课后习题

chap1

1. 计算机系统组成/层次结构
   - 组成： 软件+ 硬件
   - 层次结构：
     - 应用软件
     - 支撑软件
     - OS（系统软件）
     - 计算机硬件
2. 计算机系统资源
   - 信息资源 ： 数据 + 程序
   - 硬件资源： 处理器/外设/内存
3. OS定义/作用 ❎
   - 定义：操作系统是管理计算机系统资源，控制程序执行，优化人机界面，提供各种服务，合理组织计算机工作流程，为用户方便有效地使用计算机提供良好的运行环境
   - 作用：
     - 服务用户的观点：OS作为用户接口和公共服务程序
     - 进程交互的观点：OS作为进程执行的控制者和协调者
     - 系统实现的观点：OS作为扩展机或虚拟机
     - 资源管理的观点：OS作为资源的管理者和控制者
4. 什么是系统调用，分类 ❎
   - 系统调用把应用程序的请求传送至内核，调用相应服务例程完成所需处理，姜结果返回给应用程序。

    - 作用：
        - 内核可以基于权限和规则对资源访问进行裁决，保证系统的安全性
        - 系统调用对资源进行抽象，提供一致性接口，避免用户在使用资源时发生错误，且是编程效率大大提高。
    - 分类：
        - 进程管理
        - 文件管理
        - 设备管理
        - 存储管理
        - 进程通信
        - 信息维护
5.

chap2

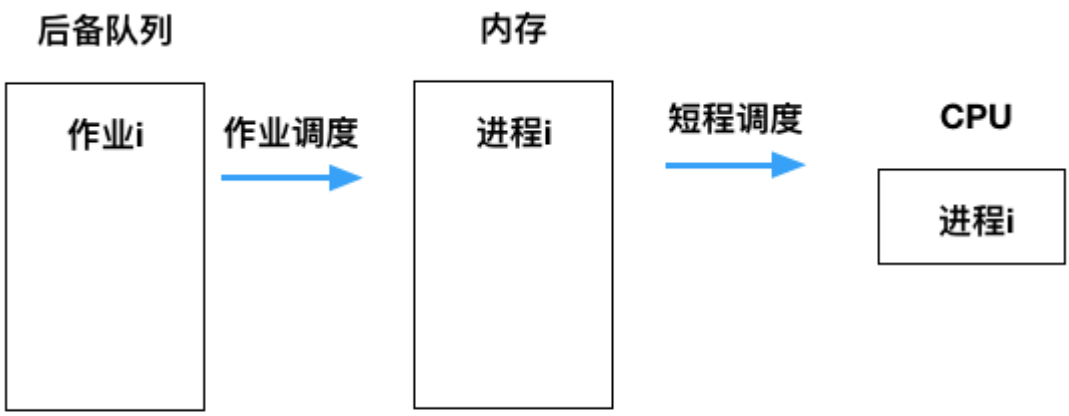
http://www.docin.com/p-961208247.html

http://www.docin.com/p-83382947.html

https://wenku.baidu.com/view/3b9878bda98271fe900ef96d.html

注意⚠️：

- 响应时间：从交互式进程提交一个请求（命令）到获得响应之间的时间间隔
    - 所输入的请求命令送到CPU的时间
    - CPU处理请求命令的时间
    - 处理所形成的响应送回到终端显示器的时间
- 周转时间包括：从向系统提交作业开始 到作业完成为止
    - 在后备队列中等待的时间
    - 进程进入内存后在就绪队列中等待的时间
    - ..
    - 直到执行完成



