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：2010年04月21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12个题目，共9页。

5. 考试完毕，必须将试题纸和答卷本一起交回。

一、 (8分)试说明系统调用与函数调用的区别和联系；并说明在xv6中当用户进程通过系统调用从用户态进入内核态后，用户态的执行状态信息保存在哪个数据结构中。

二、 (10分)试描述工作集置换算法(Working Set Page Replacement)的工作原理，并针对存储访问序列”e, d, a, c, c, d, b, c, e, c, e, a, d”给出窗口大小为3和4时的缺页情况和工作集变化情况。

三、 (8分) 请说明xv6中的进程状态和状态含义，以及说明哪些状态会发生转换以及转换的原因。

四、 (3分) 在xv6中，进程调用exit系统调用后，内核会释放进程的代码段数据段和用户栈。为什么不在这个时候也释放页表和内核栈呢？

五、 (3分)在xv6中，内核是如何设置系统调用的返回值？

六、 (3分)当创建进程时，xv6是按什么顺序分配为进程分配资源的？

1) 分配用户栈
2) 分配进程结构(struct proc)
3) 分配内核栈
4) 分配页表

七、 (3分)在xv6中，fork系统调用父进程和子进程的trapframe有什么区别？

八、 (10分)基于对下面代码的分析，试说明应用程序通过系统调用使用操作系统服务时是如何传递系统调用的参数的。

usys.S

```
#include "syscall.h"
#include "traps.h"

#define STUB(name) \
  .globl name; \
  name: \
    movl $SYS_ ## name, %eax; \
    int $T_SYSCALL; \
    ret
```

```
……
STUB(read)
……
--------------------
```

syscall.c

```
--------------------
// User code makes a system call with INT T_SYSCALL.
// System call number in %eax.
// Arguments on the stack, from the user call to the C
// library system call function. The saved user %esp points
// to a saved program counter, and then the first argument.

// Fetch the int at addr from process p.
int
fetchint(struct proc *p, uint addr, int *ip)
{
  if(addr >= p->sz || addr+4 > p->sz)
    return -1;
  *ip = *(int*)(p->mem + addr);
  return 0;
}
……
// Fetch the nth 32-bit system call argument.
int
argint(int n, int *ip)
{
  return fetchint(cp, cp->tf->esp + 4 + 4*n, ip);
}

// Fetch the nth word-sized system call argument as a pointer
// to a block of memory of size n bytes.  Check that the pointer
// lies within the process address space.
int
argptr(int n, char **pp, int size)
{
  int i;

  if(argint(n, &i) < 0)
    return -1;
  if((uint)i >= cp->sz || (uint)i+size >= cp->sz)
    return -1;
  *pp = cp->mem + i;
  return 0;
}
……
extern int sys_read(void);
……

static int (*syscalls[])(void) = {
```

```
……
[SYS_read]    sys_read,
……
};
```

----

sysfile.c

```
int
sys_read(void)
{
  struct file *f;
  int n;
  char *p;

  if(argfd(0, 0, &f) < 0 || argint(2, &n) < 0 || argptr(1, &p, n) < 0)
    return -1;
  return fileread(f, p, n);
}
```

九、 (10分)基于对下面代码的分析，试说明一个被系统认为是符合规范的硬盘主引导扇区的特征是什么？

sign.c

```
/* simple boot sector builder*/
/*          chyyuu               */
/*        2010.02.28             */
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <unistd.h>
int main(int argc, char* argv[])
{
  struct stat sb;
  FILE *ifp, *ofp;
  char buf[512];
  int i,size;

  if (argc != 3) {
      fprintf(stderr, "Usage: <input filename> <output filename>\n");
      exit(-1);
  }
  if (stat(argv[1], &sb) == -1) {
         perror("stat");
         exit(-1);
  }

  if(sb.st_size>510) {
      printf("%s size: %lld bytes >510 !\n", argv[1], (long long) sb.st_size);
      exit(-1);
  }  else
      printf("%s size: %lld bytes\n", argv[1], (long long) sb.st_size);

  for(i=0;i<512;i++) buf[i]=0;
  buf[510]=0x55; buf[511]=0xAA;

  ifp=fopen(argv[1],"r");
  size=fread(buf,1,sb.st_size,ifp);
  if(size!=sb.st_size){
     printf("read %s error, size is %d\n",argv[1],size);
     exit(-1);
  }
  fclose(ifp);
  ofp=fopen(argv[2],"w");
  size=fwrite(buf,1,512,ofp);
  if(size!=512){
     printf("write %s error,size is %d\n",argv[2],size);
     exit(-1);
  }
  fclose(ofp);
  printf("build 512 bytes boot sector: %s success!\n",argv[2]);
  exit(0);
 }
```

