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：2013 年 06 月 22 日上午 8:00~10:00

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 6 个题目，共 20 页。

5. 考试完毕, 必须将试题纸和答卷本一起交回。

一、（15 分）在虚拟存储系统中，当由虚拟地址找不到对应的物理地址时，会产生缺页故障。请完成如下任务。
1）描述缺页故障（page_fault）的处理流程;
2）补全下面缺页处理中所缺代码。

```
=============kern/trap/trap.c=============

...
static int
pgfault_handler(struct trapframe *tf) {
    extern struct mm_struct *check_mm_struct;
    if(check_mm_struct !=NULL) { //used for test check_swap
            print_pgfault(tf);
        }
    struct mm_struct *mm;
    if (check_mm_struct != NULL) {
        assert(current == idleproc);
        mm = check_mm_struct;
    }
    else {
        if (current == NULL) {
            print_trapframe(tf);
            print_pgfault(tf);
            panic("unhandled page fault.\n");
        }
        mm = current->mm;
    }
    return ...(1)...;
}
...
```

```
static void
trap_dispatch(struct trapframe *tf) {
    char c;

    int ret=0;

    switch (tf->tf_trapno) {
    case T_DEBUG:
    case T_BRKPT:
        debug_monitor(tf);
        break;
    case T_PGFLT:
        if ((ret = ...(2)...) != 0) {
            print_trapframe(tf);
            if (current == NULL) {
                panic("handle pgfault failed. ret=%d\n", ret);
            }
            else {
                if (trap_in_kernel(tf)) {
                    panic("handle pgfault failed in kernel mode. ret=%d\n", ret);
                }
                cprintf("killed by kernel.\n");
                panic("handle user mode pgfault failed. ret=%d\n", ret);
                do_exit(-E_KILLED);
            }
        }
        break;
    case T_SYSCALL:
        syscall();
        break;
    case IRQ_OFFSET + IRQ_TIMER:

//    LAB3 : If some page replacement algorithm need tick to change the priority of pages,
//    then you can add code here.
        ticks ++;
        assert(current != NULL);
        run_timer_list();
        break;
    case IRQ_OFFSET + IRQ_COM1:
    case IRQ_OFFSET + IRQ_KBD:
        if ((c = cons_getc()) == 13) {
            debug_monitor(tf);
        }
        else {
            cprintf("%s [%03d] %c\n",
                    (tf->tf_trapno != IRQ_OFFSET + IRQ_KBD) ? "serial" : "kbd", c, c);
        }
        break;
    case IRQ_OFFSET + IRQ_IDE1:
    case IRQ_OFFSET + IRQ_IDE2:
        /* do nothing */
        break;
    default:
        print_trapframe(tf);
        if (current != NULL) {
            cprintf("unhandled trap.\n");
            do_exit(-E_KILLED);
        }
        // in kernel, it must be a mistake
        panic("unexpected trap in kernel.\n");

    }
}

=============kern/mm/vmm.c=============

// do_pgfault - interrupt handler to process the page fault execption
int
do_pgfault(struct mm_struct *mm, uint32_t error_code, uintptr_t addr) {
    int ret = -E_INVAL;
    struct vma_struct *vma = find_vma(mm, addr);

    pgfault_num++;
    if (vma == NULL ) {
        cprintf("not valid addr %x, and  can not find it vma %x\n", addr, vma);
        goto failed;
    }
    else if (vma->vm_start > addr) {
        cprintf("not valid addr %x, and  can not find it vma range[%x, %x]\n", addr, vma->vm_start
, vma->vm_end);
        goto failed;
    }
    cprintf("valid addr %x, and find it in vma range[%x, %x]\n", addr, vma->vm_start, vma->vm_end)
;
    switch (error_code & 3) {
    default:
            /* default is 3: write, present */
    case 2: /* write, not present */
        if (!(vma->vm_flags & VM_WRITE)) {
            cprintf("write, not present in do_pgfault failed\n");
            goto failed;
        }
        break;
    case 1: /* read, present */
        cprintf("read, present in do_pgfault failed\n");
        goto failed;
    case 0: /* read, not present */
        if (!(vma->vm_flags & (VM_READ | VM_EXEC))) {
            cprintf("read, not present in do_pgfault failed\n");
            goto failed;
        }
    }

    uint32_t perm = PTE_U;
    if (vma->vm_flags & VM_WRITE) {
        perm |= PTE_W;
    }
    addr = ROUNDDOWN(addr, PGSIZE);

    ret = -E_NO_MEM;

    pte_t *ptep;
    // try to find a pte, if pte's PT(Page Table) isn't existed, then create a PT.
