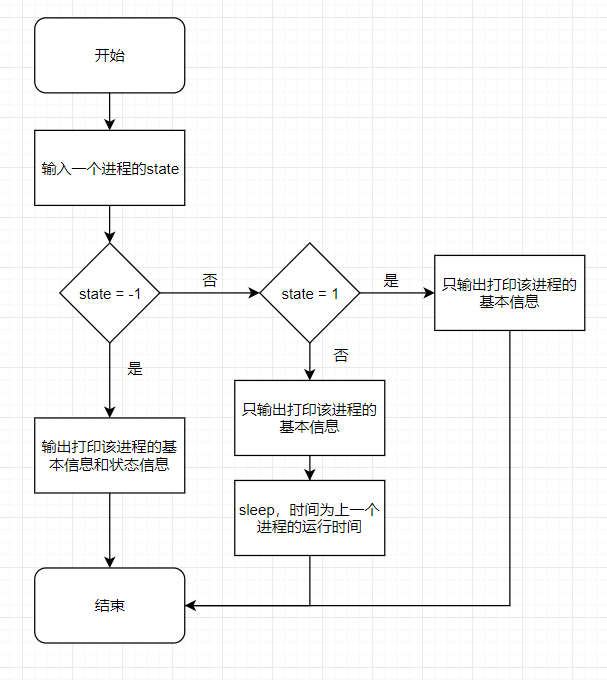
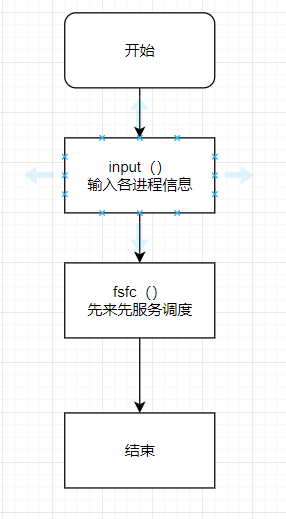
## 实验 处理机调度

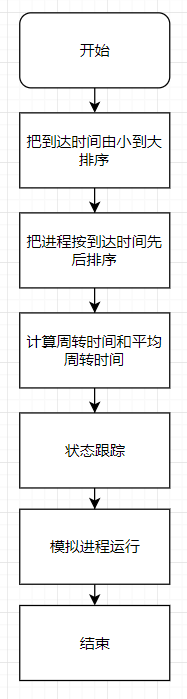
1. 实验要求
   1. 进程个数至少5个以上，也可让用户动态输入
   2. 每个进程由一个进程控制块来标识，进程控制块的内容根据情况自己设计（但至少要有进程名、进程状态、到达时间、估计运行时间信息）
   3. 调度算法设计
      1. 来先服务调度算法：调度时，总是选择到达时间最早的进程
      2. 基于优先级的调度算法：调度时，总是选择优先级最高的进程
2. 运行环境
   1. Linux内核：Linux version 3.13.0-24-generic (buildd@roseapple)
   2. Ubuntu：Ubuntu 4.8.2-19ubuntu1
3. 使用工具
   1. Gcc编译器：gcc version 4.8.2
   2. Gedit编辑器：gedit - Version 3.10.4
4. 流程图
   1. 模拟进程运行



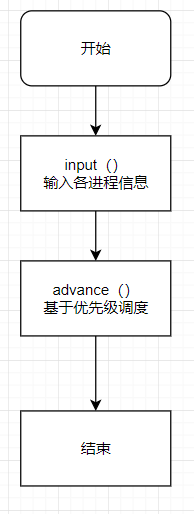
* 1. 先来先服务
     1. 主函数



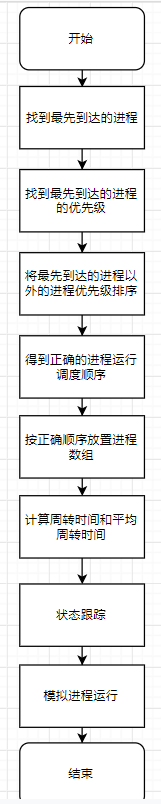
* + 1. 先来先服务调度算法



* 1. 基于优先级（非抢占式）
     1. 主函数



* + 1. 基于优先级调度算法



1. 源程序
   1. 先来先服务

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#define N 5

typedef struct

{

char pname[20];

int state;//1-正在运行，0-就绪，-1-运行结束

int arrival\_time;

int run\_time;

int start\_time;

int end\_time;

int cycle\_time;

