

# OpenEulsecond示例代码2

---

## 获取进程 ID 实验

### 1. 创建源代码文件

打开虚拟机，输入以下指令来编写代码

```
vi first.cpp
```

### 2. 编写代码

在 `first.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;
}
```

### 3. 编译并运行代码

编译代码：

```
g++ first.cpp -o first
```

运行程序：

```
./first
```

```
My process ID is 18290
```

由此可见，当前程序的进程号（PID）为 18290。

## 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 `second.cpp`：

```
vi second.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;
}
```

### 3. 编译并运行程序


编译代码：

```
g++ second.cpp -o second
```

运行程序：

```
./second
```

得到的结果如下图所示

 alt text

```
Parent process: My PID is 70845
Parent process: Child process ID is 70846
Child process:My PID is 70846
```

结果表明：

- `fork()` 系统调用成功后，会产生一个子进程。
- 父进程输出了自己的进程号以及子进程的进程号，而子进程仅输出自己的进程号。
- 这证明了父子进程之间的关系，其中子进程由父进程创建，并且它们具有各自独立的进程号。

## 父进程等待子进程退出测试

### 1.修改代码

修改second.cpp中的代码

```
vi second.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;  
}
```

如图所示



## 2.运行代码

再次编译代码并运行

```
g++ second.cpp -o second
```

```
./second
```

得到结果如下



```
Parent process: Child process ID is 140429  
Child process:My PID is 140429  
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. 创建结果保存文件

创建一个用来保存结果的文件save.txt

```
touch save.txt
```

## 3. 编译并运行程序

编译代码并将运行的结果导入到save.txt

```
g++ san.cpp -o san
```

```
./san > save.txt
```

打开save.txt，发现结果如下图所示

 alt text

这表明：每次调用 `fork()` 后，当前进程都会复制出一个新的进程。

- 首次 `fork()` 后：2 个进程
  - 再次 `fork()` 后：每个进程又复制出一个子进程，总数达到 4 个
  - 第三次 `fork()` 后：总数达到  $(2^3 = 8)$  个进程
- 因此，程序共输出 8 次 `ciao!`，验证了进程复制的倍增效果。

## 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. 运行代码

```
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。

- 父进程对变量 `x` 执行自减操作，输出结果为 0；
- 子进程对变量 `x` 执行自增操作，输出结果为 2。

这验证了进程间变量互不干扰的特性

## 总结

通过多个实例验证了 Linux 系统中进程创建的基本原理与特点：

- 通过 `getpid()` 能够获取进程号；
- `fork()` 调用成功后会创建一个子进程，父子进程各自独立运行；
- 父进程可以通过 `waitpid()` 等机制等待子进程退出，从而实现进程同步；
- 多次 `fork()` 会呈指数级地增加进程数；
- 各进程拥有独立的内存空间，对变量的修改互不影响。