OSLab4

> 191220083 蒙芷露 mzl0830@sina.com

TODO:

- main.cpp 完成测试样例
- irqHandle.cpp
  - keyboardHandle: 处理阻塞的情况
  - syscallReadStdIn: 结合类似信号量的硬件，以及keyboard
  - syscallSemInit/syscallSemwait/syscallSemPost/syscallSemDestroy: 信号量相关的操作

Implementation

keyboardHandle：唤醒阻塞的STD_IN设备

```
void keyboardHandle(struct StackFrame *sf) {
    ProcessTable *pt = NULL;
    uint32_t keyCode = getKeyCode();
    if (keyCode == 0) // illegal keyCode
        return;
    //putChar(getChar(keyCode));
    keyBuffer[bufferTail] = keyCode;
    bufferTail=(bufferTail+1)%MAX_KEYBUFFER_SIZE;

    if (dev[STD_IN].value < 0) { // with process blocked
        // TODO: deal with blocked situation
        dev[STD_IN].value++;

        pt = (ProcessTable*)((uint32_t)(dev[STD_IN].pcb.prev)-(uint32_t)&
(((ProcessTable*)0)->blocked));
        dev[STD_IN].pcb.prev = (dev[STD_IN].pcb.prev)->prev;
        (dev[STD_IN].pcb.prev)->next = &(dev[STD_IN].pcb);

        pt->state = STATE_RUNNABLE;
        pt->sleepTime = 0;//no more sleeping
    }

    return;
}
```

syscallReadIn：

dev[STD_IN]==0时 将当前进程阻塞，递减dev[STD_IN].value，同时利用内联汇编将keyBuffer的内容写入scanf传给syscallReadIn的buffer，并且返回读取的所有数据的长度；若小于0 则说明没有dev可以被申请，那么返回-1，说明读取失败

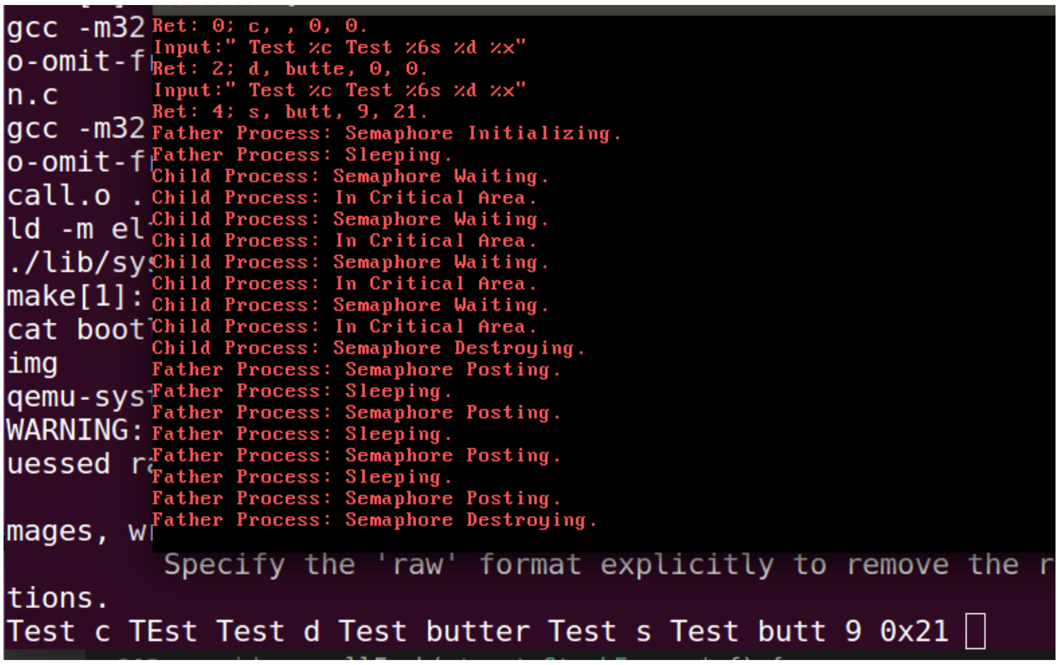
```
void syscallReadStdIn(struct StackFrame *sf) {
    // TODO: complete `stdin`
    if(dev[STD_IN].value==0)
    {
        dev[STD_IN].value--;//设备消耗

        pcb[current].blocked.next = dev[STD_IN].pcb.next;
        pcb[current].blocked.prev = &(dev[STD_IN].pcb);
        dev[STD_IN].pcb.next = &(pcb[current].blocked);//调整指针
        (pcb[current].blocked.next)->prev = &(pcb[current].blocked);

        pcb[current].state = STATE_BLOCKED;
        pcb[current].sleepTime = -1;//在pcb中设置阻塞

        asm volatile("int $0x20");
        //读取缓冲区并且将内容返回到传入的参数中包含的字符指针指向的地址
        int sel = sf->ds;
        char *str = (char*)sf->edx;
        int size = sf->ebx;
        int i=0;
        char ch = 0;
        asm volatile("movw %0,%%es"::"m"(sel));
        while(i<size-1)
        {
            if(bufferHead!=bufferTail)
            {
                ch = getChar(keyBuffer[bufferHead]);
                bufferHead = (bufferHead+1)%MAX_KEYBUFFER_SIZE;
                putChar(ch);
                if(ch!=0)
                {
                    asm volatile("movb %0,%%es:(%1)"::"r"(ch),"r"(str+i));
                    ++i;
                }
            }
            else
                break;
        }
        asm volatile("movb $0x00,%%es:(%0)"::"r"(str+i));
        pcb[current].regs.eax=i;
        return;
    }
    else if(dev[STD_IN].value<0)
    {
        //当前有其他在读 读取失败
        pcb[current].regs.eax=-1;
        return;
    }
}
```