}pcb;

//返回一个进程

pcb \*m(int i)

{

pcb \*p;

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

printf("please input name,state,arrival\_time and run\_time of the %dth process :\n",i+1);

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

scanf("%d",&(p->arrival\_time));

scanf("%d",&(p->run\_time));

return p;

}

//打印进程的基本信息或状态信息

void pp(pcb \*p,int state)

{

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

printf("%d\t",p->state);

printf(" %d\t",p->arrival\_time);

printf("\t%d\t",p->run\_time);

if(state==-1)

{

printf("%d\t",p->start\_time);

printf(" %d\t",p->end\_time);

printf("\t%d\t\n",p->cycle\_time);

}

else

printf("\n");

}

//初始化进程数组

void input(pcb \*p[N])

{

int i;

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

{

p[i]=m(i);

}

}

//冒泡排序

void maopao(int a[N],int n)

{

int i,j,t;

for(i=n-1;i>0;i--)

{

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

{

if(a[j]>a[j+1])

{

t=a[j];

a[j]=a[j+1];

a[j+1]=t;

}

}

}

}

//先来先服务调度函数

void fcfs(pcb \*p[N])

{

int i,j,k;

int b[N];

pcb \*bb[N];

float avg=0;

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

{

b[i]=p[i]->arrival\_time;

}

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

{

printf("%d ",b[i]);

}

maopao(b,N); //把到达时间排序

printf("\n");

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

{

printf("%d ",b[i]);

}

printf("\n");

printf("pname state arrival\_time run\_time start\_time end\_time cycle\_time\n");

//把进程按到达时间的先后排序

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

{

for(j=0;j<N;j++)

{

if(b[i]==p[j]->arrival\_time)

{

bb[i]=p[j];

}

}

}

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

{

if(i==0)

{

bb[i]->start\_time=bb[i]->arrival\_time;

}

else

{

bb[i]->start\_time=bb[i-1]->end\_time;

}

bb[i]->end\_time=bb[i]->start\_time+bb[i]->run\_time;

bb[i]->cycle\_time=bb[i]->end\_time-bb[i]->arrival\_time;//周转时间

avg+=((float)bb[i]->cycle\_time/N);//平均周转时间

}

//状态跟踪

for(i=0;i<N+1;i++)

{

for(j=0;j<N;j++)

{

if(i==j)

{

bb[j]->state=1;

}

else if(i>j)

{

bb[j]->state=-1;

}

else

{

bb[j]->state=0;

}

pp(bb[j],bb[j]->state);

}

printf("\n");

if(i<N)

sleep(bb[i]->run\_time);//模拟某进程运行

}

printf("the averege value of cycle\_times is: %f",avg);

}

int main()

{

pcb \*p[N];

input(p);

fcfs(p);

printf("\n");

return 0;

}

* 1. 基于优先级（非抢占式）

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#define N 5

typedef struct

{

char pname[20];

int state; //1-正在运行，0-就绪，-1-运行结束

int arrival\_time;

int run\_time;

int start\_time;

int end\_time;

int cycle\_time;

int priority;//优先级

}pcb;

//返回一个进程

pcb \*m(int i)

{

pcb \*p;

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

printf("please input name,state,arrival\_time and run\_time of the %dth process :\n",i+1);

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

scanf("%d",&(p->arrival\_time));

scanf("%d",&(p->run\_time));

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

return p;

}

//输出打印进程的基本信息或状态信息

void pp(pcb \*p,int state)

{

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

printf("%d\t",p->state);

printf(" %d\t",p->arrival\_time);

printf("\t%d\t",p->run\_time);

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

if(state==-1)

{

printf(" %d\t",p->start\_time);

printf("\t%d\t",p->end\_time);

printf(" %d\t\n",p->cycle\_time);

}

else

printf("\n");

}

//初始化进程数组

void input(pcb \*p[N])

{

int i;

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

{

p[i]=m(i);

}

}

//冒泡排序

void maopao(int a[N],int n)

{

int i,j,t;

for(i=n-1;i>0;i--)

{

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

{

if(a[j]>a[j+1])

{

t=a[j];

a[j]=a[j+1];

a[j+1]=t;

}

}

}

}

//基于优先级的调度函数

void advance(pcb \*p[N])

{

int i,j,k;

int b[N-1],c[N];

pcb \*bb[N];

float avg=0;

int f=0;

int min=p[0]->arrival\_time;

