一、实验目的与要求：

通过进程的创建、撤销和运行加深对进程概念和进程并发执行的理解，明确进程与程序之间的区别。

二、方法、步骤：(说明程序相关的算法原理或知识内容，程序设计的思路和方法，可以用流程图表述，程序主要数据结构的设计、主要函数之间的调用关系等)

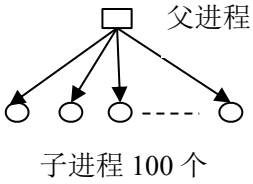
1. 掌握在在 linux 中编程编译运行的方法，试验你的第一个 helloworld 程序。

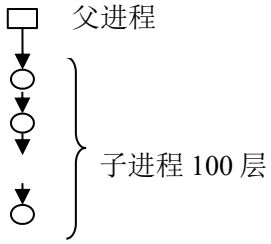
2. 学习预备材料和后面的阅读例程，理解函数 fork()、execl()、exit()、getpid()和 waitpid()的功能和用法

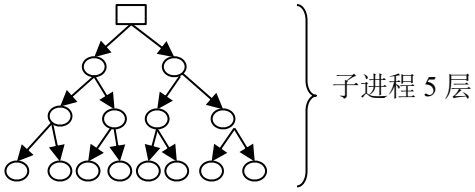
3. 编写 hello-loop.c 程序（在 helloworld 例程基础上加一个死循环）。使用 gcc hello-loop.c -o helloworld 生成可执行文件 hello-loop。并在同一个目录下，通过命令"./hello-loop"执行之。使用 top 和 ps 命令查看该进程，记录进程号以及进程状态。

4. 使用 kill 命令终止 hello-loop 进程。

5. 使用 fork()创建子进程，形成以下父子关系：通过检查进程的 pid 和 ppid 证明你成功创建相应的父子关系并用 pstree 验证其关系。







6. 编写代码实现孤儿进程，用 pstree 查看孤儿进程如何从原父进程位置转变为 init 进程所收养的过程；编写代码创建僵尸进程，用 ps j 查看其运行状态。
7.编写一个代码，使得进程循环处于以下状态 5 秒钟运行 5 秒钟阻塞（例如可以使用 sleep( )），并使用 top 或 ps 命令检测其运行和阻塞两种状态，并截图记录；

三．实验过程及内容：(对程序代码进行说明和分析，越详细越好，代码排版要整齐，可读性要高)

1. 掌握在 linux 中编程编译运行的方法，试验 helloworld 程序：

如图所示:用 vim 编辑器编写第一个 helloworld.c 程序,用 cat 查看程序代码,用 gcc 编译，最后运行，成功运行后，显示“Hello，World!”

```
[root@iZwz9j14gd1jp0mfm3206jZ ~]# vim helloworld.c
[root@iZwz9j14gd1jp0mfm3206jZ ~]# cat helloworld.c
#include <stdio.h>
int main()
 {
       printf("Hello, World!");
       return 0;
}
[root@iZwz9j14gd1jp0mfm3206jZ ~]# gcc helloworld.c -o test
[root@iZwz9j14gd1jp0mfm3206jZ ~]# ./test
Hello, World![root@iZwz9j14gd1jp0mfm3206jZ ~]#
```

2.理解函数 fork()、execl()、exit()、getpid()和 waitpid()的功能和用法

1. fork()会产生一个新的子进程，其子进程会复制父进程的数据与堆栈空间，并继承父进程的用户代码，组代码，环境变量、已打开的文件代码、工作目录和资源限制，所以子进程执行的入口也是这个程序的 main 函数，为了让我们判别子进程和父进程，需使子进程运行的代码有异于父进程；根据 fork()函数返回值可以做到，因为如果是父进程执行 fork()函数会返回创建的子进程的 PID，而子进程执行 fork()函数会返回 0，所以设计代码时使用 if-else 语句分离父进程和子进程的工作；
2. 为了表现出 waitpid(…)的功能，用 sleep()函数延长子进程的存活时间，waitpid(…)函数等待子进程中断或结束，如果执行成功则返回子进程识别码(PID)；
3.getpid()函数返回当前运行的进程的 PID。

