2023-2024期末题

一.简答题(30/20分，每题5分)

1.解释线程和进程的关系。

一个进程有多个线程。
一个线程**最多只能**在一个进程空间中的移动。
进程对应的是资源（有地址等）。
同一个进程的所有线程共享该进程所有资源。
CPU对应给线程。

2.一个进程的虚拟地址空间中，数据段、代码段、堆段、栈段是否都可以被进程的各线程共享?

2.解释“一切皆文件”思想，并说明其优缺点。

是将所有在Linux/类Linux系统中将所有的设备、操作等都视为文件。
优点：统一接口，方便使用；
缺点：效率问题：设备抽象成文件，一定程度上带来效率问题。

3.画出线程的五个基本状态转移图，并注明状态转移条件。

4.外设与主机交互主要有哪四种方式，请简要介绍。

直接查询、轮询、DMA、中断。

二.程序运行（10分）

对照以下代码回答

```
int a = 1;                  // ①

int fadd(x){
        int b = 1;          // ②
        b = x + b;
        return b;
}

int main(){
        int c = 2;          // ③
        c = fadd(a);
        printf("Hello number %d\n", c);   // ④
        return 0;
}
```

（1）第①、②、③、④行运行时分别位于哪个段
（用.text, .rodata, .rwdata, .zidata, .heap, .stack 回答）

（2）对照 c=fadd(a)语句，请画出过程调用的全景。

> *(1）读写段、栈段、栈段、代码段。*
> *(2）调用开始->保存寄存器->读取参数\初始化局部变量->调整栈框..... （看PPT）*

三.页面调度

得分	阅卷人

三、 页面调度（10 分）

假设某进程按照 0，1，2，3，4，3，2，1，0，1，2，3 的顺序访问虚拟页面，且操作系统为该进程准备了 3 个可供请求分页的物理页面。假设初始时该进程没有任何物理页面。请回答下列问题：

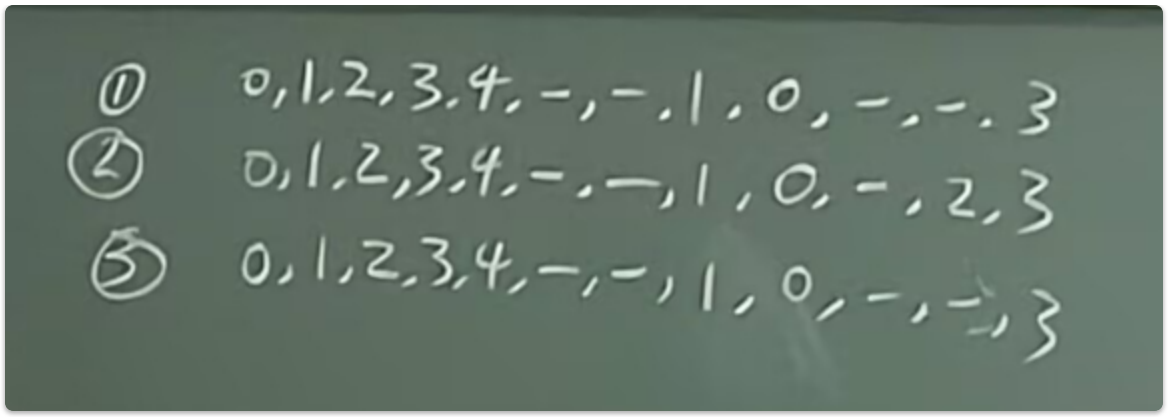
（1）按照最长前向距离（LFD，即最优）页面替换算法，列出每次访问时调入的虚拟页面的页号（若某次访问不调入页面则写“-”，答案格式为“0，1，-，…，”）。

