作业一

Part1 进程与线程

1. 进程和程序的一个本质区别是（D）

   A. 前者分时使用 CPU，后者独占 CPU　　B. 前者存储在内存，后者存储在外存

   C. 前者在一个文件中，后者在多个文件中　　D. 前者为动态的，后者为静态的

2. 下面所列进程的三种基本状态之间的转换不正确的是（B）

   A. 就绪状态->执行状态　　B. 就绪状态->阻塞状态

   C. 执行状态->阻塞状态　　D. 执行状态->就绪状态

3. 为什么进程切换的代价要比线程切换要大（C）

   A. 因为进程切换要切换栈

   B. 因为进程切换要切换控制块数据结构

   C. 因为进程切换要切换段表

   D. 因为进程切换要切换 PC 指针

4. 下列选项中，不可能在用户态发生的是（C）

   A. 系统调用　　B. 外部中断

   C. 进程切换　　D. 缺页

5. 在下述父进程和子进程的描述中，正确的是（A）

   A. 撤销父进程时，应该同时撤销子进程

   B. 父进程和子进程不可以并发执行

   C. 撤销子进程时，应该同时撤销父进程

   D. 父进程创建了子进程，因而父进程执行完后，子进程才能运行

6. 下列关于线程的叙述中，正确的是（C）

   I. 采用轮转调度算法时，一进程拥有 10 个用户级线程，则在系统调度执行时间上占用 10 个时间片

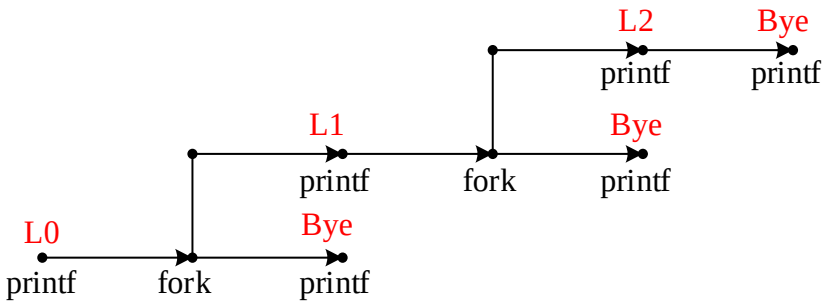
II. 属于同一个进程的各个线程共享栈空间

III. 同一进程中的线程可以并发执行，但不同进程的线程不可以并发执行

IV. 线程的切换，不会引起进程的切换

A. Ⅰ、Ⅱ、Ⅲ　　B. 仅Ⅱ、Ⅳ　　C. 全错　　D. 仅Ⅱ、Ⅲ

7. 根据进程图，以下哪个输出是不正确的（D）



A. L0, Bye, L1, Bye, L2, Bye

B. L0, L1, Bye, L2, Bye, Bye

C. L0, Bye, L1, L2, Bye, Bye

D. L0, Bye, L1, Bye, Bye, L2

8. 分析程序 homework_wait.c，回答下列问题：

```
1.  /* homework_wait.c */
2.  void homework_wait() {
3.     pid_t pid[N];// hello,
4.     int i, child_status;
5.     for (i = 0; i < N; i++) {
6.          if ((pid[i] = fork()) == 0) {
7.              exit(100+i); /* Child */
8.          }
9.      }
10.     printf("hello!\n");
11.     for (i = 0; i < N; i++) { /* Parent */
12.         pid_t wpid = wait(&child_status);
13.         if (WIFEXITED(child_status))
14.             printf("Child %d terminated with exit status %d\n",
15.                    wpid, WEXITSTATUS(child_status));
16.         else
17.             printf("Child %d terminate abnormally\n", wpid);
18.     }
```

```
19. }
```

1) 注释掉第 7 行代码后，程序执行到第 10 行，输出多少个“hello!”（用一个 N 的函数给出答案）？

   注释掉第七行则每个 fork 出来的子进程会继续执行循环，也会 fork 出进程。每次循环进程数加倍，初值为 1，故最后输出 2^N 个 hello！。

2) N=2 时，程序正常运行两次，得到的结果是否相同？若不同，请解释原因；

   不同，因为子进程退出时发出的信号父进程不一定按顺序接收到。

3) 修改程序，使得子进程能够按照其创建的顺序退出。（waitpid.c）

   将第 12 行调用 wait 函数的代码 `pid_t wpid = wait(&child_status);` 中的 `wait` 替换为 waitpid，即 pid_t wpid = waitpid(pid[i],&child_status,0);

