同济大学课程考核试卷（A 卷）
2022—2023 学年第一学期

命题教师签名：邓蓉　　审核教师签名：

课号：10102002　　课名：操作系统　　考试考查：参考答案

此卷选为：期中考试(　　)、期终考试(∨)、重考(　)试卷

一、填空、选择和判断题（25*2 = 50 分）

1、Unix V6++的进程控制块是 Process 结构和 User 结构，文件控制块是 Diskinode 或 Inode;（填内核数据结构）。　放进程图像的盘交换区在（　B　）?
磁盘高速缓存在（　A　）？

A，内存　　B、磁盘　　C、都可能

2、Unix V6++系统，磁盘数据块 512 字节。文件的 d_addr 数组管理着 6 个直接块，2 个一次索引块和 2 个二次索引块。现运行进程 PA 成功打开磁盘文件 example1。已知该文件长 400K 字节，所有磁盘数据缓存不命中。请问，访问偏移量是 100 的字节需要读入 1 个磁盘数据块；访问偏移量是 10K 的字节需要读入 2 个磁盘数据块；访问偏移量是 132K 的字节需要读入 3 个磁盘数据块；随后，访问偏移量是 132K+1 的字节需要读入 1 个磁盘数据块。注意，本题读入指的是磁盘 IO，所有数据缓存不命中。

3、T0 时刻，系统中有两个进程 P1 和 P2，分别独立打开并同时访问小文件 example2。则在内存打开文件结构中有（　A　）个内存 i 节点指向该文件？（　B　）个 file 结构记录着进程对文件的访问情况？

A. 1　　B. 2

在哪个数据结构中登记有进程对文件的访问方式（读或读写）？（　B　）
文件的读写指针保存在（　B　）？
组成文件的每个逻辑盘块（信息块）在磁盘上的地址保存在（　AC　）？

A. 内存 i 节点　　B. file 结构　　C. i_addr 数组

若 P2 进程向文件追加写入 10000 个字符后关闭该文件，引发（　A）操作；稍后，P1 关闭 example2 文件，引发（　ABC　）操作。

A. 释放 file 结构　　B. 释放内存 i 节点
C. 将内存 i 节点写回磁盘　　D. 不执行任何操作

4、现运行进程 PA SRUN，正在执行系统调用 getpid。T1 时刻，响应中断，唤醒一个内存中的睡眠进程 PB。PB 进程何时上台运行？ PA 进程执行完系统调用（如果同学强调，在此次 CPU 陷入核心态运行期间，有可能响应多次中断，PB 要等优先级比它高的内核任务全部运行完毕才能运行，正确，酌情加分）

现运行进程 PA SRUN，正在执行系统调用 read。T2 时刻，响应中断，唤醒另一个内存中的睡眠进程 PC。PC 进程何时上台运行？ PA 进程执行 read 系统调用入睡时（如果同学强调，如果 read 系统调用没有导致 PA 入睡，PA 进程 read 系统调用返回时 PC 才能运行，正确，酌情加分）

判断题。现运行进程放弃 CPU 后，被调度程序选中的一定是优先级最高的进程。 错 一定

是优先级最高的就绪进程

5、死锁是指在系统中的多个进程无限期地等待永远不会发生的事件（填空）。产生死锁的四个必要条件是：互斥、B、循环等待和不剥夺（选择）。

A. 请求与阻塞　B. 请求与保持　C. 请求与释放　D. 释放与阻塞

读者写者问题，读者优先的解在什么情况下会饿死写进程在最后一次读任务结束前，总有新的读任务到来，那么程序会持续运行读任务，从而无法执行写任务。写进程被饿死时，系统有没有死锁？系统没有死锁，因为读进程在跑，没有所有进程阻塞

6、一个磁盘组有 100 个柱面，每个柱面有 8 个磁道（磁道号=磁头号=磁面号），每根磁道 8 个扇区，每个扇区 512 字节。同根磁道，相邻磁盘数据块存放在相邻物理扇区。