3. hello-loop.c 程序

编写 hello-loop.c 程序（在 helloworld 例程基础上加一个死循环）并且运行程序

```
[root@iZwz9j14gd1jp0mfm3206jZ ~]# cat hello-loop.c
#include <stdio.h>
int main()
 {
      printf("Hello, World!");
        while(1){};
       return 0;
  }
[root@iZwz9j14gd1jp0mfm3206jZ ~]# gcc hello-loop.c -o helloworld
[root@iZwz9j14gd1jp0mfm3206jZ ~]# ./helloworld
```

打开另一个终端，分别用 top 和 ps 命令查看进程（此处本应该用 ps j 命令的，但此时程序被已经删除….），可以知道 hello-loop 进程的 pid 为 10840

```
top - 15:45:47 up  3:34,  4 users,  load average: 2.83, 2.61, 1.66
Tasks:  98 total,   9 running,  89 sleeping,   0 stopped,   0 zombie
%Cpu(s): 79.1 us,  7.7 sy,  0.0 ni, 13.2 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  3880416 total,  2393556 free,   128444 used,  1358416 buff/cache
KiB Swap:        0 total,        0 free,        0 used.  3448696 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
10840 root      20   0    4212    352    276 R  63.5  0.0   0:20.42 helloworld
 1065 root      10 -10  128500  12228   8680 S  20.6  0.3   5:04.42 AliYunDun
12147 root      20   0  133548   9672   3928 R   1.0  0.2   0:00.03 conda
10966 root      20   0  157584   6288   4596 R   0.3  0.2   0:00.30 sshd
12148 root      20   0  125696   6296   2812 R   0.3  0.2   0:00.01 conda
12152 root      20   0  125692   6128   2796 R   0.3  0.2   0:00.01 conda
    1 root      20   0   43528   3792   2580 S   0.0  0.1   0:00.76 systemd
    2 root      20   0       0      0      0 S   0.0  0.0   0:00.00 kthreadd
    3 root      20   0       0      0      0 S   0.0  0.0   0:00.22 ksoftirqd/0
```

```
(base) [root@iZwz9j14gd1jp0mfm3206jZ ~]# ps
  PID TTY          TIME CMD
10983 pts/2    00:00:00 bash
11330 pts/2    00:00:00 ps
(base) [root@iZwz9j14gd1jp0mfm3206jZ ~]#
```

4. 使用 kill 命令终止 hello-loop 进程。

打开一个新的终端 2，由于前面通过 top 命令已得知 hello-loop 进程的 pid 为 10840，所以通过 kill 10840 命令来终止进程

```
(base) [root@iZwz9j14gd1jp0mfm3206jZ ~]# kill 10840
```

此时可以看到原来终端 1 的 hello-loop 进程显示 已终止。

```
[root@iZwz9j14gd1jp0mfm3206jZ ~]# ./helloworld
已终止
[root@iZwz9j14gd1jp0mfm3206jZ ~]#
```

5. 使用 fork()创建子进程

5.1：子进程 100 个

```
#include<unistd.h>
#include<stdio.h>
#include<stdlib.h>
#include<sys/wait.h>
#define child 100 // define the max_num of child process

int process(int n);
int main()
{
        pid_t pid;
        process(child);
        return 0;
}
int process(int n) //define the process function
{
        if (n<= 0) //condition outside the function
        {
                return 0;
        }
        else
        {
```

```
                pid_t pid=fork();//set child process
                if (pid == 0)
                {
                        printf(" child process %d:%d\n",child-n+1,getpid());
                        sleep(5);
                }
                else {
                        printf("This is parent process,my process id is %d\n",getpid());
                        process(n-1);
                        sleep(5);
                        return 0;
                }
                waitpid(pid,NULL,WUNTRACED);
        }
        return 0;
}
```