因为不知道scanf需要严格按照格式输入Test这样的辅助字符，花了比较长时间调试



```
gcc -m32 Ret: 0; c, , 0, 0.
o-omit-f Input:" Test %c Test %6s %d %x"
n.c      Ret: 2; d, butte, 0, 0.
         Input:" Test %c Test %6s %d %x"
gcc -m32 Ret: 4; s, butt, 9, 21.
         Father Process: Semaphore Initializing.
o-omit-f Father Process: Sleeping.
         Child Process: Semaphore Waiting.
call.o . Child Process: In Critical Area.
         Child Process: Semaphore Waiting.
ld -m el Child Process: In Critical Area.
./lib/sy Child Process: Semaphore Waiting.
         Child Process: In Critical Area.
make[1]: Child Process: Semaphore Waiting.
cat boot Child Process: In Critical Area.
img      Child Process: Semaphore Destroying.
         Father Process: Semaphore Posting.
qemu-sys Father Process: Sleeping.
         Father Process: Semaphore Posting.
WARNING: Father Process: Sleeping.
uessed r Father Process: Semaphore Posting.
         Father Process: Sleeping.
         Father Process: Semaphore Posting.
mages, w Father Process: Semaphore Destroying.
          Specify the 'raw' format explicitly to remove the r
tions.
Test c TEst Test d Test butter Test s Test butt 9 0x21
```

