1.实现延时的系统调用

首先在`syscall.asm`中定义系统调用

```
_NR_mills_sleep     equ 1
;设置系统调用
global  mills_sleep
; 导出符号
mills_sleep:
    mov eax, _NR_mills_sleep
    mov ecx, [esp+4];由于需要参数
    int INT_VECTOR_SYS_CALL
    ret
```

由于系统调用的中断向量已经被书中代码实现，故我们只需要设置即可。

在`const.h`中设置系统调用个数

`#define NR_SYS_CALL     2`

然后去`global.c`中修改系统调用表

`PUBLIC  system_call       sys_call_table[NR_SYS_CALL] = {sys_get_ticks,sys_mills_sleep};`

系统调用表实际上就是函数指针，到这一步，系统已经可以接受你的系统调用并按照分配好的函数名进行调用了！下一步只要实现这个函数就好。

```
//proc.h->s_proc
int wake_tick;//记录进程睡醒的时刻
//proc.c
PUBILC void sys_mills_sleep(int milli_seconds){
    p_proc_ready->wake_tick = get_ticks() + milli_seconds / (1000 / HZ);
    //设置进程将在这个tick被唤醒
    schedule();
}
//proto.h
PUBLIC  void     sys_mills_sleep(int milli_seconds);
PUBLIC  void    mills_sleep(int milli_seconds);
```

但是目前设置的“sleep”并没有真的让进程停下来，它只是为进程设置了一个“唤醒时间”。为了真的做到sleep，我们需要修改调度算法。

新建一个工具函数用于判断进程是否可用。

```
PUBLIC int isRunnable(PROCESS* p){
    if(p->wakeup_ticks <= get_ticks()&&p->isBlock == 0&&p->isDone ==0){
        return 1;
    }else{
        return 0;
    }
}
```

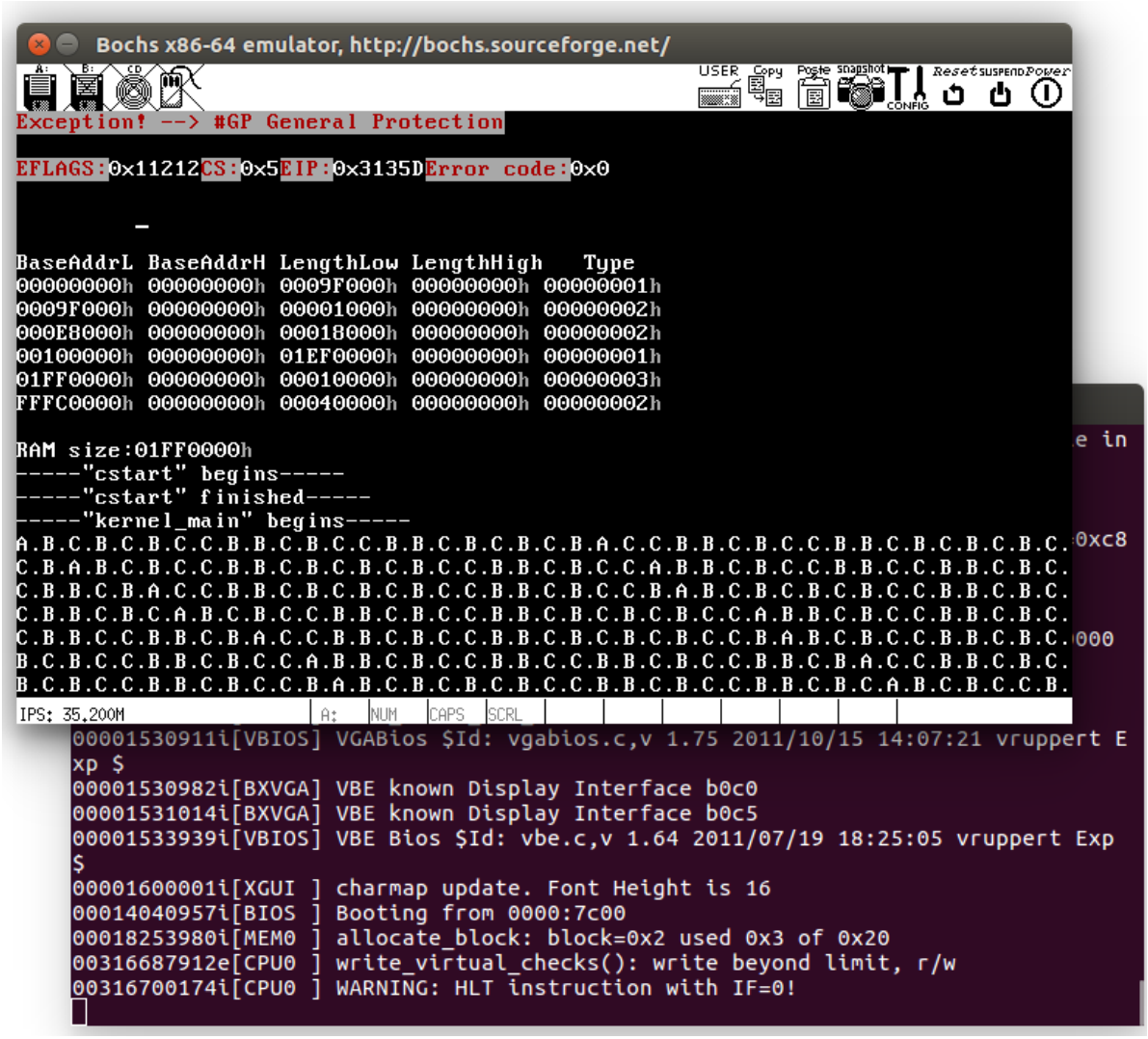
在Schedule中判断进程必须可用才行

```
PUBLIC void schedule()
{
    PROCESS* p;
    int  greatest_ticks = 0;

    while (!greatest_ticks) {
        for (p = proc_table; p < proc_table+NR_TASKS; p++) {
            if (p->ticks > greatest_ticks&&isRunnable(p)) {
                greatest_ticks = p->ticks;
                p_proc_ready = p;
            }
        }

        if (!greatest_ticks) {
            for (p = proc_table; p < proc_table+NR_TASKS; p++) {
                p->ticks = p->priority;
            }
        }
    }
}
```

