

中国农业大学

操作系统实验

（2020 -2021 学年秋季学期）

**实验3 处理机管理**

## 一、实验目的

在多道程序或多任务系统中，系统中同时处于就绪态的进程有若干个，即能运行的进程数远远大于处理机个数。为了使系统中的各个进程能有条不紊的运行，必须选择某种调度策略，以选择一个进程占用处理机。

本次实验要求在Linux环境下，设计并实现一个模拟单处理机调度的算法，以加深对处理机调度基本概念和基本算法的理解。

## 二、实验内容

学号2022308310229

3、设计一个按优先级调度的算法实现处理机调度(学号末位为7,8,9)

（1） 系统有N个进程，每个进程用一个进程控制块来代表，进程控制块的结构如下图所示。

|  |
| --- |
| 进程名 |
| 到达时间 |
| 估计运行时间 |
| 进程优先级 |
| 进程状态 |
| …… |
| 链接指针 |

图3 进程控制块PCB结构

进程的优先数、要求运行时间和估计运行时间由用户程序任意设定，且优先数越低，优先级越高。调度时，总是选择优先级最高的进程运行。

（2）为了调度方便，设计一个指针指向就绪队列中的第一个进程。另外再设一个当前运行进程指针，指向当前正运行的进程。

（3）处理机调度时，总是选择已经到达队列的优先级最高的进程运行，进程每运行一次，优先级减1。为了采用动态优先级调度，进程每运行一次，其优先级就减1。由于本实验是模拟实验，所以对选中进程并不实际启动运行，而只是执行：

优先数加1和估计运行时间减1

用这两个操作来模拟进程的一次运行。

（4）进程运行一次后，若剩余时间不为0，且其优先级低于就绪队列的其他进程优先级，则选择一个高优先级进程抢占CPU；若剩余时间为0，把它的状态改为完成状态“C”，并撤出就绪队列。

（5）若就绪队列不空，则重复上述的步骤（3）和（4），直到所有进程成为完成状态。

（6）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。

（7）计算每个进程的周转时间、带权周转时间以及进程平均周转时间。

## 三、实验报告要求

书写实验报告，应该包括以下几项内容：

（1）程序中使用的数据结构说明；

（2）程序流程图和带有注释的源程序；

（3）执行程序名，并打印程序运行时的初值和运行结果，其中包括：

①各进程控制块的初始状态；

②选中运行进程的名字、运行后各进程控制块状态以及每次调度时，就绪队列的进程排列顺序；

（4）总结本次实验的心得与体会。

上传实验报告时，将实验报告、运行结果文件和相关程序组成压缩包一起上传系统。

本进程调度算法采用动态优先级的调度算法（即把处理机分配给优先数最高的进程）。

### 3.1 程序中使用的数据结构说明

系统有N个进程，每个进程用一个进程控制块PCB来代表，进程控制块的结构如下：

struct pcb //struct

{

char pname[N]; //process name

int runtime; //estimated running time

int arrivetime; //arrival time

int priority; //priority: the bigger number the higher priority

char state; //process state: Waiting, Running, Blocked, Terminated

struct pcb\* next; //pointer: address to the next pcb

int elapsedtime; //elapsed running time

int starttime;

int finishtime;

int turnaroundtime;

float weightedturaroundtime;

int firstflag;//0: never been run; 1: being re-running

} //3 pcb pointer

\*rhead = NULL, //address of ready queue

\* current, //address of current running process

\* bhead = NULL,//address of blocked queue

\* thead = NULL; //address of terminated queue

typedef struct pcb PCB;

其中，字符串pname表示此进程的名称，由用户定义；

整型数据runtime表示进程的预估运行时间，由用户设定；

整型数据arrivetime表示进程的到达时间，由用户设定；

整型数据priority表示进程的初始优先级（本实验采用动态优先级），由用户设定初始值，优先数越高，优先级越高，调度时总是选择优先级最高的进程运行；

字符型state表示进程的状态，其中W表示进程就绪（Waiting/Ready），R表示进程正在运行（Running），B表示进程被阻塞（Blocked），这是进程运行过程中的三种基本状态，此外，C表示进程结束/终止（Completed/Terminated），由于在此实验中进程创建后即加入就绪状态（初始状态为就绪），此后由程序系统自动调度，不标记开始/创建状态；

