清华大学本科生考试试题专页纸	
考试课程：操作系统（A卷）	时间：2023年06月13日下午14:30~17:00

系别：	班级：	学号：	姓名：

	1. 答题前，请先在试题纸和答卷本上写明A卷或B卷、系别、班级、学号和姓名。
答卷注意事项：	2. 在答卷本上答题时，要写明题号，不必抄题。
	3. 答题时，要书写清楚和整洁。
	4.请回答所有题目。本试卷有1个判断题，8个简答题，共10页。
	5. 考试完毕，必须将试题纸和答卷本一起交回。

一、(20分)判断题

1. [ ] 课程实验中的操作系统内核在RISC-V的S-Mode特权模式下运行。
2. [ ] 课程实验中的操作系统内核属于宏内核（Monolithic Kernel）架构。
3. [ ] 在用户态产生或发出中断/异常/系统调用后，处理器会进入内核态并跳转到操作系统预设的处理例程进行相应的处理。
4. [ ] 操作系统为进程建立的页表是保存在物理内存中的。
5. [ ] 在RISC-V中切换页表基址的机器指令是特权指令。
6. [ ] 实现协作式调度可以在用户态完成，不需要操作系统的介入。
7. [ ] 工作集是指一个进程在当前一段时间内正在访问的逻辑页面集合。
8. [ ] Clock页面置换算法通过页表项的标志位来了解对应页的近期读写情况。
9. [ ] Belady异常现象不会在最佳页面置换算法（OPT）中出现。
10. [ ] 处于运行态的进程在时钟中断产生后将立刻进入等待状态。
11. [ ] 实时系统中的速率单调调度算法(RM, Rate Monotonic)属于静态优先级调度。
12. [ ] 实时系统中，高优先级进程长时间等待低优先级进程所占用资源的现象是一种优先级反置现象（Priority Inversion）。
13. [ ] UNIX/Linux操作系统中，目录和管道(pipe)都是文件。
14. [ ] 忙等的自旋锁机制可以用于多核架构下的互斥资源保护。
15. [ ] UNIX/Linux操作系统中，如果父进程先于子进程退出，将导致子进程直接退出。
16. [ ] 对于磁盘文件，应用程序通过执行open系统调用获得的文件描述符存在于磁盘上。
17. [ ] 硬链接与软链接都能指向跨不同文件系统的文件。
18. [ ] 共享内存机制是一种把同一个物理内存区域同时映射到多个进程的内存地址空间的进程间通信机制。
19. [ ] 信号（signal）机制是一种进程间的异步通知机制，操作系统可通过信号机制来通知或终止进程。
20. [ ] 临界区(Critical Section)是需要互斥访问的共享数据。

二、(80分)简答题

1.（8分）

1. （2分）请描述实时操作系统中的优先级继承算法的基本思路。

2. （2分）请描述实时操作系统中的优先级天花板协议的基本思路。

3. （4分）对于本课程的实验操作系统，如果要改造为实时操作系统，请问需要扩展或改进哪些系统调用，并扩展或改进内核模块的哪些功能？

2.（8分）

1. （2分）我们课上介绍了进程、线程、协程的区别与特点，请简明说明三者的区别。

2. （6分）在多核计算机系统中，对于下述应用程序编程需求，请从多进程、多线程、多协程之中选择对应应用最合适(高性能、安全、编程便捷性的综合考虑)的编程模型，并简明说明理由：

（1）编写一个C++程序来进行矩阵乘法的并行计算。

（2）编写一个评测系统来自动为选课同学的实验代码进行评测，这个评测系统将会并行运行多位同学的代码，同学们的代码可能有BUG以导致运行中出现崩溃。

（3）编写一个网络爬虫来爬取上万个网页并将结果直接保存到文件里。

3.（14分）

1. （4分）请简明描述死锁产生的4个必要条件。

2. （4分）某计算机系统中有8个同一类共享资源，有K个线程竞争使用，每个线程最多需要3个资源。则该计算机系统可能发生死锁的K的最小值是多少？请说明或给出计算过程。注：如果线程拿到它所需的最大资源数，它就可以顺利执行完毕，并释放资源。

3. （6分）假设系统中有4个进程P0、P1、P2、P3，并有三类的资源A、B、C，总共有9个A，3个B和6个C。在t0时刻，系统可用资源为：A=1， B=1， C=2。

在t0时刻，各个进程的资源情况如下表所示：

进程	最大资源需求			已分配的资源			还需要的资源		
	A	B	C	A	B	C	A	B	C
P0	3	2	2	1	0	0	2	2	2
P1	6	1	3	5	1	1	1	0	2
P2	3	1	4	2	1	1	1	0	3
P3	4	2	2	0	0	2	4	2	0

