实 验 报 告

EXPERIMENT REPORT

姓名 高星杰

学号 2021307220712

专业 计科 2102

教师 任继平

科目 操作系统

信 息 学 院

COLLEGE OF INFORMATIC

2024 年 1 月

实验 1　进程调度

1．实验目的

进程调度是处理及管理的核心内容，本次实验要求用 C 语言编写和调试一个简单的进程调度程序。调度算法可以任意选择或自行设计，例如简单轮转法和优先数法等。通过本次实验可以加深各使用进程控制块进行进程调度和各种调度算法的理解及其实施方法。

2．实验内容

(1) 在微型计算机上设计进程控制块（PCB）结构，使其分别适用于简单轮转法和优先数调度算法。PCB 通常包括以下信息：进程名、进程优先数（或轮转时间片）、进程所占用的 CPU 时间、进程的当前状态、当前队列指针等。根据调度算法的不同，PCB 结构的内容可以作适当的增删。

(2) 调度程序应包含 2~3 种不同的调度算法，运行时可任选一种，以利于各种算法的分析比较。基本要求是：优先数调度算法和简单循环轮转法。

(3) 建立进程就绪队列，对各种不同的算法编制入链子程序，同时应具有显示或打印各进程的运行状态和参数的变化情况，便于观察各进程的调度过程。

3　实验步骤

（1） 任务分析：

使用 C 语言编写一个程序，该程序应用优先数调度算法或简单轮转法对五个进程进行调度。这五个进程在任何时刻都处于以下三种状态之一：运行（Run）、就绪（Ready）和完成（Finish），并且所有进程最初都处于就绪状态。以下是对实验任务的详细阐述：

① 输入的形式和输入值的范围：

形式：通过文本界面输入，用户可输入每个进程的相关信息。

范围：进程名（字符串，无特定限制），优先数（整数，例如 1-10），时间片长度（整数，例如 10-500 毫秒），CPU 占用时间（整数，例如 1-1000 秒）。

② 输出的形式：

实时显示进程的调度和运行状态，包括进程名、当前状态（等待、运行、完成）、占用 CPU 时间等。进程的状态变化和队列变动将在屏幕上实时更新显示。

③ 程序所能达到的功能：

支持至少两种调度算法：优先数调度和简单轮转法。允许用户输入和修改进程信息。显示进程在调度中的状态变化，包括就绪、运行、阻塞、完成等。对比不同调度算法的效率和效果。

④ 测试数据：

正确的输入及其输出结果：例如，输入一个优先级为 5 的进程，时间片为 100 毫秒，在优先数调度算法中，应当看到该进程根据优先级被调度。

含有错误的输入及其输出结果：例如，输入一个优先级为-1 的进程，在输入时程序应给出

错误提示，防止错误数据影响调度算法的运行。

（2） 概要设计：

本程序中定义了一个抽象数据类型 PCB，这是一个结构体，代表进程控制块。**它包含以下字段：**

- name：进程标识符，类型为字符数组。
- prio：进程优先数，类型为整数。
- round：进程时间轮转时间片，类型为整数。
- cputime：进程占用 CPU 时间，类型为整数。
- needtime：进程到完成还要的时间，类型为整数。
- count：计数器，类型为整数。
- state：进程的状态，类型为字符。
- next：链指针，类型为指向 PCB 结构体的指针。

主程序的流程如下：

① 提示用户选择算法类型（优先级调度算法或时间片轮转法）。

② 提示用户输入进程个数。

③ 根据用户选择的算法类型，调用相应的函数创建进程并进行调度。

程序模块之间的层次（调用）关系如下：

- main 函数是程序的入口点，它根据用户的输入调用 create1 和 priority 函数（优先级调度算法），或者 create2 和 roundrun 函数（时间片轮转法）。
- create1 和 create2 函数用于创建进程，它们会调用 insert1 或 insert2 函数将新创建的进程插入到就绪队列中。
- priority 和 roundrun 函数用于进行进程调度，它们会调用 firstin 函数将就绪队列中的第一个进程投入运行，并在需要时调用 insert1 或 insert2 函数将进程重新插入到就绪队列中。
- prt 函数用于输出进程信息，它会调用 prt1 和 prt2 函数输出进程的各项属性。

