华南理工大学

实 验 报 告

课程名称：	操 作 系 统
学生姓名：	陈卓文
学生学号：	**201936380215**
学生专业：	软件工程
开课学期：	**2020-2021 第二学期**

2021 年 04 月

实验三 进程调度算法模拟实验

地　　点： B7 楼 331 房； **评　　分：**

实验日期与时间： 2021.04.18 13：50-17：25 **实验教师：**

一、 实验目标

1. 通过模拟单处理器进程调度算法，加深对进程调度的概念理解。

二、 实验内容

1. 设计一个按时间片轮转法实现进程调度的模拟程序。

三、 实验设备及环境

PC （ubuntu 操作系统）；C/C++等编程语言。

四、 实验主要步骤

1. 假设系统有 10 个进程，每个进程用一个进程控制块（PCB）来代表。进程控制块的格式如图 1 所示。

图 1 进程控制块结构

图中的参数意义如下：

进程名：即进程标识。

链接指针：按照进程到达系统的时间将处于就绪状态的进程连接成一个就绪队列。指针指出下一个到达的进程控制块地址。最后一个进程的链接指针为 NULL。

到达时间：进程创建时的系统时间或由用户指定，调度时，总是选择到达时间最早的进程。

估计运行时间：可由设计者任意指定一个时间值。

进程状态：为简单起见，这里假定进程有两种状态：就绪态和完成态。就绪状态用“R”表示，完成状态用“C”表示。假定进程一创建就处于就绪状态，运行结束时，就被置成完成状态。

```
enum State {R,C};
typedef struct PCB{
    char name; //  进程名:以单个字符简化
    int startTime; //  到达时间
    int runTime; //  估计运行时间
    enum State processState; //  进程状态
    struct PCB* next; //  链接指针
}PCB;
```

2. 按照进程到达的先后顺序排成一个**循环队列**，设一个队首指针指向第一个到达进程的首址。另外再设一个当前运行的进程指针，指向当前正运行的进程。

注：循环队列在本题中不能使用 queue，只能使用链表。

3. 执行进程调度时，首先选择队首的第一个进程运行。

4. 由于本实验是模拟实验，所以对被选中的进程并不实际启动运行，而只是执行：估计运行时间减 1，rumTime -= 1，输出当前运行进程的名字。用这个操作来模拟进程的一次运行。

5. 进程运行一次后，以后的调度则将当前指针依次下移一个位置，指向下一个进程，即调整当前运行指针指向该进程的链接指针所指进程，以指示应运行进程，同时还应判断该进程的剩余运行时间是否为 0。若不为 0，则等待下一轮的运行；若该进程的剩余运行时间为 0，则将该进程的状态置为完成状态“C”，并退出循环队列。

6. 若就绪队列不空，则重复上述步骤 4 和 5 直到所有进程都运行完为止。

7. 在所设计的调度程序中，应包含显示或打印语句，以便显示或打印每次选中进程的名称及运行一次后队列的变化情况。

8. 在完成 1-7 后，请添加一个函数 add(PCB* ptr)向队列尾新添加一个新的进程，让其加入执行，并观察打印输出。

9. 请重构 add 函数，保证加入的新进程立刻在下一个时间片轮转时优先执行进行。请思考 2 种解决方案并加以实现。

10. 在完成 1-9 后，请添加三个函数 wait(PCB* ptr)，notify_one(PCB* ptr)和 notify_all(PCB* ptr)。wait 函数指定某一个进程进入等待，时间片轮转跳过该进程继续执行；notify_one 为唤醒某一个指定进程，在下次时间片轮转时执行；notify_all 为唤醒全部的 wait 进程，均在下次时间片轮转时执行。

