

exit和return的区别

原创

firefly_2002

于 2012-09-09 16:29:06 发布

23720

★ 收藏 44

版权

分类专栏: C/C++基础

文章标签:

linux

语言

signal

os

多线程

gcc



C/C++基础 专栏收录该内容

6 订阅

76 篇文章

订阅专栏

1. exit用于结束正在运行的整个程序，它将参数返回给OS，把控制权交给操作系统；而return 是退出当前函数，返回函数值，把控制权交给调用函数。
2. exit是系统调用级别，它表示一个进程的结束；而return 是语言级别的，它表示调用堆栈 的返回。
3. 在main函数结束时，会隐式地调用exit函数，所以一般程序执行到main()结尾时，则结束主进程。exit将删除进程使用的 内存 空间，同时把错误信息返回给父进程。
4. void exit(int status); 一般status为0，表示正常退出，非0表示非正常退出。

1、exit函数和return函数的主要区别是：

1) exit用于在程序运行的过程中随时结束程序，其参数是返回给OS的。也可以这么讲：exit函数是退出应用程序,并将应用程序的一个状态返回给OS，这个状态标识了应用程序的一些运行信息。

main函数结束时也会隐式地调用exit函数，exit函数运行时首先会执行由atexit()函数登记的函数，然后会做一些自身的清理工作，同时刷新所有输出流、关闭所有打开的流并且关闭通过标准I/O函数tmpfile()创建的临时文件。

exit是系统调用级别的，它表示了一个进程的结束，它将删除进程使用的内存空间，同时把错误信息返回父进程。通常情况：exit(0)表示程序正常，exit(1)和exit(-1)表示程序异常退出，exit(2)表示系统找不到指定的文件。在整个程序中，只要调用exit就结束。

2) return是语言级别的，它表示了调用堆栈的返回；return是返回函数值并退出函数，通常0为正常退出，非0为非正常退出，请注意，如果是在主函数main, 自然也就结束当前进程了（也就是说，在main()里面，你可以用return n，也能够直接用exit(n)来做），如果不是在main函数中，那就是退回上一层调用。在多个进程时，如果有时要检测上个进程是否正常退出，就要用到上个进程的返回值。

2、进程环境与进程控制

exit(int n)其实就是直接退出程序，因为默认的标准程序入口为 int main(int argc, char** argv)，返回值是int型的。一般在shell下面，运行一个程序，然后使用命令echo \$?就能得到该程序的返回值，也就是退出值。

理论上exit可以返回小于256的任何整数，返回的不同数值主要是给调用者作不同处理的。

对于单独的进程exit的返回值是返回给父进程的。父进程里面调用waitpid()等函数



firefly_2002

关注

25



理。根据相应的返回值来让调用者作出相应的处理。

总的说来, exit () 就是当前进程把其控制权返回给调用该子程序的主程序, 括号里的是返回值,告诉调用程序该程序的运行状态。

1) 进程的开始:

C程序是从main函数开始执行, 原型如下: int main(int argc, char *argv[]); 通常main的返回值是int型, 正确返回0。

2) 进程终止:

C程序的终止分为两种: 正常终止和异常终止。正常终止分为: return, exit, _exit, _Exit, pthread_exit。异常中指分为: abort, SIGNAL, 线程响应取消。

主要说一下正常终止的前4种, 即exit系列函数。

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

```
void exit(int status);
```

```
void _Exit(int status);
```

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

```
void _exit(int status);
```

以上3个函数的区别是: exit()(或return 0)会调用终止处理程序和用户空间的标准I/O清理程序(如fclose), _exit和_Exit不调用而直接由内核接管进行清理。因此, 在main函数中exit(0)等价于return 0。

3) atexit终止处理程序:

ISO C规定, 一个进程最多可登记32个终止处理函数, 这些函数由exit按登记相反的顺序自动调用。如果同一函数登记多次, 也会被调用多次。

原型如下:

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

```
int atexit(void (*func)(void));
```

其中参数是一个函数指针, 指向终止处理函数, 该函数无参无返回值。atexit函数本身成功调用后返回0。

以下面的程序为例:

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



firefly_2002

关注

25



```
static void myexit1()
{
    printf("first exit handler\n");
}

static void myexit2()
{
    printf("second exit handler\n");
}

int main()
{
    atexit(my_exit2);

    atexit(my_exit1);

    atexit(my_exit1);

    printf("main is done\n");

    return 0;
}
```

运行结果:

\$./a.out

main is done

first exit handler

first exit handler

second exit handler

注意上面的结果，可以发现这些函数由exit按登记相反的顺序自动调用（先myexit1后myexit2）。如果同一函数登记多次，也会被调用多次（如这里的myexit1）。而这些处理函数都是在程序退出的时候利用atexit函数调用了这些处理函数。但是如果用_exit()退出程序，则它不关闭任何文件，不清除任何缓冲器、也不调用任何终止函数！

exit函数在头文件stdlib.h中。

exit (0)：正常运行程序并退出程序；

exit (1)：非正常运行导致退出程序；

return ()：返回函数，若在main主函数中，则返回0。



firefly_2002

关注

25



详细说：

1. return返回函数值，是**关键字**；exit是一个**函数**。
2. return是语言级别的，它表示了调用堆栈的返回；而exit是系统调用级别的，它表示了一个进程的结束。
3. return是函数的退出(返回)；exit是进程的退出。
4. **return是C语言提供的，exit是操作系统提供的**（或者函数库中给出的）。
5. return用于结束一个函数的执行，将函数的执行信息传出个其他调用函数使用；exit函数是退出应用程序，删除进程使用的内存空间，并将应用程序的一个状态返回给OS，这个状态标识了应用程序的一些运行信息，这个信息和机器和操作系统有关，一般是 0 为正常退出，非0 为非正常退出。
6. 非主函数中调用return和exit效果很明显，但是在main函数中调用return和exit的现象就很模糊，多数情况下现象都是一致的。

下面是几个例子：

1.

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

int main( void )
{
    pid_t pid;
    int count=0;

    pid=vfork();
    if (pid==0)
    {
        printf ( "child: count=%d\n" ,count);
        printf ( "child: getpid=%d\n" ,getpid());
        count=1;
        printf ( "child: count=%d\n" ,count);
        // return 0;//会出现段错误
        exit (0); //ok
    }
    else
    {
        printf ( "\nfather: pid=%d\n" ,pid);
        printf ( "father: count=%d\n" ,count);
    }
    return (0);
}
```

运行结果

?

```
[root@localhost part1_linux]# gcc fork2.c
[root@localhost part1_linux]# ./a
child: count=0
child: getpid=9911
```



firefly_2002

关注

25



```
child: count=1

father: pid=9911
father: count=1
```

运行结果说明：vfork时父、子进程共享数据段，fork时是进行拷贝。如果，vfork子进程中，使用return返回时，出现段错误，结果如下：

?

```
[root@localhost part1_linux]# gcc fork2.c
[root@localhost part1_linux]# ./a.out
child: count=0
child: getpid=10864
child: count=1

father: pid=10864
father: count=0
段错误
```

2. 为什么执行结果子进程打印出来 我的父亲是id:1，与父进程id不同

?

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

int main()
{
    int i=0;
    pid_t pid;
    printf ( "还没创建子进程\n" );
    i++;
    pid = fork();
    if (pid==-1)
    {
        printf ( "fork error!\n" );
    }
    else if (pid==0)
    {
        i++;
        printf ( "我是子进程,id%d\n",getpid());
        printf ( "我的父亲是id:%d\n",getppid());
        printf ( "-----i=%d-----\n",i);
    }
    else
    {
        i++;
        printf ( "我是父进程,id:%d\n",getpid());
        printf ( "-----i=%d-----\n",i);
    }
    exit (0);
}
```

子进程在打印第一句时，父进程也在打印第一句



firefly_2002

关注

25



接over了（这只是个简单的说法，实际过程可能并不如此，我要说的是，父进程先于子进程的打印语句之前就结束）。因此此时的子进程成了孤儿进程，会被init也就是1号进程领养，成为init的子进程。为了避免这样的情况，父进程最后可以执行wait来等待子进程的返回。

3. 用vfork() 创建子进程，执行后程序一直不断地重复运行，不断创建子进程，结尾用exit(0)代替return(0)后问题就能解决

return 0在一个函数中是正常的返回过程，它会使得程序返回到函数被调用处，回复之前的执行流程，return 语句不会被执行。而exit 一般是在任意位置和使用的，执行到exit 0时，整个进程就over了（这也是为什么在多线程程序中不要随意使用exit的原因），用来使程序退出运行，一般用来处理（不可挽回的）错误状态。

?

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

int main()
{
    int i=0;
    pid_t pid;
    printf ( "还没创建子进程\n" );
    i++;
    pid = vfork();
    if (pid==-1)
    {
        printf ( "fork error!\n" );
    }
    else if (pid==0)
    {
        i++;
        printf ( "我是子进程,id%d\n",getpid());
        printf ( "我的父亲是id:%d\n",getppid());
        printf ( "-----i=%d-----\n",i);
    }
    else
    {
        i++;
        printf ( "我是父进程,id:%d\n",getpid());
        printf ( "-----i=%d-----\n",i);
    }
    return (0);
}
```

文章知识点与官方知识档案匹配，可进一步学习相关知识

CS入门技能树 Linux入门 初识Linux 17839 人正在系统学习中

Shell中exit和return的区别讲解

09-15

今天小编就为大家分享一篇关于，小编觉得内容：



firefly_2002

关注

25

