**实验3 Linux进程控制与线程创建**

实验指导

1. Linux进程控制
2. exec函数

在Linux系统中，并不存在一个exec()的函数形式，exec()指的是一组函数，共有6个分别是：execl、execlp、execle、execv、execvp、execve，基本功能相同，只是以不同的方式来给出参数。

1. 函数定义

头文件：

#include <unistd.h>

系统调用格式：

int execl(char \*path, char \*arg0,char \*arg1,...,char \*argn,NULL);

int execlp(char \*file, char \*arg0,char \*arg1,..., char \*argn,NULL);

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

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

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

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

返回值：执行成功不会返回，执行失败则直接返回-1，失败原因存于errno 中。

execl()函数

函数原型：int execl(char \*path, char \*arg0,char \*arg1,...,char \*argn,NULL);

参数说明：path给出了被执行程序名字，必须是可执行程序名，可以带路径，不带路径表示在当前目录下；后续的参数arg0, arg1以及用省略号表示的其他参数一起组成了该程序执行时的参数表。按照Linux的贯例，参数表的第一项是不带路径的程序文件名。被调用的程序可以访问这个参数表，它们相当于shell下的命令行参数。由于参数的个数是任意的，所以必须用一个NULL指针来标记参数表的结尾。

1. 使用举例

#include <stdio.h>

#include <unistd.h>

main()

{

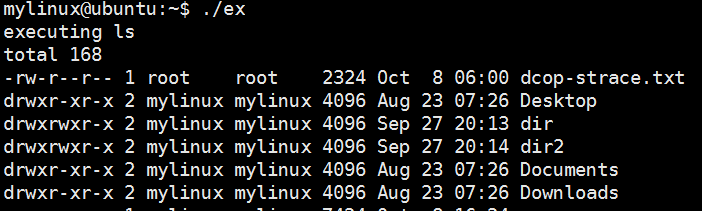
printf("executing ls\n");

execl("/bin/ls","ls","-l",NULL);

perror("execl failed to run ls"); /\*如果execl返回，说明其调用失败。\*/

}

运行结果：



进程执行printf语句后，调用execl改变了进程的程序，执行ps程序。

1. exit函数

在Linux系统中，由fork()系统调用创建子进程，由wait()系统调用而等待。可通过给进程发送信号强行终止运行进程，也可当进程完成任务后自动退出而消亡。使进程自动退出系统的系统调用是exit()或\_exit()。

1. 函数定义

exit()

终止调用进程的执行。在调用exit()之前要检查文件的打开情况，把文件缓冲区中的内容写回文件，即“清理I/O缓冲”。

头文件：

#include <stdlib.h>

系统调用格式：

void exit(status)

参数定义：

int status;

参数说明：status是返回给父进程的一个整数，用来传递进程结束时的状态，包括进程正常结束或因出现某种意外而结束。

\_exit()

终止调用进程的执行，返回进程状态，但不关闭文件，不清除输出缓存。

头文件：

#include <unistd.h>

系统调用格式：

void \_exit(status)

参数定义：

int status;

1. wait函数

父进程等待子进程运行结束。进程通过wait()与其子进程同步。当子进程结束时，将发送软中断SIGCHLD信号到父进程，子进程进入僵死状态，等待父进程调用wait()取出子进程的状态。进程一旦调用wait()，如果子进程没有完成，父进程将立即阻塞自己，一直等待。wait()自动分析当前进程是否有某个子进程已经退出。如果找到一个已经僵死的子进程，wait()将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()将一直阻塞，直到有一个僵死进程出现为止。

1. 函数定义

头文件：

#include<sys/types.h>

# include<sys/wait.h>

系统调用格式：

int wait(status)

参数定义：

int \*status;

参数说明：status用来保存有关子进程退出时的一些状态，它是一个指向int类型的指针。如果不关心子进程是如何结束的，而只想把该僵死进程销毁（事实上绝大多数是这种情况），则可以把参数status设置为NULL或0。其调用形式为：wait(0)。也可以定义整形变量记录wait()的返回值，调用形式如下所示：

pid=wait(NULL);

返回值：如果成功，wait会返回被收集子进程的ID，如果调用进程没有子进程，调用就会失败，此时wait返回-1，同时errno被置为ECHILD。

1. Linux线程创建

Linux系统下的多线程遵循POSIX线程接口，称为pthread。编写Linux下的多线程程序，需要使用头文件pthread.h

1. 线程创建

系统调用格式：

int pthread\_create(pthread\_t\* pthread, const pthread\_attr\_t\*, void\* (\*start\_routine )(void\*), void\* arg )

参数定义：

pthread\_t 类型变量的thread，线程的标识

pthread\_attr 线程属性设置，常设为NULL，默认线程属性

start\_routine 线程函数指针，是线程要执行的代码，参数为void(\*)

arg 传递给start\_routine的参数

返回值：

成功返回0，错误号表示失败

1. 等待线程终止

系统调用格式：

int pthread\_join(pthread\_t thread, void \*\*ptr)

参数定义：

thread为pthread\_create创建线程的标识

ptr 为返回码，常设为 NULL

返回值：

成功返回0，否则失败

1. 获取线程id

系统调用格式：

pthread\_self(void)

返回值：

pthread\_t

1. 程序举例

#include <stdio.h>

#include <pthread.h>

#include <unistd.h>

void \*run(void \*ptr){

pid\_t pid;pthread\_t tid;

pid = getpid(); //获取进程id

tid = pthread\_self(); //获取线程id

printf("In thread,process id:%lu,pthread id:%lu\n",(unsigned long)pid,(unsigned long)tid);

return 0;

}

int main(){

pthread\_t tid[3],mtid;

pid\_t pid;int i;

int ret=0;

for (i=0;i<3;i++){

ret=pthread\_create(&tid[i],NULL,run,NULL);

if(ret) {

printf("create thread failed ");

return 0;

}

}

for(i=0;i<3;i++) pthread\_join(tid[i],NULL);

pid = getpid(); //获取进程id

mtid = pthread\_self(); //获取线程id

printf("In main process id:%lu,pthread id:%lu\n",(unsigned long)pid,(unsigned long)tid);

return 0;

}