运行程序，可以看到如下结果：

```
(base) [root@iZwz9j14gd1jp0mfm3206jZ ~]# gcc fork53.c -o two
(base) [root@iZwz9j14gd1jp0mfm3206jZ ~]# ./two
This is parent process,my process id is 7880
This is parent process,my process id is 7880
This is parent process,my process id is 7880
 child process 1:7881
This is parent process,my process id is 7880
This is parent process,my process id is 7880
 child process 3:7883
This is parent process,my process id is 7880
This is parent process,my process id is 7880
This is parent process,my process id is 7880
 child process 2:7882
This is parent process,my process id is 7880
This is parent process,my process id is 7880
 child process 9:7889
This is parent process,my process id is 7880
This is parent process,my process id is 7880
This is parent process,my process id is 7880
 child process 8:7888
This is parent process,my process id is 7880
This is parent process,my process id is 7880
This is parent process,my process id is 7880
```

并在终端 2 用 pstree 命令查看进程，可以看到 two—100*[two],完成 100 个子进程的创建。

```
      |      |      ├─6*[bash───bash───conda]
      |      |      └─sftp-server
      |      ├─sshd─┬─2*[bash───bash───conda]
      |      |      ├─3*[bash───bash───bash]
      |      |      ├─bash───vim───bash───two───100*[two]
      |      |      ├─bash───top
      |      |      └─sftp-server
      |      └─sshd─┬─bash
      |             ├─bash───top
      |             └─sftp-server
```

通过命令可以看到在程序运行前、后进程控制块上的开销，分别为 156、241

```
(base) [root@iZwz9j14gd1jp0mfm3206jZ ~]# cat /proc/slabinfo |grep task_struct
task_struct          156    210   4160    7    8 : tunables    0    0    0 : slabdata     30     30      0
```

```
(base) [root@iZwz9j14gd1jp0mfm3206jZ ~]# cat /proc/slabinfo |grep task_struct
task_struct          241    266   4160    7    8 : tunables    0    0    0 : slabdata     38     38      0
```

5.2 子进程 100 层

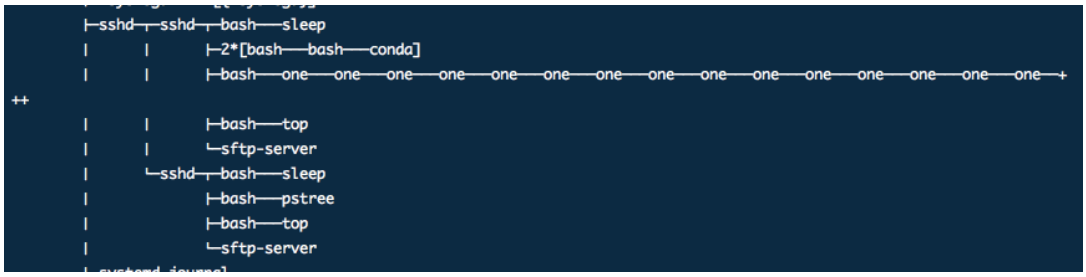
```
#include<unistd.h>
#include<stdio.h>
#include<stdlib.h>
#include<sys/wait.h>
#define child 100 // define the max_num of child process

int process(int n);
int main()
{
        pid_t pid;
        process(child);
        return 0;
}
int process(int n) //define the process function
{
        if (n<= 0) //condition outside the function
        {
                return 0;
        }
        else
        {
                pid_t pid=fork();//set child process
               if (pid == 0)
                {
                        printf(" child process %d:%d\n",child-n+1,getpid());
                        process(n-1);
                        sleep(5);
                        return 0; // child process set child process
                }
                else {
                        printf("This is parent process,my process id is %d\n",getpid());
                        sleep(5);
                }
                waitpid(pid,NULL,WUNTRACED);
          }
        return 0;
}
```

运行程序，可以看到如下结果：

```
This is parent process,my process id is 7923
 child process 90:7924
This is parent process,my process id is 7924
 child process 91:7925
This is parent process,my process id is 7925
 child process 92:7926
This is parent process,my process id is 7926
 child process 93:7927
This is parent process,my process id is 7927
 child process 94:7928
This is parent process,my process id is 7928
 child process 95:7929
This is parent process,my process id is 7929
 child process 96:7930
This is parent process,my process id is 7930
 child process 97:7931
This is parent process,my process id is 7931
 child process 98:7932
This is parent process,my process id is 7932
 child process 99:7933
This is parent process,my process id is 7933
 child process 100:7934
```