（1）文件 A 存有 640 条记录，记录从 0 开始编号，每条记录 512 字节。文件从 0 柱面、0 磁道、0 扇区顺序存放。请问，128#记录存放在2#柱面，0#磁道，0#扇区（填柱面号，磁道号和扇区号）。

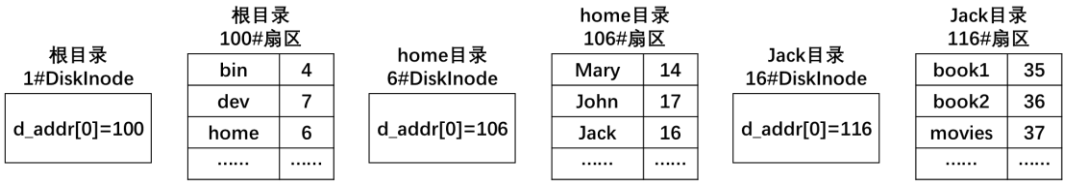
（2）文件 B 存有 64 条记录。下面的哪种物理结构，文件顺序读写性能较优？B

A、64 条记录，存在同一磁面（一个圆盘的同一面）

B、64 条记录，存在同一柱面

为什么？因为物理结构 B 只需 1 次寻道，即可读入全部记录。物理结构 A，需要 8 次寻道。

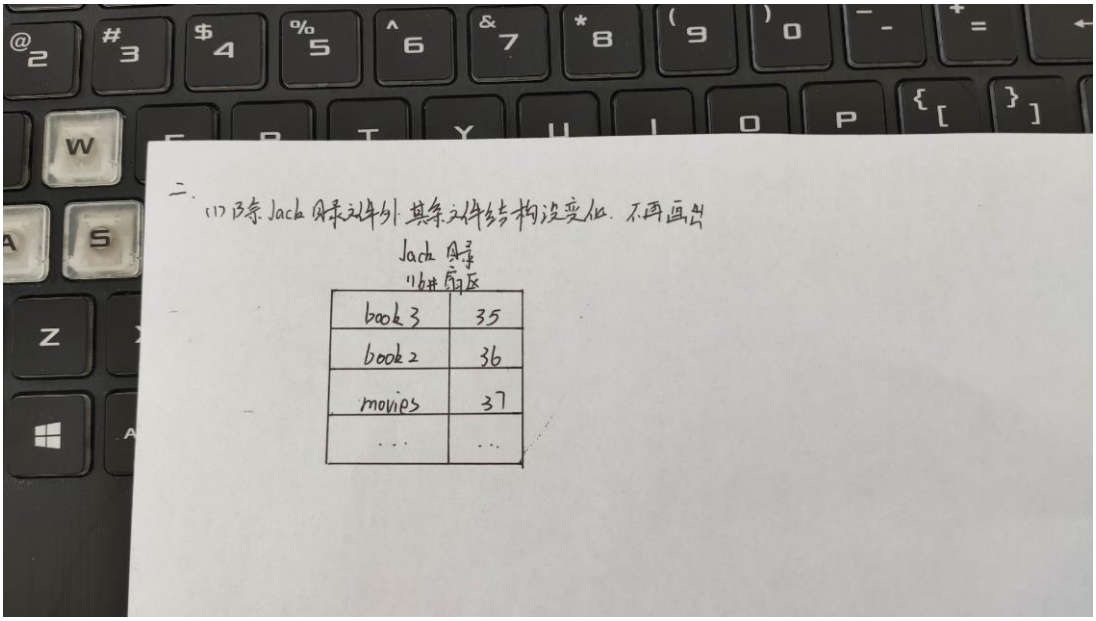
二、（9 分）某 UNIX V6++系统中，文件系统的目录结构如下图所示：



T0 时刻，SuperBlock 中空闲 inode 栈保存有 85 个空闲 inode 号。进程 PA 删除 d_link==1 的文件/home/Jack/book1。已知，book1 文件长 1500 个字节，占用磁盘上的 122#扇区（0#逻辑块）、123#扇区（1#逻辑块）和 125#扇区（2#逻辑块）。随后的 T1 时刻，系统创建第 1 个新文件：/home/Jack/book3，向其中写入字符串“Hello World！\0”，关闭该文件。已知，（T0,T1]时间区间，系统不曾对文件系统实施任何操作。

