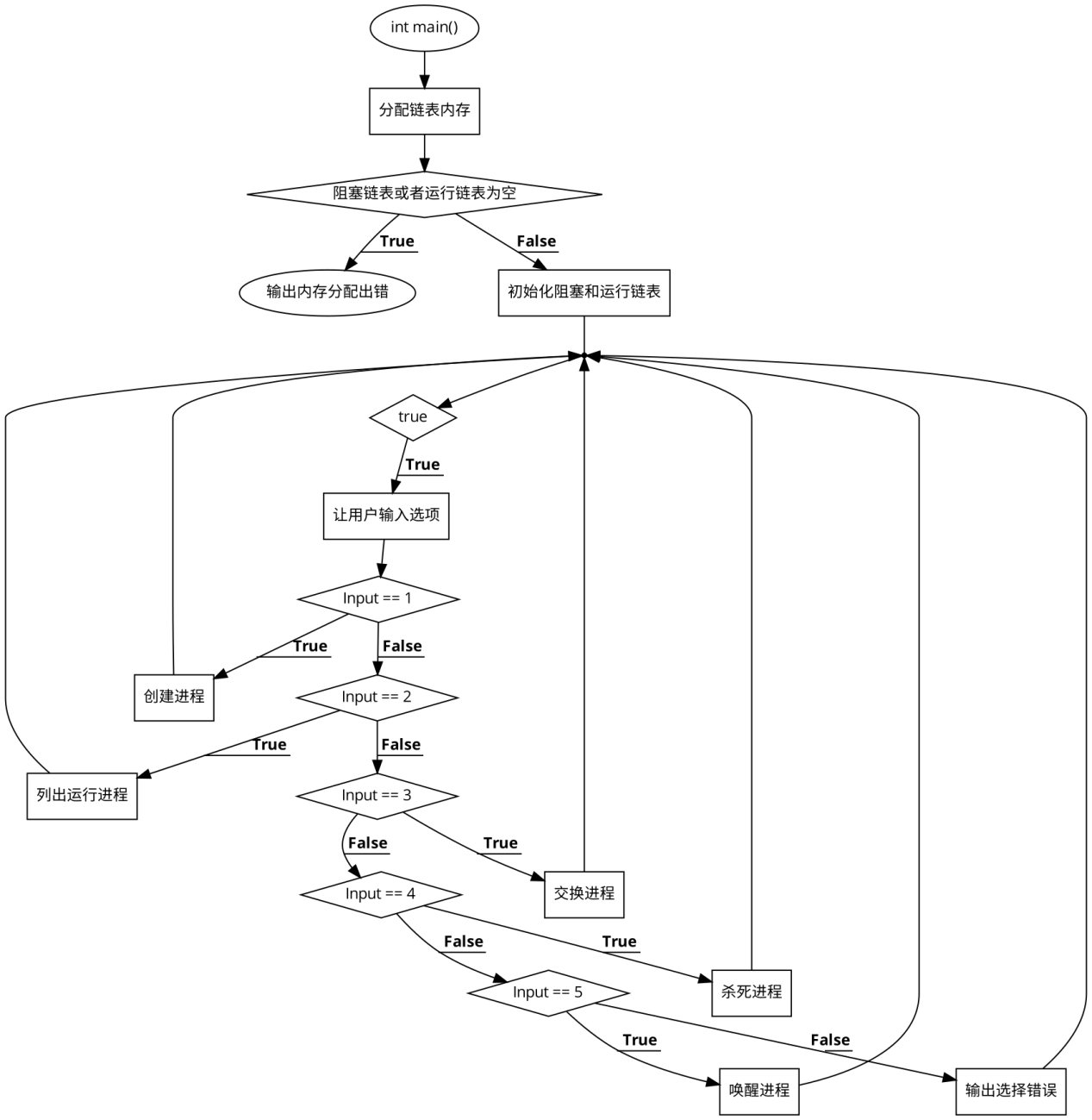
# 实验1

1. 实验分析

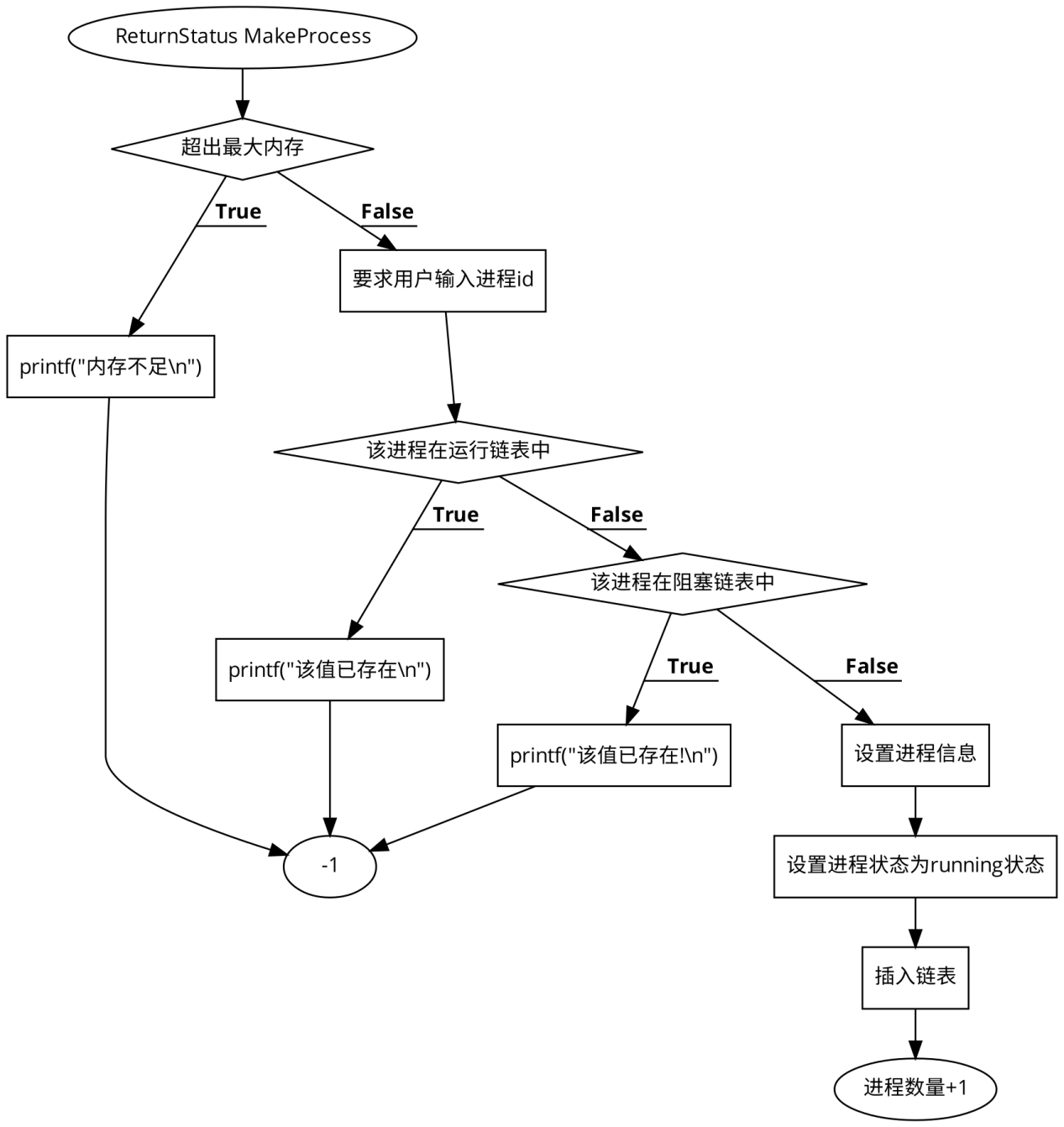
使用while无限循环结构使用户不断输入选项，从而进行不同的处理。通过malloc为pcb分配内存从而创建进程，通过free销毁进程控制块来杀死进程; 创建进程运行链表和进程阻塞链表来模拟进程的运行和阻塞。