并在终端 2 用 pstree 命令查看进程，可以看到 one-one-one---- 的进程树,完成 100 层子进程的创建。



通过命令可以看到在程序运行前、后进程控制块上的开销，分别为 156、256

```
(base) [root@iZwz9j14gd1jp0mfm3206jZ ~]# cat /proc/slabinfo |grep task_struct
task_struct          156    182   4160    7    8 : tunables    0    0    0 : slabdata     26     26      0
```

```
(base) [root@iZwz9j14gd1jp0mfm3206jZ ~]# cat /proc/slabinfo |grep task_struct
task_struct          256    266   4160    7    8 : tunables    0    0    0 : slabdata     38     38      0
```

5.3 子进程 5 层

```
#include<stdio.h>
#include<unistd.h>
int main()
{
    int i;
    for(i=0;i<5;i++){
        pid_t t1=fork();
        pid_t t2=0;
        if(t1==0){
```

```
            printf("child process,id=%d,my parent is %d\n",getpid(),getppid());
        }
        else if(t1>0){
            printf("parent process,id=%d\n",getpid());
            t2=fork();
        }
        if(t1>0&&t2>0)
        break;
}
    while(1);
    return 0;
}
```

运行程序，可以看到如下结果：

```
(base) [root@iZwz9j14gd1jp0mfm3206jZ ~]# ./test7
parent process,id=27937
child process,id=27938,my parent is 27937
parent process,id=27938
child process,id=27940,my parent is 27938
parent process,id=27940
parent process,id=27941
parent process,id=27939
child process,id=27942,my parent is 27940
parent process,id=27942
parent process,id=27943
child process,id=27944,my parent is 27941
```

并在终端 2 用 pstree 命令查看进程，可以看到 2*[test7—2*[test7]]… 的进程树,完成 5 层二叉树子进程的创建。

```
  |            ├─3*[bash───bash───conda]
  |            ├─bash───bash───bash
  |            ├─bash───test7───2*[test7───2*[test7───2*[test7───2*[test7───2*[test7]]]]]
  |            ├─bash───top
  |            ├─sftp-server
```

通过命令可以看到在程序运行前、后进程控制块上的开销，分别为 181、315（此处之所以在运行前为 181，而不是同前面一样的 156，可能是由于做实验的时间不一致，系统运行不一样）

```
(base) [root@iZwz9j14gd1jp0mfm3206jZ ~]# cat /proc/slabinfo |grep task_struct
task_struct          181    203   4160    7    8 : tunables    0    0    0 : slabdata     29     29      0
```

```
(base) [root@iZwz9j14gd1jp0mfm3206jZ ~]# cat /proc/slabinfo |grep task_struct
task_struct          315    315   4160    7    8 : tunables    0    0    0 : slabdata     45     45      0
```

6. 写孤儿进程，用 pstree 查看孤儿进程如何从原父进程位置转变为 init 进程所收养的过程；编写代码创建僵尸进程，用 ps j 查看其运行状态。

6.1 孤儿进程

编写一个 fork_demo.c 并运行，如下图所示：

```
(base) [root@iZwz9j14gd1jp0mfm3206jZ ~]# vim fork_demo.c
(base) [root@iZwz9j14gd1jp0mfm3206jZ ~]# cat fork_demo.c
#include<stdio.h>
#include<unistd.h>
int main()
{
        int pid=fork();
        int i=0;
        if(pid==-1){
                printf("error!\n");
        }else if (pid==0){
                printf("This is the child  process!\n");
                getchar();
        }else if(pid){
                printf("This is the parent process! child process id=%d\n",pid);
                fork();
                getchar();
        }
        return 0;
}
(base) [root@iZwz9j14gd1jp0mfm3206jZ ~]# gcc fork_demo.c -o demo
(base) [root@iZwz9j14gd1jp0mfm3206jZ ~]# ./demo
```

可以看到运行结果：

```
(base) [root@iZwz9j14gd1jp0mfm3206jZ ~]# gcc fork_demo.c -o demo
(base) [root@iZwz9j14gd1jp0mfm3206jZ ~]# ./demo
This is the parent process! child process id=12524
This is the child  process!
```

