

# 初识Xinu 实验报告

## part1 观察一个时间片输出字符量

将第一章导论之中的代码加入到Xinu的main主函数之中，可以观察到交替的A和B的输出。为了方便计数，修改了system/clkhandler，令每一次在因为时间片用完而发生调度之前打印一个回车，从而可以清晰地看出来每一个时间片打印的数量。如下图所示：



```
A
BBBBBBB
AAAAAAA
BBB
A
BBB
AAA
B
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
BBBBB
AAAAAAAAAAAAAA
BBBBBBBBBBBB
AAA
BBBB
AAAAAAAAAAAAAA
BBBBBBBBBBBBBB
A
B
A
B
AAAA
BBBBBB
AAAAAAA
BBB
```

可以看到，在一个时间片为2ms的情况下，打印的数量非常不稳定，最少的时候可以打印一个，目前观测到的最多的情况下可以打印四十多个，差异悬殊。初步推测可能同putc的输出缓冲区有关系。

## part2 观察创建新进程导致的抢占

按照Xinu的设计方法，每当创建一个新进程（create+resume）时，新的进程按照优先级被插入到readylist之中，并且在相同优先级的进程之中被插入到最后的位置。由于改变了全局进程状态，所以必然会调用resched进行重新调度。如果这个时候当前进程的时间片尚未用完，在调度的时候，当前进程优先级并不严格大于readylist的队首，就会导致当前进程失去CPU，被队首进程抢占，即便队首进程和当前进程的优先级是相同的。

首先，按照实验要求，在include/kernel.h之中修改QUANTUM的宏定义，将时间片长度修改为10ms，与此同时，为了便于观察，修改sndA和sndB函数，限制总的打印个数是200个，取消无限循环。为了更好地观察抢占现象，我在Xinu代码的两个位置做了修改。首先是在system/resume.c之中，在调用ready函数进而调用resched之前，打印输出“resume”。

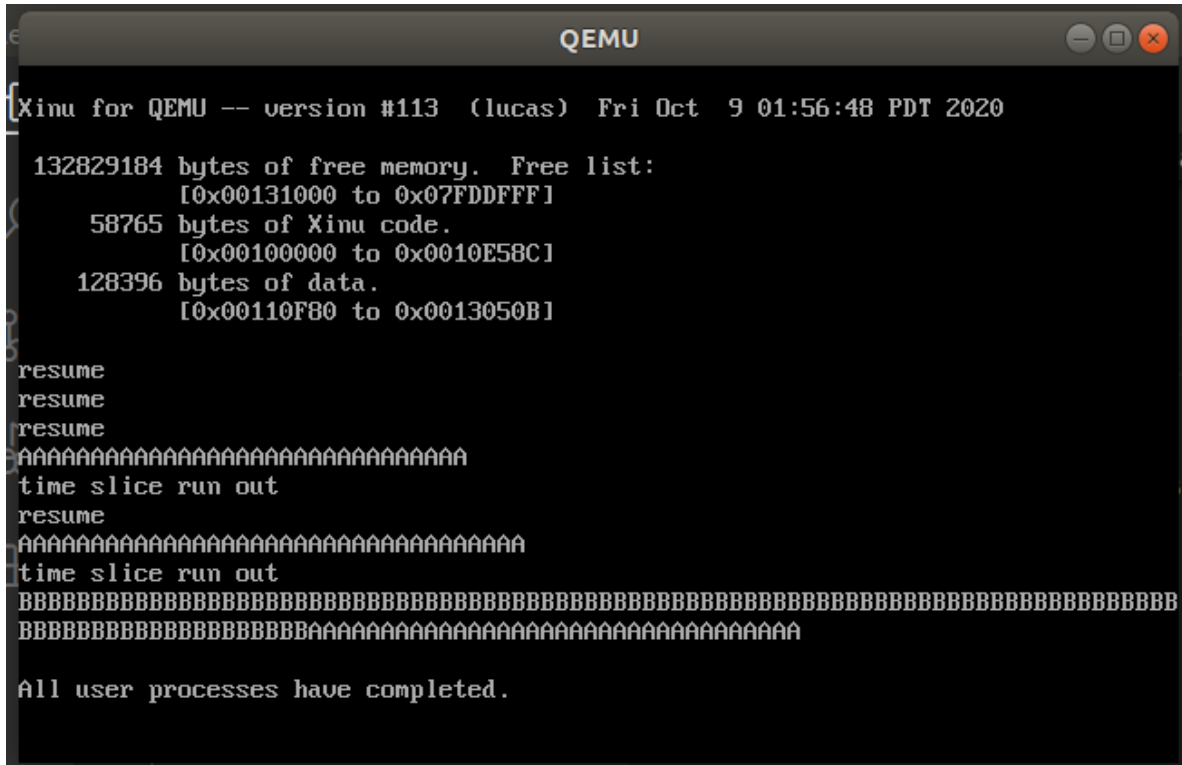
```
prio = prptr->prprio;
kprintf("resume\n");
ready(pid)
```

