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：2012 年 06 月 18 日下午 2:30~4:30

任课教师：_____ 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 23 页。
5. 考试完毕, 必须将试题纸和答卷本一起交回。

一、(18 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。

1）试描述步进调度算法(Stride Scheduling)的基本原理。

2）请补全下面 ucore 代码中调度器和步进调度算法实现中所缺代码，以实现调度器和调度算法的功能。提示：每处需要补全的代码最少只需要一行，一共有 9 个空要填。当然，你可以在需要补全代码的地方写多行来表达需要实现的功能，也允许修改已给出的代码。

3）试描述斜堆(skew heap)在这个步进调度算法中的作用。

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

```
#ifndef __KERN_PROCESS_PROC_H__
#define __KERN_PROCESS_PROC_H__

#include <defs.h>
#include <list.h>
#include <trap.h>
#include <memlayout.h>
#include <skew_heap.h>


// process's state in his life cycle
enum proc_state {
    PROC_UNINIT = 0,  // uninitialized
    PROC_SLEEPING,    // sleeping
    PROC_RUNNABLE,    // runnable(maybe running)
    PROC_ZOMBIE,      // almost dead, and wait parent proc to reclaim his resource
};

// Saved registers for kernel context switches.
// Don't need to save all the %fs etc. segment registers,
// because they are constant across kernel contexts.
// Save all the regular registers so we don't need to care
// which are caller save, but not the return register %eax.
// (Not saving %eax just simplifies the switching code.)
// The layout of context must match code in switch.S.
struct context {
    uint32_t eip;
    uint32_t esp;
    uint32_t ebx;
    uint32_t ecx;
    uint32_t edx;
    uint32_t esi;
    uint32_t edi;
    uint32_t ebp;
```

```
};

#define PROC_NAME_LEN               15
#define MAX_PROCESS                 4096
#define MAX_PID                     (MAX_PROCESS * 2)

extern list_entry_t proc_list;

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)
};

#define PF_EXITING                  0x00000001      // getting shutdown

#define WT_CHILD                    (0x00000001 | WT_INTERRUPTED)
#define WT_INTERRUPTED               0x80000000                    // the wait state could be inte
rrupted


#define le2proc(le, member)         \
    to_struct((le), struct proc_struct, member)

extern struct proc_struct *idleproc, *initproc, *current;

void proc_init(void);
void proc_run(struct proc_struct *proc);
int kernel_thread(int (*fn)(void *), void *arg, uint32_t clone_flags);

char *set_proc_name(struct proc_struct *proc, const char *name);
char *get_proc_name(struct proc_struct *proc);
void cpu_idle(void) __attribute__((noreturn));

struct proc_struct *find_proc(int pid);
int do_fork(uint32_t clone_flags, uintptr_t stack, struct trapframe *tf);
int do_exit(int error_code);
int do_yield(void);
int do_execve(const char *name, size_t len, unsigned char *binary, size_t size);
int do_wait(int pid, int *code_store);
int do_kill(int pid);
```

```
void lab6_set_priority(uint32_t priority);

#endif /* !__KERN_PROCESS_PROC_H__ */
```