结构指针next指向下一个进程的地址，用于链接队列，其中每次处于队尾的进程的next指针将指向NULL；

整型数据elapsedtime表示程序的已运行时间，初始值为0；

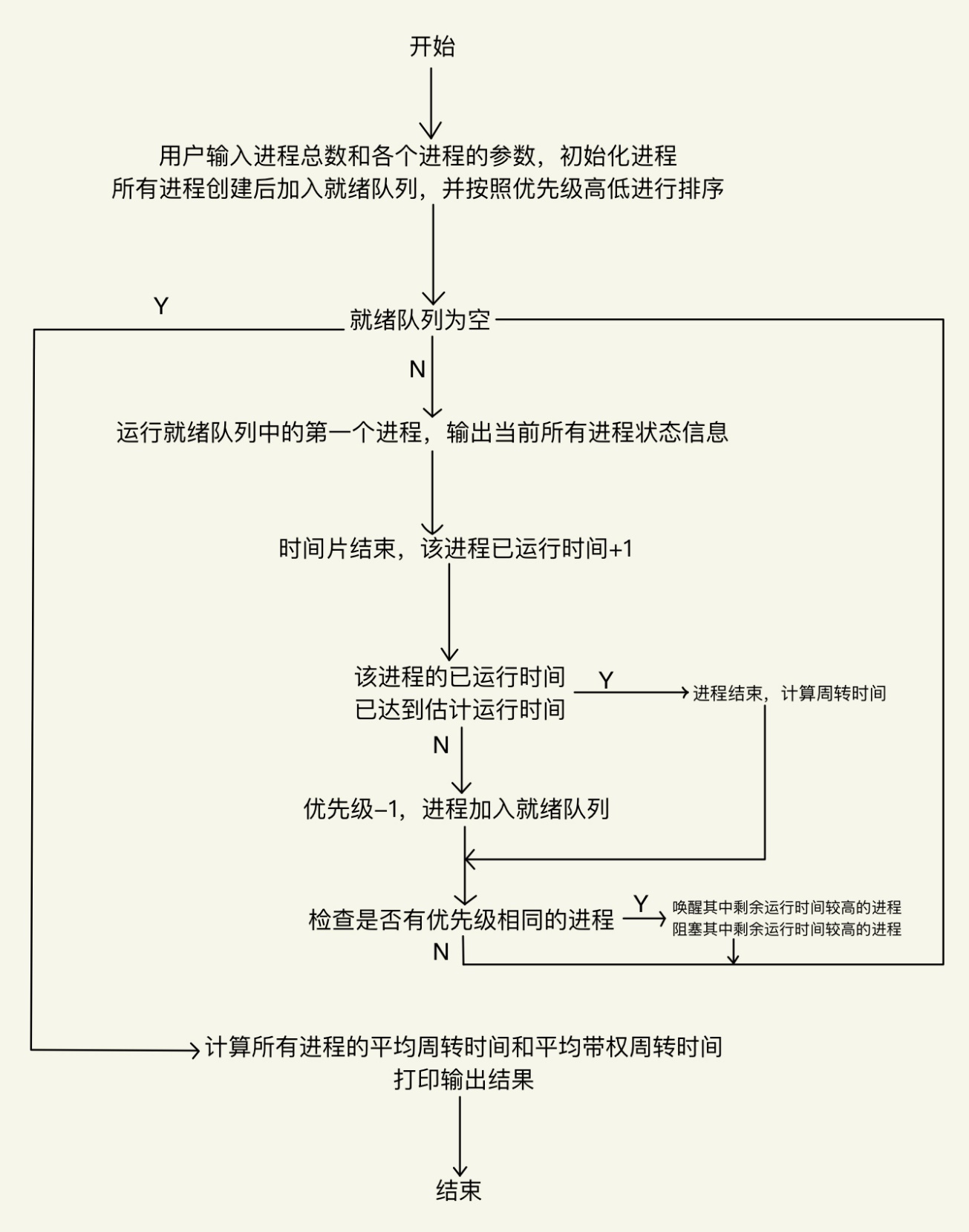
整型数据starttime和finishtime用于标记进程的开始时间和结束运行时间；

整型数据turnaroundtime和浮点型数据weightedturnaroundtime分别表示进程的周转时间和带权周转时间，将在进程结束运行后得出计算结果。

此外，为了调度方便，还定义了四个PCB结构指针：rhead为就绪队列（ready）头指针，指向就绪队列首地址；当前运行进程指针current指向正在运行中的的进程，表示该进程正在被运行；bhead为阻塞队列（block）头指针，指向阻塞队列首地址；thead表示终止队列（terminated）头指针，指向终止进程队列首地址。

### 3.2 程序流程图和带有注释的源程序

流程图：



源程序：

#include <stdio.h>

#include <stdlib.h>

#define N 10 //max length of process name

#define MAX 100 //max capacity of processes

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

static int Time = 0; //time counter of this expriment, initially 0

int TurnaroundTime[MAX] = { 0 };//a global array to record the turnaround time and calculate the average turnaround time at the end of experiment

int Timeflag = 0; //a global variable to record the total count of information recorded in the array above

struct pcb //struct

{

char pname[N]; //process name

int runtime; //estimated running time

int arrivetime; //arrival time

int priority; //priority: the bigger number the higher priority

char state; //process state: Waiting, Running, Blocked, Terminated

struct pcb\* next; //pointer: address to the next pcb

//.......

int elapsedtime; //elapsed running time

int starttime;

int finishtime;

int turnaroundtime;

float weightedturaroundtime;

int firstflag;//0: never been run; 1: being re-running

} //3 pcb pointer

\*rhead = NULL, //address of ready queue

\* current, //address of current running process

\* bhead = NULL,//address of blocked queue

\* thead = NULL; //address of terminated queue

typedef struct pcb PCB;

PCB \*head\_input;

PCB \*head\_run;

void sort() //ready queue by priority

{

PCB\* first, \* second;

int jump = 0; //mark whether there's a queue jump during sort()

if ((rhead == NULL) || ((current->priority) > (rhead->priority)))//readyqueue is empty OR priority of current running process is bigger than any other process

{

//place current process on the head of ready queue

current->next = rhead;

rhead = current;

}

else

{

//get the first and second pcb of readyqueue

first = rhead;

second = first->next;

while (second != NULL) //loop, until second=null

{

if ((current->priority) > (second->priority))

{

//put current between first and second

current->next = second;

first->next = current;

second = NULL; //(to end the loop)

jump = 1; //(queue jump occured)

}

else

{

first = first->next; second = second->next; //push first and second backwards

}

}

if (jump == 0)

{

first->next = current; //place current after first (first is now the end of queue cause loop has already ended and there was no jump)

}

}

}

void inputprocess() //create process list

{

int i, num;

printf("\nPlease enter the number of your processes:");

scanf("%d", &num);

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

{

//scan the process information (user)

printf("Please enter the basic information of Process %d\n", i + 1);

current = getpch(PCB);

printf("\tname of Process %d:", i + 1);

scanf("%s", current->pname);

printf("\tpriority of Process %d:", i + 1);

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

printf("\testimated running time of Process %d:", i + 1);

scanf("%d", &current->runtime);

//initialize the process and link it

current->elapsedtime = 0;

current->state = 'W'; //waiting

current->next = NULL; //cause the newest process is on the end of the list

current->firstflag = 0; //never been run

printf("Process %s has successfully created!\n", current->pname);

printf("\n");

sort();

}

}

//functions to print information about processes

void printtitle()

{

printf("Name\tStatus\tPriority\tRunning time\tElapsed time\n");

}

void printprocess(PCB\* ptr)

