

0.

首先将LAB1-5的代码填充，然后grep查找LAB6

```
./kern/process/proc.c: //LAB6 YOUR CODE : (update LAB5 steps)
./kern/process/proc.c: * below fields(add in LAB6) in proc_struct need to be initialized
./kern/process/proc.c: *   skew_heap_entry_t lab6_run_pool;           // FOR LAB6 ONLY: the entry in the run pool
./kern/process/proc.c: *   uint32_t lab6_stride;                         // FOR LAB6 ONLY: the current stride of the process
./kern/process/proc.c: *   uint32_t lab6_priority;                         // FOR LAB6 ONLY: the priority of process, set by lab6_set_priority(uint32_t)
./kern/process/proc.c: //FOR LAB6, set the process's priority (bigger value will get more CPU time)
./kern/process/proc.h: skew_heap_entry_t lab6_run_pool;           // FOR LAB6 ONLY: the entry in the run pool
./kern/process/proc.h: uint32_t lab6_stride;                     // FOR LAB6 ONLY: the current stride of the process
./kern/process/proc.h: uint32_t lab6_priority;                   // FOR LAB6 ONLY: the priority of process, set by lab6_set_priority(uint32_t)
./kern/process/proc.h: //FOR LAB6, set the process's priority (bigger value will get more CPU time)
./kern/schedule/default_sched_stride.c: /* LAB6: YOUR CODE */
./kern/schedule/default_sched_stride.c: /* LAB6: YOUR CODE */
./kern/schedule/default_sched_stride.c: /* LAB6: YOUR CODE */
./kern/schedule/default_sched_stride.c: /* LAB6: YOUR CODE */
./kern/schedule/default_sched_stride.c: /* LAB6: YOUR CODE */
./kern/schedule/sched.h: // For LAB6 ONLY
./kern/trap/trap.c: /* LAB6 YOUR CODE */
./libs/unistd.h: /* ONLY FOR LAB6 */
./user/libs/syscall.h: /* FOR LAB6 ONLY */
```

首先是./kern/trap/trap.c:

```
/* LAB6 YOUR CODE */
/* IMPORTANT FUNCTIONS:
 * run_timer_list
 *
 * you should update your lab5 code (just add ONE or TWO lines of code):
 * Every tick, you should update the system time, iterate the timers, and trigger the timer
 * You can use one functions to finish all these things.
 */
break;
```

这个很显然，直接按要求调用run\_timer\_list即可。

然后是kern/process/proc.c:

```
//LAB6 YOUR CODE : (update LAB5 steps)
/*
 * below fields(add in LAB6) in proc_struct need to be initialized
 *   struct run_queue *rq;           // running queue contains Process
 *   list_entry_t run_link;          // the entry linked in run queue
 *   int time_slice;                 // time slice for occupying the CPU
 *   skew_heap_entry_t lab6_run_pool; // FOR LAB6 ONLY: the entry in the run pool
 *   uint32_t lab6_stride;            // FOR LAB6 ONLY: the current stride of the pro
 *   uint32_t lab6_priority;          // FOR LAB6 ONLY: the priority of process, set
 */
```

可以发现是要我们为这次lab增加的几个proc的字段初始化，大致看一下可发现，基本都还是初始化为0，那么由于之前是memset的，所以不用改，但是其中struct run\_queue \*rq例外，我们尝试认为其初始化为NULL，make qemu可得：

```
use SLOB allocator
kmalloc_init() succeeded!
check_vma_struct() succeeded!
page fault at 0x00000100: K/W [no page found].
check_pgfault() succeeded!
check_vmm() succeeded.
sched_class: RR_scheduler
kernel panic at kern/schedule/default_sched.c:15:
  assertion failed: list_empty(&(proc->run_link))
Welcome to the kernel debug monitor!!
Type 'help' for a list of commands.
K> 
```