1. 流程图

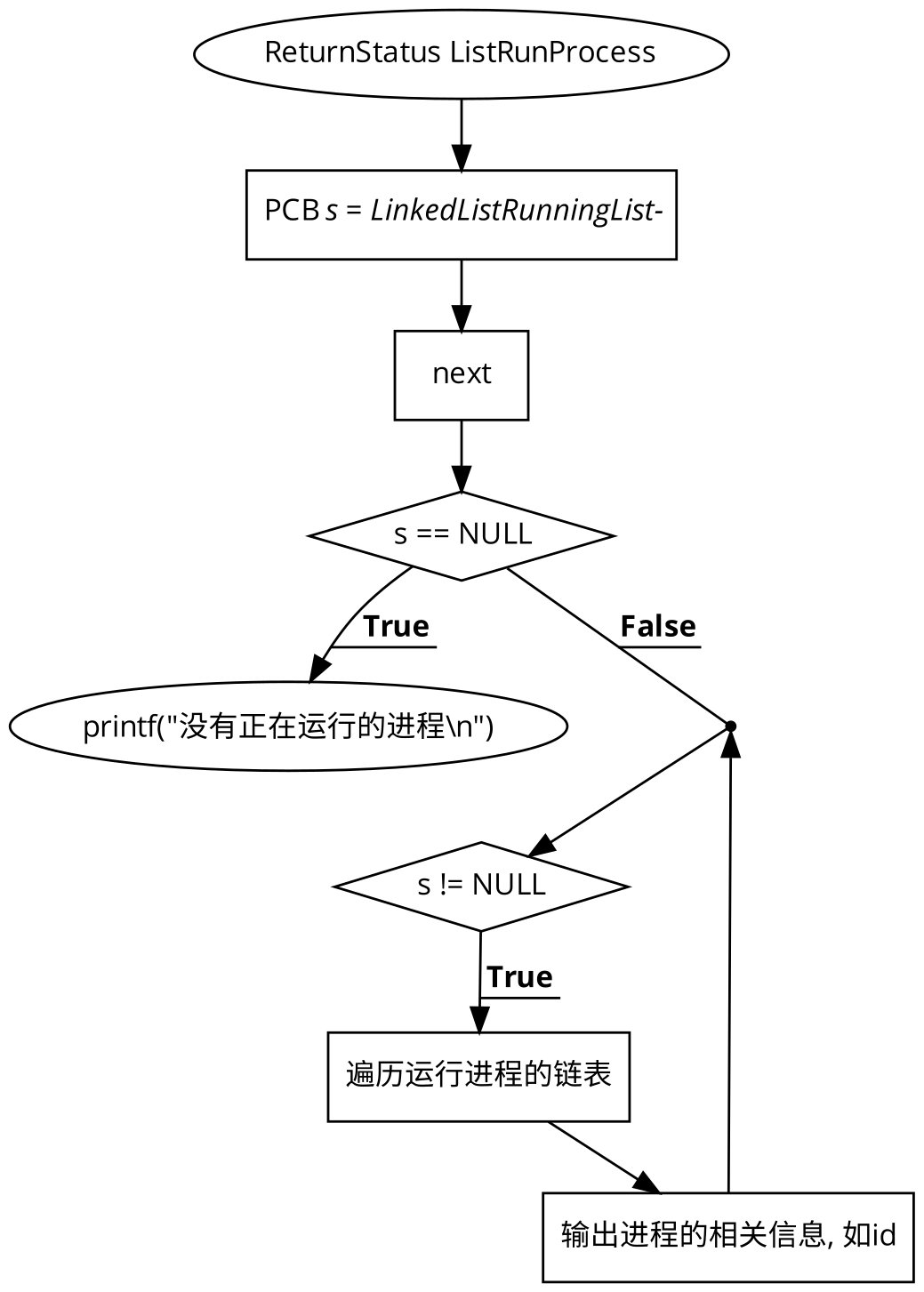
主函数:



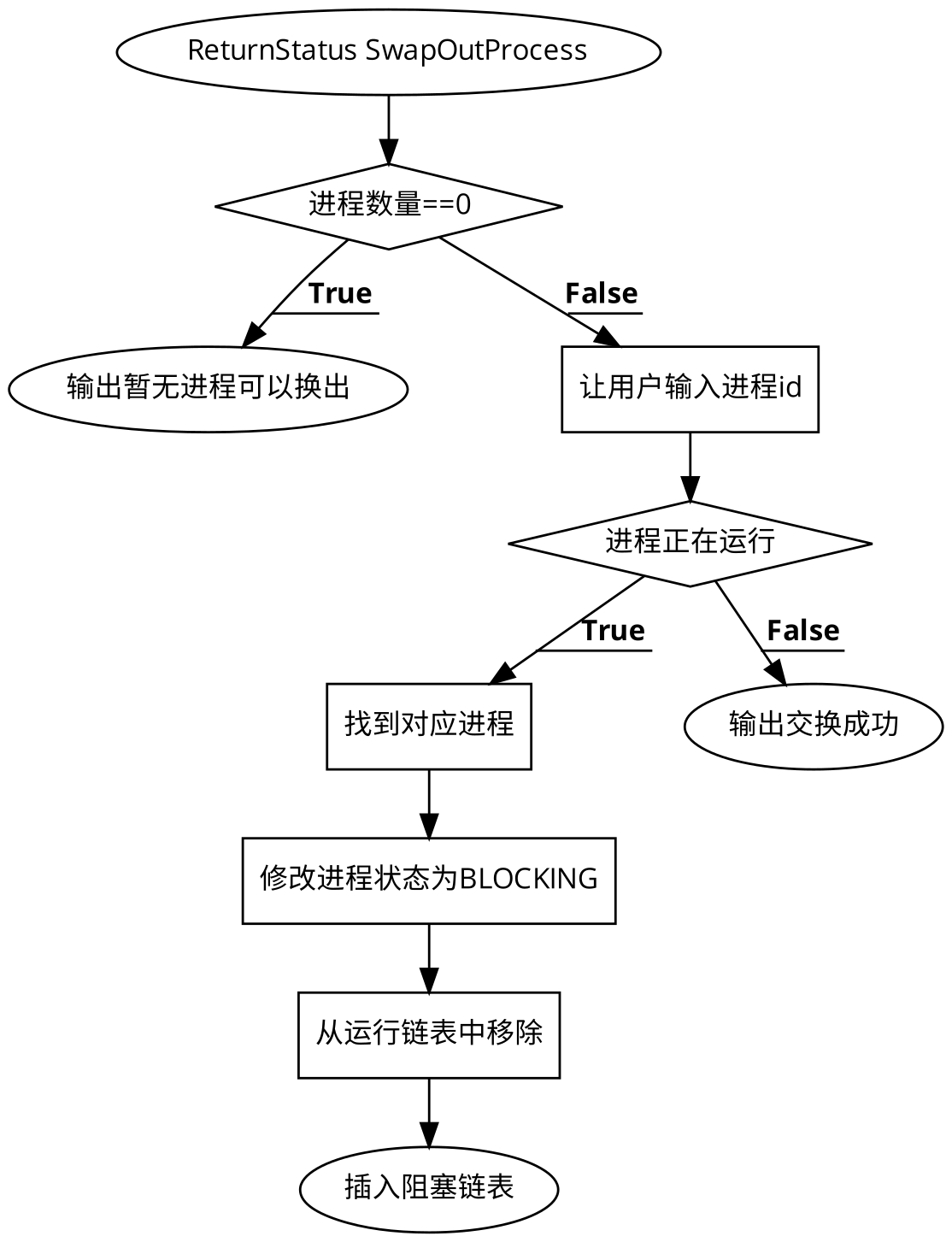
MakeProcess函数:



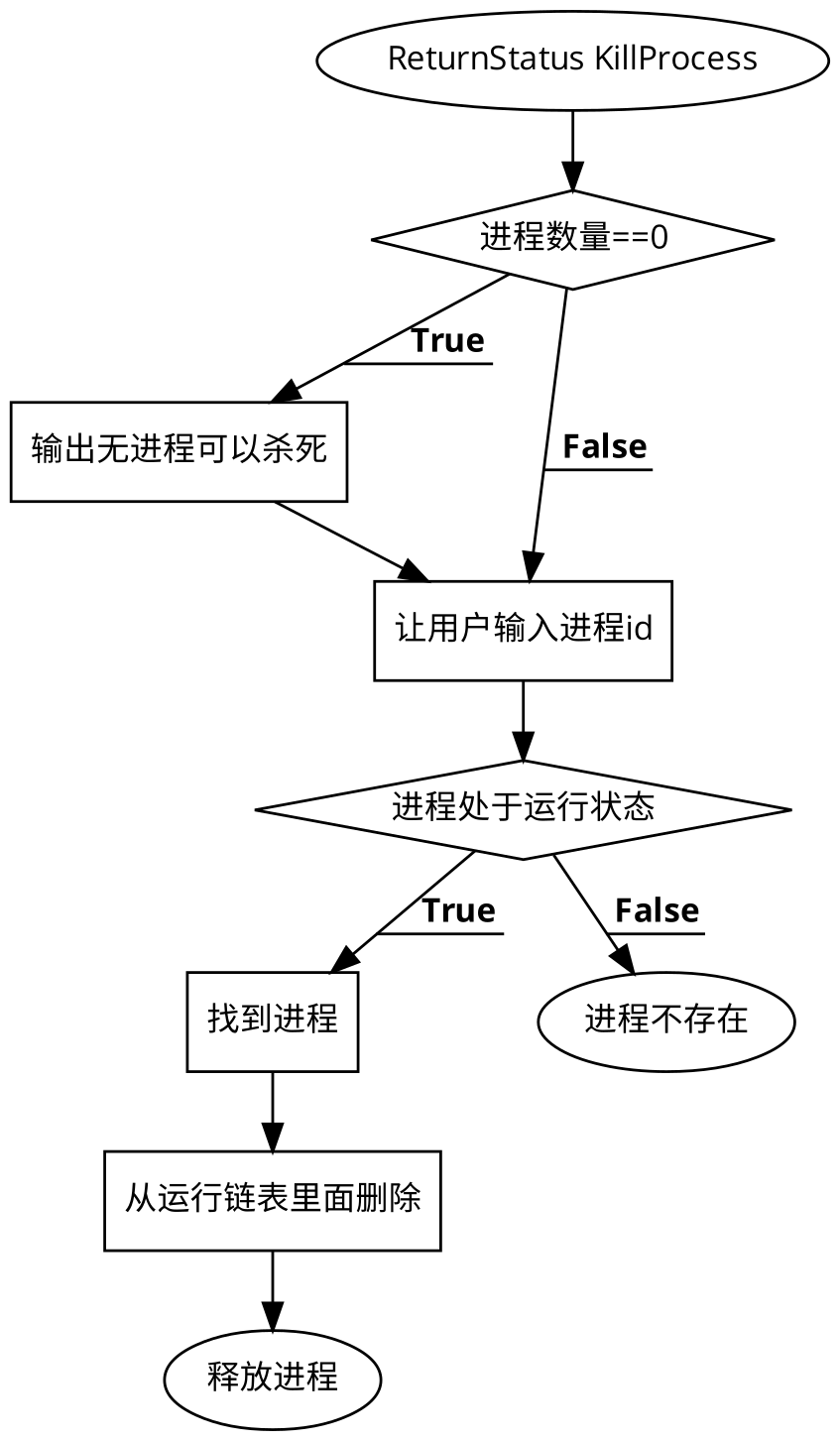
ListRunProcess函数:



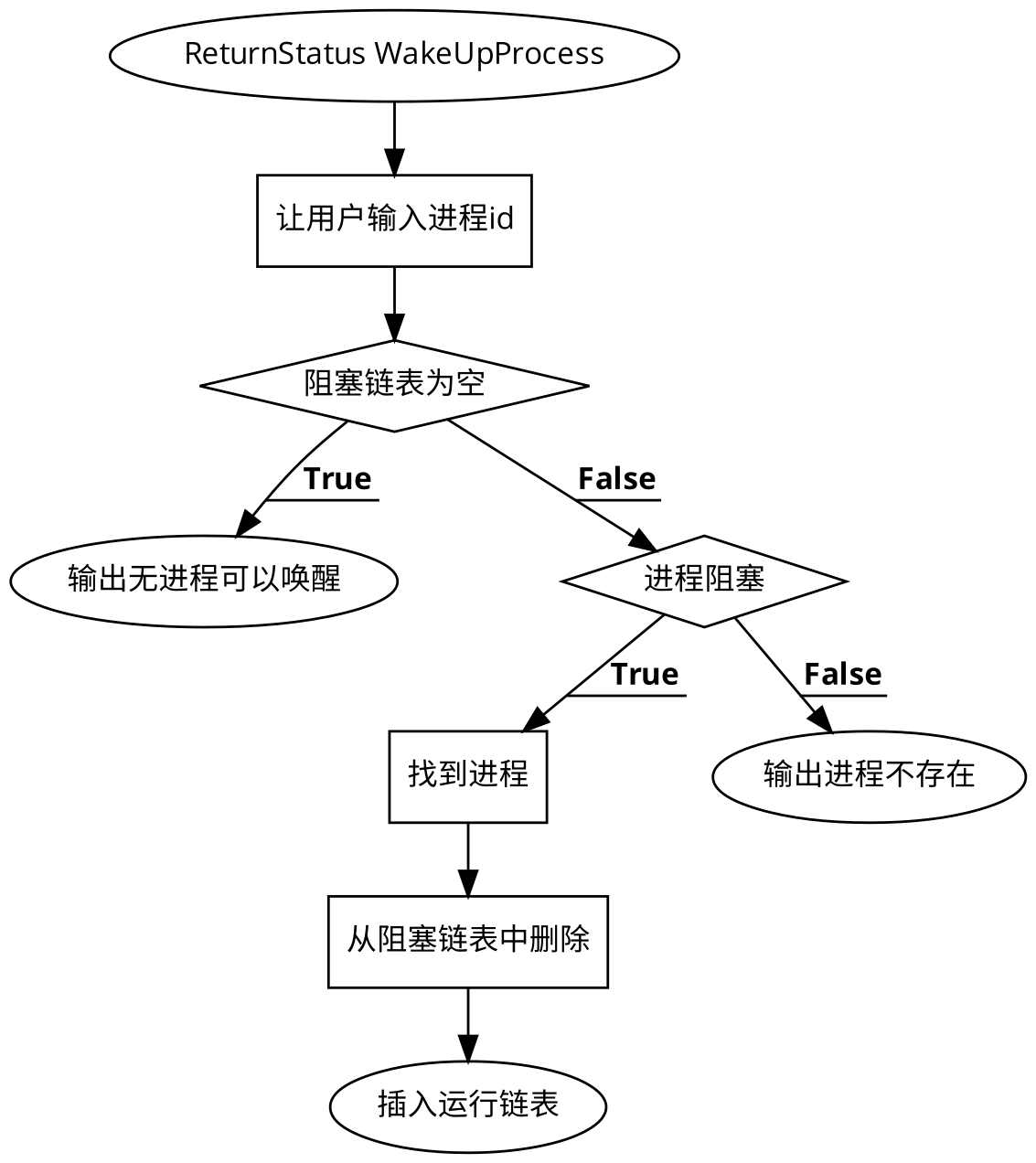
SwapOutProcess函数:



KillProcess函数:



WakeUpProcess函数:



1. 源代码

#include <stdio.h>

#include <stdlib.h>

#include <assert.h>

#include <stdbool.h>

#define MAX\_SIZE 30

typedef int ReturnStatus;

#define getpch(type) (type \*)malloc(sizeof(type))

PCB \*FindProcess(PCB \*list,int pid);

ReturnStatus MakeProcess(PCB \*LinkedListRunningList,

PCB \*LinkedListBlockingList,int \*cnt);

ReturnStatus ListRunProcess(PCB \*LinkedListRunningList);

ReturnStatus SwapOutProcess(PCB \*LinkedListRunningList,

PCB \*LinkedListBlockingList,int \*cnt);

ReturnStatus KillProcess(PCB \*LinkedListRunningList,int \*cnt);

ReturnStatus WakeUpProcess(PCB \*LinkedListRunningList,

PCB \*LinkedListBlockingList,int \*cnt);

