清华大学本科生考试试题纸	
考试课程：操作系统	时间：2023年01月03日上午09:00~11:00

姓名：_____	班级：_____	学号：_____

答卷注意事项:	1. 答题前在答卷纸每一页左上角写明姓名、班级、学号、页码和总页数。
	2. 在答卷纸上答题时, 要写明题号, 不必抄题。
	3. 答题时, 要书写清楚和整洁。
	4. 请注意回答所有试题。本试卷有17个小题，共6页。
	5. 考试完毕，必须将答卷拍照合成一个word或pdf，命名为“姓名-班级-学号”，考试结束后15分钟内将文件上传至网络学堂的作业提交区。
	6. 纸质答卷由同学个人妥善保存拍照后的纸质答卷处于提交时的状态，留存备查，直到期末考试成绩确认工作结束。

一、对错题（14分）

1. [__] 进程执行系统调用后从内核返回用户态前，将检查所有进程是否有待处理信号。
2. [__] 在多CPU环境下，正占用CPU在用户态运行中的进程会在每条指令执行结束后检查硬件中断请求。如果有硬件中断请求，则立即打断进程的执行，进入内核执行中断处理例程。
3. [__] 处理硬件中断时，由操作系统软件保存程序计数器（PC）的内容。
4. [__] 任何时候只能有一个进程在一个管程中执行。
5. [__] 进程P时间片用完会导致，进程P由运行态转为等待状态。
6. [__] 银行家算法可以判断系统是否处于死锁状态。
7. [__] 设置当前工作目录的目的是加快文件的检索速度。

二、填空题（14分）

8. 假设在单CPU场景下操作系统内核采用CFS调度算法，系统中只有两个进程A和B。假定初始情况下A的nice值显著高于B，将A的nice值减少1。则比较修改A的nice值前后，A将多获得___(8.1)___10%的时间，B将少获得___(8.2)___10%的时间。
   可能的选项：远大于、约等于、远小于
9. 系统中有4个不同的临界资源R1、R2、R3和R4，被5个进程p1、p2、p3、p4及p5共享。各进程对资源的需求为：p1申请R1和R2，p2申请R2和R3，p3申请R3和R4，p4申请R1和R4，p5申请R4。若系统出现死锁，则处于死锁状态的进程数至少是_____个。
10. 在下列同步机制中，可以用于任意数目进程的临界区访问控制的是_____。
    可能的选项：Peterson算法、Dekker算法、信号量方法、TestAndSet指令
11. 设文件F1的当前引用计数值为1，先建立F1的符号链接（软链接）文件F2，再建立F1的硬链接文件F3，然后删除F1。此时，F2的引用计数值是___(11.1)___，F3的引用计数值是___(11.2)___。
12. 某文件系统的目录项由文件名和索引节点号构成。若每个目录项长度为64字节，其中4个字节存放索引节点号，60个字节存放文件名，文件名由小写英文字母构成，则该文件系统能创建的文件数量的上限为_____。

三、简答题

13.（12分）

考虑下列程序

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/wait.h>
int main(){
    int val = 5;
    int i;

    for (i=0;i<2;i++) {
      if (fork() == 0) {
        val++;
      } else {
        val--;
        wait(NULL);
      }
    }
    printf("%d:%d\n", getpid(), val);
    fflush(stdout);
    exit(0);
}
```

上述代码没有检查返回值，我们假设所有函数均正确返回。

请针对上述代码执行结束时的情况，回答如下问题：

13.1）创建了多少个进程?

13.2）画出创建进程间的父子关系树。

13.3）在进程父子关系树中标出每个进程在退出前变量“val”的值。

14.（12分）

假设在单CPU场景下内核采用CFS调度算法，只有两个进程A和B。假定初始时A和B的nice值都为0，它们都将获得一半的CPU时间。此时将A的nice值减少1，则A多获得约10%的时间，约占总体时间的55%。

判断下面问题是否正确并给出原因：

14.1）出现**题设结果**的原因是nice=-1对应的权重比nice=0对应的权重高了约10%。

14.2）如果A和B的初始nice值均为10，则将A的nice值减少1后，A仍将多获得约10%的时间。

15.（6分）

15.1）小明同学想用多个线程爬取网络上的一些数据并写入同一个文件，但是每个线程都可以写入该文件，这时会出现数据错乱的情况。为什么会出现这种情况?

15.2）为了解决上述问题，小明想到了两种方案，分别是：

A. 使用一个互斥锁，在每次写入文件之前加锁，在写入完成之后解锁。这样可以保证每次只有一个线程可以写入文件。

B. 用一个线程负责写入文件，其他线程将数据写入一个队列，写入文件的线程从队列中取出数据写入文件。这样可以保证每次只有一个线程可以写入文件。

请问这两种方案哪种更好？为什么？

16.（22分）

理发店理有一位理发师、一把理发椅和5把供等候理发的顾客坐的椅子。理发师与顾客配合完成理发的过程中，理发师和顾客按如下规则进行合作。

1. ）如果没有顾客，理发师在理发椅上睡觉；
2. ）顾客到来时，如果理发师在睡觉，则该顾客叫醒理发师；
3. ）顾客到来时，如果理发师正在理发，则顾客在空椅子上坐下来等待；没有空椅子时直接离开。

请基于下面代码框架回答如下问题：（注：你可以不理睬这个代码框架，给出自己的独立实现）

16.1）用信号量机制实现理发师与顾客间正确且高效的同步与互斥；要求用伪代码实现，并给出必要代码注释。

16.2）请给一组测试用例，以测试实现的正确性；要求至少给出4种可能情况的测试用例，并在每个测试用例中注明至少一种不同的情况。

理发师问题的参考实现代码框架：