    // (notice the 3th parameter '1')
    if ((ptep = get_pte(mm->pgdir, addr, 1)) == NULL) {
        cprintf("get_pte in do_pgfault failed\n");
        goto failed;
    }

    if (*ptep == 0) { // if the phy addr isn't exist, then alloc a page & map the phy addr with lo
gical addr
        if (...(3)... == NULL) {
            cprintf("pgdir_alloc_page in do_pgfault failed\n");
            goto failed;
        }
    }
    else {
        struct Page *page=NULL;
        cprintf("do pgfault: ptep %x, pte %x\n",ptep, *ptep);
        if (*ptep & PTE_P) {
            page = ...(4)...;
        } else{
           // if this pte is a swap entry, then load data from disk to a page with phy addr
           // and call page_insert to map the phy addr with logical addr
           if(swap_init_ok) {
               if ((ret = ...(5)...) != 0) {
                   cprintf("swap_in in do_pgfault failed\n");
```

```
                    goto failed;
                }


            }
            else {
             cprintf("no swap_init_ok but ptep is %x, failed\n",*ptep);
             goto failed;
            }
        }
        page_insert(mm->pgdir, page, addr, perm);
        swap_map_swappable(mm, addr, page, 1);
    }
    ret = 0;
failed:
    return ret;
}

=============kern/mm/swap.c=============



...
int
swap_out(struct mm_struct *mm, int n, int in_tick)
{
     int i;
     for (i = 0; i != n; ++ i)
     {
          uintptr_t v;
          //struct Page **ptr_page=NULL;
          struct Page *page;
          // cprintf("i %d, SWAP: call swap_out_victim\n",i);
          int r = sm->swap_out_victim(mm, &page, in_tick);
          if (r != 0) {
                  cprintf("i %d, swap_out: call swap_out_victim failed\n",i);
                  break;
          }
          //assert(!PageReserved(page));

          //cprintf("SWAP: choose victim page 0x%08x\n", page);

          v=page->pra_vaddr;
          pte_t *ptep = get_pte(mm->pgdir, v, 0);
          assert((*ptep & PTE_P) != 0);

          if (...(6)... != 0) {
```

```
                    cprintf("SWAP: failed to save\n");
                    sm->map_swappable(mm, v, page, 0);
                    continue;
          }
          else {
                    cprintf("swap_out: i %d, store page in vaddr 0x%x to disk swap entry %d\n", i,
 v, page->pra_vaddr/PGSIZE+1);
                    *ptep = (page->pra_vaddr/PGSIZE+1)<<8;
                    free_page(page);
          }

          tlb_invalidate(mm->pgdir, v);
     }
     return i;
}
int
swap_in(struct mm_struct *mm, uintptr_t addr, struct Page **ptr_result)
{
     struct Page *result = alloc_page();
     assert(result!=NULL);

     pte_t *ptep = get_pte(mm->pgdir, addr, 0);
     // cprintf("SWAP: load ptep %x swap entry %d to vaddr 0x%08x, page %x, No %d\n", ptep, (*ptep
)>>8, addr, result, (result-pages));

     int r;
     if ((r = ...(7)...) != 0)
     {
        assert(r!=0);
     }
     cprintf("swap_in: load disk swap entry %d with swap_page in vadr 0x%x free_area.nr_free %d\n"
, (*ptep)>>8, addr, free_area.nr_free);
     *ptr_result=result;
     return 0;
}


=============kern/mm/pmm.h=============

...