（1） 请填空：创建文件 book3 后，文件系统的目录结构（3 分） 和 文件 book3 的关键字段（3 分）。

创建文件 book3 后，文件系统的目录结构如下所示：



文件 book3 的关键字段 d_addr[0]=125，因为这是系统回收的最后一个数据块。

（2）（3 分）Unix V6++的这种空闲资源管理方式有哪些缺点？优势在哪里？

这种栈式空闲资源管理方式，优势在于：

1、 系统利用 SuperBlock 和 空闲数据块自身管理空闲资源，无需单独设置磁盘上的数据结构，降低了文件系统元数据的存储开销。

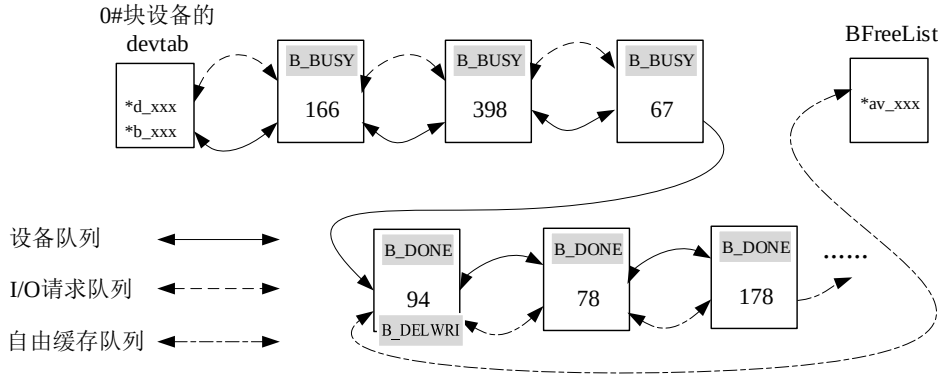
2、 空闲资源分配释放速度极快。算法复杂度 O(1)。

缺点在于：

1、很难为文件分配连续数据块。

2、栈底资源几乎得不到利用，存储资源利用不均衡。

三、（6 分）T0 时刻，系统缓存使用情况如下图所示。以下 2 小题彼此独立。



请回答：

（1）（3 分）如果此时进程 pa 读取该设备上的 166 号数据块，请描述缓存的分配过程。进程 pa 在缓存分配（执行 GetBlk 函数）的过程中是否会睡眠？如果会，何时被唤醒，如果不会，请说明理由。

1. 缓存的分配过程：PA 进程在设备缓存队列中线性搜索，166 块数据块缓存命中。但，目标缓存块在 IO 请求队列，B_BUSY 是 1。所以，PA 进程入睡，等待持锁进程释放缓存时被唤醒。

2. 在缓存分配（执行 GetBlk 函数）的过程中进程 PA 会睡眠。原因在于，目标缓存块在 IO 请求队列里，锁是关的（B_BUSY==1），PA 需要入睡等待持锁进程解锁时被唤醒。

（2）（3 分）如果此时进程 pa 读取该设备上的 94 号数据块，请描述缓存的分配过程。进程 pa 在缓存分配（执行 GetBlk 函数）的过程中是否会睡眠？如果会，何时被唤醒，如果不会，请说明理由。

1. 缓存的分配过程：PA 进程在设备缓存队列中线性搜索，94#数据块缓存命中。目标缓存块位于自由缓存队列，PA 进程上锁成功，可以直接使用缓存块中数据。
2. 在缓存分配（执行 GetBlk 函数）的过程中进程 PA 不会睡眠，原因在于缓存块的锁是开的，在自由缓存队列中。

