# 四 川 大 学 计 算 机 学 院、软 件 学 院

# 实 验 报 告

学号：2022141460176 姓名：杨一舟 专业：计算机科学与技术 第 11 周

| | | | |
|---|---|---|---|
| 课程名称 | 操作系统课程设计 | 实验课时 | 4 |
| 实验项目 | 处理器调度 | 实验时间 | 2024 年 5 月 6 日 |
| 实验目的 | 1. 理解操作系统中调度的概念和调度算法。<br>2. 学习和应用 Linux 下进程控制以及进程之间通信的知识。<br>3. 理解在操作系统中作业是如何被调度的，如何协调和控制各个作业对 CPU 的使用。 | | |
| 实验环境 | Ubuntu 操作系统 | | |
| **实验创新** | **除了要求算法外，还设计实现了高响应比优先调度算法** | | |

| | |
|---|---|
| 实验内容（算法、程序、步骤和方法） | 一. 实验基本内容：<br><br>1. 作业调度模型要求及特点<br>① 基本特性：时间片为 100 毫秒，作业每次执行的基本单位为 100 毫秒， 在这 100 毫秒内，作业一直在执行，直到时间片到期或执行结束。<br>② 三种作业状态：<br><br>READY：作业准备就绪可以运行<br><br>RUNNING：作业正在运行<br><br>DONE：作业已经运行结束，可以退出。<br><br>设 4 个等级的优先级：0、1、2 和 3，3 最高。<br><br>③ 每个作业有两种优先级：<br><br>初始优先级（initial priority），在作业提交时指定，作业执行过程中保持不变；<br><br>当前优先级（current priority），Scheduler 总是选择当前优先级最高的作业来执行，作业执行过程中可变，更新的情况有两种：<br><br>④ 若作业在就绪队列中等待了 100 毫秒，则将它的当前优先级加 1（最高为 3）；<br><br>若当前运行的作业时间片到，使其暂停执行，将其放入就绪队列中，当前优先级恢复为初始优先级。<br><br>⑤ 建立一个就绪队列，每个提交的作业都放在就绪队列中，scheduler 遍历该队列，找出优先级最高的作业让它执行。<br><br>⑥ 作业调度进程 scheduler：负责整个系统的运行处理作业的入队、出队及状态查看请求，在合适的时间调度各作业运行。如果有新的作业到来，为其创建一个进程，其状态为就绪，将其放入就绪队列中；如果有出队请求则使该作业出队，然后清除相关的数据结构，若该作业当前正在运行，则发信号给它，停止运行，然后出队；如果是状态查看请求，则输出当前运行的作业及就绪队列中所有作业的信息。 |

⑦ 作业入队命令 enq：给 scheduler 调度程序发出入队请求，将作业提交给系统运行；Scheduler 调度程序为每个作业分配一个唯一的 jid（作业号）；为每个作业创建一个进程，并将其状态置为 READY，然后放入就绪队列中。

格式：enq [-p num] e_file args

-p num：可选，该选项指定作业的初始优先级

e_file args：e_file是可执行文件的名字，args是可执行文件的参数。

⑧ 作业出队命令 deq：给 scheduler 调度程序发出一个出队请求

格式：deq jid

⑨ 作业状态查看命令 stat：

输出当前运行作业及就绪队列中各作业的信息：

进程的 pid；

作业提交者的 user name；

作业执行的时间；

在就绪队列中总的等待时间；

作业创建的时刻；

此时作业的状态。

⑩ 时间片与优先级结合：

1、调度程序以时间片为单位进行作业调度（在时间片内，即使有高优先级的作业到来，调度程序不会马上调度）2、每个作业有其动态的优先级，在用完分配的时间片后，可以被优先级更高的作业抢占运行。3、等待队列中的作业等待时间越长，优先级越高。