同时在由于时间片用尽而导致进程切换时，也在system/clkhandler.c打印类似提示信息：

```

if(--preempt) <= 0) {
    preempt = QUANTUM;
    printf("\ntime slice run out\n");
    resched();
}

```



```

Xinu for QEMU -- version #113 (lucas) Fri Oct 9 01:56:48 PDT 2020

132829184 bytes of free memory. Free list:
    [0x00131000 to 0x07FDDFFF]
58765 bytes of Xinu code.
    [0x00100000 to 0x0010E58C]
128396 bytes of data.
    [0x00110F80 to 0x0013050B]

resume
resume
resume
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
time slice run out
resume
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
time slice run out
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

All user processes have completed.

```

如图所示，首先输出开始的两个resume，一个是start up初始化函数的，一个是main主函数的，当第三个resume 打印出来之后，立即开始打印A而没有出现“time slice run out”字样，说明main在创建了sndA之后，还没有等到一个时间片结束，与main具有相同优先级的sndA就抢占了main，直到sndA的时间片用尽。

之后，main获得CPU的控制权，最后一个resume代表了main创建了sndB，这时候main是当前进程，readylist之中队首是sndA，sndB与sndA具有相同的优先级而排在sndA后面，这时没有打印“time slice run out”字样而直接开始打印sndA，说明还没有到一个完整的时间片时sndA抢占了main。

## part3 设法消除不合理的抢占

在part2的基础之上，为了消除不合理的抢占，使得当前进程和新进程在优先级相同并且时间片没有用完的情况下不会被切换出去，在system/ready.c中进行如下修改：

```

prptr = &proctab[pid];
prptr->prstate = PR_READY;
insert(pid, readylist, prptr->prprio);
//add a condition
if(proctab[currpid].prprio!=firstkey(readylist))
    resched();

```

也就是说，ready在就绪链表之中插入一个新进程之后调用resched的目的，只是为了保证当前进程的优先级是最高的（之一），就绪链表之中没有优先级更高的任务了。所以，如果当前任务的优先级同readylist之中的最高优先级（优先级队列，就是第一个）相等，就没有必要调度了。这样做，就从根本上解决了时间片还没有到，CPU就被同等优先级的任务抢占了这一个问题。

在增加了上述语句之后，在此运行Xinu，得到的运行效果如下图所示：

```
QEMU

Xinu for QEMU -- version #114 (lucas) Fri Oct 9 02:04:04 PDT 2020

132829184 bytes of free memory. Free list:
    [0x00131000 to 0x07FDDFFF]
58829 bytes of Xinu code.
    [0x00100000 to 0x0010E5CC]
128396 bytes of data.
    [0x00110F80 to 0x0013050B]

resume
resume
resume
resume
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB

All user processes have completed.
```

可以看到，首先两个resume是start up和main的，然后main创立了sndA和sndB两个进程。两个resume之间没有打印A或者是B，这代表着同优先级的新进程加入到readylist之后没有发生不合理的抢占情况。四个resume结束之后，main执行完毕退出，然后sndA在一个时间片结束之前执行完毕退出了，最后sndB单独执行。