#define alloc_page() alloc_pages(1)
#define free_page(page) free_pages(page, 1)


=============kern/mm/pmm.c=============
...

// pgdir_alloc_page - call alloc_page & page_insert functions to
//                    - allocate a page size memory & setup an addr map
//                    - pa<->la with linear address la and the PDT pgdir
struct Page *
pgdir_alloc_page(pde_t *pgdir, uintptr_t la, uint32_t perm) {
    struct Page *page = alloc_page();
    if (page != NULL) {
        if (page_insert(pgdir, page, la, perm) != 0) {
            free_page(page);
            return NULL;
        }
        if (swap_init_ok){
            if(check_mm_struct!=NULL) {
                swap_map_swappable(check_mm_struct, la, page, 0);
                page->pra_vaddr=la;
                assert(page_ref(page) == 1);
                //cprintf("get No. %d  page: pra_vaddr %x, pra_link.prev %x, pra_link_next %x in p
gdir_alloc_page\n", (page-pages), page->pra_vaddr,page->pra_page_link.prev, page->pra_page_link.ne
xt);
            }
            else  {  //now current is existed, should fix it in the future
                //swap_map_swappable(current->mm, la, page, 0);
                //page->pra_vaddr=la;
                //assert(page_ref(page) == 1);
                //panic("pgdir_alloc_page: no pages. now current is existed, should fix it in the
future\n");
            }
        }
    }

    return page;
}



=============kern/fs/swapfs.c=============



...
int
swapfs_read(swap_entry_t entry, struct Page *page) {
    return ide_read_secs(SWAP_DEV_NO, swap_offset(entry) * PAGE_NSECT, page2kva(page), PAGE_NSECT)
;
}
```

```
int
swapfs_write(swap_entry_t entry, struct Page *page) {
    return ide_write_secs(SWAP_DEV_NO, swap_offset(entry) * PAGE_NSECT, page2kva(page), PAGE_NSECT
);
}

=============kern/mm/swap_fifo.c=============

...
struct swap_manager swap_manager_fifo =
{
     .name            = "fifo swap manager",
     .init            = &_fifo_init,
     .init_mm         = &_fifo_init_mm,
     .tick_event      = &_fifo_tick_event,
     .map_swappable   = &_fifo_map_swappable,
     .set_unswappable = &_fifo_set_unswappable,
     .swap_out_victim = &_fifo_swap_out_victim,
     .check_swap      = &_fifo_check_swap,
};
```

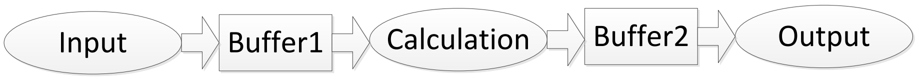
二、(15 分)调度器是操作系统内核中依据调度算法进行进程切换选择的模块。请完成如下任务。
1）试描述步进调度算法(Stride Scheduling)的基本原理。
2）请给出下面测试程序（user/priority.c）执行时的进程调度顺序。建议说明每次进程切换后当前执行进程的 ID、lab6_priority、lab6_stride 和已切换次数。

```
=============kern/process/proc.h=============

...
struct proc_struct {
    enum proc_state state;                      // Process state
    int pid;                                    // Process ID
    int runs;                                   // the running times of Proces
    uintptr_t kstack;                           // Process kernel stack
    volatile bool need_resched;                 // bool value: need to be rescheduled to release C
PU?
