操作系统实验

——作业调度

2016年3月

14级春季

1

作业调度实验报告

夏　梦	**郑棵丹**
14061175	14061147

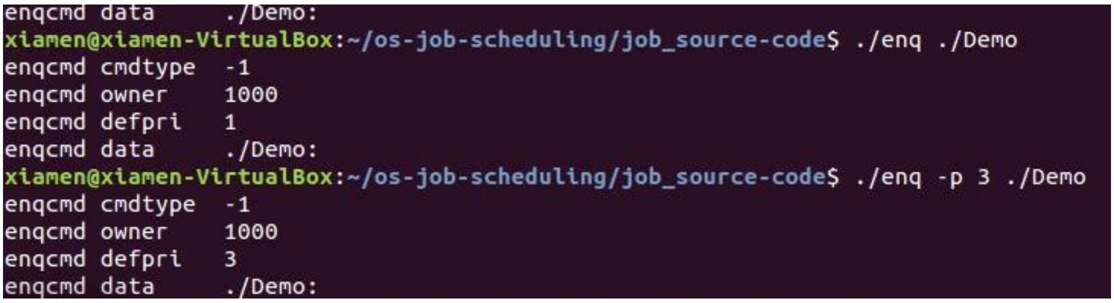
助教：王昕

2016. 4. 16

目录

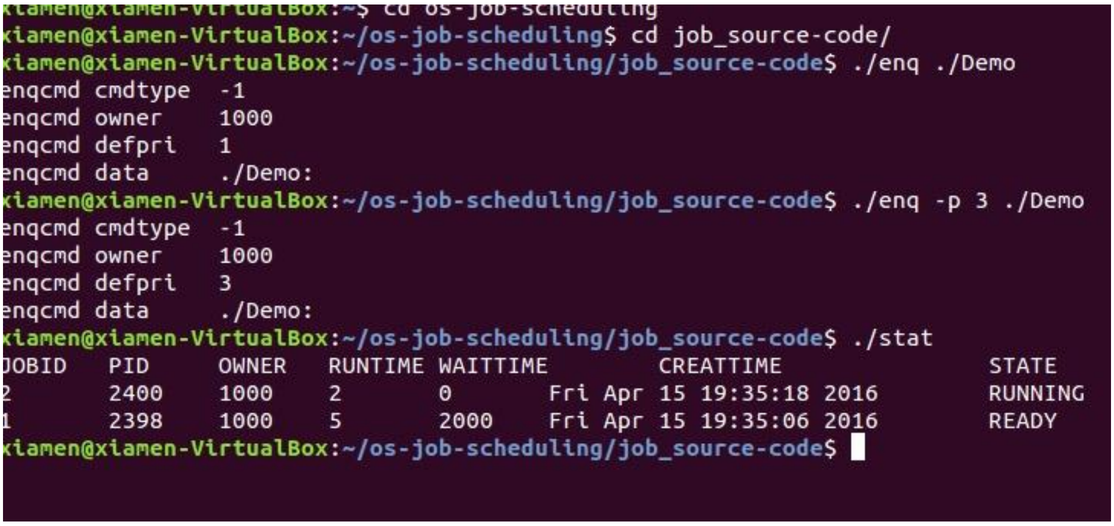
基本要求

1. 实现作业的入队操作：



```
enqcmd data     ./Demo:
xiamen@xiamen-VirtualBox:~/os-job-scheduling/job_source-code$ ./enq ./Demo
enqcmd cmdtype  -1
enqcmd owner    1000
enqcmd defpri   1
enqcmd data     ./Demo:
xiamen@xiamen-VirtualBox:~/os-job-scheduling/job_source-code$ ./enq -p 3 ./Demo
enqcmd cmdtype  -1
enqcmd owner    1000
enqcmd defpri   3
enqcmd data     ./Demo:
```

