Q1

进程是描述程序执行过程和资源共享的基本单位
主要目的：控制和协调程序的执行

狭义定义：进程是正在运行的程序的实例（an instance of a computer program that is being executed）。

广义定义：进程是一个具有一定独立功能的程序关于某个数据集合的一次运行活动。它是操作系统动态执行的基本单元，在传统的操作系统中，进程既是基本的分配单元，也是基本的执行单元。

进程的概念主要有两点：第一，进程是一个实体。每一个进程都有它自己的地址空间，一般情况下，包括文本区域（text region）、数据区域（data region）和堆栈（stack region）。文本区域存储处理器执行的代码；数据区域存储变量和进程执行期间使用的动态分配的内存；堆栈区域存储着活动过程调用的指令和本地变量。第二，进程是一个“执行中的程序”。程序是一个没有生命的实体，只有处理器赋予程序生命时（操作系统执行之），它才能成为一个活动的实体，我们称其为进程。 [3]

进程是操作系统中最基本、重要的概念。是多道程序系统出现后，为了刻画系统内部出现的动态情况，描述系统内部各道程序的活动规律引进的一个概念,所有多道程序设计操作系统都建立在进程的基础上。

Q2

代码段 A 最后会有 32 个进程，代码段 B 最后会有 6 个进程

因为一个进程调用 fork（）函数后，系统先给新的进程分配资源，例如存储数据和代码的空间。然后把原来的进程的所有值都复制到新的新进程中，只有少数值与原来的进程的值不同。相当于克隆了一个自己。代码段 A 最开始的父进程和每个新产生的子进程都在循环里，每次循环每个进程都会再分出来 1 个进程，所以是 2^5=32 个进程，代码段 B 每次 fork() 的子进程都会跳出循环，只有父进程留在循环里，所以最后有 6 个进程。

Q3

在 unix/linux 中，正常情况下，子进程是通过父进程创建的，子进程在创建新的进程。子进程的结束和父进程的运行是一个异步过程,即父进程永远无法预测子进程 到底什么时候结束。 当一个 进程完成它的工作终止之后，它的父进程需要调用 wait()或者 waitpid()系统调用取得子进程的终止状态。

僵尸进程：一个进程使用 fork 创建子进程，如果子进程退出，而父进程并没有调用 wait 或 waitpid 获取子进程的状态信息，那么子进程的进程描述符仍然保存在系统中。这种进程称之为僵死进程。

进程一旦调用了 wait，就立即阻塞自己，由 wait 自动分析是否当前进程的某个子进程已经退出，如果让它找到了这样一个已经变成僵尸的子进程，wait 就会收集这个子进程的信息，并把它彻底销毁后返回；如果没有找到这样一个子进程，wait 就会一直阻塞在这里，直到有一个出现为止,当然 如果在调用 wait（）时子进程已经结束，则 wait（）就会立即返回

子进程结束状态值。　　参数 status 用来保存被收集进程退出时的一些状态，它是一个指向 int 类型的指针。但如果我们对这个子进程是如何死掉的毫不在意，只想把这个僵尸进程消灭掉，我们就可以设定这个参数为 NULL，pid = wait(NULL)；如果成功，wait 会返回被收集的子进程的进程 ID，如果调用进程没有子进程，调用就会失败，此时 wait 返回-1，同时 errno 被置为 ECHILD。

如果参数 status 的值不是 NULL，wait 就会把子进程退出时的状态取出并存入其中，这是一个整数值（int），指出了子进程是正常退出还是被非正常结束的，以及正常结束时的返回值，或被哪一个信号结束的等信息。由于这些信息被存放在一个整数的不同二进制位中，所以用常规的方法读取会非常麻烦，人们就设计了一套专门的宏（macro）来完成这项工作：

1，WIFEXITED(status) 这个宏用来指出子进程是否为正常退出的，如果是，它会返回一个非零值。

2， WEXITSTATUS(status) 当 WIFEXITED 返回非零值时，我们可以用这个宏来提取子进程的返回值，如果子进程调用 exit(5)退出，WEXITSTATUS(status) 就会返回 5；如果子进程调用 exit(7)，WEXITSTATUS(status)就会返回 7。请注意，如果进程不是正常退出的，也就是说， WIFEXITED 返回 0，这个值就毫无意义。