提示：可以修改 State 进程状态，加入等待状态。可以为等待进程创建单独的记录表方便更改。

五、 问题与算法

1. 问题描述

本次实验的是通过模拟调度进程块，从而加深对进程调度的概念理解。

这里采用的进程调度方法为轮转法，即进程无优先级，轮流执行一个时间片，并且一个时间片为1。

2. 算法思路

为了实现PCB的轮转，需要使用单向循环链表储存。并且使用一个指针指向队首。再设一个指针指向当前正运行的进程。

第8步与第9步的add函数，实际上是单项循环链表的插入。

第10步，通过增加一个新的State从而新增一个等待状态

3. 算法实现关键点

在模拟运行的时候，同时也要记录当前运行进程的前一个 PCB，此举是为了实现单向循环链表的插入、删除。

单项循环链表的插入，只是插入的位置不一样而已，所以可以通过将插入的 ptr 赋值给 pre 与 cur 实现插入到队尾和队首。

新增的阻塞态与其相关操作只是操作自身或者遍历整个循环链表。

六、 实验结果与分析

1. 实验数据及结果

(1) 第一步 ~ 第七步

```
simon@Aliyun:~/code/study$ gcc pcb_7.c && ./a.out
a 2 R <- next running
b 1 R
c 1 R

a 1 R
b 1 R <- next running
c 1 R

a 1 R
c 1 R <- next running

a 1 R <- next running
```

图1 第七步及之前的程序输出

(2) 第八步

在这里，每隔5个时间块，就随机插入下一个PCB

```
simon@Aliyun:~/code/study$ gcc pcb_8.c && ./a.out
add: d 2
a 2 R <- next running
b 1 R
c 1 R
d 2 R

a 1 R
b 1 R <- next running
c 1 R
d 2 R

a 1 R
c 1 R <- next running
d 2 R

a 1 R
d 2 R <- next running

add: e 1
a 1 R <- next running
d 1 R
e 1 R

d 1 R <- next running
e 1 R

e 1 R <- next running
```

图2 第八步的程序输出

(3) 第九步

这一步，与上一步大致相同，除了插入的进程将会被立即运行

```
simon@Aliyun:~/code/study$ gcc pcb_9.c && ./a.out
add: d 2
a 2 R
b 1 R
c 1 R
d 2 R <- next running

a 2 R <- next running
b 1 R
c 1 R
d 1 R

a 1 R
b 1 R <- next running
c 1 R
d 1 R

a 1 R
c 1 R <- next running
d 1 R

add: e 1
a 1 R
e 1 R <- next running
d 1 R

a 1 R
d 1 R <- next running

a 1 R <- next running
```

图3 第九步的程序输出

(4) 第十步

这一步，添加了wait和notify函数，添加了W状态代表处于等待状态

```
simon@Aliyun:~/code/study$ gcc pcb_10.c && ./a.out
make a waited
add: d 2
a 2 W
b 1 R
c 1 R
d 2 R <- next running

a 2 W <- next running
b 1 R
c 1 R
d 1 R
process in a wait state

a 2 W
b 1 R <- next running
c 1 R
d 1 R

notify all process
a 2 R
c 1 R <- next running
d 1 R

a 2 R
d 1 R <- next running

add: e 1
a 2 R
e 1 R <- next running

a 2 R <- next running

a 1 R <- next running
```