==========kern/schedule/default_sched.c==========

```
#include <defs.h>
#include <list.h>
#include <proc.h>
#include <assert.h>
#include <default_sched.h>

#define USE_SKEW_HEAP 1

/* You should define the BigStride constant here*/
/* LAB6: YOUR CODE */
#define BIG_STRIDE    0x7FFFFFFF /* ??? */

/* The compare function for two skew_heap_node_t's and the
 * corresponding procs*/
static int
proc_stride_comp_f(void *a, void *b)
{
     struct proc_struct *p = le2proc(a, lab6_run_pool);
     struct proc_struct *q = le2proc(b, lab6_run_pool);
     int32_t c = p->lab6_stride - q->lab6_stride;
     if (c > 0) return 1;
     else if (c == 0) return 0;
     else return -1;
}

/*
 * stride_init initializes the run-queue rq with correct assignment for
 * member variables, including:
 *
 *   - run_list: should be a empty list after initialization.
 *   - lab6_run_pool: NULL
 *   - proc_num: 0
 *   - max_time_slice: no need here, the variable would be assigned by the caller.
 *
 * hint: see proj13.1/libs/list.h for routines of the list structures.
 */
static void
stride_init(struct run_queue *rq) {
     /* LAB6: YOUR CODE */
     list_init(&(rq->run_list));
     rq->lab6_run_pool = NULL;
     rq->proc_num = 0;
}

/*
 * stride_enqueue inserts the process ``proc'' into the run-queue
 * ``rq''. The procedure should verify/initialize the relevant members
 * of ``proc'', and then put the ``lab6_run_pool'' node into the
 * queue(since we use priority queue here). The procedure should also
 * update the meta date in ``rq'' structure.
 *
 * proc->time_slice denotes the time slices allocation for the
 * process, which should set to rq->max_time_slice.
 *
 * hint: see proj13.1/libs/skew_heap.h for routines of the priority
 * queue structures.
 */
static void
stride_enqueue(struct run_queue *rq, struct proc_struct *proc) {
```

```
     /* LAB6: YOUR CODE */
#if USE_SKEW_HEAP
     rq->lab6_run_pool = ……(1)……;
#else
     assert(list_empty(&(proc->run_link)));
     list_add_before(&(rq->run_list), &(proc->run_link));
#endif
     if (proc->time_slice == 0 || proc->time_slice > rq->max_time_slice) {
          proc->time_slice = rq->max_time_slice;
     }
     proc->rq = rq;
     rq->proc_num ++;
}

/*
 * stride_dequeue removes the process ``proc'' from the run-queue
 * ``rq'', the operation would be finished by the skew_heap_remove
 * operations. Remember to update the ``rq'' structure.
 *
 * hint: see proj13.1/libs/skew_heap.h for routines of the priority
 * queue structures.
 */
static void
stride_dequeue(struct run_queue *rq, struct proc_struct *proc) {
     /* LAB6: YOUR CODE */
#if USE_SKEW_HEAP
     rq->lab6_run_pool =  ……(2)……;
#else
     assert(!list_empty(&(proc->run_link)) && proc->rq == rq);
     list_del_init(&(proc->run_link));
#endif
     rq->proc_num --;
}
/*
 * stride_pick_next pick the element from the ``run-queue'', with the
 * minimum value of stride, and returns the corresponding process
 * pointer. The process pointer would be calculated by macro le2proc,
 * see proj13.1/kern/process/proc.h for definition. Return NULL if
 * there is no process in the queue.
 *
 * When one proc structure is selected, remember to update the stride
 * property of the proc. (stride += BIG_STRIDE / priority)
 *
 * hint: see proj13.1/libs/skew_heap.h for routines of the priority
 * queue structures.
 */
static struct proc_struct *
stride_pick_next(struct run_queue *rq) {
     /* LAB6: YOUR CODE */
#if USE_SKEW_HEAP
     if (rq->lab6_run_pool == NULL) return NULL;
     struct proc_struct *p = le2proc(rq->lab6_run_pool, lab6_run_pool);
#else
     list_entry_t *le = list_next(&(rq->run_list));

     if (le == &rq->run_list)
          return NULL;

     struct proc_struct *p = le2proc(le, run_link);
     le = list_next(le);
     while (le != &rq->run_list)
     {
          struct proc_struct *q = le2proc(le, run_link);
          if ((int32_t)(p->lab6_stride - q->lab6_stride) > 0)
               p = q;
```

```
          le = list_next(le);
     }
#endif
     if (p->lab6_priority == 0)
          p->lab6_stride += BIG_STRIDE;
     else p->lab6_stride = ……(3)……;
     return p;
}

/*
 * stride_proc_tick works with the tick event of current process. You
 * should check whether the time slices for current process is
 * exhausted and update the proc struct ``proc''. proc->time_slice
 * denotes the time slices left for current
 * process. proc->need_resched is the flag variable for process
 * switching.
 */
static void
stride_proc_tick(struct run_queue *rq, struct proc_struct *proc) {
     /* LAB6: YOUR CODE */
     if (proc->time_slice > 0) {
          ……(4)……;
     }
     if (proc->time_slice == 0) {
          ……(5)……;
     }
}

struct sched_class default_sched_class = {
     .name = "stride_scheduler",
     .init = stride_init,
     .enqueue = ……(6)……,
     .dequeue = ……(7)……,
     .pick_next = ……(8)……,
     .proc_tick = ……(9)……,
};
```

==========libs/skew_heap.h==========

```
#ifndef __LIBS_SKEW_HEAP_H__
#define __LIBS_SKEW_HEAP_H__

struct skew_heap_entry {
     struct skew_heap_entry *parent, *left, *right;
};

typedef struct skew_heap_entry skew_heap_entry_t;

typedef int(*compare_f)(void *a, void *b);

static inline void skew_heap_init(skew_heap_entry_t *a) __attribute__((always_inline));
static inline skew_heap_entry_t *skew_heap_merge(
     skew_heap_entry_t *a, skew_heap_entry_t *b,
     compare_f comp);
static inline skew_heap_entry_t *skew_heap_insert(
     skew_heap_entry_t *a, skew_heap_entry_t *b,
     compare_f comp) __attribute__((always_inline));
static inline skew_heap_entry_t *skew_heap_remove(
     skew_heap_entry_t *a, skew_heap_entry_t *b,
     compare_f comp) __attribute__((always_inline));

static inline void
skew_heap_init(skew_heap_entry_t *a)
{
     a->left = a->right = a->parent = NULL;
}
```

```
static inline skew_heap_entry_t *
skew_heap_merge(skew_heap_entry_t *a, skew_heap_entry_t *b,
                compare_f comp)
{
     if (a == NULL) return b;
     else if (b == NULL) return a;

     skew_heap_entry_t *l, *r;
     if (comp(a, b) == -1)
     {
          r = a->left;
          l = skew_heap_merge(a->right, b, comp);

          a->left = l;
          a->right = r;
          if (l) l->parent = a;

          return a;
     }
     else
     {
          r = b->left;
          l = skew_heap_merge(a, b->right, comp);

          b->left = l;
          b->right = r;
          if (l) l->parent = b;

          return b;
     }
}

static inline skew_heap_entry_t *
skew_heap_insert(skew_heap_entry_t *a, skew_heap_entry_t *b,
                 compare_f comp)
{
     skew_heap_init(b);
     return skew_heap_merge(a, b, comp);
}

static inline skew_heap_entry_t *
skew_heap_remove(skew_heap_entry_t *a, skew_heap_entry_t *b,
                 compare_f comp)
{
     skew_heap_entry_t *p   = b->parent;
     skew_heap_entry_t *rep = skew_heap_merge(b->left, b->right, comp);
     if (rep) rep->parent = p;

     if (p)
     {
          if (p->left == b)
               p->left = rep;
          else p->right = rep;
          return a;
     }
     else return rep;
}

#endif  /* !__LIBS_SKEW_HEAP_H__ */
```