（3） 详细设计

这里定义的所有数据类型，对每个操作只需要写出伪码算法；对主程序和其他模块也都需要写出伪码算法(伪码算法达到的详细程度建议为：按照伪码算法可以在计算机键盘直接输入高级程序设计语言程序)；

定义的数据类型：

```
typedef struct node
{
    char name[10];      /*进程标识符*/
    int prio;           /*进程优先数*/
    int round;          /*进程时间轮转时间片*/
    int cputime;        /*进程占用CPU时间*/
    int needtime;       /*进程到完成还要的时间*/
    int count;          /*计数器*/
    char state;         /*进程的状态*/
    struct node *next; /*链指针*/
} PCB;
```

定义的队列以及全局变量：

```
PCB *finish, *ready, *tail, *run; /*队列指针*/
int N;                            /*进程数*/
```

以下是每个函数的伪代码以及主程序的伪代码：

1. `firstin` 函数

函数 firstin

    将运行指针指向就绪队列的头部

    将运行进程的状态设为 'R'

    将就绪队列的头部指向下一个进程

结束函数

2. `prt1` 函数

函数 prt1 接收一个字符 a

    如果 a 是 'P' 或 'p'

        打印优先级调度算法的标题

    否则

        打印时间片轮转法的标题

结束函数

3. `prt2` 函数

函数 prt2 接收一个字符 a 和一个 PCB 指针 q

    如果 a 是 'P' 或 'p'

