实验一实验报告

实验源码：

```
#include <stdlib.h>
#include <string.h>
#include <conio.h>
#include <iostream.h>
#include <windows.h>

// 内存最大装备作业数
#define N 5

#define P_TIME 50

struct PCB{
    int pid;            // 进程 id
    int priority;     // 优先权
    int size;           // 大小
    int state;          // 进程状态    0 表示不存在      1 表示运行状态       2 表示
阻塞状态
    char info[10];   // 消息
};

struct PCB neicun[N];

int shumu = 0, guaqi = 0;
int pid;
int flag = 0;          // 查找相应进程是否存在的标志

// 进程调度

enum state{
    ready,
    working,
    block,
    finish
};//定义进程状态

struct D_PCB{
    char name[4];    // 进程名
    int priority;    // 优先权
```

```
        int cputime;      // CPU 运行时间
        int needtime;     // 进程运行所需时间
        int count;        // 进程执行次数
        int round;        // 时间片轮转轮次
        state process; //   进程状态
        D_PCB *next;
};//定义进程 PCB


// 创建进程
void create()
{
                                内存已满，请先换出或杀死进程
else{
    for(int i = 0; i < N; i++)
        // 定位，找到还未创建的进程
        if(neicun[i].state == 0) break;
            请输入新进程

    for(int j = 0; j < i; j++)
        if(neicun[j].pid == neicun[i].pid){
                            该进程已存在
            return;
        }
            请输入新进程优先级

            请输入新进程大小

            请输入新进程内容

            // 创建进程，使标记位为 1
            neicun[i].state = 1;
            shumu++;
    }
}

// 查看运行进程
void run()
{


    for(int i = 0; i < N; i++){
        if(neicun[i].state == 1){
```

```
                // 输出运行进程的各个属性值


eicun[i].state,neicun[i].info);
                flag = 1;
            }
        }
                            当前没有运行进程
}

// 换出进程
void huanchu()
{
    if(!shumu){
                当前没有运行进程
        return;
    }
            输入换出运行进程 ID值

    for(int i = 0; i < N; i++){
        // 定位，找到所要换出的进程，根据其状态做相应处理
        if(pid == neicun[i].pid){
            if(neicun[i].state == 1){
                neicun[i].state = 2;
                guaqi++;
                            已经成功换出进程
            }
                                                    要换出的进程不存在
                            换出的进程已被挂起
            flag = 1;
                break;
            }
        }
        // 找不到，则说明不存在
                                要换出的进程不存在
}

// 杀死进程
void kill()
{

    if(!shumu){
                当前没有运行进程
```

```
        return;
}
                输入杀死进的的 ID值

for(int i = 0; i < N; i++){
        // 定位，找到所要杀死的进程，根据其状态做相应处理
        if(pid == neicun[i].pid){
                if(neicun[i].state == 1){
                        neicun[i].state = 0;
                        shumu--;
                                        已成功杀死进程
                }
                                                        要杀死的进程不存在
                                        要杀死的进程已被挂起
                        flag = 1;
                        break;
                }
        }
        // 找不到，则说明进程不存在
                                要杀死的进程不存在
}

// 唤醒进程
void huanxing()
{

        if(!shumu){
                        当前没有运行进程
                return;
        }
        if(!guaqi){
                        当前没有挂起进程
        return;
}
                输入 pid：

for(int i = 0; i < N; i++){
        // 定位，找到所要杀死的进程，根据其状态做相应处理
        if(pid == neicun[i].pid){
                flag = 0;
                if(neicun[i].state = 2){
                        neicun[i].state = 1;
                        guaqi--;
                                        已经成功唤醒进程
```

```
                } else if (neicun[i].state == 0)
                                要唤醒的进程不存在
                break;
            }
        }
        // 找不到，则说明进程不存在
                        要唤醒的进程不存在
}

// 进程通信
void tongxin()
{
        int ps,pr;



        for(int i = 0; i < N; i++)
        {
            for(int j = 0; j < N; j++)
            {
                if((neicun[i].pid == ps) && (neicun[j].pid == pr))
                    strcpy(neicun[j].info, neicun[i].info);
            }
        }
}

// 进程调度

// 优先权调度算法
// 输入模拟测试的进程名和执行所需时间，初始设置可模拟5个进程的调度
D_PCB *get_process(){
        D_PCB *q;
        D_PCB *t;
        D_PCB *p;
        int i = 0;
        int P_NUM;



        while(i < P_NUM){
            q = (struct D_PCB *)malloc(sizeof(D_PCB));
```

```
q->cputime = 0;
q->priority = P_TIME - q->needtime;