    struct proc_struct *parent;                 // the parent process
    struct mm_struct *mm;                       // Process's memory management field
    struct context context;                     // Switch here to run process
    struct trapframe *tf;                       // Trap frame for current interrupt
    uintptr_t cr3;                              // CR3 register: the base addr of Page Directroy T
able(PDT)
```

```
    uint32_t flags;                           // Process flag
    char name[PROC_NAME_LEN + 1];             // Process name
    list_entry_t list_link;                   // Process link list
    list_entry_t hash_link;                   // Process hash list
    int exit_code;                            // exit code (be sent to parent proc)
    uint32_t wait_state;                      // waiting state
    struct proc_struct *cptr, *yptr, *optr;   // relations between processes
    struct run_queue *rq;                     // running queue contains Process
    list_entry_t run_link;                    // the entry linked in run queue
    int time_slice;                           // time slice for occupying the CPU
    skew_heap_entry_t lab6_run_pool;          // FOR LAB6 ONLY: the entry in the run pool
    uint32_t lab6_stride;                     // FOR LAB6 ONLY: the current stride of the proces
s
    uint32_t lab6_priority;                   // FOR LAB6 ONLY: the priority of process, set by
lab6_set_priority(uint32_t)
};



=============user/priority.c=============



#include <ulib.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define TOTAL 6
/* to get enough accuracy, MAX_TIME (the running time of each process) should >1000 mseconds. */
#define MAX_TIME  2000
unsigned int acc[TOTAL];
int status[TOTAL];
int pids[TOTAL];

static void
spin_delay(void)
{
     int i;
     volatile int j;
     for (i = 0; i != 200; ++ i)
     {
          j = !j;
     }
}
```

```
int
main(void) {
     int i,time;
     memset(pids, 0, sizeof(pids));
     lab6_set_priority(TOTAL + 1);

     for (i = 0; i < TOTAL; i ++) {
          acc[i]=0;
          if ((pids[i] = fork()) == 0) {
               lab6_set_priority(i + 1);
               acc[i] = 0;
               while (1) {
                    spin_delay();
                    ++ acc[i];
                    if(acc[i]%4000==0) {
                        if((time=gettime_msec())>MAX_TIME) {
                            cprintf("child pid %d, acc %d, time %d\n",getpid(),acc[i],time);
                            exit(acc[i]);
                        }
                    }
               }

          }
          if (pids[i] < 0) {
               goto failed;
          }
     }

     cprintf("main: fork ok,now need to wait pids.\n");

     for (i = 0; i < TOTAL; i ++) {
         status[i]=0;
         waitpid(pids[i],&status[i]);
         cprintf("main: pid %d, acc %d, time %d\n",pids[i],status[i],gettime_msec());
     }
     cprintf("main: wait pids over\n");
     cprintf("stride sched correct result:");
     for (i = 0; i < TOTAL; i ++)
     {
         cprintf(" %d", (status[i] * 2 / status[0] + 1) / 2);
     }
     cprintf("\n");

     return 0;

failed:
    for (i = 0; i < TOTAL; i ++) {
        if (pids[i] > 0) {
            kill(pids[i]);
        }
    }
    panic("FAIL: T.T\n");
}
```

三、(20 分)一个从键盘输入到打印机输出的数据处理流程图如下图所示。其中键盘输入进程（input_process）通过缓冲区 Buffer1 把数据传送给计算进程（calculation_process），计算进程把处理结果通过 Buffer2 传送给打印进程（output_process）。键盘输入进程、计算进程及打印进程对缓冲区 Buffer1 和 Buffer2 的访问满足下列条件。



1）任何时刻只有一个进程在对缓冲区 Buffer1 进行数据读写操作；只有一个进程在对缓冲区 Buffer2 进行数据读写操作；允许进程对 Buffer1 和 Buffer2 的同时读写操作。
2）两个缓冲区的大小是无限大。
请用信号量（semaphore）机制实现键盘输入进程 input_process()、计算进程 calculation_process()和打印进程 output_process()。要求：用信号量方法（不允许使用信号量集），并给出信号量定义和初始值；在代码中要有适当的注释，以说明信号量定义的作用和代码的含义；用类 C 语言描述共享变量和函数。

四、Bakery 算法(Lamport 1979)是一种解决 n 个线程访问临界区（Critical Section）问题的软件同步算法。该算法定义了两个共享数组如下：