到此，这项功能即可正常工作。

如图，将A的睡眠时间设置为B、C的10倍，可以看到A很少出现了。

还有一些报错，等待后面能解决



2.包装实现输出的系统调用

采用类似上述步骤，配置新增一个系统调用的过程。此处仅介绍一下具体的输出函数。

考虑到是要对每一个任务用不同的颜色，同一个进程不会有不同颜色。因此，我们只需要把进程的号作为不同颜色即可

```
PUBLIC void sys_my_print(char* str){
    if (disp_pos > 80 * 25 * 2){
        return;
    }
    //这是因为输出超过显存会报错，当前环境下还不能提供超过显存的输出，这是有待解决的问题
    switch(p_proc_ready - proc_table){
        case 0:
            disp_color_str(str, BRIGHT | MAKE_COLOR(BLACK, RED));
            break;
        case 1:
            disp_color_str(str, BRIGHT | MAKE_COLOR(BLACK, GREEN));
            break;
        case 2:
            disp_color_str(str, BRIGHT | MAKE_COLOR(BLACK, BLUE));
            break;
        case 5:
            disp_str(str);
            break;
        case 4:
            disp_color_str(str, BRIGHT | MAKE_COLOR(BLACK, PURPLE));
```

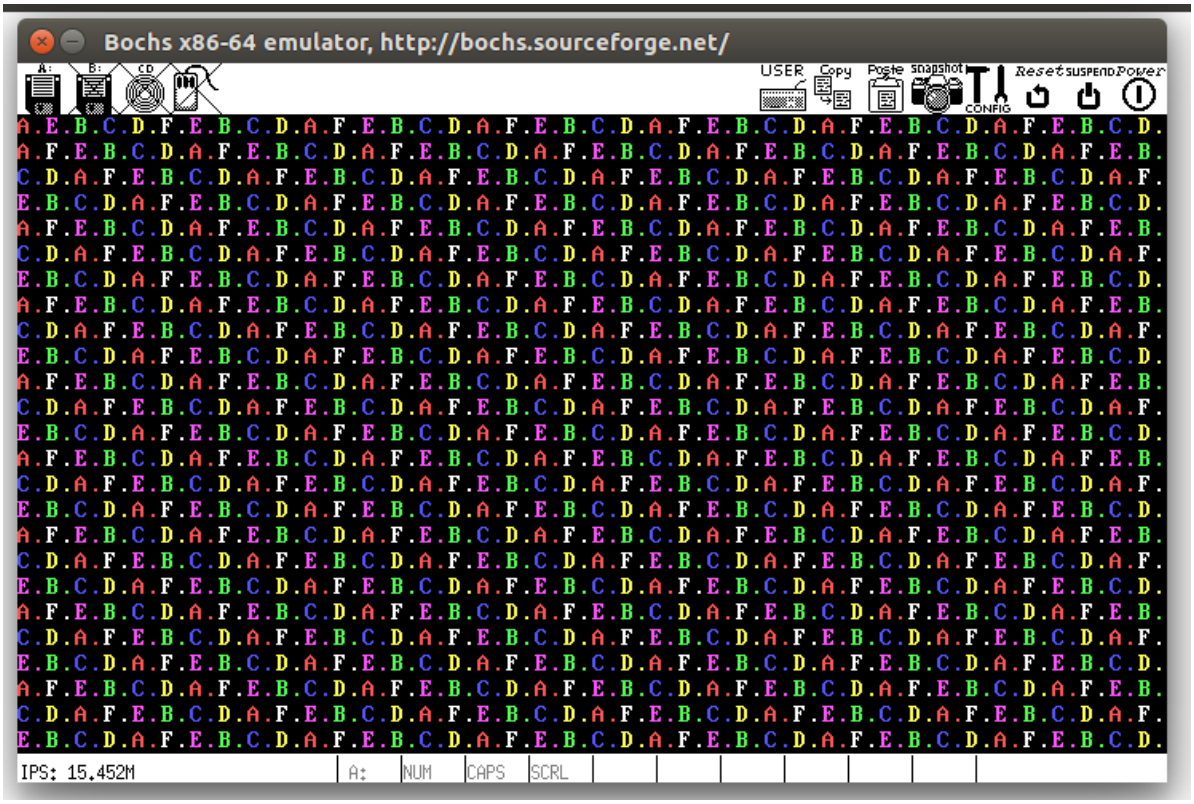
```
            break;
        case 3:
            disp_color_str(str, BRIGHT | MAKE_COLOR(BLACK, YELLO));
            break;
        default:
            disp_str(str);
            break;
    }
}
```

3.PV操作

3.1 新建几个进程

参考Orange'S 6.4.6的内容，为系统添加三个额外进程

```
void WriterD()
{
    while(1){
        my_print("D.");
        mills_sleep(10);
    }
}
void WriterE()
{
    while(1){
        my_print("E.");
        mills_sleep(10);
    }
}
void NormalF()
{
    while(1){
        my_print("F.");
        mills_sleep(10);
    }
}
```



3.2 PV操作系统调用

按照1中所述的添加系统调用的方法添加P、V两个系统调用。

首先在global.h中添加Semaphore的定义

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

然后完成对PV操作的实现

```
/*======================================================================*
                            sys_P
 *======================================================================*/
PUBLIC void sys_P(void *mutex){
    disable_irq(CLOCK_IRQ);//PV原语要求不能被中断
    Semaphore *semaphore = (Semaphore *)mutex;
    semaphore->value--;
    if (semaphore->value < 0){
        block(semaphore);
        //如果小于0，就要陷入阻塞
    }
    enable_irq(CLOCK_IRQ);
}

PUBLIC void block(Semaphore *mutex){
    mutex->queue[-mutex->value - 1] = p_proc_ready;
    //-mutex->value - 1正好保证了任务按顺序进入队列
    p_proc_ready->isBlock = 1; // 阻塞
    schedule();
}
/*======================================================================*
```

```
                                sys_V
 *======================================================================*/
PUBLIC void sys_V(void *mutex){
    disable_irq(CLOCK_IRQ);//PV原语要求不能被中断
    Semaphore *semaphore = (Semaphore *)mutex;
    semaphore->value++;
    if (semaphore->value <= 0){
        wake(semaphore);
        //如果小于0，就要唤醒一个被阻塞的进程
    }
    enable_irq(CLOCK_IRQ);
}

PUBLIC void wake(Semaphore *mutex){
    mutex->queue[0]->isBlock = 0;
    for(int i=0;i<-mutex->value;i++){
        mutex->queue[i] = mutex->queue[i+1];
    }
}
```

