

Linux下典型的进程控制操作

获取进程 ID

1. 创建源代码文件

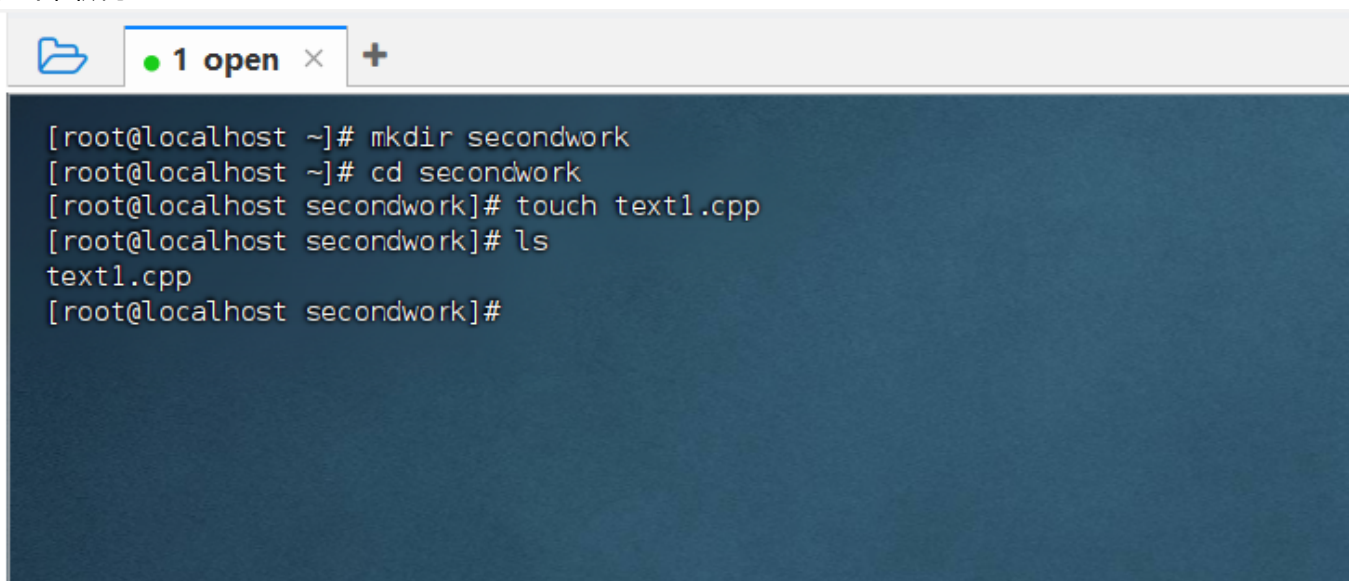
在命令行输入以下指令创建一个新的文件夹及cpp文件

```
mkdir secondwork  
touch text1.cpp
```

输入以下指令使用vim编辑text1.cpp文件

```
vim text1.cpp
```

如下图所示



A terminal window with a dark blue background. The window title bar shows a folder icon, a green dot, and the text '1 open' with a close button. The terminal content shows the following commands and output:

```
[root@localhost ~]# mkdir secondwork  
[root@localhost ~]# cd secondwork  
[root@localhost secondwork]# touch text1.cpp  
[root@localhost secondwork]# ls  
text1.cpp  
[root@localhost secondwork]#
```

2. 编写代码

在 `text1.cpp` 中写入如下代码：

```
#include<stdio.h>  
#include<sys/types.h>  
#include<unistd.h>  
  
int main()  
{  
    pid_t my_pid;  
    my_pid = getpid();  
    printf("My process ID is %d\n", my_pid);  
}
```

```
    return 0;
}
```

如下图所示

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

int main()
{
    pid_t my_pid;
    my_pid = getpid();
    printf("My process ID is %d\n", my_pid);
    return 0;
}
~
~
~
```

3. 编译并运行代码

编译代码：