ReturnStatus IsProcessRunning(PCB \*LinkedListRunningList,int pid);

ReturnStatus IsProcessBlock(PCB \*LinkedListBlockingList,int pid);

typedef struct PCB\_Type{

int pid;

int priority;

int cnt;

char content[20];

int state;

//下一个要执行的进程

struct PCB\_Type \*next;

} PCB;

enum ProcessTypeValue{

RUNNING\_STATE = 1000,

BLOCKING\_STATE = -1000,

READYING\_STATE = 500

};

void sort()

{

// PCB \*first, \*second;

// int insert = 0;

}

PCB \*FindProcess(PCB \*list,int pid)

{

PCB \*s = list;

while (s->next)

{

if (s->next->pid == pid) {

return s;

}

s = s->next;

}

return NULL;

}

ReturnStatus MakeProcess(PCB \*LinkedListRunningList,

PCB \*LinkedListBlockingList,int \*cnt)

{

PCB \*s = LinkedListRunningList;

if (\*cnt >= MAX\_SIZE)

{

printf("内存不足\n");

return -1;

}

PCB \*p = (PCB \*)malloc(sizeof(PCB));

assert(p != NULL);

int pid;

printf("pid:\n");

scanf("%d",&pid);

if (IsProcessRunning(LinkedListRunningList, pid))

{

printf("该值已存在\n");

return -1;

}

if (IsProcessBlock(LinkedListBlockingList, pid))

{

printf("该值已存在!\n");

return -1;

}

//没重复，保存

p->pid = pid;

printf("优先级:\n");

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

printf("大小:\n");

scanf("%d",&p->cnt);

printf("内容:\n");

scanf("%s",p->content);

p->state = RUNNING\_STATE;

p->next = NULL;

while (s->next != NULL)

{

s = s->next;

}

s->next = p;

\*cnt = \*cnt + 1;

}

ReturnStatus ListRunProcess(PCB \*LinkedListRunningList)

{

PCB \*s = LinkedListRunningList->next;

if (s == NULL)

{

printf("没有正在运行的进程\n");

return -1;

}

while (s != NULL)

{

printf("进程id:%d\n",s->pid);

printf("进程优先级:%d\n",s->priority);

printf("进程大小:%d\n",s->cnt);

printf("进程内容:%s\n",s->content);

s = s->next;

}

}

//换出某个进程

ReturnStatus SwapOutProcess(PCB \*LinkedListRunningList,

PCB \*LinkedListBlockingList,int \*cnt)

{

if (\*cnt == 0) {

printf("无可换出的进程\n");

return -1;

}

int pid;

printf("请输入需要换出进程的id:\n");

scanf("%d",&pid);

if (IsProcessRunning(LinkedListRunningList, pid)) {

PCB \*s = FindProcess(LinkedListRunningList, pid);

s->next->state = BLOCKING\_STATE;

//将该线程存入阻塞队列中

PCB \*p = LinkedListBlockingList;

while (p->next != NULL) {

p = p->next;

}

p->next = s->next;

s->next = s->next->next;

p->next->next = NULL;

\*cnt = \*cnt - 1;

printf("交换成功\n");

}else{

printf("处于阻塞状态\n");

}

}

ReturnStatus KillProcess(PCB \*LinkedListRunningList,int \*cnt)

{

if (\*cnt == 0) {

printf("无可杀死的进程\n");

return -1;

}

int pid;

printf("输入要杀死的进程id:\n");

scanf("%d",&pid);

if (IsProcessRunning(LinkedListRunningList, pid)) {

PCB \*s = FindProcess(LinkedListRunningList, pid);

PCB \*thisThread = s->next;

\*cnt = \*cnt - 1;

s->next = s->next->next;

free(thisThread);

printf("成功杀死\n");

}

else

{

printf("该线程不存在或已处于阻塞状态\n");

}

}

ReturnStatus WakeUpProcess(PCB \*LinkedListRunningList,

PCB \*LinkedListBlockingList,int \*cnt)

{

PCB \*s = LinkedListBlockingList;

int pid;

printf("请输入要唤醒的进程id:\n");

scanf("%d",&pid);

if (s->next == NULL) {

printf("没有可唤醒的线程\n");

return -1;

}

if (IsProcessBlock(LinkedListBlockingList, pid)) {

s = FindProcess(LinkedListBlockingList, pid);

s->next->state = RUNNING\_STATE;

PCB \*p = LinkedListRunningList;

while (p->next != NULL)

p = p->next;

p->next = s->next;

s->next = s->next->next;

p->next->next = NULL;

\*cnt = \*cnt + 1;

printf("唤醒成功\n");

}else{

printf("该线程不存在\n");

}

}

ReturnStatus IsProcessRunning(PCB \*LinkedListRunningList,int pid){

int result = 0;

PCB \*s = LinkedListRunningList->next;

while (s != NULL)

{

if (s->pid == pid) {

//存在，直接返回

result = 1;

break;

}

s = s->next;

}

return result;

}

ReturnStatus IsProcessBlock(PCB \*LinkedListBlockingList,int pid)

{

int result = 0;

PCB \*s = LinkedListBlockingList->next;

while (s != NULL)

{

if (s->pid == pid) {

result = 1;

break;

}

s = s->next;

}

return result;

}

int main()

{

int StorageNumber = 0;

int Input = 1;

PCB \*LinkedListRunningList = (PCB \*)malloc(sizeof(PCB));

PCB \*LinkedListBlockList = (PCB \*)malloc(sizeof(PCB));

if (!LinkedListRunningList || !LinkedListBlockList)

{

puts("内存分配出错");

exit(1);

}

// 初始化

LinkedListBlockList->next = NULL;

LinkedListRunningList->next = NULL;

while (true)

{

puts("");

printf("1.创建新的进程 2.查看运行进程\n");

printf("3.换出某个进程 4.杀死运行进程\n");

printf("5.唤醒某个进程 6.退出程序 \n");

printf("输入(1-6):\n");

scanf("%d",&Input);

if (Input == 1)

{

MakeProcess(LinkedListRunningList, LinkedListBlockList, &StorageNumber);

}

else if (Input == 2)

{

ListRunProcess(LinkedListRunningList);

}

else if (Input == 3)

{

SwapOutProcess(LinkedListRunningList, LinkedListBlockList, &StorageNumber);

}

else if (Input == 4)

{

KillProcess(LinkedListRunningList, &StorageNumber);

}

else if (Input == 5)

{

WakeUpProcess(LinkedListRunningList, LinkedListBlockList, &StorageNumber);