Part 2. 并发与同步

1. 一个正在访问临界资源的进程由于申请等待 I/O 操作而被中断时，它（**C**）

   A. 允许其他进程进入与该进程相关的临界区

   B. 不允许其他进程进入任何临界区

   C. 允许其他进程抢占处理器，但不得进入该进程的临界区

   D. 不允许任何进程抢占处理器

2. 设与某资源相关联的信号量初值为 3，当前值为 1，若 M 表示该资源此时可以用的个数，N 表示等待该资源的进程数，那么 M 和 N 分别为（**B**）

   A. 0，1

   B. 1，0

   C. 1，2

   D. 2，0

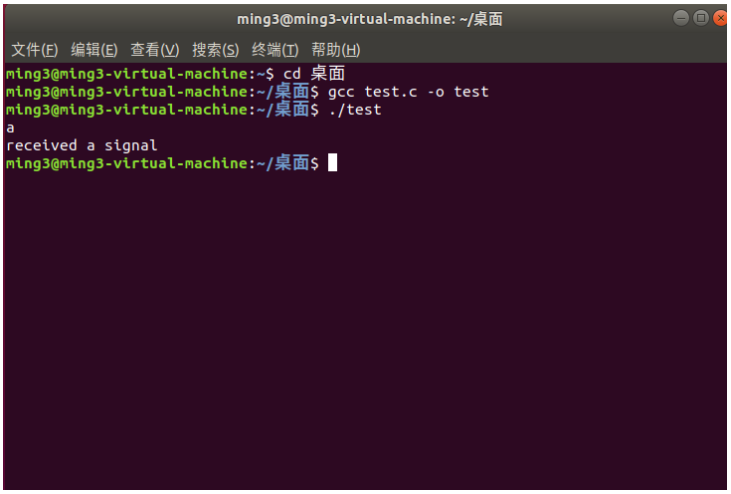
3. 如下所示的程序在运行后永远也不会结束。请改写该程序，在不删除和修改任何现有语句的前提下，只添加新语句，使得程序在用户输入任意字符后能够正常结束。并要求程序在结束前只可输出一条语句“received a signal\n”。（提示：使用信号；已给出所需要的头文件；建议实际运行进行测试）

```
1.  #include <stdio.h>
2.  #include <sys/types.h>
3.  #include <signal.h>
4.  #include <unistd.h>
5.  #include <stdlib.h>
6.  #include <sys/wait.h>
7.  int main(){
8.      int child_pid;
9.      if((child_pid=fork())==0){
10.         while(1);
11.         printf("forbidden zone\n");
12.         exit(0);
13.     }
14.     else{
15.         while(getc(stdin)){
16.             wait(0);
17.             exit(0);
20.         }
```

```
21.     }
22. }
```

```
#include <stdio.h>
#include <sys/types.h>
#include <signal.h>
#include <unistd.h>
#include <stdlib.h>
#include <sys/wait.h>
void handler(){
        exit(0);
}
int main(){
        int child_pid;
        if((child_pid=fork())==0){
        signal(SIGINT,handler);
                while(1);
                printf("forbidden zone\n");
                exit(0);
        }
        else{
                while(getc(stdin)){
                kill(child_pid,SIGINT);
                printf("received a signal\n");
                        wait(0);
                        exit(0);
}
        }
}
```

输入一个字符时，调用 kill 向子进程发送信号，子进程接受到信号后，会执行 handler 退出



如图，输入一个字符(a)后运行成功。

4. 三个进程P_1、P_2、P_3互斥使用一个包含 N（N>0）个单元的缓冲区。P_1每次使用 produce()

生成一个正整数并用 put()送入缓冲区某一空单元；P_2每次用 get_odd()从该缓冲区取出一个奇数，然后用 count_odd()统计已经获取的奇数个数；P_3每次用 get_even()从该缓冲区取出一个偶数，然后用 count_even()统计已经获取的偶数个数。请用信号量机制实现这三个进程的互斥与同步活动，并说明所定义的信号量的含义（要求用伪代码描述）。

定义信号量 s1,s2 初值为 0,0 分别表示缓冲区奇数和偶数的个数。s3 初值为 N，表示缓存区的空闲单元个数。 mutex 用于互斥。

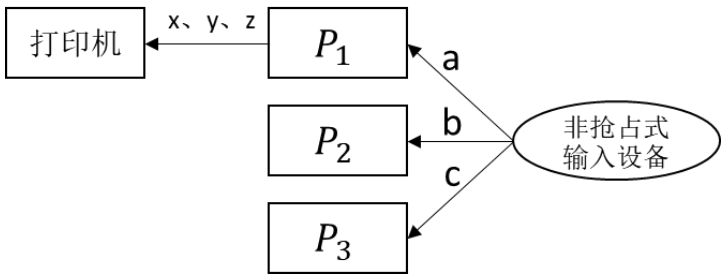
```
P1(){
    while(true){
        int x = produce();
        P(s3);
        P(mutex);
        put(x);
        V(mutex);
        if(x % 2== 0) V(s2);
        else V(s1);
    }
}
P2(){
    while(true){
        P(s1);
        P(mutex);
        get_odd();
        V(mutex);
        V(s3)
        count_odd();
    }
}
P3(){
    while(true){
```

```
        P(s2);
        P(mutex);
        get_even();
        V(mutex);
        V(s3);
        count_even();
    }
}
```

5. 如下图所示，三个合作进程P_1、P_2、P_3，它们都需要通过同一设备输入各自的数据 a、b、c，该输入设备必须互斥地使用，而且其第一个数据必须由P_1进程读取，第二个数据必须由P_2进程读取，第三个数据必须由P_3进程读取（读取后所有进程都可以使用）。然后，三个进程分别只能对输入数据进行下列计算：

$$P_1: x = a + b$$
$$P_2: y = a * b$$
$$P_3: z = y + c - a$$

最后，P_1进程通过所连接的打印机将计算结果 x、y、z 的值打印出来。请用信号量实现它们的互斥与同步。



设置三个信号量 s1,s2,s3,为了确保读取数据的顺序为 P1、P2、P3，设置 s1 的信号量初值为 1，s2，s3 初值为 0。设置信号量 s4,s5 用于指示 y 的输入和 y、z 是否计算完成，初值为 0

```
P1(){
    P(s1);
    输入 a;
    V(s2);
    P(s4);
    x = a + b;
    P(s6)
```

```
        打印 x,y,z
}
P2(){
        P(s2);
        输入 b;
        V(s3);
        V(s4);
        y=a*b;
        V(s5);
}
P3(){
        P(s3);
        输入 c;
        P(s5);
        z = y + c – a;
        V(s6);
}
```