道数是指内存（就绪队列+等待队列+CPU正在运行的）中可以容纳的作业数目

- 时间片轮转算法中：同一时刻时一个进程时间片完,另一个到达,那个会先调用?
  - 先调用刚到达的
  - 相当于，先把刚到达的放入就绪队列尾部，再把刚刚用完时间片的进程放入就绪队列尾部。
- ❓ 作业调度和进程调度时：某个时刻，正在运行的作业执行完成，内存有空位，此时既需要从后备作业中调度一个作业进入内存，又要从就绪队列中调度一个进程占有处理器运行。而即将调入作业可能会影响进程调度的结果（可能即将调如的作业优先级最高），那么是先进行作业调度，还是进行进程调度

PV管程 -- 阅览室问题

```
int number[100];
string name[100];
semaphore seatcount = 100;
semaphore mutex;

cobegin
process reader_i(string name){
    P(seatcount);
    P(mutex);
        //进入阅览室
    V(mutex);
    V(seatcount);

}
```

```
Type read_book = monitor{
    int seat_count = 0;
    semaphore seat = 0;
```

```
    int reader_count = 0;

    InterfaceModule IM;
    define reader_come,reader_leave;
    use enter,leave,wait,signal;
}
void reader_come(InterfaceModule &IM){
    enter(IM);
    if(reader_count>100){
        wait(seat,seat_count,IM);
    }
    reader_count++;
    //进入阅览室
    leave(IM);
}
void reader_leave(InterfaceModule &IM){
    enter(IM);
    reader_count--;
    signal(seat,seat_count);
}

cobegin
void reader_i(){
    read_book.reader_come();
    //
    read_book.reader_leave();
    //
}
```

PV管程 --- 捡棋子

```
{
    p(s1);

    V(s2);
}
{
    P(s2);

    V(s1);
}
```

```
Type pick_chess = monitor{
    semaphore white,black;
    int white_count,black_count;
    bool flag;//true:捡白棋子，false：捡黑棋子
    InterfaceModule IM;
    define pickup_white,pickup_black;
```

```
    use enter,leave,wait,leave;
}
void pickup_black(){
    if(flag){
        wait(black,black_count,IM);
    }
    //pick up black
    flag = true;
    signal(white,white_count,IM);
}
void pickup_white(){
    if(!flag){
        wait(white,white_count,IM);
    }
    //
    flag = false;
    signal(black,black_count,IM);
}
```

PV 管程 -- 信箱问题

```
full0 = 0;
empty0 = 3;
full1 = 0;
empty1 = 3;
full2 = 0;
empty = 2;
full3 = 0;
empty3 = 2;

mutex0 = mutex1 = mutex2 = mutex3 = 0;
process0(){
    P(full0);
    P(mutex0);
    //
    V(mutex0);
    V(empty0);
    //process
    P(empty1);
    P(mutex1);
    //
    V(mutex1);
    V(full1);
}

/////////
full[4] = {0};
empty[4] = {3,3,2,2};
mutex[4] = {0};
```

```
process_i(){
    P(full[i]);
    P(mutex[i]);
    //
    V(mutex[i]);
    V(empty[i]);
    //process
    P(empty[(i+1)%4]);
    P(mutex[(i+1)%4]);
    //
    V(mutex[(i+1)%4]);
    P(full[[(i+1)%4]]);
}
```

```
type message = monitor{
    count[4] = {0};
    limit[4] = {3,3,2,2};
    cond p[4];
    int p_count[4];
    InterfaceModule IM;
    define take_i,sent_i;
    use enter,leave,wait,signal;
}
void take_i(InterfaceModule &IM){
    if(count[i]<=0){
        wait(p[i],pcount[i],IM);
    }
    //
    signal(p[(i-1)%4],pcount[(i-1)/4],IM);
}
void send_i(InterfaceModule &IM){
    if(count[(i+1)%4]>=limit[(i+1)%4]){
        wait(p[i],pcount[i],IM);
    }
    //
    signal(p[(i+1)%4],pcount[(i+1)/4],IM)
}
```