        打印优先级调度算法的进程信息

    否则

        打印时间片轮转法的进程信息

结束函数

4. `prt` 函数

函数 prt 接收一个字符 algo

    调用 prt1 函数打印标题

    如果运行队列不为空

        调用 prt2 函数打印运行进程的信息

    对就绪队列和完成队列中的每个进程

        调用 prt2 函数打印进程信息

    等待用户输入

结束函数

5. `insert1` 函数

函数 insert1 接收一个 PCB 指针 q

    初始化指针和变量

    在就绪队列中找到合适的插入位置

    插入新的 PCB

结束函数

6. `insert2` 函数

函数 insert2 接收一个 PCB 指针 p2

    将新的 PCB 插入到就绪队列的尾部

    更新尾部指针

结束函数

7. `create1` 函数

函数 create1 接收一个字符 alg

    初始化队列

    输入进程信息并创建 PCB

    插入新的 PCB 到就绪队列

    打印进程信息

    将就绪队列的第一个进程投入运行

结束函数

8. `create2` 函数

函数 create2 接收一个字符 alg

    初始化队列

    输入进程信息并创建 PCB

插入新的 PCB 到就绪队列

打印进程信息

将就绪队列的第一个进程投入运行

结束函数

9. `priority` 函数

函数 priority 接收一个字符 alg

当运行队列不为空时

更新运行进程的信息

如果进程已完成

将其插入到完成队列

如果就绪队列不为空

将就绪队列的第一个进程投入运行

否则

如果就绪队列不为空且运行进程的优先级低于就绪队列的头部进程

将运行进程插入到就绪队列

将就绪队列的第一个进程投入运行

打印进程信息

结束函数

10. `roundrun` 函数

函数 roundrun 接收一个字符 alg

当运行队列不为空时

更新运行进程的信息

如果进程已完成

将其插入到完成队列

如果就绪队列不为空

将就绪队列的第一个进程投入运行

否则

如果时间片已到

如果就绪队列不为空

将运行进程插入到就绪队列

将就绪队列的第一个进程投入运行

打印进程信息

结束函数

11.主程序的伪代码：

主程序

    输入算法类型

    输入进程个数

    如果选择的是优先级调度算法

        调用 create1 函数创建进程

        调用 priority 函数进行调度

    否则

        调用 create2 函数创建进程

        调用 roundrun 函数进行调度

结束主程序

函数和过程的调用关系图：

```
main()
|
├─── create1() 或 create2() (取决于用户输入的算法类型)
|    ├─── insert1() 或 insert2() (取决于用户输入的算法类型)
|
└─── priority() 或 roundrun() (取决于用户输入的算法类型)
     ├─── firstin()
     ├─── insert1() 或 insert2() (取决于用户输入的算法类型)
     └─── prt()
          └─── prt1()
          └─── prt2()
```

`main()` 函数是程序的入口点,它首先根据用户的输入调用 `create1()` 或 `create2()` 函数创建进程控制块（PCB)。然后，它调用 `priority()` 或 `roundrun()` 函数进行进程调度。

`create1()` 和 `create2()` 函数在创建 PCB 时会调用 `insert1()` 或 `insert2()` 函数将 PCB 插入到就绪队列中。

`priority()` 和 `roundrun()` 函数在进行进程调度时会调用 `firstin()` 函数将就绪队列中的第一个进程投入运行，同时也可能会调用 `insert1()` 或 `insert2()` 函数将 PCB 插入到就绪队列中。此外，它们还会调用 `prt()` 函数输出进程的状态。

`prt()` 函数会调用 `prt1()` 和 `prt2()` 函数输出进程的信息。

（4） 调试分析：

① 遇到问题及解决方案

内存分配失败：

问题：使用 malloc 分配内存时，如果系统内存不足，可能会失败。

解决方法：在每次调用 malloc 后检查返回值是否为 NULL。如果是，应处理错误，例如通过打印错误消息并退出程序。

头指针处理不当：

问题描述：在操作链表时，若不当处理头指针（如 ready），可能导致链表断裂或丢失数据。

解决方法：在插入和删除操作中仔细处理头指针的更改，确保链表的完整性。内存泄漏：

问题：在使用动态分配的内存后没有适当地释放内存。

解决方法：确保在不再需要时使用 free 释放所有通过 malloc 分配的内存。

资源竞争和优先级反转：

问题描述：在多进程环境下，资源竞争和优先级反转可能导致性能问题或死锁。

解决方法：实施优先级继承协议或其他死锁预防机制。

无限循环或逻辑错误：

问题：程序可能因逻辑错误而陷入无限循环，或者不按预期工作。

解决方法：进行彻底的测试，特别是边界情况测试，以及使用调试工具跟踪和修复错误。

用户输入错误处理：

问题：用户可能输入无效数据，导致程序崩溃或行为异常。

解决方法：对用户输入进行验证和错误处理。确保处理所有潜在的无效输入。

代码可维护性和可读性问题：

问题：代码难以理解和维护，特别是对于复杂的逻辑。

解决方法：使用清晰的命名约定，适当地注释代码，并保持模块化来提高代码的可读性和可维护性。

平台兼容性问题：

问题：代码可能在不同的操作系统或硬件平台上表现不同。

解决方法：使用标准库函数，并在不同平台上测试程序以确保兼容性。

② 复杂度分析

这段代码主要实现了两种调度算法：优先级调度算法和时间片轮转法。这两种算法的时间复杂度和空间复杂度主要取决于进程的数量。

a. **时间复杂度：**在这段代码中，主要的操作是遍历进程列表，所以时间复杂度主要取决于进程的数量。因此，我们可以认为这段代码的时间复杂度是 O(N)，其中 N 是进程的数量。

b. **空间复杂度：**这段代码中，每个进程都有一个 PCB（进程控制块）结构体来存储其信息，

所以空间复杂度也是O(N)，其中N是进程的数量。

③ 改进设想

这段代码的优化可能会从减少遍历次数和优化数据结构两个方面进行。例如，可以使用优先队列来存储进程，这样在选择下一个要运行的进程时，可以在O(logN)的时间复杂度内完成，而不是遍历所有进程。

