操作系统实验四 说明文档

项目结构

读者优先：lab4/reader

写者优先：lab4/writer

修改说明

读者优先

在框架代码7o的基础上修改

Makefile

增加了make run命令

```
run : image
    bochs
```

const.h

修改系统调用方法总数和新的输出颜色

```
/* Color */
/*
 * e.g. MAKE_COLOR(BLUE, RED)
 *      MAKE_COLOR(BLACK, RED) | BRIGHT
 *      MAKE_COLOR(BLACK, RED) | BRIGHT | FLASH
 */
#define BLACK   0x0     /* 0000 */
#define WHITE   0x7     /* 0111 */
#define RED     0x4     /* 0100 */
#define GREEN   0x2     /* 0010 */
#define BLUE    0x1     /* 0001 */
#define YELLOW  0x6     /* 0110 */
#define PURPLE  0x5     /* 0101 */

/* system call */
#define NR_SYS_CALL     5
```

global.h

增加了供调度算法选择当前执行的进程的变量、信号量结构体、读写相关的锁、读写数量变量、当前状态变量（读或写或普通）和消除饥饿开关。

```
EXTERN  PROCESS*    p_proc_pre;

typedef struct semaphore {
    int val;
    PROCESS* queue[NR_TASKS];
}Semaphore;
```

```
EXTERN Semaphore readMutex;
EXTERN Semaphore writeMutex;
EXTERN Semaphore readCountMutex;
EXTERN Semaphore writeMutexMutex;

EXTERN int maxReadNum;
EXTERN int maxWriteNum;
EXTERN int readCount;
EXTERN int prereadCount;

EXTERN char status;
EXTERN int  hungry;
```

proc.h

修改了PROCESS结构体，加入了一些变量。修改了tasks和procs的数量、相关任务堆栈大小和总堆栈大小，使适用于后面实现的读者写者问题。

```
typedef struct s_proc {
    STACK_FRAME regs;          /* process registers saved in stack frame */

    u16 ldt_sel;               /* gdt selector giving ldt base and limit */
    DESCRIPTOR ldts[LDT_SIZE]; /* local descriptors for code and data */

        int ticks;                 /* remained ticks */
        int priority;

    u32 pid;                   /* process id passed in from MM */
    char p_name[16];           /* name of the process */

    int nr_tty;

    int sleep_to_ticks;

    int block;

    char type;

    int needTimeSlice;

    int done;
}PROCESS;

/* Number of tasks & procs */
#define NR_TASKS    6
#define NR_PROCS    0

#define STACK_SIZE_A        0x8000
#define STACK_SIZE_B        0x8000
#define STACK_SIZE_C        0x8000
#define STACK_SIZE_D        0x8000
#define STACK_SIZE_E        0x8000
#define STACK_SIZE_F        0x8000

#define STACK_SIZE_TOTAL    (STACK_SIZE_A + \
                STACK_SIZE_B + \
```

```
                STACK_SIZE_C + \
                STACK_SIZE_D + \
                STACK_SIZE_E + \
                STACK_SIZE_F)
```

proto.h

添加了实验增加的各种函数的声明

```
/* main.c */
void A();
void B();
void C();
void D();
void E();
void F();

/* 系统调用 - 系统级 */
/* proc.c */
PUBLIC  int     sys_get_ticks();
// PUBLIC  int     sys_write(char* buf, int len, PROCESS* p_proc);
PUBLIC  int     sys_write(char* buf);
PUBLIC  void    sys_sleep(int milli_seconds);
PUBLIC  void    sys_p(void* mutex);
PUBLIC  void    sys_v(void* mutex);

/* 系统调用 - 用户级 */
PUBLIC  int     get_ticks();
PUBLIC  void    write(char* buf);
PUBLIC  void    sleep(int milli_seconds);
PUBLIC  void    p(void* mutex);
PUBLIC  void    v(void* mutex);
```

clock.c

修改了clock_handler函数，弃用了PROCESS结构体中的ticks，并增加了对调度函数的调用

```
PUBLIC void clock_handler(int irq)
{
    ticks++;
    p_proc_ready->ticks--;

    /*
        if (k_reenter != 0) {
        return;
    }

    if (p_proc_ready->ticks > 0) {
        return;
    }
        */

    schedule();

}
```