3.1 请描述银行家算法，并阐述其在预防死锁中的作用。

3.2 在 t0 时刻，系统是否处于安全状态？请给出简明的解释说明。

3.3 在 t0 时刻后，P1 发出资源请求向量（A=1，B=0，C=1），如果系统基于银行家算法来考虑，能否将资源分配给 P1？请给出简明的解释说明。

4. (12分)

假设有一个基于索引节点（inode）的 too-easy-fs 文件系统，支持符号链接（软链接）与硬链接。每个文件都有一个关联的 inode，其中包含了文件的元数据（引用计数、文件创建时间、文件访问时间等）以及对数据块的直接和间接指针。

1. （4 分）设文件 F1 的当前引用计数值为 1，一个用户先建立了 F1 文件的符号链接文件 F2，再建立了 F1 文件的硬链接文件 F3，然后删除了 F1 文件。请问此时文件 F2 和 F3 的引用计数值分别是多少？
2. （2 分）请简述索引节点（inode）的直接和间接指针的作用。
3. （4 分）假设这个文件系统中，每个索引节点包含 10 个直接指针和 1 个间接指针，每个指针都是 4 字节，每个数据块的大小为 1KB。请计算以下内容：

   (1) 仅使用直接指针，一个文件可以包含的最大数据量是多少？

   (2) 如果可使用直接指针和间接指针，一个文件可以包含的最大数据量是多少？
4. （2 分）考虑文件的读取操作。如果我们要读取文件中某数据块，操作系统需要执行哪些步骤？请问在读文件过程中，是否会对文件的索引节点进行写操作？请解释原因。

5. (6分)

考虑如下程序：

```c
#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 < 3; i++) {
      if (fork() == 0) { //子进程
        val--;
      } else {
        val++;
        wait(NULL);
      }
    }
    printf("%d:%d\n", getpid(), val);
    fflush(stdout);
    exit(0);
}
```

请针对上述代码执行结束时的情况，回答下列问题：

1. （2 分）除最开始执行该程序的进程外，新创建了多少个进程？
2. （2 分）请画出各个进程间的父子关系树。

3. （2 分）请在父子关系树中标出每个进程在退出前变量 val 的值。

7. (8分)

下面是一个可以并行完成数组求和的程序，但缺少了一部分系统调用的代码：

```
int main() {
  int n;
  int data[4096];
  scanf("%d", &n);
  for (int i = 0; i < n; i++) scanf("%d", data + i);
  int id = 0;
  int fd[8][2];
  for (int i = 1; i < 8; i++) {
    (1)__________________;
    if ( (2)__________ ) {
      id = i;
      break;
    }
  }
  int slice = (n + 7) / 8;
  int start = slice * id, stop = min(start + slice, n);
  int sum = 0;
  for (int i = start; i < stop; i++)
    sum += data[i];
  if (id == 0) {
    for (int i = 1; i < 8; i++) {
      int child_sum;
      (3)_____________________________________________;
      sum += child_sum;
    }
  }
  else {
    (4)_____________________________________;
  }
  if (id == 0) printf("sum: %d\n", sum);
  return 0;
}
```

已知如下系统调用接口：

接口	用途
`int pipe (int *pipefd)`	创建一组管道，`pipefd[0]` 用于管道读端的文件描述符， `pipedfd[1]` 用于管道写端的文件描述符
`int fork()`	创建子进程，该函数在父进程会返回子进程的进程号，在子进程会返回 0
`ssize_t read(int fd, void *buf, size_t nbytes)`	从文件描述符 `fd` 对应文件读取 `nbytes` 个字节到起始地址为 `buf` 的内存中
`ssize_t write(int fd, const void *buf, size_t n)`	向文件描述符 `fd` 对应文件写入从内存地址 `buf` 开始的 `n` 个字节

（在参数使用正确的前提下）我们假设所有系统调用均会成功，无需做异常处理。请你补全上述四处缺失的具体代码内容，每个地方仅限补充一个表达式和函数，需要明确函数中的参数，不应包含分号。

8. (12分)

1. （6分）下面是基于软件的Perterson互斥算法，存在一些错误的地方。请指出这个有误的算法会导致哪些临界区访问规则被破坏，并给出具体例子的执行过程来说明如何破坏访问规则的。请把Perterson算法修复正确。

Perterson算法

```
// 共享变量
let mut flag = [false; 2]; // 标识线程是否请求进入临界区
let mut turn = 0; // 记录应该让哪个线程进入临界区

// 线程P0
while (true) {
  flag[0] = true;
  turn = 1;
  while (flag[1] != true && turn != 1) ;
  // 进入临界区执行任务
  // 退出临界区
  flag[1] = true;
}

// 线程P1
while (true) {
  flag[1] = true;
  turn = 0;
  while (flag[0] != true && turn != 0) ;
  // 进入临界区执行任务
  // 退出临界区
  flag[0] = true;
}
```

