操作系统课程设计四实验报告

丁勇铭 5122119046

完成情况：

完成了 1-6 的所有功能

实现思路：

首先在 lib 中增加 yoursyscall 库函数：

```
PUBLIC void yoursyscall(arg,deadline)
int arg;
int deadline;
{
  message m;

  m.m1_i1 = arg;
  m.m1_i3 = deadline;
  return(_syscall(PM_PROC_NR, YOURSYSCALL, &m));
}
```

向 pm 发起系统调用，同时将 arg 和 deadline 通过消息传递给 pm

pm 调用 do_yoursyscall 处理此系统调用：

```
PUBLIC void do_yoursyscall()
{
    int deadline;
    clock_t ticks;
    struct mproc *rmp;
    message m;
    rmp = mp;
    m.m1_i1 = m_in.m1_i1;
    m.m1_i2 = rmp->mp_endpoint;
    deadline = m_in.m1_i3;
    m.m1_i3 = deadline;

    if(deadline>0){
  ticks = sys_hz()*deadline;
  set_timer(&rmp->mp_timer,ticks,deadline_coming,rmp->mp_endpoint);
  rmp->mp_flags |= ALARM_ON;
    }
    else{
      printf("deadline error!");
  return;
    }

    if(_taskcall(rmp->mp_scheduler,SCHEDULING_YOURSYSCALL,&m)){
  printf("fail in taskcall.\n");
    }

}
```

判断 deadline 的值，若 deadline 大于 0，则使用 set_timer 定时，时间到后调用 watchdog 函数 deadline_coming：

```
}
PUBLIC void deadline_coming(struct timer *tp)
{
    int proc_nr_n;
    register struct mproc *rmp;

    if(pm_isokendpt(tmr_arg(tp)->ta_int, &proc_nr_n) != OK){
  printf("PM: invaild endpoint from yoursyscall\n");
  return;
    }
    rmp = &mproc[proc_nr_n];
    printf("deadline arrived for pid:%d\n",rmp->mp_pid);
    sig_proc(rmp, SIGKILL, TRUE, FALSE);
}
```

deadline_coming 函数输出此进程的 id，然后发送 SIGKILL 信号杀死进程
do_yoursyscall()同时还将相关信息存入消息中通过任务调用发给 sched 服务器
sched 收到消息后使用 do_yoursyscall 完成任务调用

```
PUBLIC void do_yoursyscall(message *m_ptr)
{
    struct schedproc *rmp;
    int proc_nr_n;

    if(sched_isokendpt(m_ptr->m1_i2,&proc_nr_n)!=OK){
        printf("    SCHED:INVALID ENDPOINT\n");
        return;
    }

    rmp = &schedproc[proc_nr_n];

    rmp->max_priority = rmp->priority = YOUR_Q ;
    rmp->yourprocess = 1;
    rmp->arg = m_ptr->m1_i1;
    rmp->deadline = m_ptr->m1_i3;
    if(schedule_process(rmp)!= OK){
        printf("fail in schedule_process\n");
    }

}
```

将此进程的优先级设为 YOUR_Q 等于 8，因为如果优先级再高的话会和其他系统进程的优先级冲突，比如说捕捉键盘的 tty 进程的优先级为 7。
将 arg 和 deadline 的值传入 scheproc 结构中，并将 yourprocess 的值设为 1，标示该进程为调用了 yoursyscall 的进程。

在 schedproc.h 中修改 schedproc 为其增加 yourprocess、arg、deadline 三个属性

```
EXTERN struct schedproc {
    endpoint_t endpoint;     /* pr
    endpoint_t parent;  /* parent
    unsigned flags;     /* flag b

    /* User space scheduling */
    unsigned max_priority;  /* th
    unsigned priority;      /* th
    unsigned time_slice;

    unsigned yourprocess;
    int arg;
    int deadline;
} schedproc[NR_PROCS];
```