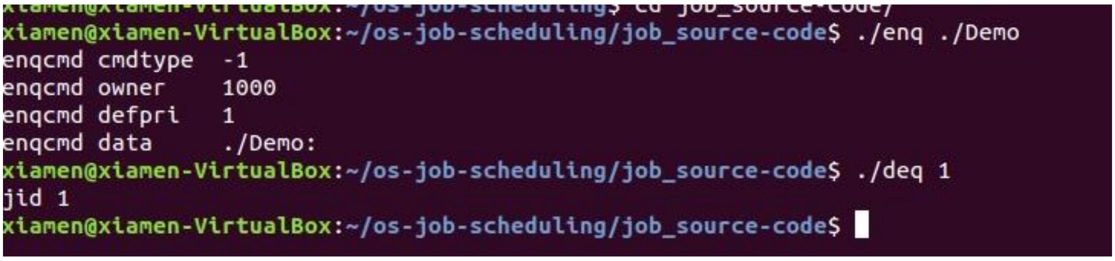
打印队列显示添加成功（注：此输出已完成提高实验内容）：



```
xiamen@xiamen-VirtualBox:~$ cd os-job-scheduling
xiamen@xiamen-VirtualBox:~/os-job-scheduling$ cd job_source-code/
xiamen@xiamen-VirtualBox:~/os-job-scheduling/job_source-code$ ./enq ./Demo
enqcmd cmdtype  -1
enqcmd owner    1000
enqcmd defpri   1
enqcmd data     ./Demo:
xiamen@xiamen-VirtualBox:~/os-job-scheduling/job_source-code$ ./enq -p 3 ./Demo
enqcmd cmdtype  -1
enqcmd owner    1000
enqcmd defpri   3
enqcmd data     ./Demo:
xiamen@xiamen-VirtualBox:~/os-job-scheduling/job_source-code$ ./stat
JOBID   PID     OWNER   RUNTIME WAITTIME        CREATTIME               STATE
2       2400    1000    2       0       Fri Apr 15 19:35:18 2016        RUNNING
1       2398    1000    5       2000    Fri Apr 15 19:35:06 2016        READY
xiamen@xiamen-VirtualBox:~/os-job-scheduling/job_source-code$ █
```

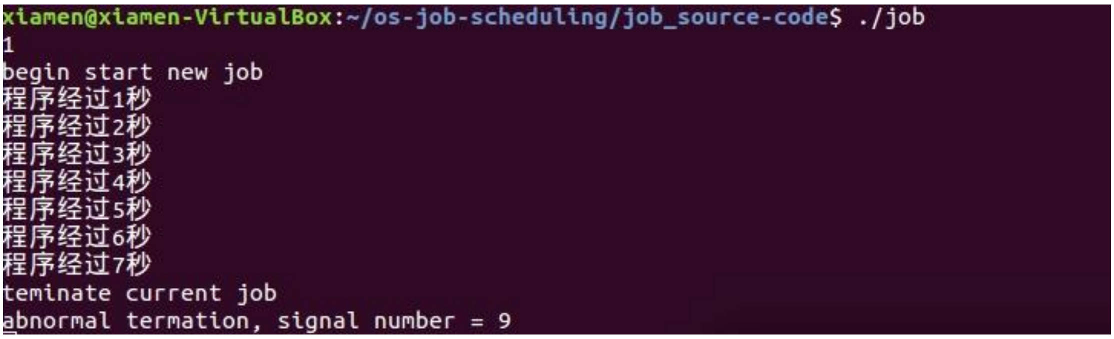
注：第一个作业未给出优先级，所以默认为最低优先级 1，第二个作业给出优先级为 3。

2. 实现作业状态命令查看：如上图，将在后面的提高实验中一并详细描述。
3. 实现出队命令操作：



```
xiamen@xiamen-VirtualBox:~/os-job-scheduling$ cd job_source-code/
xiamen@xiamen-VirtualBox:~/os-job-scheduling/job_source-code$ ./enq ./Demo
enqcmd cmdtype  -1
enqcmd owner    1000
enqcmd defpri   1
enqcmd data     ./Demo:
xiamen@xiamen-VirtualBox:~/os-job-scheduling/job_source-code$ ./deq 1
jid 1
xiamen@xiamen-VirtualBox:~/os-job-scheduling/job_source-code$ █
```

对应 job 中输出为：

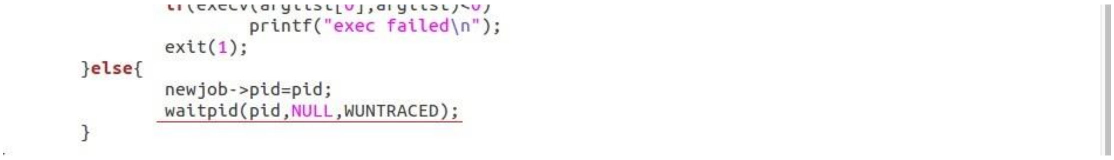


```
xiamen@xiamen-VirtualBox:~/os-job-scheduling/job_source-code$ ./job
1
begin start new job
程序经过1秒
程序经过2秒
程序经过3秒
程序经过4秒
程序经过5秒
程序经过6秒
程序经过7秒
teminate current job
abnormal termation, signal number = 9
```

