计算机操作系统实验之进程调度（一）轮转调度法（C语言）

原创

发量充足的小姚 于 2019-11-04 15:33:41 发布 6164 收藏 83

分类专栏： 计算机操作系统 文章标签： 轮转法 进程调度

本文链接：https://blog.csdn.net/weixin_42643216/article/details/102897159

计算机操作系统实验之进程调度（一）轮转调度法（C语言）

进程调度有多种方法，前一周的实验只写了两个，打算每种方法都分开整理一下，就是不知道什么时候腾出时间写完了。今天就先说轮转调度法吧。

实验目的

1、理解进程调度的任务、机制和方式

2、了解轮转调度算法的基本原理

实验内容与基本要求

用C，C++等语言编写程序，模拟使用轮转调度算法实现进程调度

轮转调度法的基本原理

在轮转调度法中，系统根据FCFS(先来先服务)原则，把所有的就绪进程排成一个就绪队列，并可设置每隔一定时间间隔产生一次中断，当当前运行进程结束或者时间用完时，执行一次进程调度，将CPU分配给新的队首进程。这样可以保证所有进程在一个确定的时间段内，都可以获得一次CPU执行。

在这个算法中，隐含着一个基本假设：所有进程的紧迫性是一致的，这样才能按照其到达的先后顺序来执行。显然这在实际当中并不存在，因此才有了后面要说的优先级调度算法。

附：优先级调度法计算机操作系统实验之进程调度（二）优先级调度法（C语言）

轮转调度法中，需要考虑的核心问题是对时间片大小的确定。假如时间片选择过小，那么将会频繁地进行进程调度和切换，这样就增大了系统开销；而时间片选择过大，所有进程都能在一个时间片内完成，轮转法实际上就退化成了FCFS算法，无法满足短作业和交互式用户的需求。当然，我们实验中不考虑这个问题，只要设置好一个固定的时间片就行了。

实现思路及功能分析

要进行进程调度，依然要模拟出PCB的结构。在本次实验中，要求展示时间片、cpu时间等信息，因此设计PCB的属性包括：1.进程名字2.进程优先级3.进程的状态4.时间片5.总共需要的运行时间6.cpu时间（已运行时间）7.还需要运行的时间8.计数器9.下一个要执行的进程指针。

这样，我们使用带头结点的单链表来存储进程队列。按理来说，当进程运行结束后，应该将它移出队列，但是为了统一展示进程调度的过程、信息，已完成的进程不移出队列，而是沉到队列底部，只利用进程的状态属性来区分进程的完成情况。

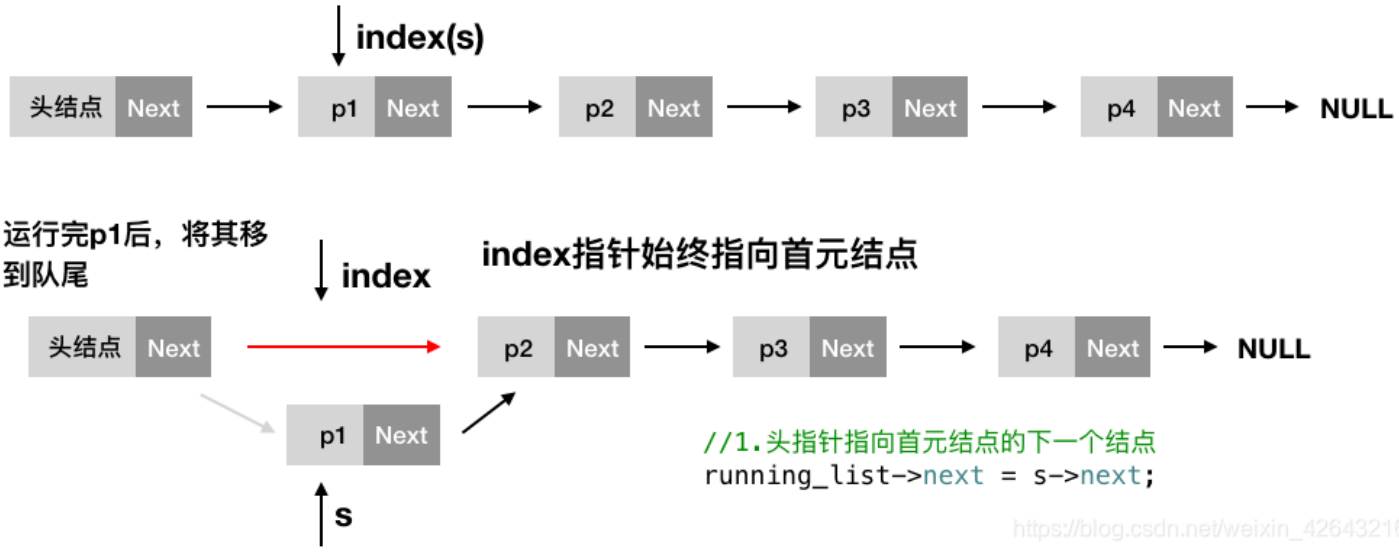
实现轮转调度算法本质就是按插入进程队列的顺序依次运行，已完成的进程沉到队列最低端，直到所有进程都完成为止。因此进程队列逻辑上可以分成两个部分，前一部分是未完成的进程，后一部分是已完成的进程，当我们执行完一个进程，要将它调到队列尾部时，实际上调到的是未完成部分的尾部，已完成的部分不需要考虑。