一系列Sem相关：

同时补充了getPid需要的相关系统调用；并将getpid补充在lib/syscall.cpp中

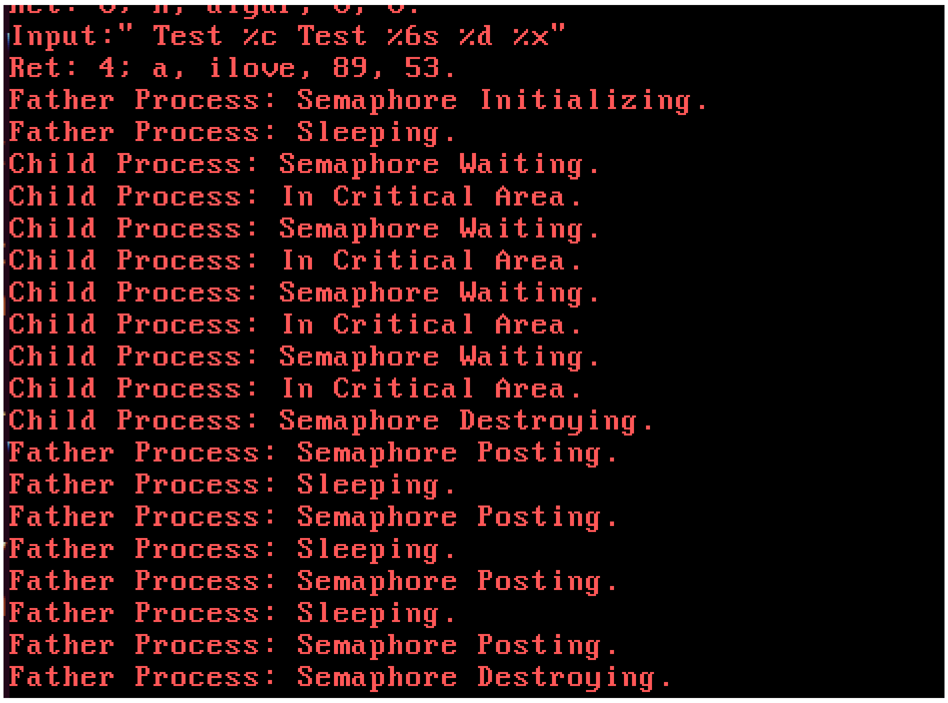
```
void syscallSemInit(struct StackFrame *sf) {
    // TODO: complete `SemInit`
    int i=0;
    for(;i<MAX_SEM_NUM;++i)
    {
        if(sem[i].state==0)//free
            break;
    }
    if(i!=MAX_SEM_NUM)//still have available sem
    {
        sem[i].state=1;
        sem[i].value = (int32_t)sf->edx;
        sem[i].pcb.next = &(sem[i].pcb);
        sem[i].pcb.prev = &(sem[i].pcb);
        pcb[current].regs.eax=i;
    }
    else
        pcb[current].regs.eax=-1;
    return;
}

void syscallSemWait(struct StackFrame *sf) {
    // TODO: complete `SemWait` and note that you need to consider some special
situations
    int i = sf->edx;
    if (sem[i].state == 1) {
        pcb[current].regs.eax = 0;
        sem[i].value--;
        if (sem[i].value < 0) {
            pcb[current].blocked.next = sem[i].pcb.next;
            pcb[current].blocked.prev = &(sem[i].pcb);
            sem[i].pcb.next = &(pcb[current].blocked);
            (pcb[current].blocked.next)->prev = &(pcb[current].blocked);//调整指针
            pcb[current].state = STATE_BLOCKED;
            pcb[current].sleepTime = -1;//在pcb中将进程阻塞
            asm volatile("int $0x20");
        }
    }
    else
        pcb[current].regs.eax = -1;
    return;
}

void syscallSemPost(struct StackFrame *sf) {
    int i = sf->edx;
    ProcessTable *pt = NULL;
    if (i < 0 || i >= MAX_SEM_NUM) {
        pcb[current].regs.eax = -1;
        return;
    }
    // TODO: complete other situations
    if(sem[i].state==1)
    {
        pcb[current].regs.eax=0;
        sem[i].value++;
        if(sem[i].value<=0)
        {
            pt = (ProcessTable*)((uint32_t)(sem[i].pcb.prev) -
                    (uint32_t)&(((ProcessTable*)0)->blocked));
            sem[i].pcb.prev = (sem[i].pcb.prev)->prev;
            (sem[i].pcb.prev)->next = &(sem[i].pcb);//调整指针
            pt->state=STATE_RUNNABLE;
            pt->sleepTime = 0;//在pcb中将进程唤醒
        }
    }
    else
        pcb[current].regs.eax=-1;
    return ;

}

void syscallSemDestroy(struct StackFrame *sf) {
    // TODO: complete `SemDestroy`
    int i=sf->edx;
    if(sem[i].state==1)
    {
        pcb[current].regs.eax=0;
        sem[i].state=0;
        asm volatile("int $0x20");
    }
    else
        pcb[current].regs.eax=-1;
    return;
}
void syscallPid(struct StackFrame *sf)
{
    pcb[current].regs.eax = current;
    return;
}
```