global.c

修改了task_table，将任务替换为读者写者问题的内容。修改了系统调用表，加入了新增的系统调用

```
PUBLIC  TASK    task_table[NR_TASKS] = {
    //{task_tty, STACK_SIZE_TTY, "tty"}
    {A, STACK_SIZE_A, "A"},
    {B, STACK_SIZE_B, "B"},
    {C, STACK_SIZE_C, "C"},
    {D, STACK_SIZE_D, "D"},
    {E, STACK_SIZE_E, "E"},
    {F, STACK_SIZE_F, "F"},};

PUBLIC  system_call sys_call_table[NR_SYS_CALL] = {sys_get_ticks, sys_write,
sys_sleep, sys_p, sys_v};
```

kernel.asm

这里借用了7o的框架代码中已有的系统调用，不过框架的是有三个参数，我改成了只有一个参数

```
sys_call:
        call    save
    ; push  dword [p_proc_ready]
        sti

    push    ecx
    ; push  ebx
        call    [sys_call_table + eax * 4]
    ; add   esp, 4 * 3
    add esp, 4

        mov     [esi + EAXREG - P_STACKBASE], eax
        cli
        ret
```

main.c

修改了任务表初始化和各种全局变量初始化部分，以及增加了各个任务的定义。

```
PUBLIC int kernel_main()
{
    ...
    proc_table[0].needTimeSlice = proc_table[0].priority = 2;
    proc_table[1].needTimeSlice = proc_table[1].priority = 3;
    proc_table[2].needTimeSlice = proc_table[2].priority = 3;
    proc_table[3].needTimeSlice = proc_table[3].priority = 3;
    proc_table[4].needTimeSlice = proc_table[4].priority = 4;
    proc_table[5].needTimeSlice = proc_table[5].priority = 1;

    proc_table[0].type = proc_table[1].type = proc_table[2].type = 'r';
    proc_table[3].type = proc_table[4].type = 'w';
    proc_table[5].type = 'n';

    proc_table[0].done = proc_table[1].done = proc_table[2].done =
proc_table[3].done = proc_table[4].done = proc_table[5].done = 0;
```

```
    //proc_table[1].nr_tty = 0;
    //proc_table[2].nr_tty = 1;
    //proc_table[3].nr_tty = 1;

    k_reenter = 0;
    ticks = 0;

    // 饿死
    hungry = 1;

    p_proc_ready = proc_table;
    p_proc_pre = proc_table;

    // 读者上限
    maxReadNum = readMutex.val = 1;
    maxWriteNum = writeMutex.val = writeMutexMutex.val = 1;
    readCountMutex.val = 1;
    readCount = prereadCount = 0;

    init_clock();
    init_keyboard();

    disp_pos = 0;
    for(i=0;i<80*25;i++){
        disp_str(" ");
    }
    disp_pos = 0;
    ...
}


void A()
{
    Reader("A");
}

void B()
{
    Reader("B");
}

void C()
{
    Reader("C");
}

void D()
{
    Writer("D");
}

void E()
{
    Writer("E");
}

void F()
{
```

```
    Normal("F");
}

void Reader(char *name)
{
    sleep(1000);
    while (1)
    {
        p(&readCountMutex);
        if (prereadCount == 0)
        {
            p(&writeMutex);
        }
        prereadCount++;
        v(&readCountMutex);
        p(&readMutex);
        readCount++;

        write(name);
        write(" start. ");

        for (int i = 0; i < p_proc_ready->needTimeSlice; ++i)
        {
            write(name);
            write(" reading. ");
            if (i == p_proc_ready->needTimeSlice - 1)
            {
                write(name);
                write(" finish. ");
            }
            else
            {
                milli_delay(1000);
            }
        }

        readCount--;
        v(&readMutex);
        p(&readCountMutex);
        if (prereadCount == 1)
        {
            v(&writeMutex);
        }
        prereadCount--;
        v(&readCountMutex);

        p_proc_ready->done = hungry;
        milli_delay(1000);
    }
}

void Writer(char *name)
{
    while (1)
    {
        p(&writeMutexMutex);
        p(&writeMutex);
```

```
        write(name);
        write(" start. ");

        for (int i = 0; i < p_proc_ready->needTimeSlice; ++i)
        {
            write(name);
            write(" writing. ");
            if (i == p_proc_ready->needTimeSlice - 1)
            {
                write(name);
                write(" finish. ");
            }
            else
            {
                milli_delay(1000);
            }
        }

        v(&writeMutex);
        v(&writeMutexMutex);

        p_proc_ready->done = hungry;
        milli_delay(1000);
    }
}

void Normal(char *name)
{
    while (1)
    {
        switch (status)
        {
        case 'r':
            write(" READ! ");
            write("reader num is ");
            char num[4] = {'0' + readCount, '.', ' ', '\0'};
            write(num);
            break;
        case 'w':
            write(" WRITE! ");
            break;
        default:
            write(" START! ");
            break;
        }
        sleep(1000);
    }
}
```