打开另一个终端 2，用 ps j 和 pstree 查看进程，共有三个进程（12523、12524、12525 ）呈现父子关系，如下图所示

```
(base) [root@iZwz9j14gd1jp0mfm3206jZ ~]# ps j
 PPID   PID  PGID   SID TTY      TPGID STAT   UID   TIME COMMAND
    1   642   642   642 tty1       642 Ss+      0   0:00 /sbin/agetty --noclear tty1 linux
    1   644   644   644 ttyS0      644 Ss+      0   0:00 /sbin/agetty --keep-baud 115200,38400,9600 ttyS0 vt220
 1150  1152  1152  1152 pts/0    12523 Ss       0   0:00 -bash
 1150  1171  1171  1171 pts/1     1386 Ss       0   0:00 -bash
 1171  1386  1386  1171 pts/1     1386 S+       0   0:00 top
 1152 12523 12523  1152 pts/0    12523 S+       0   0:00 ./demo
12523 12524 12523  1152 pts/0    12523 S+       0   0:00 ./demo
12523 12525 12523  1152 pts/0    12523 S+       0   0:00 ./demo
12831 13016 13016 13016 pts/2    13360 Ss       0   0:00 -bash
12831 13079 13079 13079 pts/3    13079 Ss+      0   0:00 -bash
13016 13360 13360 13016 pts/2    13360 R+       0   0:00 ps j
(base) [root@iZwz9j14gd1jp0mfm3206jZ ~]# pstree 12523
demo──2*[demo]
```

在终端 1 中敲击回车键一次，此时父进程退出，在终端 2 可以看到子进程成为孤儿进程并被 init 进程收养（子进程 12524、12525 的 PPID 修改为指向 1 号进程）

```
(base) [root@iZwz9j14gd1jp0mfm3206jZ ~]# ps j
 PPID   PID  PGID   SID TTY      TPGID STAT   UID   TIME COMMAND
    1   642   642   642 tty1       642 Ss+      0   0:00 /sbin/agetty --noclear tty1 linux
    1   644   644   644 ttyS0      644 Ss+      0   0:00 /sbin/agetty --keep-baud 115200,38400,9600 ttyS0 vt220
 1150  1152  1152  1152 pts/0     1152 Ss+      0   0:00 -bash
 1150  1171  1171  1171 pts/1     1386 Ss       0   0:00 -bash
 1171  1386  1386  1171 pts/1     1386 S+       0   0:00 top
    1 12524 12523  1152 pts/0     1152 S        0   0:00 ./demo
    1 12525 12523  1152 pts/0     1152 S        0   0:00 ./demo
12831 13016 13016 13016 pts/2    18263 Ss       0   0:00 -bash
12831 13079 13079 13079 pts/3    13375 Ss       0   0:00 -bash
13079 13375 13375 13079 pts/3    13375 S+       0   0:00 top
13016 18263 18263 13016 pts/2    18263 R+       0   0:00 ps j
(base) [root@iZwz9j14gd1jp0mfm3206jZ ~]# pstree 12524
demo
```

最后再在终端 1 点击回车键两次，此时子进程 12524、12525 正常结束，没有浪费任何资源

```
(base) [root@iZwz9j14gd1jp0mfm3206jZ ~]# gcc fork_demo.c -o demo
(base) [root@iZwz9j14gd1jp0mfm3206jZ ~]# ./demo
This is the parent process! child process id=12524
This is the child  process!

(base) [root@iZwz9j14gd1jp0mfm3206jZ ~]#
```

```
(base) [root@iZwz9j14gd1jp0mfm3206jZ ~]# ps j
 PPID   PID  PGID   SID TTY      TPGID STAT   UID   TIME COMMAND
    1   642   642   642 tty1       642 Ss+      0   0:00 /sbin/agetty --noclear tty1 linux
    1   644   644   644 ttyS0      644 Ss+      0   0:00 /sbin/agetty --keep-baud 115200,38400,9600 ttyS0 vt220
 1150  1152  1152  1152 pts/0     1152 Ss+      0   0:00 -bash
 1150  1171  1171  1171 pts/1     1386 Ss       0   0:00 -bash
 1171  1386  1386  1171 pts/1     1386 S+       0   0:00 top
12831 13016 13016 13016 pts/2    22686 Ss       0   0:00 -bash
12831 13079 13079 13079 pts/3    13375 Ss       0   0:00 -bash
13079 13375 13375 13079 pts/3    13375 S+       0   0:00 top
13016 22686 22686 13016 pts/2    22686 R+       0   0:00 ps j
(base) [root@iZwz9j14gd1jp0mfm3206jZ ~]#
```