（2）按照先进先出（FIFO）页面替换算法，列出每次访问时调入的虚拟页面的页号（答案格式同上）。

（3）按照最久未用（LRU）页面替换算法，列出每次访问时调入的虚拟页面的页号（答案格式同上）。

（4）增加资源总量（比如内存总量、页面数）反而导致性能下降和效率降低的现象，称作什么现象？

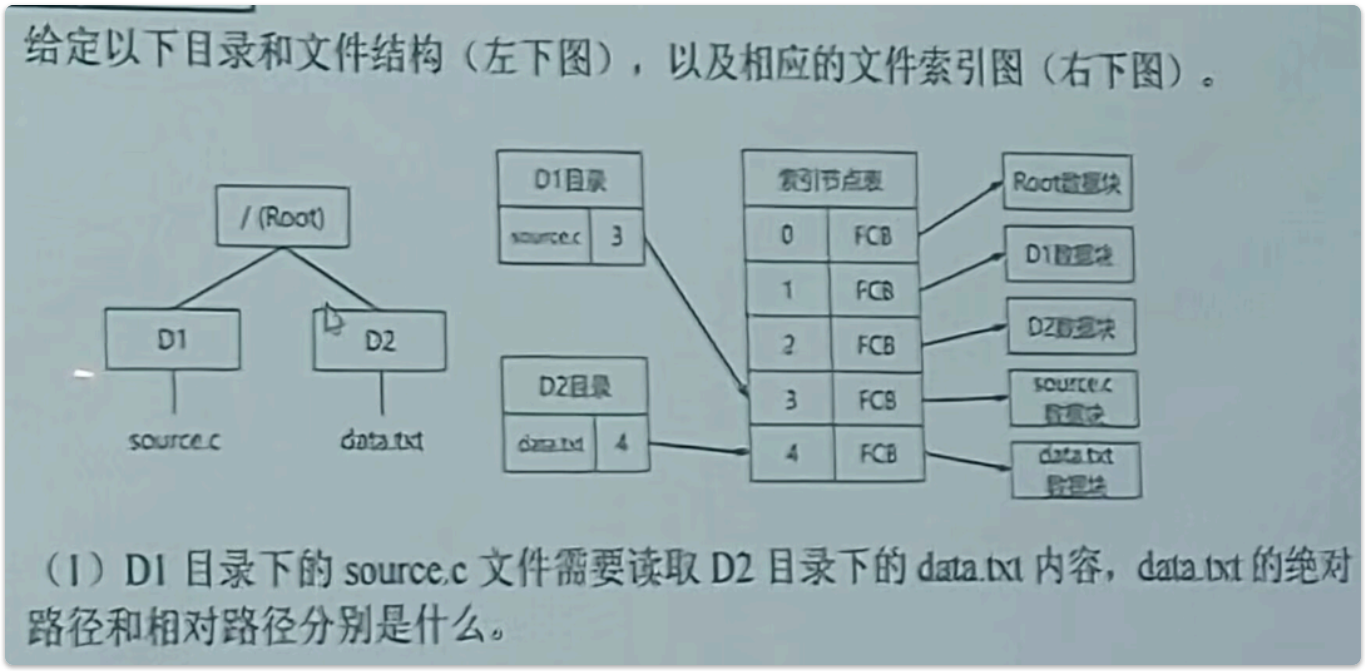
（5）LFD（最优）页面替换算法为什么难以在现实的系统中落地使用



（4）belady

（5）核心要答出离线，要用到未来的信息。

四.文件系统



（2）移动文件：/D2/data.txt -->/D1/data.txt，参考文件右上图，画出文件索引图。

（3）移动文件和复制文件哪个更慢，从文件系统的角度。

复制。复制要创建新的数据块，创建新的索引节点表。

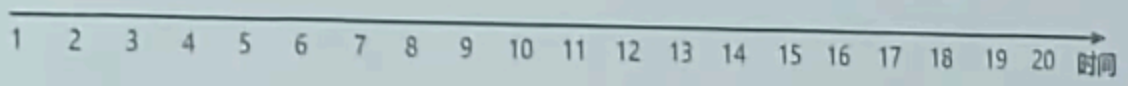
（4）FCB与数据块的索引有多种方式（比如连续分配、链表分配、索引分配等），请解释混合索引的思想。