proc.c

修改了调度函数，并添加了新增的系统调用函数

```
PUBLIC void schedule()
{
    disable_irq(CLOCK_IRQ);
```

```
    int allDone = 1;
    for (PROCESS *p = proc_table; p < proc_table + NR_TASKS + NR_PROCS - 1; ++p)
    {
        if (p->done == 0)
        {
            allDone = 0;
            break;
        }
    }

    if (allDone == 1)
    {
        for (PROCESS *p = proc_table; p < proc_table + NR_TASKS + NR_PROCS - 1;
++p)
        {
            p->done = 0;
        }
    }

    if ((proc_table + NR_TASKS + NR_PROCS - 1)->block == 0 && get_ticks() >=
(proc_table + NR_TASKS + NR_PROCS - 1)->sleep_to_ticks)
    {
        p_proc_ready = proc_table + NR_TASKS + NR_PROCS - 1;
    }
    else
    {
        int flag = 0;
        while (1)
        {
            if (p_proc_pre->done == 0 && p_proc_pre->block == 0 && get_ticks()
>= p_proc_pre->sleep_to_ticks)
            {
                p_proc_ready = p_proc_pre;
                flag = 1;
            }
            p_proc_pre++;
            if (p_proc_pre == (proc_table + NR_TASKS + NR_PROCS - 1))
            {
                p_proc_pre = proc_table;
            }
            if (flag == 1)
            {
                break;
            }
        }
    }

    if (p_proc_ready->type != 'n')
    {
        status = p_proc_ready->type;
    }

    enable_irq(CLOCK_IRQ);
}

PUBLIC int sys_get_ticks()
{
    return ticks;
```

```
}

PUBLIC void sys_sleep(int milli_seconds)
{
    p_proc_ready->sleep_to_ticks = get_ticks() + milli_seconds / (1000 / HZ);
    schedule();
}

PUBLIC void sys_p(void *mutex)
{
    disable_irq(CLOCK_IRQ);
    Semaphore *semaphore = (Semaphore *)mutex;
    semaphore->val--;
    if (semaphore->val < 0)
    {
        getDown(semaphore);
    }
    enable_irq(CLOCK_IRQ);
}

PUBLIC void sys_v(void *mutex)
{
    disable_irq(CLOCK_IRQ);
    Semaphore *semaphore = (Semaphore *)mutex;
    semaphore->val++;
    if (semaphore->val <= 0)
    {
        wakeUp(semaphore);
    }
    enable_irq(CLOCK_IRQ);
}

PUBLIC void getDown(Semaphore *mutex)
{
    int index = -(mutex->val + 1);
    mutex->queue[index] = p_proc_ready;
    p_proc_ready->block = 1;
    schedule();
}

PUBLIC void wakeUp(Semaphore *mutex)
{
    PROCESS *process = mutex->queue[0];
    int max = 0;
    int index = -mutex->val + 1;
    for (int i = 0; i < index; ++i)
    {
        if (mutex->queue[i]->priority > process->priority)
        {
            max = i;
            process = mutex->queue[i];
        }
    }
    for (int i = max + 1; i < index; ++i)
    {
        mutex->queue[i - 1] = mutex->queue[i];
    }
    process->block = 0;
```

```
}
```