3.3 基于此背景下的调度算法

应该首先判断F进程是否可以运行，如果可以应该首先运行F进程，用于监视操作。然后应该按照进程顺序进行

```
PUBLIC void schedule()
{
    isAllDone();//如果所有进程都完成了，就重启
    PROCESS* p = proc_table+5;//F进程
    if(isRunnable(p)){
        p_proc_ready = p;
    }
    else{
        while(!isRunnable(ptr_schedule)){
            ptr_schedule++;
            if(ptr_schedule==p){
                ptr_schedule = proc_table;
            }
        }
        p_proc_ready = ptr_schedule;
        ptr_schedule++;
        if(ptr_schedule==p){
            ptr_schedule = proc_table;
        }
    }
    if(p_proc_ready-proc_table<=4){
        nowStatus = p_proc_ready-proc_table;
    }
}
```

4. 读优先算法

read

```
while(1){
            // 同时只有一个进程可以修改在读人数
```

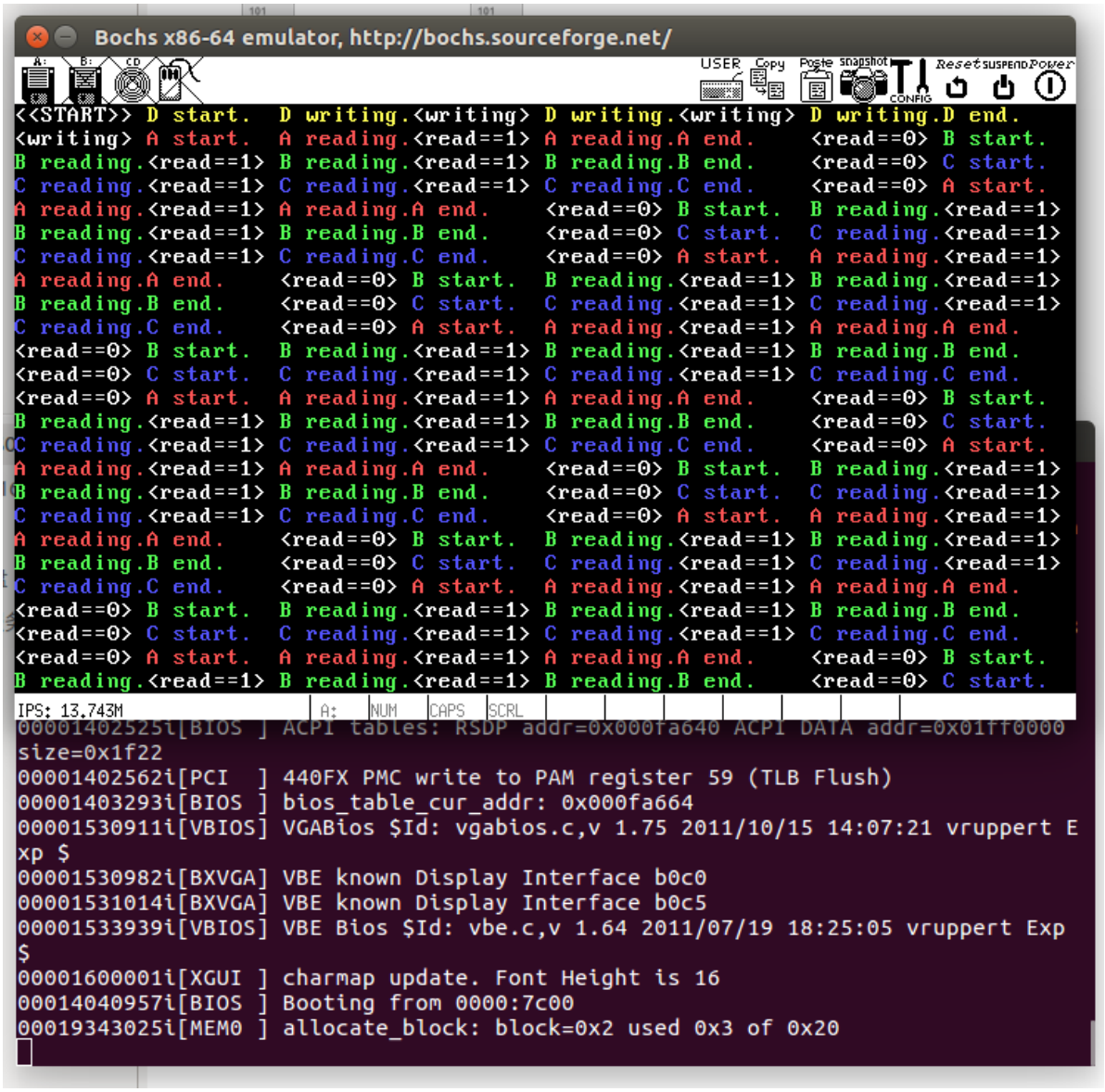
```
            P(&countMutex);
            if (readPreparedCount == 0)
            {
                //如果没人读了，就写
                P(&writeMutex);
            }
            readPreparedCount++;
            V(&countMutex);

            P(&readMutex);
            //限制同时读的人数
            readCount++;
            my_print(pname);
            my_print(" start.  ");
            int j;
            //用于输出运行信息
            for (j = 0; j < p_proc_ready->priority; ++j)
            {
                my_print(pname);
                my_print(readStr);
                if (j == p_proc_ready->priority - 1)
                {
                    my_print(pname);
                    my_print(endStr);
                }
                else
                {
                    milli_delay(10);
                }
            }
            readCount--;
            V(&readMutex);

            P(&countMutex);
            // 同时只有一个进程可以修改在读人数
            readPreparedCount--;
            if (readPreparedCount == 0)
            {
                V(&writeMutex);
            }
            V(&countMutex);

            p_proc_ready->isDone = solveHunger;
            milli_delay(10); // 废弃当前时间片，至少等到下个时间片才能进入循环
        }
```