也就是说，在default\_sched.c的RR\_enqueue中，强制要求proc->run\_link是一个空链表，那么我们最开始初始化的时候，也就必须按要求对proc->run\_link做个list\_init，将其初始化为空链表。同时，我们会发现，为了使得一个proc被enqueue后又dequeue出来之后proc->run\_link仍为空链表，程序在dequeue的时候是调用的list\_del\_init，而非直接调用的list\_del：

```

/* *
 * list_del_init - deletes entry from list and reinitialize it.
 * @listelm: the element to delete from the list.
 *
 * Note: list_empty() on @listelm returns true after this.
 * */
static inline void
list_del_init(list_entry_t *listelm) {
    list_del(listelm);
    list_init(listelm);
}

```

至此，对前面lab的修改已经完成，make grade可以发现只有priority的检查失败了：

```

parallels@ubuntu: ~/Mac/ucore_lab/labcodes/lab6
File Edit View Search Terminal Help
- check result: OK
- check output: OK
softint: (2.0s)
- check result: OK
- check output: OK
faultread: (2.2s)
- check result: OK
- check output: OK
faultreadkernel: (2.0s)
- check result: OK
- check output: OK
hello: (2.0s)
- check result: OK
- check output: OK
testbss: (2.1s)
- check result: OK
- check output: OK
pgdir: (2.1s)
- check result: OK
- check output: OK
yield: (2.3s)
- check result: OK
- check output: OK
badarg: (2.0s)
- check result: OK
- check output: OK
exit: (2.4s)
- check result: OK
- check output: OK
spin: (2.6s)
- check result: OK
- check output: OK
waitkill: (4.3s)
- check result: OK
- check output: OK
forktest: (2.1s)
- check result: OK
- check output: OK
forktree: (2.1s)
- check result: OK
- check output: OK
matrix: (14.9s)
- check result: OK
- check output: OK
priority: (22.1s)
- check result: WRONG
!! error: missing 'sched class: stride_scheduler'
!! error: missing 'stride sched correct result: 1 2 3 4 5'
- check output: OK
Total Score: 163/170
make: *** [grade] Error 1
→ lab6 gtt:(nine) X

```

1.

在kern\_init中，调用了sched\_init对调度框架进行了初始化，采取了默认的RR调度算法。

```
/* LAB6 YOUR CODE */
/* IMPORTANT FUNCTIONS:
 * run_timer_list
 *-----
 * you should update your lab5 code (just add ONE or TWO lines of code):
 * Every tick, you should update the system time, iterate the timers, and trigger the timer
 * You can use one functions to finish all these things.
 */
break;
```

syscall中新增了2个系统调用，SYS\_gettime用于获取系统tick时间，SYS\_lab6\_set\_priority用于设置当前进程的优先级。

trap中在每次时间中断的时候，都会调用run\_timer\_list。

改动最大的是sched，首先是增加了调度框架的enqueue、dequeue、pick\_next和proc\_tick，同时增加了sched\_init用于初始化调度框架和算法。然后wakeup\_proc进行了修改，在成功唤醒进程之后，会将进程加入调度使用的运行队列中。schedule也是进行了较大的修改，之前是不停循环查找runnable进程，现在是每次使用调度算法选出下一个需要执行的进程，如果没找到则选择idle进程，然后执行之。

最后是sched中加入了3个关于timer的函数，add\_timer、del\_timer和run\_timer\_list，所有的timer被连成一个链表，然后按照expire time从小到大排序，同时记录相邻两个节点间的expire time时间差，通过这样的方式，使得del\_timer和run\_timer\_list的时间复杂度都为O(1)，add\_timer的时间复杂度为O(n)；然后run\_timer\_list中每次在完成对所有的需要的进程的wakeup之后，都会调用一下调度框架的proc\_tick来实现调度。

2.