```
boolean choosing[n];
int      number[n]; //ticket
```

所有 choosing[i]的初始值为 false，而 number[i]的初始值为 0。进程 i 访问临界区的伪代码如下。

```
do {
    choosing[i] = true;
    number[i] = max(number[0], number[1], …, number [n – 1])+1;
    choosing[i] = false;

    for (j = 0; j < n; j++) {
       while (choosing[j]) ; //（A）
       while ((number[j] != 0) && (  (number[j],j) < (number[i],i)  ) ) ;
    }
    critical section
    number[i] = 0;
    remainder section
} while (1);
```

请完成如下任务。
1）删除（A）行的代码，算法还能正确控制对临界区的访问吗？
2）如果能，请说明理由；如果不能，请给出一个出现错误的情形，并解释原因。

五、（20 分）在 ucore 中采用的文件系统是 UNIX 文件系统 UFS 的简化版本 SFS。请完成如下任务。
1）描述 UFS 的多级间接索引文件（Multi-level Indexed Allocation）的存储结构；
2）补全下面文件系统代码。

```
=============kern/fs/sfs/sfs.h=============


#ifndef __KERN_FS_SFS_SFS_H__
#define __KERN_FS_SFS_SFS_H__

#include <defs.h>
#include <mmu.h>
#include <list.h>
#include <sem.h>
#include <unistd.h>

#define SFS_MAGIC                                   0x2f8dbe2a              /* magic number for sf
s */
#define SFS_BLKSIZE                                 PGSIZE                  /* size of block */
#define SFS_NDIRECT                                 12                      /* # of direct blocks
in inode */
#define SFS_MAX_INFO_LEN                            31                      /* max length of infom
ation */
#define SFS_MAX_FNAME_LEN                           FS_MAX_FNAME_LEN        /* max length of filen
ame */
#define SFS_MAX_FILE_SIZE                           (1024UL * 1024 * 128)   /* max file size (128M
) */
#define SFS_BLKN_SUPER                              0                       /* block the superbloc
k lives in */
#define SFS_BLKN_ROOT                               1                       /* location of the roo
t dir inode */
#define SFS_BLKN_FREEMAP                            2                       /* 1st block of the fr
eemap */

/* # of bits in a block */
#define SFS_BLKBITS                                 (SFS_BLKSIZE * CHAR_BIT)

/* # of entries in a block */
```

```
#define SFS_BLK_NENTRY                                  (SFS_BLKSIZE / sizeof(uint32_t))

/* file types */
#define SFS_TYPE_INVAL                                  0       /* Should not appear on disk */
#define SFS_TYPE_FILE                                   1
#define SFS_TYPE_DIR                                    2
#define SFS_TYPE_LINK                                   3

/*
 * On-disk superblock
 */
struct sfs_super {
    uint32_t magic;                                     /* magic number, should be SFS_MAGIC */
    uint32_t blocks;                                    /* # of blocks in fs */
    uint32_t unused_blocks;                             /* # of unused blocks in fs */
    char info[SFS_MAX_INFO_LEN + 1];                    /* infomation for sfs  */
};

/* inode (on disk) */
struct sfs_disk_inode {
    uint32_t size;                                      /* size of the file (in bytes) */
    uint16_t type;                                      /* one of SYS_TYPE_* above */
    uint16_t nlinks;                                    /* # of hard links to this file */
    uint32_t blocks;                                    /* # of blocks */
    uint32_t direct[SFS_NDIRECT];                       /* direct blocks */
    uint32_t indirect;                                  /* indirect blocks */
//    uint32_t db_indirect;                               /* double indirect blocks */
//   unused
};

/* file entry (on disk) */
struct sfs_disk_entry {
    uint32_t ino;                                       /* inode number */
    char name[SFS_MAX_FNAME_LEN + 1];                   /* file name */
};

#define sfs_dentry_size                                 \
    sizeof(((struct sfs_disk_entry *)0)->name)

/* inode for sfs */
struct sfs_inode {
    struct sfs_disk_inode *din;                         /* on-disk inode */
    uint32_t ino;                                       /* inode number */