==========

二、(15 分)公平的读者-写者（Reader-Writer Problem）问题是指，多个读者进程（Reader）与多个

写者进程（Writer）共享一个数据区；读者进程和写者进程对共享数据区的访问满足下列条件。

1）多个读者进程可以同时对共享数据区进行访问；

2）多个写者进程只能对共享数据区进行互斥访问；

3）读者进程与写者进程只能对共享数据区进行互斥访问；

4）当有写者进程等待时，其后到达的读者进程不能先于该写者进程对共享数据区进行访问；

5）当有读者进程等待时，其后到达的写者进程不能先于该读者进程对共享数据区进行访问；

试用信号量机制实现读者进程 Reader（）和写者进程 Writer（）。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。

三、(8 分)某计算机系统中有 18 个共享资源，有 K 个进程竞争使用，每个进程最多需要 3 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。

四、(8 分)给出下面程序 fork-example.cpp 的输出结果；

==========fork-example.cpp==========

```
#include <iostream>
#include <string>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>

using namespace std;

int globalVariable = 2;

main()
{
   string sIdentifier;
   int    iStackVariable = 20;

   pid_t pID = fork();
   if (pID == 0)
   {
      sIdentifier = "Child Process: ";
      globalVariable++;
      iStackVariable++;
    }
    else if (pID < 0)
    {
        cerr << "Failed to fork" << endl;
        exit(1);
    }
    else
    {
      sIdentifier = "Parent Process:";
    }
    cout << sIdentifier;
    cout << " Global variable: " << globalVariable;
    cout << " Stack variable: "  << iStackVariable << endl;
}
```

==========

五、(16 分)下面是 ucore 内核中与 yield()系统调用实现相关源代码，可实现用户线程主动放弃 CPU 使用权的功能。

1）试描述 ucore 中用户进程利用 yield()进行主动让出 CPU 的工作过程；

2）请补全其中所缺的代码，以正确完成从用户态函数 yield()的功能。提示：每处需要补全的代码最少只需要一行，一共有 11 个空要填。当然，你可以在需要补全代码的地方写多行来表达需要实现的功能，也允许修改已给出的代码。

========== libs-user-ucore/syscall.h ==========

```
#ifndef __USER_LIBS_SYSCALL_H__
#define __USER_LIBS_SYSCALL_H__

#include <types.h>

......
int sys_yield(void);
......
#endif /* !__USER_LIBS_SYSCALL_H__ */
```

==========libs-user-ucore/arch/i386/syscall.c==========

```
#include <unistd.h>
#include <types.h>
#include <stdarg.h>
#include <syscall.h>
#include <mboxbuf.h>
#include <stat.h>
#include <dirent.h>

#define MAX_ARGS            5

uint32_t
syscall(int num, ...) {
    va_list ap;
    va_start(ap, num);
    uint32_t a[MAX_ARGS];
    int i;
    for (i = 0; i < MAX_ARGS; i ++) {
        a[i] = va_arg(ap, uint32_t);
    }
    va_end(ap);

    uint32_t ret;
    asm volatile (
        "int %1;"
        : "=a" (ret)
        : "i" (T_SYSCALL),
          "a" (num),
          "d" (a[0]),
          "c" (a[1]),
          "b" (a[2]),
          "D" (a[3]),
          "S" (a[4])
        : "cc", "memory");
    return ret;
}
```

==========libs-user-ucore/syscall.c==========

```
#include <types.h>
#include <unistd.h>
#include <stdarg.h>
#include <syscall.h>
#include <mboxbuf.h>
#include <stat.h>
#include <dirent.h>
```

```
extern uintptr_t syscall (int num, ...);

......

int
sys_yield(void) {
    return ……(1)……;
}

......
```

==========kern-ucore/glue-ucore/libs/unistd.h==========

```
#ifndef __LIBS_UNISTD_H__
#define __LIBS_UNISTD_H__

#define T_SYSCALL           0x80

/* syscall number */
......
#define SYS_yield           10
......
#endif /* !__LIBS_UNISTD_H__ */
```