2. 数据结构

× 作业信息结构：

```
struct jobinfo {
  int       jid;        /* job id */
  int       pid;        /* process id */
  char** cmdarg;      /* the command &
                         args to execute */
  int    defpri;      /* default priority */
  int    curpri;      /* current priority */


  int    ownerid;        /* the job owner id */
  int    wait_time;      /* the time job in
                            waitqueue */

  time_t create_time;    /* the time job create */
  int    run_time;       /* the time job running*/
  enum   jobstate state;  /* job state */

};
```

× 作业调度命令结构：

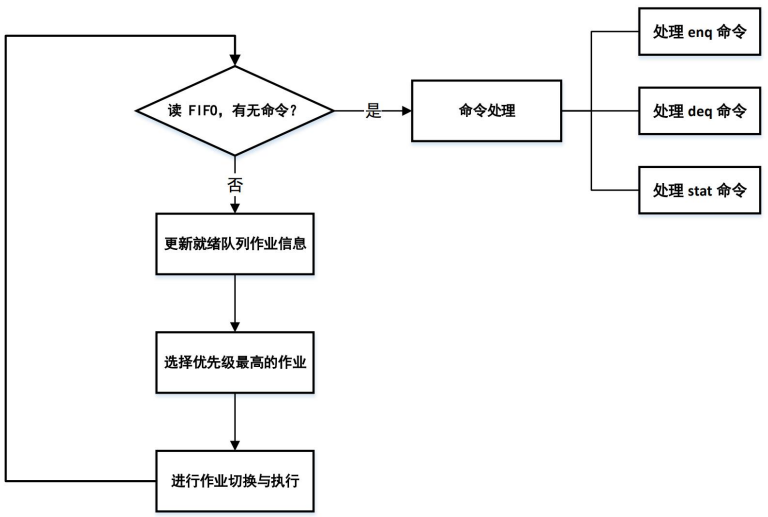
```
struct  jobcmd {
  enum          cmdtype  type;
  int           argnum;
  int           owner;
  int           defpri;
  char    data[BUFLEN];
};


  enum cmdtype
  {
        ENQ = -1,
        DEQ = -2,
        STAT = -3
  };
```

**就绪队列结构**

```
struct waitqueue {
        struct    waitqueue    *next;
        struct    jobinfo      *job;
};
```

3. 调度程序的实现




4. scheduler.c 代码补充：

① 源代码：

```
struct waitqueue* jobselect() {

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

int highest = -1;

int choice = 3;


select = NULL;

selectprev = NULL;
```

```
if(choice == 0) { // HPF 最高优先级优先
    if (head) {
        for (prev = head, p = head; p != NULL; prev = p, p = p->next) {
            if (p->job->curpri > highest) {
                select = p;
                selectprev = prev;
                highest = p->job->curpri;
            }
        }
        selectprev->next = select->next;

        if (select == selectprev) head = NULL;
    }
}
else if (choice == 1){  //FCFS 先到先服务
    int curjid  = 1e5;
    if (head) {
        for (prev = head, p = head; p != NULL; prev = p, p = p->next) {
            if (p->job->jid < curjid) {
                select = p;
                selectprev = prev;
                curjid = p->job->jid;
            }
```

```
            }
            selectprev->next = select->next;

            if (select == selectprev) head = NULL;
        }
    }
    else if (choice == 2){  // SJF 短作业优先
        int curtime  = 1e5;
        if (head) {
            for (prev = head, p = head; p != NULL; prev = p, p = p->next) {
                if (p->job->deftime < curtime) {
                    select = p;
                    selectprev = prev;
                    curtime = p->job->deftime;
                }
            }
            selectprev->next = select->next;

            if (select == selectprev) head = NULL;
        }
    }
    return select;
    }
```

## 二．实验创新内容：实现高响应比优先调度算法

1. HRRN 部分代码：

```
        else if (choice == 3){  //HRRN 高响应比优先
            int curCR  = 0;
            if (head) {
                for (prev = head, p = head; p != NULL; prev = p, p = p->next)
    {
                    int jobCR = p->job->wait_time/p->job->deftime;
                    if (jobCR > curCR) {
                        select = p;
                        selectprev = prev;
                        curCR = jobCR;
                    }
                }
                selectprev->next = select->next;

                if (select == selectprev) head = NULL;
            }
        }
```

