

西安科技大学

**操作系统**

**《进程的表示》实验报告**

**学院：** **计算机科学与技术学院**

**专业：** **网络工程系**    
 **班级：** **网络工程1801班**

**姓名：** **吴斌**

**学号：** **18408020129**

**日期：**  **2020年10月**

**进程的表示**

一、实验目的说明：

通过编写模拟程序,理解进程在内存中的表示方

法以及进程的状态转换。

二、实验内容和方法：

引入的头文件process.h:

#ifndef \_PROCESS\_ //条件编译,检查该常量是否被宏定义过

#define \_PROCESS\_ //宏定义,定义process这个常量

enum STATUS {RUNNING,BLOCKED,READY};

//枚举类型，列举出状态的可能值。

enum PRIORITY {LOW,NORMAL,HIGH};

//枚举类型，列举各种优先级。

struct ProcessNode //创建processnode这个结构体，包括pid，状态，优先级，下一个进程等变量。

{

int pid;

enum STATUS status;

enum PRIORITY priority;

struct ProcessNode\* next;

};

void input\_processes(); //引用输入函数，并已在下文定义该函数

void output\_processes(); //引用输出函数，同上。

void FCFS\_Scheduling(); //做函数声明

#endif //固定格式，与#ifndef共同出现。

源文件process.c:

#include "Process.h" //引用之前写好的头文件process.h

struct ProcessNode\* running\_head=0;

struct ProcessNode\* running\_tail=0;

struct ProcessNode\* blocked\_head=0;

struct ProcessNode\* blocked\_tail=0;

struct ProcessNode\* ready\_head=0;

struct ProcessNode\* ready\_tail=0;

void ready\_insert(struct ProcessNode\* p); //声明就绪状态的函数。

void blocked\_insert(struct ProcessNode\*p);

//声明堵塞状态下的函数。

void running\_insert(struct ProcessNode\*p);

//声明运行状态下的函数。

void list\_insert(struct ProcessNode\*\* head, struct ProcessNode\*\* tail, struct ProcessNode\* p); //作用同上，但函数不同

int exist(struct ProcessNode\*p); //作用同上，但函数不同

int list\_exist(struct ProcessNode\* head, struct ProcessNode\* p);

//作用同上，但函数不同

void insert\_process(struct ProcessNode\* p); //作用同上，但函数不同

void input\_one\_process(); //作用同上，但函数不同

void list\_print(struct ProcessNode\* head); //作用同上，但函数不同

void insert\_process(struct ProcessNode\* p) //开始定义函数

{

if(p==0) //p为0时，用户不可以插入一个空进程

{

printf("error! you can't insert an empty process!\n");

return;

}

if(exist(p)) //新进程id已存在，则打印下列内容。

{

printf("there is already a process with the same ID in the system\n");

return;

}

if(!(p->priority==LOW||p->priority==NORMAL||p->priority==HIGH)) //进程优先级无效

{

printf("error! the process priority is invalid!\n");

return;

}

switch(p->status) //优先级和状态都有效，则使用switch语句进行选择跳转

{

case(RUNNING): //运行状态

running\_insert(p);

break;

case(BLOCKED): //堵塞状态

blocked\_insert(p);

break;

case(READY): //就绪状态

ready\_insert(p);

break;

default: //三种都不是，则状态无效

printf("error! the process status is invalid!\n");

break;

}

}

void running\_insert(struct ProcessNode\*p) //定义运行状态下的函数

{

if(running\_head) //已经存在一个运行中的进程

{

printf("error! there is a process in running already!");

return;

}

list\_insert(&running\_head,&running\_tail,p);

}

void blocked\_insert(struct ProcessNode\*p) //定义堵塞状态下的函数

{

list\_insert(&blocked\_head,&blocked\_tail,p); //调用函数

}

void ready\_insert(struct ProcessNode\* p) //定义

{

list\_insert(&ready\_head,&ready\_tail,p); //调用函数

}

void list\_insert(struct ProcessNode\*\* head, struct ProcessNode\*\* tail, struct ProcessNode\* p) //定义list\_insert函数

{

if(\*head==0)

{

\*head=\*tail=p;

p->next=0;

}

Else // if else条件语句

{

(\*tail)->next=p;

p->next=0;

\*tail=p;

}

}

int exist(struct ProcessNode\* p) //定义存在函数

{

if(list\_exist(running\_head,p))

return 1;

if(list\_exist(blocked\_head,p))

return 1;

if(list\_exist(ready\_head,p))

return 1;

return 0;

}

int list\_exist(struct ProcessNode\* head, struct ProcessNode\* p) //定义函数

{

struct ProcessNode\* temp;

temp=head;

while(temp) //while条件语句

{

if(temp->pid==p->pid)

return 1;

temp=temp->next;

}

return 0;

}

void input\_one\_process() //定义输入一个进程的函数

{

struct ProcessNode\* p=malloc(sizeof(struct ProcessNode));

char temp;

printf("please input the ID of the process (an integer)\n");

scanf("%d%c",&(p->pid),&temp);

// temp is used to clear up the buffer

printf("please input the status of the process: 0-running;1-blocked;2-ready\n");

scanf("%d%c",&(p->status),&temp);

printf("please input the priority of the process: 0-low;1-normal;2-high\n");

scanf("%d%c",&(p->priority),&temp);

insert\_process(p); //调用函数

}

void input\_processes() //定义该函数，与开头的声明相对应

{

char c;

char temp;