四、（5 分）假定磁盘的移动臂现在正处在第 12 柱面，有如下 6 个请求等待访问磁盘。假设寻道时间>>旋转延迟。请列出最省时间的响应次序：

序号	柱面号	磁头号	扇区号
(1)	9	6	3
(2)	7	5	6
(3)	15	20	6
(4)	9	4	4
(5)	20	9	5
(6)	7	15	2

1. 因为寻道时间>>旋转延迟，所以主要考虑使用磁臂调度算法对寻道时间进行优化；
2. 利用电梯调度算法结合初始的柱面号 12，由于 12-7=5<20-12=8，所以选择小端为初始运动方向；
3. 最优柱面访问顺序为 12，9，7，15，20 得到最省时间的响应次序为{(1),(4)},{(2),(6)},(3),(5)，大括号内的请求因同处同一个柱面，可以互换。

   也可以是 {(2),(6)},{(1),(4)}, (3),(5)　　这是 CSCAN 的思路。

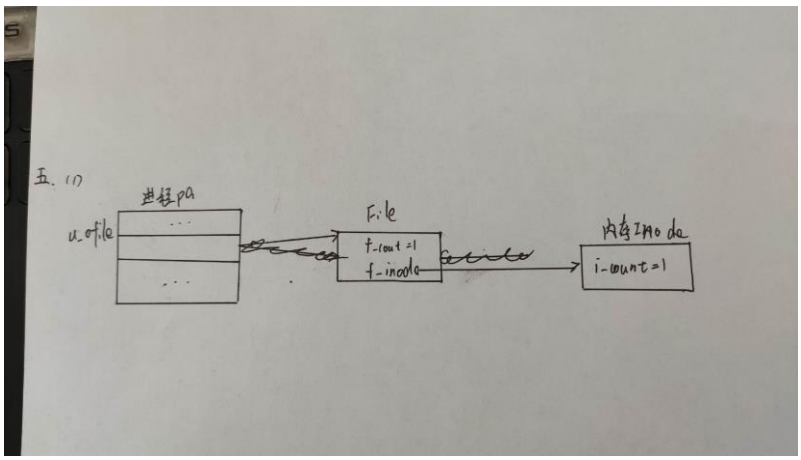
五、（15 分）假如文件 abc 大小为 600 字节。程序运行时需要引用的所有磁盘数据缓存不命中。

```
main( )
{
    int fd = open("abc", O_RDWR);   //以可读可写方式打开文件
    char data[600];
    int count = read ( fd, data, 600);
    write ( fd, data, count);
    ……
}
```

请回答以下问题：

（1）open 系统调用会引发 IO 操作吗？ 会 如果会，进程会从磁盘读入 DiskInode。请绘制这个进程的打开文件结构。

打开文件结构如下所示：



（2）read 系统调用会引发预读操作吗？会。为什么？因为执行 open 系统调用打开文件后初始 i_lastr==-1,而 read 系统调用第一次循环需要读入的逻辑块号为 0#，因为 i_lastr+1=第一次循环需要读入的逻辑块号，所以满足预读条件，需要预读下一个逻辑块 1#

（3）这段代码对文件实施的是顺序读写还是随机读写？顺序读写。为什么？原因在于整个对文件的操作过程中不涉及利用 lseek 或 seek 人为改变文件读写指针，文件读写指针的变化均是执行 read 和 write 系统调用产生。