首先是决定BIG\_STRIDE，因为stride采取的是unsigned int，故而为了保证判断的正确性，

BIG\_STRIDE最大为max signed int，即0x7fffffff。

然后剩下的几个函数，按照注释一个个的填上即可，需要注意的是：

要看清楚变量类型，分清楚哪里是指针，哪里不是；

skew\_heap\_insert和skew\_heap\_remove是有返回值的，不是像list\_add\_before和list\_del\_init一样使用；

在stride\_dequeue里面，要记得把rq的proc\_num减下来（注释里没说），保证计数正确，虽说好像在lab6中没什么影响；

在stride\_pick\_next里，需要注意处理没找到的情况。

完成后运行：

```
++ setup timer interrupts
trapframe at 0xc012cf10
edi 0x00000001
esi 0x00000000
ebp 0xc012cf68
oesp 0xc012cf30
ebx 0x00000000
edx 0x00000000
ecx 0x00000000
eax 0x7fffffff
ds 0x---0010
es 0x---0010
fs 0x---0023
gs 0x---0023
trap 0x00000000 Divide error
err 0x00000000
eip 0xc010b003
cs 0x---0008
flag 0x00000082 SF,IOPL=0
unhandled trap.
kernel panic at kern/process/proc.c:463:
idleproc exit.

Welcome to the kernel debug monitor!!
Type 'help' for a list of commands.
k> 
```

```
/* you should update your lab5
 * Every tick, you should up
 * You can use one functions
 */
break;
case IRQ_OFFSET + IRQ_COM1:
c = cons_getc();
cprintf("serial [%03d] %c\n", c, c);
break;
case IRQ_OFFSET + IRQ_KBD:
c = cons_getc();
cprintf("kbd [%03d] %c\n", c, c);
break;
```



很明显，发生了除0错误，那么这肯定是由于某个进程的优先级没有设置导致的，为了避免这个问题，我们修改alloc\_proc中的初始化过程，将优先级默认初始化为1，此时再运行：

```
++ setup timer interrupts
kernel_execve: pid = 2, name = "priority".
main: fork ok,now need to wait pids.
child pid 6, acc 1524000, time 2002
child pid 4, acc 784000, time 2003
child pid 7, acc 1220000, time 2004
child pid 5, acc 992000, time 2006
child pid 3, acc 396000, time 2007
main: pid 3, acc 396000, time 2007
main: pid 4, acc 784000, time 2007
main: pid 5, acc 992000, time 2008
main: pid 6, acc 1524000, time 2008
main: pid 7, acc 1220000, time 2008
main: wait pids over
stride sched correct result: 1 2 3 4 3
all user-mode processes have quit.
init check memory pass.
kernel panic at kern/process/proc.c:466:
    initproc exit.

Welcome to the kernel debug monitor!!
Type 'help' for a list of commands.
K>
```

查看priority.c，发现其是给定了一段时间，看5个子进程抢占的比例，那么结果应该是1:2:3:4:5，而这里结果是1 2 3 4 3，最后一个错了，甚至于还比第4个要小，我们可以尝试增大priority.c中给定的时间，以使得结果更精确，可是仍旧会发现结果不对。

