Linux下多进程编程（C语言）

一、进程简介

1、进程是程序的执行。程序是静态的，进程是动态的。

2、进程在内存中有三部分组成：数据段、堆栈段和代码段。

代码段：就是存放程序代码的数据，如果有数个进程运行同一个一个程序，那么它们就可以使用同一个代码段（代码段是可以共享的）；

堆栈段：存放的是子程序的返回地址、参数以及程序的局部变量，主要是保存进程的执行的环境，这里用到了栈先进后出的特性，可以看做具有记忆上一次执行的环境。

数据段：存放程序的全局变量 、常数

//动态数据分配的数据空间

系统如果同时运行数个相同的程序，它们之间就不能使用同一个堆栈段和数据段，但是可以使用同一个代码段(代码段是可以共享的)。

二、所要用到的函数

1 、fork() 功能：产生新的进程

1.1 依据返回值可以判断产生的是父进程还是子进程：

（1）== 0 子线程 ；（2）>0 父进程（此时返回值为子进程的进程号） ；（3） < 0 创建进程失败

1.2 父进程的进程号要比子进程的进程号小，因为父进程要比子线程早产生

1.3 进程的三个组成部分中代码段是父子进程共享的（同一个程序中），系统会复制数据段和堆栈段给子进程，父子进程会独立运行，互不影响。

1.4 fork函数执行一次，产生一个新进程，复制数据段、堆栈段，但是如果一个大程序在运行的时候，fork的时候复制数据段和堆栈段会开销很大。系统的解决的方法：一般CPU都是以页为单位分配空间的，无论是数据段还是堆栈段都是由很多页组成的，当实际执行fork的时候，物理空间上，两个进程的数据段和堆栈段还是共享的，逻辑上是分开的，只有当一个进程写某个数据的时候，这时两个进程之间的数据才有了区别，系统会将有区别的页从物理上 分开，而其他的不动，从而使系统在空间上的开销达到最小。（也就是说只有当有进程间的堆栈段和数据段是有区别的时候才会将二者分开，否则在物理空间上，二者依然是共享的）

2、getpid() 获得当前进程的进程ID

3、getppid() 获得当前进程的父进程的进程ID

4、exec类函数

在linux环境下，一个进程启动一个程序的执行可以使用exec类的函数来完成，这个exec类的函数包括：

4.1 int execv (const char \*filename, char \*const argv[])

4.2 int execl (const char \*filename, const char \*arg0, . . . )

4.3 int execve (const char \*filename, char \*const argv[], char \*const env[])

4.4 int execle (const char \*filename, const char \*arg0, char \*const env[], . . . )

4.5 int execvp (const char \*filename, char \*const argv[])

4.6 int execlp (const char \*filename, const char \*arg0, . . . )

一个进程一旦调用了exec类的函数，它本身就死了，它的代码段会被替换成新的程序代码，并且废弃原有的数据段和堆栈段，重新分配新的数据段和堆栈段，唯一留下的就是进程号，对于系统而言，还是同一个进程，不过这个进程已经是另一个程序了。如果一个程序想启动另一个程序的执行但是继续运行自己的话，需要结合fork函数和exec函数一块使用。

5、wait函数保证父进程运行完后等待子进程退出才退出，否则父进程结束了，子进程还在运行。

三、进程间通信

linux系统中实现进程间通信的方式中常用的有：

（1）管道（2） 消息队列（3） 共享内存（4） 信号量 （5）套接口等。

1、管道

管道是进程间通信的最古老的方式，分两种：

1.1无名管道（父进程和子进程间通信使用）。

1.2有名管道（同一机器上的任意两个进程间的通信使用）

无名管道

无名管道由函数pipe()来创建 int pipe(int filedis[2]) 参数filedis返回两个文件的描述符号：filedis[0]为读而打开（只用来从里面读数据，不可写） ；filedis[1]为写而打开（只用来向里面写数据，不可读）。 filedis[1]的输出是filedis[0]的输入。

有名管道(有名管道就是有名字的管道)：

linux系统下的有名管道由两方式创建：命令行方式mknod和函数mkfifo

方式一： mkfifo("myfifo","rw")

方式二： mknod myfifo p（使用file myfifo可以看出它是一个命名管道）

上述两种方式都在当前目录下生成了一个名为fifo的有名管道。生成了有名管道后就

可以使用一般的文件io函数如open、 close、 read和writre对其进行操作了

int mkfifo (const char \*filename, mode t mode) sys/stat.h

2、共享内存

共享内存是运行在同一机器上的进程间的通信最快的方式，因为数据直接存在内存不需要在不同的进程间进行复制，通常由一个进程创建一个共享内存区，其余进程对这块内存进行读写。