6.2 僵尸进程

编写一个 zombie_demo.c 并运行,子进程打印一行提示信息后将退出，但是此时父进程正在 sleep（）还未准备处理子进程的 PCB 等遗留的数据对象，因此子进程处于僵尸状态。

```
(base) [root@iZwz9j14gd1jp0mfm3206jZ ~]# cat zombie_demo.c
#include<sys/types.h>
#include<unistd.h>
#include<stdio.h>
#include<stdlib.h>

int main()
{
        pid_t pid=fork();
if(pid<0)
         printf("error!\n");
else if (pid == 0){
        printf("i am zomble\n");
        exit(0);//no one waits for this process
}
else{
        sleep(10);
        wait(NULL);//the zombie process will be reaped now
}
}

(base) [root@iZwz9j14gd1jp0mfm3206jZ ~]#
```

编译运行 zombie 程序，打开终端 2.执行 ps j 命令可以看到子进程处于“Z+”状态，表明是僵尸状态 defunct，如图所示

```
(base) [root@iZwz9j14gd1jp0mfm3206jZ ~]# gcc zombie_demo.c -o zombie
(base) [root@iZwz9j14gd1jp0mfm3206jZ ~]# ./zombie
i am zomble
```

```
(base) [root@iZwz9j14gd1jp0mfm3206jZ ~]# ps j
 PPID   PID  PGID   SID TTY      TPGID STAT   UID   TIME COMMAND
32546   403   403 32546 pts/3      403 S+       0   0:00 top
    1   642   642   642 tty1       642 Ss+      0   0:00 /sbin/agetty --noclear tty1 linux
    1   644   644   644 ttyS0      644 Ss+      0   0:00 /sbin/agetty --keep-baud 115200,38400,9600 ttyS0 vt220
19979  3006  3006 19979 pts/0     3006 S+       0   0:00 ./zombie
 3006  3007  3006 19979 pts/0     3006 Z+       0   0:00 [zombie] <defunct>
32505  3209  3209 32505 pts/2     3209 R+       0   0:00 ps j
19977 19979 19979 19979 pts/0     3006 Ss       0   0:00 -bash
19977 20016 20016 20016 pts/1    20213 Ss       0   0:00 -bash
20016 20213 20213 20016 pts/1    20213 S+       0   0:00 top
32089 32505 32505 32505 pts/2     3209 Ss       0   0:00 -bash
32089 32546 32546 32546 pts/3      403 Ss       0   0:00 -bash
(base) [root@iZwz9j14gd1jp0mfm3206jZ ~]#
```

在运行 10s 后，父进程将执行 wait()处理子进程的遗留数据对象，子进程将从僵尸状态转化为完全消失状态。

```
(base) [root@iZwz9j14gd1jp0mfm3206jZ ~]# ps j
 PPID   PID  PGID   SID TTY      TPGID STAT   UID   TIME COMMAND
32546   403   403 32546 pts/3      403 S+       0   0:00 top
    1   642   642   642 tty1       642 Ss+      0   0:00 /sbin/agetty --noclear tty1 linux
    1   644   644   644 ttyS0      644 Ss+      0   0:00 /sbin/agetty --keep-baud 115200,38400,9600 ttyS0 vt220
32505  7113  7113 32505 pts/2     7113 R+       0   0:00 ps j
19977 19979 19979 19979 pts/0    19979 Ss+      0   0:00 -bash
19977 20016 20016 20016 pts/1    20213 Ss       0   0:00 -bash
20016 20213 20213 20016 pts/1    20213 S+       0   0:00 top
32089 32505 32505 32505 pts/2     7113 Ss       0   0:00 -bash
32089 32546 32546 32546 pts/3      403 Ss       0   0:00 -bash
(base) [root@iZwz9j14gd1jp0mfm3206jZ ~]#
```