========== kern-ucore/arch/i386/glue-ucore/trap.c ==========

```
......

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

    int ret;
    switch (tf->tf_trapno) {
    case T_DEBUG:
    case T_BRKPT:
        debug_monitor(tf);
        break;
    case T_PGFLT:
        if ((ret = pgfault_handler(tf)) != 0) {
            print_trapframe(tf);
            if (pls_read(current) == NULL) {
                panic("handle pgfault failed. %e\n", ret);
            }
            else {
                if (trap_in_kernel(tf)) {
                    panic("handle pgfault failed in kernel mode. %e\n", ret);
                }
                kprintf("killed by kernel.\n");
                do_exit(-E_KILLED);
            }
        }
        break;
    case ……(2)……:
        syscall();
        break;
    case IRQ_OFFSET + IRQ_TIMER:
        ticks ++;
        assert(pls_read(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 {
            extern void dev_stdin_write(char c);
            dev_stdin_write(c);
        }
        break;
    case IRQ_OFFSET + IRQ_IDE1:
    case IRQ_OFFSET + IRQ_IDE2:
        /* do nothing */
        break;
    default:
        print_trapframe(tf);
        if (pls_read(current) != NULL) {
            kprintf("unhandled trap.\n");
            do_exit(-E_KILLED);
        }
        panic("unexpected trap in kernel.\n");
    }
}

void
trap(struct trapframe *tf) {
    // used for previous projects
    if (pls_read(current) == NULL) {
        trap_dispatch(tf);
    }
    else {
        // keep a trapframe chain in stack
        struct trapframe *otf = pls_read(current)->tf;
        pls_read(current)->tf = tf;

        bool in_kernel = trap_in_kernel(tf);

        trap_dispatch(tf);

        pls_read(current)->tf = otf;
        if (!in_kernel) {
            may_killed();
            if (pls_read(current)->need_resched) {
                ……(3)……;
            }
        }
    }
}
```

==========kern-ucore/schedule/sched.c==========

```
#include <list.h>
#include <sync.h>
#include <proc.h>
#include <sched.h>
#include <stdio.h>
#include <assert.h>
#include <sched_MLFQ.h>
#include <kio.h>
#include <mp.h>

#define current (pls_read(current))
#define idleproc (pls_read(idleproc))

......

#include <vmm.h>
```

```
#define MT_SUPPORT

void
schedule(void) {
    bool intr_flag;
    struct proc_struct *next;
#ifndef MT_SUPPORT
    list_entry_t head;
    int lapic_id = pls_read(lapic_id);
#endif

    local_intr_save(intr_flag);
    int lcpu_count = pls_read(lcpu_count);
    {
        current->need_resched = ……(4)……;
#ifndef MT_SUPPORT
        if (current->mm)
        {
            assert(current->mm->lapic == lapic_id);
            current->mm->lapic = -1;
        }
#endif
        if (current->state == PROC_RUNNABLE && current->pid >= lcpu_count) {
            sched_class_enqueue(current);
        }
#ifndef MT_SUPPORT
        list_init(&head);
        while (1)
        {
            next = ……(5)……;
            if (next != NULL) sched_class_dequeue(next);

            if (next && next->mm && next->mm->lapic != -1)
            {
                list_add(&head, &(next->run_link));
            }
            else
            {
                list_entry_t *cur;
                while ((cur = list_next(&head)) != &head)
                {
                    list_del_init(cur);
                    sched_class_enqueue(le2proc(cur, run_link));
                }

                break;
            }
        }
#else
        next = ……(6)……;
        if (next != NULL)
            sched_class_dequeue(next);
#endif  /* !MT_SUPPORT */
        if (next == NULL) {
            next = ……(7)……;
        }
        next->runs ++;
        /* Collect information here*/
        if (sched_collect_info) {
            int lcpu_count = pls_read(lcpu_count);
            int lcpu_idx = pls_read(lcpu_idx);
            int loc = sched_info_head[lcpu_idx];
            int prev = sched_info_pid[loc*lcpu_count + lcpu_idx];
            if (next->pid == prev)
                sched_info_times[loc*lcpu_count + lcpu_idx] ++;
```

```
            else {
                sched_info_head[lcpu_idx] ++;
                if (sched_info_head[lcpu_idx] >= PGSIZE / sizeof(uint16_t) / lcpu_count)
                    sched_info_head[lcpu_idx] = 0;
                loc = sched_info_head[lcpu_idx];
                uint16_t prev_pid = sched_info_pid[loc*lcpu_count + lcpu_idx];
                uint16_t prev_times = sched_info_times[loc*lcpu_count + lcpu_idx];
                if (prev_times > 0 && prev_pid >= lcpu_count + 2)
                    sched_slices[lcpu_idx][prev_pid % SLICEPOOL_SIZE] += prev_times;
                sched_info_pid[loc*lcpu_count + lcpu_idx] = next->pid;
                sched_info_times[loc*lcpu_count + lcpu_idx] = 1;
            }
        }
#ifndef MT_SUPPORT
        assert(!next->mm || next->mm->lapic == -1);
        if (next->mm)
            next->mm->lapic = lapic_id;
#endif
        if (next != current) {
            ……(8)……;
        }
    }
    local_intr_restore(intr_flag);
}

void
add_timer(timer_t *timer) {
    bool intr_flag;
    local_intr_save(intr_flag);
    {
        assert(timer->expires > 0 && timer->proc != NULL);
        assert(list_empty(&(timer->timer_link)));
        list_entry_t *le = list_next(&timer_list);
        while (le != &timer_list) {
            timer_t *next = le2timer(le, timer_link);
            if (timer->expires < next->expires) {
                next->expires -= timer->expires;
                break;
            }
            timer->expires -= next->expires;
            le = list_next(le);
        }
        list_add_before(le, &(timer->timer_link));
    }
    local_intr_restore(intr_flag);
}
......
```