得到共享内存有两种方式：（1）映射/dev/mem设备（2）映射内存映像文件

映射/dev/mem不给系统带来额外的开销，但是在实际中应用不常用，因为它控制存取的是实际的物理内存，在linux系统下，只有通过限制linux系统存取的内存才可以做到

通常常用的方式是使用shmXXX函数族来实现共享内存进行存储

3、所要用到的函数

int shmget(key\_t key,int size,int flag)

获得一个共享存储标识符，也就是请求分配size大小的内存用作共享内存 key\_t关键字用来存储引用标识符，是个长整型数据，在头文件sys/types.h中定义。

上述函数执行完毕，共享内存就创建完毕了。

其余进程使用 shmat()函数就可以连接到自身的地址空间中了

void\* shmat(int shmid,void\* addr,int flag)

shmid为shmget函数返回的共享存储标识符

addr和flag决定了以什么方式来确定连接地址

函数的返回值就是该进程数据段所连接的实际地址，也就是共享内存的地址，进程可以对其进行读写操作

使用共享存储来实现进程间的通信的注意点是对数据存取的同步，必须确保一个进程去读取数据时，它所想要的数据已经写好了，通常信号量可以被用来实现对共享存储数据存取的同步

4、套接字

套接字是实现linux操作系统和其他操作系统之间的进程间通信的方式之一，www ftp telnet等服务都是基于套接字的。

除了适用于异地的计算机进程间通信外，套接字同样适用于本地同一计算机内部的进程间通信。

#include <unistd.h>

#include <stdio.h>

int main()

{

pid\_t fpid;

int count = 0 ;

printf("start \n");

fpid = fork();

printf("fork \n");

if(fpid<0)

{

printf("error in fork! \n");

}

else if (fpid==0)

{

printf("child process %d \n",getpid());

count ++;

}

else

{

printf("parent process %d \n",getpid());

count ++;

}

printf("统计结果是:%d \n",count);

return 0 ;

}

输出结果

start

fork

parent process 1579

统计结果是:1

root@localhost:~# fork

child process 1580

统计结果是:1

总结： fork函数 会创建一个新进程，pid为0时是子进程 ,pid 不为零时是父进程。

子进程执行是从fork开始执行的。fork之前的，不用再次执行。

参考答案：

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

int main(){

int pid;

if((pid=fork())==0)

system("./bb.out");

return 0;

}

#include<time.h>

#include<stdio.h>

#include<stdlib.h>

#include<pthread.h>

int main(){

time\_t now;

struct tm \*timenow;

while(1){

time(&now);

timenow=localtime(&now);

printf("Those output come from child,%s\n",asctime(timenow));

}

return 0;

}

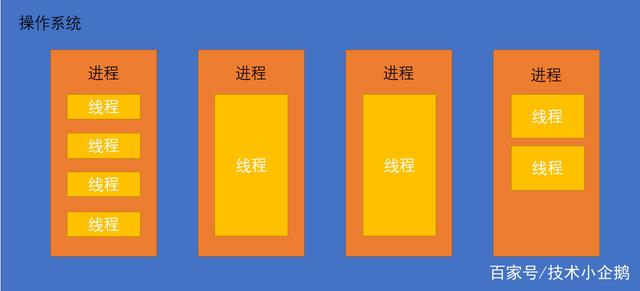
**探讨C语言系统编程中线程的原理以及实现**

线程的概念

我们今天来聊一聊线程，之前有写过一篇关于进程的文章，今天我们聊的线程，与进程差不多，我们首先要知道的一件事情是一个进程里面可以包括多个线程，不能反过来，我们之前了解到的不同进程可以做不一样的事情，我们举一个简单易懂的例子：我们的电脑上面有很多的应用程序，微信、QQ，酷狗音乐等等，我们运行一个应用程序就有一个进程，那什么是线程呢？比如其中一个，我们打开了酷狗音乐，打开音乐听着，这时候你觉得歌好听你就去看评论，这时候你想一下：播放音乐，看评论。是两件事情，但是都可以同时进行，那这里我们说酷狗音乐是一个进程，听音乐和看评论是它下面的两个线程。