2. （6分）系统中有多个生产者线程和消费者线程，共享用一个可以存100个产品的环形缓冲区（初始为空），当缓冲区为未满时，生产者线程可以放入一件其生产的产品，否则等待；当缓冲区为未空时，消费者线程可以取走一件产品，否则等待。要求一个消费者线程从缓冲区连续取出5件产品后，其他消费者线程才可以取产品，请用信号量P/V操作实现线程间的互斥和同步，要求写出完整的过程；并指出所用信号量和所用变量的含义和初值。

9. (12分)

日志文件系统是为了解决机器异常掉电或操作系统异常崩溃带来的文件系统崩溃一致性（crash-consistency）问题。某 simple-journal-fs 日志文件系统的日志模块核心伪代码如下，请回答下面的相关问题。

相关数据结构定义和重要函数：

```c
struct buf {
    int block_id;             // 数据所在的磁盘块
    char data[BLOCK_SIZE];  // 实际数据
}

struct membuf {
    int len;                  // 有效 buf 的数量
    struct buf bufs[MAX_BUF_NUM];
} MEM_BUF;  // 全局 in memory buf

struct loghdr {
    int len;                  // 有效 block_id 的数量
    int block_id[MAX_BUF_NUM];
}

// 将 data 写入第 block_id 个磁盘块，但仅会写入磁盘的非持久化缓存，需要 disk_flush() 完成实际写入。
void disk_write(int block_id, char* data);

// 从第 block_id 个磁盘块读出 data。
char* disk_read(int block_id);

// 通知磁盘将磁盘缓存中的数据完成实际写入。
void disk_flush();
```

将 log 和数据写入磁盘相关函数：

```c
struct loghdr* log_build() {
    if MEM_BUF->len == 0 {
        return NULL;
    }
    loghdr *hdr = empty_hdr();
    for(int i = 0; i < MEM_BUF->len; ++i) {
        // 对于将所有 memory buf 的数据写入从 LOG_BASE 开始的块
        disk_write(LOG_BASE + i, MEM_BUF->bufs[i].data);
        // 将对应的 block_id 写入 loghdr
        hdr->block_id[i] = MEM_BUF->bufs[i].block_id;
    }
    hdr->len = MEM_BUF->len;
    return hdr;
}

void data_apply() {
    for(int i = 0; i < MEM_BUF->len; ++i) {
        // 将 memory buf 的数据写入磁盘
        disk_write(MEM_BUF->bufs[i].block_id, MEM_BUF->bufs[i].data);
    }
}

```

```
23 int log_commit() {
24     loghdr *hdr = log_build();
25     if hdr == NULL {
26         return 0;
27     }
28     disk_flush();
29     disk_write(LOG_HEADER, (char*)hdr);
30     disk_flush();
31     data_apply();
32     disk_flush();
33     hdr->len = 0;
34     disk_write(LOG_HEADER, (char*)hdr);
35     disk_flush();
36     MEM_BUF.clear();
37     return 0;
38 }
39
40 int log_recover() {
41     loghdr *hdr = disk_read(LOG_HEADER);
42     for(int i = 0; i < hdr->len; ++i) {
43         char* data = disk_read(LOG_BASE + i);
44         // YOUR CODE，仅有一行
45     }
46     disk_flush();
47     hdr->len = 0;
48     disk_write(LOG_HEADER, (char*)hdr);
49     disk_flush();
50     return 0;
51 }
```

1. （4 分）在 `log_commit` 函数中，以下操作各对应哪一行代码？（写行号即可，已知不是 `disk_flush()` ）

   - Journal write（日志写入）：将事务的内容（包括 TxB 和更新内容）写入日志，等待这些写入完成。
   - Journal commit（日志提交）：将事务提交块（包括 TxE）写入日志，等待写完成，事务被认为已提交（committed）。
   - Data write（数据写入）：将数据写入最终位置，等待完成。
   - Journal free（释放）：稍后，在日志超级块中将事务标记为空闲。

2. （2 分）请补全 `log_recover` 中第 44 行空缺的代码

3. （3 分）此文件系统开发者在更新中，不小心把第 35 行的“`disk_flush()`”注释掉了，请问是否可能会导致错误？何种情形下会导致错误？

4. （3 分）进一步改进的 advanced-journal-fs 日志文件系统在缺省情况下，日志只记录元数据（metadata），其磁盘写入顺序为 ：

   - Data write（数据写入）：将数据写入最终位置，等待完成。
   - Journal metadata write（日志元数据写入）：将开始块和元数据写入日志，等待写入完成。
   - Journal commit（日志提交）：将事务提交块（包括 TxE）写入日志，等待写完成，现在认为事务（包括数据）已提交（committed）

- Metadata write（加检查点元数据）：将元数据更新的内容写入文件系统中的最终位置。
- Journal free（释放）：稍后，在日志超级块中将事务标记为空闲。

这样做与上述 simple-journal-fs 日志文件系统的做法相比，有何好处？为何先写入 data？