Q4

信号：进程通讯机制

信号是发送给进程的特殊异步消息

当进程接收到信息时立即处理，此时并不需要完成当前函数调用甚至当前代码行

Linux 系统中有多种信号，各具有不同的意义；系统以数字标识不同的信号，程序一般以名称引用之

系统信号

缺省处理逻辑：终止进程，生成内核转储文件

使用“kill-l”命令可查看操作系统支持的信号列表，不同的系统可能有所不同

信号用来通知进程发生了异步事件。在软件层次上是对中断机制的一种模拟，在原理上，一个进程收到一个信号与处理器收到一个中断请求可以说是一样的。信号是进程间通信机制中唯一的异步通信机制，一个进程不必通过任何操作来等待信号的到达，事实上，进程也不知道信号到底什么时候到达。进程之间可以互相通过系统调用 kill 发送软中断信号。内核也可以因为内部事件而给进程发送信号，通知进程发生了某个事件。信号机制除了基本通知功能外，还可以传递附加信息。

收到信号的进程对各种信号有不同的处理方法。处理方法可以分为三类：

第一种是类似中断的处理程序，对于需要处理的信号，进程可以指定处理函数，由该函数来处理。

第二种方法是，忽略某个信号，对该信号不做任何处理，就像未发生过一样。

第三种方法是，对该信号的处理保留系统的默认值，这种缺省操作，对大部分的信号的缺省操作是使得进程终止。进程通过系统调用 signal 来指定进程对某个信号的处理行为。

Q5

未决状态：发送的信号被阻塞，无法到达进程，内核就会将该信号的状态设置为未决

作用：

信号屏蔽机制专门用于解决常规信号不可靠这一问题。在进程的 PCB 中存在两个信号集，一个称为信号掩码(signal mask)，另一个称为未决信号集(signal pending)。这两个信号集的实质都是位图，其中的每一位对应一个信号：若 mask 中某个信号对应的位被设置

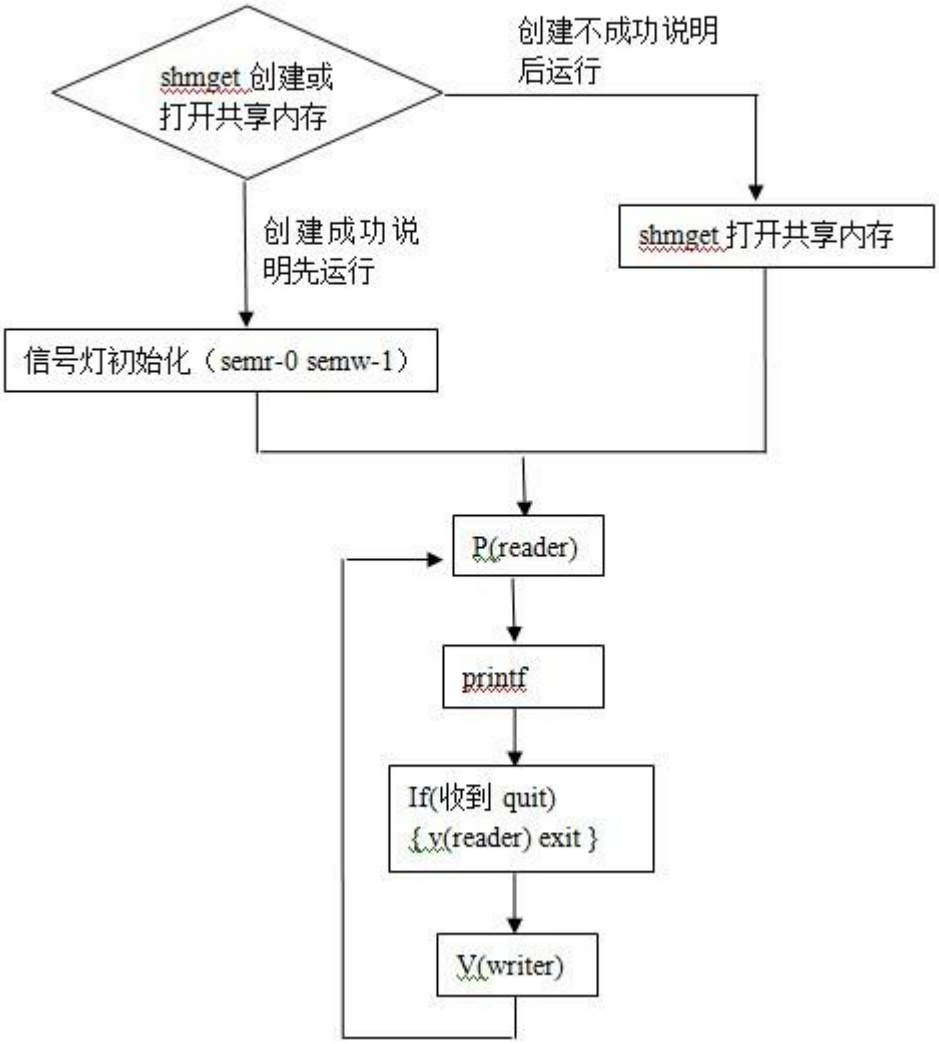
为 1，信号会被屏蔽，进入阻塞状态；此时内核会修改 pending 中该信号对应的位为 1，使该信号处于未决态，之后除非该信号被解除屏蔽，否则内核不会再向进程发送这个信号。