2. 实现思路

定义指针变量 select 和指向 select 的前一个节点的指针变量 selectprev。

设置变量 curCR，用于记录当前找到的最高响应比。遍历等待队列中的进程节点，使用指针变量 p 和 prev 进行迭代。计算当前进程节点的响应比，即等待时间与预计完成时间的比值。

计算公式为 ：

jobCR = p->job->wait_time / p->job->deftime。

如果当前进程节点的响应比大于 curCR，则更新 select、selectprev 和 curCR 的值，并根据找到的 select 和 selectprev，将选中的进程节点。

（接上）实验内容（算法、程序、步骤和方法）

## 三. 验证程序功能

1. HPF

设置 switch 值为 0，分别设置 sample1 优先级为 1，sample2 优先级为 2，观察哪个进程被优先调度。

```
mountain@Lumous:~$ ./runstat
mountain@Lumous:~$ ./runenq -p 1 sample1
mountain@Lumous:~$ ./runenq -p 2 sample2
mountain@Lumous:~$ ./runstat
mountain@Lumous:~$ ./runstat
```

2. FCFS

设置 switch 值为 1，分别设置 sample1 优先级为 1，sample2 优先级为 2，观察哪个进程被优先调度。

```
mountain@Lumous:~$ ./runstat
mountain@Lumous:~$ ./runenq -p 1 sample1
mountain@Lumous:~$ ./runenq -p 2 sample2
mountain@Lumous:~$ ./runstat
mountain@Lumous:~$ ./runstat
```

3. SJF

设置 switch 值为 2，分别设置 sample 优先级为 2，预估完成时间为 5，sample1 优先级为 1，预估完成时间为 1，观察哪个进程被优先调度。

```
mountain@Lumous:~$ ./runstat
mountain@Lumous:~$ ./runenq -p 2 -t 5 sample
mountain@Lumous:~$ ./runenq -p 1 -t 1 sample1
mountain@Lumous:~$ ./runstat
```

4. HRRN

设置 switch 值为 3，分别设置 sample 优先级为 3，预估完成时间为 6，sample1 优先级为 2，预估完成时间为 8，sample2 优先级为 1，预估完成时间为 6，观察哪个进程被优先调度。

```
mountain@Lumous:~$ ./runenq -p 3 -t 6 sample
mountain@Lumous:~$ ./runenq -p 2 -t 8 sample1
mountain@Lumous:~$ ./runenq -p 1 -t 6 sample2
mountain@Lumous:~$ ./runstat
mountain@Lumous:~$ ./runstat
```

数据记录和计算

1. HPF:高优先级优先调度算法结果：

```
mountain@Lumous:~$ gcc scheduler.c -o runsch
mountain@Lumous:~$ ./runsch
OK! Scheduler is starting now!!
JID     PID     OWNER   RUNTIME WAITTIME        CREATTIME       STATE

pid : 3585***********

new job: jid=1, pid=3585
begin start new job
pid : 0***********
pid : 3589***********

new job: jid=2, pid=3589
begin running
arglist sample1
pid : 0***********      ./runenq -p 3 -t 6 sample
begin running           ./runenq -p 2 -t 8 sample1
arglist sample2         ./runenq -p 1 -t 6 sample2
JID     PID     OWNER   ./runstat                               STATE
2       3589    1000    ./runstat                  25 2023      RUNNING
1       3585    1000                               15 2023      READY

normal termation, exit status = 0       jid = 1, pid = 3585

begin start new job
normal termation, exit status = 0       jid = 2, pid = 3589

JID     PID     OWNER   RUNTIME WAITTIME        CREATTIME       STATE
```