因此执行完某个进程后，遍历链表寻找进程的插入位置时，停止遍历的条件有两个，第一个条件是结点的next值为空（此时该结点是最后一个结点，进程要插入的是链表最末尾），第二个条件是结点的next结点的状态为已完成（此时该结点往后均是已完成的结点，不需要考虑）

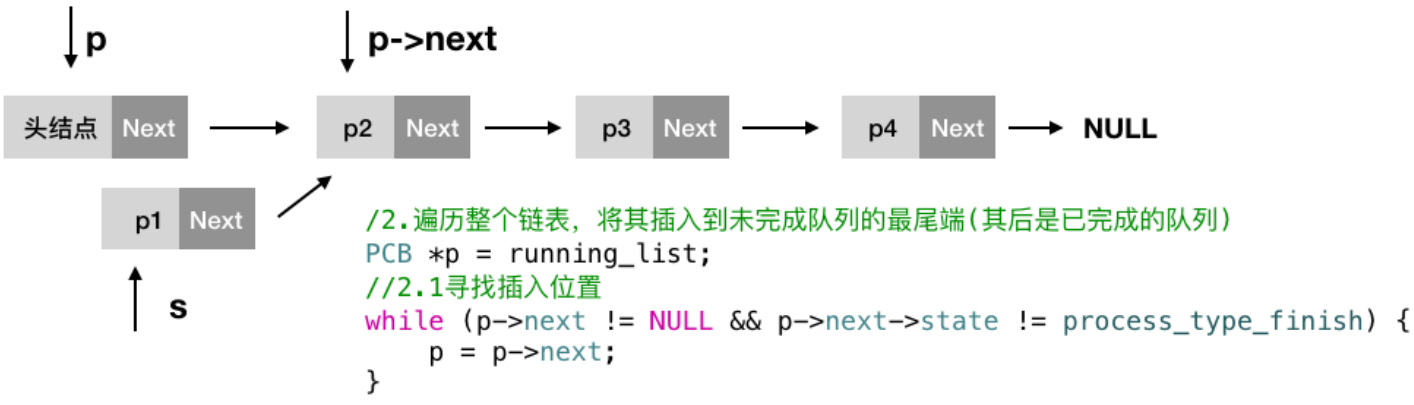
同时，因为原来的首元节点被调到了后方，它的下一个结点成为新的首元结点，我们遍历链表执行调度时只需要让指针保持指向首元结点即可。判断调度是否结束只需要判断首元结点的状态是否为已完成。若首元结点已完成，则说明它后面的结点也完成了。