基本要求

Bug 修复：

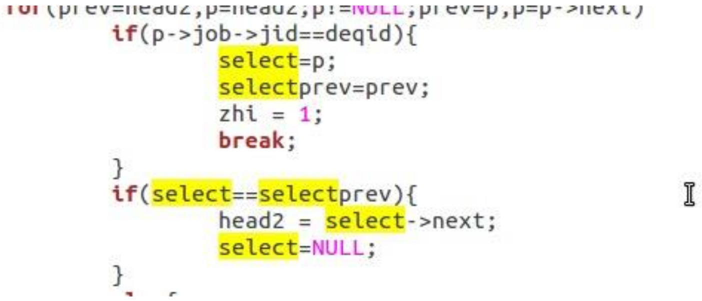
①job.c 的 do_enq 函数内，只入队一个作业时，该作业不能执行。相应修改如下：



```
        if(execv(arglist[0],arglist)<0)
                printf("exec failed\n");
        exit(1);
}else{
        newjob->pid=pid;
        waitpid(pid,NULL,WUNTRACED);
}
```

新增 waitpid 语句，使父进程等到子进程被阻塞之后才继续往下执行，保证了从第一个作业开始就能正常执行。

②～④链表指针丢失，select 仍与等待队列相连。相应修改如下：



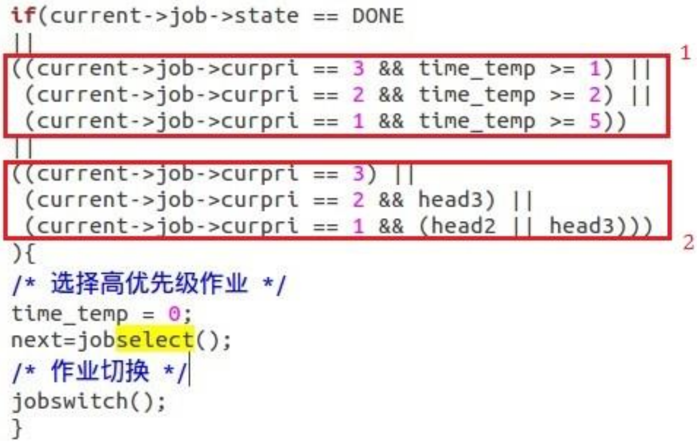
```
for(prev=head2,p=head2;p!=NULL;prev=p,p=p->next)
        if(p->job->jid==deqid){
                select=p;
                selectprev=prev;
                zhi = 1;
                break;
        }
        if(select==selectprev){
                head2 = select->next;
                select=NULL;
        }
```

以上程序段已完成提高实验相应要求，因此每次从合适的优先级队列中取出头指针指向的作业，然后将头指针往后移一位，并将取出的作业独立，脱离队伍。

提高要求

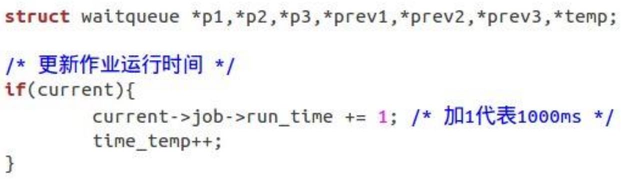
1. 实现多级反馈的轮转调度算法：

①时间片：



```
if(current->job->state == DONE
||
((current->job->curpri == 3 && time_temp >= 1) ||
 (current->job->curpri == 2 && time_temp >= 2) ||
 (current->job->curpri == 1 && time_temp >= 5))
||
((current->job->curpri == 3) ||
 (current->job->curpri == 2 && head3) ||
 (current->job->curpri == 1 && (head2 || head3)))
){
/* 选择高优先级作业 */
time_temp = 0;
next=jobselect();
/* 作业切换 */
jobswitch();
}
```