{

printf("%s\t", ptr->pname);

printf("%c\t", ptr->state);

printf("%d\t", ptr->priority); printf("\t");

printf("%d\t", ptr->runtime); printf("\t");

printf("%d\t", ptr->elapsedtime);

printf("\n");

}

void printreadylist() //print the ready list

{

PCB\* pr;

pr = rhead;

if (pr == NULL)//ready list is empty

{

printf("####### READY PROCESS: N/A #######\n");

}

else

{

printf("####### READY PROCESS QUEUE #######\n");

printtitle();

while (pr != NULL)//loop until the end

{

printprocess(pr);

pr = pr->next;

}

}

}

void printblocklist() //print the block list

{

PCB\* pb;

pb = bhead;

if (pb == NULL)//block list is empty

{

printf("####### BLOCKED PROCESS: N/A #######\n");

}

else

{

printf("####### BLOCKED PROCESS QUEUE #######\n");

printtitle();

while (pb != NULL)//loop until the end

{

printprocess(pb);

pb = pb->next;

}

}

}

void printall()

{

//printf("\n");

printf("####### CURRENT RUNNING PROCESS - %s #######\n", current->pname);

printtitle();

printprocess(current); //print current running process

printreadylist(); //print ready list

printblocklist(); //print block list

}

void printtime(PCB\* x)//print process information with time's

{

printtitle();

printprocess(x);

printf(" Turnaround time\t Weighted turnaround time\t\n");

printf(" %d\t\t\t %f\n", x->turnaroundtime, x->weightedturaroundtime);

}

void block() //block consecutive processes with the same priority in the readyqueue

{

PCB\* pb, \* index = rhead;

int count = 0; //block-able process: consecutive processes with the same priority in the readyqueue

if (rhead == NULL) return; //ready queue is empty

while ((index->next != NULL)) //loop until the end

{

if ((index->next->priority) >= rhead->priority)

{

index->next->state = 'B'; //state=block

index = index->next; //index backwards

count++; //count the blockable processes

}

else

{

break;

}

}

if (count > 0)

{

//block the process

pb = index->next; //take out a sub-queue of equal-priority processes within the readyqueue

index->next = bhead;//and put the sub-queue into head of blockqueue

bhead = rhead->next;

rhead->next = pb; //and re-link the readyqueue

}

}

void wakeup() //wake up the process with the most priority and least remaining running time in the blockqueue to be READY

{

PCB\* index, \* prior, \* t; //index, prior, temp

int exchange = 0; //mark whether theres need to exchange the place

if (bhead == NULL) return; //blockqueue is empty

if (bhead->next != NULL) //blockqueue has more than one process

{

index = bhead->next;

prior = bhead;

while (index != NULL) //loop until the end

{

if ((index->priority) > (bhead->priority)) //criteria 1: biggest priority

{

exchange = 1;

}

else if ((index->priority) == (bhead->priority)) //criteria 2: least remaining running time

{

if ((index->runtime - index->elapsedtime) < (bhead->runtime - bhead->elapsedtime))

{

exchange = 1;

}

}

if (exchange == 1)

{

//swap with first process in blockqueue

prior->next = bhead;

t = bhead->next;

bhead->next = index->next;

index->next = t;

bhead = index;

exchange = 0;

}

//put prior and index backwards

prior = index;

index = index->next;

}

}

//place the first process in blockqueue to the firstplace in readyqueue

t = bhead->next;

bhead->next = rhead;

rhead = bhead;

bhead = t;

}

int listlength() //get the length of the process list

{

int length = 0; PCB\* pr = rhead;

while (pr != NULL)

{

length++;

pr = pr->next;

}

return length;

}

void destroy() //destroy current running process

{

printf("Process %s is destroyed!\n", current->pname);

free(current); //free memory of current running process

}

//void terminate();

//int readyprocess( );

void runprocess() //运行进程的函数；

{

(current->elapsedtime)++;

if (current->elapsedtime == current->runtime) //process completed, so terminate it

{

current->state = 'C'; //set the process state to Completed

current->finishtime = Time; //set the time of termination

current->turnaroundtime = current->finishtime - current->arrivetime;//calculate the turnaround time

current->weightedturaroundtime = ((float)(current->turnaroundtime)) / ((float)(current->elapsedtime));//calculate the weighted turnaround time

printf("Process %s is terminating...\n", current->pname);

printtime(current);//print all the information before it gets destroyed

//record the turnaround time information of the process into a global array

TurnaroundTime[Timeflag] = current->turnaroundtime;

Timeflag++;

destroy();//free

}

else

{

(current->priority)--; //priority--: dynamic priority

current->state = 'W'; //state=wait

sort();

if (bhead != NULL) //no process blocked

{

wakeup();

}

block();

}

}

//main function

void main()

{

//fp=open（"result.txt”, “w”）;

inputprocess();

int len = listlength();

int round = 0;

char ch;

float totalturnaroundtime = 0.0, averageturnaroundtime;

while ((len != 0) && (rhead != NULL))

{

ch = getchar();

round++;

printf("~~~~~ROUND %d~~~~~\n", round);

//run the first process in the readyqueue

current = rhead;

rhead = current->next;

current->next = NULL;

current->state = 'R';

//set start time

if (current->firstflag == 0)

{

current->starttime = Time;

current->firstflag = 1;

}

printall(); //print all process information

runprocess();//run

Time++; //absolute time

printf("Press any key to continue.....");

ch = getchar();

printf("\n");

}

printf("All the %d processes terminated. ", len);

for (int i = 0; i < len; i++)

{

totalturnaroundtime += TurnaroundTime[i];

}

averageturnaroundtime = totalturnaroundtime / (float)len;

printf("The average turnaround time = %f.\n", averageturnaroundtime);

printf("\n.....EXPERIMENT 3 DONE.....Press any key to end.....\n");

ch = getchar();

return;

