考试科目名称 操作系统

考试方式： 闭卷 考试日期 2019 年 6 月 30 日 教师_____

系（专业）_____ 年级_____ 班级_____

学号_____ 姓名_____ 成绩_____

题号	一	二
分数		

得分	

一、综合题（共 42 分）

1. UNIX 系统中，若能不出错地运行如下代码：（8 分）

```c
#include<stdio.h>
#include<unistd.h>

int main(void)
{
    int i;
    for (i = 0; i < 4; i++)
        if (fork() == 0) break;
    printf("%d-%d: %d\n", getpid(), getppid(), i);
    sleep(1);
}
```

试回答如下问题：

（一）代码涉及了哪些 API 函数？简要讨论 API 函数与系统调用之间的区别与联系。

（二）简要描述 fork 的实现；

（三）代码运行时，最多创建了多少个进程？画出进程树（包括 shell 进程）。

（四）代码运行的输出结果是什么？（假设 Shell 进程的 PID 为 1000，其他进程的 PID 按创建先后顺序从 1001 开始按序分配）

说明：getpid()：获取当前进程的 pid；getppid()：获取当前进程的父进程 pid。

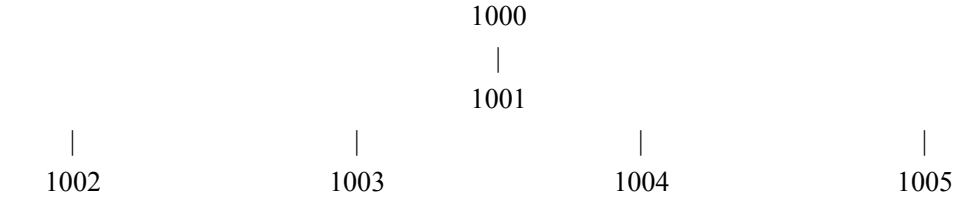
（一） fork(),printf(),getpid(),getppid(),sleep() （1 分）

系统调用是操作系统提供给用户访问内核空间的特殊接口，其对应的服务例程属于系统程序，在内核态运行；API 函数是应用程序接口，为应用程序开发者提供便携的功能支持。区别：系统调用必然访问内核态，而 API 函数强调的是如何通过接口来获得所需服务，部分 API 函数可以在用户态运行。联系：一个 API 函数根据是否需要访问内核态，可能不需要或需要一至多个系统调用来实现特定功能。（1 分）

（二）通过系统调用陷入内核态，寻找空闲 PCB 初始化，为子进程分配新空间并复制父进程内容，父进程返回子进程 pid，子进程返回 0，创建失败返回负数。（2 分）

（三）5 （1 分）

（1 分）



（四）顺序可以不同 （2 分）

1002-1001: 0

1003-1001: 1

1004-1001: 2

1005-1001: 3

1001-1000: 4

2. 一个具有**多道**作业的批处理系统，用户可使用的主存为 120KB，主存管理采用伙伴(Buddy)算法，作业调度采用先来先服务算法，进程调度采用时间片轮转法（平分 CPU 时间）。（5 分）

作业名	到达时间	估计运行时间（分钟）	主存需求(KB)
Job1	9:00	30	25
Job2	9:20	35	35
Job3	9:25	15	60
Job4	9:30	10	20
Job5	9:50	20	15

试回答如下问题：

（一） 列出各作业**进入主存时间**与结束时间；

（二） 画出**各关键时刻（发生变动时）**主存用户区分配情况。

说明：伙伴算法的基本思想是通过将用户内存区域对半分割，以实现最佳适应的分配的算法。

（一） 进入主存时间 结束时间 （3 分）

Job1	9:00	9:45
Job2	9:20	10:37:30
Job3	10:32:30	10:50
Job4	9:30	9:57:30
Job5	9:50	10:32:30

（二） （2 分）

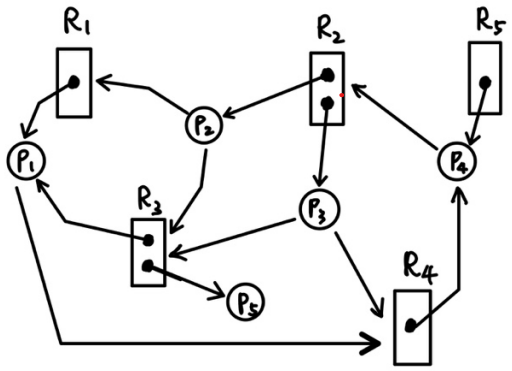
	30KB	30KB	60KB
9:00	Job1		
9:20	Job1		Job2
9:30	Job1	Job4	Job2
9:45		Job4	Job2

