清华大学本科生考试试题专用纸

考试课程：**操作系统（A 卷）**　　时间：2014 年 05 月 20 日上午 9:50~11:50

系别：_____ 班级：_____ 学号：_____ 姓名：_____

答卷注意事项：1. 在开始答题前，请在试题纸和答卷本上写明系别、班级、学号和姓名。
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. 请注意回答所有试题。本试卷有 8 个题目，共 6 页。
5. 考试完毕，必须将试题纸和答卷本一起交回。

一、(12 分)在 Linux/Unix 中，一个用户从 shell 中执行了一个运行时间较长且不知何时能够结束的程序，Linux/UNIX 可以让用户根据个人需求随时通过敲击 Ctrl-C 组合键来终止这个程序的执行。请回答如下问题。要求设计应该具有通用性，列出的设计实现不超过 6 点，每点不超过 4 行。问题的执行流程描述不超过 8 行。

1）如果要在 ucore 中实现 Linux/UNIX 同样的功能，请问应该如何修改 ucore 来支持此功能？

2）uCore 的 shell 也是一个程序，我们希望避免这个 shell 在执行中被用户敲入的 Ctrl-C 所终止，请问在保证 1）的要求请看下，如何修改 ucore 和 shell 来支持此功能？

3）说明在你的设计下，shell 和某一可被终止程序在执行过程中，用户敲击 Ctrl-C 后，uCore 和 shell 的执行流程。

二、(15 分)在具备了执行用户态进程的能力之后，uCore 要为这些进程提供的一个重要服务，是用户进程之间的消息传递机制（Inter-Process Communication，简写为 IPC）。现在，我们要为 uCore 实现以下两个系统调用，以实现一种同步的 IPC 机制（暂不考虑超时等功能）：

```
int sys_send_event(int pid, int event);
参数：pid -  该消息的目标进程的进程号；
      event –  消息内容，用一个整型表示。
返回值：消息成功发送时，返回 0；否则，返回相应的错误代码。
```

```
int sys_recv_event(int *pid, int *event);
参数：pid -  函数返回时，*pid 保存发出消息的进程的进程号，可以为 NULL；
      event –  函数返回时，*event 保存消息内容，可以为 NULL。
返回值：消息成功接收时，返回 0；否则，返回相应的错误代码。
```

1）以下是一个基于上述 IPC 机制求质数的用户程序：

```
#include <ulib.h>
#include <stdio.h>
#include <unistd.h>
#include <string.h>

const int total = 1000;
```

```
void primeproc(void)
{
	int index = 0, this, num, pid = 0;
top:
	recv_event(NULL, &this);
	cprintf("%d is a primer.\n", this);

	while (recv_event(NULL, &num) == 0) {
		if ((num % this) == 0) {
			continue;
		}
		if (pid == 0) {
			if (index + 1 == total) {
				goto out;
			}
			if ((pid = fork()) == 0) {
				index++;
				goto top;
			}
			if (pid < 0) {
				goto out;
			}
		}
		if (send_event(pid, num) != 0) {
			goto out;
		}
	}
out:
	cprintf("[%04d] %d quit.\n", getpid(), index);
}

int main(void)
{
	int i, pid;
	unsigned int time = gettime_msec();
	if ((pid = fork()) == 0) {
		primeproc();
		exit(0);
	}
```

```
        assert(pid > 0);

        for (i = 2;; i++) {
                if (send_event(pid, i) != 0) {
                        break;
                }
        }
        cprintf("use %d msecs.\n", gettime_msec() - time);
        cprintf("primer3 pass.\n");
        return 0;
}
```

简述这个程序是如何判断并输出前五个质数的。

2）给出一种基于等待队列的上述 IPC 机制的实现方案。

三、(10 分)在 uCore 中，信号量的定义如下

```
typedef struct {
    int value;
    wait_queue_t wait_queue;
} semaphore_t;
```

__up 函数是信号量 V 操作的具体实现函数

```
static __noinline void __up(semaphore_t *sem, uint32_t wait_state) {
    bool intr_flag;
    local_intr_save(intr_flag);
    {
        wait_t *wait;
        if((wait=wait_queue_first(&(sem->wait_queue)))==NULL){
            _________;
        } else {
            wakeup_wait(&(sem->wait_queue), wait, wait_state, 1);
        }
    }
    local_intr_restore(intr_flag);
}
```

1）补全程序中的空行_____。 uCore其实不会打印。

2）信号量的 value 值>0 时，表示_____的数量;value 值<0 时，表示__等待__的数量。

3）local_intr_save 和 local_intr_restore 这两个函数的功能分别是什么？为什么要调用这两个函数？ 存.关 中断 开.恢复中断

四、(15 分)假设一个 MOOC 网站有 1、2、3 三种不同的课程视频可由学生选择学习，网站播放课程视频的规则为：

1）任一时刻最多只能播放一种课程视频，正在播放的课程视频是自动循环播放的，最后一个学生主动离开时结束当前课程视频的播放；

2）选择当前正在播放的课程视频的学生可立即进入播放页面，允许同时有多位选择同一种课程视频的学生观看，同时观看的学生数量不受限制；

3）等待观看其它课程视频的学生按到达顺序排队，当一种新的课程视频开始放映时，所有等待观看该课程视频的学生可依次序进入播放页面同时观看。

用一个进程代表一个学生，要求：用信号量的 P、V 操作实现上述规则，并给出信号量的定义和初始值。

五、(12 分)在 lab6 中，我们实现了 Stride Scheduling 调度算法，并声称它对“进程的调度次数正比于其优先级”。对于优先级为 2、3、5、7 的 4 个进程，选取 210 为 MAX_STRIDE，则：