==========kern-ucore/process/proc.c==========

```
......

// proc_run - make process "proc" running on cpu
// NOTE: before call switch_to, should load  base addr of "proc"'s new PDT
void
proc_run(struct proc_struct *proc) {
    if (proc != current) {
        bool intr_flag;
        struct proc_struct *prev = current, *next = proc;
        // kprintf("(%d) => %d\n", lapic_id, next->pid);
        local_intr_save(intr_flag);
        {
            pls_write(current, proc);
            load_rsp0(next->kstack + KSTACKSIZE);
            mp_set_mm_pagetable(next->mm);
```

```
            ……(9)……;
        }
        local_intr_restore(intr_flag);
    }
}

......

// do_yield - ask the scheduler to reschedule
int
do_yield(void) {
    current->need_resched = ……(10)……;
    return 0;
}
......
```

==========kern-ucore/arch/i386/syscall/syscall.c==========

```
......
static uint32_t
sys_yield(uint32_t arg[]) {
    return ……(11)……;
}

......

static uint32_t (*syscalls[])(uint32_t arg[]) = {
......
    [SYS_yield]             sys_yield,
......
};

#define NUM_SYSCALLS        ((sizeof(syscalls)) / (sizeof(syscalls[0])))

void
syscall(void) {
    struct trapframe *tf = pls_read(current)->tf;
    uint32_t arg[5];
    int num = tf->tf_regs.reg_eax;
    if (num >= 0 && num < NUM_SYSCALLS) {
        if (syscalls[num] != NULL) {
            arg[0] = tf->tf_regs.reg_edx;
            arg[1] = tf->tf_regs.reg_ecx;
            arg[2] = tf->tf_regs.reg_ebx;
            arg[3] = tf->tf_regs.reg_edi;
            arg[4] = tf->tf_regs.reg_esi;
            tf->tf_regs.reg_eax = syscalls[num](arg);
            return ;
        }
    }
    print_trapframe(tf);
    panic("undefined syscall %d, pid = %d, name = %s.\n",
            num, pls_read(current)->pid, pls_read(current)->name);
}
```

==========

六、(18 分)文件系统是操作系统内核中用于持久保存数据的功能模块。

1）试描述 SFS 文件系统中的文件存储组织，即文件内部数据块存储位置和顺序的组织方法；

2）试描述 ucore 文件系统在一个 SFS 文件的最后附加一个新数据块实现方法；

3）试解释下面 ucore 代码中文件系统实现中与 append_block()函数相关的指定代码行的作用。注意：需要解释的代码共有 10 处。

=======================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 sfs */
#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 infomation */
#define SFS_MAX_FNAME_LEN   FS_MAX_FNAME_LEN        /* max length of filename */
#define SFS_MAX_FILE_SIZE   (1024UL * 1024 * 128)   /* max file size (128M) */
#define SFS_BLKN_SUPER      0                    /* block the superblock lives in */
#define SFS_BLKN_ROOT       1                    /* location of the root dir inode */
#define SFS_BLKN_FREEMAP    2                    /* lst block of the freemap */

/* # 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;                                /* ……(1)…… */
    uint32_t direct[SFS_NDIRECT];                   /* ……(2)…… */
    uint32_t indirect;                              /* ……(3)…… */
//    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__ */
```

========== tools/mksfs.c==========

```
......
#define SFS_MAGIC                               0x2f8dbe2a
#define SFS_NDIRECT                             12
#define SFS_BLKSIZE                             4096                                    // 4K
#define SFS_MAX_NBLKS                           (1024UL * 512)                          // 4K * 51
2K
#define SFS_MAX_INFO_LEN                        31
#define SFS_MAX_FNAME_LEN                       255
#define SFS_MAX_FILE_SIZE                       (1024UL * 1024 * 128)                   // 128M

#define SFS_BLKBITS                             (SFS_BLKSIZE * CHAR_BIT)
#define SFS_TYPE_FILE                           1
#define SFS_TYPE_DIR                            2
#define SFS_TYPE_LINK                           3

#define SFS_BLKN_SUPER                          0
#define SFS_BLKN_ROOT                           1
#define SFS_BLKN_FREEMAP                        2

struct cache_block {
    uint32_t ino;
    struct cache_block *hash_next;
    void *cache;
};

struct cache_inode {
    struct inode {
        uint32_t size;
        uint16_t type;
        uint16_t nlinks;
        uint32_t blocks;
        uint32_t direct[SFS_NDIRECT];
        uint32_t indirect;
        uint32_t db_indirect;
    } inode;
    ino_t real;
    uint32_t ino;
    uint32_t nblks;
    struct cache_block *l1, *l2;
    struct cache_inode *hash_next;
};

struct sfs_fs {
    struct {
        uint32_t magic;
        uint32_t blocks;
        uint32_t unused_blocks;
        char info[SFS_MAX_INFO_LEN + 1];
    } super;
    struct subpath {
        struct subpath *next, *prev;
        char *subname;
    } __sp_nil, *sp_root, *sp_end;
    int imgfd;
    uint32_t ninos, next_ino;
    struct cache_inode *root;
    struct cache_inode *inodes[HASH_LIST_SIZE];
    struct cache_block *blocks[HASH_LIST_SIZE];
};

struct sfs_entry {
    uint32_t ino;
    char name[SFS_MAX_FNAME_LEN + 1];
};
```

```
static uint32_t
sfs_alloc_ino(struct sfs_fs *sfs) {
    if (sfs->next_ino < sfs->ninos) {
        sfs->super.unused_blocks --;
        return sfs->next_ino ++;
    }
    bug("out of disk space.\n");
}

......

