清华大学本科生考试试题纸

考试课程：操作系统　　时间：2023年12月14日上午09:50~12:15　　A卷

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。第6页是空白页，可用于草稿。
5. 完成答卷后，请将试题和答卷一起交回。

一、对错题（*10*分）

1. [__] 条件变量能够提供互斥访问控制。
2. [__] 在多核系统中，中断屏蔽机制不适用于临界区的访问控制。
3. [__] 在Go语言中，不同的goroutine能够复用相同的堆栈。
4. [__] 应用程序能够通过系统调用创建用户级线程（User-level Threads）。
5. [__] 单处理机系统中，进程与进程是可并行的。

二、单选题（*10*分）

6. 中断处理和子程序调用都需要压栈以保护现场，中断处理一定会保存而子程序调用不需要保存其内容的是
   A．程序计数器　　B．程序状态字寄存器
   C．通用数据寄存器　　D．通用地址寄存器
7. 在支持多线程的系统中，进程P创建的若干个线程不能共享的是
   A．进程P的代码段　　B．进程P中打开的文件
   C．进程P的全局变量　　D．进程P中某线程的栈指针
8. 某系统有n台互斥使用的同类设备，三个并发进程分别需要3、4、5台设备。可确保系统不发生死锁的设备数n最小为
   A．9　　B．10
   C．11　　D．12
9. 下列关于虚拟存储的叙述中，正确的是
   A．虚拟存储只能基于连续分配技术　　B．虚拟存储只能基于非连续分配技术
   C．虚拟存储容量只受外存容量的限制　　D．虚拟存储容量只受内存容量的限制
10. 若目录dir下有文件file1，则为删除该文件内核不必完成的工作是
    A．删除file1的快捷方式　　B．释放file1的文件控制块
    C．释放file1占用的磁盘空间　　D．删除目录dir中与file1对应的目录项

三、简答题

11. （15分）

本题的描述中我们统一采用简化的 EXT 文件系统的规范。下面给出了文件系统中一个 Block Group 的总体分布，其中各个块占用空间后面会详细说明，**各区域的最小单位与 Block 大小一致。**

```
+-------------+-------------------+--------------+--------------+-------------+------------+
| Super Block | Group Description | 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，本题目中默认整个文件系统**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：本题目中 Block 大小默认为 **512 B**， Inode 大小默认为 **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。
- **Group Descriptor**：记录各区域分布的起始位置，本题目中文件系统已创建完毕，因此默认后续操作不会对这块区域进行修改，本题目中相关计算可**不考虑**这些信息。
- **Block Bitmap**：顾名思义，用来对空闲 Block 进行管理，需要根据 Block 的**分配**和**释放**进行相应修改。**注意**：本题目中 Block Bitmap 管理的是 Data Block 的分配情况，不包括 Super Block、Group Description 等区域。
- **Inode Bitmap**：与 Block Bitmap 同理，用来对空闲 Inode 进行管理，需要根据 Inode 的**分配**和**释放**进行相应修改。
- **Inode Table**：顺序存放 Inode，在本题目中，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。为简化题目，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**：为简化题目，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，可支持 128 个直接索引，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

1. 该文件系统最大能支持多大的文件？要求给出计算过程。
2. 假定只考虑数据块和索引块，不考虑目录或 Inode。存储一个大小为 **1M B** 的文件总共需要多少个 Block？ 要求给出计算过程。
3. 请简述文件系统中软链接和硬链接的区别。

12. （6分）

由于硬盘的机械操作比较慢，操作系统通常会对文件写入操作进行缓冲。write 系统调用在写入一个缓冲区以后就先返回，之后一段时间在后台再将多个写入一起写进硬盘里。如果在“写进硬盘”完成前掉电，会导致操作失败，但用户并不知道操作失败。

1. 请给出一个故障模式，让以下程序的行为出现功能性错误。

```
void handle_web_request(string path, string content) {
  check_path_exists(path);
  if(!write_file(path, content)) send_web_response("error");
  else send_web_response("success");
}
```

2. 文件写入操作的内容和操作本身的元数据可以以不同的方式缓冲，或者部分不进行缓冲。如果被缓冲的是一个数据日志（Data Journaling）文件系统，那么 write 系统调用在内核进行完哪一步之后就可以返回了呢？并请解释理由。（提示：write 返回代表数据已经“写到”硬盘上了。）

jornal commit

13. （9分）

请描述管程的Hoare、Hanson和MESA三种实现方式的区别。

14. （15分）

某小型滑冰场最多可容纳30人同时参观，有一个出入口，该出入口一次仅允许一个人通过。参观者的活动描述如下：

```
参观者进程i：
{
  …
  进门；
  …
  参观；
  …
  出门；
  …
}
```

请添加必要的信号量和P、V（或wait()、signal()）操作，以实现上述过程中的互斥与同步。要求写出完整的过程，说明信号量的含义并赋初值。

15. （10分）

1. 请简要描述CFS调度算法的基本工作原理。
2. 在CFS调度算法中，所有进程的虚拟时间vruntime增长速度宏观上是同时推进的，可用虚拟时间vruntime作为就绪进程队列的排队顺序。假定每个时间片的长度为一个时间单位；优先数表示CPU时间占比。请给出下面场景下CFS调度算法在前20个时间单位中，各进程的虚拟时间变化情况和处理机调度顺序。

时间	**进程A**	**进程B**	**进程C**	**调度顺序**
优先数	2	3	2	
进入就绪时间	0	0.1	7.1	
时间	虚拟时间			调度顺序
0	0			
1				A
2				
3				
.....				

16. （10分）

请补全下面信号量和条件变量的伪代码实现。

信号量的伪代码实现：

```
class Semaphore {
  int sem;
  WaitQueue q;
}

Semaphore::P() {
  ......
}

Semaphore::v() {
  ......
}
```

条件变量的伪代码实现：

```
class Condition {
  int numWaiting = 0;
  WaitQueue q;
}

Condition::wait(lock) {
  ......
}

Condition::signal() {
  ......
}
```

17. （15分）

下面是软件同步方法中的Peterson算法的伪代码。

共享变量的初始化代码：

```
01  // 共享变量
02  let mut flag = [false; N]; // 标识线程是否请求进入临界区
03  let mut turn = 0; // 记录应该让哪个线程进入临界区
```

线程Ti和Tj的伪代码： $(i \neq j,\ i,j = 0\ or\ 1)$

```
04  // 线程Ti                                    // 线程Tj
05  while (true) {                               while (true) {
06      flag[i] = true;                            flag[j] = true;
07      turn = j;                                  turn = i;
08      while (flag[j] == true && turn == j);      while (flag[i] == true && turn == i);
09      执行任务临界区;                            执行任务临界区;
10      // 退出临界区                              // 退出临界区
11      flag[i] = false;                           flag[j] = false;
12  }                                            }
```

请补全Peterson算法在上面场景下，从线程Ti执行完第08行开始，到两个线程都执行到第11行中间的合理执行序列。

线程Ti	线程Tj
	.....
	06 flag[j] = true;
06 flag[i] = true;	
07 turn = j;	
08 while (flag[j] == true && turn == j);	