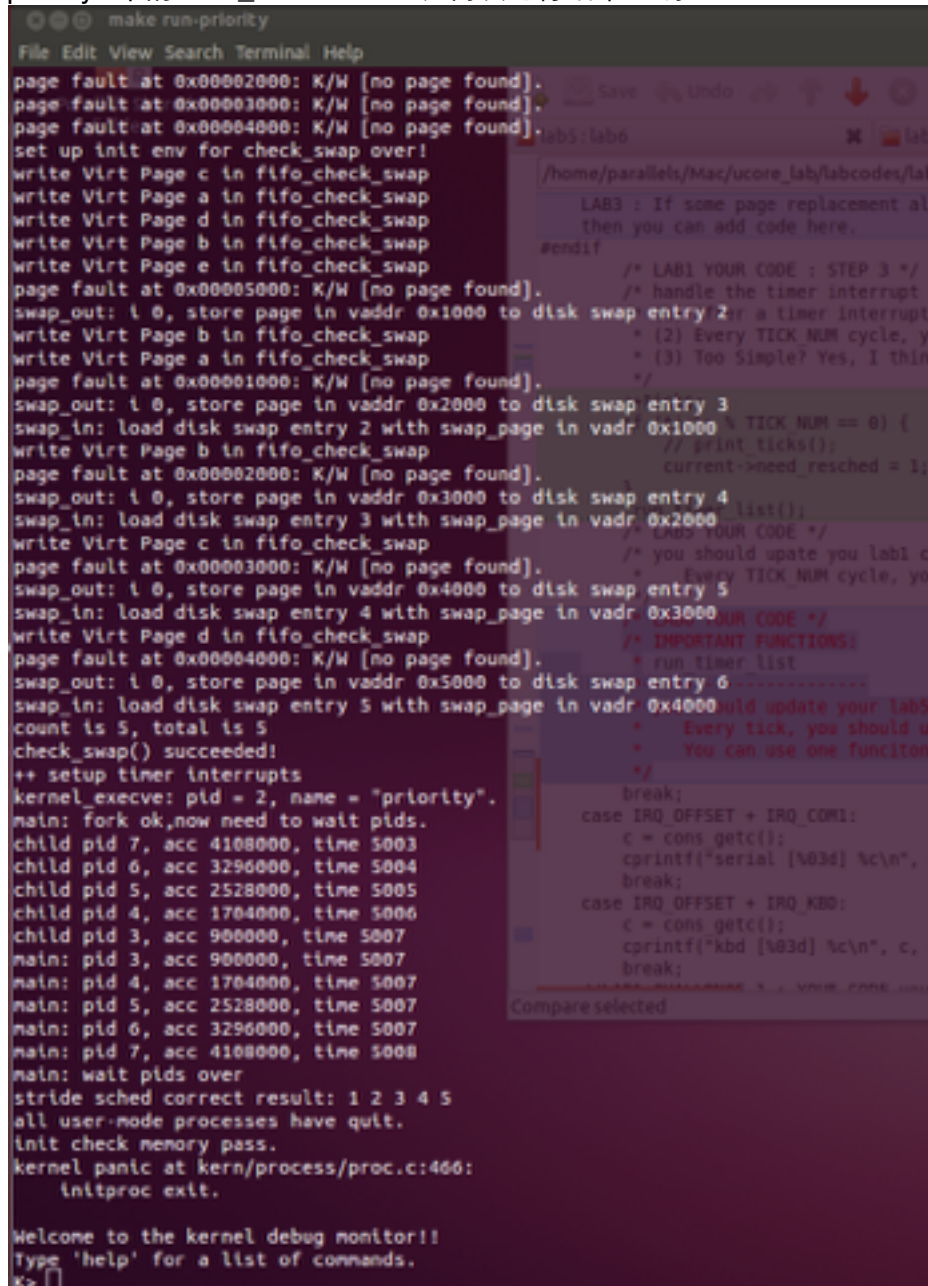
最后，检查程序，我们会发现，在trap.c的时间中断处理中，我们还保留了LAB5留下的处理，每100 ticks的时候，会将让当前进程的need\_resched置为1，放弃CPU使用权，显然，这样是会对调度产生影响的，于是乎我们应该删掉这段。（注释里说好的just add ONE or TWO lines of code果然又是在欺骗我们的感情）

再次运行程序得到：

```
++ setup timer interrupts
kernel_execve: pid = 2, name = "priority".
main: fork ok,now need to wait pids.
child pid 3, acc 396000, time 2001
main: pid 3, acc 396000, time 2001
child pid 7, acc 1576000, time 2003
child pid 6, acc 1296000, time 2003
child pid 5, acc 988000, time 2004
child pid 4, acc 704000, time 2004
main: pid 4, acc 704000, time 2004
main: pid 5, acc 988000, time 2005
main: pid 6, acc 1296000, time 2005
main: pid 7, acc 1576000, time 2005
main: wait pids over
stride sched correct result: 1 2 2 3 4
all user-mode processes have quit.
init check memory pass.
kernel panic at kern/process/proc.c:466:
    initproc exit.

Welcome to the kernel debug monitor!!
Type 'help' for a list of commands.
K>
```