write

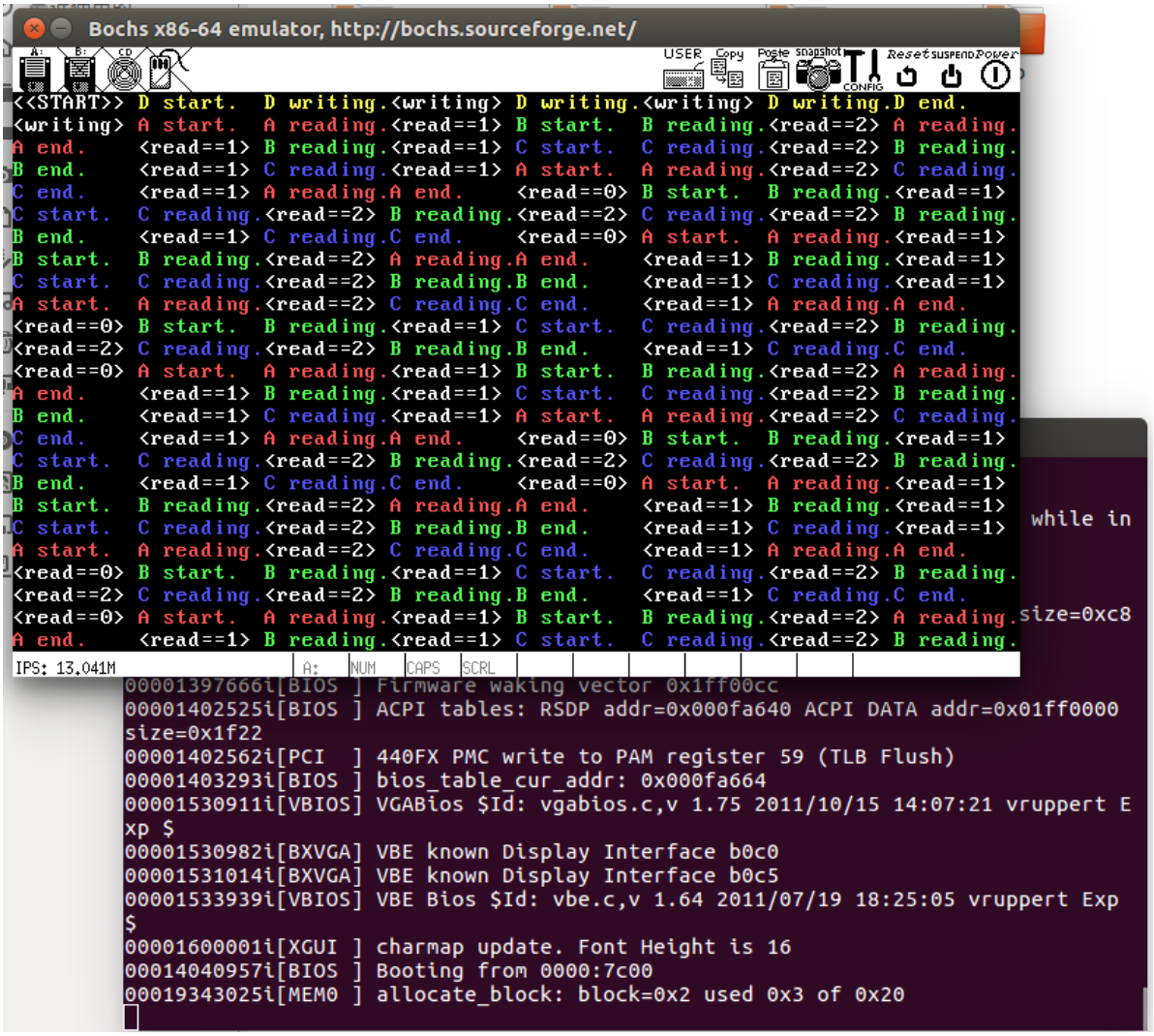
```
while (1){
            P(&writeMutexMutex); // 这样可以防止写进程结束之后唤醒下一个写进程，而不是被卡住的读进程
            P(&writeMutex);
            my_print(pname);
            my_print(" start.  ");
            int j;
            for (j = 0; j < p_proc_ready->priority; ++j)
            {
```

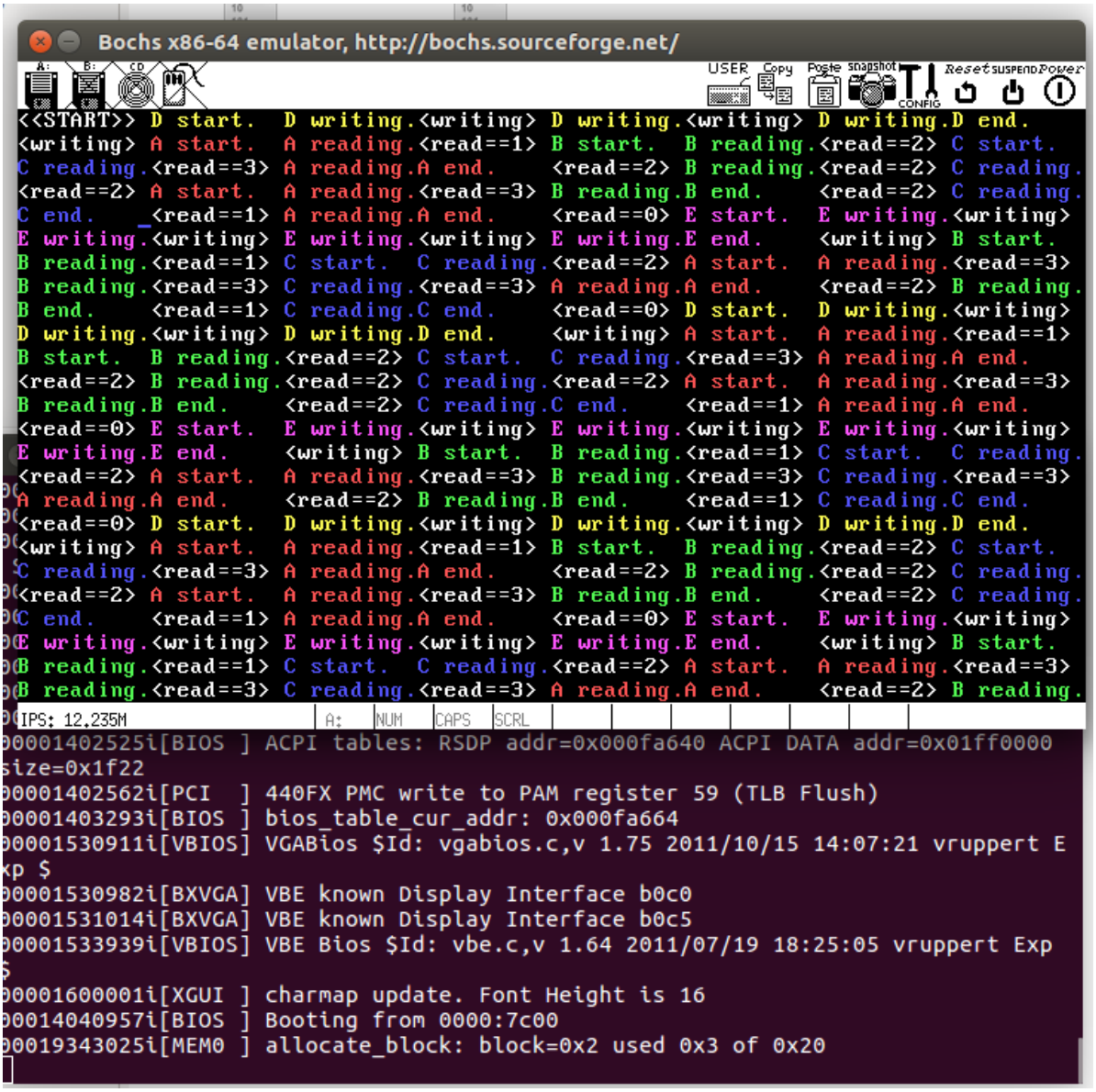
```
                my_print(pname);
                my_print(writeStr);
                if (j == p_proc_ready->priority - 1)
                {
                    my_print(pname);
                    my_print(endStr);
                }
                else
                {
                    milli_delay(10);
                }
            }
            V(&writeMutex);
            V(&writeMutexMutex);

            p_proc_ready->isDone = solveHunger;
            milli_delay(10);
        }
```