对进程执行结束后的插入操作而言，并不关心进程是否已经完成，若它已经完成，则下次遍历到它前一个就会停止；没有完成，就会继续拿来执行。假设其他进程都完成后，还有一个进程需要运行多次，显然该进程开始所在的位置时首元结点，当运行完一个时间片后要为它寻找插入位置，由于首元结点往后的结点均为已完成，该结点被插入的为止仍是首元结点，然后被再次调出来执行，直到完成为止。

如图所示，当p1运行结束后，需要将p1移到队尾，首先让头指针指向p1的下一个结点，这样p1就脱离了整个链表，所以需要一个s指针记录它



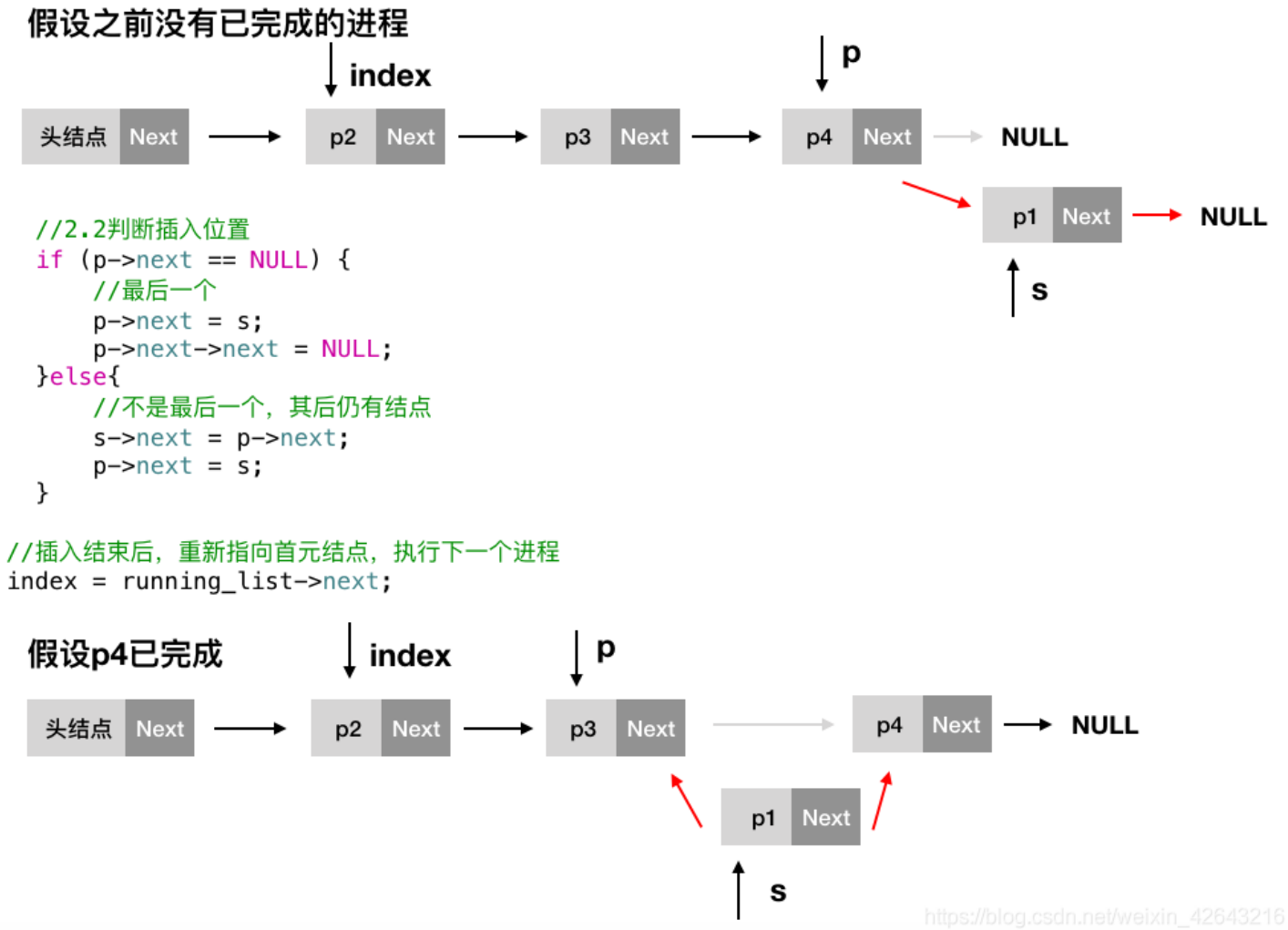
遍历整个链表为p1寻找插入位置，在p指针指向的结点之后进行插入




```
/2.遍历整个链表，将其插入到未完成队列的最尾端(其后是已完成的队列)
PCB *p = running_list;
//2.1寻找插入位置
while (p->next != NULL && p->next->state != process_type_finish) {
    p = p->next;
}
```


