1. 如果系统中有 N 个进程，那么运行进程最多几个，最少几个？就绪进程最多几个，最少几个？等待进程最多几个，最少几个？

在单处理器系统中，运行进程最多只有 1 个，如果是多处理器系统，则最多有与处理器数量相同的运行进程；最少为 0 个。

除去运行进程之外其余 n-1 个进程都可以是就绪进程；最少为 0 个即所有进程都处于阻塞态或运行态。

等待进程最多为 n 个，最少为 0 个。

2. 进程有无如下状态转换，为什么？

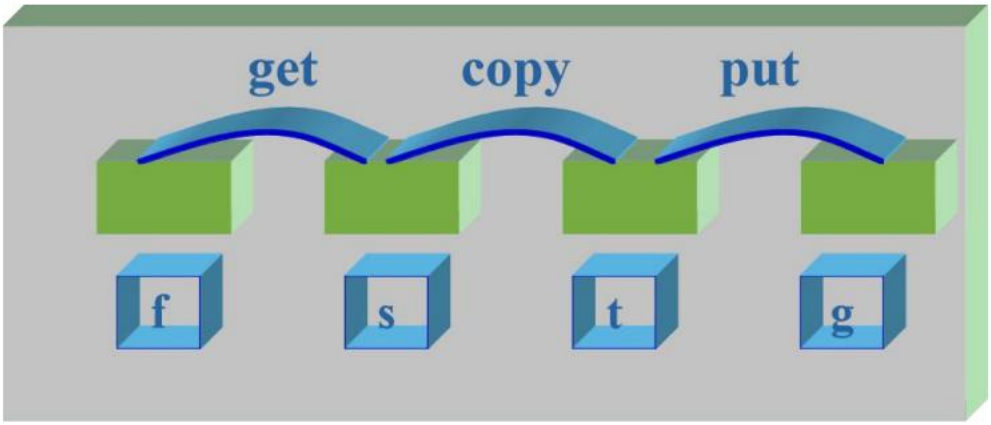
（1）等待—运行

（2）就绪—等待

没有阻塞态到运行态，因为进程处于阻塞态是因为处理机或其他资源被占用，当占用的资源被释放后，处于阻塞态的进程会先从阻塞队列中移至就绪队列，因此，进程必须先经过就绪态，等待 CPU 调度器选择它进入运行态。

没有就绪态到阻塞态，因为进入阻塞态是进程主动请求的，必然需要进程在运行时才能发出这种请求。

3. 用 P.V 操作解决下图之同步问题



Semaphore empty_s = 1, empty_t = 1, full_s = 0,full_t = 0, mutex_s = 1, mutex_t = 1;

进程 get:

```
While(1){
    P(empty_s);
    P(mutex_s);
    Get;
    Add to s;
    V(mutex_t);
    V(full_s);
}
```

进程 copy:

```
while (1) {
    P(full_s);
    P(mutex_s);
    Remove from s;
    V(mutex_s);
    V(empty_s);
    Copy:
    P(empty_t);
    P(mutex_t);
    Add to t;
    V(mutex_t);
    V(full_t);
}
```

进程 put:

```
While(1){
    P(empty_t);
    P(mutex_t);
    Put;
    Add to g;
    V(mutex_t);
    V(full_t);
}
```

4. 试从动态性、并发性、独立性和异步性上比较进程和程序。

1. 动态性：

程序：程序是静态的代码文件，通常是指令和数据的有序集合。它是一个不随时间变化的固定存在，描述了完成某项任务的指令序列。在程序未运行时，它仅仅是静态的文件，不会主动执行或做任何操作。

进程：进程是程序的执行实例，是动态的。当程序被加载到内存中并运行时，就变成了进程。进程是一个动态实体，能够在系统中执行操作，占用系统资源，并在其生命周期内不断变化状态。

进程的实质是程序在多道程序系统中的一次执行过程，进程是动态产生，动态消亡的。

2. 并发性：

程序：程序本身是不能并发的，因为它是静态的指令集。

进程：进程具有并发性，多个进程可以在同一时刻被系统调度器交替执行，在多处理器系统上，多个进程可以真正并行运行。

3. 独立性：

程序：程序作为静态文件，可以被多个进程同时使用和加载，彼此之间没有直接联系。

进程：每个进程是独立的执行实体，拥有自己的地址空间、资源和执行路径。

进程之间是相互隔离的，一个进程的崩溃或异常不会直接影响其他进程的执行。

4. 异步性：

程序：程序是指令的集合，异步执行并不是程序的本质特性。程序在运行时如果没有多任务或多线程机制，执行过程是按部就班的顺序进行的，无法表现出异步性。

进程：进程具有异步性，因为多个进程在操作系统的调度下可以按照不同的速度执行，它们之间的执行顺序不一定是确定的。由于进程间的相互制约，使进程具有执行的间断性，即进程按各自独立的、不可预知的速度向前推进。

5. 为什么进程在进入临界区之前应先执行“进入区”代码？而在退出前又要执行“退出区”代码？请说明原因。

进入区代码的目的是确保互斥访问，即在同一时刻，只有一个进程能够进入临界区，防止多个进程同时操作共享资源；退出区代码的目的是在进程离开临界区后释放锁定，允许其他等待的进程进入临界区，从而避免资源长期被一个进程独占。

6. 设 P、Q、R 共享一个缓冲区，P，Q 构成一对生产者和消费者，R 既为生产者又为消费者，使用 P，V 操作实现三个进程同步。

```
Semaphore mutex = 1, empty = n, full = 0;
进程 P:
While(1){
    Produce;

    P(empty);
    P(mutex);
    Add to buffer;
    V(mutex);
    V(full);
}

进程 Q:
While(1){
    P(full);
    P(mutex);
    Remove from buffer
    V(mutex);
    V(empty);
    Consume;
}

进程 R:
While(1):{
```

```
        P(full);
        P(mutex);
        Remove from buffer
        V(mutex);
        V(empty);
        Consume;
        Produce;
        P(empty);
        P(mutex);
        Add to buffer;
        V(mutex);
        V(full);
}
```