如上图 1 框所示，优先级从高到低为 3、2、1，因此对应时间片为 1、2、5，若时间片符合要求则选择切换作业。



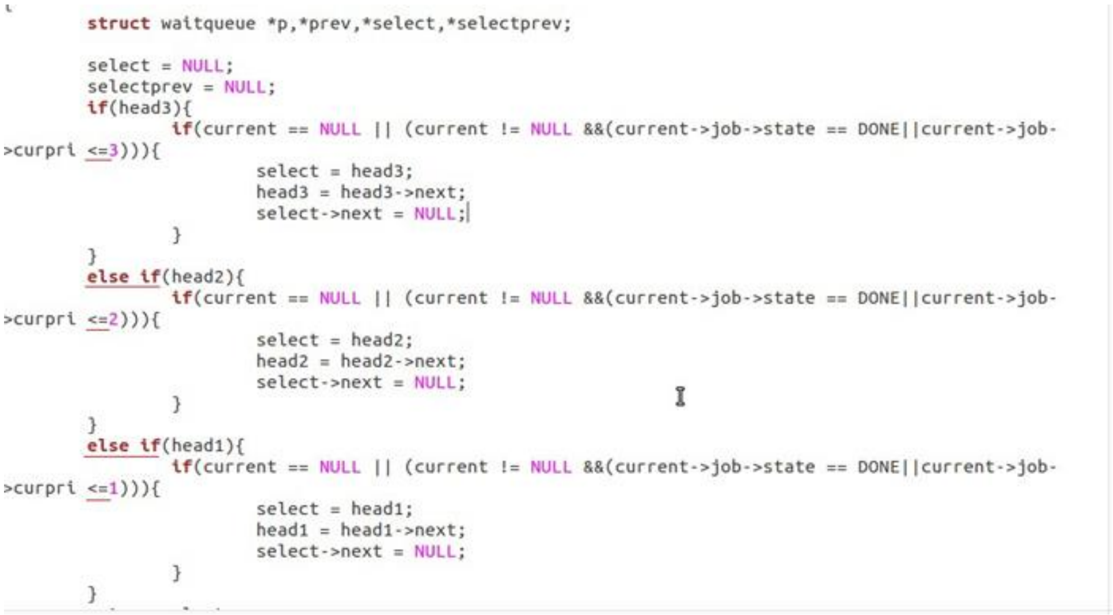
```
struct waitqueue *p1,*p2,*p3,*prev1,*prev2,*prev3,*temp;

/* 更新作业运行时间 */
if(current){
        current->job->run_time += 1; /* 加1代表1000ms */
        time_temp++;
}
```

更新作业运行时间，每一轮增加 1 秒。

②抢占式运行。如上图 2 框所示，若存在高于当前优先级的作业，则不论时间片是否结束，均抢占运行。

③高优先级队列结束前，不轮转低优先级队列，等待时间超过 10s 自动升高优先级一级。

选择：



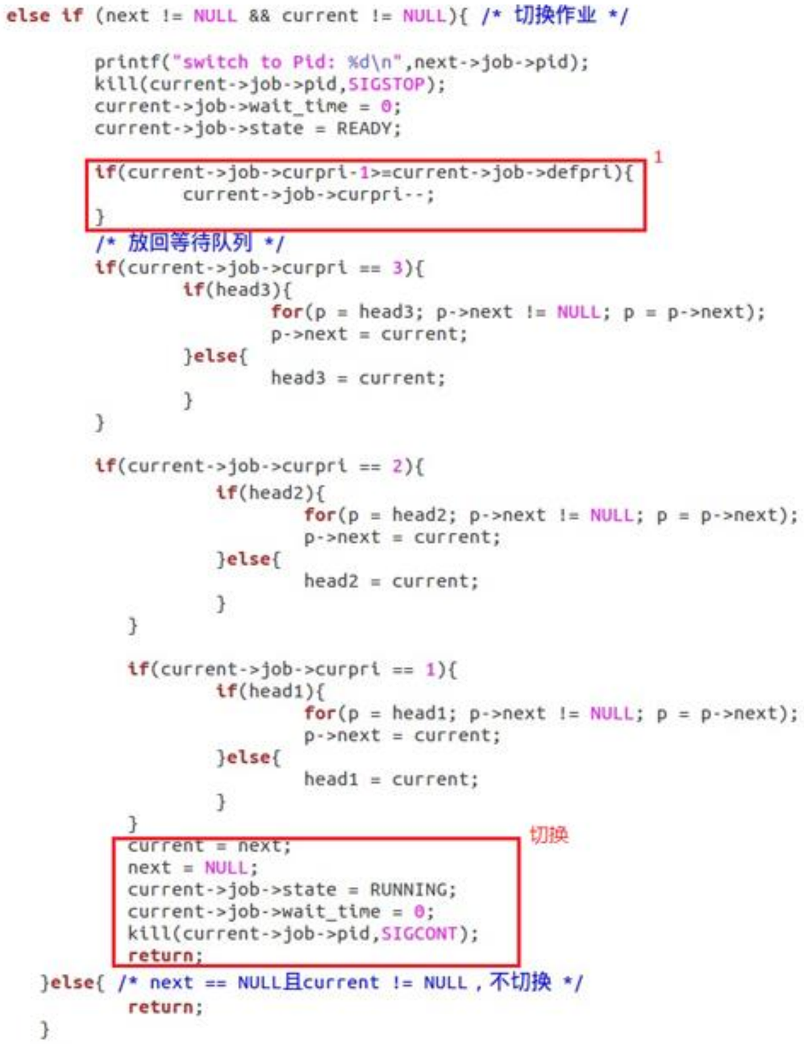
```
        struct waitqueue *p,*prev,*select,*selectprev;

        select = NULL;
        selectprev = NULL;
        if(head3){
                if(current == NULL || (current != NULL &&(current->job->state == DONE||current->job-
>curpri <=3))){
                        select = head3;
                        head3 = head3->next;
                        select->next = NULL;
                }
        }
        else if(head2){
                if(current == NULL || (current != NULL &&(current->job->state == DONE||current->job-
>curpri <=2))){
                        select = head2;
                        head2 = head2->next;
                        select->next = NULL;
                }
        }
        else if(head1){
                if(current == NULL || (current != NULL &&(current->job->state == DONE||current->job-
>curpri <=1))){
                        select = head1;
                        head1 = head1->next;
                        select->next = NULL;
                }
        }
```