五.线程的调度

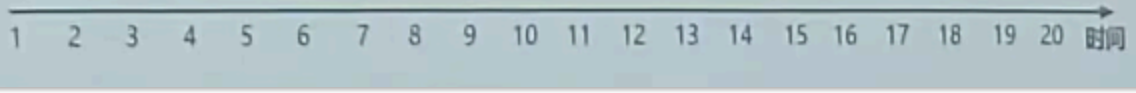
三个线程E1、E2、E3，参数如下，提交给操作系统调度。

线程	到达时间	运行时长
E1	1	5
E2	3	3
E3	5	2

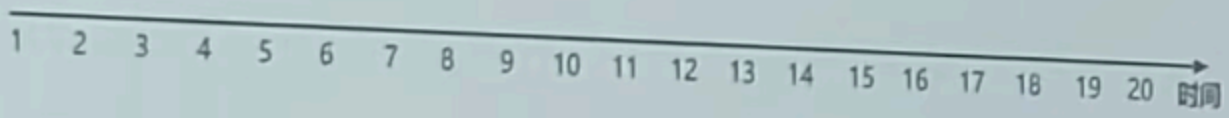
（1）先到先服务（FCFS）。画出调度情况。



（2）短作业优先（SJF），抢占式。画出调度情况。



（3）时间片轮转法（时间片长度为2）。画出调度情况。



（4）先到先服务、短作业优先、时间片轮转法，哪些是在线算法？

（5）线程调度属于短期调度、中期调度、还是长期调度？

（4）都是。

（5）CPU调度都是短期的（线程），内存调度都是中期的（页面）。

六.用户态和内核态

（1）这两种模式有什么区别?
能否执行特权指令。
（2）这两种模式对现代操作系统的设计及有什么帮助?
更为安全。
（3）printf（“Hello World\n”）会不会陷入内核态执行？为什么?
会。因为他用到IO指令（访问到了硬件）。
函数中的计算是不会陷入的。
（4）库结构、宏内核、微内核、外核结构，哪些不区分内核模式与用户模式?
库结构不区分。

七.中断

假设一台主机接收到的网络下载速度为1Gbps，网络数据包的大小为10000b，CPU响应一个中断的时间是1微秒。
（1）中断的频率是多少?
10的5次。
（2）CPU为了处理中断的占用率?
百分之10
（3）如果每收到100个数据包产生一次中断，CPU的占用率是多少?
百分之0.1
（4）问题（3）中的方法的问题是什么？如何解决?
系统的响应比较慢，容易造成时延过长。
解决方法：带超时时间的中断节流。

八.死锁

假设系统的当前资源分配状况如下表所示。

资源表	已分配 U			最大 M			剩余量 A		
指令流	R1	R2	R3	R1	R2	R3	R1	R2	R3
A	1	1	0	7	5	3	3	3	2
B	2	0	0	3	2	3			
C	3	0	2	9	0	2			
D	2	1	1	2	3	2			
E	0	0	2	4	3	3			

（1）系统当前是否处于安全状态？（是否死锁）如果是，请给出安全执行顺序，在其中指令流尽量按照从E到A的顺序执行完毕（优先字母大的先执行）DEBCA

（2）在当前执行情况下，指令流A请求分配（0，2，1），是否允许此次分配。如果可以，请给出安全执行顺序。不行。

（3）死锁的必要条件是什么？三个/四个。

（4）银行家算法可以避免死锁，他是摒弃掉了哪一个死锁的必要条件？循环等待。

同步

某进程中有 3 个工作线程 T1、T2 和 T3，其伪代码如下所示。

```
// 复数的结构类型定义
typedef struct
{
   float a;
   float b;
} cnum;

cnum x, y, z; // 全局变量

// 计算两个复数之和
cnum add( cnum& p, cnum& q )
{
   cnum s;

   s.a = p.a + q.a;
   s.b = p.b + q.b;

   return s;
}
```

```
T1
{
   cnum w;

   while(1) //不断运行
   {
      w = add( x, y );
      …… //使用 w
   }
}

T2
{
   cnum w;

   while(1) //不断运行
   {
      w = add( y, z );
      …… //使用 w
   }
}
```

```
T3
{
   cnum w = {2.0f, 3.0f};

   //初始化全局变量
   x.a = 1.0f;
   x.b = 5.0f;
   y.a = 8.0f;
   y.b = 6.0f;
   z.a = 4.0f;
   z.b = 7.0f;

   while(1) //不断运行
   {
      //延时一小时
      delay( ONE_HOUR );
      //要求 xyz 一起更新
      x = add( x, w );
      y = add( y, w );
      z = add( z, w );
   }
}
```

（1）指出T1-T3中哪些线程是读者，哪些线程是写者，以及为何要互斥访问x，y，z。

T1，2是读者，T3是写者。

x，y，z是临界资源，临界资源不同时访问。

（2）请重写T1-T3线程的while循环，在 内部添加必要的互斥锁lock操作，保证程序正确执行。

> *先确定位置，一定是在while循环里，然后确定位置。*
> *读写锁完全与PPT一致。*

```
T1/T2{
    cnum w;
    while(1){
        rlock();
        w=add(x,y);
        runlock();
```

```
    }
}
T3{
    ......
    while(1){
        delay(ONE_HOUR);
        wlock();
        x=add(x,w);
        y=add(y,w);
        z=add(z,w);
        wunlock();
    }
}
//然后把课件中的读写锁写上
```

（3）补充要求-正确初始化：在（2）中，额外修改while循环之前的代码，使用信号量acquire和release操作，要确保全局变量初始化之前，读者不会启动读操作。

```
T1/T2{
    cnum w;
    P();
    while(1){
        rlock();
        w=add(x,y);
        runlock();
    }
}
T3{
    ......
    V();
    V();
    while(1){
        delay(ONE_HOUR);
        wlock();
        x=add(x,w);
        y=add(y,w);
        z=add(z,w);
        wunlock();
    }
}
```