Week11 Assignment

> 班级：
> 学号：
> 姓名：

阅读教材第七章，回答以下问题。

1. 简述信号量的作用，如何利用信号量实现同步和互斥?

> 信号量是进程之间同步与互斥的一种机制，通过它可以实现对共享资源的保护，防止多进程或多线程的并非带来的不一致问题。信号量是一个变量，对应于某一种资源，取一个非负的整型值，信号量值指的是当前可用的该资源的数量，那么当一个进程使用某个资源，首先进行P操作，如果信号量>0则使用该资源并将信号量-1，否则阻塞进程，直到有资源可用。在进程使用完资源后进行V操作，如果在该信号量的等待队列中有进程在等待该资源，则唤醒一个进程，否则释放一个资源（信号量+1），这样就能保证在有限资源的前提下实现同步和互斥。

2.简述共享内存的作用和用法，共享内存为什么需要与信号量一起使用?

> 共享内存允许不相关的进程访问同一个逻辑内存，是实现进程间通信的一种方式。任何可访问该段共享内存的任何进程均可读写该数据，共享内存的优点是会映射到进程的虚拟地址空间，进程可以对其直接访问，避免了数据复制过程。是可用的最快速的进程间通信机制。用法：
>
> 1）创建共享内存
>
> 2）映射共享内存
>
> 3）撤销映射
>
> 因为共享内存机制不提供对该内存在多个进程之间的保护机制，因此程序开发人员必须使用信号量或者其他机制实现对共享内存的同步或者互斥访问。

3. 有以下代码：

```
int fd1,fd2,fd3,fd4;
fd1=open("a.txt",O_RDONLY);
fd2=open("b.txt",O_WRONLY);
fd3=dup(fd1);
fd4=dup2(fd2,0);
```

请问，最后fd1,fd2,fd3,fd4的值分别是多少？并解释原因。

> 3 4 5 0
>
> 每个进程都有一个独立的文件描述符表， 文件描述符是一个从0开始的索引，通过该索引可以找到对应文件。其中0 1 2三个索引已经默认被标准输入、标准输出、标准错误输出占用了。新产生的文件描述符索引位置是最小的还没有被占用的索引，因此fd1，fd2是3，4。fd3复制了fd1的描述符，赋给了新的描述符5，此时fd3、fd1都指向相同的文件a.txt，虽然他们的值不同(3和5)。fd4先关闭了0对应的文件描述符即标准输入，将df2复制，赋给了新的描述符，此时，最小尚未分配的索引是0，因此答案是3 4 5 0。

> 注意，dup与dup2只有当执行失败时才会返回-1。

4. 请查阅资料简述进程间通信的 System V、POSIX 两种标准之间的差异性

> System V只能用于有父子进程关系或统一进程内多个线程之间实现同步与互斥。而POSIX可用于线程，也可用于相关甚至不相关进程。
>
> 与 System V IPC 接口不同，POSIX IPC 接口均为多线程安全接口。
>
> POSIX 在无竞争条件下，不需要陷入内核，其实现是非常轻量级的; System V 则不同，无论有无竞争都要执行系统调用，因此性能落了下风。
>
> 总体来说，System V IPC存在时间比较老，许多系统都支持，但是接口复杂，并且可能各平台上实现略有区别。POSIX比较轻量级。

5. 编写C程序实现如下功能：创建父子进程，父子进程之间通过管道进行通信，父进程向子进程发送字符串，子进程收到该字符串后，将该字符串的最后5个字符发送给父进程。

```c
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include<signal.h>
#include <sys/types.h>
#include <sys/wait.h>
#include<unistd.h>
#include <string.h>
#define BUFSIZE 22
void child(int read_pipe, int write_pipe) {
    char str[100];
    read(read_pipe, str, 100);
    printf("[Child] get message: %s\n", str);
    int len = strlen(str);
    for (int i = 0;i < 5;i++) str[i] = str[len - 5 + i];
    write(write_pipe, str, 5);
}
void parent(int read_pipe, int write_pipe) {
    char str[100];
    printf("[Parent] input a message: ");
    scanf("%s", str);
    write(write_pipe, str, strlen(str));
    read(read_pipe, str, 5);
    str[5] = 0;
    printf("[Parent] Receive returning: %s\n", str);
}
int main() {
    //使用两个无名管道，一个用于父进程把字符串发给子进程，一个用于子进程提取后发回来。
    //因为当管道内无任何东西可读时，read将被阻塞，保证了顺序。
    //也可以仅使用一个管道，但需要用一些手段保障父子进程顺序，例如wait。
    int fd_parent_write[2], fd_child_write[2], pid;
    pipe(fd_parent_write);
    pipe(fd_child_write);
    if (pid = fork()) parent(fd_child_write[0], fd_parent_write[1]);
```

```
    else child(fd_parent_write[0], fd_child_write[1]);
    return 0;
}
```