syscall.asm

增加了新增系统调用的调用号、系统调用的导出符号、系统调用的汇编代码

```
INT_VECTOR_SYS_CALL equ 0x90
_NR_get_ticks       equ 0
_NR_write       equ 1
_NR_sleep       equ 2
_NR_p               equ 3
_NR_v               equ 4

; 导出符号
global  get_ticks
global  write
global  sleep
global  p
global  v

get_ticks:
    mov eax, _NR_get_ticks
    int INT_VECTOR_SYS_CALL
    ret

write:
        mov     eax, _NR_write
        mov     ecx, [esp + 4]
        int     INT_VECTOR_SYS_CALL
        ret

sleep:
        mov     eax, _NR_sleep
        mov     ecx, [esp + 4]
        int     INT_VECTOR_SYS_CALL
        ret

p:
        mov     eax, _NR_p
        mov     ecx, [esp + 4]
        int     INT_VECTOR_SYS_CALL
        ret

v:
        mov     eax, _NR_v
        mov     ecx, [esp + 4]
        int     INT_VECTOR_SYS_CALL
        ret
```

tty.c

修改了原框架中的sys_write函数，使其支持接收单个char*参数，并能根据不同进程输出不同颜色的文字。

```
PUBLIC int sys_write(char* str)
{
        // tty_write(&tty_table[p_proc->nr_tty], buf, len);
        int offset = p_proc_ready - proc_table;
        switch(offset)
        {
            case 0:
                disp_color_str(str, BRIGHT | MAKE_COLOR(BLACK, RED));
                break;
            case 1:
                disp_color_str(str, BRIGHT | MAKE_COLOR(BLACK, YELLOW));
                break;
            case 2:
                disp_color_str(str, BRIGHT | MAKE_COLOR(BLACK, BLUE));
                break;
            case 3:
                disp_color_str(str, BRIGHT | MAKE_COLOR(BLACK, GREEN));
                break;
            case 4:
                disp_color_str(str, BRIGHT | MAKE_COLOR(BLACK, PURPLE));
                break;
            default:
                disp_str(str);
                break;
        }
        return 0;
}
```

kliba.asm

因为出现了部分输出被吞的问题，在disp_str和disp_color_str的汇编代码中加入对参数的保存，即esi、edi的入栈和出栈，修改完以后输出正常。

```
disp_str:
    push esi
    push edi
    push    ebp

    ...

    pop ebp
    pop edi
    pop esi
    ret

disp_color_str:
    push esi
    push edi
    push    ebp

    ...
```

```
    pop ebp
    pop edi
    pop esi
    ret
```

写者优先

在读者优先的代码基础上进行修改

global.h

修改了读写相关的锁和读写数量变量，其他不变。

```
EXTERN Semaphore readMutex;
EXTERN Semaphore writeMutex;
EXTERN Semaphore readCountMutex;
EXTERN Semaphore writeCountMutex;
EXTERN Semaphore readPermission;
EXTERN Semaphore readPermissionMutex;

EXTERN int maxReadNum;
EXTERN int maxWriteNum;
EXTERN int readCount;
EXTERN int prereadCount;
EXTERN int writeCount;
```