}

else

{

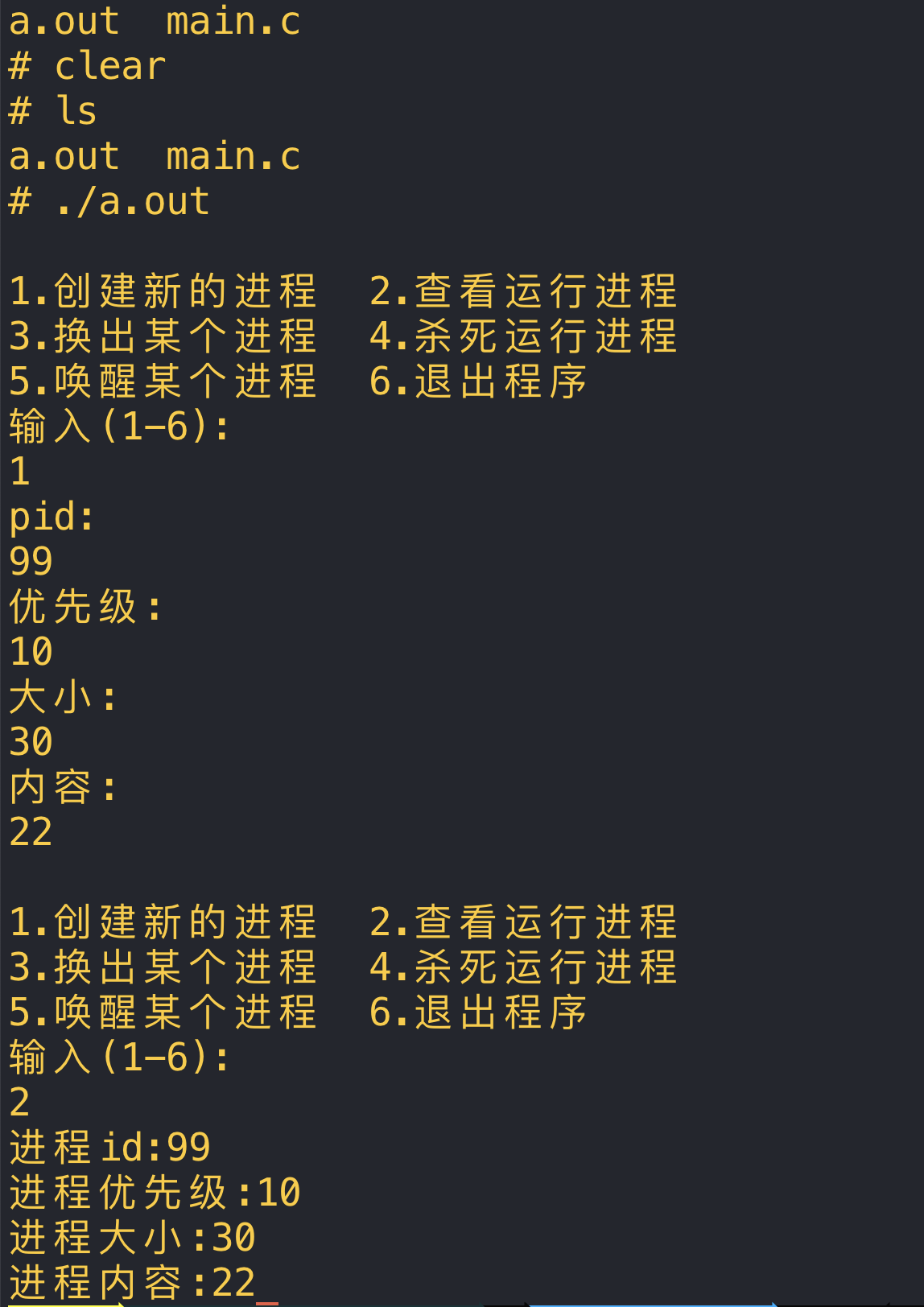
fprintf(stderr, "没有这个选择项");

}

}

}

1. 实验结果



实验二

FCFS进程算法

源代码如下:

#include<stdio.h> //for using printf like function

#include<stdlib.h> //for using abs function

int main()

{

int pn,pno[20],at[20],bt[20],i,tat[20],com[20],wt[20],temp,j;

float avg,avg2,avgtat,avgwt;

printf("Enter the number of process for FCFS SCHEDULING\n");

scanf("%d",&pn);

printf("Enter the process 1 by 1\n");

for(i=0;i<pn;i++)

scanf("%d",&pno[i]);

printf("Enter the process of Arrival time\n");

for(i=0;i<pn;i++)

scanf("%d",&at[i]);

printf("Enter the process burst time\n");

for(i=0;i<pn;i++)

scanf("%d",&bt[i]); //accepting burst time process number and arrival time from user

for(i=0;i<pn;i++)

{

for(j=i+1;j<pn;j++)

{

if(at[i]>at[j])

{

temp=at[i];

at[i]=at[j];

at[j]=temp;

temp=pno[i];

pno[i]=pno[j];

pno[j]=temp;

temp=bt[i];

bt[i]=bt[j];

bt[j]=temp;

}

}

} //sorting according to arrival time

com[0]=bt[0]+at[0];

tat[0]=bt[0];

wt[0]=0;

for(i=1;i<pn;i++)

com[i]=abs(com[i-1]+bt[i]); //completion time

for(i=0;i<pn;i++)

tat[i]=abs(com[i]-at[i]); //turn aroun time

for(i=0;i<pn;i++)

wt[i]=abs(tat[i]-bt[i]); //waiting time

for(i=0;i<pn;i++)

avg=avg+tat[i];

avgtat=avg/pn; //average TAT

for(i=0;i<pn;i++)

avg2=avg2+wt[i];

avgwt=avg2/pn; //average WT

printf("pno\tat\tbt\tcom\ttat\twt\t\n");

for(i=0;i<pn;i++)

{

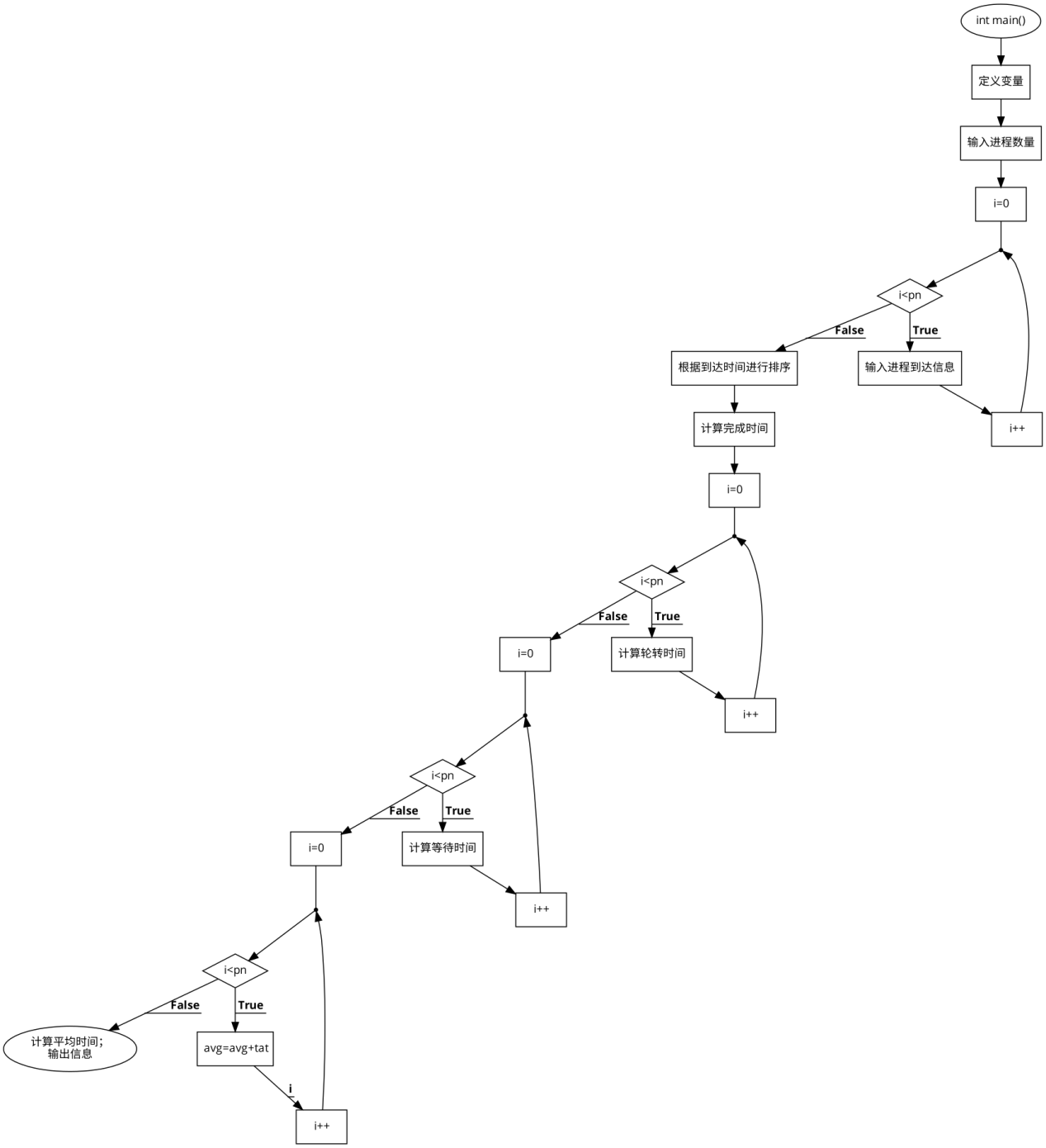
printf("P%d\t%d\t%d\t%d\t%d\t%d\n",pno[i],at[i],bt[i],com[i],tat[i],wt[i]);