Else-if：若高优先级队列不为空，则不论转低优先级队列。

提高要求

<=：轮转队列优先级应大于等于当前优先级。

切换：



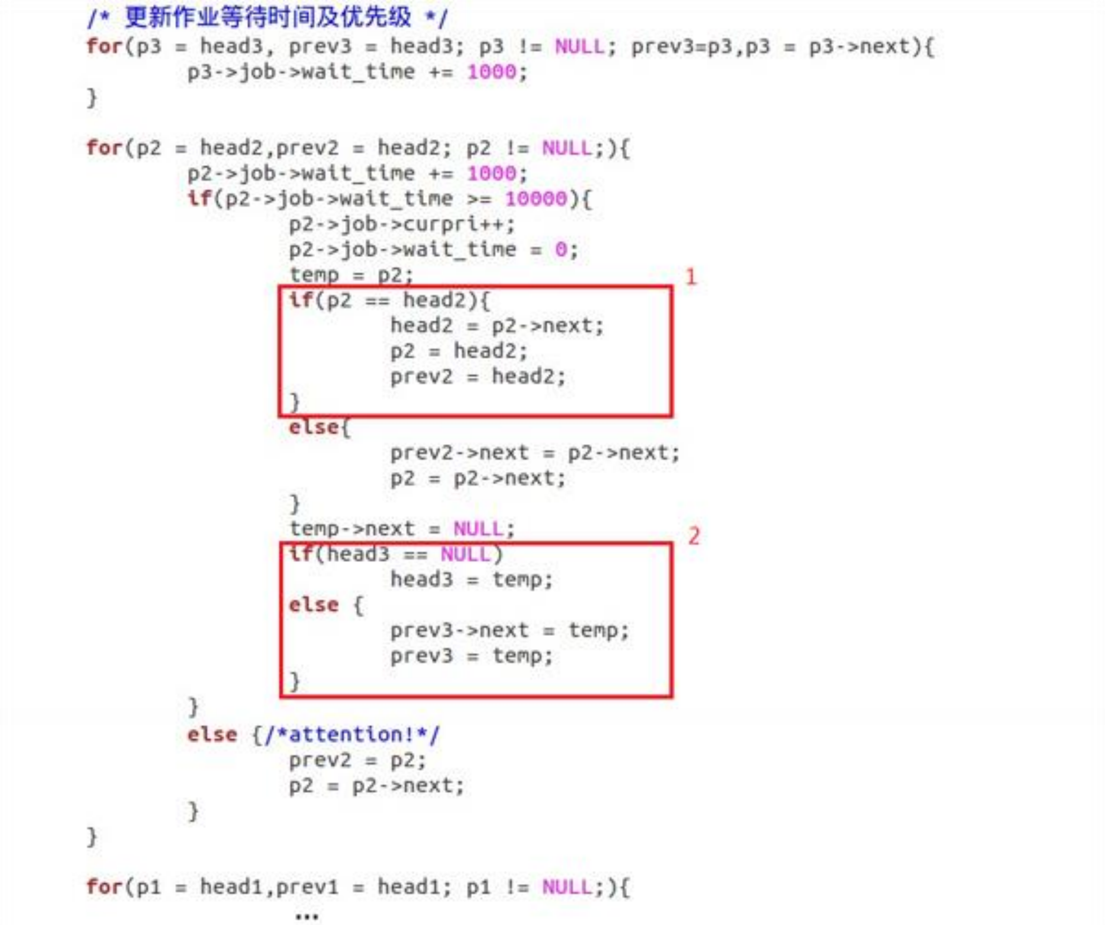
```
else if (next != NULL && current != NULL){ /* 切换作业 */

        printf("switch to Pid: %d\n",next->job->pid);
        kill(current->job->pid,SIGSTOP);
        current->job->wait_time = 0;
        current->job->state = READY;

        if(current->job->curpri-1>=current->job->defpri){
                current->job->curpri--;
        }
        /* 放回等待队列 */
        if(current->job->curpri == 3){
                if(head3){
                        for(p = head3; p->next != NULL; p = p->next);
                        p->next = current;
                }else{
                        head3 = current;
                }
        }

        if(current->job->curpri == 2){
                   if(head2){
                           for(p = head2; p->next != NULL; p = p->next);
                           p->next = current;
                   }else{
                           head2 = current;
                   }
           }

           if(current->job->curpri == 1){
                   if(head1){
                           for(p = head1; p->next != NULL; p = p->next);
                           p->next = current;
                   }else{
                           head1 = current;
                   }
           }
           current = next;
           next = NULL;
           current->job->state = RUNNING;
           current->job->wait_time = 0;
           kill(current->job->pid,SIGCONT);
           return;
   }else{ /* next == NULL且current != NULL，不切换 */
           return;
   }
```