图4 第十步的程序输出

2. 实验分析及结论

本次实验成功的模拟了实验步骤中描述的功能，实现了轮换式的进程调度模拟。

七、 心得与展望

1. 自我评价及心得体会

本次实验个人觉得难点在于实现单向循环列表的实现，及其插入和删除。以及其中伴随的指针操作。但是仍然成功的实现了所需功能。

2. 展望

能够尝试实际编写操作系统中的进程调度。

八、 附录

1. 主要界面

Ubuntu 18.04 + Vscode 1.55.2 + Gcc 7.5.0

2. 源程序

```
Exp3
├─── pcb_7.c
├─── pcb_8.c
├─── pcb_9.c
└─── pcb_10.c
```

```
// pcb_7.c step 1~7
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

const int TOTAL_PCB = 3;
const int MAX_RUNTIME = 2;

enum State { R, C };
typedef struct PCB {
    char name;
    int startTime;
```

```
    int runTime;
    enum State processState;
    struct PCB* next;
} PCB;

PCB* create_data() {
    PCB* p = malloc(sizeof(PCB));
    PCB* res = p;
    PCB* pre;
    unsigned int seed = 10;
    srand(seed);
    for (int i = 0; i < TOTAL_PCB; i++) {
        p->name = 'a' + i;
        p->startTime = i;
        p->runTime = rand() % MAX_RUNTIME + 1;
        p->processState = R;
        p->next = malloc(sizeof(PCB));
        pre = p;
        p = p->next;
    }
    pre->next = res;       // 循环队列
    return res;
}

void show_list(const PCB* const lst, PCB* cur) {
    if (lst) {
        const PCB* p = lst;
        do {
            printf("%c %d %c %s\n",
                    p->name,
                    p->runTime,
                    p->processState == R ? 'R' : 'C',
                    p == cur ? "<- next running" : "");
            p = p->next;
```

```
            } while (p != lst);
        }
        else {
            printf("there is no PCB in list\n");
        }
}

PCB* get_endp(PCB* lst) {
        PCB* p = lst;
        while (p->next != lst) p = p->next;
        return p;
}
void add(PCB* lst, PCB* ptr);

void run_until_complete(PCB** lstp) {
        PCB* p = *lstp;
        PCB* pre = get_endp(*lstp);
        while (*lstp != NULL) {
            // print list & execute PCB
            show_list(*lstp, p);        // print detail of list
            (p->runTime)--;
            printf("\n");

            if (!(p->runTime)) {
                // remove completed PCB
                p->processState = C;
                if (p->next != p) {
                    if (*lstp == p) *lstp = p->next;
                    p = p->next;
                    free(pre->next);
                    pre->next = p;
                }
                else {
                    // only one PCB in list
```

```
                        free(p);
                        *lstp = NULL;
                    }
                }
                else {
                    pre = p;
                    p = p->next;
                }
        }
}

int main() {
        PCB* lst = create_data();
        run_until_complete(&lst);
        return 0;
}
```

```
// pcb_8.c step 8
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

const int TOTAL_PCB = 3;
const int MAX_RUNTIME = 2;
const int ADD_STEP = 4;

enum State { R, C };
typedef struct PCB {
        char name;
        int startTime;
        int runTime;
        enum State processState;
        struct PCB* next;
```

```
} PCB;

PCB* create_data() {
	PCB* p = malloc(sizeof(PCB));
	PCB* res = p;
	PCB* pre;
	unsigned int seed = 10;
	srand(seed);
	for (int i = 0; i < TOTAL_PCB; i++) {
		p->name = 'a' + i;
		p->startTime = i;
		p->runTime = rand() % MAX_RUNTIME + 1;
		p->processState = R;
		p->next = malloc(sizeof(PCB));
		pre = p;
		p = p->next;
	}
	pre->next = res;	// 循环队列
	return res;
}

void show_list(const PCB* const lst, PCB* cur) {
	if (lst) {
		const PCB* p = lst;
		do {
			printf("%c %d %c %s\n",
				p->name,
				p->runTime,
				p->processState == R ? 'R' : 'C',
				p == cur ? "<- next running" : "");
			p = p->next;
		} while (p != lst);
	}
	else {
```

```
            printf("there is no PCB in list\n");
        }
}

PCB* get_endp(PCB* lst) {
        PCB* p = lst;
        while (p->next != lst) p = p->next;
        return p;
}

PCB* add(PCB* lst, PCB* ptr);

void run_until_complete(PCB** lstp) {
        PCB* p = *lstp;
        PCB* pre = get_endp(*lstp);
        int cnt = 0;
        while (*lstp != NULL) {
                // step:8 every ten times add a new PCB