（4）请简述本题代码中 write 系统调用的执行过程。

1. 整体运行情况概括：将 data 数组中的 600（read 系统调用成功读取 abc 文件的 600 个字节，返回值 count 为 600）个字节追加写到 abc 文件尾部，写操作完成后，文件长度增加至 600+600=1200 个字节
2. 运行细节，共需两次循环写两个块，并且需要给 2#逻辑块分配新扇区，初始状态 m_offset=f_offset=600;
   a) 第一次循环：
      i. 当前逻辑块号 bn=1，从快内地址（600-512）=88 开始写，写入 511-88+1=424 个字节；
      ii. 计算当前物理块号，假设为 x，由于没有写满，所以需要先读后写，bread(x)；
      iii. 又因为上一次 read 系统调用刚刚使用完此缓存块,所以此时该缓存块一定可用(假设没有其他进程正在使用该缓存块)，因此缓存命中，当前进程锁住该缓存块，不睡；
      iv. 当前进程执行 IOmove,将用户空间 data 数组中的 data[0]~data[423]写入该缓存块 88#~511#字节；
      v. 缓存写到底部，将该缓存送入 IO 队列尾部，将 1#逻辑块对应的 x 物理块异步写回磁盘（IO 完成后，中断处理程序会解锁该缓存）；
      vi. 当前进程不会睡，继续写下一块，调整 m_offset=424+600=1024;
   b) 第二次循环：
      i. 当前逻辑块 bn=2，从块内地址 0 开始写，写入（600-424）=176 个字节；
      ii. 计算当前物理块号，得 0，所以系统需要为 2#逻辑块分配新的物理块号，假设块号为 new，即 i_addr[2]=new；
      iii. 因为此时写入 176 个字节，没有写满，所以需要先读后写，bread(new);
      iv. 进程此时会锁住分配给 new 物理块的缓存块 buffer[i],并且会睡眠等待同步读结束；
      v. 当前进程执行 IOmove,将用户空间 data 数组中的 data[424]~data[599]写入该缓存块 0#~175#字节；

vi. 因为没有写到该缓存块 buffer[i]的底部，所以延迟写，置 B_DELWR，此时现进程便会释放该缓存块 buffer[i];

vii. 最终 Write 系统调用结束，调整 m_offset=1024+176=1200,m_count=0,文件长度 L=1200(m_offset),系统调用 write 返回，返回值为 600；

（5）如果 write 系统调用执行完毕后系统断电，文件系统会出现什么情况。应当怎样加以弥补。以此题为背景论述缓存技术的优势和需要付出的代价。

1. 文件系统会出现最后延迟写的部分没有落盘保存（缓存中的数据掉电丢失），需要重新执行本进程操作，从而实现预期效果；
2. 缓存技术的优势在于可以大幅度减少 IO 次数，从而提升磁盘系统吞吐量，缩短进程读写磁盘数据的平均耗时。但同时也带来了数据不一致性的风险，可以通过定期写盘和缓存恢复系统等方法减少此不良影响。

六、（10 分）T0 时刻，某 UNIX V6++系统进程状态如下表所示。内存空间已满，除图示空间外，其余空间不可用。请简述以下时刻系统中与进程调度和对换操作（swap in，swap out）相关的行为，并修改下表中的相关字段。

序号	占用空间	状态	位置	内存起始地址
0#	-	高睡（RunOut）	SLOAD	****
p1	50K	低睡	SLOAD	0x00500000
p2	40K	执行	SLOAD	0x00550000
p3	40K	低睡	~SLOAD	0x00600000

（1）T0 时刻（非整数秒），现运行进程 p2 执行 read 系统调用读磁盘文件（磁盘高速缓存中没有 p2 需要的文件数据）。修改后的表格如下：

序号	占用空间	状态	位置	内存起始地址
0#	-	高睡（RunOut）	SLOAD	****
p1	50K	低睡	SLOAD	0x00500000
p2	40K	高睡(SSLEEP)	SLOAD	0x00550000
p3	40K	低睡	~SLOAD	0x00600000

1. P2 进程执行 read 系统调用高优先权入睡放弃 CPU，此时现运行进程变为高优先权入睡的 0#进程，0#进程执行 select 函数，内存中一个 SRUN 进程都没有，系统 Idle。

（2）T1（T1=T0+1 秒）时刻，已完成 read 系统调用的 p2 进程运行在用户态。p3 等待的 I/O 操作完成。修改后的表格如下（P2 和 P3，一个执行，一个就绪。随意排列）：