main.c

修改了任务表初始化和各种全局变量初始化部分、reader函数和writer函数，以支持写者优先。

```
PUBLIC int kernel_main()
{
    ...

    // 读者上限
    maxReadNum = readMutex.val = 1;
    maxWriteNum = writeMutex.val = writeCountMutex.val = 1;
    readCountMutex.val = 1;
    readPermission.val = 1;
    readPermissionMutex.val = 1;
    readCount = prereadCount = writeCount = 0;

    ...
}

void Reader(char *name)
{
    while (1)
    {
        p(&readPermissionMutex);
        p(&readPermission);
        p(&readCountMutex);
        if (prereadCount == 0)
        {
            p(&writeMutex);
```

```
        }
        prereadCount++;
        v(&readCountMutex);
        v(&readPermission);
        v(&readPermissionMutex);
        p(&readMutex);
        readCount++;

        write(name);
        write(" start. ");

        for (int i = 0; i < p_proc_ready->needTimeSlice; ++i)
        {
            write(name);
            write(" reading. ");
            if (i == p_proc_ready->needTimeSlice - 1)
            {
                write(name);
                write(" finish. ");
            }
            else
            {
                milli_delay(1000);
            }
        }

        readCount--;
        v(&readMutex);
        p(&readCountMutex);
        prereadCount--;
        if (prereadCount == 0)
        {
            v(&writeMutex);
        }
        v(&readCountMutex);

        p_proc_ready->done = hungry;
        milli_delay(1000);
    }
}

void Writer(char *name)
{
    sleep(1000);
    while (1)
    {
        p(&writeCountMutex);
        if (writeCount == 0)
        {
            p(&readPermission);
        }
        writeCount++;
        v(&writeCountMutex);
        p(&writeMutex);

        write(name);
        write(" start. ");
```

```
        for (int i = 0; i < p_proc_ready->needTimeSlice; ++i)
        {
            write(name);
            write(" writing. ");
            if (i == p_proc_ready->needTimeSlice - 1)
            {
                write(name);
                write(" finish. ");
            }
            else
            {
                milli_delay(1000);
            }
        }

        v(&writeMutex);
        p(&writeCountMutex);
        if (writeCount == 1)
        {
            v(&readPermission);
        }
        writeCount--;
        v(&writeCountMutex);

        p_proc_ready->done = hungry;
        milli_delay(1000);
    }
```

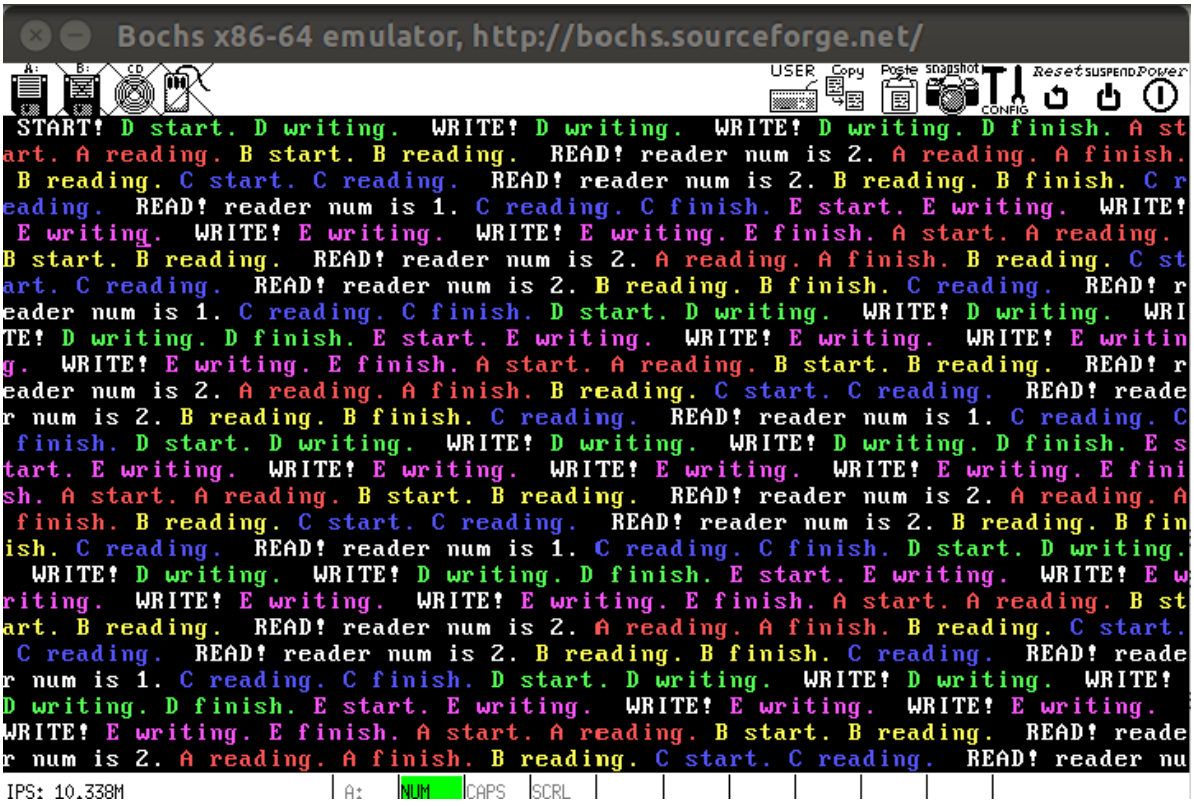
运行截图

已开启无饥饿。消除饥饿的原理是利用结构体新增的done变量，若该进程的任务执行完成，则done置1。每次调度只调度未完成（即done为0）的进程执行，若所有进程的任务都完成了，再全部把done置0，重新开始调度。这样可以消除饥饿。