GETINPUT: printf("please input continue or not: y-yes; n-not\n");

scanf("%c",&c); // this is the real input

scanf("%c",&temp); // to clear up the buffer

switch(c) //switch选择语句

{

case 'y': //输入y则调用输入进程函数

{

input\_one\_process();

goto GETINPUT;

}

case 'n': //输入n则退出

break;

default:

{

printf("the input is not valid. please try another\n");

goto GETINPUT;

}

}

}

void output\_processes() //定义该函数，与开头的声明相对应

{

printf("the running process information:\n");

list\_print(running\_head); //调用函数

printf("the blocked processes information:\n");

list\_print(blocked\_head);

printf("the ready processes information:\n");

list\_print(ready\_head);

}

void list\_print(struct ProcessNode\* head) //定义list\_print函数

{

struct ProcessNode\* temp=head;

while(temp)

{

printf("ProcessID: %d; ProcessPriority: %d\n",temp->pid,temp->priority);

temp=temp->next;

}

}

void FCFS\_Scheduling() //定义该函数，与开头的声明相对应

{

int count=0;

struct ProcessNode\* temp=ready\_head;

if(ready\_head==0)

{

printf("there is no process to be scheduled\n");

return;

}

printf("the sheduling order under FCFS is as follows:\n");

while(temp) //while条件语句

{

printf("Scheduling order %d: ProcessID %d\n", ++count,temp->pid);

temp=temp->next;

}

}

主函数main.c:

#include "Process.h" //引入写好的process.h头文件

void main()

{

input\_processes();

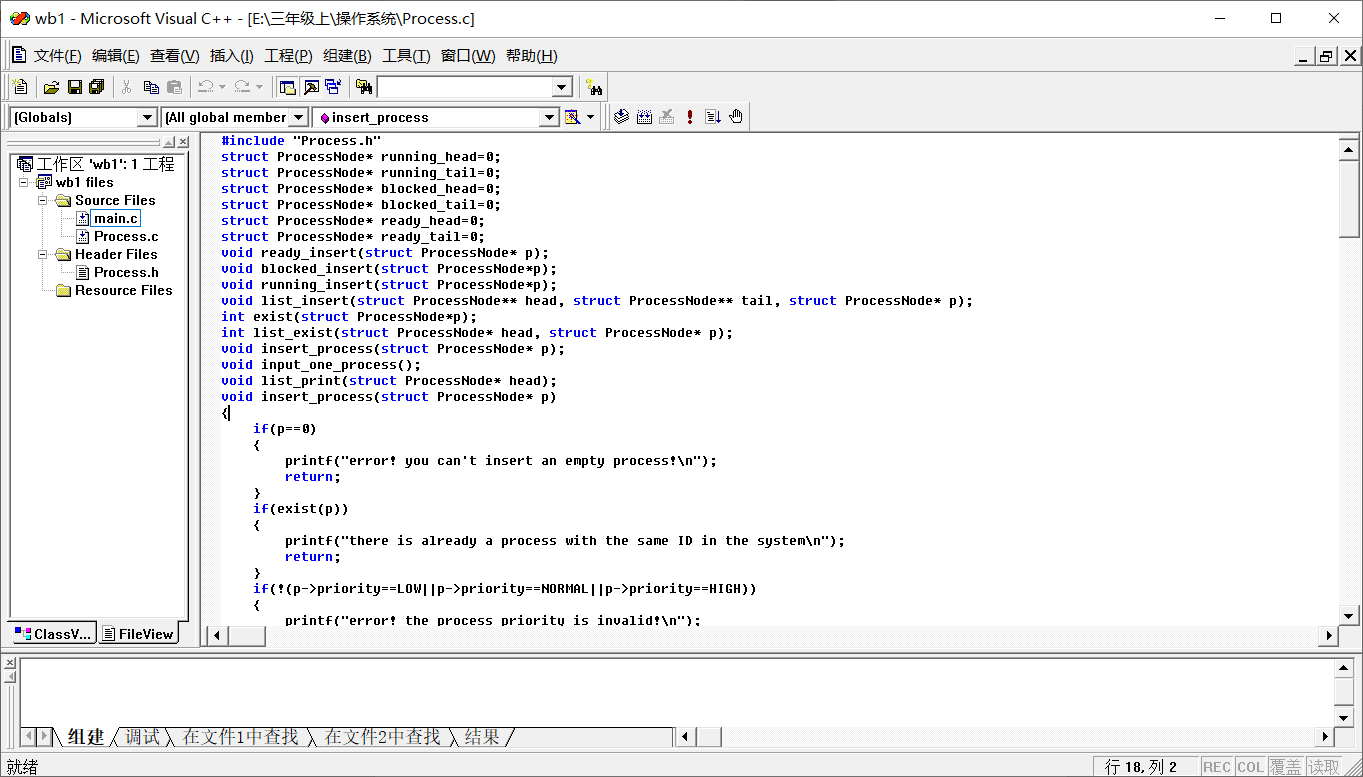
// output\_processes();

FCFS\_Scheduling(); //调用写好的函数

}

三、实验结果：

Vc++6.0下的代码文本框



执行成功后的执行框,在执行框里完成输入控制.