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

{

if(p[i]->arrival\_time<min)

{

min=p[i]->arrival\_time;//找到最先到达的进程

}

}

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

{

if(p[i]->arrival\_time==min)

{

f=p[i]->priority;// 找到最先到达的进程的优先级

}

}

printf("%d %d\n",min,f);

for(i=0,j=0;i<N;i++)

{

if(f!=p[i]->priority)

{

b[j]=p[i]->priority;

j++;

}

}

maopao(b,N-1);//将最先到达的进程以外的进程优先级排序

for(i=0;i<N-1;i++)

{

printf("%d ",b[i]);

}

printf("\n");

//得到正确的进程调度运行顺序

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

{

if(i==0)

c[i]=f;

else

c[i]=b[i-1];

}

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

{

printf("%d ",c[i]);

}

printf("\n");

printf("pname state arrival\_time run\_time priority start\_time end\_time cycle\_time\n");

//按正确顺序放置进程数组

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

{

for(j=0;j<N;j++)

{

if(c[i]==p[j]->priority)

{

bb[i]=p[j];

}

}

}

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

{

if(i==0)

{

bb[i]->start\_time=bb[i]->arrival\_time;

}

else

{

bb[i]->start\_time=bb[i-1]->end\_time;

}

bb[i]->end\_time=bb[i]->start\_time+bb[i]->run\_time;

bb[i]->cycle\_time=bb[i]->end\_time-bb[i]->arrival\_time;//周转时间

avg+=((float)bb[i]->cycle\_time/N);//平均周转时间

}

//状态跟踪

for(i=0;i<N+1;i++)

{

for(j=0;j<N;j++)

{

if(i==j)

{

bb[j]->state=1;

}

else if(i>j)

{

bb[j]->state=-1;

}

else

{

bb[j]->state=0;

}

pp(bb[j],bb[j]->state);

}

printf("\n");

if(i<N)

sleep(bb[i]->run\_time);//模拟某进程运行

}

printf("the averege value of cycle\_times is: %f",avg);

}

int main()

{

pcb \*p[N];

input(p);

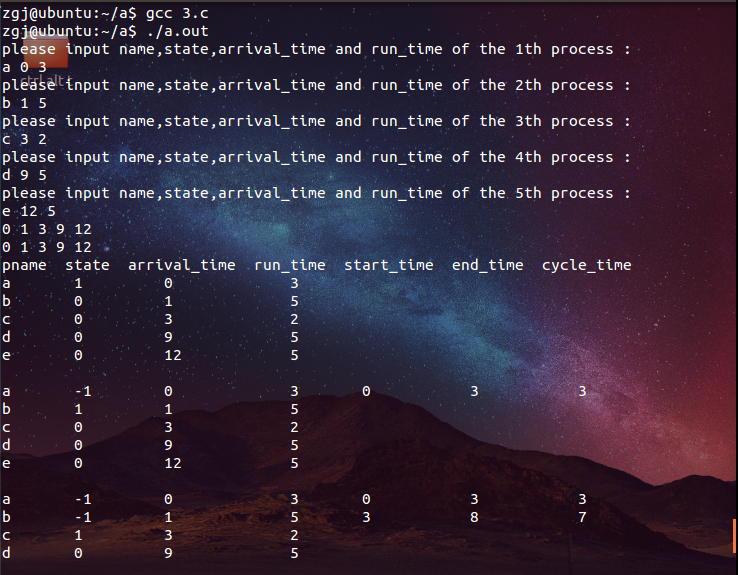
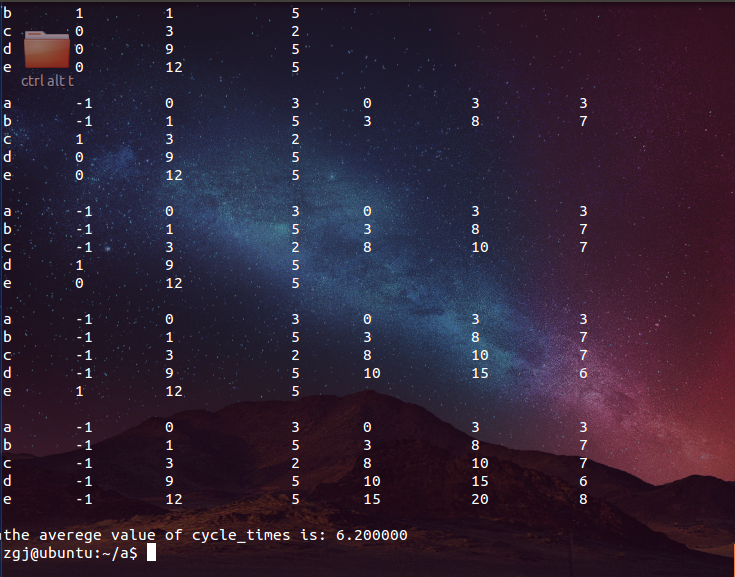
advance(p);

printf("\n");