7. 编写一个代码，使得进程循环处于以下状态 5 秒钟运行 5 秒钟阻塞（例如可以使用 sleep( )），并使用 top 或 ps 命令检测其运行和阻塞两种状态，并截图记录；

```
#include<stdio.h>
#include<unistd.h>
int main()
{
        int i=1;
        while(1) {
            if(i){
                sleep(5);
                printf("sleep over\n");
                i=0;
            }
        };
        return 0;
}
```

编写并运行代码，运行结果如下：

```
(base) [root@iZwz9j14gd1jp0mfm3206jZ ~]# vim sleep7.c
(base) [root@iZwz9j14gd1jp0mfm3206jZ ~]# gcc sleep7.c -o sleep7
(base) [root@iZwz9j14gd1jp0mfm3206jZ ~]# ./sleep7
sleep over
```

通过终端 ps j 命令，可以看到进程./sleep7 的运行（R）和阻塞(S)的两种状态

```
(base) [root@iZwz9j14gd1jp0mfm3206jZ ~]# ps j
 PPID   PID  PGID   SID TTY      TPGID STAT   UID   TIME COMMAND
    1   955   955   955 ttyS0      955 Ss+      0   0:00 /sbin/agetty --keep-baud 115200,38400,9600 ttyS0 vt220
  953   993   993   993 tty1       993 Ssl+     0   0:00 /usr/bin/X -core -noreset :0 -seat seat0 -auth /var/run/
31977 27230 27230 31977 pts/0    27230 R+       0   0:00 ./sleep7
32197 27566 27566 32197 pts/2    27566 R+       0   0:00 ps j
31975 31977 31977 31977 pts/0    27230 Ss       0   0:00 -bash
31975 32008 32008 32008 pts/1    32363 Ss       0   0:00 -bash
32181 32197 32197 32197 pts/2    27566 Ss       0   0:00 -bash
32181 32239 32239 32239 pts/3    32484 Ss       0   0:00 -bash
32008 32363 32363 32008 pts/1    32363 S+       0   0:01 top
32239 32484 32484 32239 pts/3    32484 S+       0   0:01 top
```

```
(base) [root@iZwz9j14gd1jp0mfm3206jZ ~]# ps j
 PPID   PID  PGID   SID TTY      TPGID STAT   UID   TIME COMMAND
    1   955   955   955 ttyS0      955 Ss+      0   0:00 /sbin/agetty --keep-baud 115200,38400,9600 ttyS0 vt220
  953   993   993   993 tty1       993 Ssl+     0   0:00 /usr/bin/X -core -noreset :0 -seat seat0 -auth /var/run
31977 27230 27230 31977 pts/0    27230 S+       0   0:00 ./sleep7
32197 27397 27397 32197 pts/2    27397 R+       0   0:00 ps j
31975 31977 31977 31977 pts/0    27230 Ss       0   0:00 -bash
31975 32008 32008 32008 pts/1    32363 Ss       0   0:00 -bash
32181 32197 32197 32197 pts/2    27397 Ss       0   0:00 -bash
32181 32239 32239 32239 pts/3    32484 Ss       0   0:00 -bash
32008 32363 32363 32008 pts/1    32363 S+       0   0:01 top
32239 32484 32484 32239 pts/3    32484 S+       0   0:01 top
```

四、实验结论：（提供运行结果，对结果进行探讨、分析、评价，并提出结论性意见和改进想法）

1. 通过这次实验，了解到进程的概念，所谓进程，就是指在系统中能独立运行并作为资源分配的基本单
2. 并发执行指的是在一段时间内宏观上有多个程序同时运行，但在单处理机系统中，每一时刻只能有一道程序执行；所以在上述实验中，多个进程同时运行时，各个进程的输出结果是交替出现的。

五、实验体会：（根据自己情况填写）

1. 学会了在阿里云服务器上运行 Linux 系统，并在 Linux 系统上编译运行 C 语言代码，虽然在这个过程中遇到很多问题，但是通过各大 Linux 论坛和老师助教们的帮助得以解决，打开了新世界的大门！
2. 了解了进程的概念，初步接触操作系统。

注：“指导教师批阅意见”栏请单独放置一页