}

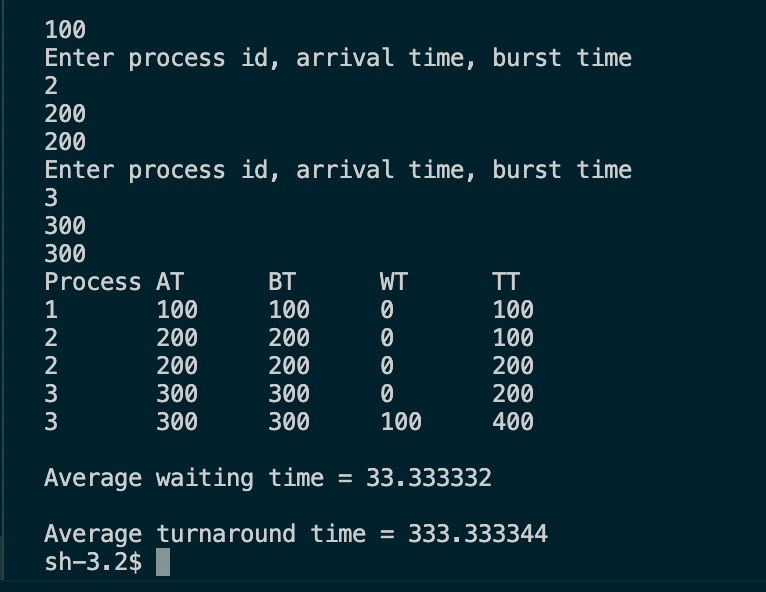
printf("ATAT= %f\nAWT= %f\n",avgtat,avgwt); //displaying every values

}

流程图:



运行结果



PSA算法

源代码:

头文件:

#ifndef PSA\_H\_

#define PSA\_H\_

#include <iostream>

#include <algorithm>

#include <iomanip>

#include <vector>

//作业结构体

typedef struct PCB

{

int ID; //标识符

int Level; //优先级

int ComeTime; //到达时间

int ServerTime; //服务时间

int FinishTime; //完成时间

int TurnoverTime; //周转时间

double WeightedTurnoverTime; //带权周转时间

}PCB;

/\*

函数功能：输入作业信息

参数说明：

PCBList std::vector<PCB>& PCB链

\*/

void InputPCB(std::vector<PCB> &PCBList);

/\*

函数功能：PSA算法

参数说明：

PCBList std::vector<PCB>& PCB链

\*/

void PSA(std::vector<PCB> &PCBList);

/\*

函数功能：显示结果

参数说明：

PCBList std::vector<PCB>& PCB链

\*/

void show(std::vector<PCB> &PCBList);

/\*

函数功能：比较函数，用于sort()，按ComeTime升序排列

参数说明：

p1 const PCB& PCB

p2 const PCB& PCB

\*/

bool CmpByComeTime(const PCB &p1, const PCB &p2);

bool CmpByLevel(const PCB &p1, const PCB &p2);

#endif

void InputPCB(std::vector<PCB> &PCBList)

{

do {

PCB temp;

std::cout << "input id: ";

std::cin >> temp.ID;

std::cout << "input priority: ";

std::cin >> temp.Level;

std::cout << "input arrive time: ";

std::cin >> temp.ComeTime;

std::cout << "input server time: ";

std::cin >> temp.ServerTime;

PCBList.push\_back(temp);

std::cout << "continue? Y/N: ";

char ans;

std::cin >> ans;

if ('Y' == ans || 'y' == ans)

continue;

else

break;

} while (true);

}

//PSA算法

void PSA(std::vector<PCB> &PCBList)

{

std::sort(PCBList.begin(), PCBList.end(), CmpByComeTime); //按到达时间排序

//同时到达的按优先级降序排序，决定首先运行的作业

int i = 1;

std::vector<PCB>::iterator it = PCBList.begin() + 1;

while ((\*it).ComeTime == (\*(it - 1)).ComeTime)

{

++i;

++it;

}

std::sort(PCBList.begin(), PCBList.begin() + i, CmpByLevel);

int FinishTime = -1;

for (it = PCBList.begin(); it < PCBList.end(); ++it)

{

if ((\*it).ComeTime >= FinishTime) (\*it).FinishTime = (\*it).ComeTime + (\*it).ServerTime;

else //有作业正在运行，等待作业完毕，此作业再运行

(\*it).FinishTime = FinishTime + (\*it).ServerTime;

(\*it).TurnoverTime = (\*it).FinishTime - (\*it).ComeTime;

(\*it).WeightedTurnoverTime = (double)(\*it).TurnoverTime / (\*it).ServerTime;

FinishTime = (\*it).FinishTime;

//在一个作业运行期间，如果有其他作业到达，将他们按照优先级降序排列

i = 1;

while ((it + i) < PCBList.end() && (\*(it + i)).ComeTime <= FinishTime)

++i;

std::sort(it + 1, it + i, CmpByLevel);

}

std::sort(PCBList.begin(), PCBList.end(), CmpByComeTime); //重新排列，用于显示结果

}

void show(std::vector<PCB> &PCBList)

{

int SumTurnoverTime = 0;

double SumWeightedTurnoverTime = 0;

std::cout.setf(std::ios::left);

std::cout << std::setw(20) << "id";

for (std::vector<PCB>::iterator it = PCBList.begin(); it < PCBList.end(); ++it)

std::cout << std::setw(5) << (\*it).ID;

std::cout << std::endl;

std::cout << std::setw(20) << "priority";

for (std::vector<PCB>::iterator it = PCBList.begin(); it < PCBList.end(); ++it)

std::cout << std::setw(5) << (\*it).Level;

std::cout << std::endl;

std::cout << std::setw(20) << "arrive time";

for (std::vector<PCB>::iterator it = PCBList.begin(); it < PCBList.end(); ++it)

std::cout << std::setw(5) << (\*it).ComeTime;

std::cout << std::endl;

std::cout << std::setw(20) << "server time";

for (std::vector<PCB>::iterator it = PCBList.begin(); it < PCBList.end(); ++it)

std::cout << std::setw(5) << (\*it).ServerTime;

std::cout << std::endl;

std::cout << std::setw(20) << "finish time";

for (std::vector<PCB>::iterator it = PCBList.begin(); it < PCBList.end(); ++it)

std::cout << std::setw(5) << (\*it).FinishTime;

std::cout << std::endl;

std::cout << std::setw(20) << "avg turn over time";

for (std::vector<PCB>::iterator it = PCBList.begin(); it < PCBList.end(); ++it)

{

std::cout << std::setw(5) << (\*it).TurnoverTime;

SumTurnoverTime += (\*it).TurnoverTime;;

}

std::cout << std::endl;

std::cout << std::setw(20) << "weighted turn over time";

for (std::vector<PCB>::iterator it = PCBList.begin(); it < PCBList.end(); ++it)

{

std::cout << std::setw(5) << (\*it).WeightedTurnoverTime;

SumWeightedTurnoverTime += (\*it).WeightedTurnoverTime;;

}

std::cout << std::endl;

std::cout << "avg turn over time: " << (double)SumTurnoverTime / PCBList.size() << std::endl;

std::cout << "weighted turn over time: " << SumWeightedTurnoverTime / PCBList.size() << std::endl;

}

bool CmpByComeTime(const PCB &p1, const PCB &p2)

{

return p1.ComeTime < p2.ComeTime;

}

bool CmpByLevel(const PCB &p1, const PCB &p2)

{

return p1.Level > p2.Level;

}

主程序:

//main.cpp

#include "PSA.h"

int main()

{

std::vector<PCB> PCBList;

InputPCB(PCBList);

PSA(PCBList);