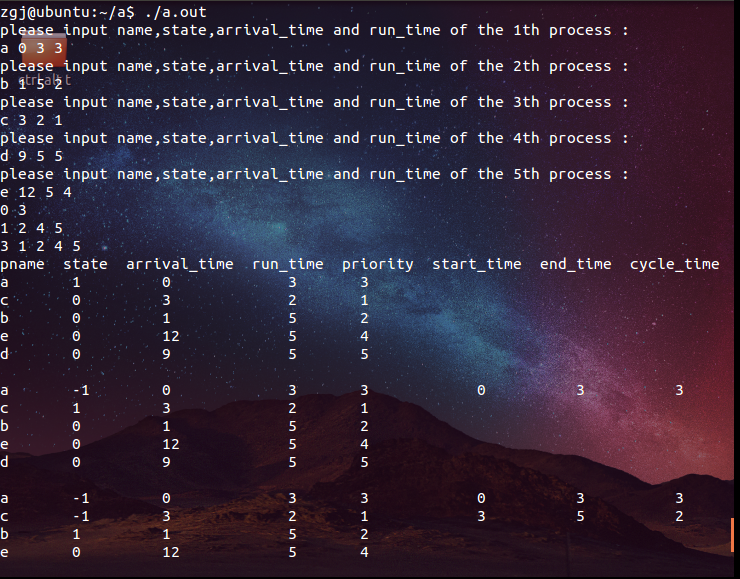
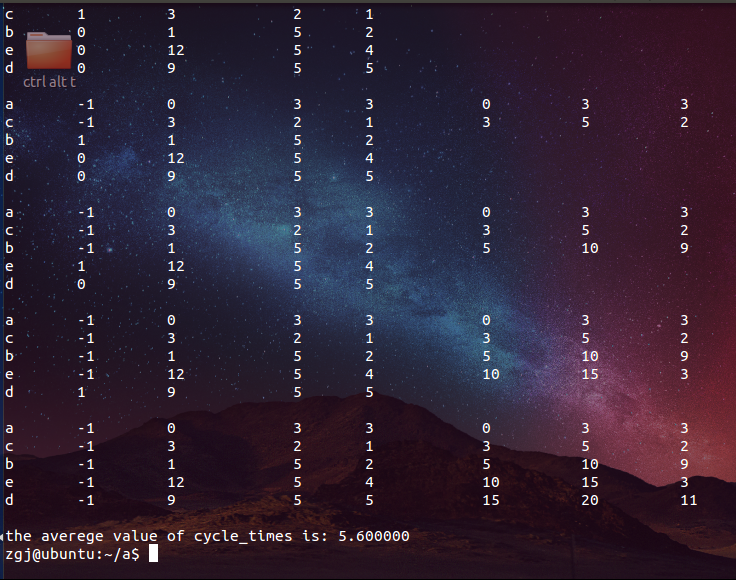
return 0;

}

1. 运行结果
   1. 先来先服务

* 1. 基于优先级（非抢占式）

1. 实验分析
   1. 先来先服务：系统按照作业到达的先后次序来进行调度，或者说它优先考虑在系统中等待时间最长的作业，而不管该作业所需执行时间的长短，从后备作业队列中选择几个最先进入该队列的作业，将它们调入内存，为它们分配资源和创建进程。然后把它放入就绪队列。
   2. 基于优先级：

最高优先级调度算法原则上总是调度就绪队列中优先级最高的那个进程。非抢占式和抢占式进程调度都属于最高优先级进程调度。采用非抢占式最高优先级调度算法，当就绪队列中某进程的最高优先级高于正在处理器中运行的进程的最高优先级时，并不会让正在运行的进程退出处理器，而是将高优先数的排在就绪队列的首部。而采用抢占式最高优先级进程调度算法，则高优先数的进程会抢占处理器，让正在处理的进程处于就绪队列。