框 1：每一次切换将当前作业优先级降 1，但不能低于其默认优先级。

放回等待队列：若队列不为空，则放到队尾，否则则为队头。

框切换：完成切换内容。

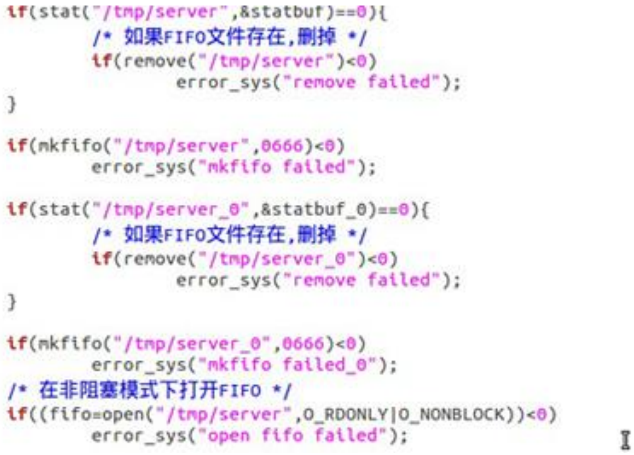
更新作业等待时间及优先级：



```
/* 更新作业等待时间及优先级 */
for(p3 = head3, prev3 = head3; p3 != NULL; prev3=p3,p3 = p3->next){
        p3->job->wait_time += 1000;
}

for(p2 = head2,prev2 = head2; p2 != NULL;){
        p2->job->wait_time += 1000;
        if(p2->job->wait_time >= 10000){
                p2->job->curpri++;
                p2->job->wait_time = 0;
                temp = p2;
                if(p2 == head2){
                        head2 = p2->next;
                        p2 = head2;
                        prev2 = head2;
                }
                else{
                        prev2->next = p2->next;
                        p2 = p2->next;
                }
                temp->next = NULL;
                if(head3 == NULL)
                        head3 = temp;
                else {
                        prev3->next = temp;
                        prev3 = temp;
                }
        }
        else {/*attention!*/
                prev2 = p2;
                p2 = p2->next;
        }
}

for(p1 = head1,prev1 = head1; p1 != NULL;){
                ...
```