可知，优先级更高的 sample2 先被调度。

2. FCFS:先来先服务调度算法结果：

```
mountain@Lumous:~$ gcc scheduler.c -o runsch
mountain@Lumous:~$ ./runsch
OK! Scheduler is starting now!!
JID     PID     OWNER   RUNTIME WAITTIME        CREATTIME       STATE

pid : 3734***********

new job: jid=1, pid=3734
begin start new job
pid : 0***********
pid : 3746***********

new job: jid=2, pid=3746
pid : 0***********
begin running
arglist sample1
                ./runenq -p 3 -t 6 sample
begin running   ./runenq -p 2 -t 8 sample1
arglist sample2 ./runenq -p 1 -t 6 sample2
JID     PID     OWNER   ./runstat                               STATE
2       3746    1000    ./runstat                       56 2023         RUNNING
1       3734    1000                                    20 2023         READY

JID     PID     OWNER   RUNTIME WAITTIME        CREATTIME       STATE
2       3746    1000    976     100     Sun Jun  4 03:57:56 2023         RUNNING
1       3734    1000    9779    0       Sun Jun  4 03:57:20 2023         READY

normal termation, exit status = 0       jid = 1, pid = 3734

begin start new job
normal termation, exit status = 0       jid = 2, pid = 3746
```

可知，尽管 sample2 优先级更高，但是 sample1 先到达，所以依然是 sample1 先被调度。

3. SJF:短作业优先调度算法结果：

```
mountain@Lumous:~$ gcc scheduler.c -o runsch
OK! Scheduler is st ./runsch
JID     PID     OWNER   RUNTIME WAITTIME        CREATTIME       STATE

pid : 2638***********

new job: jid=1, pid=2638
begin start new job
pid : 0**********
pid : 2650***********

new job: jid=2, pid=2650
pid : 0**********
begin running
arglist sample
begin running
arglist sample1
JID     PID     OWNER   RUNTIME WAITTIME        CREATTIME       STATE
1       2638    1000    5066    100     Sun Jun  4 01:50:21 2023        RUNNING
2       2650    1000    716     0       Sun Jun  4 01:50:39 2023        READY

normal termation, exit status = 0       jid = 1, pid = 2638

begin start new job
normal termation, exit status = 0       jid = 2, pid = 2650
```

可知，作业时长更短的 sample 先被调用

4. HRRN:高响应比优先调度算法结果：

```
mountain@Lumous:~$ ./runsch
OK! Scheduler is starting now!!
JID     PID     OWNER   RUNTIME WAITTIME        CREATTIME       STATE

pid : 4069**********

new job: jid=1, pid=4069
begin start new job
pid : 0**********
pid : 4078**********

new job: jid=2, pid=4078
pid : 0**********
begin running
arglist sample
begin running
arglist sample1
normal termation, exit status = 0       jid = 2, pid = 4078

begin start new job
normal termation, exit status = 0       jid = 1, pid = 4069

pid : 4083**********

new job: jid=3, pid=4083
begin start new job
pid : 0**********

JID     PID     OWNER   RUNTIME WAITTIME        CREATTIME       STATE
3       4083    1000    6620    100     Sun Jun  4 04:32:49 2023        RUNNING

JID     PID     OWNER   RUNTIME WAITTIME        CREATTIME       STATE
3       4083    1000    11532   100     Sun Jun  4 04:32:49 2023        RUNNING
```

结论（结果）

1、成功实现以下命令功能：

①作业入队命令 enq：给 scheduler 调度程序发出入队请求，将作业提交给系统，使其得到分配的 jid 作业号，并为每个作业创建一个进程，并将其状态置为 READY，放入就绪队列中。

②作业出队命令 deq：给 scheduler 调度程序发出一个出队请求。

③作业状态查看命令 stat：输出当前运行作业及就绪队列各作业的所有信息。

2、成功模拟实现 HPF FCFS SJF HRRN 调度算法。

| | |
|---|---|
| 小结 | 通过本次实验，我深入了解了处理器调度策略的工作原理。实验模拟了多种调度算法，并分析了它们在不同场景下的性能。实验结果表明，合适的调度策略能显著提升系统效率，确保任务的高效执行。 |
| 指导老师评议 | 成绩评定：　　指导教师签名： |