（5） 测试结果：

5.1 优先级调度算法

此算法根据进程的优先级进行调度。通常情况下，数字越小，优先级越高。

测试案例 1：基本场景

输入：

算法类型：优先级（P）

进程数目：3

进程详情：

进程 1：名称 - P1，所需时间 - 5

进程 2：名称 - P2，所需时间 - 3

进程 3：名称 - P3，所需时间 - 1

预期输出：进程应按其优先级的顺序完成，优先级与所需时间成反比。因此，执行顺序应为P3, P2, P1。

实际输出：

```
选择算法类型:P/R(优先算法/轮转法)
P
输入进程个数:
3
输入进程名称和运行时间
P1 5
P2 3
P3 1
 优先算法的输出:
************************************************
 name cputime needtime priority state
 P3        0         1         49        w
 P2        0         3         47        w
 P1        0         5         45        w
 name cputime needtime priority state
 P2        0         3         47        R
 P1        0         5         45        w
 P3        1         0         47        F

 name cputime needtime priority state
 P2        1         2         45        R
```

```
P1        0         5         45        w
P3        1         0         47        F

name cputime needtime priority state
P1        0         5         45        R
P2        2         1         43        W
P3        1         0         47        F

name cputime needtime priority state
P1        1         4         43        R
P2        2         1         43        W
P3        1         0         47        F

name cputime needtime priority state
P2        2         1         43        R
P1        2         3         41        W
P3        1         0         47        F

name cputime needtime priority state
P1        2         3         41        R
P2        3         0         41        F
P3        1         0         47        F

name cputime needtime priority state
P1        3         2         39        R
P2        3         0         41        F
P3        1         0         47        F

name cputime needtime priority state
P1        4         1         37        R
P2        3         0         41        F
P3        1         0         47        F

name cputime needtime priority state
P1        5         0         35        F
P2        3         0         41        F
P3        1         0         47        F
```

测试案例 2：相同优先级

输入：与上面相同，但是进程 2 和进程 3 的所需时间都是 3。

预期输出：具有相同优先级的进程（P2 和 P3）的执行顺序可以是任意的，但应优先于 P1。

实际输出：

```
选择算法类型:P/R(优先算法/轮转法)
P
输入进程个数:
3
输入进程名称和运行时间
P1 5
P2 3
```

```
P3 3
 优先算法的输出:
**********************************************
 name cputime needtime priority state
 P2        0         3         47        w
 P3        0         3         47        w
 P1        0         5         45        w
 name cputime needtime priority state
 P3        0         3         47        R
 P1        0         5         45        w
 P2        1         2         45        W

 name cputime needtime priority state
 P3        1         2         45        R
 P1        0         5         45        w
 P2        1         2         45        W

 name cputime needtime priority state
 P1        0         5         45        R
 P2        1         2         45        W
 P3        2         1         43        W

 name cputime needtime priority state
 P2        1         2         45        R
 P3        2         1         43        W
 P1        1         4         43        W

 name cputime needtime priority state
 P2        2         1         43        R
 P3        2         1         43        W
 P1        1         4         43        W

 name cputime needtime priority state
 P3        2         1         43        R
 P1        1         4         43        W
 P2        3         0         41        F

 name cputime needtime priority state
 P1        1         4         43        R
 P3        3         0         41        F
 P2        3         0         41        F

 name cputime needtime priority state
 P1        2         3         41        R
 P3        3         0         41        F
 P2        3         0         41        F

 name cputime needtime priority state
 P1        3         2         39        R
 P3        3         0         41        F
 P2        3         0         41        F
```

```
 name cputime needtime priority state
 P1        4         1         37        R
 P3        3         0         41        F
 P2        3         0         41        F

 name cputime needtime priority state
 P1        5         0         35        F
 P3        3         0         41        F
 P2        3         0         41        F
```