你要记住单进程里面执行多个任务，这就是线程。

线程原理图



线程的标识

我们之前说的进程有一个唯一的标识，我们今天说的线程也是一样的我们的进程标识是pit\_t数据类型来表示，我们的线程是用pthread\_t数据类型来表示。获取自己的线程标识我们用函数pthread\_self();

线程创建

在外面没有创建新的线程的时候，运行的时候他和普通进程没没有什么区别。我们创建出来的线程是共享全部变量在子线程改变全局变量那么主线程也会被改变，这一点一会我们可以验证一下。

创建新线程可以使用函数pthread\_create()来创建，我们来看看看书的主要参数：

参数1：pthread\_t \*tid

传出参数，表示新创建的子线程ID。

参数2：const pthread\_attr\_t \*attr

线程属性，传入NULL表示使用默认属性

参数3：void \*(\*start\_rountn)(void \*)

子线程的回调函数，创建成功，pthread\_create函数返回时，该函数会被自动调用；

参数4：void \*arg

给回调函数传的参数，如果函数不需要传参传NULL.

代码实现

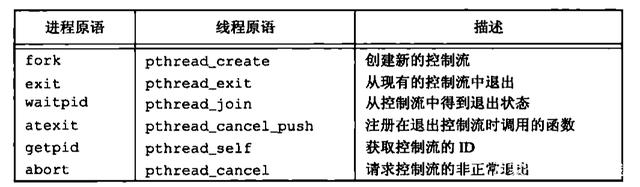
1 #include<stdio.h>  
2 #include<stdlib.h>  
3 #include<string.h>  
4 #include<unistd.h>  
5 #include<pthread.h>  
6  
7  
8 int a=10;  
9 void \* fun1(void \* arg)  
10 {  
11 pthread\_detach(pthread\_self());  
12 a=100;  
13 printf("子线程 进程号:%d, 线程号：%lu\n",getpid(),pthread\_self());  
14  
15 }  
16  
17 int main ()  
18 {  
19 pthread\_t tid;  
20 printf("主线程 进程号:%d, 线程号：%lu\n",getpid(),pthread\_self());  
21  
22 //创建线程  
23  
24 printf("a=%d\n",a);  
25 int ret =pthread\_create(&tid,NULL,fun1,NULL);  
26 if(ret==0)  
27 {  
28 printf("线程创建成功！\n");  
29 }  
30 sleep(2);  
31 printf("a=%d\n",a);  
32  
33 return 0;  
34 }

我们看一下下面程序执行的结果，我们可以看出来两个进程号是一样的，但是线程号却不一样。

另外一个值得我们注意的是我们的全局变量a我们在定义的时候值是10，然后我们在线程里面把值改为了100，最后打印出来的结果验证了我们前面说的线程是共享全局变量的。



线程与进程



int pthread\_detach(pthread\_t thread); 设置线程分离，使用detach分离线程，分离后的线程会自动回收。

参数： thread 是要分离的线程ID。

成功返回0，失败返回errno。

线程退出

pthread\_exit 退出

void pthread\_exit(void \*retval); 退出当前线程。

retval：退出值。 无退出值时，NULL

exit(); 退出当前进程。

return: 返回到调用者那里去。pthread\_exit(): 退出当前线程。

结语

关于线程的使用不仅于此，在线程编程中还有很多重要的知识，后期再更新对线程更深入地探讨。

请研读并运行下列程序：

#include<pthread.h>

#include<stdio.h>

void \*workThreadEntry(void \*args){

char\*str = (char\*)args;

printf("threadId:%lu,argv:%s\n",pthread\_self(),str);

}

int main(int argc,char \*agrv[]){

pthread\_t thread\_id;

char\* str = "hello world";

pthread\_create(&thread\_id,NULL,workThreadEntry,str);

printf("threadId=%lu\n",pthread\_self());

pthread\_join(thread\_id,NULL);

}

参考答案

#include<stdio.h>

#include<stdlib.h>

#include<pthread.h>

pthread\_t ntid;

int shared\_var=0;

void mainloop(){

while(1){

printf("%d\n",shared\_var++);

}

}

void \*tloop(viod \*arg){

//tloop前的\*,表示指向函数的指针

while(1)

printf("%d\n",shared\_var--);

}

int main(void){

if(pthread\_creat(&ntid,NULL,tloop,NULL)!=0)

pritf("can't creat thread:%s\n");

//注意pthread\_creat的第三个参数

//子线程的回调函数，创建成功，

//pthread\_create函数返回时，该函数会被自动调用

mainloop();

return 0；

}