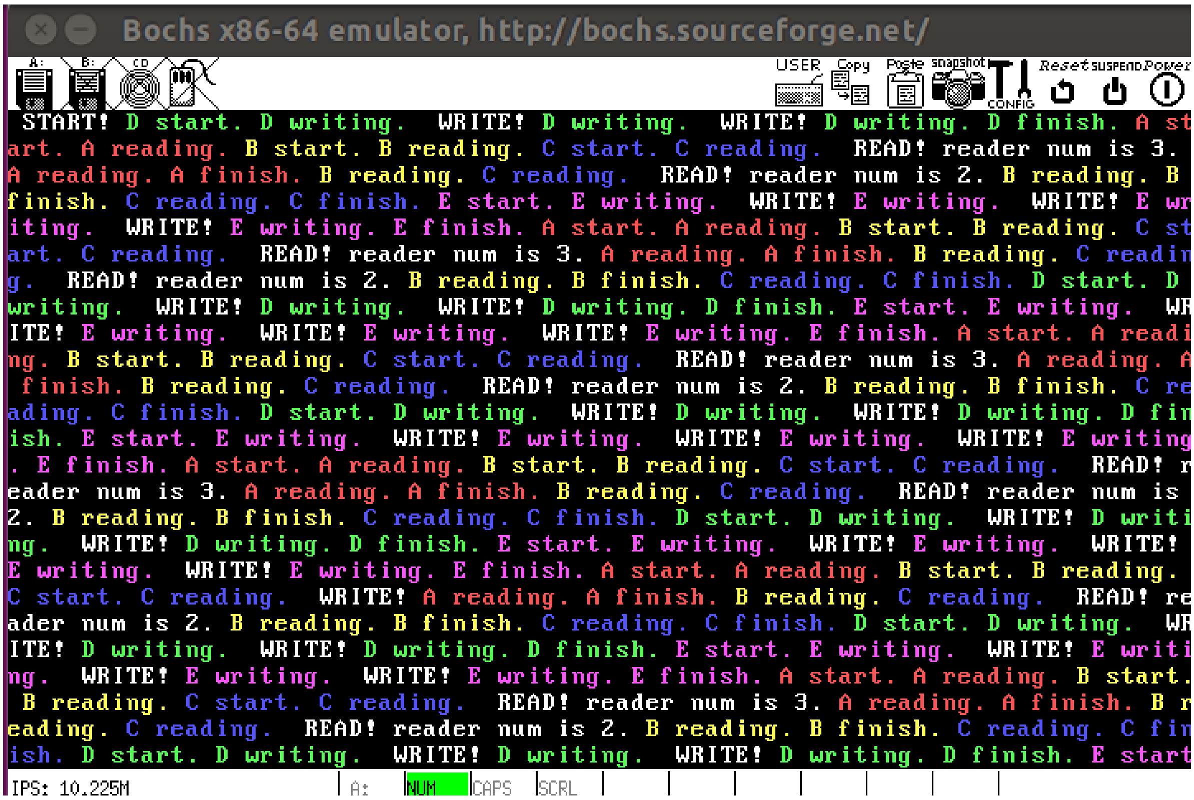
读者优先

读者上限数为1

读者上限数为2



读者上限数为3



写者优先

读者上限数为1

读者上限数为2


Bochs x86-64 emulator, http://bochs.sourceforge.net/
START! A start. A reading. B start. B reading. READ! reader num is 2. A readin
g. A finish. B reading. C start. C reading. READ! reader num is 2. B reading. B
finish. C reading. READ! reader num is 1. C reading. C finish. E start. E writ
ing. WRITE! E writing. WRITE! E writing. WRITE! E writing. E finish. D start.
D writing. WRITE! D writing. WRITE! D writing. D finish. E start. E writing.
WRITE! E writing. WRITE! E writing. WRITE! E writing. E finish. D start. D wr
iting. WRITE! D writing. WRITE! D writing. D finish. A start. A reading. B sta
rt. B reading. READ! reader num is 2. A reading. A finish. B reading. C start.
C reading. READ! reader num is 2. B reading. B finish. C reading. READ! reader
num is 1. C reading. C finish. D start. D writing. READ! reader num is 0. D wr
iting. WRITE! D writing. D finish. E start. E writing. WRITE! E writing. WRIT
E! E writing. WRITE! E writing. E finish. A start. A reading. B start. B readin
g. READ! reader num is 2. A reading. A finish. B reading. C start. C reading.
READ! reader num is 2. B reading. B finish. C reading. READ! reader num is 1. C
reading. C finish. D start. D writing. WRITE! D writing. WRITE! D writing. D
finish. E start. E writing. WRITE! E writing. WRITE! E writing. WRITE! E writ
ing. E finish. A start. A reading. B start. B reading. READ! reader num is 2. A
reading. A finish. B reading. C start. C reading. READ! reader num is 2. B rea
ding. B finish. C reading. READ! reader num is 1. C reading. C finish. D start.
D writing. READ! reader num is 0. D writing. WRITE! D writing. D finish. E st
art. E writing. WRITE! E writing. WRITE! E writing. WRITE! E writing. E finis
h. A start. A reading. B start. B reading. READ! reader num is 2. A reading. A
finish. B reading. C start. C reading. READ! reader num is 2. B reading. B fini
sh. C reading. READ! reader num is 1. C reading. C finish. D start. D writing.
WRITE! D writing. WRITE! D writing. D finish. E start. E writing. WRITE! E wr
