# 操作系统实验之一--进程调度算法的模拟实现

原创

takefetter 于 2018-01-11 13:22:13 发布 18072 收藏 118
分类专栏： 操作系统 文章标签： 实验 操作系统实验 进程调度算法

本文链接：https://blog.csdn.net/qq_26667901/article/details/79033038
版权

操作系统 专栏收录该内容
3 篇文章 2 订阅
订阅专栏

发一些大三操作系统的实验代码吸引阅读量吧，当时做实验的时候看见网上很多人写的代码并不好，而且很多人都有错误的地方。如果好的话希望能点赞关注。

常见的进程调度算法有：先来先服务 （FCFS，first come first served） 、最短作业优先（SJF, Shortest Job First） 、

时间片轮转算法（RR，Round-Robin） 、多级反馈队列(Multilevel Feedback Queue) 、

最高响应比优先法(HRRN，Highest Response Ratio Next)

**在本C++程序中模拟实现了FCFS，SJF和RR算法，还有一个简单的优先级算法（仅有优先级的比较，类似于短作业优先算法），排序部分都使用的冒泡排序（我知道时间复杂度很大），代码写**代码如下：

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <string>
#include <cstdio>
#include <iomanip>//setw
#include <cstdlib>//malloc
#include <queue>
using namespace std;
struct FCFS_SJF_r//先来先服务FCFS，短作业优先算法SJF，优先级算法r的结构体
{
 char name;   //进程名
 float arrivetime; //到达时间
 float servetime; //服务时间
 float finishtime; //完成时间
 float roundtime; //周转时间
 float daiquantime; //带权周转时间
 struct FCFS_SJF_r *link;//结构体指针
};
FCFS_SJF_r *head = NULL;
FCFS_SJF_r a[100];//初始化指针和数组
struct FCFS_SJF_r *sortarrivetime(struct FCFS_SJF_r a[], int n); //声明到达时间冒泡排序函数
struct FCFS_SJF_r *sortservetime(struct FCFS_SJF_r a[], int n);//声明服务时间冒泡排序函数
void FCFS(struct FCFS_SJF_r a[], int n, float &t1, float &t2);//先来先服务算法
void SJF(struct FCFS_SJF_r a[], int n, float &t1, float &t2);//短作业优先调度算法
void r(struct FCFS_SJF_r a[], int n, float &t1, float &t2);//优先级算法
struct FCFS_SJF_r *sortarrivetime(struct FCFS_SJF_r a[], int n)//按到达时间进行冒泡排序
{
 int i, j;
 struct FCFS_SJF_r t;
 int flag;
 for (i = 1; i<n; i++)
 {
  flag = 0;
  for (j = 0; j<n - i; j++)
  {
   if (a[j].arrivetime>a[j + 1].arrivetime) //将到达时间短的交换到前边
   {
    t = a[j];
    a[j] = a[j + 1];
    a[j + 1] = t;
    flag = 1;//交换
   }
  }
  if (flag == 0)//如果一趟排序中没发生任何交换，则排序结束
  {
   break;
  }
 }
 return a; //返回排序后进程数组
}
//按服务时间进行冒泡排序
struct FCFS_SJF_r *sortservetime(struct FCFS_SJF_r a[], int n)
{
 int i, j;
 struct FCFS_SJF_r t;
 int flag;
 for (i = 1; i<n; i++)
 {
  flag = 0;
  for (j = 0; j<n - i; j++)
  {
   if (a[j].arrivetime>a[j + 1].arrivetime) //将到达时间短的交换到前边
   {
    t = a[j];
    a[j] = a[j + 1];
    a[j + 1] = t;
    flag = 1;//交换
   }
  }
  for (j = 0; j<n - i; j++)
  {
   if ((a[j].servetime>a[j + 1].servetime)&&(a[j].arrivetime>=a[j + 1].arrivetime))//将服务时间短的交换到前边
   {
    t = a[j];
    a[j] = a[j + 1];
    a[j + 1] = t;
    flag = 1;//交换
   }
  }
  if (flag == 0)//如果一趟排序中没发生任何交换，则排序结束
  {
   break;
  }
 }
 return a; //返回排序后进程数组
}
//先来先服务算法
void FCFS(struct FCFS_SJF_r a[], int n, float &t1, float &t2)
{
 int i;
 a[0].finishtime = a[0].arrivetime + a[0].servetime; //完成时间=到达时间-服务时间
 a[0].roundtime = a[0].finishtime - a[0].arrivetime; //周转时间=完成时间-提交时间
 a[0].daiquantime = a[0].roundtime / a[0].servetime; //带权时间=周转时间/服务时间
 for (i = 1; i<n; i++)
 {
  if (a[i].arrivetime<a[i - 1].finishtime) //当前到达时间在上一个作业结束时间之前
  {
   a[i].finishtime = a[i - 1].finishtime + a[i].servetime; //完成时间=上一个完成时间+服务时间
   a[i].roundtime = a[i].finishtime - a[i].arrivetime;  //周转时间=完成时间-到达时间
   a[i].daiquantime = a[i].roundtime / a[i].servetime;  //带权时间=周转时间/服务时间
  }
  else //当前到达时间在上一个作业结束时间之后
  {
   a[i].finishtime = a[i].arrivetime + a[i].servetime;
   a[i].roundtime = a[i].finishtime - a[i].arrivetime;
   a[i].daiquantime = a[i].roundtime / a[i].servetime;
  }
 }