要插入的位置有两种情况：第一个是队列的最末尾，这说明目前还没有已经完成的进程；第二个是队列中间，p结点之后都是已完成的进程。因为链表中在尾部插入和在两个结点之间插入涉及的指针操作有区别，所以要区分对待。

这样，p1就被移到了队列最尾部，在最后还需要将inedx指针指回首元结点，运行进程时，运行的始终是index指针指向的进程。

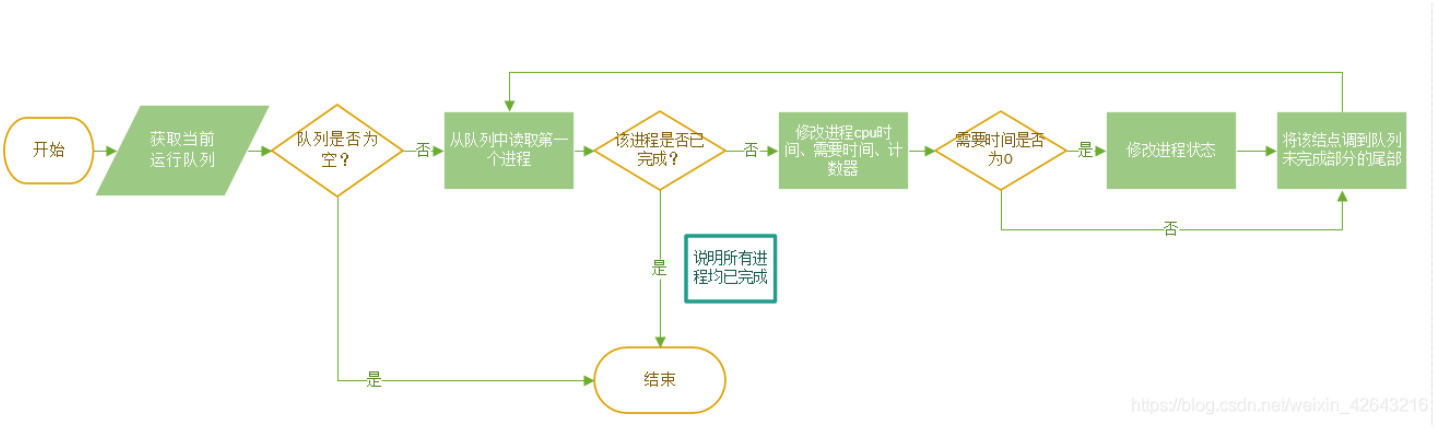



```
//2.2判断插入位置
if (p->next == NULL) {
    //最后一个
    p->next = s;
    p->next->next = NULL;
}else{
    //不是最后一个，其后仍有结点
    s->next = p->next;
    p->next = s;
}
```

```
//插入结束后，重新指向首元结点，执行下一个进程
index = running_list->next;
```

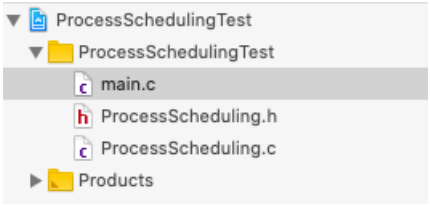

之所以说不需要考虑目前p1是否已经完成，是因为这样：当p2运行完后，用p指针遍历链表为p2寻找插入位置，假如p1没有完成，p指针仍会指到p1的位置，p2会插入到p1之后；假如p1已经完成，p指针只会指到p1的前一个结点，p2会插入到p1之前。以此类推，最先完成的进程会被沉到最底。