为了观察这种情况下的时间片轮换，特地在打印A和B的时候加入一层无用循环令其进行空转，此时修改之后的sndA函数和sndB函数如下图所示：

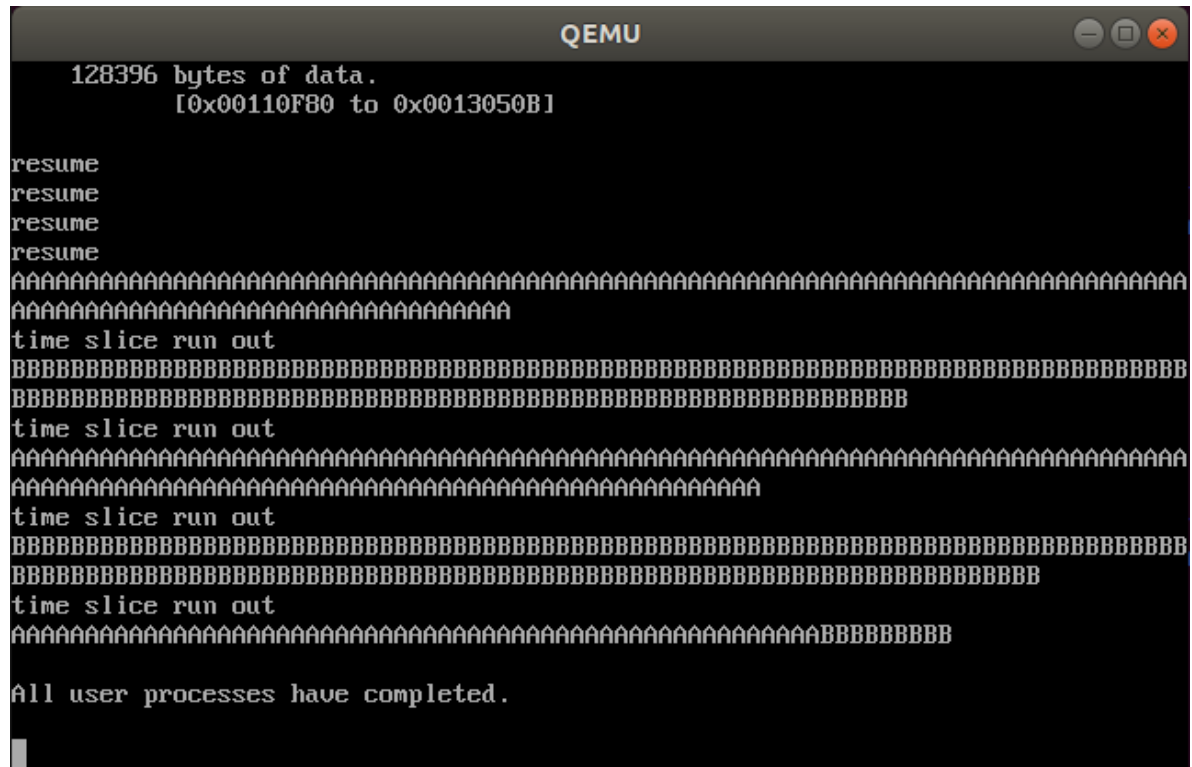
```
void sndA(void){
    int cnta=0;
    while (1)
    {
        int i;
        for (i = 0; i < 6000;i=i+1)
            ;
        putc(CONSOLE, 'A');
        //printf(" %d\n", i);
        cnta++;
        if(cnta==PRINTCNT)
            break;
    }
}

void sndB(void){
    int cntb=0;
    while(1){
        int i;
        for (i = 0; i < 6000;i=i+1)
            ;
        putc(CONSOLE, 'B');
        //printf(" %d\n", i);
        cntb++;
        if(cntb==PRINTCNT)
            break;
    }
}
```

除此之外，main函数以及main.c的其余部分展示如下：

```
#define PRINTCNT 300
pid32 apid;
pid32 bpid;
process main(void)
{
    resume(apid = create(sndA, 1024, 20, "process 1", 0));
    resume(bpid=create(sndB, 1024, 20, "process 2", 0));
}
```

得到的效果如下图所示：



```
128396 bytes of data.
[0x00110F80 to 0x0013050B]

resume
resume
resume
resume
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
time slice run out
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
time slice run out
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
time slice run out
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
BBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
time slice run out
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABBBBBBBB

All user processes have completed.
```

代码和实验报告将上传至obe平台和github帐号 <https://github.com/Lucasftc>