序号	占用空间	状态	位置	内存起始地址
0#	-	高睡（RunOut）	SLOAD	****
p1	50K	低睡	~SLOAD	0x00500000
p2	40K	执行	SLOAD	0x00550000
p3	40K	就绪	SLOAD	0x00500000

2. T1 秒 p2 进程执行中断处理程序，同时唤醒 p3 和 0#进程。中断处理程序执行完毕后，p2 将 cpu 让给 0#进程执行 sched()函数。0#进程会去为盘交换区上的 p3 进程分配内存空间，首先找低睡 SWAIT 或 SSTOP 的进程，找到 p1 进程，0#进程首先换出 p1 进程，之后换进盘交换区上的 p3 进程，就放在 p1 进程之前的内存区域；
3. 完成对换操作后，0#进程执行 sleep(&Runout,-100)入睡，执行 Swtch 函数将 cpu 让给内存中就绪的 p3 进程，p3 进程执行系统调用的后半部分，完成后 p2 和 p3 开始时间片轮转。

七、简答题（5 分）。系统调用和普通的子程序调用有什么不同？请问：UNIX V6++的系统调用如何

系统调用调用前是用户态，调用后在核心态；而普通的子程序调用，调用前后，进程运行模式不变。

a） 传递 APP 想要执行的系统调用号？ 在钩子函数中将系统调用号给 EAX(内联汇编语句)

b） 传递系统调用的参数？ 在钩子函数中赋给 EBX,ECX,EDX,ESI,EDI，saveContext 将其放入核心栈，系统调用处理函数将这些参数放入 u_args，之后需要时便从 u_args 中取；

c） 将系统调用的返回结果传给应用程序？ 系统调用的返回结果存在 EAX 中，需要返回的其他信息放在钩子函数中传入的指针所指向的用户内存空间区域。

八、写代码（10 分）。父进程 PA 执行 fork 系统调用创建 2 个子进程 P1 和 P2。所有进程输出 p_pid 和 p_ppid。程序输出结果如下，可以不按下图次序输出。如果能够确保如下输出次序，加 5 分。

parent: pid = PA 的 pid，ppid = _____shell 进程的 pid 号_____。

first child: pid = P1 的 pid，ppid = PA 的 pid。

second child: pid = P2 的 pid，ppid = PA 的 pid。

```
#include   <stdio.h>
#include   <sys.h>
int main()
{
   int i,j;
   if(i=fork())
   {
        printf("parent:pid = &d,ppid=&d",getpid(),getppid());
        if(i=fork())   // 第 2 个子进程
        {
             i=wait(&j);     // 睡眠等待第 1 个子进程终止
             i=wait(&j);     // 睡眠等待第 2 个子进程终止
        }
        else
```

```
        {
            sleep(200);
            printf("second child: pid =&d,ppid=&d",getpid(),getppid());
            exit(2);
        }

    }
    else     // 子进程 1
    {
        sleep(100);
        printf("first child: pid =&d,ppid=&d",getpid(),getppid());
        exit(1);
    }
    return 0;
}
// 要防止父进程终止后，子进程的 ppid 变成 1.所以父进程要 wait 2 次。
```

通过 sleep 函数控制，能够确保和题目要求输出次序一样。

参考答案 2　周宏韬

```
int main(){
    int i,j;
    if(i=fork()){
        printf("parent: pid =%d，ppid =%d",i,getppid());
        if(i=fork()){
            sleep(100)//等待子程序完成避免自己提前结束，子进程的父亲变成 1#
        }
        else{
            sleep(50);//最后执行
            printf("second child: pid =%d，ppid =%d",getpid(),getppid());
        }
    }
    else{
            sleep(25);
            printf("first child: pid =%d，ppid =%d",getpid(),getppid());

    }
}
```