截图



```
```


Bochs x86-64 emulator, http://bochs.sourceforge.net/
USER
Copy
Paste
snapshot
CONFIG
Reset
suspend
Power
<<START>> D start. D writing.<writing> D writing.<writing> D writing.D end.
<writing> A start. A reading.<read==1> B start. B reading.<read==2> A reading.
A end. <read==1> B reading.<read==1> C start. C reading.<read==2> B reading.
B end. <read==1> C reading.<read==1> A start. A reading.<read==2> C reading.
C end. <read==1> A reading.A end. <read==0> B start. B reading.<read==1>
C start. C reading.<read==2> B reading.<read==2> C reading.<read==2> B reading.
B end. <read==1> C reading.C end. <read==0> A start. A reading.<read==1>
B start. B reading.<read==2> A reading.A end. <read==1> B reading.<read==1>
C start. C reading.<read==2> B reading.B end. <read==1> C reading.<read==1>
A start. A reading.<read==2> C reading.C end. <read==1> A reading.A end.
<read==0> B start. B reading.<read==1> C start. C reading.<read==2> B reading.
<read==2> C reading.<read==2> B reading.B end. <read==1> C reading.C end.
<read==0> A start. A reading.<read==1> B start. B reading.<read==2> A reading.
A end. <read==1> B reading.<read==1> C start. C reading.<read==2> B reading.
B end. <read==1> C reading.<read==1> A start. A reading.<read==2> C reading.
C end. <read==1> A reading.A end. <read==0> B start. B reading.<read==1>
C start. C reading.<read==2> B reading.<read==2> C reading.<read==2> B reading.
B end. <read==1> C reading.C end. <read==0> A start. A reading.<read==1>
B start. B reading.<read==2> A reading.A end. <read==1> B reading.<read==1>
C start. C reading.<read==2> B reading.B end. <read==1> C reading.<read==1>
A start. A reading.<read==2> C reading.C end. <read==1> A reading.A end.
<read==0> B start. B reading.<read==1> C start. C reading.<read==2> B reading.
<read==2> C reading.<read==2> B reading.B end. <read==1> C reading.C end.
<read==0> A start. A reading.<read==1> B start. B reading.<read==2> A reading.
A end. <read==1> B reading.<read==1> C start. C reading.<read==2> B reading.
IPS: 13.041M
A:
NUM
CAPS
SCRL
while in
size=0xc8
00001397666i[BIOS] Firmware waking vector 0x1ff00cc
00001402525i[BIOS] ACPI tables: RSDP addr=0x000fa640 ACPI DATA addr=0x01ff0000
size=0x1f22
00001402562i[PCI] 440FX PMC write to PAM register 59 (TLB Flush)
00001403293i[BIOS] bios_table_cur_addr: 0x000fa664
00001530911i[VBIOS] VGABios $Id: vgabios.c,v 1.75 2011/10/15 14:07:21 vruppert E
xp $
00001530982i[BXVGA] VBE known Display Interface b0c0
00001531014i[BXVGA] VBE known Display Interface b0c5
00001533939i[VBIOS] VBE Bios $Id: vbe.c,v 1.64 2011/07/19 18:25:05 vruppert Exp
$
00001600001i[XGUI] charmap update. Font Height is 16
00014040957i[BIOS] Booting from 0000:7c00
00019343025i[MEM0] allocate_block: block=0x2 used 0x3 of 0x20



5.写优先算法

reader

```
while (1)
        {
            P(&readPermissionMutex); // 保证只有一个被卡在readPermission
            P(&readPermission);
            // 判断修改在读人数
            P(&readCountMutex);
            if (readPreparedCount == 0)
            {
                P(&writeMutex);
                //夺取写权限，或等待写完
            }
            readPreparedCount++;
            V(&readCountMutex);
            V(&readPermission);
            V(&readPermissionMutex);//保证优先写

            P(&readMutex);
            readCount++;
            my_print(pname);
```