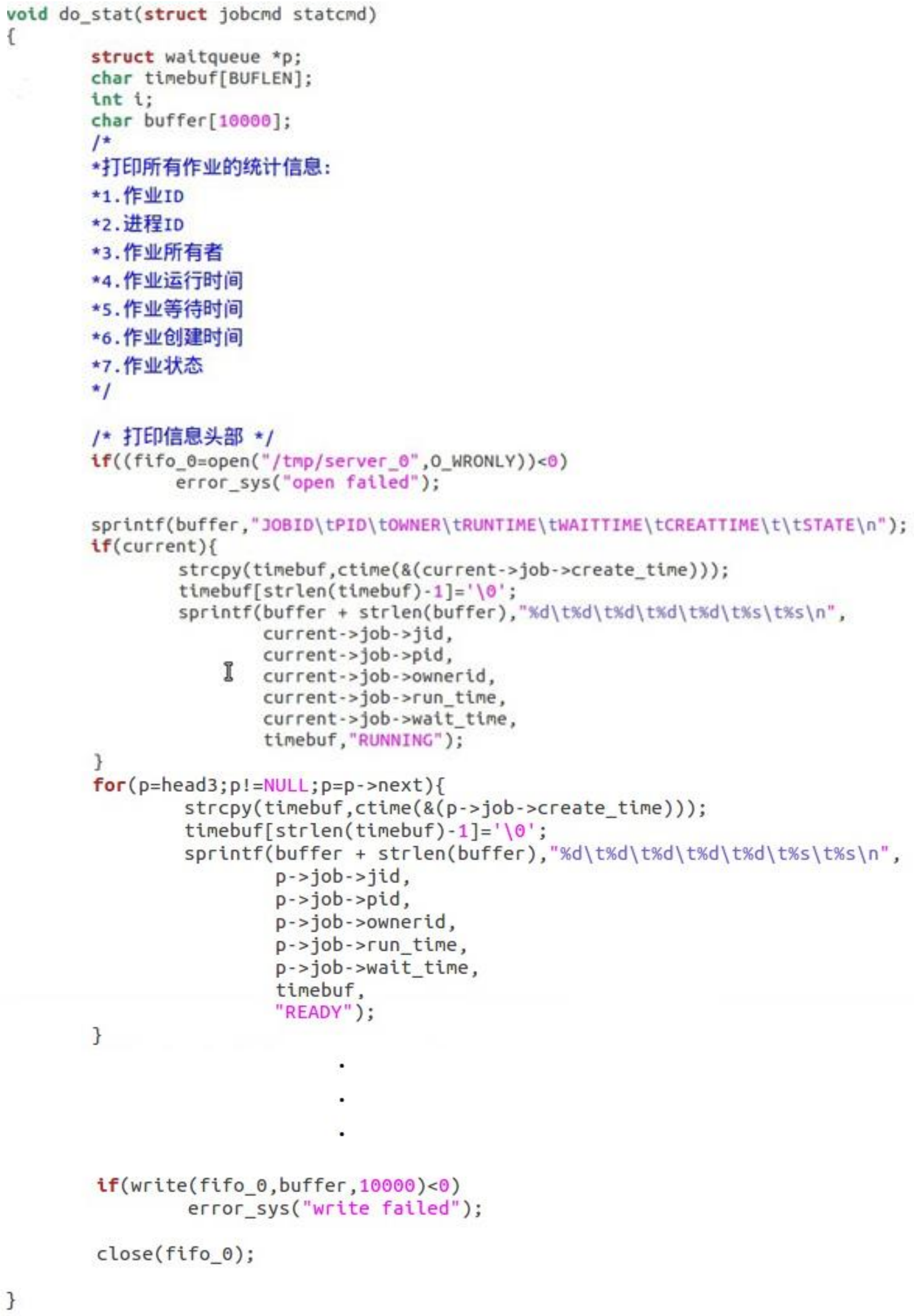
框 1：若当前需要升级优先级的为队头，则将其提出，将队头及循环指针均后移一位。若不是，则仅将循环指针后移。框 2：若升级后队列目前为空，则将该作业放在升级后队头，否则置于队尾且相应队尾指针后移。优先级为 1 的队列处理同 2，因此在报告中省略。

2. STAT：



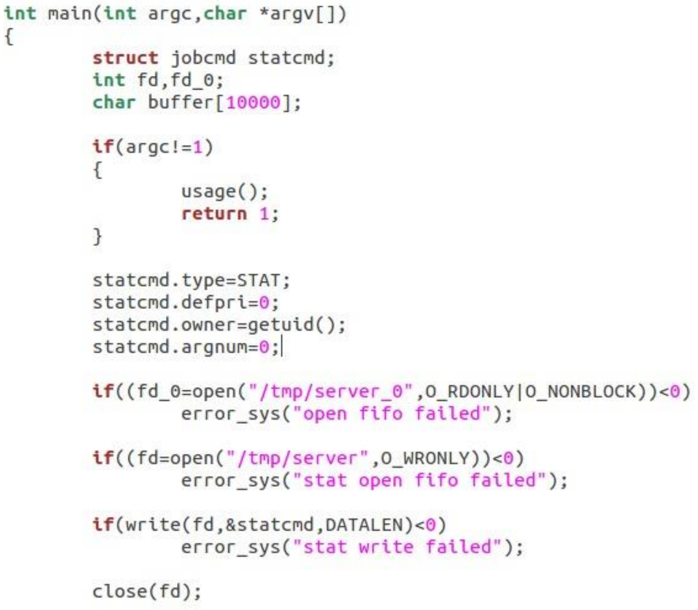
```
if(stat("/tmp/server",&statbuf)==0){
        /* 如果FIFO文件存在,删掉 */
        if(remove("/tmp/server")<0)
                error_sys("remove failed");
}

if(mkfifo("/tmp/server",0666)<0)
        error_sys("mkfifo failed");

if(stat("/tmp/server_0",&statbuf_0)==0){
        /* 如果FIFO文件存在,删掉 */
        if(remove("/tmp/server_0")<0)
                error_sys("remove failed");
}

if(mkfifo("/tmp/server_0",0666)<0)
        error_sys("mkfifo failed_0");
/* 在非阻塞模式下打开FIFO */
if((fifo=open("/tmp/server",O_RDONLY|O_NONBLOCK))<0)
        error_sys("open fifo failed");
```

