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04月10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7个题目，共16页。
5. 考试完毕，必须将试题纸和答卷本一起交回。

一、（12分）1）试说明硬中断（hardware interrupt）、异常（exception）和系统调用（system call）的相同点和不同点。

2）下面代码完成在进入trap()函数前的准备工作。其中pushal完成包括esp在内的CPU寄存器压栈。试说明“pushl %esp”的作用是什么？

```
=============trapentry.S (kern\trap)=============
#include <memlayout.h>

# vectors.S sends all traps here.
.text
.globl __alltraps
__alltraps:
   # push registers to build a trap frame
   # therefore make the stack look like a struct trapframe
   pushl %ds
   pushl %es
   pushl %fs
   pushl %gs
   pushal

   # load GD_KDATA into %ds and %es to set up data segments for kernel
   movl $GD_KDATA, %eax
   movw %ax, %ds
   movw %ax, %es
   pushl %esp
   call trap
   # pop the pushed stack pointer
   popl %esp

   # return falls through to trapret...
.globl __trapret
```

```
__trapret:
    # restore registers from stack
    popal

    # restore %ds, %es, %fs and %gs
    popl %gs
    popl %fs
    popl %es
    popl %ds

    # get rid of the trap number and error code
    addl $0x8, %esp
    iret
=============Trap.c (kern\trap)=============
......
/* *
 * trap - handles or dispatches an exception/interrupt. if and when trap() returns,
 * the code in kern/trap/trapentry.S restores the old CPU state saved in the
 * trapframe and then uses the iret instruction to return from the exception.
 * */
void
trap(struct trapframe *tf) {
    // dispatch based on what type of trap occurred
    trap_dispatch(tf);
}
......
```

二、（13 分）1）系统调用的参数传递有几种方式？各有什么特点？

2）sys_exec 是一个加载和执行指定可执行文件的系统调用。请说明在下面的 ucore 实现中，它的三个参数分别是以什么方式传递的。

```
=============Proc.c (kern\process)=============
......
// do_execve - call exit_mmap(mm)&pug_pgdir(mm) to reclaim memory space of current process
//           - call load_icode to setup new memory space accroding binary prog.
int
do_execve(const char *name, int argc, const char **argv) {
    static_assert(EXEC_MAX_ARG_LEN >= FS_MAX_FPATH_LEN);
    struct mm_struct *mm = current->mm;
    if (!(argc >= 1 && argc <= EXEC_MAX_ARG_NUM)) {
        return -E_INVAL;
    }

    char local_name[PROC_NAME_LEN + 1];
    memset(local_name, 0, sizeof(local_name));
```

```c
    char *kargv[EXEC_MAX_ARG_NUM];
    const char *path;

    int ret = -E_INVAL;

    lock_mm(mm);
    if (name == NULL) {
        snprintf(local_name, sizeof(local_name), "<null> %d", current->pid);
    }
    else {
        if (!copy_string(mm, local_name, name, sizeof(local_name))) {
            unlock_mm(mm);
            return ret;
        }
    }
    if ((ret = copy_kargv(mm, argc, kargv, argv)) != 0) {
        unlock_mm(mm);
        return ret;
    }
    path = argv[0];
    unlock_mm(mm);
    files_closeall(current->filesp);

    /* sysfile_open will check the first argument path, thus we have to use a user-space pointer, and
argv[0] may be incorrect */
    int fd;
    if ((ret = fd = sysfile_open(path, O_RDONLY)) < 0) {
        goto execve_exit;
    }
    if (mm != NULL) {
        lcr3(boot_cr3);
        if (mm_count_dec(mm) == 0) {
            exit_mmap(mm);
            put_pgdir(mm);
            mm_destroy(mm);
        }
        current->mm = NULL;
    }
    ret= -E_NO_MEM;;
    if ((ret = load_icode(fd, argc, kargv)) != 0) {
        goto execve_exit;
    }
    put_kargv(argc, kargv);
    set_proc_name(current, local_name);
    return 0;
```

```
execve_exit:
    put_kargv(argc, kargv);
    do_exit(ret);
    panic("already exit: %e.\n", ret);
}
......
=============Syscall.c (kern\syscall)=============
......
static int
sys_exec(uint32_t arg[]) {
    const char *name = (const char *)arg[0];
    int argc = (int)arg[1];
    const char **argv = (const char **)arg[2];
    return do_execve(name, argc, argv);
}
......
static int (*syscalls[])(uint32_t arg[]) = {
    [SYS_exit]              sys_exit,
    [SYS_fork]              sys_fork,
    [SYS_wait]              sys_wait,
    [SYS_exec]              sys_exec,
    [SYS_yield]             sys_yield,
    [SYS_kill]              sys_kill,
    [SYS_getpid]            sys_getpid,
    [SYS_putc]              sys_putc,
    [SYS_pgdir]             sys_pgdir,
};

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

void
syscall(void) {
    struct trapframe *tf = 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, current->pid, current->name);
}
......
=============libs-user-ucore/syscall.c=============
......

int sys_exec(const char *filename, const char **argv, const char **envp)
{
    return syscall(SYS_exec, filename, argv, envp);
}

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

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;
}
```