    bool dirty;                                         /* true if inode modified */
    int reclaim_count;                                  /* kill inode if it hits zero */
    semaphore_t sem;                                    /* semaphore for din */
    list_entry_t inode_link;                        /* entry for linked-list in sfs_fs */
    list_entry_t hash_link;                         /* entry for hash linked-list in sfs_fs */
};

#define le2sin(le, member)                          \
    to_struct((le), struct sfs_inode, member)

/* filesystem for sfs */
struct sfs_fs {
    struct sfs_super super;                         /* on-disk superblock */
    struct device *dev;                             /* device mounted on */
    struct bitmap *freemap;                         /* blocks in use are mared 0 */
    bool super_dirty;                               /* true if super/freemap modified */
    void *sfs_buffer;                               /* buffer for non-block aligned io */
    semaphore_t fs_sem;                             /* semaphore for fs */
    semaphore_t io_sem;                             /* semaphore for io */
    semaphore_t mutex_sem;                          /* semaphore for link/unlink and rename */
    list_entry_t inode_list;                        /* inode linked-list */
    list_entry_t *hash_list;                        /* inode hash linked-list */
};

/* hash for sfs */
#define SFS_HLIST_SHIFT                             10
#define SFS_HLIST_SIZE                              (1 << SFS_HLIST_SHIFT)
#define sin_hashfn(x)                               (hash32(x, SFS_HLIST_SHIFT))

/* size of freemap (in bits) */
#define sfs_freemap_bits(super)                     ROUNDUP((super)->blocks, SFS_BLKBITS)

/* size of freemap (in blocks) */
#define sfs_freemap_blocks(super)                   ROUNDUP_DIV((super)->blocks, SFS_BLKBITS)

struct fs;
struct inode;

void sfs_init(void);
int sfs_mount(const char *devname);

void lock_sfs_fs(struct sfs_fs *sfs);
void lock_sfs_io(struct sfs_fs *sfs);
void lock_sfs_mutex(struct sfs_fs *sfs);
void unlock_sfs_fs(struct sfs_fs *sfs);
void unlock_sfs_io(struct sfs_fs *sfs);
void unlock_sfs_mutex(struct sfs_fs *sfs);

int sfs_rblock(struct sfs_fs *sfs, void *buf, uint32_t blkno, uint32_t nblks);
int sfs_wblock(struct sfs_fs *sfs, void *buf, uint32_t blkno, uint32_t nblks);
int sfs_rbuf(struct sfs_fs *sfs, void *buf, size_t len, uint32_t blkno, off_t offset);
int sfs_wbuf(struct sfs_fs *sfs, void *buf, size_t len, uint32_t blkno, off_t offset);
int sfs_sync_super(struct sfs_fs *sfs);
int sfs_sync_freemap(struct sfs_fs *sfs);
int sfs_clear_block(struct sfs_fs *sfs, uint32_t blkno, uint32_t nblks);

int sfs_load_inode(struct sfs_fs *sfs, struct inode **node_store, uint32_t ino);

#endif /* !__KERN_FS_SFS_SFS_H__ */



=============kern/fs/sfs/sfs_inode.c=============


......
static int
sfs_bmap_get_sub_nolock(struct sfs_fs *sfs, uint32_t *entp, uint32_t index, bool create, uint32_t
*ino_store) {
    assert(index < SFS_BLK_NENTRY);
    int ret;
    uint32_t ent, ino = 0;
    off_t offset = index * sizeof(uint32_t);
    if ((ent = *entp) != 0) {
        if ((ret = ...(1)...) != 0) {
            return ret;
        }
        if (ino != 0 || !create) {
            goto out;
        }
    }
    else {
        if (!create) {
            goto out;
        }
        if ((ret = sfs_block_alloc(sfs, &ent)) != 0) {
            return ret;
        }
    }

    if ((ret = sfs_block_alloc(sfs, &ino)) != 0) {
        goto failed_cleanup;
    }
    if ((ret = ...(2)...) != 0) {
        sfs_block_free(sfs, ino);
        goto failed_cleanup;
    }

out:
    if (ent != *entp) {
        *entp = ent;
    }
    *ino_store = ino;
    return 0;

failed_cleanup:
    if (ent != *entp) {
        sfs_block_free(sfs, ent);
    }
    return ret;
}

......