#define show_fullpath(sfs, name) subpath_show(stderr, sfs, name)

void open_dir(struct sfs_fs *sfs, struct cache_inode *current, struct cache_inode *parent);
void open_file(struct sfs_fs *sfs, struct cache_inode *file, const char *filename, int fd);
void open_link(struct sfs_fs *sfs, struct cache_inode *file, const char *filename);

#define SFS_BLK_NENTRY                          (SFS_BLKSIZE / sizeof(uint32_t))
#define SFS_L0_NBLKS                            SFS_NDIRECT
#define SFS_L1_NBLKS                            (SFS_BLK_NENTRY + SFS_L0_NBLKS)
#define SFS_L2_NBLKS                            (SFS_BLK_NENTRY * SFS_BLK_NENTRY + SFS_L1_NBLKS)
#define SFS_LN_NBLKS                            (SFS_MAX_FILE_SIZE / SFS_BLKSIZE)

static void
update_cache(struct sfs_fs *sfs, struct cache_block **cbp, uint32_t *inop) {
    uint32_t ino = *inop;
    struct cache_block *cb = *cbp;
    if (ino == 0) {
        cb = alloc_cache_block(sfs, 0);
        ino = cb->ino;
    }
    else if (cb == NULL || cb->ino != ino) {
        cb = search_cache_block(sfs, ino);
        assert(cb != NULL && cb->ino == ino);
    }
    *cbp = cb, *inop = ino;
}

static void
append_block(struct sfs_fs *sfs, struct cache_inode *file, size_t size, uint32_t ino, const char *
filename) {
    static_assert(SFS_LN_NBLKS <= SFS_L2_NBLKS);
    assert(size <= SFS_BLKSIZE);
    uint32_t nblks = file->nblks;
    struct inode *inode = &(file->inode);
    if (nblks >= SFS_LN_NBLKS) {
        open_bug(sfs, filename, "file is too big.\n");
    }
    if (nblks < SFS_L0_NBLKS) {    /* ……(4)…… */
        inode->direct[nblks] = ino;     /* ……(5)…… */
    }
    else if (nblks < SFS_L1_NBLKS) {    /* ……(6)…… */
        nblks -= SFS_L0_NBLKS;     /* ……(7)…… */
        update_cache(sfs, &(file->l1), &(inode->indirect));
        uint32_t *data = file->l1->cache;
        data[nblks] = ino;         /* ……(8)…… */
    }
    else if (nblks < SFS_L2_NBLKS) {    /* ……(9)…… */
        nblks -= SFS_L1_NBLKS;     /* ……(10)…… */
        update_cache(sfs, &(file->l2), &(inode->db_indirect));
        uint32_t *data2 = file->l2->cache;
        update_cache(sfs, &(file->l1), &data2[nblks / SFS_BLK_NENTRY]);
        uint32_t *data1 = file->l1->cache;
        data1[nblks % SFS_BLK_NENTRY] = ino;     /* ……(11)…… */
```

```
    }
    file->nblks ++;
    inode->size += size;
    inode->blocks ++; /* ……(12)…… */
}
......
```

==========

七、(6 分)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 和 F3 的引用计数值分别是多少？要求说明理由。

八、(11 分)I/O 子系统是操作系统中负责计算机系统与外界进行信息交互功能。键盘和显示器是计算机系统中最基本的 I/O 设备。

1）试描述 ucore 内核中是如何实现命令行状态的键盘输入时屏幕回显的；

2）试解释下面与 I/O 子系统中指定代码行的作用。注意：需要解释的代码共有 10 处。

========== kern-ucore/arch/i386/driver/console.c==========

```
#include <types.h>
#include <arch.h>
#include <stdio.h>
#include <string.h>
#include <kbdreg.h>
#include <picirq.h>
#include <trap.h>
#include <memlayout.h>
#include <sync.h>
#include <kio.h>

/* stupid I/O delay routine necessitated by historical PC design flaws */
static void
delay(void) {
    inb(0x84);
    inb(0x84);
    inb(0x84);
    inb(0x84);
}
......

static uint16_t *crt_buf;
static uint16_t crt_pos;
static uint16_t addr_6845;

/* TEXT-mode CGA/VGA display output */

static void
cga_init(void) {
    volatile uint16_t *cp = (uint16_t *)(CGA_BUF + KERNBASE);
    uint16_t was = *cp;
    *cp = (uint16_t) 0xA55A;
    if (*cp != 0xA55A) {
        cp = (uint16_t*)(MONO_BUF + KERNBASE);
        addr_6845 = MONO_BASE;
    } else {
        cp = was;
        addr_6845 = CGA_BASE;
    }

    // Extract cursor location
    uint32_t pos;
    outb(addr_6845, 14);
    pos = inb(addr_6845 + 1) << 8; /* ……(1)…… */
    outb(addr_6845, 15);
```

```
    pos |= inb(addr_6845 + 1); /* ……(2)…… */
    crt_buf = (uint16_t*) cp;  /* ……(3)…… */
    crt_pos = pos;
}

static bool serial_exists = 0;

static void
serial_init(void) {
......
}
......