修改 do_noquantum 函数，使其只降低 yourprocess 不等于 1 的进程，保证调用了 yoursyscall 的进程优先级不会降低，同时使用 set_yoursyscall_slice 函数为 yourprocess 等于 1 的进程分配时间片：

```
PUBLIC int do_noquantum(message *m_ptr)
{
    register struct schedproc *rmp;
    int rv, proc_nr_n;

    if (sched_isokendpt(m_ptr->m_source, &proc_nr_n) != OK) {
        printf("SCHED: WARNING: got an invalid endpoint in OOQ msg %u.\n",
        m_ptr->m_source);
        return EBADEPT;
    }

    rmp = &schedproc[proc_nr_n];
    if (rmp->yourprocess!=1 && rmp->priority < MIN_USER_Q) {
        rmp->priority += 1; /* lower priority */
    }
    if(rmp->yourprocess==1){
        rmp->time_slice = set_yoursyscall_slice(rmp->arg,rmp->deadline);
    }
    if ((rv = schedule_process(rmp)) != OK) {
        return rv;
    }
    return OK;
}
```

为 yourprocess 分配时间片的函数如下，可见时间片是一个与 arg 的值成正比同时与 deadline 的值成反比的值，这样就保证了 arg 越大或越靠近 deadline 的进程运行的机会越多：

```
PRIVATE unsigned set_yoursyscall_slice(int arg,int deadline){
    return (unsigned)(arg+20/(deadline+1))*DEFAULT_USER_TIME_SLICE;
}
```

修改 balance_quenes 函数，对于 yourprocess 等于 1 的进程，每次调用 balance_quenes 时为其更新 deadline 的值，而对于 yourprocess 不等于 1 的进程，只有在其优先级-1 之后仍比 8 大时才允许其提升优先级，这样就保证了普通进程的优先级永远不会比调用了 yoursyscall 的进程的优先级高：

```
PRIVATE void balance_queues(struct timer *tp)
    {
    struct schedproc *rmp;
    int proc_nr;
    int rv;

    for (proc_nr=0, rmp=schedproc; proc_nr < NR_PROCS; proc_nr++, rmp++) {
        if(rmp->yourprocess==1){
            rmp->deadline = rmp->deadline - 5;
            if(rmp->deadline<0) rmp->deadline = 0;
        }
        if (rmp->yourprocess!=1 & rmp->flags & IN_USE) {
            if (rmp->priority > YOUR_Q + 1) {
                rmp->priority -= 1; /* increase priority */
                schedule_process(rmp);
            }
        }
    }

    set_timer(&sched_timer, balance_timeout, balance_queues, 0);
}
```

运行结果：

输出 1 的进程不调用 yoursyscall

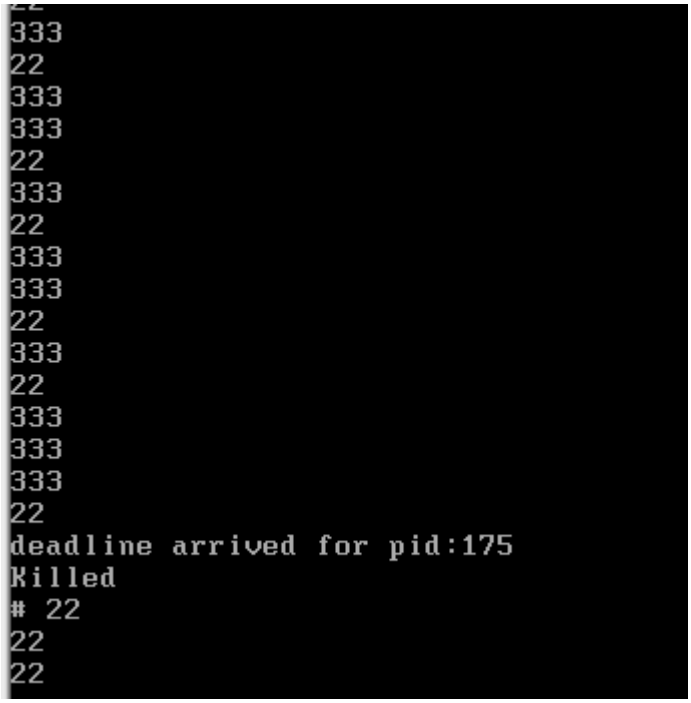
输出 22 的进程调用了 yoursyscall(2, 100)

输出 333 的进程调用了 yoursyscall(2, 30)

在运行初期可以看到 22 和 333 均匀出现而 1 不会出现

```
# ./do.sh
1
22
333
22
333
22
333
22
333
22
333
22
333
22
333
22
333
22
333
22
```

在接近 deadline 后 333 的出现频率增加：



333 被 kill 后只有 22 输出

在 kill 掉 22 后才有 1 输出：