c’。观察屏幕上的显示结果，并分析原因。

1）打开终端[czj@czj-virtual-machine~]$，创建c文件，然后进入vim编辑器，编辑该文件：

- touch test1026.c

- vim test1026.c

然后按下键盘“i”键，编写 C 语言代码如下所示：

#include<stdio.h>

#include<unistd.h>

int main(){

int p1, p2;

while((p1 = fork()) ==-1);

if(p1 ==0) putchar('b');

else{

while((p2 = fork()) ==-1);

if(p2 ==0) putchar('c');

else putchar('a');

}

printf("\n");

}



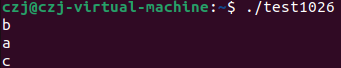
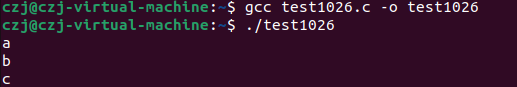
编辑完成后，按下 esc 键，然后输入:wq 保存并退出。

2）编译链接test1026.c文件，然后运行该文件：

- gcc t test1026.c -o test1026

- ./test1026

运行后可见终端依次输出abc,bac等情况，截图如下：



3）观察屏幕上的显示结果，分析依次显示“bac”的原因：

调用 fork 时系统会创建一个与父进程相同的新进程，其与父进程有相同的数据、连接关系和在程序同一处执行的连续性。子进程被创建以后，处于可运行状态，与父进程以及系统中的其他进程平等参与系统调度。fork 调用将执行两次返回，即分别从父进程和子进程分别返回，即子进程一旦创建就绪，就与父进程一样被平等地调度执行。因此，从 fork 返回以后，不能确切知道执行哪一个进程。因此，分析过程如下：

从父进程创建第一个子进程成功以后，fork从子进程1返回，返回值0赋给p1，并继续执行子进程1之后的语句，即输出字符“b”；

子进程1执行结束后，fork从父进程返回子进程2的进程标识号 p2（>0），随后父进程开始执行之后的语句，即输出字符“a”；

子进程2执行结束后，fork从父进程返回子进程1的进程标识号 p1（>0），随后父进程开始创建第二个子进程直到成功，fork从子进程2返回，返回值0赋给p2，并继续执行子进程2之后的语句，即输出字符“c”；

所以最终得到的输出结果为bac。

* 1. 编写一段程序，使用系统调用 fork( )创建一个子进程。子进程通过系统调用 exec 系列函数调用命令 ls，调用 exit( )结束。而父进程则调用 waitpid( )等待子进程结束， 并在子进程结束后显示子进程的标识符，然后正常结束。

1）创建一个新的c文件，然后进入vim编辑器，编辑该文件：

- touch test1026\_2.c

- vim test1026\_2.c

然后按下键盘“i”键，编写 C 语言代码如下所示：

#include<stdio.h>

#include<unistd.h>

#include<stdlib.h>

#include<sys/wait.h>

int main()

{

int p;

if((p = fork()) == 0){

execl("/bin/ls", "ls", (char\*)0);

exit(0); }

else {

waitpid(p, NULL, 0);

printf("child\_id = %d\n", p);

}

}



2）编译链接test1026\_2.c文件，然后运行该文件：

- gcc test1026\_2.c -o test1026\_2

- ./test1026\_2

运行后可见，终端依次输出了该目录下的所有文件，并在子进程结束后打印输出了子进程的标识符，截图如下：

# 

使用ll命令显示当前目录下所有文件，并再次运行test1026\_2.c，可见child\_id发生了变化：

# 

# 思考题

## 怎样用 C 程序实现进程的控制？当首次调用新创建进程时，其入口在哪里？

（1）用C程序实现进程的控制：

主要通过在C程序中调用各类函数，来实现对进程的控制，如：fork()函数用于创建新的进程；schedule()函数用于进程调度执行；exit()函数用于终止某一进程；exec系列函数用新程序覆盖调用它的进程的地址空间，把一个新程序装入调用进程的内存空间，来改变调用进程的执行代码；waitpid()函数用于阻塞调用进程，直到该进程的某一个子进程结束运行，以及其他类型的代码。

（2）首次调用新创建进程时：

fork系统调用创建的子进程继承了原进程的context，即fork调用成功后，子进程与父进程并发执行相同的代码。但由于子进程也继承了父进程的程序指针，所以子进程是从fork()后的语句开始执行的，这也就是新进程调用的入口。

另外，fork在子进程和父进程中的返回值是不同的。在父进程中返回子进程的PID，而在子进程中返回0。所以可以在程序中检查PID的值，使父进程和子进程执行不同的分支。

1. **系统调用 fork( )是如何创建进程的？系统调用 exit( )是如何终止一个进程的？**

（1）系统调用fork()创建进程：

内部创建的过程为：首先申请空白的进程控制块PCB，然后为新进程分配资源，随后初始化PCB，最后将新创建的进程插入到就绪队列之中。

调用 fork 时系统将创建一个与当前进程相同的新进程，其与原有进程具有相同的数据、连接关系和在程序同一处执行的连续性。子进程是父进程的一个拷贝，子进程获得同父进程相同的数据，但是同父进程使用不同的数据段和堆栈段。子进程被创建以后，处于可运行状态，与父进程以及系统中的其他进程平等地参与系统调度。

（2）系统调用exit()终止进程：

内部终止的过程为：执行到exit系统调用，该进程停止剩下的所有操作，清除包括PCB在内的各种数据结构，并终止本进程的运行。在进程执行过程中，可通过给进程发送信号强行终止运行的进程，也可当完成任务后进程执行函数自动退出而消亡。

exit 系统调用带有一个整数类型的参数status，可以利用这个参数传递进程结束时的状态，比如该进程是正常结束的，还是出现某种意外而结束的。一般来说，0表示没有意外的正常结束；其他的数值表示出现了错误，进程非正常结束。在实际编程时，可以用wait系统调用接收子进程的返回值，从而针对不同的情况进行不同的处理。

1. **系统调用 exec 系列函数是如何更换进程的可执行代码的？**

系统调用exec，用新程序覆盖调用它的进程的地址空间，该系列函数将一个新的程序装入调用进程的内存空间，来改变调用进程的执行代码。此时，系统把代码段替换成新程序的代码，废弃原有的数据段和堆栈段，并为新程序分配新的数据段和堆栈段，唯一留下的就是进程号。对系统而言，还是同一个进程，只不过运行另一个可执行程序。

exec加后缀，可有多种格式：

int execl(const char \*path,const char \*arg0,const char \*arg1,…,(const char \*)0);

int execlp(const char \*file,const char \*arg0,const char \*arg1,…,(const char \*)0);

int execle(const char \*path, const char \*arg0, const char \*arg1, ... ,(const char \*)0, char \*const envp[]);

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

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

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

其中系统调用名称中的l代表长格式，v代表利用argv传参，e代表从envp传递环境变量，p代表从PATH指定路径搜索文件。

使用以上函数需包含头文件unistd.h（#include<unistd.h>）

以上六个函数中，execve是系统调用函数，其它5个函数都是在用户空间中实现的，实际最终也是调用execve实现最终功能。这些函数的共同特点是：在运行成功后没有返回值，因为把原来的程序置换了；运行失败时返回值为-1，失败原因存储于errno 中。