算法流程图



全部代码

工程图



ProcessScheduling.h

```
//
//  ProcessScheduling.h
//  ProcessSchedulingTest
//
//  Created by Apple on 2019/10/21.
//  Copyright © 2019 Yao YongXin. All rights reserved.
//

#ifndef ProcessScheduling_h
#define ProcessScheduling_h

#include <stdio.h>
#include <stdlib.h>
#include <assert.h>
#include <time.h>

#define TIME_SLICE 2

//线程状态：就绪 等待 完成
enum process_type{
    process_type_waitting = 'W',
    process_type_ready = 'R',
    process_type_finish = 'F'
};

//进程控制块结构体
typedef struct PCB_Type{
    //进程的名字
    char *name;
    //进程的优先级
    int priority;
    //仍需运行时间
    int need_time;
    //进程的状态 就绪  等待
    char state;
    //时间片
    int time_slice;
    //cpu时间 ->已运行时间
    int cpu_time;
    //计数器
    int time_count;
    //总共需要的运行时间
    int total_time;

    //下一个要执行的进程
    struct PCB_Type *next;
}PCB;

//创建新的进程
void create_process(PCB *running_list,char *name,int need_time);
//展示当前就绪队列状态
void show(PCB *running_list);

//轮转法
void round_robin(PCB *running_list);
//找到当前队列中第一个进程，将它状态变为就绪
void set_readyR(PCB *running_list);

#endif /* ProcessScheduling_h */
```

ProcessScheduling.c

```c
//
//  ProcessScheduling.c
//  ProcessSchedulingTest
//
//  Created by Apple on 2019/10/21.
//  Copyright © 2019 Yao YongXin. All rights reserved.
//

#include "ProcessScheduling.h"

//创建新的进程
void create_process(PCB *running_list,char *name,int need_time){
    //申请一个内存控制块的空间
    PCB *p = (PCB *)malloc(sizeof(PCB));
    assert(p != NULL);

    //设置该控制块的值
    p->name = name;
    p->need_time = need_time;

    //状态
    p->state = process_type_waitting;
    //时间片
    p->time_slice = 0;
    //cpu时间
    p->cpu_time = 0;
    //计数器
    p->time_count = 0;
    //总需用时
    p->total_time = need_time;

    //默认优先级一致
    p->priority = 1;

    //下个进程
    p->next = NULL;

    //放入运行队列中
    PCB *s = running_list;
    while (s->next != NULL) {
        s = s->next;
    }
    s->next = p;

}

//展示当前就绪队列状态
void show(PCB *running_list){
    PCB *p = running_list->next;
    if (p == NULL) {
        printf("当前队列中无进程\n");
        return;
    }

    printf("进程名  优先级  时间片  cpu时间  需要时间  进程状态  计数器\n");
    while (p != NULL) {
        printf("%s    %4d   %4d   %4d     %4d       %c    %4d\n",p->name,p->priority,p->time_slice,p->cpu_time,p
->need_time,p->state,p->time_count);
```

```
>need_time,p->state,p->time_count);

        p = p->next;
    }
    printf("\n");
}

//轮转法
void round_robin(PCB *running_list){
    /*
     每次运行完进程后，会将该进程从队首调到队尾
     即下一次运行的仍是链表第一个结点的进程
     最先完成的进程处于最末尾
     */
    //记录第一个结点的位置
    PCB *index = running_list->next;
    if (index == NULL) {
        printf("当前队列已空\n");
        return;
    }


    while (index != NULL && index->state != process_type_finish) {
        //按时间片运行该进程，即修改进程的cpu时间和需要时间、计数器
        PCB *s = index;
        s->time_slice = TIME_SLICE;

        //cpu时间（即已运行时间) = 总需时间 -（当前cpu时间+时间片)
        //若已完成则直接显示总需时间
        s->cpu_time = (s->cpu_time + TIME_SLICE)<s->total_time?s->cpu_time + TIME_SLICE:s->total_time;
        //若当前仍需时间减时间片小于等于零，则说明进程已完成
        s->need_time = (s->need_time - TIME_SLICE)>0?s->need_time - TIME_SLICE:0;
        //计数器+1
        s->time_count += 1;

        //判断该进程是否结束
        if (s->need_time == 0) {
            //修改进程状态
            s->state = process_type_finish;
        }else{
            s->state = process_type_waitting;
        }

        //将该进程调到队尾
        //1.头指针指向首元结点的下一个结点
        running_list->next = s->next;

        //2.遍历整个链表，将其插入到未完成队列的最尾端(其后是已完成的队列)
        PCB *p = running_list;
        //2.1寻找插入位置
        while (p->next != NULL && p->next->state != process_type_finish) {
            p = p->next;
        }
        //2.2判断插入位置
        if (p->next == NULL) {
            //最后一个
            p->next = s;
            p->next->next = NULL;
        }else{
            //不是最后一个，其后仍有结点
```

```
            s->next = p->next;
            p->next = s;
        }

        //重新指向首元结点
        index = running_list->next;
        set_readyR(running_list);
        //展示当前队列状况
        show(running_list);
    }
}
//找到当前队列中第一个进程，将它状态变为就绪
void set_readyR(PCB *running_list){
    PCB *s = running_list->next;
    if (s == NULL) {
        printf("当前队列已空\n");
        return;
    }

    if (s->state != process_type_finish) {
        s->state = process_type_ready;
        return;
    }
}
```