三、（15 分）1）描述伙伴系统（Buddy System）中对物理内存的分配和回收过程。2）假定一个操作系统内核中由伙伴系统管理的物理内存有 1MB，试描述按下面顺序进行物理内存分配和回收过程中，每次分配完成后的分配区域的首地址和大小，或每次回收完成后的空闲区域队列（要求说明，每个空闲块的首地址和大小）。建议给出分配和回收的中间过程。

a) 进程 A 申请50KB；

b) 进程 B 申请100KB；

c) 进程 C 申请40KB；

d) 进程 D 申请70KB；

e) 进程 B 释放100KB;

f) 进程 E 申请127KB;

g) 进程 D 释放70KB;

h) 进程 A 释放50KB;

i) 进程 E 释放127KB;

j) 进程 C 释放40KB;

四、(10 分) 当一个进程释放一个包含某虚地址的物理内存页时，需要让对应此物理内存页的管理数据结构 Page 进行清除处理，使得此物理内存页成为空闲。同时，还需把表示虚地址与物理地址映射关系的二级页表项清除，这个工作由 page_remove_pte 函数完成。 page_remove_pte 函数的调用关系图如下所示。请补全在 kern/mm/pmm.c 中的 page_remove_pte 函数。

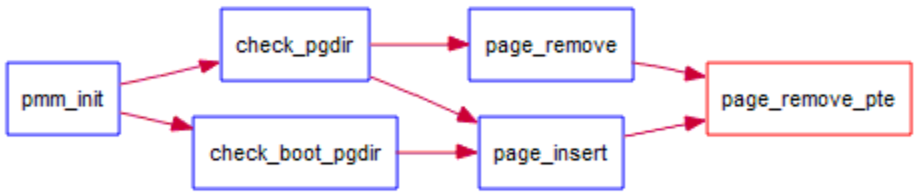

图 1 page_remove_pte 函数的调用关系图

```
=============Pmm.h (kern\mm)=============
#define alloc_page() alloc_pages(1)
#define free_page(page) free_pages(page, 1)
......
static inline struct Page *
pte2page(pte_t pte) {
    if (!(pte & PTE_P)) {
        panic("pte2page called with invalid pte");
    }
    return pa2page(PTE_ADDR(pte));
}
......
static inline int
page_ref_inc(struct Page *page) {
    page->ref += 1;
    return page->ref;
}
static inline int
page_ref_dec(struct Page *page) {
    page->ref -= 1;
    return page->ref;
}
......
```

```
=============Pmm.c (kern\mm)=============
......
//page_remove_pte - free an Page sturct which is related linear address la
//              - and clean(invalidate) pte which is related linear address la
//note: PT is changed, so the TLB need to be invalidate
static inline void
page_remove_pte(pde_t *pgdir, uintptr_t la, pte_t *ptep) {
    /* LAB2 EXERCISE 3: YOUR CODE
     *
     * Please check if ptep is valid, and tlb must be manually updated if mapping is updated
     *
     * Maybe you want help comment, BELOW comments can help you finish the code
     *
     * Some Useful MACROs and DEFINEs, you can use them in below implementation.
     * MACROs or Functions:
     *   struct Page *page pte2page(*ptep): get the according page from the value of a ptep
     *   free_page : free a page
     *   page_ref_dec(page) : decrease page->ref. NOTICE: ff page->ref == 0 , then this page should be
free.
     *   tlb_invalidate(pde_t *pgdir, uintptr_t la) : Invalidate a TLB entry, but only if the page tables
being
     *                    edited are the ones currently in use by the processor.
     * DEFINEs:
     *   PTE_P           0x001                   // page table/directory entry flags bit : Present
     */
#if 0
    if (0) {                      //(1) check if page directory is present
        struct Page *page = NULL; //(2) find corresponding page to pte
                                  //(3) decrease page reference
                                  //(4) and free this page when page reference reachs 0
                                  //(5) clear second page table entry
                                  //(6) flush tlb
    }
#endif
===Your code 1===
}
......
// invalidate a TLB entry, but only if the page tables being
// edited are the ones currently in use by the processor.
void
tlb_invalidate(pde_t *pgdir, uintptr_t la) {
    if (rcr3() == PADDR(pgdir)) {
        invlpg((void *)la);
    }
}
static void
```

```
check_alloc_page(void) {
    pmm_manager->check();
    cprintf("check_alloc_page() succeeded!\n");
}
```