十、 (12分)试在下面代码中补充完成计算虚拟地址的页目录序号的宏 PDX(la)、 虚拟地址的页表序号的宏 PTX(la)、虚拟地址的页内偏移的宏 PGOFF(la) 和逻辑页号的宏 PPN(la)。

mmu.h

```
// A linear address 'la' has a three-part structure as follows:
//
// +--------10------+-------10-------+---------12----------+
// | Page Directory |   Page Table   | Offset within Page  |
// |      Index     |      Index     |                     |
// +----------------+----------------+---------------------+
//  \--- PDX(la) --/ \--- PTX(la) --/ \---- PGOFF(la) ----/
//  \----------- PPN(la) -----------/
//
// The PDX, PTX, PGOFF, and PPN macros decompose linear addresses as shown.
// To construct a linear address la from PDX(la), PTX(la), and PGOFF(la),
// use PGADDR(PDX(la), PTX(la), PGOFF(la)).

// page directory index
#define PDX(la) ...(1)...

// page number field of address
#define PPN(la) ...(2)...
```

```
// page table index
#define PTX(la) ...(3)...

// offset in page
#define PGOFF(la) ...(4)...

// address in the page table
#define PTE_ADDR(pte) ((paddr_t)(pte) & ~0xFFF)

// construct linear address from indexes and offset
#define PGADDR(d, t, o) ((void*) ((d) << PDXSHIFT | (t) << PTXSHIFT | (o)))

// page directory and page table constants
#define PDXSHIFT   22      // offset of PDX in a linear address
#define PTXSHIFT   12      // offset of PTX in a linear address
#define PTENTRY    1024    // number of entries in page table
#define PDENTRY    1024    // number of entries in page table
#define PTSIZE     PAGE * PTENTRY// size of the whole page table
#define PGSIZE     PAGE
```

十一、　　(15分)试完成alloc_pages_bulk_buddy函数的实现，即补全该函数中用“...(x)...”处的代码。其他文件是在补全时可能用到的代码。

queue.h

```
/*
 * List functions.
 */

/*
 * Is the list named "head" empty?
 */
#define   LIST_EMPTY(head) ((head)->lh_first == NULL)

/*
 * Return the first element in the list named "head".
 */
#define   LIST_FIRST(head) ((head)->lh_first)

/*
 * Return the element after "elm" in the list.
 * The "field" name is the link element as above.
 */
#define   LIST_NEXT(elm, field)   ((elm)->field.le_next)

/*
 * Iterate over the elements in the list named "head".
 * During the loop, assign the list elements to the variable "var"
 * and use the LIST_ENTRY structure member "field" as the link field.
 */
#define   LIST_FOREACH(var, head, field)                 \
    for ((var) = LIST_FIRST((head));           \
        (var);                       \
        (var) = LIST_NEXT((var), field))

/*
 * Reset the list named "head" to the empty list.
 */
#define   LIST_INIT(head) do {                    \
    LIST_FIRST((head)) = NULL;              \
} while (0)

/*
 * Insert the element "elm" *after* the element "listelm" which is
 * already in the list.  The "field" name is the link element
 * as above.
 */
#define   LIST_INSERT_AFTER(listelm, elm, field) do {       \
    if ((LIST_NEXT((elm), field) = LIST_NEXT((listelm), field)) != NULL)\
       LIST_NEXT((listelm), field)->field.le_prev =      \
           &LIST_NEXT((elm), field);           \
    LIST_NEXT((listelm), field) = (elm);          \
    (elm)->field.le_prev = &LIST_NEXT((listelm), field);    \
} while (0)

/*
 * Insert the element "elm" *before* the element "listelm" which is
 * already in the list.  The "field" name is the link element
 * as above.
 */
#define   LIST_INSERT_BEFORE(listelm, elm, field) do {          \
    (elm)->field.le_prev = (listelm)->field.le_prev;     \
    LIST_NEXT((elm), field) = (listelm);          \
    *(listelm)->field.le_prev = (elm);            \
    (listelm)->field.le_prev = &LIST_NEXT((elm), field);    \
} while (0)

/*
 * Insert the element "elm" at the head of the list named "head".
 * The "field" name is the link element as above.
 */
#define   LIST_INSERT_HEAD(head, elm, field) do {           \
    if ((LIST_NEXT((elm), field) = LIST_FIRST((head))) != NULL) \
       LIST_FIRST((head))->field.le_prev = &LIST_NEXT((elm), field);\
    LIST_FIRST((head)) = (elm);                \
    (elm)->field.le_prev = &LIST_FIRST((head));          \
} while (0)

/*
```

```
 * Remove the element "elm" from the list.
 * The "field" name is the link element as above.
 */
#define   LIST_REMOVE(elm, field) do {                \
    if (LIST_NEXT((elm), field) != NULL)           \
       LIST_NEXT((elm), field)->field.le_prev =       \
           (elm)->field.le_prev;            \
    *(elm)->field.le_prev = LIST_NEXT((elm), field);     \
} while (0)
```