```
#include <pthread.h>
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <semaphore.h>
#include <fcntl.h>
#include <assert.h>
#define N 15
typedef struct{
int num;
int time;
}arg;
int b[]={1,10,3,9,4,8,5,6,7,6,4,6,7,8};
//信号量和控制量定义

// YOUR WORK --(1)--

//理发师线程
void*barber(){
  while(1){

  // YOUR WORK --(2)--

    printf("barber is serving %d\n",current);//一个顾客接受服务
    sleep(3);//3秒钟理发时间
    printf("%d over\n",current);

  // YOUR WORK --(3)--
```

```
  }
  return NULL;
}
void*customer(void*args){
  int *arg = (int*)args;
  if(arg[0]%3==0)sleep(0);
  else if(arg[0]%3==1)sleep(10);
  else sleep(arg[1]);

  // YOUR WORK --(4)--

  for(int i=0;i<=arg[0];i++)printf("\t");
  printf("come\n",arg[0]);//宣告到来

  // YOUR WORK --(5)--

    for(int i=0;i<=arg[0];i++)printf("\t");
    printf("waiting\n",arg[0]);

  // YOUR WORK --(6)--

    current=arg[0];
    for(int i=0;i<=arg[0];i++)printf("\t");
    printf("served\n",arg[0]);

  // YOUR WORK --(7)--

    for(int i=0;i<=arg[0];i++)printf("\t");
    printf("leave\n",arg[0]);
  }
  return NULL;
}
int main(){
  //以下对信号量进行初始化

  // YOUR WORK --(8)--

  printf("barber\t");
  for(int i=0;i<N;i++)printf("%d\t",i);
  printf("\n");
  //创建1个理发师和N个顾客线程
  pthread_t *barberid = malloc(sizeof(pthread_t));
  pthread_create(barberid,NULL,barber,NULL);
  arg *a = malloc(N*sizeof(arg));
  for(int i=0;i<N;i++){
    a[i].num=i;
    a[i].time=b[i];
  }
  pthread_t*customerid=malloc(N*sizeof(pthread_t));
  for(int i=0;i<N;i++)pthread_create(&customerid[i],NULL,customer,&(a[i]));
  sleep(36);
```

```
  return 0;
}
```

17.（20分）

下面给出了简化的EXT文件系统的总体分布，**各区域的最小单位与 Block 大小一致**。其中各个块占用空间说明如下。

```
+-------------+--------------+--------------+-------------+------------+
| Super Block | Block Bitmap | inode Bitmap | inode Table | Data Block |
+-------------+--------------+--------------+-------------+------------+
```

- **Super Block**: 超级块用于记录整个文件系统的信息，整个文件系统**只有一个**超级块，其记录的信息包括：
  - Block 与 inode 的总数量。
  - **未使用**的 Block 与 inode 的总数量。
  - Block 与 inode 的大小：Block 大小为 **512 B**， inode 大小为 **128 B**。
  - 其他文件系统的元信息，无特殊说明情况下，本题目中相关计算可**不考虑**这些信息。
- **Block Bitmap**：用来管理空闲 Block，根据 Block 的**分配**和**释放**进行相应修改。假定Block Bitmap 管理的是 Data Block 的分配情况，不包括 Super Block等区域。
- **Inode Bitmap**：用来管理空闲 inode，根据 inode 的**分配**和**释放**进行相应修改。
- **Inode Table**：顺序存放 inode，假定inode Table的空间在文件系统创建时就已分配完毕，新创建的 inode 需要对 inode Table 进行写入。
- **Data Block**：存放数据，这些数据可能包括：
  - 文件数据：每个 Block 内最多只能存放和一个文件有关的数据，一个文件可能占用多个 Block。
  - 目录数据：对于类型为 **目录** 的 inode，需要使用数据块来存放 **Dirent** (Directory Entry) 信息。 每个 Block 内最多只能存放和一个目录有关的数据，多个 Dirent 可能占用多个 Block。假定Dirent 采用**顺序分配**的方式，释放 Dirent 后其状态被置为无效，但占用的空间不会被释放。
- **inode**：默认其包含如下信息：
  - inode 类型：**文件**、**目录**、**软链接**等。
  - 文件大小
  - 分配的 Block 数量
  - **直接索引和间接索引**：inode 中有 **60 B** 用来记录索引，每个索引占用 **4 B**。前 12 个为直接索引，第 13 个为一级间接索引，第 14 个为二级间接索引，第 15 个为三级间接索引。一级索引指向的数据块为索引块，二、三级索引以此类推。
  - **硬链接数**
  - 时间戳：访问时间、创建时间、修改时间
  - 权限控制

现在我们有一个大小为 **20230103 B** 的文件，存放在目录 `/A/B/C/` 中，文件名为 `os.txt` ，即该文件的完整路径为 `/A/B/C/os.txt`。

请回答如下问题：

17.1）只使用直接索引时，一个 inode 表示的文件最大是多少？使用直接索引和一级索引时，一个 inode 表示的文件最大是多少？（要求给出计算过程。）

17.2）存储文件`os.txt`的所有数据总共需要多少个 Block？（只考虑数据块和索引块，不考虑目录或 inode；要求给出计算过程。）

17.3）假设不在内存中缓存，每次访问文件系统都需要直接访问磁盘，每次访问磁盘都只能访问到一个块，查找到该文件对应的 inode 至少需要多少次磁盘访问？请简要写出查找过程（最后一次磁盘访问应该读到该文件对应的 inode 的内容）。