信号发生的时候到递送之间的时间间隔，我们称为信号时未决的。在未决的信号期间，如果我们阻塞了某些信号，那么，即使我们产生信号，这些都会被阻塞。除非 等解除阻塞以后，系统才会处理这些信号。而且，阻塞期间多次产生的同一个信号，解除阻塞后，系统处理且只处理一次。

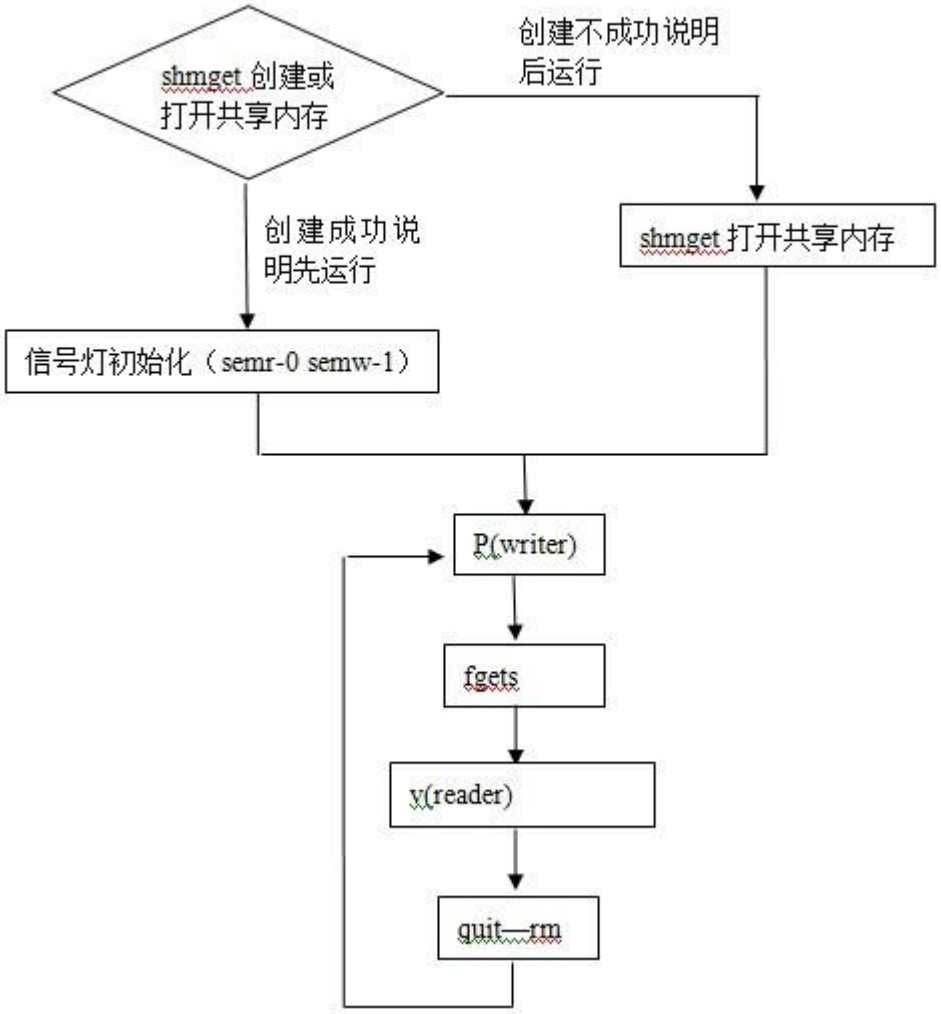
Q6

两个进程，对同一个共享内存读写，可利用有名信号量来进行同步。一个进程写，另一个进程读，利用两个有名信号量 semr, semw。semr 信号量控制能否读，初始化为 0。 semw 信号量控制能否写，初始为 1。

reader 流程图


shmget 创建或
打开共享内存
创建不成功说明
后运行
创建成功说
明先运行
shmget 打开共享内存
信号灯初始化（semr-0 semw-1）
P(reader)
printf
If(收到 quit)
{ v(reader) exit }
V(writer)

writer 流程图


shmget 创建或
打开共享内存
创建不成功说明
后运行
创建成功说
明先运行
shmget 打开共享内存
信号灯初始化（semr-0 semw-1）
P(writer)
fgets
v(reader)
quit—rm