----

pmap.h

```
// flags describing the status of a page frame
#define PG_reserved  1  // the page frame is reserved for kernel code or is unusable
#define PG_property  2  // the property field of the page descriptor stores meaningful data
#define PG_locked    4  // the page is locked
#define PG_dirty     8  // the page has been modified
……
/* Physical pages descriptor, each Page describes a physical page*/
typedef LIST_HEAD(Page_list, Page) page_list_head_t;
typedef LIST_ENTRY(Page) page_list_entry_t;

struct Page {
    uint32_t flags;  // flags for page descriptors
    uint32_t mapcount;  // number of page table entries that refer to the page frame
    uint32_t property;  // when the page is free , this field is used by the buddy system
    uint32_t index;
    page_list_entry_t lru; /* free list link */
};

typedef struct Page page_t;
……
#define PageReserved(page) ((page)->flags & PG_reserved)
#define SetPageProperty(page) ((page)->flags |= PG_property)
```

----

buddy.h

```
#ifndef _BUDDY_H_
#define _BUDDY_H_
#include "pmap.h"
#include "phymem_manager.h"

extern const struct phymem_manager_class pmmc_buddy;

#define MAX_ORDER  11

typedef struct free_area {
    page_list_head_t free_list;
    unsigned long nr_free;
} free_area_t;

void init_memmap_buddy(struct Page *base, unsigned long nr);

extern struct Page *mem_map;
struct Page *alloc_pages_buddy(int nr);
void free_pages_buddy(struct Page *page, int nr);
struct Page *alloc_pages_bulk_buddy(int order);
void free_pages_bulk_buddy(struct Page *page, int order);

#endif
```

----

buddy.c

```
#include "buddy.h"
#include "defs.h"

free_area_t free_area[MAX_ORDER];
struct Page *mem_map;
const int FreeAreaSize[MAX_ORDER] = { 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024 };
……
// implement the buddy system strategy for freeing page frames
struct Page *alloc_pages_bulk_buddy(int order)
{
    int isalloc = 0;
    int current_order, size = 0;

    struct Page *page = NULL, *buddy = NULL;

    // try to find a suitable block in the buddy system
    for (current_order = order; current_order < MAX_ORDER; current_order++) {
       if (!LIST_EMPTY(&(free_area[current_order].free_list))) {
           isalloc = 1;
           break;
       }
    }

    if (!isalloc)
       return NULL;
    else {
       page = ...(1)...;
       LIST_REMOVE(page, lru);
       page->property = 0;
       ClearPageProperty(page);
       ...(2)...;
    }

    size = 1 << current_order;
    while (current_order > order) {
       ...(3)...;
       size >>= 1;
       buddy = page + size;
       ...(4)...;
       buddy->property = current_order;
       SetPageProperty(buddy);
       ...(5)...;
    }

    return page;
}
```

十二、 (15分)给出程序fork.c的输出结果，并用画示描述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，该程序执行时创建的第一个进程的标识为1000。

```
fork.c
----------------------
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/wait.h>
#define LOOP 3
int main()
{
    pid_t  pid;
    int i;

    for (i=0; i<LOOP; i++)
    {
       /* fork another process */
       pid = fork();
       if (pid < 0) { /* error occurred */
          fprintf(stderr, "Fork Failed");
          exit(-1);
       }
       else if (pid == 0) { /* child process */
          fprintf(stdout, "i=%d, pid=%d, parent pid=%d\n",i, getpid(),getppid());
       }
    }
    wait(NULL);
    exit(0);
}
----------------------
```