static int
sfs_bmap_get_nolock(struct sfs_fs *sfs, struct sfs_inode *sin, uint32_t index, bool create, uint32
_t *ino_store) {
    struct sfs_disk_inode *din = sin->din;
    int ret;
    uint32_t ent, ino;
    if (...(3)...) {
        if ((ino = din->direct[index]) == 0 && create) {
            if ((ret = sfs_block_alloc(sfs, &ino)) != 0) {
                return ret;
            }
            din->direct[index] = ino;
            sin->dirty = 1;
        }
        goto out;
    }

    index = ...(4)...;
    if (index < SFS_BLK_NENTRY) {
        ent = din->indirect;
        if ((ret = ...(5)...) != 0) {
            return ret;
        }
        if (ent != din->indirect) {
            assert(din->indirect == 0);
            din->indirect = ent;
            sin->dirty = 1;
        }
        goto out;
    }

    index = ...(6)...;
    if ((ent = ino) != 0) {
        if ((ret = sfs_bmap_get_sub_nolock(sfs, &ent, index % SFS_BLK_NENTRY, create, &ino)) != 0)
 {
            return ret;
        }
    }

out:
    assert(ino == 0 || sfs_block_inuse(sfs, ino));
    *ino_store = ino;
    return 0;
}
......
static int
sfs_io_nolock(struct sfs_fs *sfs, struct sfs_inode *sin, void *buf, off_t offset, size_t *alenp, b
ool write) {
    struct sfs_disk_inode *din = sin->din;
    assert(din->type != SFS_TYPE_DIR);
    off_t endpos = offset + *alenp, blkoff;
    *alenp = 0;
    if (offset < 0 || offset >= SFS_MAX_FILE_SIZE || offset > endpos) {
        return -E_INVAL;
    }
    if (offset == endpos) {
        return 0;
    }
    if (endpos > SFS_MAX_FILE_SIZE) {
        endpos = SFS_MAX_FILE_SIZE;
    }
    if (!write) {
        if (offset >= din->size) {
            return 0;
        }
        if (endpos > din->size) {
            endpos = din->size;
        }
    }

    int (*sfs_buf_op)(struct sfs_fs *sfs, void *buf, size_t len, uint32_t blkno, off_t offset);
    int (*sfs_block_op)(struct sfs_fs *sfs, void *buf, uint32_t blkno, uint32_t nblks);
    if (write) {
        sfs_buf_op = ...(7.1)..., sfs_block_op = ...(7.2)...;
    }
    else {
        sfs_buf_op = sfs_rbuf, sfs_block_op = sfs_rblock;
    }

    int ret = 0;
    size_t size, alen = 0;
    uint32_t ino;
    uint32_t blkno = offset / SFS_BLKSIZE;
    uint32_t nblks = endpos / SFS_BLKSIZE - blkno;

    if ((blkoff = offset % SFS_BLKSIZE) != 0) {
        size = (nblks != 0) ? (SFS_BLKSIZE - blkoff) : (endpos - offset);
        if ((ret = sfs_bmap_load_nolock(sfs, sin, blkno, &ino)) != 0) {
            goto out;
        }
        if ((ret = ...(8)...) != 0) {
            goto out;
        }
        alen += size;
        if (nblks == 0) {
            goto out;
        }
        buf += size, blkno ++, nblks --;
    }

    size = SFS_BLKSIZE;
    while (nblks != 0) {
        if ((ret = sfs_bmap_load_nolock(sfs, sin, blkno, &ino)) != 0) {
            goto out;
        }
        if ((ret = sfs_block_op(sfs, buf, ino, 1)) != 0) {
            goto out;
        }
        alen += size, buf += size, blkno ++, nblks --;
    }

    if ((size = endpos % SFS_BLKSIZE) != 0) {
        if ((ret = sfs_bmap_load_nolock(sfs, sin, blkno, &ino)) != 0) {
            goto out;
        }
        if ((ret = sfs_buf_op(sfs, buf, size, ino, 0)) != 0) {
            goto out;
        }
        alen += size;
    }
```

```
out:
    *alenp = alen;
    if (offset + alen > sin->din->size) {
        sin->din->size = offset + alen;
        sin->dirty = 1;
    }
    return ret;
}



......