/* cga_putc - print character to console */
static void
cga_putc(int c) {
    // set black on white
    if (!(c & ~0xFF)) {
        c |= 0x0700;
    }

    switch (c & 0xff) {
    case '\b':
        if (crt_pos > 0) {
            crt_pos --;
            crt_buf[crt_pos] = (c & ~0xff) | ' ';
        }
        break;
    case '\n':
        crt_pos += CRT_COLS;
    case '\r':
        crt_pos -= (crt_pos % CRT_COLS);
        break;
    default:
        crt_buf[crt_pos ++] = c;     // write the character
        break;
    }

    // What is the purpose of this?
    if (crt_pos >= CRT_SIZE) {
        int i;
        memmove(crt_buf, crt_buf + CRT_COLS, (CRT_SIZE - CRT_COLS) * sizeof(uint16_t));
        for (i = CRT_SIZE - CRT_COLS; i < CRT_SIZE; i ++) {
            crt_buf[i] = 0x0700 | ' ';
        }
        crt_pos -= CRT_COLS;
    }

    // move that little blinky thing
    outb(addr_6845, 14);
    outb(addr_6845 + 1, crt_pos >> 8);
    outb(addr_6845, 15);
    outb(addr_6845 + 1, crt_pos);
}
......
/* *
 * Here we manage the console input buffer, where we stash characters
 * received from the keyboard or serial port whenever the corresponding
 * interrupt occurs.
 * */

#define CONSBUFSIZE 512

static struct {
    uint8_t buf[CONSBUFSIZE];
```

```
    uint32_t rpos;
    uint32_t wpos;
} cons;

/* *
 * cons_intr - called by device interrupt routines to feed input
 * characters into the circular console input buffer.
 * */
static void
cons_intr(int (*proc)(void)) {
    int c;
    while ((c = (*proc)()) != -1) {
        if (c != 0) {
            cons.buf[cons.wpos ++] = c; /* ……(4)…… */
            if (cons.wpos == CONSBUFSIZE) {
                cons.wpos = 0;      /* ……(5)…… */
            }
        }
    }
}

/* serial_proc_data - get data from serial port */
static int
serial_proc_data(void) {
    if (!(inb(COM1 + COM_LSR) & COM_LSR_DATA)) {
        return -1;
    }
    int c = inb(COM1 + COM_RX);
    if (c == 127) {
        c = '\b';
    }
    return c;
}

/* serial_intr - try to feed input characters from serial port */
void
serial_intr(void) {
    if (serial_exists) {
        cons_intr(serial_proc_data);
    }
}

/***** Keyboard input code *****/

#define NO              0

#define SHIFT           (1<<0)
#define CTL             (1<<1)
#define ALT             (1<<2)

#define CAPSLOCK        (1<<3)
#define NUMLOCK         (1<<4)
#define SCROLLLOCK      (1<<5)

#define E0ESC           (1<<6)

static uint8_t shiftcode[256] = {
    [0x1D] CTL,
    [0x2A] SHIFT,
    [0x36] SHIFT,
    [0x38] ALT,
    [0x9D] CTL,
    [0xB8] ALT
};
```

```
static uint8_t togglecode[256] = {
    [0x3A] CAPSLOCK,
    [0x45] NUMLOCK,
    [0x46] SCROLLLOCK
};

static uint8_t normalmap[256] = {
    NO,   0x1B, '1',  '2',  '3',  '4',  '5',  '6',  // 0x00
    '7',  '8',  '9',  '0',  '-',  '=',  '\b', '\t',
    'q',  'w',  'e',  'r',  't',  'y',  'u',  'i',  // 0x10
    'o',  'p',  '[',  ']',  '\n', NO,   'a',  's',
    'd',  'f',  'g',  'h',  'j',  'k',  'l',  ';',  // 0x20
    '\'', '`',  NO,   '\\', 'z',  'x',  'c',  'v',
    'b',  'n',  'm',  ',',  '.',  '/',  NO,   '*',  // 0x30
    NO,   ' ',  NO,   NO,   NO,   NO,   NO,   NO,
    NO,   NO,   NO,   NO,   NO,   NO,   NO,   '7',  // 0x40
    '8',  '9',  '-',  '4',  '5',  '6',  '+',  '1',
    '2',  '3',  '0',  '.',  NO,   NO,   NO,   NO,   // 0x50
    [0xC7] KEY_HOME,    [0x9C] '\n' /*KP_Enter*/,
    [0xB5] '/' /*KP_Div*/,  [0xC8] KEY_UP,
    [0xC9] KEY_PGUP,    [0xCB] KEY_LF,
    [0xCD] KEY_RT,      [0xCF] KEY_END,
    [0xD0] KEY_DN,      [0xD1] KEY_PGDN,
    [0xD2] KEY_INS,     [0xD3] KEY_DEL
};