	15KB	15KB	30KB	60KB
9:50	Job5		Job4	Job2
9:57:30	Job5			Job2

	60KB	60KB
10:32:30	Job3	Job2
10:37:30	Job3	

	120KB
10:50	

3. 设系统中某时刻，各进程的资源占有和申请状况如下左图（进程-资源分配图）所示：




```
mutex A=1;
mutex B=1;

t1:                 t2:
P(A);               P(B);
P(B);               P(A);
...                 ...
V(B);               V(A);
V(A);               V(B);
```

试回答如下问题：（6 分）

（一）使用死锁检测算法判断该时刻是否发生死锁，若发生了死锁，则哪些进程陷入死锁？

（二）假设 P5 进程运行时会创建 t1 和 t2 两个线程，代码框架如上右图所示，此进程运行时，线程间是否有可能发生死锁？如果有可能发生死锁，发生死锁的概率大吗？运行过程中，操作系统有能力检测出此类死锁吗（简要说明理由）？

（一）发生死锁 P_1、P_2、P_3、P_4 （3 分）

（二）有可能（2 分） 极小 不能检测，不是操作系统管辖的资源 （1 分）

4. 一个 32 位系统的计算机，具有 2GB 物理内存，其上的操作系统采用请求式分页存储管理技术，页面大小为 4KB。试回答如下问题：（9 分）

（一） 简要评述分页式存储管理技术的主要缺点和应对手段？

（二） 如果采用反置页表，则反置页表最多包含多少页表项？简要评述反置页表的优点和缺点。

（三） 如果一个进程的逻辑地址（十进制）访问序列如下:4400、8500、3950、12500、8850、605、6200、16500、7100、12000，分配给该进程 3 个固定页框，分别为 10，50，55(页框按编号从小到大依次分配)，若采用 LRU 页面替换算法。则 1）给出对应的页面访问序列；2）画出页框中页面变化情况；3）接下来要访问的逻辑地址为 7001，给出对应的物理地址。

（四） LRU 页面替换算法在实际系统中能实现吗？为什么？ LRU 算法有哪些近似实现算法？（至少列举两种）

（五） 简要描述最佳页面替换算法，该算法有何意义？

（一）缺点：页表项过多占用空间大，访问速度慢。（1 分）

应对手段：采用多级页表、快表。（1 分）

（二）2^{19}。（1 分）

优点：只需为所有进程维护一张表。

缺点：仅包含调入内存的页面，不包含未调入的页面，仍需要为进程建立传统页表，存放在磁盘上，发生缺页异常时需要多访问一次磁盘，速度会比较慢。

（三）1，2，0，3，2，0，1，4，1，2。（1 分）

1-12-120-320-320-320-120-140-140-142

二进制：1010 1011 0101 1001

十进制：43865

十六进制：ab59 （1 分）

（四）不能实现 需要维护特殊队列，代价较大（1 分）

NRU、aging （1 分）

（五）当要调入一页而必须淘汰旧页时，应该淘汰以后不再访问的页，或距现在最长时间后才访问的页（1 分）

此理论算法可用做衡量各种具体算法的标准 （1 分）

5. 设某 UNIX 系统，文件系统的每个 inode 包含 10 个直接索引项和一、二、三级间接索引项各一个，物理块大小为 1KB，每个索引项占 4B。每个目录项占 16B（包含文件名和 inode 号）。inode 区占 5000 个扇区（每扇区 512B），每个 inode 占 64B，根目录区占 200 个扇区。（9 分）

   试回答如下问题：

   （一） 该文件系统根目录下最多能创建多少个文件或子目录（包括. 和..）？该文件系统最多能容纳多少个文件或目录？

   （二） 如果某个目录下有 610 个文件和 40 个子目录，则该目录文件占多少个物理块（列出计算过程）？

   （三） 若能成功执行 Shell 命令“ln /users/tom/a.txt /users/ben/a.txt”，则在命令执行过程中需要读取哪些文件的 inode，需要修改哪些文件的 inode？需要新增 inode 吗?