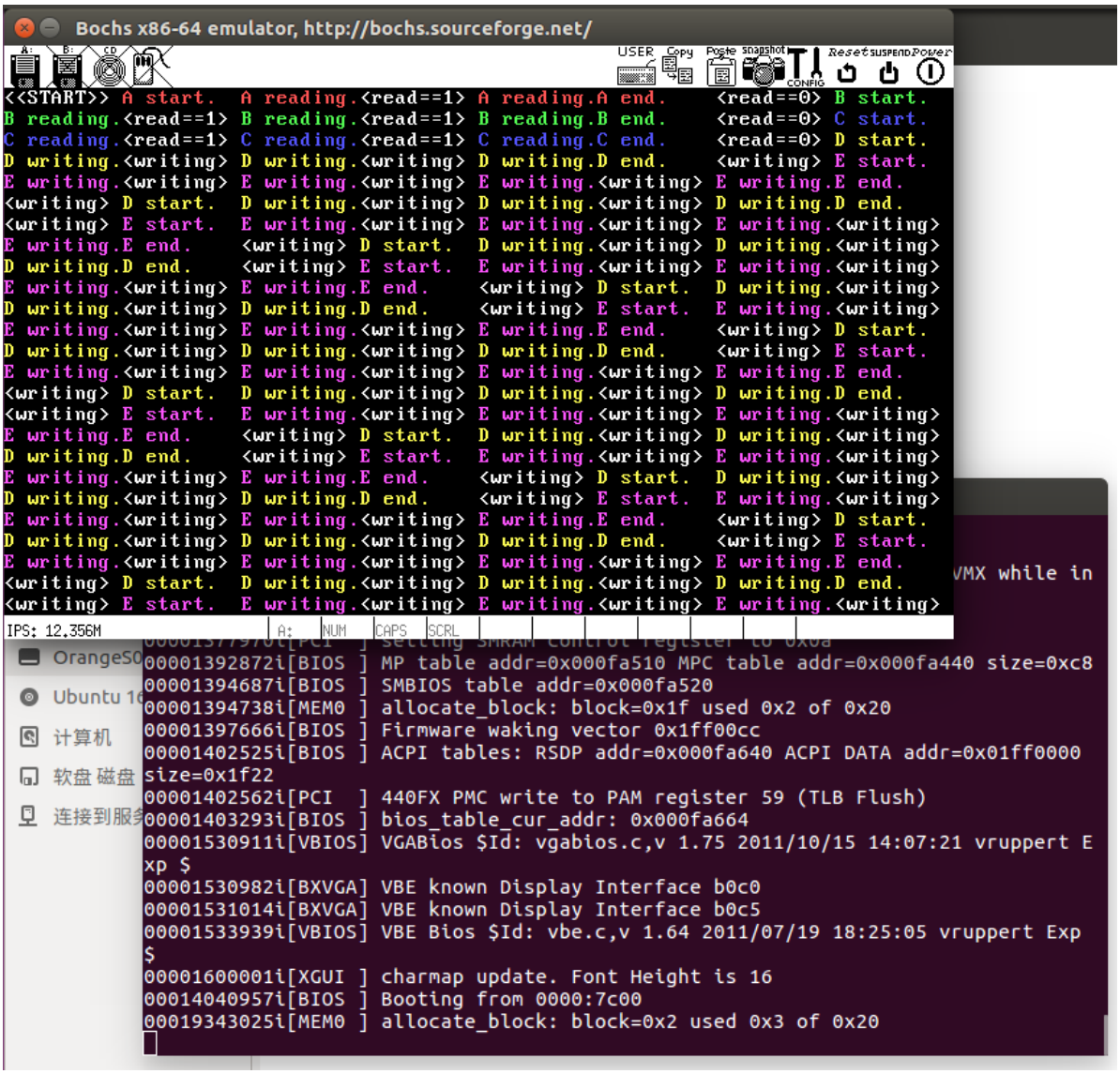
```
            my_print(" start.  ");
            int j;
            for (j = 0; j < p_proc_ready->priority; ++j)
            {
                my_print(pname);
                my_print(readStr);
                if (j == p_proc_ready->priority- 1)
                {
                    my_print(pname);
                    my_print(endStr);
                }
                else
                {
                    milli_delay(10);
                }
            }
            readCount--;
            V(&readMutex);

            P(&readCountMutex);
            readPreparedCount--;
            if (readPreparedCount == 0)
            {
                V(&writeMutex);
            }
            V(&readCountMutex);

            p_proc_ready->isDone = solveHunger;
            milli_delay(10);
        }
```