5.2 轮转调度算法

此算法为每个进程分配固定的时间单元，并依次循环执行。

测试案例 3：基本场景

输入：

算法类型：轮转（R）

进程数目：3

进程详情（假设时间片为 2）：

进程 1：名称 - P1，所需时间 - 4

进程 2：名称 - P2，所需时间 - 5

进程 3：名称 - P3，所需时间 - 2

预期输出：

进程应按照轮转的方式执行。P1 运行 2 个单位时间，然后是 P2 运行 2 个单位时间，接着是 P3 运行 2 个单位时间，依此类推，直到所有进程完成。

实际输出：

```
选择算法类型:P/R(优先算法/轮转法)
R
输入进程个数:
3
输入进程的名称和运行时间:
P1 4
P2 5
P3 2
 简单时间片轮转输出
************************************************
 name cputime needtime count round state
 P1        0         4         0         2         w
 P2        0         5         0         2         w
 P3        0         2         0         2         w
 name cputime needtime count round state
 P1        1         3         1         2         R
 P2        0         5         0         2         w
 P3        0         2         0         2         w
```

```
name cputime needtime count round state
P2        0         5         0         2         R
P3        0         2         0         2         w
P1        2         2         0         2         W

name cputime needtime count round state
P2        1         4         1         2         R
P3        0         2         0         2         w
P1        2         2         0         2         W

name cputime needtime count round state
P3        0         2         0         2         R
P1        2         2         0         2         W
P2        2         3         0         2         W

name cputime needtime count round state
P3        1         1         1         2         R
P1        2         2         0         2         W
P2        2         3         0         2         W

name cputime needtime count round state
P1        2         2         0         2         R
P2        2         3         0         2         W
P3        2         0         2         2         F

name cputime needtime count round state
P1        3         1         1         2         R
P2        2         3         0         2         W
P3        2         0         2         2         F

name cputime needtime count round state
P2        2         3         0         2         R
P1        4         0         2         2         F
P3        2         0         2         2         F

name cputime needtime count round state
P2        3         2         1         2         R
P1        4         0         2         2         F
P3        2         0         2         2         F

name cputime needtime count round state
P2        4         1         0         2         R
P1        4         0         2         2         F
P3        2         0         2         2         F

name cputime needtime count round state
P2        5         0         1         2         F
P1        4         0         2         2         F
P3        2         0         2         2         F
```

测试案例 4：进程在时间片到期前完成

输入：与上面相同，但进程 3 的所需时间为 1。

预期输出：P3 应在其第一次分配的 1 个时间单位后完成，并且不会占用整个时间片。

实际输出：

```
选择算法类型:P/R(优先算法/轮转法)
R
输入进程个数:
3
输入进程的名称和运行时间:
P1 4
P2 5
P3 1
 简单时间片轮转输出
**********************************************
 name cputime needtime count round state
 P1        0         4         0         2         w
 P2        0         5         0         2         w
 P3        0         1         0         2         w
 name cputime needtime count round state
 P1        1         3         1         2         R
 P2        0         5         0         2         w
 P3        0         1         0         2         w

 name cputime needtime count round state
 P2        0         5         0         2         R
 P3        0         1         0         2         w
 P1        2         2         0         2         W

 name cputime needtime count round state
 P2        1         4         1         2         R
 P3        0         1         0         2         w
 P1        2         2         0         2         W

 name cputime needtime count round state
 P3        0         1         0         2         R
 P1        2         2         0         2         W
 P2        2         3         0         2         W

 name cputime needtime count round state
 P1        2         2         0         2         R
 P2        2         3         0         2         W
 P3        1         0         1         2         F

 name cputime needtime count round state
 P1        3         1         1         2         R
 P2        2         3         0         2         W
 P3        1         0         1         2         F
```

```
name cputime needtime count round state
P2        2         3         0         2         R
P1        4         0         2         2         F
P3        1         0         1         2         F

name cputime needtime count round state
P2        3         2         1         2         R
P1        4         0         2         2         F
P3        1         0         1         2         F

name cputime needtime count round state
P2        4         1         0         2         R
P1        4         0         2         2         F
P3        1         0         1         2         F

name cputime needtime count round state
P2        5         0         1         2         F
P1        4         0         2         2         F
P3        1         0         1         2         F
```