=============Mmu.h (kern\mm)=============

```
/* page table/directory entry flags */
#define PTE_P           0x001                   // Present
#define PTE_W           0x002                   // Writeable
#define PTE_U           0x004                   // User
#define PTE_PWT         0x008                   // Write-Through
#define PTE_PCD         0x010                   // Cache-Disable
#define PTE_A           0x020                   // Accessed
#define PTE_D           0x040                   // Dirty
#define PTE_PS          0x080                   // Page Size
#define PTE_MBZ         0x180                   // Bits must be zero
#define PTE_AVAIL       0xE00                   // Available for software use
                                                // The PTE_AVAIL bits aren't used by the kernel or interpreted by the
                                                // hardware, so user processes are allowed to set them arbitrarily.

#define PTE_USER        (PTE_U | PTE_W | PTE_P)
```

五、（20 分）1）试用图示描述 32 位 X86 系统在采用 4KB 页面大小时的虚拟地址结构和地址置换过程。2）在采用 4KB 页面大小的 32 位 X86 的 ucore 虚拟存储系统中，进程页面的起始地址由宏 VPT 确定。

```
#define VPT                 0x0D000000
```

请计算：2a)试给出页目录中自映射页表项的虚拟地址；2b)虚拟地址 0X87654321 对应的页目录项和页表项的虚拟地址。

六、（15 分）试描述 FIFO 页面替换算法的基本原理，并 swap_fifo.c 中未完成 FIFA 页面替换算法实验函数 map_swappable()和 swap_out_vistim() 。