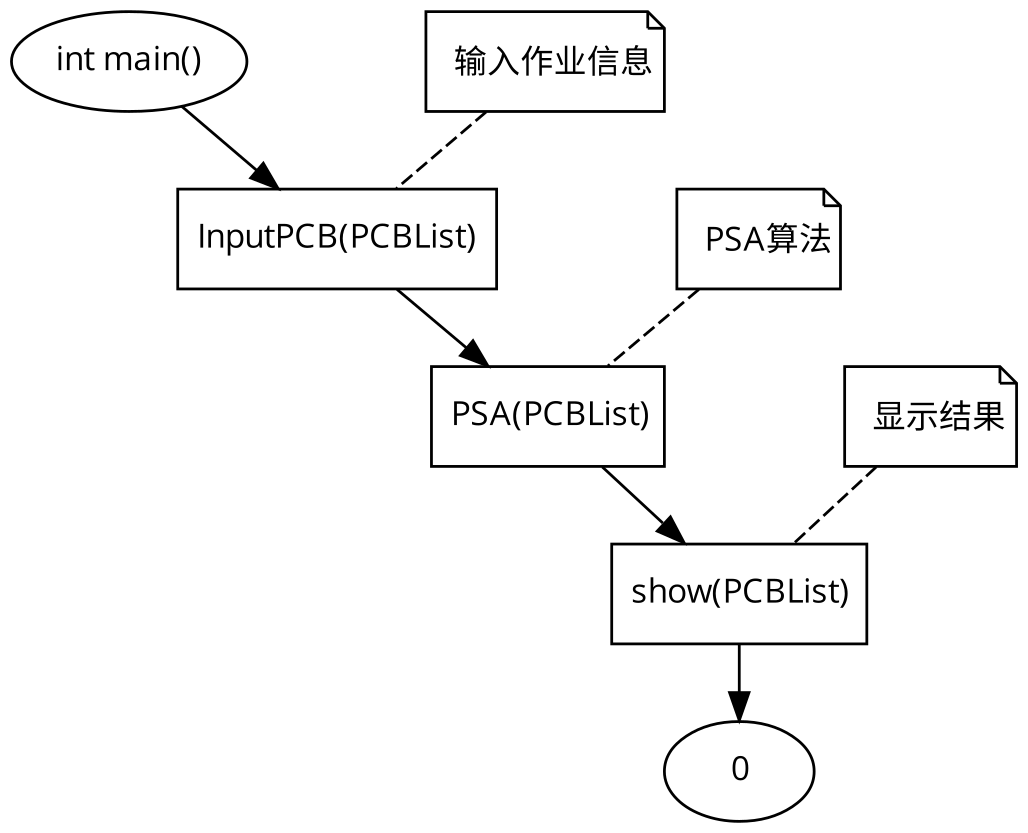
show(PCBList);

return 0;

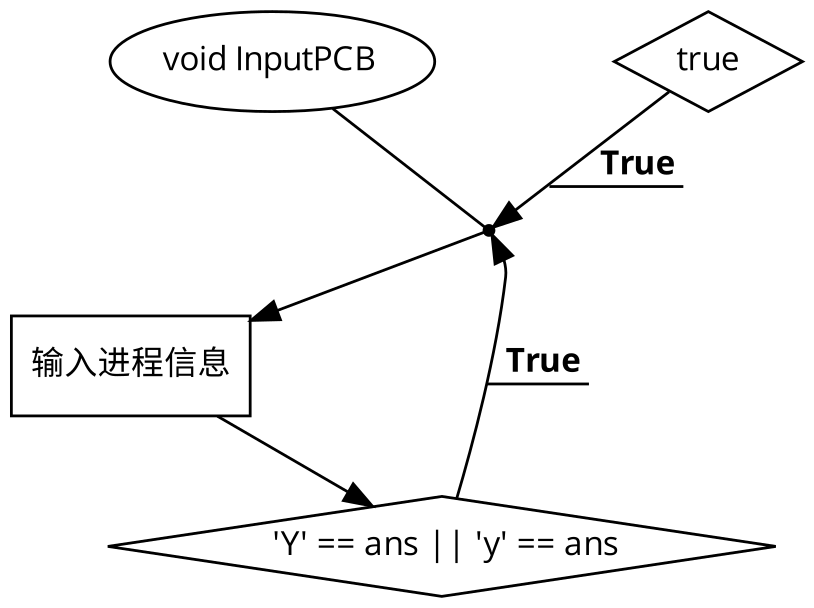
}

流程图:

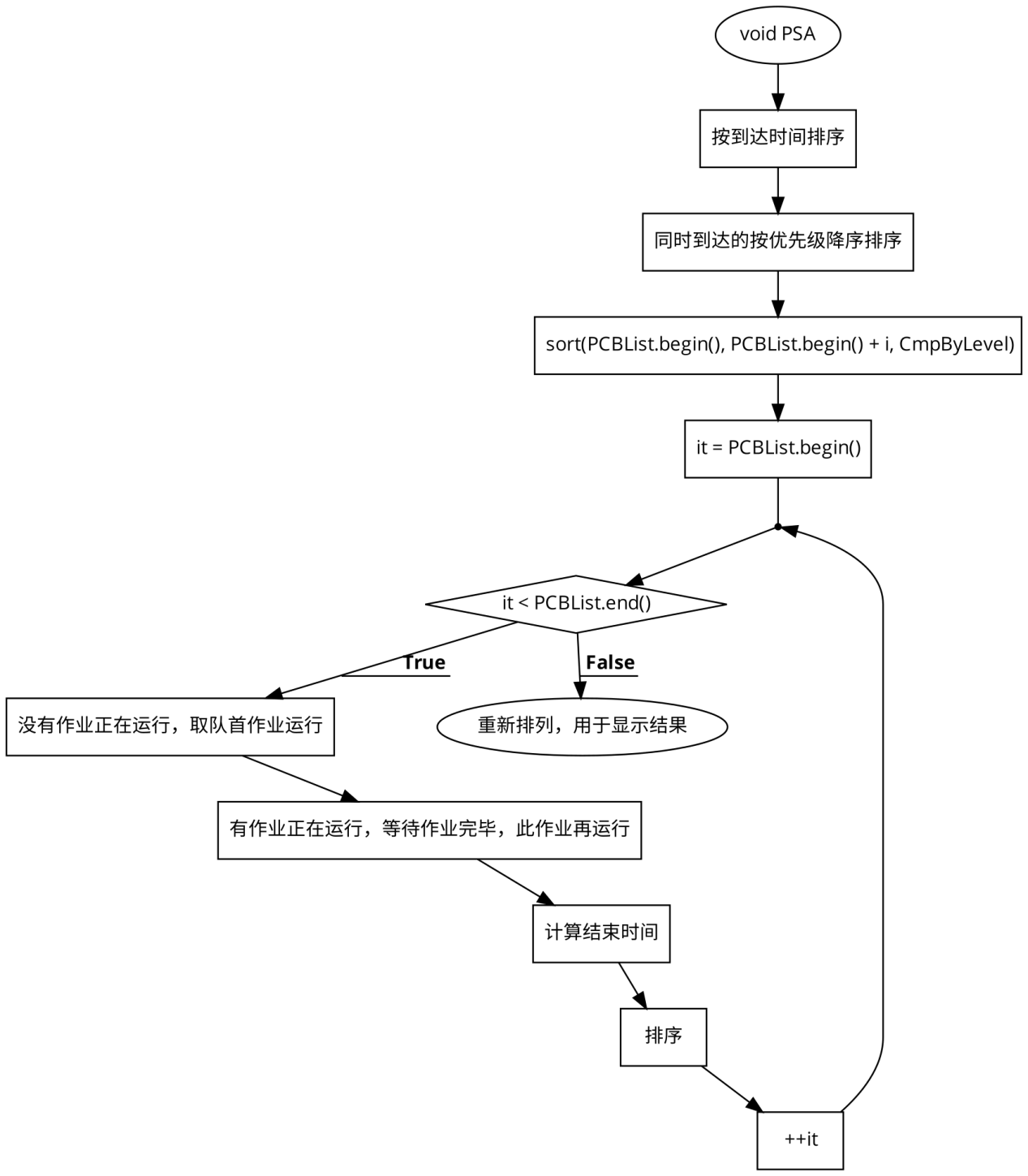
main函数:



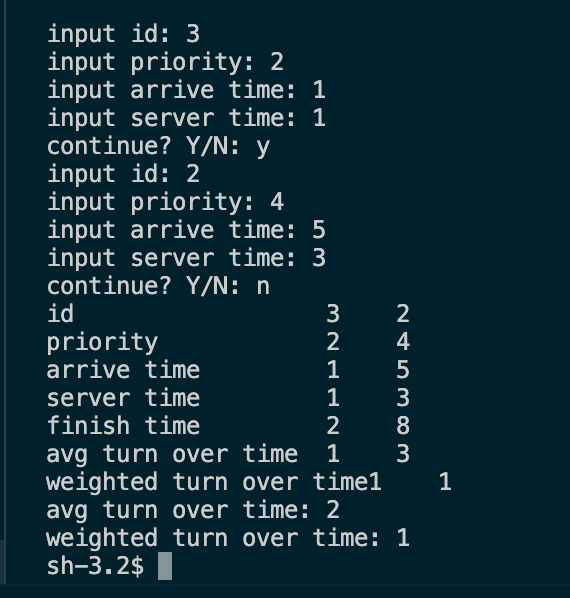
InputPCB:



PSA算法



结果:



SJF算法

源代码:

#include<stdio.h>

#include<stdlib.h>

int main()

{

int pn,pno[20],at[20],bt[20],i,tat[20],com[20],wt[20],temp,j;

float avg,avg2,avgtat,avgwt;

printf("Enter the number of process for SJF SCHEDULING\n");

scanf("%d",&pn);

printf("Enter the process 1 by 1\n");

for(i=0;i<pn;i++)

scanf("%d",&pno[i]);

printf("Enter the process of Arrival time\n");

for(i=0;i<pn;i++)

scanf("%d",&at[i]);

printf("Enter the process burst time\n");

for(i=0;i<pn;i++)

scanf("%d",&bt[i]); //accepting burst time process number and arrival time from user

for(i=0;i<pn;i++)

{

for(j=i+1;j<pn;j++)

{

if(at[i]>at[j])

{

temp=at[i];

at[i]=at[j];

at[j]=temp;

temp=pno[i];

pno[i]=pno[j];

pno[j]=temp;

temp=bt[i];

bt[i]=bt[j];

bt[j]=temp;

}

}

}

com[0]=bt[0]+at[0];

if(com[0]>at[pn-1])

{

for(i=1;i<pn;i++)

{

for(j=i+1;j<pn;j++)

{

if(bt[i]>bt[j])

{

temp=at[i];

at[i]=at[j];

at[j]=temp;

temp=pno[i];

pno[i]=pno[j];

pno[j]=temp;

temp=bt[i];

bt[i]=bt[j];

bt[j]=temp;

}

}

}

tat[0]=bt[0];

wt[0]=0;

for(i=1;i<pn;i++)

com[i]=abs(com[i-1]+bt[i]); //completion time

}

else

{

for(i=1;i<pn;i++)

{

for(j=i+1;j<pn;j++)

{

if(at[i]>at[j])

{

temp=at[i];

at[i]=at[j];

at[j]=temp;

temp=pno[i];

pno[i]=pno[j];

pno[j]=temp;

temp=bt[i];

bt[i]=bt[j];

bt[j]=temp;

}

}

}

com[1]=bt[1]+at[1];

for(i=2;i<pn;i++)

{

for(j=i+1;j<pn;j++)

{

if(bt[i]>bt[j])

{

temp=at[i];

at[i]=at[j];

at[j]=temp;

temp=pno[i];

pno[i]=pno[j];

pno[j]=temp;

temp=bt[i];

bt[i]=bt[j];

bt[j]=temp;

}

}

}

tat[0]=bt[0];

wt[0]=0;

for(i=2;i<pn;i++)

com[i]=abs(com[i-1]+bt[i]); //completion time

}

for(i=0;i<pn;i++)

tat[i]=abs(com[i]-at[i]); //turn aroun time

for(i=0;i<pn;i++)

wt[i]=abs(tat[i]-bt[i]); //waiting time

for(i=0;i<pn;i++)

avg=avg+tat[i]; //average TAT

avgtat=avg/pn;

for(i=0;i<pn;i++)

avg2=avg2+wt[i]; //average WT

avgwt=avg2/pn;

printf("pno\tat\tbt\tcom\ttat\twt\t\n");

for(i=0;i<pn;i++)

{

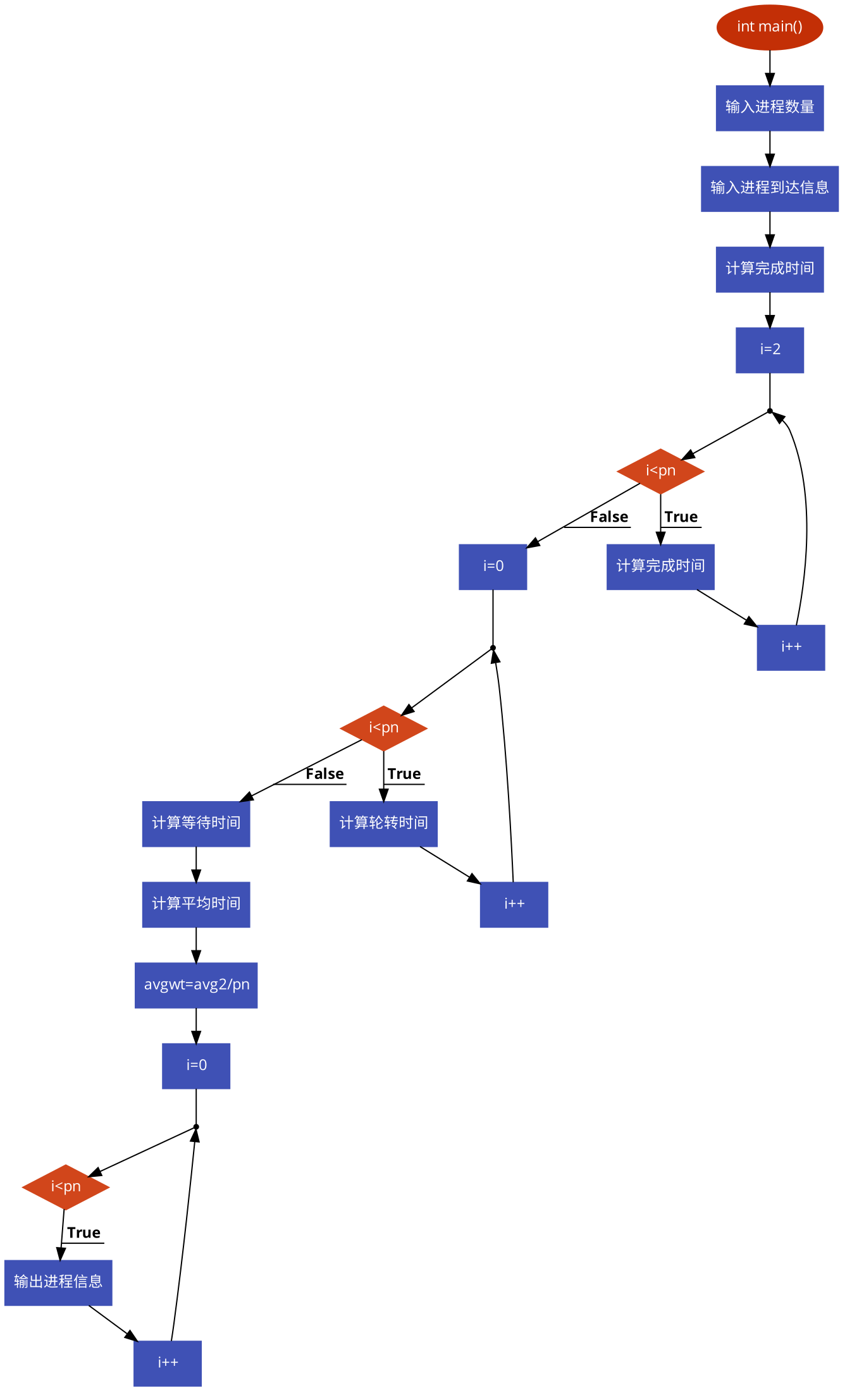
printf("P%d\t%d\t%d\t%d\t%d\t%d\n",pno[i],at[i],bt[i],com[i],tat[i],wt[i]);

}

printf("ATAT= %f\nAWT= %f\n",avgtat,avgwt); //displaying every values

}

流程图:



运行结果:

