Linux 进程控制

一、Fork() 创建新进程的系统调用

#include<sys/types.h> /\* 提供类型pid\_t的定义 \*/  
#include<unistd.h> /\* 提供函数的定义 \*/  
pid\_t fork(void);

（1）Fork的返回值有多种，其用法如下

Pid=fork();

If(pid<0)

error

Else if(pid==0)

//child process

Else

//father process

（2）Fork示例代码

#include “stdio.h”

#include<sys/types.h> /\* 提供类型pid\_t的定义 \*/  
#include<unistd.h> /\* 提供函数的定义 \*/

main()

{

Pid\_t id;

printf("hello Linux!\n");

id=fork();

if(id>0)

printf(“parent\n”);

else if(id==0)

printf(“child\n”);

else

printf("fork error\n");

}

------------------------------------------------------------------------

/\* fork\_test.c \*/  
#include<sys/types.h>  
#inlcude<unistd.h>  
main()  
{  
pid\_t pid;  
  
/\*此时仅有一个进程\*/  
pid=fork();  
/\*此时已经有两个进程在同时运行\*/  
if(pid<0)  
 printf("error in fork!");  
else if(pid==0)  
 printf("I am the child process, my process ID is %d\n",getpid());  
else  
 printf("I am the parent process, my process ID is %d\n",getpid());  
}

$gcc fork\_test.c -o fork\_test  
$./fork\_test  
I am the parent process, my process ID is 1991  
I am the child process, my process ID is 1992

---------------------------------------------------

#include <sys/types.h>

#include <stdio.h>

#include <unistd.h>

int value=5; // 全局

int main(){

pid\_t pid;

pid=fork();

if(pid==0){

value += 15;

printf("Child: value = %d\n",value);

}

else if(pid>0){

wait(NULL);

printf("PARNET: value = %d\n",value);

exit(0); //父进程终止

}

}

（3）看这个程序的时候，头脑中必须首先了解一个概念：在语句pid=fork()之前，只有一个进程在执行这段代码，但在这条语句之后，就变成两个进程在执行了，这两个进程的代码部分完全相同，将要执行的下一条语句都是if(pid==0)……。两 个进程中，原先就存在的那个被称作“父进程”，新出现的那个被称作“子进程”。父子进程的区别除了进程标志符（process ID）不同外，变量pid的值也不相同，pid存放的是fork的返回值。fork调用的一个奇妙之处就是它仅仅被调用一次，却能够返回两次，它可能有三 种不同的返回值：

（4）在父进程中，fork返回新创建子进程的进程ID；在子进程中，fork返回0；如果出现错误，fork返回一个负值；

（5）fork出错可能有两种原因：（1）当前的进程数已经达到了系统规定的上限，这时errno的值被设置为EAGAIN。（2）系统内存不足，这时errno的值被设置为ENOMEM。

二、Wait() -- Linux的阻塞原语

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

1. wait示例代码

int main()

{int i;

if(fork()>0)

{//wait(NULL);

for(i=0;i<5;i++)

{ printf("AAAAAAAAAAAAAAAA\n");

sleep(2);

}

}

else

for(i=0;i<5;i++)

{

printf("BBBBBBBBBBBBBBBBB\n");

sleep(2);

}

return 1;

}

-----------------------------------

/\* wait1.c \*/  
#include <sys/types.h>  
#include <sys/wait.h>  
#include <unistd.h>  
#include <stdlib.h>  
main()  
{  
pid\_t pc,pr;  
pc=fork();  
if(pc<0) /\* 如果出错 \*/  
 printf("error ocurred!\n");  
else if(pc==0){ /\* 如果是子进程 \*/   
 printf("This is child process with pid of %d\n",getpid());  
 sleep(10); /\* 睡眠10秒钟 \*/  
}  
else{ /\* 如果是父进程 \*/  
 pr=wait(NULL); /\* 在这里等待 \*/  
 printf("I catched a child process with pid of %d\n",pr);  
}   
exit(0);  
}

三、exec族函数-加载一个新的可执行程序到进程，书P93

exec() is a family of functions, there are many version of exec(), such as execlp(),execvp()....

#include <stdio.h>

#include <sys/types.h>

#include <unistd.h>

int main()

{

pid\_t id=0；

char \*parameter[3];

parameter[0] = "ls";

parameter[1] = "-a";

parameter[2] = 0 ;

id=fork();

if(id==0)

{

execvp( "ls" , parameter );

printf("cannot execute the command!\n");

}

else

{

wait(NULL);

printf("\* \* \* ls is done. bye\n");

printf("\* \* \* BBBBBBBBBB\*\*\*\*\*\n");

}

return 1;

}

-------------------------------------------------------

shell0.c

//shell0.c

#include “stdio.h”

#include “string.h”

int main()

{ char cmd[100];

char \*argv[2];

while(1)

{ printf(“%%”);

scanf(“%s”,cmd);

if(strcmp(cmd,“exit”)==0)

break;

argv[0]=cmd;

argv[1]=0;

if(fork()>0)

{

wait(NULL);

continue;

}

else

{

execvp(cmd,argv);

printf("cannot excute the command!\n");

exit(1);

}

}

return 1;

}

---------------------------------------

四．进程通信-管道（父子进程交换信息）

#include <stdio.h>

#include <linux/limits.h>

#include <unistd.h>

int main(void)

{

printf("PIPE\_BUF %d(bytes)\n",PIPE\_BUF);

int x, fd[2];

char buf1[40],buf2[40];

pipe(fd);

if((x=fork())==-1)

{

perror("fork err");

exit(1);

}

if(x==0)

{

close(fd[0]);

sprintf(buf1,"hello");

write(fd[1],buf1,40);

exit(0);

}

else

{

close(fd[1]);

wait(NULL);

read(fd[0],buf2,40);

printf("%s\n",buf2);

}

}