   （四） Shell 命令“ln -s /users/tom/a.txt /users/ben/a.txt”，与（三）中 Shell 命令有何不同？并简要讨论这两者的优劣；

   （五） 简要描述 UNIX 系统中 open 系统调用的内核实现过程。（结合文件描述符、用户已打开文件表、系统已打开文件表、活动 inode 列表等概念）

   说明：ln 命令用于创建链接文件。

（一）6400　40000 （1 分）

（二）650*16=10400B 需要 11 个物理块，还需要 1 个存储一级间接索引，共 12 个 （2 分）

（三）需要读取 users、tom、ben、/users/tom/a.txt 文件的 inode

需要修改/users/tom/a.txt 文件的 inode

不需要新增 （2 分）

（四）前一问是硬链接，这里是软链接（符号链接）（2 分）

硬链接只能用于单个文件系统，却不能跨越文件系统，可用于文件共享但不能用于目录共享，其优点是实现简单，访问速度快

软链接的优点是能用于链接计算机系统中不同文件系统中的文件，也可用于链接目录，进一步可链接计算机网络中不同机器上的文件，这种方法的缺点是搜索文件路径的开销大，需要额外的空间查找存储路径

（五）通过系统调用陷入内核，查找目标文件对应的 inode 节点，如果未找到该文件，则进行出错处理，如果找到该文件，若它已被其他用户打开，对应 inode 已在活动 inode 表中，否则创建系统打开文件表 file 结构表项，并在活动 inode 表中分配表项，再用磁盘 inode 填充其内容并用指针进行连接，最后将打开文件的 file 结构的指针安装到用户打开文件表中已分配的表项处，返回文件描述符 （2 分）

6. 设有一个包含了 16 个磁头(编号 0-15)和 200 个柱面(编号 0-199)的磁盘，每磁道扇区数 200 个(编号 0-199)，每个扇区 512B，磁盘的转速为 6000rpm（转每分钟），相邻柱面间的移动臂移动时间为 1ms。试回答如下问题:：（5 分）

（一） 若此时磁头位于 20 号柱面，刚刚完成 19 号柱面访问，依次到来如下磁盘访问请求（CHS 格式：柱面、磁头、扇区）：(12, 0, 20)，(25, 10, 25)，(12, 11, 150)，(78, 15, 55)，(12, 12, 75)，(101, 7, 101)，(197, 8, 20)，(92，5, 5)，(12, 11, 10)。如果移动臂调度算法采用电梯调度算法，同一柱面的请求进行优化排序，请给出这些磁盘访问请求重排序的结果，并大致估算总的花费时间。

（二） 实际系统中的磁盘驱动调度算法往往会区分读请求和写请求，请简要讨论其合理性？

（一）

(25,10,25),(78,15,55),(92,5,5),(101,7,101),(197,8,20),(12,11,10),(12,0,20),(12,12,75),(12,11,150)

5+53+14+9+96+185=362ms （2 分）

旋转一圈需要 10ms，平均旋转延迟为 5ms

362+5*6+140/200*10=399ms （1 分）

（二）

读写请求的时间容忍度不同，写的容忍度高，因为可以延迟写 （2 分）

得分	

二、编程题（8）

1. 试用管程实现 UNIX 中提出的匿名管道通信机制，管道维持一个容量为 K 字节的环形队列，并满足消息的先进先出，其上的 read 和 write 操作满足如下同步要求：1）当管道为空时，则 read 操作阻塞直到 write 操作写入数据；2）管道不为空时，如果 read 操作要求读取的字节数大于管道实际数据字节数，则读出数据并返回实际读出字节数，否则读出 read 要求的字节数；3）当 write 操作要求写入的数据字节数大于管道实际空闲容量，则写入尽可能多的数据，并阻塞直到 read 操作取出数据，而后继续执行写入操作，直至完成所有数据写入。（需要定义管程 Monitor，引入适当的条件变量 Condition，以及条件变量上的 wait 和 signal 操作，定义必要的管程变量，如 read_offset, write_offset, count 等，只需定义 read 和 write 函数）

```
type rw_monitor = monitor {
cond read, write;
int read_offset, write_offset, count, number; read_offset = 0, write_offset = 0, count = 0, number = 0;
Interface Module IM;
define read,write;
use wait,signal;
}

procedure read(int i) {
enter(IM);
if (count == 0)
    wait(read);
if (i > count) {
    读出数据
    read_offset = write_offset;
    number = count;
    if (count == K) {
        count = 0;
        signal(write);
    }
    else
        count = 0
    return number;
}
else {
    读出数据
    read_offset = (read_offset + i) % K;
    if (count == K) {
        count = count – i;
```

```
            signal(write);
        }
        else
            count = count – i;
}
leave(IM);
}

procedure write(int i) {
enter(IM);
for {
if (i > K – count) {
        写入数据
        i = i – (K - count);
        write_offset = read_offset;
        if (count == 0) {
            count = K;
            signal(read);
        }
        else
            count = K;
        wait(write);
}
else {
        写入数据
        write_offset = (write_offset + i) % K;
        if (count == 0) {
            count = count + i;
            signal(read);
        }
        else
            count = count + i;
        break;
}
}
leave(IM);
}
```