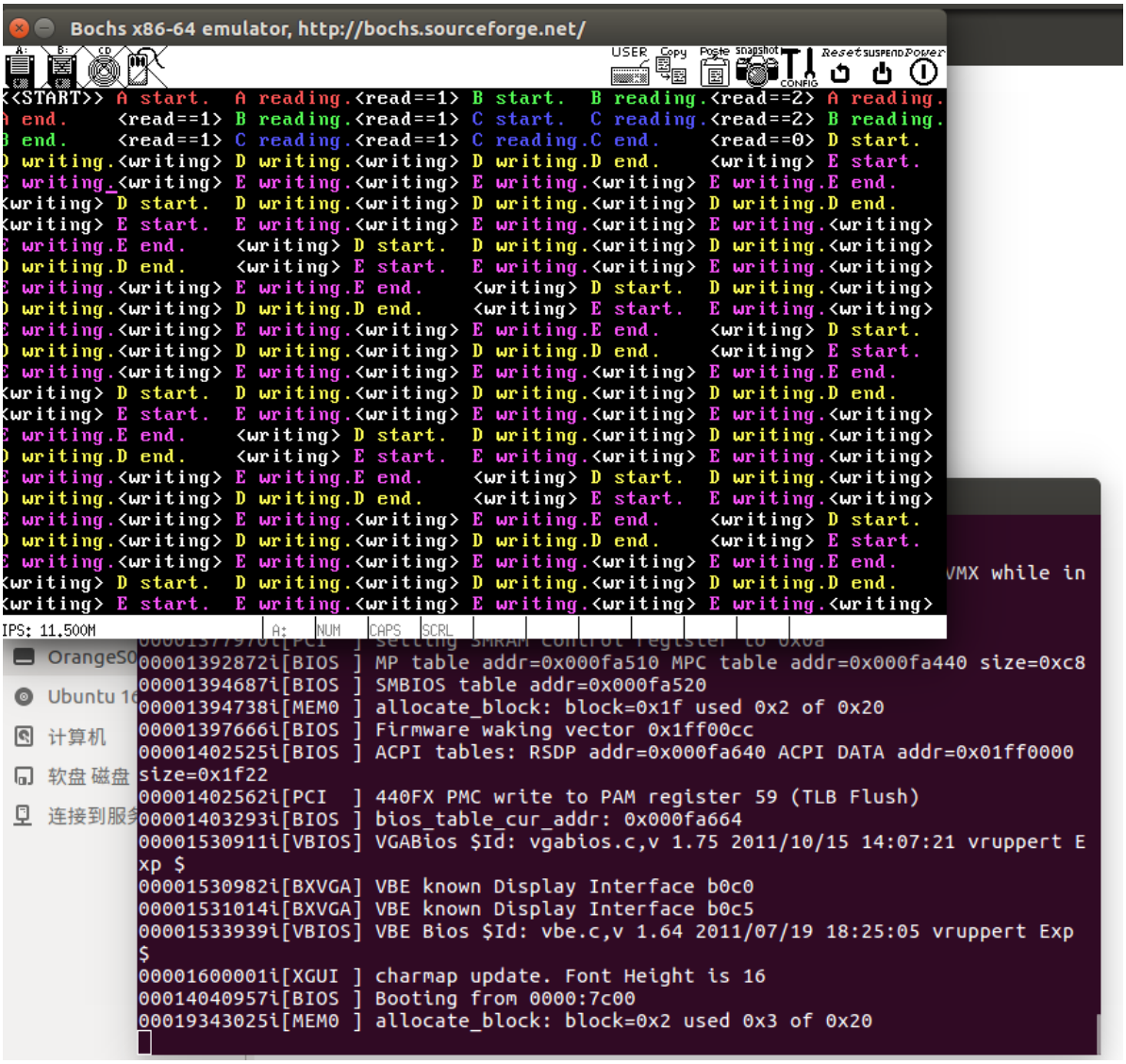
writer

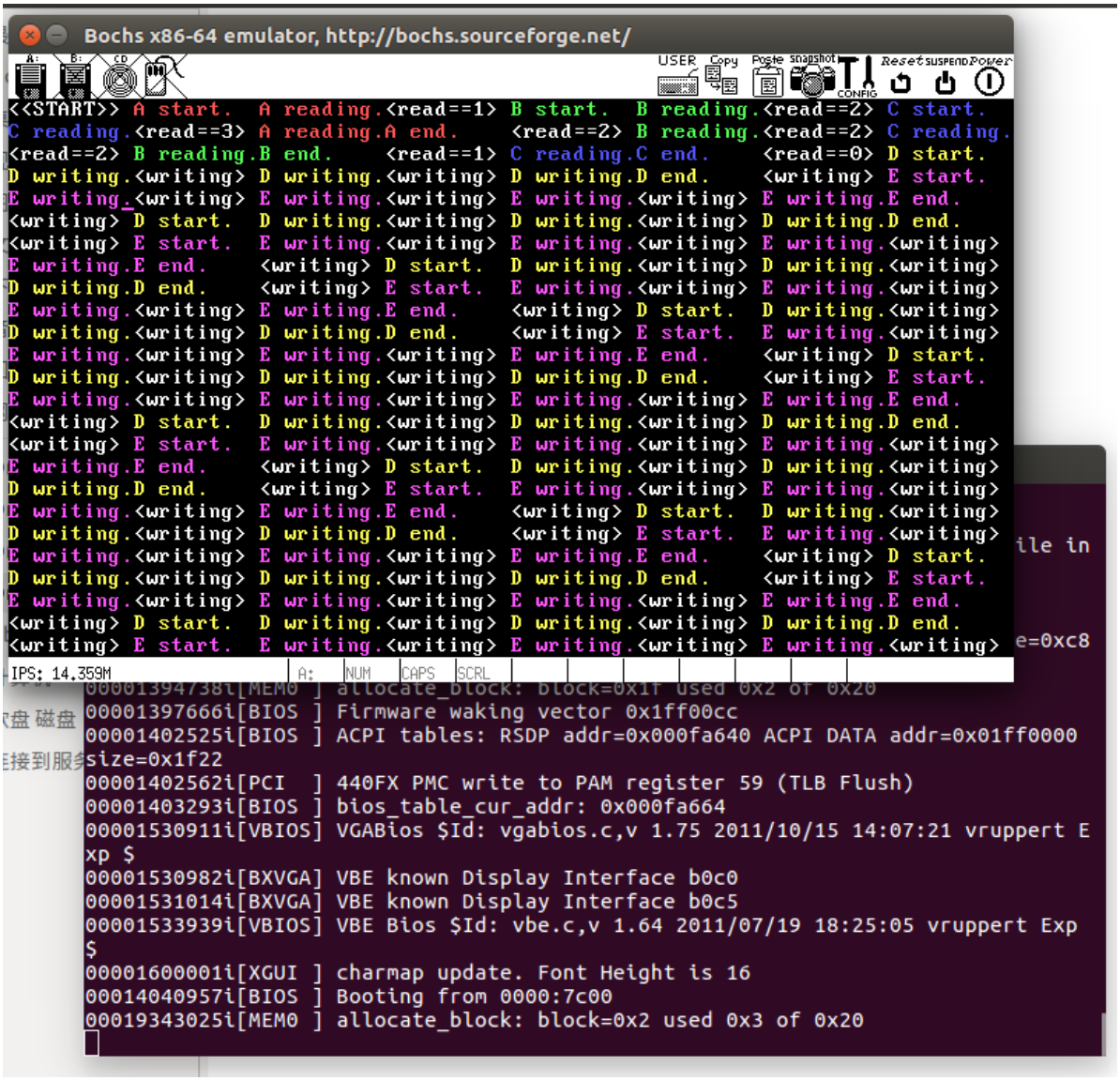
```
while (1)
        {
            P(&writeCountMutex);
            //同时只允许一个写进程改变写进程数目
            if(writeCount==0){
                P(&readPermission);
                //夺取读的权限，或者等待读完
            }
            writeCount++;
            V(&writeCountMutex);
            P(&writeMutex);
            my_print(pname);
            my_print(" start.  ");
            int j;
            for (j = 0; j < p_proc_ready->priority; ++j)
            {
                my_print(pname);
                my_print(writeStr);
                if (j == p_proc_ready->priority - 1)
                {
                    my_print(pname);
                    my_print(endStr);
                }
                else
```

```
                {
                    milli_delay(10);
                }
            }
            V(&writeMutex);

            P(&writeCountMutex);
            writeCount--;
            if (writeCount == 0)
            { // 没有写进程了，才释放读进程的权限
                V(&readPermission);
            }
            V(&writeCountMutex);

            p_proc_ready->isDone = solveHunger;
            milli_delay(10);
        }
```