发现还是不对，但是至少acc的趋势很明显是呈现上升趋势的，应该是精度导致的，我们此时增大priority.c中的MAX\_TIME至5000，再次运行结果正确：



```
make run-priority
File Edit View Search Terminal Help
page fault at 0x00002000: K/W [no page found].
page fault at 0x00003000: K/W [no page found].
page fault at 0x00004000: K/W [no page found].
set up init env for check_swap over!
write Virt Page c in fifo_check_swap
write Virt Page a in fifo_check_swap
write Virt Page d in fifo_check_swap
write Virt Page b in fifo_check_swap
write Virt Page e in fifo_check_swap
page fault at 0x00005000: K/W [no page found].
swap_out: i 0, store page in vaddr 0x1000 to disk swap entry 2; a timer interrupt
write Virt Page b in fifo_check_swap
write Virt Page a in fifo_check_swap
page fault at 0x00006000: K/W [no page found].
swap_out: i 0, store page in vaddr 0x2000 to disk swap entry 3
swap_in: load disk swap entry 2 with swap_page in vadr 0x1000
write Virt Page b in fifo_check_swap
page fault at 0x00007000: K/W [no page found].
swap_out: i 0, store page in vaddr 0x3000 to disk swap entry 4
swap_in: load disk swap entry 3 with swap_page in vadr 0x2000
write Virt Page c in fifo_check_swap
page fault at 0x00008000: K/W [no page found].
swap_out: i 0, store page in vaddr 0x4000 to disk swap entry 5
swap_in: load disk swap entry 4 with swap_page in vadr 0x3000
write Virt Page d in fifo_check_swap
page fault at 0x00009000: K/W [no page found].
swap_out: i 0, store page in vaddr 0x5000 to disk swap entry 6
swap_in: load disk swap entry 5 with swap_page in vadr 0x4000
count is 5, total is 5
check_swap() succeeded!
++ setup timer interrupts
kernel_execve: pid = 2, name = "priority".
main: fork ok, now need to wait pids.
child pid 7, acc 4108000, time 5003
child pid 6, acc 3296000, time 5004
child pid 5, acc 2528000, time 5005
child pid 4, acc 1704000, time 5006
child pid 3, acc 900000, time 5007
main: pid 3, acc 900000, time 5007
main: pid 4, acc 1704000, time 5007
main: pid 5, acc 2528000, time 5007
main: pid 6, acc 3296000, time 5007
main: pid 7, acc 4108000, time 5008
main: wait pids over
stride sched correct result: 1 2 3 4 5
all user-mode processes have quit.
init check memory pass.
kernel panic at kern/process/proc.c:466:
initproc exit.

Welcome to the kernel debug monitor!!
Type 'help' for a list of commands.
K>

lab5:lab6
/home/parallels/Mac/ucore_lab/labcodes/lab
LAB3 : If some page replacement al
then you can add code here.
#endif
/* LAB1 YOUR CODE : STEP 3 */
/* handle the timer interrupt.
 * (2) Every TICK_NUM cycle, y
 * (3) Too Simple? Yes, I thin
 */
/* TICK_NUM == 0) {
// print_ticks();
current->need_resched = 1;
list();
/* LAB1 YOUR CODE */
/* you should upate you lab1 c
 * Every TICK_NUM cycle, yo
 * LAB1 YOUR CODE */
/* IMPORTANT FUNCTIONS!
 * run timer list
 * update your lab5
 * Every tick, you should u
 * You can use one functio
 */
break;
case IRQ_OFFSET + IRQ_COM1:
c = cons.getc();
cprintf("serial [%03d] %c\n",
break;
case IRQ_OFFSET + IRQ_KBD:
c = cons.getc();
cprintf("kbd [%03d] %c\n", c,
break;
//***** YOUR CODE *****
Compare selected
```

然后make grade可得：



```
priority: (52.1s)
-check result: OK
-check output: OK
Total Score: 170/170
→ lab6 git:(mine) X
```

extra.

最后实现基于队列的stride。