}

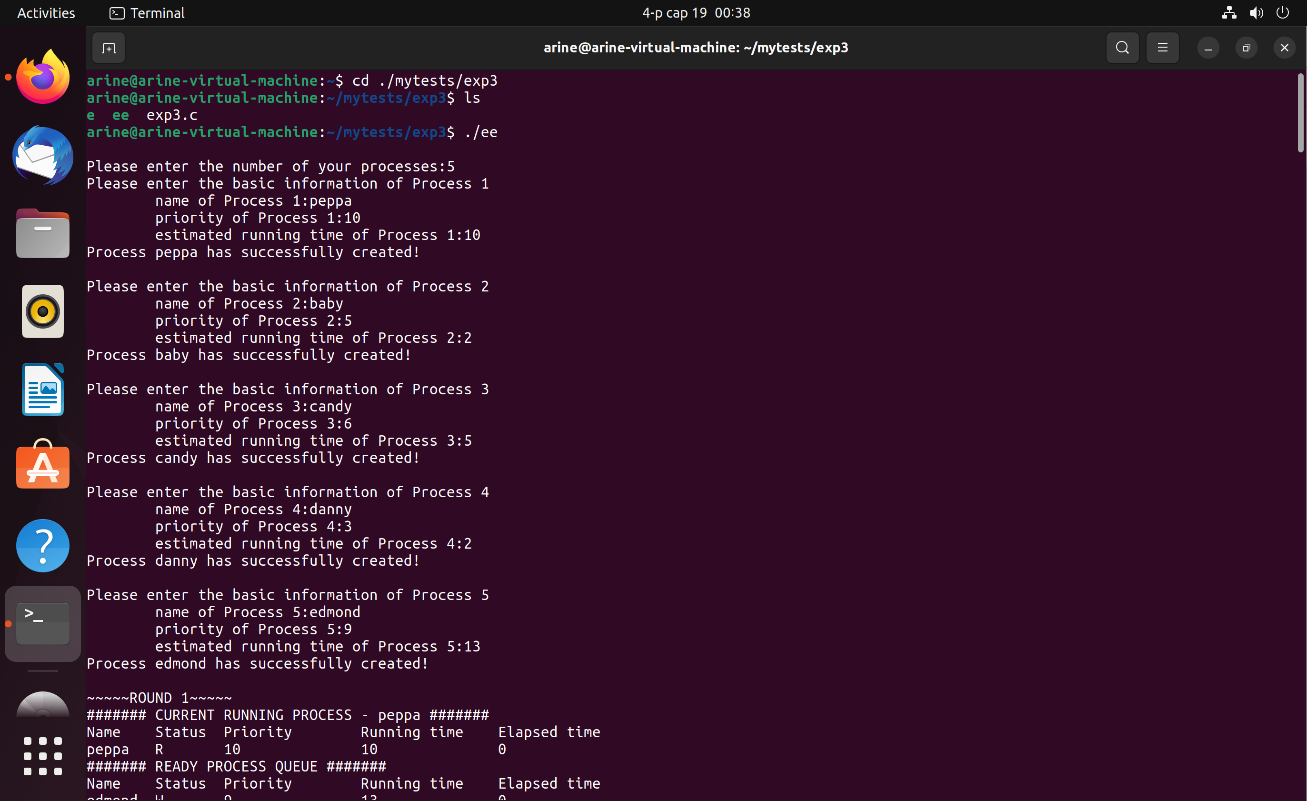
### 3.3 执行程序名，并打印程序运行时的初值和运行结果，其中包括：

### 3.3.1 各进程控制块的初始状态

输入各个进程的信息，我们首先假定此次实验的总进程数量为5，其次依次输入peppa, baby, candy, danny, edmond五个进程并为其设置优先级和预估运行时间：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 进程序号 | 进程名称 | 优先级 | 运行时间 | 初始状态 | Next指针 |
| 1 | Peppa | 10 | 10 | W | Baby |
| 2 | Baby | 5 | 2 | W | Candy |
| 3 | Candy | 6 | 5 | W | Danny |
| 4 | Danny | 3 | 2 | W | Edmond |
| 5 | Edmond | 9 | 13 | W | NULL |

如下所示：



### 3.3.2 选中运行进程的名字、运行后各进程控制块状态以及每次调度时，就绪队列的进程排列顺序

接下来我们开始运行程序：

本实验采用动态优先级处理机调度：处理机调度时，总是选择已经到达队列的优先级最高的进程运行，进程每运行一次，其优先级加1、估计运行时间减1。由于本实验是模拟实验，所以对选中进程并不实际启动运行，而只是执行，用这两个操作来模拟进程的一次运行。