```
g++ text1.cpp -o text1
```

运行程序：

```
./text1
```

结果如下图所示

```
[root@localhost secondwork]# vim text1.cpp
[root@localhost secondwork]# g++ text1.cpp -o text1
[root@localhost secondwork]# ls
text1  text1.cpp
[root@localhost secondwork]# ./text1
My process ID is 946813
[root@localhost secondwork]# █
```

```
My process ID is 946813
```

由此可见，当前程序的进程号（PID）为946813

进程创建与父子进程关系实验

1. 创建源代码文件

使用以下命令创建并编辑文件 `text2.cpp`:

```
vim text2.cpp
```

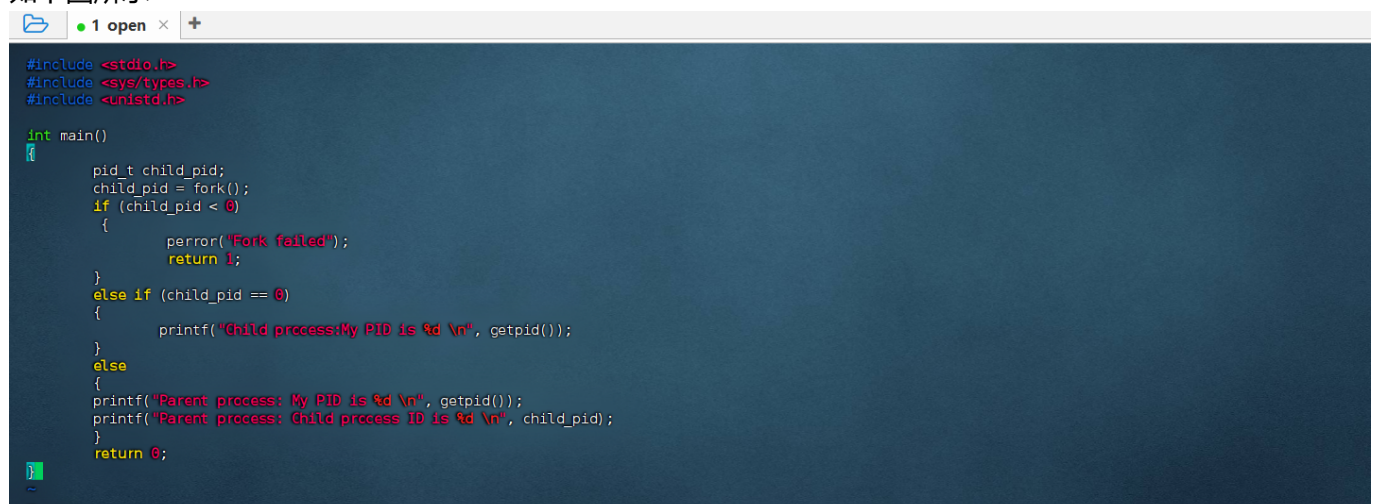
2.编写代码

编写代码如下

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

int main()
{
    pid_t child_pid;
    child_pid = fork();
    if (child_pid < 0)
    {
        perror("Fork failed");
        return 1;
    }
    else if (child_pid == 0)
    {
        printf("Child process:My PID is %d \n", getpid());
    }
    else
    {
        printf("Parent process: My PID is %d \n", getpid());
        printf("Parent process: Child process ID is %d \n", child_pid);
    }
    return 0;
}
```

如下图所示



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

int main()
{
    pid_t child_pid;
    child_pid = fork();
    if (child_pid < 0)
    {
        perror("Fork failed");
        return 1;
    }
    else if (child_pid == 0)
    {
        printf("Child process:My PID is %d \n", getpid());
    }
    else
    {
        printf("Parent process: My PID is %d \n", getpid());
        printf("Parent process: Child process ID is %d \n", child_pid);
    }
    return 0;
}
```

3. 编译并运行程序

得到的结果如下图所示