=============Defs.h (libs)=============

```
/* *
 * to_struct - get the struct from a ptr
 * @ptr:    a struct pointer of member
 * @type:   the type of the struct this is embedded in
 * @member: the name of the member within the struct
 * */
#define to_struct(ptr, type, member)                               \
    ((type *)((char *)(ptr) - offsetof(type, member)))
```

```
=============Memlayout.h (kern\mm)=============
// convert list entry to page
#define le2page(le, member)                 \
    to_struct((le), struct Page, member)

=============List.h (libs)=============
#ifndef __LIBS_LIST_H__
#define __LIBS_LIST_H__

#ifndef __ASSEMBLER__

#include <defs.h>

/* *
 * Simple doubly linked list implementation.
 *
 * Some of the internal functions ("__xxx") are useful when manipulating
 * whole lists rather than single entries, as sometimes we already know
 * the next/prev entries and we can generate better code by using them
 * directly rather than using the generic single-entry routines.
 * */

struct list_entry {
    struct list_entry *prev, *next;
};

typedef struct list_entry list_entry_t;

static inline void list_init(list_entry_t *elm) __attribute__((always_inline));
static inline void list_add(list_entry_t *listelm, list_entry_t *elm) __attribute__((always_inline));
static inline void list_add_before(list_entry_t *listelm, list_entry_t *elm)
__attribute__((always_inline));
static inline void list_add_after(list_entry_t *listelm, list_entry_t *elm)
__attribute__((always_inline));
static inline void list_del(list_entry_t *listelm) __attribute__((always_inline));
static inline void list_del_init(list_entry_t *listelm) __attribute__((always_inline));
static inline bool list_empty(list_entry_t *list) __attribute__((always_inline));
static inline list_entry_t *list_next(list_entry_t *listelm) __attribute__((always_inline));
static inline list_entry_t *list_prev(list_entry_t *listelm) __attribute__((always_inline));

static inline void __list_add(list_entry_t *elm, list_entry_t *prev, list_entry_t *next)
__attribute__((always_inline));
static inline void __list_del(list_entry_t *prev, list_entry_t *next) __attribute__((always_inline));

/* *
 * list_init - initialize a new entry
```

```
 * @elm:       new entry to be initialized
 * */
static inline void
list_init(list_entry_t *elm) {
    elm->prev = elm->next = elm;
}

/* *
 * list_add - add a new entry
 * @listelm:    list head to add after
 * @elm:        new entry to be added
 *
 * Insert the new element @elm *after* the element @listelm which
 * is already in the list.
 * */
static inline void
list_add(list_entry_t *listelm, list_entry_t *elm) {
    list_add_after(listelm, elm);
}

/* *
 * list_add_before - add a new entry
 * @listelm:    list head to add before
 * @elm:        new entry to be added
 *
 * Insert the new element @elm *before* the element @listelm which
 * is already in the list.
 * */
static inline void
list_add_before(list_entry_t *listelm, list_entry_t *elm) {
    __list_add(elm, listelm->prev, listelm);
}

/* *
 * list_add_after - add a new entry
 * @listelm:    list head to add after
 * @elm:        new entry to be added
 *
 * Insert the new element @elm *after* the element @listelm which
 * is already in the list.
 * */
static inline void
list_add_after(list_entry_t *listelm, list_entry_t *elm) {
    __list_add(elm, listelm, listelm->next);
}
```

```
/* *
 * list_del - deletes entry from list
 * @listelm:    the element to delete from the list
 *
 * Note: list_empty() on @listelm does not return true after this, the entry is
 * in an undefined state.
 * */
static inline void
list_del(list_entry_t *listelm) {
   __list_del(listelm->prev, listelm->next);
}

/* *
 * list_del_init - deletes entry from list and reinitialize it.
 * @listelm:    the element to delete from the list.
 *
 * Note: list_empty() on @listelm returns true after this.
 * */
static inline void
list_del_init(list_entry_t *listelm) {
   list_del(listelm);
   list_init(listelm);
}

/* *
 * list_empty - tests whether a list is empty
 * @list:      the list to test.
 * */
static inline bool
list_empty(list_entry_t *list) {
   return list->next == list;
}

/* *
 * list_next - get the next entry
 * @listelm:    the list head
 **/
static inline list_entry_t *
list_next(list_entry_t *listelm) {
   return listelm->next;
}

/* *
 * list_prev - get the previous entry
 * @listelm:    the list head
 **/
```

```
static inline list_entry_t *
list_prev(list_entry_t *listelm) {
    return listelm->prev;
}

/* *
 * Insert a new entry between two known consecutive entries.
 *
 * This is only for internal list manipulation where we know
 * the prev/next entries already!
 * */
static inline void
__list_add(list_entry_t *elm, list_entry_t *prev, list_entry_t *next) {
    prev->next = next->prev = elm;
    elm->next = next;
    elm->prev = prev;
}

/* *
 * Delete a list entry by making the prev/next entries point to each other.
 *
 * This is only for internal list manipulation where we know
 * the prev/next entries already!
 * */
static inline void