1）简要描述 Stride Scheduling 调度算法。

2）四个进程的步长分别为：_____、_____、_____、_____。

3）假设四个进程的初始 stride 值均为 0，证明：总有一个时刻，四个进程的 stride 值都是 210，且此时四个进程被调度的次数正比于其优先级。

六、(12 分)死锁是操作系统中资源共享时面临的一个难题。请回答下列与死锁相关的问题。

1）设系统中有下述解决死锁的方法：

a)银行家算法；

b)检测死锁，终止处于死锁状态的进程，释放该进程占有的资源；

c)资源预分配。

简述哪种办法允许最大的并发性，即哪种办法允许更多的进程无等待地向前推进？请按“并发性”从大到小对上述三种办法进行排序。

2）假设一个使用银行家算法的系统，当前有 5 个进程 P0, P1, P2, P3, P4，系统中有三类资源 A、B、C，假设在某时刻有如下状态：

	Allocation			Max			Available		
	A	B	C	A	B	C	A	B	C
P_0	0	0	3	0	0	4	1	4	0
P_1	1	0	0	1	7	5			
P_2	1	3	5	2	3	5			
P_3	0	0	2	0	6	4			
P_4	0	0	1	0	6	5			

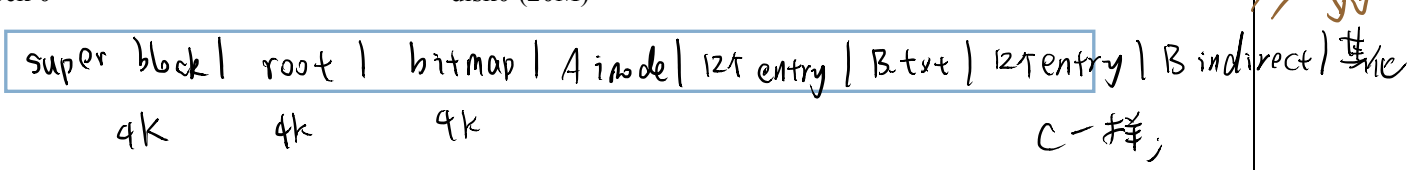
请问当前系统是否出于安全状态？如果系统中的可利用资源为（0, 6, 2），系统是否安全？如果系统处在安全状态，请给出安全序列；如果系统处在非安全状态，请简要说明原因。

七、(12 分)uCore 实现了一个简单的文件系统 Simple FS，假设该文件系统现已经装载到一个硬盘中（disk0），该硬盘的大小为 20M，目前有三个文件 A.txt,B.txt 和 C.txt 存放在该硬盘中，三个文件的大小分别是 48K, 1M 和 4M。

1）简要描述 SFS 文件系统中文件数据的组织结构（即：SFS 文件的数据的存放位置组织方式）。 inode 结构； 读代码； Lab8

2）请根据 Simple FS 的设计实现情况，画出该文件系统当前在 disk0 上的布局情况，需要给出相应结构的名称和起始块号。

block 0 disk0 (20M)



八、(12 分)uCore 的文件管理主要由以下四个部分组成：通用文件系统访问接口层，文件系统抽象层(VFS)，具体文件系统层以及外设接口层，其中 VFS 层的作用是用来管理不同的文件系统并向上提供一致的接口给内核其他部分访问，在 ucore 中我们已经实现了一个具体的文件系统：Simple FS，并将该文件系统装载到了 disk0 上，假设 ucore 又实现了一个文件系统 FAT32，并将这个新的文件系统装载到了 disk1 上。

1）请简单描述一下如何修改 VFS 层的数据结构使其可以有效的管理上述已安装的具体文件系统。 涉及 VFS 层的数据结构如下：

VFS: file → inode { device info / SFS inode ↓ disk inode

看代码

```
struct file {
    enum {
        FD_NONE, FD_INIT, FD_OPENED, FD_CLOSED,
    } status;
    bool readable;
    bool writable;
    int fd;
    off_t pos;
    struct inode *node;
    atomic_t open_count;
};
```

```
struct inode {
    union {
        struct device __device_info;
        struct sfs_inode __sfs_inode_info;
    } in_info;
    enum {
        inode_type_device_info = 0x1234,
        inode_type_sfs_inode_info,
    } in_type;
    atomic_t ref_count;
```

```
    atomic_t open_count;
    struct fs *in_fs;
    const struct inode_ops *in_ops;
};
```

```
struct fs {
    union {
        struct sfs_fs __sfs_info;
    } fs_info;
    enum {
        fs_type_sfs_info,
    } fs_type;
    int (*fs_sync)(struct fs *fs);
    struct inode *(*fs_get_root)(struct fs *fs);
    int (*fs_unmount)(struct fs *fs);
    void (*fs_cleanup)(struct fs *fs);
};
```

```
struct inode_ops {
    unsigned long vop_magic;
    int (*vop_open)(struct inode *node, uint32_t open_flags);
    int (*vop_close)(struct inode *node);
    int (*vop_read)(struct inode *node, struct iobuf *iob);
    int (*vop_write)(struct inode *node, struct iobuf *iob);
    int (*vop_getdirentry)(struct inode *node, struct iobuf *iob);
    int (*vop_create)(struct inode *node, const char *name,
                      bool excl, struct inode **node_store);
    int (*vop_lookup)(struct inode *node, char *path,
                      struct inode **node_store);
    ……
};
```

2）两个具体文件系统均已实现了对数据文件的 4 种基本操作。现在有某个用户态进程执行了一个 copy（source_path, dest_path,...）函数，该函数是把 disk1 根目录下的一个文件 A.txt 拷贝到了 disk0 的根目录下（不用考虑文件的大小），请结合 ucore 中对数据文件的操作流程描述一下这个函数的执行过程。