PV - X-Y

```
要求:
    -N < X-Y < M
semaphore X_Y = m-1;
semaphore Y_X = n-1;
semaphore mutex = 1;
void X(){
    while(1){
        P(X_Y);
```

```
        P(mutex);
        //
        V(mutex);
        V(Y_X);
    }
}
void Y(){
    while(1){
        P(Y_X);
        P(mutex);
        //
        V(mutex);
        V(X_Y);
    }
}
```

PV -- 零件

```
semaphore A_B = n-1;
semaphore B_A = n-1;
semaphore empty = m;//二者加起来不超过 m
semaphore emptyA = m-1,emptyB = m-1;//任何一个都不能超过 m-1，否则无法各拿一个进行组装
semaphore fullA = 0,fullB = 0;
semaphore mutex;//互斥访问临界区
void A(){
    while(true){
        P(A_B);
        P(emptyA);
        P(empty);
        P(mutex);
        //
        V(mutex);
        V(empty);//删掉
        V(fullA);
        V(B_A);
    }
}
void B(){
    while(true){
        P(B_A);
        P(emptyB);
        P(empty);
        P(mutex);
        //
        V(mutex);
        V(empty);//删掉
        V(fullB);
        V(A_B);
    }
```

```
}
void get(){
    while(true){
        P(fullA);
        P(fullB);

        P(mutex);
        //get A
        V(mutex);
        P(mutex);
        //get B
        V(mutex);

        V(emptyA);
        V(emptyB);
        V(empty);
        V(empty);
    }
}
```

T29 PV -- 发送消息

```
//可以想象有n2个缓冲区
semaphore full[n2] = {0};
semaphore empty[n2] = {m};
semahpore mutex = 1;
send(){
    for(int i = 0;i<n2;i++){
        P(empty[i]);
        P(mutex);
        //
        V(mutex);
        V(full[i]);
    }
}
receive(int i){
    P(full[i]);
    P(mutex);
    ///
    V(mutex);
    V(empty[i]);
}
A_i(){
    while(1){
        send();
    }
}
B_i(){
    while(1){
```

```
        receive(i);
    }
}
```

chap4

思考题

1. 存储管理
    - 存储分配
    - 地址映射
    - 存储保护
    - 存储共享
    - 存储扩充
2. 存储层次
    - 寄存器
    - 缓存
    - 内存
    - 磁盘
    - 磁带
3. 逻辑地址/物理地址
4. 分区分配策略
    - 固定分区
        - 优点：
            - 能解决单道程序运行在并发环境下不能与CPU速度匹配问题
            - 单道程序运行时内存空间利用率低低问题
        - 缺点：
    - 可变分区

31. 缺页中断与一般中断的区别

    7.试比较缺页中断机构与一般的中断，他们之间有何明显的区别？

    答：缺页中断作为中断，同样需要经历保护 CPU 现场、分析中断原因、转缺页中断处理程序进行处理、恢复 CPU 现场等步骤。但缺页中断又是一种特殊的中断，它与一般中断的主要区别是：
    （1）在指令执行期间产生和处理中断信号。通常，CPU 都是在一条指令执行完后去检查是否有中断请求到达。若有便去响应中断；否则继续执行下一条指令。而缺页中断是在指令执行期间，发现所要访问的指令或数据不在内存时产生和处理的。
    (2)一条指令在执行期间可能产生多次缺页中断。例如，对于一条读取数据的多字节指令，指令本身跨越两个页面，假定指令后一部分所在页面和数据所在页面均不在内存，则该指令的执行至少产生两次缺页中断。

chap5

1. spooling和作业调度的关系

答: SPOOLING 系统是用一类物理设备模拟另一类物理设备的技术，是使独占使用的设备变成多台虚拟设备的一种技术，也是一种速度匹配技术。作业调度程序根据预定的调度算法选择收容状态的作业运行，作业表是作业调度程序进行作业调度的依据，是 SPOOLING 系统和作业调度程序共享的数据结构。