Main.c

```c
//
//  main.c
//  ProcessSchedulingTest
//
//  Created by Apple on 2019/10/21.
//  Copyright © 2019 Yao YongXin. All rights reserved.
//

#include "ProcessScheduling.h"

int main(int argc, const char * argv[]) {

    //运行(就绪)队列(头结点不储存信息)
    PCB *running_list = (PCB *)malloc(sizeof(PCB));
    running_list->next = NULL;

    int p_number;
    printf("请输入要创建的进程数目:\n");
    scanf("%d",&p_number);


    printf("请输入进程名字和所需时间:\n");
    for (int i = 0; i < p_number; i++) {
        //create(running_list);
        char *name = (char *)malloc(sizeof(char));
        int time;
        scanf("%s %d",name,&time);
        create_process(running_list, name, time);
    }

    //轮转法
    set_readyR(running_list);
    printf("调度前:\n");
    show(running_list);
    printf("调度后:\n");
    round_robin(running_list);

    return 0;
}
```

结果截图

```
请输入要创建的进程数目:
3
请输入进程名字和所需时间:
p1 5
p2 3
p3 4
调度前:
进程名  优先级  时间片  cpu时间  需要时间  进程状态  计数器
p1       1       0       0        5        R        0
p2       1       0       0        3        W        0
p3       1       0       0        4        W        0

调度后:
进程名  优先级  时间片  cpu时间  需要时间  进程状态  计数器
p2       1       0       0        3        R        0
p3       1       0       0        4        W        0
p1       1       2       2        3        W        1

进程名  优先级  时间片  cpu时间  需要时间  进程状态  计数器
p3       1       0       0        4        R        0
p1       1       2       2        3        W        1
p2       1       2       2        1        W        1

进程名  优先级  时间片  cpu时间  需要时间  进程状态  计数器
p1       1       2       2        3        R        1
p2       1       2       2        1        W        1
p3       1       2       2        2        W        1

进程名  优先级  时间片  cpu时间  需要时间  进程状态  计数器
p2       1       2       2        1        R        1
p3       1       2       2        2        W        1
p1       1       2       4        1        W        2

进程名  优先级  时间片  cpu时间  需要时间  进程状态  计数器
p3       1       2       2        2        R        1
p1       1       2       4        1        W        2
p2       1       2       3        0        F        2

进程名  优先级  时间片  cpu时间  需要时间  进程状态  计数器
p1       1       2       4        1        R        2
p3       1       2       4        0        F        2
p2       1       2       3        0        F        2

进程名  优先级  时间片  cpu时间  需要时间  进程状态  计数器
p1       1       2       5        0        F        3
p3       1       2       4        0        F        2
p2       1       2       3        0        F        2

Program ended with exit code: 0
```