__list_del(list_entry_t *prev, list_entry_t *next) {
    prev->next = next;
    next->prev = prev;
}

#endif /* !__ASSEMBLER__ */

#endif /* !__LIBS_LIST_H__ */


============= Swap_fifo.c (kern\mm)=============

#include <defs.h>
#include <x86.h>
#include <stdio.h>
#include <string.h>
#include <swap.h>
#include <swap_fifo.h>
#include <list.h>

/* [wikipedia]The simplest Page Replacement Algorithm(PRA) is a FIFO algorithm.
```

```
* (1) Prepare: In order to implement FIFO PRA, we should manage all swappable pages, so we can
 *            link these pages into pra_list_head according the time order. At first you should
 *            be familiar to the struct list in list.h. struct list is a simple doubly linked list
 *            implementation. You should know howto USE: list_init, list_add(list_add_after),
 *            list_add_before, list_del, list_next, list_prev. Another tricky method is to transform
 *            a general list struct to a special struct (such as struct page). You can find some MACRO:
 *            le2page (in memlayout.h), (in future labs: le2vma (in vmm.h), le2proc (in proc.h),etc.
 */

list_entry_t pra_list_head;
/*
 * (2) _fifo_init_mm: init pra_list_head and let  mm->sm_priv point to the addr of pra_list_head.
 *            Now, From the memory control struct mm_struct, we can access FIFO PRA
 */
static int
_fifo_init_mm(struct mm_struct *mm)
{
    list_init(&pra_list_head);
    mm->sm_priv = &pra_list_head;
    //cprintf(" mm->sm_priv %x in fifo_init_mm\n",mm->sm_priv);
    return 0;
}
/*
 * (3)_fifo_map_swappable: According FIFO PRA, we should link the most recent arrival page at the back
of pra_list_head qeueue
 */
static int
_fifo_map_swappable(struct mm_struct *mm, uintptr_t addr, struct Page *page, int swap_in)
{
   list_entry_t *head=(list_entry_t*) mm->sm_priv;
   list_entry_t *entry=&(page->pra_page_link);

   assert(entry != NULL && head != NULL);
   //record the page access situlation
   /*LAB3 EXERCISE 2: YOUR CODE*/
   //(1)link the most recent arrival page at the back of the pra_list_head qeueue.
   ===Your code 2===
   return 0;
}
/*
 *  (4)_fifo_swap_out_victim: According FIFO PRA, we should unlink the  earliest arrival page in front
of pra_list_head qeueue,
 *                        then set the addr of addr of this page to ptr_page.
 */
static int
_fifo_swap_out_victim(struct mm_struct *mm, struct Page ** ptr_page, int in_tick)
```

```
{
    list_entry_t *head=(list_entry_t*) mm->sm_priv;
       assert(head != NULL);
    assert(in_tick==0);
    /* Select the victim */
    /*LAB3 EXERCISE 2: YOUR CODE*/
    //(1)  unlink the  earliest arrival page in front of pra_list_head qeueue
    //(2)  set the addr of addr of this page to ptr_page
    /* Select the tail */
   ===Your code 3===
    return 0;
}

static int
_fifo_check_swap(void) {
   cprintf("write Virt Page c in fifo_check_swap\n");
   *(unsigned char *)0x3000 = 0x0c;
   assert(pgfault_num==4);
   cprintf("write Virt Page a in fifo_check_swap\n");
   *(unsigned char *)0x1000 = 0x0a;
   assert(pgfault_num==4);
   cprintf("write Virt Page d in fifo_check_swap\n");
   *(unsigned char *)0x4000 = 0x0d;
   assert(pgfault_num==4);
   cprintf("write Virt Page b in fifo_check_swap\n");
   *(unsigned char *)0x2000 = 0x0b;
   assert(pgfault_num==4);
   cprintf("write Virt Page e in fifo_check_swap\n");
   *(unsigned char *)0x5000 = 0x0e;
   assert(pgfault_num==5);
   cprintf("write Virt Page b in fifo_check_swap\n");
   *(unsigned char *)0x2000 = 0x0b;
   assert(pgfault_num==5);
   cprintf("write Virt Page a in fifo_check_swap\n");
   *(unsigned char *)0x1000 = 0x0a;
   assert(pgfault_num==6);
   cprintf("write Virt Page b in fifo_check_swap\n");
   *(unsigned char *)0x2000 = 0x0b;
   assert(pgfault_num==7);
   cprintf("write Virt Page c in fifo_check_swap\n");
   *(unsigned char *)0x3000 = 0x0c;
   assert(pgfault_num==8);
   cprintf("write Virt Page d in fifo_check_swap\n");
   *(unsigned char *)0x4000 = 0x0d;
   assert(pgfault_num==9);
   return 0;
```

```
}


static int
_fifo_init(void)
{
   return 0;
}

static int
_fifo_set_unswappable(struct mm_struct *mm, uintptr_t addr)
{
   return 0;
}

static int
_fifo_tick_event(struct mm_struct *mm)
{ return 0; }


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,
};
```

七、(15 分) 描述 int fork(void)系统调用的功能的接口过程，给出程序 fork.c 的输出结果，并用图示给出所有进程的父子关系。注：1）getpid()和 getppid()是两个系统调用，分别返回本进程标识和父进程标识。2）你可以假定每次新进程创建时生成的进程标识是顺序加 1 得到的；在进程标识为 1000 的命令解释程序 shell 中启动该程序的执行。

```
#include <sys/types.h>
#include <unistd.h>

/* getpid() and fork() are system calls declared in unistd.h.  They return */
/* values of type pid_t.  This pid_t is a special type for process ids. */
/* It's equivalent to int. */

int main(void)
{
```

```
    pid_t childpid;

    int x = 5;
      int i;
    childpid = fork();
    for ( i = 0;  i < 3;  i++)  {
        printf("This is process %d; childpid = %d; The parent of this process has id %d; i = %d; x = %d\n", getpid(), childpid, getppid(), i, x);
            sleep(1);
        x++;
    }

    return 0;
}
```