以相同的方式创建一个属于自己的管道。



```
void do_stat(struct jobcmd statcmd)
{
        struct waitqueue *p;
        char timebuf[BUFLEN];
        int i;
        char buffer[10000];
        /*
        *打印所有作业的统计信息:
        *1.作业ID
        *2.进程ID
        *3.作业所有者
        *4.作业运行时间
        *5.作业等待时间
        *6.作业创建时间
        *7.作业状态
        */

        /* 打印信息头部 */
        if((fifo_0=open("/tmp/server_0",O_WRONLY))<0)
                error_sys("open failed");

        sprintf(buffer,"JOBID\tPID\tOWNER\tRUNTIME\tWAITTIME\tCREATTIME\t\tSTATE\n");
        if(current){
                strcpy(timebuf,ctime(&(current->job->create_time)));
                timebuf[strlen(timebuf)-1]='\0';
                sprintf(buffer + strlen(buffer),"%d\t%d\t%d\t%d\t%d\t%s\t%s\n",
                        current->job->jid,
                        current->job->pid,
                        current->job->ownerid,
                        current->job->run_time,
                        current->job->wait_time,
                        timebuf,"RUNNING");
        }
        for(p=head3;p!=NULL;p=p->next){
                strcpy(timebuf,ctime(&(p->job->create_time)));
                timebuf[strlen(timebuf)-1]='\0';
                sprintf(buffer + strlen(buffer),"%d\t%d\t%d\t%d\t%d\t%s\t%s\n",
                        p->job->jid,
                        p->job->pid,
                        p->job->ownerid,
                        p->job->run_time,
                        p->job->wait_time,
                        timebuf,
                        "READY");
        }
                                .
                                .
                                .

        if(write(fifo_0,buffer,10000)<0)
                error_sys("write failed");

        close(fifo_0);

}
```

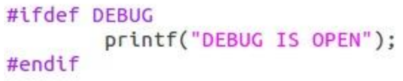
以阻塞只写的形式打开 server_0，控制线程的执行顺序，将所有需要输出的均先写入缓冲区 buffer，优先级为 1、2 的队列同 3 因此在报告中省略。最后将 buffer 写入 server_0 文件。



```
int main(int argc,char *argv[])
{
        struct jobcmd statcmd;
        int fd,fd_0;
        char buffer[10000];

        if(argc!=1)
        {
                usage();
                return 1;
        }

        statcmd.type=STAT;
        statcmd.defpri=0;
        statcmd.owner=getuid();
        statcmd.argnum=0;

        if((fd_0=open("/tmp/server_0",O_RDONLY|O_NONBLOCK))<0)
                error_sys("open fifo failed");

        if((fd=open("/tmp/server",O_WRONLY))<0)
                error_sys("stat open fifo failed");

        if(write(fd,&statcmd,DATALEN)<0)
                error_sys("stat write failed");

        close(fd);
```

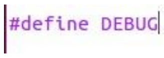
在执行 do_stat 程序段之前，已经以非阻塞只读的方式打开管道（以防阻塞），并始终等待./job 端的输出。读出信息，继续执行。

3. 调试程序状态信息



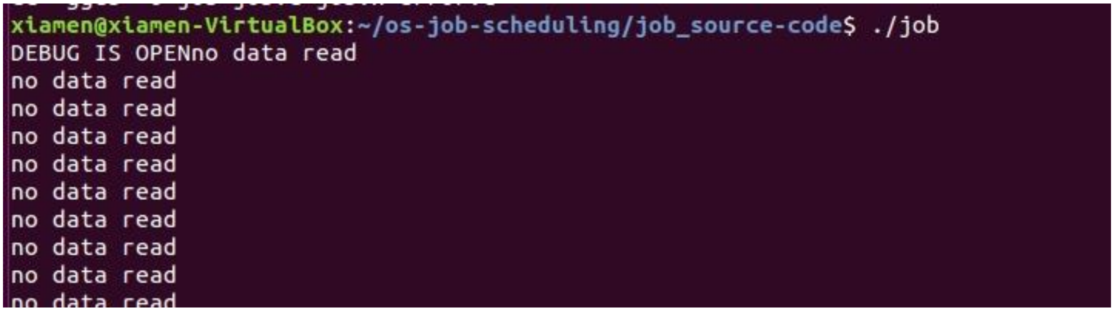
```
#ifdef DEBUG
            printf("DEBUG IS OPEN");
#endif
```

…



```
#define DEBUG
```

对应 job 中输出：



```
xiamen@xiamen-VirtualBox:~/os-job-scheduling/job_source-code$ ./job
DEBUG IS OPENno data read
no data read
no data read
no data read
no data read
no data read
no data read
no data read
no data read
no data read
```