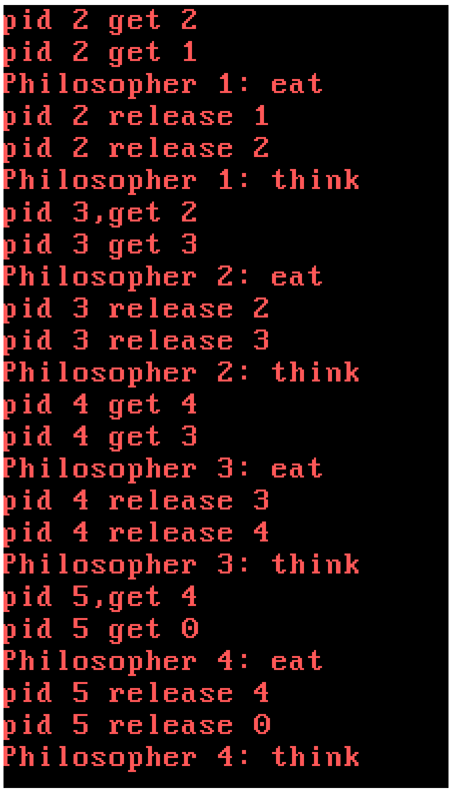


```
Ret: 6, h, digal, 0, 0.
Input:" Test %c Test %6s %d %x"
Ret: 4; a, ilove, 89, 53.
Father Process: Semaphore Initializing.
Father Process: Sleeping.
Child Process: Semaphore Waiting.
Child Process: In Critical Area.
Child Process: Semaphore Waiting.
Child Process: In Critical Area.
Child Process: Semaphore Waiting.
Child Process: In Critical Area.
Child Process: Semaphore Waiting.
Child Process: In Critical Area.
Child Process: Semaphore Destroying.
Father Process: Semaphore Posting.
Father Process: Sleeping.
Father Process: Semaphore Posting.
Father Process: Sleeping.
Father Process: Semaphore Posting.
Father Process: Sleeping.
Father Process: Semaphore Posting.
Father Process: Semaphore Destroying.
```

哲学家问题的设计：

```
    //初始化信号量
    for(int i=0;i<5;++i)
    {
        ret = sem_init(&ph_fork[i],1);
        if(ret==-1)
            printf("fork initial failed\n");
    }

    //创建五个哲学家进程
    for(int i=0;i<2;++i)
        fork();
    if(getpid()==4)
        fork();
    int id = getpid()-1;
    while(1)
    {
        printf("Philosopher %d: think\n", id);
        sleep(128);
        if(id%2==0)
        {
            sem_wait(&ph_fork[id]);
            printf("pid %d,get %d\n",getpid(),id);
            sem_wait(&ph_fork[(id+1)%5]);
            printf("pid %d get %d\n",getpid(),(id+1)%5);
        }
        else
        {
            sem_wait(&ph_fork[(id+1)%5]);
            printf("pid %d get %d\n",getpid(),(id+1)%5);
            sem_wait(&ph_fork[id]);
            printf("pid %d get %d\n",getpid(),id);
        }
        printf("Philosopher %d: eat\n", id);
        sem_post(&ph_fork[id]);
        printf("pid %d release %d\n",getpid(),id);
        sem_post(&ph_fork[(id+1)%5]);
        printf("pid %d release %d\n",getpid(),(id+1)%5);
    }
```



```
pid 2 get 2
pid 2 get 1
Philosopher 1: eat
pid 2 release 1
pid 2 release 2
Philosopher 1: think
pid 3,get 2
pid 3 get 3
Philosopher 2: eat
pid 3 release 2
pid 3 release 3
Philosopher 2: think
pid 4 get 4
pid 4 get 3
Philosopher 3: eat
pid 4 release 3
pid 4 release 4
Philosopher 3: think
pid 5,get 4
pid 5 get 0
Philosopher 4: eat
pid 5 release 4
pid 5 release 0
Philosopher 4: think
```