2. 各种IO控制方式及其优缺点

①程序直接控制方式: 由用户来直接控制内存或 CPU 和外围设备之间的数据传送。

它的优点是: 控制简单，也不需要多少硬件支持

它的缺点是: CPU 和外围设备只能串行，设备串行，无法发现和处理由于设备或其他硬件所产生的错误。

②中断控制方式: 利用向 CPU 发送中断的方式控制外围设备和 CPU 之间的数据传送

优点: 大大提高了 CPU 的利用率，支持多道 程序和设备并行。

缺点: 占用大量 CPU 时间，中断次数多，发生中断丢失的现象，数据丢失现象。

③DMA 方式：在外围设备和内存之间开辟直接的数据交换通路进行数据传送，

优点：在数据传送开始需要 CPU 的启动指令，结束时发中断通知 CPU 进行中断处理之外，不需要 CPU 的干涉。

缺点：在外围设备越来越多的情况下，多个 DMA 控制器的同时使用，会引起内存地址的冲突并使得控制过程进一步复杂

④通道方式：使用通道来控制内存或 CPU 和外围设备之间的数据传送，通道是一个独立于 CPU 的专管 I/O 的机构，控制内存与设备直接进行数据交换，有自己的通道指令。这些指令受 CPU 启动，并在操作结束时向 CPU 发中断信号

优点：减轻 CPU 的工作负担，增加了并行工作程度

缺点：增加额外的硬件，造价昂贵。

3.

(1)由于只有输出设备，因此设计这个系统时只须要考虑 SPOOLing 系统的缓输出部分即可。系统组织如下：磁盘中一个足够大的缓冲区(输出井)，用于存放系统和用户的打印作业；一个缓冲区管理程序(井管理程序)；一个后台打印程序(缓输出程序)。

(2)采用 SPOOLing 系统后，两台打印机可供若干个系统或用户进程同时使用。根据题意，可设立 3 个并发进程完成两台打印机的使用，它们分别是输出井管理进程、系统打印进程和网络打印进程。为此，输出井中设立了两个缓冲区队列，一个用于存放系统打印作业，一个用于存放网络打印作业，两个队列分别需要互斥使用。同时系统打印进程还要和网络打印进程通信，以决定一般用户可否使用系统使用的打印机。具体程序实现如下：

```
begin
var s1,s2:semaphore;
var sys - pcount,net_pcount:integer;
s1=1;        //系统打印队列互斥信号量
s2=1;      //网络打印队列互斥信号量
sys_pcount=0;      //系统打印作业计数器
net_pcount=1;      //网络打印机作业计数器
      cobegin
          buffer=manageQ      //打印缓冲区管理进程
          begin
              接收一个打印作业：
              if(系统打印作业)
                  P(s1);
                  放入系统打印队列;
                  sys_pcount:=sys_pcount+1;
                  V(s1);
               else
                  P(s2)
                  放入网络打印队列;
                  net_pcouny:=net_pcount+1;
                  V(s2);
          end;
          system_print()
              begin
                  P(s1);
                  if(sys_pcount＞0)
                      取出系统打印队列一个打印作业打印；
                      sys_pcount:=sys_pcount-1;
                  else
                      P(s2);
                      取出网络打印队列一个打印作业打印：
                      net_pcount:=net_pcount-1;
                      V(S2);
                      V(S1);
                end1。
      user_Print()
      begin
          P(s2);
          if(net_pcount＜＞0)
              取出网络打印队列一个打印作业打印；
              net_pcoun:=net_pcount-1;
              V(S2)
      end;
       coend;
end;
```



```
semaphore S_sys = S_net = 1;
int sys_count = 0;
int net_count = 0;
void manage(){
    //接收一个打印作业
    if(系统打印作业){
        P(S_sys);
        //放入系统打印机队列
        sys_count++;
```

```
        V(S_sys);
    }else{
        P(S_net);
        //放入网络打印机队列
        net_count++;
        V(S_net);
    }
}
void System_Print(){

}
```