IPS: 12.228M
A:
NUM
CAPS
SCRL

读者上限数为3


Bochs x86-64 emulator, http://bochs.sourceforge.net/
START! A start. A reading. B start. B reading. C start. C reading. READ! reade
r num is 3. A reading. A finish. B reading. C reading. READ! reader num is 2. B
reading. B finish. C reading. C finish. E start. E writing. WRITE! E writing.
WRITE! E writing. WRITE! E writing. E finish. D start. D writing. WRITE! D wr
iting. WRITE! D writing. D finish. E start. E writing. WRITE! E writing. WRIT
E! E writing. WRITE! E writing. E finish. D start. D writing. WRITE! D writing
. WRITE! D writing. D finish. A start. A reading. B start. B reading. C start.
C reading. READ! reader num is 3. A reading. A finish. B reading. C reading. R
EAD! reader num is 2. B reading. B finish. C reading. C finish. D start. D writi
ng. READ! reader num is 0. D writing. WRITE! D writing. D finish. E start. E w
riting. WRITE! E writing. WRITE! E writing. WRITE! E writing. E finish. A sta
rt. A reading. B start. B reading. C start. C reading. READ! reader num is 3. A
reading. A finish. B reading. C reading. READ! reader num is 2. B reading. B f
inish. C reading. C finish. D start. D writing. WRITE! D writing. WRITE! D wri
ting. D finish. E start. E writing. WRITE! E writing. WRITE! E writing. WRITE
! E writing. E finish. A start. A reading. B start. B reading. C start. C readin
g. READ! reader num is 3. A reading. A finish. B reading. C reading. READ! rea
der num is 2. B reading. B finish. C reading. C finish. D start. D writing. REA
D! reader num is 0. D writing. WRITE! D writing. D finish. E start. E writing.
WRITE! E writing. WRITE! E writing. WRITE! E writing. E finish. A start. A re
ading. B start. B reading. C start. C reading. READ! reader num is 3. A reading
. A finish. B reading. C reading. READ! reader num is 2. B reading. B finish. C
reading. C finish. D start. D writing. READ! reader num is 0. D writing. WRIT
E! D writing. D finish. E start. E writing. WRITE! E writing. WRITE! E writing
. WRITE! E writing. E finish. A start. A reading. B start. B reading. C start.
IPS: 14.446M
A:
NUM
CAPS
SCRL