                if (cnt % ADD_STEP == 0) {
                        PCB* tmp = malloc(sizeof(PCB));
                        tmp->name = 'a' + cnt / ADD_STEP + TOTAL_PCB;
                        tmp->startTime = cnt + TOTAL_PCB;
                        tmp->runTime = rand() % MAX_RUNTIME + 1;
                        tmp->processState = R;
                        pre = add(p, tmp);
                }

                // print list & execute PCB
                show_list(*lstp, p);        // print detail of list
                (p->runTime)--;
                printf("\n");
                cnt++;

                if (!(p->runTime)) {
```

```
                // remove completed PCB
                p->processState = C;
                if (p->next != p) {
                        if (*lstp == p) *lstp = p->next;
                        p = p->next;
                        free(pre->next);
                        pre->next = p;
                }
                else {
                        // only one PCB in list
                        free(p);
                        *lstp = NULL;
                }
        }
        else {
                pre = p;
                p = p->next;
        }
    }
}

// step: 8
PCB* add(PCB* lst, PCB* ptr) {
    printf("add: %c %d\n", ptr->name, ptr->runTime);
    PCB* p = get_endp(lst);
    ptr->next = p->next;
    p->next = ptr;
    return ptr;
}

int main() {
    PCB* lst = create_data();
    run_until_complete(&lst);
    return 0;
```

```
}
```

```
// pcb_9.c step 9
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

const int TOTAL_PCB = 3;
const int MAX_RUNTIME = 2;
const int ADD_STEP = 4;

enum State { R, C };
typedef struct PCB {
	char name;
	int startTime;
	int runTime;
	enum State processState;
	struct PCB* next;
} PCB;

PCB* create_data() {
	PCB* p = malloc(sizeof(PCB));
	PCB* res = p;
	PCB* pre;
	unsigned int seed = 10;
	srand(seed);
	for (int i = 0; i < TOTAL_PCB; i++) {
		p->name = 'a' + i;
		p->startTime = i;
		p->runTime = rand() % MAX_RUNTIME + 1;
		p->processState = R;
		p->next = malloc(sizeof(PCB));
		pre = p;
```

```
            p = p->next;
        }
        pre->next = res;        //  循环队列
        return res;
}

void show_list(const PCB* const lst, PCB* cur) {
        if (lst) {
                const PCB* p = lst;
                do {
                        printf("%c %d %c %s\n",
                                        p->name,
                                        p->runTime,
                                        p->processState == R ? 'R' : 'C',
                                        p == cur ? "<- next running" : "");
                        p = p->next;
                } while (p != lst);
        }
        else {
                printf("there is no PCB in list\n");
        }
}

PCB* get_endp(PCB* lst) {
        PCB* p = lst;
        while (p->next != lst) p = p->next;
        return p;
}

PCB* add(PCB* lst, PCB* ptr);

void run_until_complete(PCB** lstp) {
        PCB* p = *lstp;
        PCB* pre = get_endp(*lstp);
```

```
int cnt = 0;
while (*lstp != NULL) {
    // step:8 every ten times add a new PCB
    if (cnt % ADD_STEP == 0) {
        PCB* tmp = malloc(sizeof(PCB));
        tmp->name = 'a' + cnt / ADD_STEP + TOTAL_PCB;
        tmp->startTime = cnt + TOTAL_PCB;
        tmp->runTime = rand() % MAX_RUNTIME + 1;
        tmp->processState = R;
        // step 9: run immediately
        p = add(p, tmp);
    }

    // print list & execute PCB
    show_list(*lstp, p);       // print detail of list
    (p->runTime)--;
    printf("\n");
    cnt++;

    if (!(p->runTime)) {
        // remove completed PCB
        p->processState = C;
        if (p->next != p) {
            if (*lstp == p) *lstp = p->next;
            p = p->next;
            free(pre->next);
            pre->next = p;
        }
        else {
            // only one PCB in list
            free(p);
            *lstp = NULL;
        }
    }
```

```
            else {
                pre = p;
                p = p->next;
            }
        }
}