static uint8_t shiftmap[256] = {
    NO,   033,  '!',  '@',  '#',  '$',  '%',  '^',  // 0x00
    '&',  '*',  '(',  ')',  '_',  '+',  '\b', '\t',
    'Q',  'W',  'E',  'R',  'T',  'Y',  'U',  'I',  // 0x10
    'O',  'P',  '{',  '}',  '\n', NO,   'A',  'S',
    'D',  'F',  'G',  'H',  'J',  'K',  'L',  ':',  // 0x20
    '"',  '~',  NO,   '|',  'Z',  'X',  'C',  'V',
    'B',  'N',  'M',  '<',  '>',  '?',  NO,   '*',  // 0x30
    NO,   ' ',  NO,   NO,   NO,   NO,   NO,   NO,
    NO,   NO,   NO,   NO,   NO,   NO,   NO,   '7',  // 0x40
    '8',  '9',  '-',  '4',  '5',  '6',  '+',  '1',
    '2',  '3',  '0',  '.',  NO,   NO,   NO,   NO,   // 0x50
    [0xC7] KEY_HOME,    [0x9C] '\n' /*KP_Enter*/,
    [0xB5] '/' /*KP_Div*/,  [0xC8] KEY_UP,
    [0xC9] KEY_PGUP,    [0xCB] KEY_LF,
    [0xCD] KEY_RT,      [0xCF] KEY_END,
    [0xD0] KEY_DN,      [0xD1] KEY_PGDN,
    [0xD2] KEY_INS,     [0xD3] KEY_DEL
};

#define C(x) (x - '@')

static uint8_t ctlmap[256] = {
    NO,      NO,      NO,      NO,      NO,      NO,      NO,      NO,
    NO,      NO,      NO,      NO,      NO,      NO,      NO,      NO,
    C('Q'),  C('W'),  C('E'),  C('R'),  C('T'),  C('Y'),  C('U'),  C('I'),
    C('O'),  C('P'),  NO,      NO,      '\r',    NO,      C('A'),  C('S'),
    C('D'),  C('F'),  C('G'),  C('H'),  C('J'),  C('K'),  C('L'),  NO,
    NO,      NO,      NO,      C('\\'), C('Z'),  C('X'),  C('C'),  C('V'),
    C('B'),  C('N'),  C('M'),  NO,      NO,      C('/'),  NO,      NO,
    [0x97] KEY_HOME,
    [0xB5] C('/'),      [0xC8] KEY_UP,
    [0xC9] KEY_PGUP,    [0xCB] KEY_LF,
    [0xCD] KEY_RT,      [0xCF] KEY_END,
    [0xD0] KEY_DN,      [0xD1] KEY_PGDN,
    [0xD2] KEY_INS,     [0xD3] KEY_DEL
};

static uint8_t *charcode[4] = {
```

```
    normalmap,
    shiftmap,
    ctlmap,
    ctlmap
};

/* *
 * kbd_proc_data - get data from keyboard
 *
 * The kbd_proc_data() function gets data from the keyboard.
 * If we finish a character, return it, else 0. And return -1 if no data.
 * */
static int
kbd_proc_data(void) {
    int c;
    uint8_t data;
    static uint32_t shift;

    if ((inb(KBSTATP) & KBS_DIB) == 0) {
        return -1;
    }

    data = inb(KBDATAP);

    if (data == 0xE0) {
        // E0 escape character
        shift |= E0ESC;
        return 0;
    } else if (data & 0x80) {
        // Key released
        data = (shift & E0ESC ? data : data & 0x7F);
        shift &= ~(shiftcode[data] | E0ESC);
        return 0;
    } else if (shift & E0ESC) {
        // Last character was an E0 escape; or with 0x80
        data |= 0x80;
        shift &= ~E0ESC;
    }

    shift |= shiftcode[data]; /* ……(6)…… */
    shift ^= togglecode[data];

    c = charcode[shift & (CTL | SHIFT)][data];
    if (shift & CAPSLOCK) {
        if ('a' <= c && c <= 'z')
            c += 'A' - 'a';    /* ……(7)…… */
        else if ('A' <= c && c <= 'Z')
            c += 'a' - 'A';
    }

    // Process special keys
    // Ctrl-Alt-Del: reboot
    if (!(~shift & (CTL | ALT)) && c == KEY_DEL) {
        kprintf("Rebooting!\n");
        outb(0x92, 0x3); // courtesy of Chris Frost
    }
    return c;
}

/* kbd_intr - try to feed input characters from keyboard */
void
kbd_intr(void) {
    cons_intr(kbd_proc_data);  /* ……(8)…… */
}
```

```
static void
kbd_init(void) {
    // drain the kbd buffer
    kbd_intr();
    pic_enable(IRQ_KBD);
}

/* cons_init - initializes the console devices */
void
cons_init(void) {
    cga_init();
    serial_init();
    kbd_init();
    if (!serial_exists) {
        kprintf("serial port does not exist!!\n");
    }
}

/* cons_putc - print a single character @c to console devices */
void
cons_putc(int c) {
    bool intr_flag;
    local_intr_save(intr_flag);
    {
        lpt_putc(c);
        cga_putc(c);
        serial_putc(c);
    }
    local_intr_restore(intr_flag);
}

/* *
 * cons_getc - return the next input character from console,
 * or 0 if none waiting.
 * */
int
cons_getc(void) {
    int c = 0;
    bool intr_flag;
    local_intr_save(intr_flag);
    {
        // poll for any pending input characters,
        // so that this function works even when interrupts are disabled
        // (e.g., when called from the kernel monitor).
        serial_intr();
        kbd_intr();

        // grab the next character from the input buffer.
        if (cons.rpos != cons.wpos) {
            c = cons.buf[cons.rpos ++]; /* ……(9)…… */
            if (cons.rpos == CONSBUFSIZE) {
                cons.rpos = 0; /* ……(10)…… */
            }
        }
    }
    local_intr_restore(intr_flag);
    return c;
}

============================================================
```