q->process = ready;
q->next = NULL;
if(i == 0){
	p = q;
	t = q;
}else{
	t->next = q; //  创建就绪进程队列
	t = q;
}
	i++;
	}// end while
	return p;
}

// 显示模拟结果，包含进程名、CPU时间、运行所需时间以及优先级
void display(D_PCB *p){

	while(p){


		switch(p->process){
		case ready:

			break;
		case working:

			break;
		case block:

			break;
		case finish:

			break;
		}
		p = p->next;
	}
}

// 结束进程，即将队列中各进程所需时间设置为 0
int process_finish(D_PCB *q){
```

```
    int bl = 1;
    while(bl && q){
        bl = bl && q->needtime == 0;
        q = q->next;
    }
    return bl;
}

// 选择某一进程，给它分配 CPU
void cpuexe(D_PCB *q){
    D_PCB *t = q;
    int tp = 0;
    while(q){
    if(q->process != finish){
        q->process = ready;
        if(q->needtime == 0){
            q->process = finish;
        }
    }
    if(tp < q->priority && q->process != finish){
        tp = q->priority;
        t = q;
    }
    q = q->next;
}
if(t->needtime != 0){
    t->priority -= 3;
    t->needtime--;
    t->process = working;
    t->cputime++;
}

// 计算进程优先级
void priority_cal(){
    D_PCB * p;

    p = get_process();
    int cpu = 0;

    while(!process_finish(p)){
        cpu++;

        cpuexe(p);
```

```
            display(p);
            Sleep(2);
        }

}

// 显示调度算法菜单，可供用户选择优先权调度算法和时间片轮转算法
void display_menu(){

                        CHOOSE THE ALGORITHM:
                                                                    // 优先权调度算法
                                                                      // 时间片轮转算法

}

// 时间片轮转算法
// 时间片轮转调度算法创建就绪进程队列
D_PCB * get_process_round(){
        D_PCB *q;
        D_PCB *t;
        D_PCB *p;
        int i = 0;

        int P_NUM;




        while(i < P_NUM){
q = (struct D_PCB *)malloc(sizeof(D_PCB));


q->cputime = 0;
q->round = 0;
q->count = 0;
q->process = ready;
q->next = NULL;
if(i == 0){
        p = q;
        t = q;
}else{
        t->next = q;
```

```
                t = q;
            }
            i++;
        }// end while
        return p;
}

// 采用时间片轮转算法执行某一进程
void cpu_round(D_PCB *q){
        q->cputime += 2;
        q->needtime -= 2;
        if(q->needtime < 0){
            q->needtime = 0;
        }
        q->count++;
        q->round++;
        q->process = working;
}

// 获取下一个进程
D_PCB * get_next(D_PCB *k, D_PCB *head){
        D_PCB *t;
        t = k;
        do{
            t = t->next;
        }while(t && t->process == finish);
        if(t == NULL){
            t = head;
            while(t->next != k && t->process == finish){
                t = t->next;
            }
        }
        return t;
}

// 设置队列中进程执行状态
void set_state(D_PCB *p){
        while(p){
            if(p->needtime == 0){
                p->process = finish; // 如果所需执行时间为0，则设置运行状态为结
束
            }
            if(p->process == working){
                p->process = ready; // 如果所需执行状态则设置为就绪
```

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

//
void display_round(D_PCB *p){

        while(p){


->round);
            switch(p->process){
            case ready:

                break;
            case working:

                break;
            case block:

                break;
            case finish:

                break;
            }
            p = p->next;
        }
}

// 时间片轮转调度算法计算轮次及输出调度信息
void round_cal(){
        D_PCB * p;
        D_PCB *r;

        p = get_process_round();
        int cpu = 0;

        r = p;
        while(!process_finish(p)){
            cpu+=2;
            cpu_round(r);
            r = get_next(r,p);
```

```
            display_round(p);
            set_state(p);
            Sleep(2);
        }
}

void return_menu();

//
void is_return_menu()
{

    char k;
    k = getch();
    switch(k){
    case 'Y':
        return_menu();
        break;
    case 'N':
        break;
    case 'y':
        return_menu();
        break;
    case 'n':
        break;
    default:
        is_return_menu();
        break;
    }
}

// 返回主菜单
void return_menu()
{
    display_menu();
    int k;

    switch(k){
    case 1:
        priority_cal();
        is_return_menu();
        break;
    case 2:
        round_cal();
```

以上内容仅为本文档的试下载部分，为可阅读页数的一半内容。如要下载或阅读全文，请访问：https://d.book118.com/727145051126006135