```
[root@localhost secondwork]# g++ text2.cpp -o text2
[root@localhost secondwork]# ls
text1  text1.cpp  text2  text2.cpp
[root@localhost secondwork]# ./text2
Parent process: My PID is 995289
Child process:My PID is 995290
Parent process: Child process ID is 995290
[root@localhost secondwork]#
```

```
Parent process: My PID is 995289
Parent process: Child process ID is 995290
Child process:My PID is 995290
```

结果表明：

- fork()调用后，内核会将父进程的内存空间、文件描述符、寄存器状态等完整复制到子进程中，形成两个独立的执行流
- 证明父子进程拥有独立的内存空间

父进程等待子进程退出测试

1.修改代码

修改er.cpp中的代码

```
vim text2.cpp
```

修改为下面的代码

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

int main()
{
    pid_t child_pid;
    child_pid = fork();
    if (child_pid < 0)
    {
        perror("Fork failed");
        return 1;
    }
    else if (child_pid == 0)
    {

```

```
        printf("Child process:My PID is %d \n", getpid());
    }
    else
    {
        printf("Parent process: Child process ID is %d \n", child_pid);
        int status;
        waitpid(child_pid, &status, 0);
        if (WIFEXITED(status))
        {
            printf("Parent process: Child exited with status %d\n",
WEXITSTATUS(status));
        }
    }
    return 0;
}
```

2.运行代码

得到结果如下

```
[root@localhost secondwork]# vim text2.cpp
[root@localhost secondwork]# g++ text2.cpp -o text2
[root@localhost secondwork]# ./text2
Parent process: Child process ID is 1068494
Child process:My PID is 1068494
Parent process: Child exited with status 0
[root@localhost secondwork]#
```

```
Parent process: Child process ID is 1068494
Child process:My PID is 1068494
Parent process: Child exited with status 0
```

结果显示:

- 父进程在调用 `waitpid()` 后进入等待状态, 直至子进程退出后才继续执行后续代码。
- 子进程正常退出 (退出状态为 0), 并且父进程通过 `WIFEXITED` 与 `WEXITSTATUS` 检查子进程的退出状态。

多次 fork() 进程创建实验

1. 编写代码

编写代码如下

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

int main()
```

```
{
    fork();
    fork();
    fork();
    printf("ciao!\n");
    return 0;
}
```

2. 编译并运行程序

结果如下图：

```
[root@localhost secondwork]# g++ 3.cpp -o text3
[root@localhost secondwork]# ls
3.cpp  text1  text1.cpp  text2  text2.cpp  text3
[root@localhost secondwork]# ./text3
ciao!
ciao!
ciao!
ciao!
ciao!
[root@localhost secondwork]# ciao!
ciao!
ciao!
```

这表明：每次调用 `fork()` 后，当前进程都会复制出一个新的进程。

- 第一次`fork()`：父进程P0创建子进程P1，总数变为2（P0和P1）。 - 第二次`fork()`：P0和P1各自调用`fork()`，分别创建子进程P2和P3，总数变为（P0、P1、P2、P3）。
- 第三次`fork()`：4个进程各自调用`fork()`，创建P4-P7，总数达到8。此时每个进程均会执行后续代码（如输出`ciao!`），最终输出8次。

进程独立性实验

1. 编写代码

编写代码如下

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

int main()
{
    int x = 1;
    pid_t p = fork();
    if (p < 0)
    {
        perror("fork fail");
        exit(1);
    }
}
```



```
    }  
    else if (p == 0)  
        printf("Child has x = %d \n", ++x);  
    else  
        printf("Parent has x = %d\n", --x);  
  
    return 0;  
}
```

2. 编译并运行程序

编译并运行得到结果如下

```
[root@localhost secondwork]# vim 4.cpp  
[root@localhost secondwork]# g++ 4.cpp -o text4  
[root@localhost secondwork]# ./text4  
Parent has x = 0  
Child has x = 2  
[root@localhost secondwork]#
```

```
Parent has x = 0  
Child has x = 2
```

结果表明，父子进程在 `fork()` 调用后拥有各自独立的内存空间。

体现了进程间变量互不干扰的特性