进程运行一次后，若剩余时间不为0，且其优先级低于就绪队列的其他进程优先级，则选择一个高优先级进程抢占CPU。

每次运行后的进程队列将输出在屏幕上，包括进程名称、进程状态、实时优先级、剩余运行时间、已运行时间。若就绪队列不空，则重复上述的步骤直到所有进程成为完成状态。

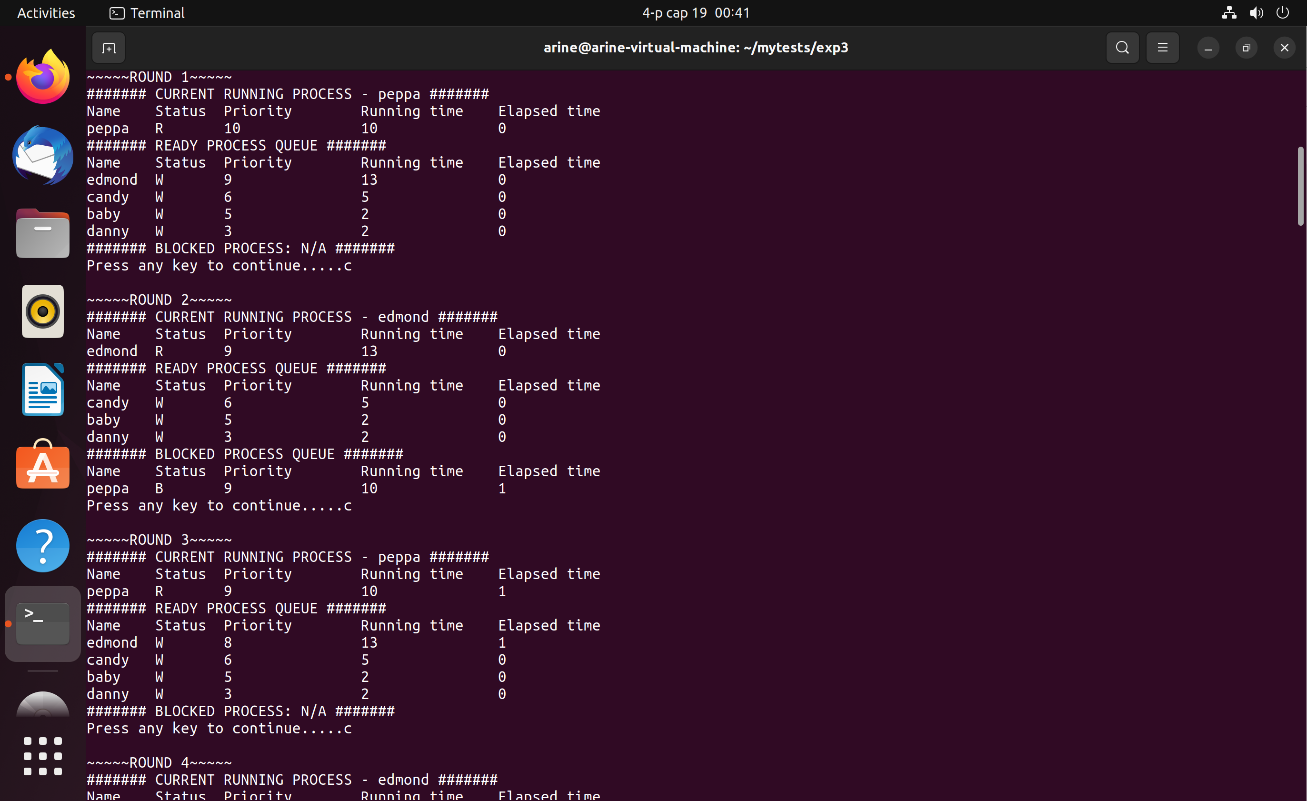
当进程终止（即剩余时间为0）后，我们把它的状态改为完成状态“C”并撤出就绪队列，然后计算并在屏幕输出进程此次的所有信息包括周转时间、带权周转时间。当所有进程结束运行后，将计算并在屏幕输出所有进程的平均周转时间。公式如下：