（6） 使用说明：

使用这个程序的操作步骤如下：

6.1 编译程序:

首先，确保你有一个 C 语言编译器，如 GCC。

将上述代码保存为一个 C 文件，例如 process_scheduler.c。

在命令行中，使用编译器编译此文件。例如，如果使用 GCC，输入命令 gcc -o process_scheduler process_scheduler.c。

6.2 运行程序:

在编译成功后，你会得到一个可执行文件。在命令行中，运行这个文件。例如，在 Windows 上，运行 process_scheduler.exe；在 Unix-like 系统上，运行./process_scheduler。

6.3 选择调度算法:

程序启动后，首先会提示你选择算法类型。输入 P 或 p 选择优先级调度算法，输入 R 或 r 选择时间片轮转法。

6.4 输入进程数量:

程序接着会要求输入进程的数量。输入一个整数表示你想要模拟的进程数。

6.5 输入进程信息:

根据选择的算法类型，程序会要求你为每个进程输入特定的信息：

对于优先级调度算法（P），输入每个进程的名称和运行时间。

对于时间片轮转算法（R），输入每个进程的名称和运行时间。

6.6 观察调度过程:

输入完进程信息后，程序会开始模拟进程调度过程。屏幕上将显示每个进程的状态变化，包括它们在 CPU 上的运行时间、剩余需要时间、状态等。

对于优先级调度，会看到进程根据优先级和剩余时间的变化被调度。

对于时间片轮转，进程会根据分配给它们的时间片进行调度。

6.7 程序结束：

当所有进程都完成执行后，程序会结束。你可以通过观察最后的进程状态来了解它们的完成情况。

6.8 程序退出：

在程序运行完毕后，根据提示，可能需要按任意键退出程序。

确保在输入时遵循程序提示，并正确输入所需的信息。如果输入错误，程序可能无法正确执行或崩溃。此外，由于程序使用控制台输入，非法输入（如字母代替数字）可能导致未定义行为。

4. 实验总结

本次实验旨在通过编写和调试一个简单的进程调度程序，加深对进程调度和不同调度算法的理解及实施方法。在实验中，我们设计了进程控制块（PCB）的结构，并针对简单轮转法和优先数调度算法进行了实现。

在实验过程中，我们成功创建了进程就绪队列，并编写了与不同调度算法相对应的调度程序。通过输入进程的相关信息，我们能够实时显示进程的调度和运行状态，包括进程名、当前状态、占用 CPU 时间等。同时，我们还能观察各进程在调度过程中的状态变化和队列变动。通过本次实验，我们掌握了使用 C 语言编写进程调度程序的基本技巧。我们了解了进程控制块的结构和内容，并学会了根据不同调度算法对进程进行调度和管理。实验中的任务分析、概要设计和详细设计等步骤，提供了对问题进行分析和解决的思路和方法。

此外，我们还测试了正确输入和错误输入的情况，并观察了相应的输出结果。正确的输入能够得到预期的调度效果，而错误的输入则会得到错误提示，避免了错误数据对调度算法的影响。

本次实验使我们深入了解了进程调度的核心概念和实现方法。通过编写和调试进程调度程序，我们加深了对进程控制块、调度算法和队列管理的理解，同时也培养了问题分析和解决的能力。这些知识和技能对于进一步学习和应用操作系统相关内容具有重要意义。