// step: 8
PCB* add(PCB* lst, PCB* ptr) {
        printf("add: %c %d\n", ptr->name, ptr->runTime);
        PCB* p = get_endp(lst);
        ptr->next = p->next;
        p->next = ptr;
        return ptr;
}

int main() {
        PCB* lst = create_data();
        run_until_complete(&lst);
        return 0;
}
```

```
// pcb_10.c step 10
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

const int TOTAL_PCB = 3;
const int MAX_RUNTIME = 2;
const int ADD_STEP = 4;

enum State { R, C, W };
typedef struct PCB {
```

```
    char name;
    int startTime;
    int runTime;
    enum State processState;
    struct PCB* next;
} PCB;

PCB* create_data() {
    PCB* p = malloc(sizeof(PCB));
    PCB* res = p;
    PCB* pre;
    unsigned int seed = 10;
    srand(seed);
    for (int i = 0; i < TOTAL_PCB; i++) {
        p->name = 'a' + i;
        p->startTime = i;
        p->runTime = rand() % MAX_RUNTIME + 1;
        p->processState = R;
        p->next = malloc(sizeof(PCB));
        pre = p;
        p = p->next;
    }
    pre->next = res;       // 循环队列
    return res;
}

void show_list(const PCB* const lst, PCB* cur) {
    if (lst) {
        const PCB* p = lst;
        do {
            printf("%c %d %c %s\n",
                    p->name,
                    p->runTime,
                    p->processState == R ? 'R' : (p->processState == C ? 'C' : 'W'),
```

```
                        p == cur ? "<- next running" : "");
                p = p->next;
            } while (p != lst);
        }
        else {
            printf("there is no PCB in list\n");
        }
}

PCB* get_endp(PCB* lst) {
        PCB* p = lst;
        while (p->next != lst) p = p->next;
        return p;
}

PCB* add(PCB* lst, PCB* ptr);
void wait(PCB* ptr);
void notify_one(PCB* ptr);
void notify_all(PCB* ptr);

void run_until_complete(PCB** lstp) {
        PCB* p = *lstp;
        PCB* pre = get_endp(*lstp);
        int cnt = 0;
        while (*lstp != NULL) {
            // step:8 every ten times add a new PCB
            if (cnt % ADD_STEP == 0) {
                PCB* tmp = malloc(sizeof(PCB));
                tmp->name = 'a' + cnt / ADD_STEP + TOTAL_PCB;
                tmp->startTime = cnt + TOTAL_PCB;
                tmp->runTime = rand() % MAX_RUNTIME + 1;
                tmp->processState = R;
                // step 9: run immediately
                p = add(p, tmp);
```

```
}

// print list & execute PCB
show_list(*lstp, p);        // print detail of list
if (p->processState == R) {
        (p->runTime)--;
        cnt++;
}
else {
        printf("process in a wait state\n");
}
printf("\n");

if (!(p->runTime)) {
        // remove completed PCB
        p->processState = C;

        // step:10 awake A if B done
        if (p->name == 'b') notify_all(p);

        if (p->next != p) {
                if (*lstp == p) *lstp = p->next;
                p = p->next;
                free(pre->next);
                pre->next = p;
        }
        else {
                // only one PCB in list
                free(p);
                *lstp = NULL;
        }
}
else {
        pre = p;
```

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

// step: 8
PCB* add(PCB* lst, PCB* ptr) {
        printf("add: %c %d\n", ptr->name, ptr->runTime);
        PCB* p = get_endp(lst);
        ptr->next = p->next;
        p->next = ptr;
        return ptr;
}

// step: 10
// add State W
void wait(PCB* ptr) {
        printf("make %c waited\n", ptr->name);
        if (ptr->processState == R) ptr->processState = W;
}
void notify_one(PCB* ptr) {
        printf("notify %c\n", ptr->name);
        if (ptr->processState == W) ptr->processState = R;
}
void notify_all(PCB* ptr) {
        printf("notify all process\n");
        PCB* p = ptr;
        do {
            if (p->processState == W) p->processState = R;
            p = p->next;
        } while (p != ptr);
}

int main() {
```

```
    PCB* lst = create_data();
    // make A waited
    wait(lst);
    run_until_complete(&lst);
    return 0;
}
```

九、 参考文献

[1] 《计算机操作系统教程》张尧学等，清华大学出版社，2006 年 10 月第 3 版