截图




Bochs x86-64 emulator, http://bochs.sourceforge.net/
USER
Copy
Paste
snapshot
CONFIG
Reset
SUSPEND
Power
<START>> A start. A reading.<read==1> B start. B reading.<read==2> A reading.
A end. <read==1> B reading.<read==1> C start. C reading.<read==2> B reading.
B end. <read==1> C reading.<read==1> C reading.C end. <read==0> D start.
D writing.<writing> D writing.<writing> D writing.D end. <writing> E start.
E writing.<writing> E writing.<writing> E writing.<writing> E writing.E end.
<writing> D start. D writing.<writing> D writing.<writing> D writing.D end.
<writing> E start. E writing.<writing> E writing.<writing> E writing.<writing>
E writing.E end. <writing> D start. D writing.<writing> D writing.<writing>
D writing.D end. <writing> E start. E writing.<writing> E writing.<writing>
E writing.<writing> E writing.E end. <writing> D start. D writing.<writing>
D writing.<writing> D writing.D end. <writing> E start. E writing.<writing>
E writing.<writing> E writing.<writing> E writing.E end. <writing> D start.
D writing.<writing> D writing.<writing> D writing.D end. <writing> E start.
E writing.<writing> E writing.<writing> E writing.<writing> E writing.E end.
<writing> D start. D writing.<writing> D writing.<writing> D writing.<writing> D writing.D end.
<writing> E start. E writing.<writing> E writing.<writing> E writing.<writing> E writing.<writing>
E writing.E end. <writing> D start. D writing.<writing> D writing.<writing>
D writing.D end. <writing> E start. E writing.<writing> E writing.<writing>
E writing.<writing> E writing.E end. <writing> D start. D writing.<writing>
D writing.<writing> D writing.<writing> D writing.D end. <writing> E start.
E writing.<writing> E writing.<writing> E writing.<writing> E writing.E end.
<writing> D start. D writing.<writing> D writing.<writing> D writing.<writing> D writing.D end.
<writing> E start. E writing.<writing> E writing.<writing> E writing.<writing> E writing.<writing>
IPS: 11.500M
A:
NUM
CAPS
SCRL
VMX while in
00001392872i[BIOS] MP table addr=0x000fa510 MPC table addr=0x000fa440 size=0xc8
00001394687i[BIOS] SMBIOS table addr=0x000fa520
00001394738i[MEM0] allocate_block: block=0x1f used 0x2 of 0x20
00001397666i[BIOS] Firmware waking vector 0x1ff00cc
00001402525i[BIOS] ACPI tables: RSDP addr=0x000fa640 ACPI DATA addr=0x01ff0000
size=0x1f22
00001402562i[PCI] 440FX PMC write to PAM register 59 (TLB Flush)
00001403293i[BIOS] bios_table_cur_addr: 0x000fa664
00001530911i[VBIOS] VGABios $Id: vgabios.c,v 1.75 2011/10/15 14:07:21 vruppert E
xp $
00001530982i[BXVGA] VBE known Display Interface b0c0
00001531014i[BXVGA] VBE known Display Interface b0c5
00001533939i[VBIOS] VBE Bios $Id: vbe.c,v 1.64 2011/07/19 18:25:05 vruppert Exp
$
00001600001i[XGUI] charmap update. Font Height is 16
00014040957i[BIOS] Booting from 0000:7c00
00019343025i[MEM0] allocate_block: block=0x2 used 0x3 of 0x20
OrangeS0
Ubuntu 1
计算机
软盘 磁盘
连接到服


Bochs x86-64 emulator, http://bochs.sourceforge.net/
USER Copy Paste snapshot CONFIG Reset SUSPEND Power
<<START>> A start. A reading.<read==1> B start. B reading.<read==2> C start.
C reading.<read==3> A reading.A end. <read==2> B reading.<read==2> C reading.
<read==2> B reading.B end. <read==1> C reading.C end. <read==0> D start.
D writing.<writing> D writing.<writing> D writing.D end. <writing> E start.
E writing.<writing> E writing.<writing> E writing.<writing> E writing.E end.
<writing> D start. D writing.<writing> D writing.<writing> D writing.D end.
<writing> E start. E writing.<writing> E writing.<writing> E writing.<writing>
E writing.E end. <writing> D start. D writing.<writing> D writing.<writing>
D writing.D end. <writing> E start. E writing.<writing> E writing.<writing>
E writing.<writing> E writing.E end. <writing> D start. D writing.<writing>
D writing.<writing> D writing.D end. <writing> E start. E writing.<writing>
E writing.<writing> E writing.<writing> E writing.E end. <writing> D start.
D writing.<writing> D writing.<writing> D writing.D end. <writing> E start.
E writing.<writing> E writing.<writing> E writing.<writing> E writing.E end.
<writing> D start. D writing.<writing> D writing.<writing> D writing.D end.
<writing> E start. E writing.<writing> E writing.<writing> E writing.<writing>
E writing.E end. <writing> D start. D writing.<writing> D writing.<writing>
D writing.D end. <writing> E start. E writing.<writing> E writing.<writing>
E writing.<writing> E writing.E end. <writing> D start. D writing.<writing>
D writing.<writing> D writing.D end. <writing> E start. E writing.<writing>
E writing.<writing> E writing.<writing> E writing.E end. <writing> D start.
D writing.<writing> D writing.<writing> D writing.D end. <writing> E start.
E writing.<writing> E writing.<writing> E writing.<writing> E writing.E end.
<writing> D start. D writing.<writing> D writing.<writing> D writing.D end.
<writing> E start. E writing.<writing> E writing.<writing> E writing.<writing>
IPS: 14.359M A: NUM CAPS SCRL
00001394738i[MEM0] allocate_block: block=0x1f used 0x2 of 0x20
00001397666i[BIOS] Firmware waking vector 0x1ff00cc
00001402525i[BIOS] ACPI tables: RSDP addr=0x000fa640 ACPI DATA addr=0x01ff0000
size=0x1f22
00001402562i[PCI] 440FX PMC write to PAM register 59 (TLB Flush)
00001403293i[BIOS] bios_table_cur_addr: 0x000fa664
00001530911i[VBIOS] VGABios $Id: vgabios.c,v 1.75 2011/10/15 14:07:21 vruppert E
xp $
00001530982i[BXVGA] VBE known Display Interface b0c0
00001531014i[BXVGA] VBE known Display Interface b0c5
00001533939i[VBIOS] VBE Bios $Id: vbe.c,v 1.64 2011/07/19 18:25:05 vruppert Exp
$
00001600001i[XGUI] charmap update. Font Height is 16
00014040957i[BIOS] Booting from 0000:7c00
00019343025i[MEM0] allocate_block: block=0x2 used 0x3 of 0x20