static int
sfs_read(struct inode *node, struct iobuf *iob) {
    return ...(9)...;
}
static int
sfs_write(struct inode *node, struct iobuf *iob) {
    return ...(10)...;
}
......
static inline int
sfs_io(struct inode *node, struct iobuf *iob, bool write) {
    struct sfs_fs *sfs = fsop_info(vop_fs(node), sfs);
    struct sfs_inode *sin = vop_info(node, sfs_inode);
    int ret;
    lock_sin(sin);
    {
        size_t alen = iob->io_resid;
        ret = ...(11)...;
        if (alen != 0) {
            iobuf_skip(iob, alen);
        }
    }
    unlock_sin(sin);
    return ret;
}
......
static const struct inode_ops sfs_node_dirops = {
    .vop_magic                    = VOP_MAGIC,
    .vop_open                     = sfs_opendir,
    .vop_close                    = sfs_close,
    .vop_read                     = NULL_VOP_ISDIR,
    .vop_write                    = NULL_VOP_ISDIR,
    .vop_fstat                    = sfs_fstat,
    .vop_fsync                    = sfs_fsync,
    .vop_mkdir                    = NULL_VOP_UNIMP,
    .vop_link                       = NULL_VOP_UNIMP,
    .vop_rename                     = NULL_VOP_UNIMP,
    .vop_readlink                   = NULL_VOP_ISDIR,
    .vop_symlink                    = NULL_VOP_UNIMP,
    .vop_namefile                   = sfs_namefile,
    .vop_getdirentry                = sfs_getdirentry,
    .vop_reclaim                    = sfs_reclaim,
    .vop_ioctl                      = NULL_VOP_INVAL,
    .vop_gettype                    = sfs_gettype,
    .vop_tryseek                    = NULL_VOP_ISDIR,
    .vop_truncate                   = NULL_VOP_UNIMP,
    .vop_create                     = NULL_VOP_UNIMP,
    .vop_unlink                     = NULL_VOP_UNIMP,
    .vop_lookup                     = sfs_lookup,
    .vop_lookup_parent              = NULL_VOP_UNIMP,
};
static const struct inode_ops sfs_node_fileops = {
    .vop_magic                      = VOP_MAGIC,
    .vop_open                       = sfs_openfile,
    .vop_close                      = sfs_close,
    .vop_read                       = sfs_read,
    .vop_write                      = sfs_write,
    .vop_fstat                      = sfs_fstat,
    .vop_fsync                      = sfs_fsync,
    .vop_mkdir                      = NULL_VOP_NOTDIR,
    .vop_link                       = NULL_VOP_NOTDIR,
    .vop_rename                     = NULL_VOP_NOTDIR,
    .vop_readlink                   = NULL_VOP_NOTDIR,
    .vop_symlink                    = NULL_VOP_NOTDIR,
    .vop_namefile                   = NULL_VOP_NOTDIR,
    .vop_getdirentry                = NULL_VOP_NOTDIR,
    .vop_reclaim                    = sfs_reclaim,
    .vop_ioctl                      = NULL_VOP_INVAL,
    .vop_gettype                    = sfs_gettype,
    .vop_tryseek                    = sfs_tryseek,
    .vop_truncate                   = sfs_truncfile,
    .vop_create                     = NULL_VOP_NOTDIR,
    .vop_unlink                     = NULL_VOP_NOTDIR,
    .vop_lookup                     = NULL_VOP_NOTDIR,
    .vop_lookup_parent              = NULL_VOP_NOTDIR,
};
```

六、(15 分)某计算机系统中有 M 个同类型共享资源，有 N 个进程竞争使用，每个进程最多需要 K 个共享资源。该系统不会发生死锁的 K 的最大值是多少？要求给出计算过程，并说明理由。

$M = N(k-1)+1$