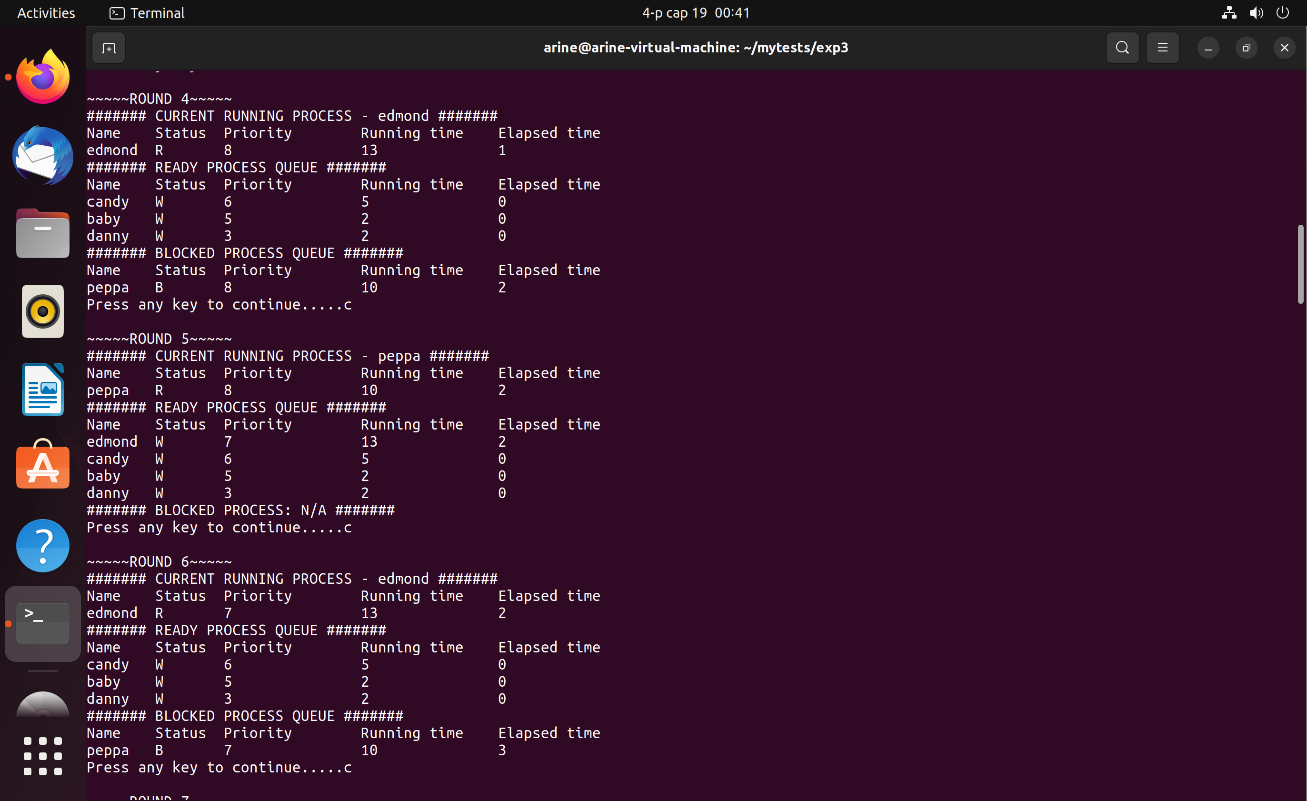
周转时间=作业完成时刻-作业到达时刻；

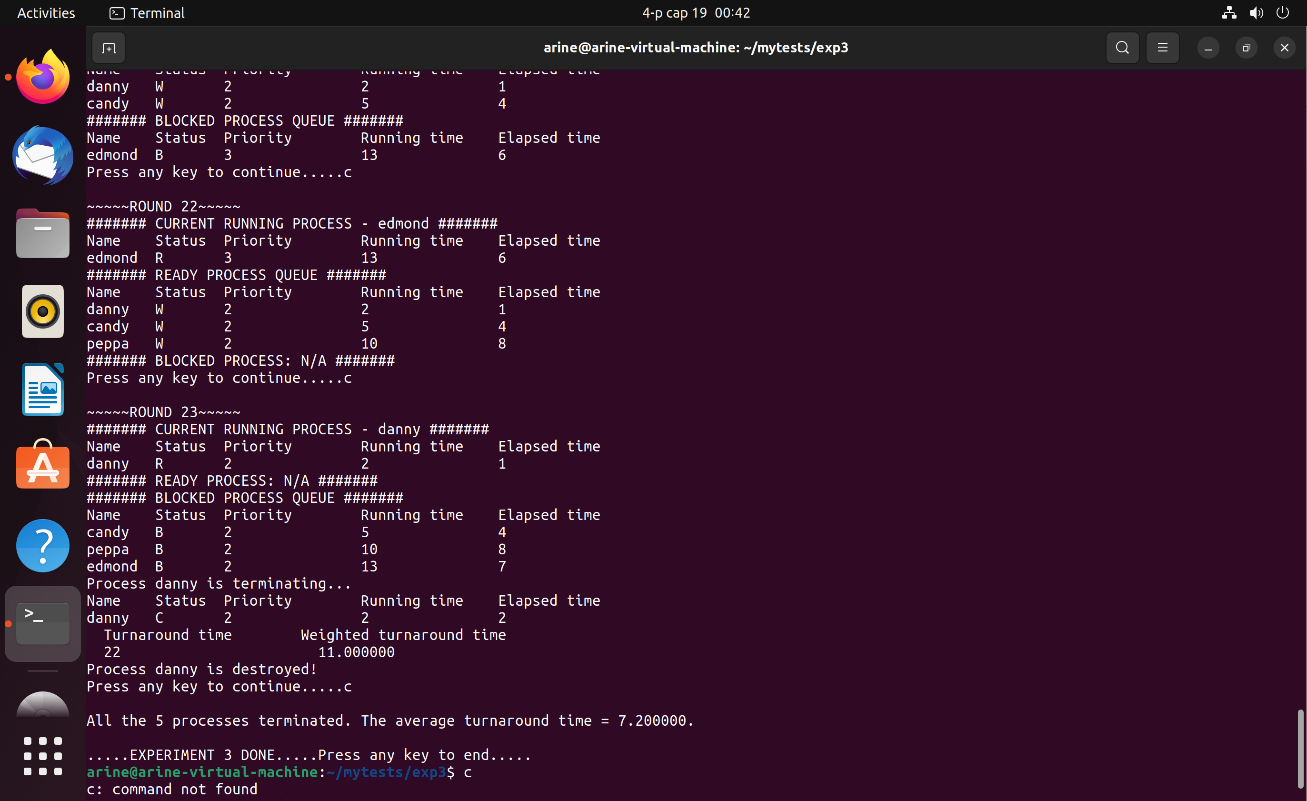
带权周转时间=周转时间/服务时间；

平均周转时间=作业周转总时间/作业个数。

运行结果如下所示：







中间的截图略。本次程序运行全部结果（exp3output.txt）：

Please enter the number of your processes:5

Please enter the basic information of Process 1

name of Process 1:peppa

priority of Process 1:10

estimated running time of Process 1:10

Process peppa has successfully created!

Please enter the basic information of Process 2

name of Process 2:baby

priority of Process 2:5

estimated running time of Process 2:2

Process baby has successfully created!

Please enter the basic information of Process 3

name of Process 3:candy

priority of Process 3:6

estimated running time of Process 3:5

Process candy has successfully created!

Please enter the basic information of Process 4

name of Process 4:danny

priority of Process 4:3

estimated running time of Process 4:2

Process danny has successfully created!

Please enter the basic information of Process 5

name of Process 5:edmond

priority of Process 5:9

estimated running time of Process 5:13

Process edmond has successfully created!

~~~~~ROUND 1~~~~~

####### CURRENT RUNNING PROCESS - peppa #######

Name Status Priority Running time Elapsed time

peppa R 10 10 0

####### READY PROCESS QUEUE #######

Name Status Priority Running time Elapsed time

edmond W 9 13 0

candy W 6 5 0

baby W 5 2 0

danny W 3 2 0

####### BLOCKED PROCESS: N/A #######

Press any key to continue.....c

~~~~~ROUND 2~~~~~

####### CURRENT RUNNING PROCESS - edmond #######

Name Status Priority Running time Elapsed time

edmond R 9 13 0

####### READY PROCESS QUEUE #######

Name Status Priority Running time Elapsed time

candy W 6 5 0

baby W 5 2 0

danny W 3 2 0

####### BLOCKED PROCESS QUEUE #######

Name Status Priority Running time Elapsed time

peppa B 9 10 1

Press any key to continue.....c

~~~~~ROUND 3~~~~~

####### CURRENT RUNNING PROCESS - peppa #######

Name Status Priority Running time Elapsed time

peppa R 9 10 1

####### READY PROCESS QUEUE #######

Name Status Priority Running time Elapsed time

edmond W 8 13 1

candy W 6 5 0

baby W 5 2 0

danny W 3 2 0

####### BLOCKED PROCESS: N/A #######

Press any key to continue.....c