5. 附录

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h> // Add the missing header for the toupper function

typedef struct node
{
```

```
    char name[10];      /*进程标识符*/
    int prio;           /*进程优先数*/
    int round;          /*进程时间轮转时间片*/
    int cputime;        /*进程占用CPU时间*/
    int needtime;       /*进程到完成还要的时间*/
    int count;          /*计数器*/
    char state;         /*进程的状态*/
    struct node *next; /*链指针*/
} PCB;

PCB *finish, *ready, *tail, *run; /*队列指针*/
int N;                            /*进程数*/

/*将就绪队列中的第一个进程投入运行*/
void firstin() // Add the return type 'void'
{
    run = ready;          /*就绪队列头指针赋值给运行头指针*/
    run->state = 'R';     /*进程状态变为运行态*/
    ready = ready->next; /*就绪对列头指针后移到下一进程*/
}

void prt1(char a)
{
    if (toupper(a) == 'P') /*优先级调度算法*/
        printf(" name cputime needtime priority state\n");
    else
        printf(" name cputime needtime count round state\n");
}

/*进程PCB输出*/
void prt2(char a, PCB *q)
{
    if (toupper(a) == 'P') /*优先级调度算法的输出*/
        printf(" %-10s%-10d%-10d%-10d %c\n", q->name,
               q->cputime, q->needtime, q->prio, q->state);
    else /*时间片轮转算法的输出*/
        printf(" %-10s%-10d%-10d%-10d%-10d %-c\n", q->name,
               q->cputime, q->needtime, q->count, q->round, q->state);
}

/*输出函数*/
void prt(char algo)
{
    PCB *p;
```

```
    prt1(algo);           /*输出标题*/
    if (run != NULL)      /*如果运行指针不空*/
        prt2(algo, run); /*输出当前正在运行的PCB*/
    p = ready;            /*输出就绪队列PCB*/
    while (p != NULL)
    {
        prt2(algo, p);
        p = p->next;
    }
    p = finish; /*输出完成队列的PCB*/
    while (p != NULL)
    {
        prt2(algo, p);
        p = p->next;
    }
    getchar(); /*压任意键继续*/
}

/*优先数的插入算法*/
void insert1(PCB *q) // Add the return type 'void'
{
    PCB *p1, *s, *r;
    int b;
    s = q;       /*待插入的PCB 指针*/
    p1 = ready; /*就绪队列头指针*/
    r = p1;      /*r 做p1 的前驱指针*/
    b = 1;
    while ((p1 != NULL) && b) /*根据优先数确定插入位置*/
        if (p1->prio >= s->prio)
        {
            r = p1;
            p1 = p1->next;
        }
        else
            b = 0;
    if (r != p1) /*如果条件成立说明插入在r 与p1 之间*/
    {
        r->next = s;
        s->next = p1;
    }
    else
    {
        s->next = p1; /*否则插入在就绪队列的头*/
        ready = s;
```

```
    }
}

/*轮转法插入函数*/
void insert2(PCB *p2) // Add the return type 'void'
{
    tail->next = p2; /*将新的PCB 插入在当前就绪队列的尾*/
    tail = p2;
    p2->next = NULL;
}

/*优先数创建初始PCB 信息*/
void create1(char alg)
{
    PCB *p;
    int i, time;
    char na[10];
    ready = NULL;                          /*就绪队列头指针*/
    finish = NULL;                         /*完成队列头指针*/
    run = NULL;                            /*运行队列指针*/
    printf("输入进程名称和运行时间\n"); /*输入进程标识和所需时间创建PCB*/
    for (i = 1; i <= N; i++)
    {
        p = malloc(sizeof(PCB));
        scanf("%s", na);
        scanf("%d", &time);
        strcpy(p->name, na);
        p->cputime = 0;
        p->needtime = time;
        p->state = 'w';
        p->prio = 50 - time;
        if (ready != NULL) /*就绪队列不空调用插入函数插入*/
            insert1(p);
        else
        {
            p->next = ready; /*创建就绪队列的第一个PCB*/
            ready = p;
        }
    }
    // clrscr();
    printf(" 优先算法的输出:\n");
    printf("************************************************\n");
    prt(alg);    /*输出进程PCB 信息*/
    run = ready; /*将就绪队列的第一个进程投入运行*/
```

