

##作业要求：

对于这种编程实践形式的作业，推荐按照代码+截图+文字说明的方式，汇总成一个Word文档（对字数没有特别要求，关键是能够把问题解答清楚，排版不要特别丑陋即可），本文档最后一页附有示例格式。

第二次作业：

作业 1

2. 程序并发执行时的特征

3) 不可再现性

程序并发执行，由于失去了封闭性，从而也失去了可再现性。

○ 例如，有两个循环程序A和B，它们共享一个变量N。程序A每执行一次时，都要做 $N = N + 1$ 操作；程序B每执行一次时，都要执行Print(N)操作，然后再将N置成“0”。程序A和B以不同的速度运行。

程序A和B以不同的速度运行出现的情况：

- 1、 $N=N+1$ ，在Print (N) 和 $N=0$ 之前执行，即执行次序：

$N=N+1$	n+1
Print (N)	n+1
$N=0$	0
- 2、 $N=N+1$ ，在Print和 $N=0$ 之后执行，即执行次序：

Print (N)	n
$N=0$	0
$N=N+1$	1
- 3、 $N=N+1$ ，在Print和 $N=0$ 之间执行，即执行次序：

Print (N)	n
$N=N+1$	n+1
$N=0$	0

彩色的为执行结果，各不相同

在Linux操作系统中实现上页(上图)中例子，运行多次，每次的结果有什么不同？

作业 2：

阅读Linux 3.0以上版本的内核代码中进程控制块和进程调度的代码，然后回答下面的问题：

Linux的进程控制块的组织方式是什么？

请问它里面设定了那些进程状态，这些状态代表什么意义？

状态之间如何转换？并画出状态转换图。

作业 3：

```
#include <stdio.h>

int main()
{
    int child;
    char *args[ ] = {"./bin/echo", "Hello", "World!", NULL};

    if (!(child = fork())) // 子进程从fork返回时，返回值为0；父进程从fork()返回时，返回值为子进程的id时，非0。
    {
        /* child */
        printf("pid %d: %d is my father\n", getpid(), getppid());
        execl("./bin/echo", args, NULL);
        printf("pid %d: I am back, something is wrong!\n", getpid());
    }
    else
    {
        int myself = getpid();
        printf("pid %d: %d is my son\n", myself, child);
        wait4(child, NULL, 0, NULL); // 等待子进程执行结束。
        printf("pid %d: done\n", myself);
    }
    return 0;
}
```

将上页中的例子在 Linux 操作系统中运行，并得出执行结果，学会 wait4 函数和 fork 函数的功能和使用方法。

学习 Linux 系统调用 wait3, wait4, waitpid 的功能和使用方法，他们都是指定要等待某进程结束，他们有何区别？编程练习这 3 个函数的使用。

附：作业 1 示例

操作系统第二次作业

作业 1

1. 相关代码：

(1) 程序 A:

```
#include...
int main() {
    ...
}
```

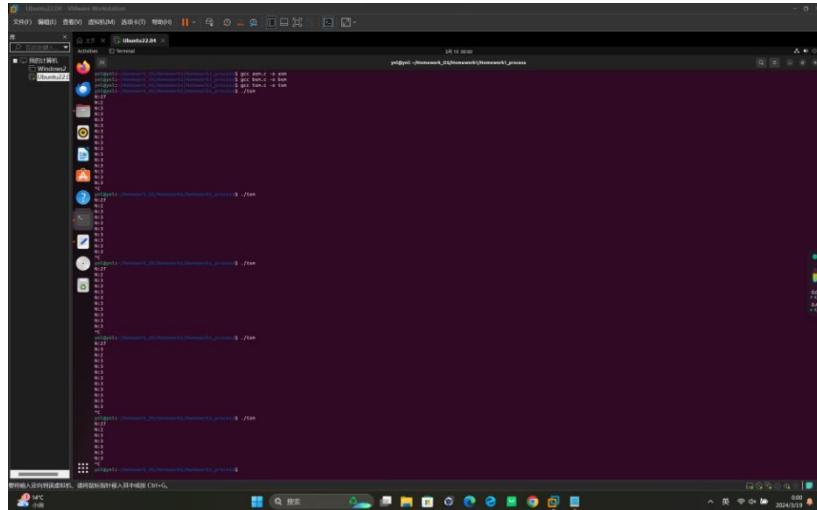
(2) 程序 B

```
#####
```

(3) 主程序

2. 实验截图（A、B 在不同速度下进行实验）

(1) A: usleep (100000); B: usleep (300000)



(2) A: usleep (100000); B: usleep (200000)。

```
#####
```

(3) A: usleep (100000); B: usleep (600000)

```
#####
```

3. 解释说明

(1) 关于程序的构思

#####。

(2) 关于实验结果

#####。