~~~~~ROUND 4~~~~~

####### CURRENT RUNNING PROCESS - edmond #######

Name Status Priority Running time Elapsed time

edmond R 8 13 1

####### READY PROCESS QUEUE #######

Name Status Priority Running time Elapsed time

candy W 6 5 0

baby W 5 2 0

danny W 3 2 0

####### BLOCKED PROCESS QUEUE #######

Name Status Priority Running time Elapsed time

peppa B 8 10 2

Press any key to continue.....c

~~~~~ROUND 5~~~~~

####### CURRENT RUNNING PROCESS - peppa #######

Name Status Priority Running time Elapsed time

peppa R 8 10 2

####### READY PROCESS QUEUE #######

Name Status Priority Running time Elapsed time

edmond W 7 13 2

candy W 6 5 0

baby W 5 2 0

danny W 3 2 0

####### BLOCKED PROCESS: N/A #######

Press any key to continue.....c

~~~~~ROUND 6~~~~~

####### CURRENT RUNNING PROCESS - edmond #######

Name Status Priority Running time Elapsed time

edmond R 7 13 2

####### READY PROCESS QUEUE #######

Name Status Priority Running time Elapsed time

candy W 6 5 0

baby W 5 2 0

danny W 3 2 0

####### BLOCKED PROCESS QUEUE #######

Name Status Priority Running time Elapsed time

peppa B 7 10 3

Press any key to continue.....c

~~~~~ROUND 7~~~~~

####### CURRENT RUNNING PROCESS - peppa #######

Name Status Priority Running time Elapsed time

peppa R 7 10 3

####### READY PROCESS QUEUE #######

Name Status Priority Running time Elapsed time

candy W 6 5 0

edmond W 6 13 3

baby W 5 2 0

danny W 3 2 0

####### BLOCKED PROCESS: N/A #######

Press any key to continue.....c

~~~~~ROUND 8~~~~~

####### CURRENT RUNNING PROCESS - candy #######

Name Status Priority Running time Elapsed time

candy R 6 5 0

####### READY PROCESS QUEUE #######

Name Status Priority Running time Elapsed time

baby W 5 2 0

danny W 3 2 0

####### BLOCKED PROCESS QUEUE #######

Name Status Priority Running time Elapsed time

edmond B 6 13 3

peppa B 6 10 4

Press any key to continue.....c

~~~~~ROUND 9~~~~~

####### CURRENT RUNNING PROCESS - peppa #######

Name Status Priority Running time Elapsed time

peppa R 6 10 4

####### READY PROCESS QUEUE #######

Name Status Priority Running time Elapsed time

baby W 5 2 0

candy W 5 5 1

danny W 3 2 0

####### BLOCKED PROCESS QUEUE #######

Name Status Priority Running time Elapsed time

edmond B 6 13 3

Press any key to continue.....c

~~~~~ROUND 10~~~~~

####### CURRENT RUNNING PROCESS - edmond #######

Name Status Priority Running time Elapsed time

edmond R 6 13 3

####### READY PROCESS QUEUE #######

Name Status Priority Running time Elapsed time

baby W 5 2 0

candy W 5 5 1

peppa W 5 10 5

danny W 3 2 0

####### BLOCKED PROCESS: N/A #######

Press any key to continue.....c

~~~~~ROUND 11~~~~~

####### CURRENT RUNNING PROCESS - baby #######

Name Status Priority Running time Elapsed time

baby R 5 2 0

####### READY PROCESS QUEUE #######

Name Status Priority Running time Elapsed time

danny W 3 2 0

####### BLOCKED PROCESS QUEUE #######

Name Status Priority Running time Elapsed time

candy B 5 5 1

peppa B 5 10 5

edmond B 5 13 4

Press any key to continue.....c

~~~~~ROUND 12~~~~~

####### CURRENT RUNNING PROCESS - candy #######

Name Status Priority Running time Elapsed time

candy R 5 5 1

####### READY PROCESS QUEUE #######

Name Status Priority Running time Elapsed time

baby W 4 2 1

danny W 3 2 0

####### BLOCKED PROCESS QUEUE #######

Name Status Priority Running time Elapsed time

peppa B 5 10 5

edmond B 5 13 4

Press any key to continue.....c

~~~~~ROUND 13~~~~~

####### CURRENT RUNNING PROCESS - peppa #######

Name Status Priority Running time Elapsed time

peppa R 5 10 5

####### READY PROCESS QUEUE #######

Name Status Priority Running time Elapsed time

baby W 4 2 1

candy W 4 5 2

danny W 3 2 0

####### BLOCKED PROCESS QUEUE #######

Name Status Priority Running time Elapsed time

edmond B 5 13 4

Press any key to continue.....c

~~~~~ROUND 14~~~~~

####### CURRENT RUNNING PROCESS - edmond #######

Name Status Priority Running time Elapsed time

edmond R 5 13 4

####### READY PROCESS QUEUE #######

Name Status Priority Running time Elapsed time

baby W 4 2 1

candy W 4 5 2

peppa W 4 10 6

danny W 3 2 0

####### BLOCKED PROCESS: N/A #######

Press any key to continue.....c

~~~~~ROUND 15~~~~~

####### CURRENT RUNNING PROCESS - baby #######

Name Status Priority Running time Elapsed time

baby R 4 2 1

####### READY PROCESS QUEUE #######

Name Status Priority Running time Elapsed time

danny W 3 2 0

####### BLOCKED PROCESS QUEUE #######

Name Status Priority Running time Elapsed time

candy B 4 5 2

peppa B 4 10 6

edmond B 4 13 5

Process baby is terminating...

Name Status Priority Running time Elapsed time

baby C 4 2 2

Turnaround time Weighted turnaround time

14 7.000000

Process baby is destroyed!

Press any key to continue.....c

~~~~~ROUND 16~~~~~

####### CURRENT RUNNING PROCESS - danny #######

Name Status Priority Running time Elapsed time

danny R 3 2 0

####### READY PROCESS: N/A #######

####### BLOCKED PROCESS QUEUE #######

Name Status Priority Running time Elapsed time

candy B 4 5 2

peppa B 4 10 6

edmond B 4 13 5

Press any key to continue.....c

~~~~~ROUND 17~~~~~

####### CURRENT RUNNING PROCESS - candy #######

Name Status Priority Running time Elapsed time

candy R 4 5 2

####### READY PROCESS QUEUE #######

Name Status Priority Running time Elapsed time

danny W 2 2 1

####### BLOCKED PROCESS QUEUE #######

Name Status Priority Running time Elapsed time

peppa B 4 10 6

edmond B 4 13 5

Press any key to continue.....c

~~~~~ROUND 18~~~~~

####### CURRENT RUNNING PROCESS - peppa #######

Name Status Priority Running time Elapsed time

peppa R 4 10 6

####### READY PROCESS QUEUE #######

Name Status Priority Running time Elapsed time

candy W 3 5 3

danny W 2 2 1

####### BLOCKED PROCESS QUEUE #######

Name Status Priority Running time Elapsed time

edmond B 4 13 5

Press any key to continue.....c

~~~~~ROUND 19~~~~~

####### CURRENT RUNNING PROCESS - edmond #######

Name Status Priority Running time Elapsed time

edmond R 4 13 5

####### READY PROCESS QUEUE #######

Name Status Priority Running time Elapsed time

candy W 3 5 3

peppa W 3 10 7

danny W 2 2 1

####### BLOCKED PROCESS: N/A #######

Press any key to continue.....c

~~~~~ROUND 20~~~~~

####### CURRENT RUNNING PROCESS - candy #######

Name Status Priority Running time Elapsed time

candy R 3 5 3

####### READY PROCESS QUEUE #######

Name Status Priority Running time Elapsed time

danny W 2 2 1

####### BLOCKED PROCESS QUEUE #######

Name Status Priority Running time Elapsed time

peppa B 3 10 7

edmond B 3 13 6

Press any key to continue.....c

~~~~~ROUND 21~~~~~

####### CURRENT RUNNING PROCESS - peppa #######

Name Status Priority Running time Elapsed time

peppa R 3 10 7

####### READY PROCESS QUEUE #######

Name Status Priority Running time Elapsed time

danny W 2 2 1

candy W 2 5 4

####### BLOCKED PROCESS QUEUE #######

Name Status Priority Running time Elapsed time

edmond B 3 13 6

Press any key to continue.....c

~~~~~ROUND 22~~~~~

####### CURRENT RUNNING PROCESS - edmond #######

Name Status Priority Running time Elapsed time

edmond R 3 13 6

####### READY PROCESS QUEUE #######

Name Status Priority Running time Elapsed time

danny W 2 2 1

candy W 2 5 4

peppa W 2 10 8

####### BLOCKED PROCESS: N/A #######

Press any key to continue.....c

~~~~~ROUND 23~~~~~

####### CURRENT RUNNING PROCESS - danny #######

Name Status Priority Running time Elapsed time

danny R 2 2 1

####### READY PROCESS: N/A #######

####### BLOCKED PROCESS QUEUE #######

Name Status Priority Running time Elapsed time

candy B 2 5 4

peppa B 2 10 8

edmond B 2 13 7

Process danny is terminating...

Name Status Priority Running time Elapsed time

danny C 2 2 2

Turnaround time Weighted turnaround time

22 11.000000

Process danny is destroyed!

Press any key to continue.....c

All the 5 processes terminated. The average turnaround time = 7.200000.

.....EXPERIMENT 3 DONE.....Press any key to end.....

### 3.4 总结本次实验的心得与体会。

本实验为单机模拟进程调度算法，在程序设计时不需真正地建立线程或进程。为了清楚地观察诸进程的调度过程，程序中应有显示或打印语句，能显示或打印每次被选中进程的进程名，以及运行一次后进程队列的变化。实验中我利用指针结构连接成循环链表，并且采用头指针）的思路，每一步修改当前指针的下一指针结构的内容，以避免在运算过程中需要改变循环结构时（即某一进程完成后将其从循环队列中剔除）的问题，完成了一个按优先级高低实现处理机调度的程序。

通过此次实验，我对进程控制块（PCB）中相关参数，以及存储的数据结构有了较为清晰的认识，同时对于按优先级调度的算法有了更深入的理解。

进程调度算法是操作系统的核心组成部分之一，需要考虑各种因素如进程状态、优先级、I/O操作等，同时还要考虑进程同步和互斥问题，确保系统的可靠性和性能。因此，设计和实现一个高效的进程调度算法是一项需要技术和经验的挑战。

优先级调度算法是一种常见的进程调度算法，它根据进程的优先级来分配处理机时间，优先级高的进程先被调度执行。这种算法可以保证高优先级的进程获得更多的处理机时间，提高了系统的响应速度和效率，但如果优先级不合理或者存在饥饿情况，低优先级的进程可能会一直等待处理机时间而无法得到执行。因此，在设计和实现处理及调度算法时，需要充分考虑各种情况，确保系统的公平性和高效性。

## 四、实验指导

在Linux环境下， 用C语言编写，程序中的数据结构定义和程序主框架如下。

typedef struct pcb //进程控制块定义

{ char pname[N]; //进程名

int runtime; //运行时间

int arrivetime; //到达时间

char state; //进程状态

int priority; //进程优先级

int   finishtime;   //进程结束时间

struct pcb \*next; //链接指针

……

}PCB;

PCB \*head\_input; //就绪队列头指针

PCB \*head\_run; //运行进程指针

……

void inputprocess( ); //建立进程，输入进程数目N，输入各个进程信息，并建立链表;

void printreadylist( ); //输出就绪队列

int readyprocess( )； //检查就绪队列并运行进程

int runprocess( ); //运行进程的函数；

……

void main() //主函数

{ //fp=open（"result.txt”, “w”）;

inputprocess();

readyprocess();

……

getch();

//fclose(fp);

}

void inputprocess( ) //建立进程，输入进程数目N，输入各个进程信息，并建立链表;

{ // 输入需要运行的进程数量

…..

// 输入进程信息：进程名，运行时间，到达时间等信息

…...

}

int readyprocess( ) //检查就绪队列并运行进程

{ // 就绪队列为空时，结束运行

…..

// 取出一个进程运行它,例如,如果是时间片调度算法，则将时间减1

…..

// 修改进程的状态

……

// 输出进程运行信息

}