```
    ready = ready->next;
    run->state = 'R';
}

/*轮转法创建进程PCB*/
void create2(char alg)
{
    PCB *p;
    int i, time;
    char na[10];
    ready = NULL;
    finish = NULL;
    run = NULL;
    printf("输入进程的名称和运行时间:\n");
    for (i = 1; i <= N; i++)
    {
        p = malloc(sizeof(PCB));
        scanf("%s", na);
        scanf("%d", &time);
        strcpy(p->name, na);
        p->cputime = 0;
        p->needtime = time;
        p->count = 0; /*计数器*/
        p->state = 'w';
        p->round = 2; /*时间片*/
        if (ready != NULL)
            insert2(p);
        else
        {
            p->next = ready;
            ready = p;
            tail = p;
        }
    }
    // clrscr();
    printf(" 简单时间片轮转输出 \n");
    printf("************************************************\n");
    prt(alg);    /*输出进程PCB 信息*/
    run = ready; /*将就绪队列的第一个进程投入运行*/
    ready = ready->next;
    run->state = 'R';
}

/*优先级调度算法*/
```

```
void priority(char alg) // Add the return type 'void'
{
    while (run != NULL) /*当运行队列不空时，有进程正在运行*/
    {
        run->cputime = run->cputime + 1;
        run->needtime = run->needtime - 1;
        run->prio = run->prio - 2; /*每运行一次优先数降低 2 个单位*/
        if (run->needtime == 0)    /*如所需时间为 0 将其插入完成队列*/
        {
            run->next = finish;
            finish = run;
            run->state = 'F';  /*置状态为完成态*/
            run = NULL;        /*运行队列头指针为空*/
            if (ready != NULL) /*如就绪队列不空*/
                firstin();     /*将就绪队列的第一个进程投入运行*/
        }
        else /*没有运行完同时优先数不是最大，则将其变为就绪态插入到就绪队列*/
            if ((ready != NULL) && (run->prio < ready->prio))
            {
                run->state = 'W';
                insert1(run);
                firstin(); /*将就绪队列的第一个进程投入运行*/
            }
        prt(alg); /*输出进程 PCB 信息*/
    }
}

/*时间片轮转法*/
void roundrun(char alg) // Add the return type 'void'
{
    while (run != NULL)
    {
        run->cputime = run->cputime + 1;
        run->needtime = run->needtime - 1;
        run->count = run->count + 1;
        if (run->needtime == 0) /*运行完将其变为完成态，插入完成队列*/
        {
            run->next = finish;
            finish = run;
            run->state = 'F';
            run = NULL;
            if (ready != NULL)
                firstin(); /*就绪队列不空，将第一个进程投入运行*/
        }
```

```
        else if (run->count == run->round) /*如果时间片到*/
        {
            run->count = 0;    /*计数器置0*/
            if (ready != NULL) /*如就绪队列不空*/
            {
                run->state = 'W'; /*将进程插入到就绪队列中等待轮转*/
                insert2(run);
                firstin(); /*将就绪队列的第一个进程投入运行*/
            }
        }
        prt(alg); /*输出进程信息*/
    }
}

/*主函数*/
int main()
{
    char flag; /*算法标记*/
    printf("选择算法类型:P/R(优先算法/轮转法)\n");
    scanf("%c", &flag); /*输入字符确定算法*/
    printf("输入进程个数:\n");
    scanf("%d", &N); /*输入进程数*/
    if (flag == 'P' || flag == 'p')
    {
        create1(flag); /*优先级调度算法*/
        priority(flag);
    }
    else if (flag == 'R' || flag == 'r')
    {
        create2(flag); /*时间片轮转法*/
        roundrun(flag);
    }
}
```