九、（10 分）共享打印间，一台打印机供顾客自助使用。长凳上有 5 个位子供顾客休息等待。顾客完成打印任务后，离开系统，长凳上的下一位顾客起身使用打印机。顾客来到打印间时，如果有空位子则坐下来；如果没有则必须离开。

样例代码 1，得 5 分。错在 2 处，（1）共享打印间内，顾客数量上限不应该是 5，应该是 6。（2）

自助打印，没有 printer 进程。

样例代码 1

```
semaphore customers,printer,mutex
customer.value=0;
printer.value=1;
mutex.value=1;

#define chair=5
int waiting =0;

void printer()
{
    While(true)
    {
    P(customer);
    print();
    V(printer);
    }
}
void customeri()
{
    P(mutex);
    if(waiting<chair)
    {
        waiting++;
        V(customer);
        V(mutex);
        P(printer);
        print();
        P(mutex);
        Waiting--;
        V(mutex);
    }
    else
    {
        V(mutex);
        离开;
    }
}

void main()
{
    parbegin(printer,customer0,customer1,…)
```

```
}

// 我的
#define CHAIR 5
Semaphore waiting=0;
Semaphore mutex=1;
Semaphore print=1;

void customeri()
{
    P(mutex);
    If(waiting<CHAIR){
        waiting=waiting+1;
        V(mutex);

        P(print);
        P(mutex);
        waiting=waiting-1;
        V(mutex);
        print files;
        V(print);
        leave；
    }
    else{
        V(mutex);
        leave;
    }
}

void main()
{
    parbegin(customer0,customer1,…)
}

//周宏韬的解
semaphore machine，mutex,customer
machine.value=1, mutex.value=1, customer.value=0;
int waiting=0;
#define seatnum 5
void printmachine(){
    p(cusomer);//服务一个客户
     p(mutex);
    waiting--;
    v(mutex);
```

```
    v(machine);//这句话也只能在这里，因为在下面自主打印完前放这句话会使得机器没运行，顾客也能打印
}
void customerFunc(){
    p(mutex);//mutex 负责保护 waiting
    if(waiting== seatnum){
        v(mutex);
        离开;
    }
    waiting++;
    v(mutex);
    p(machine);
    自主打印;
    v(customer);//这句话必须放在这里，顾客占用机器打印完成后才能让 waiting--
    离开;
}
void main(){
    beginerun（printmachine，customerFunc）;
}
```

十、（10 分）打印机是必需互斥使用的外设。并发 2 个打印任务，两个输出内容交织在一起，打印结果是不可用的。请设计一个系统，将原先必需互斥使用的打印机改造成能够同时为多个进程提供打印服务的共享设备。

- 系统维护一个 FIFS 的打印队列。
- 进程需要打印时，
  - 新建一个临时磁盘文件，命名之。把要打印的内容写入这个文件。
  - 生成一个打印作业控制块，包含进程 ID，用户 ID，临时文件名……，送打印队列尾。
  - 进程返回。无需等待打印 IO 完成。
- 打印机完成当前打印任务后，取打印队列队首打印作业控制块，构造针对打印机的新的 IO 命令。

磁盘，相对打印机，是很快的设备，所以，打印内容存入磁盘比看到打印结果，快很多很多。此外，磁盘 IO 借助磁盘高速缓存。后者在内存中，可以并发接纳多个打印机进程的输出结果。随后，打印进程会在后台串行从打印队列中取出打印任务（是中断处理程序驱动的），将存在高速缓存里的数据送至打印机。如此，借助磁盘高速缓存，上述 Spooling 系统便将一个必需互斥访问的硬件设备，改造成了可供多个进程共享使用的共享设备。

伪代码，参考答案：　　张睿，参考硬盘中断处理逻辑

伪代码如下：

打印机运行程序：

```
while(1){
if(获得打印请求){
if(不在打印)
```

{开始打印}
}
else{
加入打印缓存
}

if(打印任务完成){
下一个打印任务
if(有缓存打印任务)
弹出顶端打印任务
开始打印
}
else{
空转
}
}