 for (int i = 0; i<n; i++)
 {
  printf("\n-----------------------------------------------------\n");
  cout <<setw(2) << "进程名：" << a[i].name<<" ";
  cout << setw(2) << "到达时间：" << a[i].arrivetime << " ";
  cout << setw(2) << "服务时间：" << a[i].servetime<<endl;
  cout << setw(2) << "完成时间：" << a[i].finishtime<<endl;
  cout << setw(2) << "周转时间：" << a[i].roundtime<<endl;
  cout << setw(2) << "带权周转时间" << a[i].daiquantime<<endl;
  t1 += a[i].roundtime;
  t2 += a[i].daiquantime;
 }
}
//短作业优先算法
void SJF(struct FCFS_SJF_r a[], int n, float &t1, float &t2)
{
 int i;
 struct FCFS_SJF_r t;
 a[0].finishtime = a[0].arrivetime + a[0].servetime; //完成时间=到达时间+服务时间
 a[0].roundtime = a[0].finishtime - a[0].arrivetime; //周转时间=完成时间-提交时间
 a[0].daiquantime = a[0].roundtime / a[0].servetime; //带权时间=周转时间/服务时间

 for (i = 1; i < n; i++)
 {

   for (int c = i; c < n - 1; c++)
   { for (int d=i+1;d<n;d++)
    if ((a[i - 1].finishtime >= a[c].arrivetime)&&(a[i - 1].finishtime >= a[d].arrivetime) && (a[c].servetime > a[d].servetime))
    {
     t = a[c];
     a[c] = a[d];
     a[d] = t;
    }
       }

  if (a[i].arrivetime<a[i - 1].finishtime) //当前到达时间在上一个作业结束时间之前
  {
   a[i].finishtime = a[i - 1].finishtime + a[i].servetime; //完成时间=上一个完成时间+服务时间
   a[i].roundtime = a[i].finishtime - a[i].arrivetime;  //周转时间=完成时间-到达时间
   a[i].daiquantime = a[i].roundtime / a[i].servetime;  //带权时间=周转时间/服务时间
  }
  else //当前到达时间在上一个作业结束时间之后
  {
   a[i].finishtime = a[i].arrivetime + a[i].servetime;
   a[i].roundtime = a[i].finishtime - a[i].arrivetime;
   a[i].daiquantime = a[i].roundtime / a[i].servetime;
  }

 }
 for (int i = 0; i<n; i++)
 {
  printf("\n-----------------------------------------------------\n");
  cout << setw(2) << "进程名：" << a[i].name << " ";
  cout << setw(2) << "到达时间：" << a[i].arrivetime << " ";
  cout << setw(2) << "服务时间：" << a[i].servetime << endl;
  cout << setw(2) << "完成时间：" << a[i].finishtime << endl;
  cout << setw(2) << "周转时间：" << a[i].roundtime << endl;
  cout << setw(2) << "带权周转时间" << a[i].daiquantime << endl;
  t1 += a[i].roundtime;
  t2 += a[i].daiquantime;
 }
}
//优先级算法
void r(struct FCFS_SJF_r a[], int n, float &t1, float &t2)
{
 int i;
 struct FCFS_SJF_r t;
 a[0].finishtime = a[0].arrivetime + a[0].servetime; //完成时间=到达时间+服务时间
 a[0].roundtime = a[0].finishtime - a[0].arrivetime; //周转时间=完成时间-提交时间
 a[0].daiquantime = a[0].roundtime / a[0].servetime; //带权时间=周转时间/服务时间
 for (i = 1; i < n; i++)
 {

  for (int c = i; c < n - 1; c++)
  {
   for (int d = i + 1; d<n; d++)
    if ((a[i - 1].finishtime >= a[c].arrivetime) && (a[i - 1].finishtime >= a[d].arrivetime) && (a[c].servetime > a[d].servetime))
    {
     t = a[c];
     a[c] = a[d];
     a[d] = t;
    }
  }

  if (a[i].arrivetime<a[i - 1].finishtime) //当前到达时间在上一个作业结束时间之前
  {
   a[i].finishtime = a[i - 1].finishtime + a[i].servetime; //完成时间=上一个完成时间+服务时间
   a[i].roundtime = a[i].finishtime - a[i].arrivetime;  //周转时间=完成时间-到达时间
   a[i].daiquantime = a[i].roundtime / a[i].servetime;  //带权时间=周转时间/服务时间
  }
  else //当前到达时间在上一个作业结束时间之后
  {
   a[i].finishtime = a[i].arrivetime + a[i].servetime;
   a[i].roundtime = a[i].finishtime - a[i].arrivetime;
   a[i].daiquantime = a[i].roundtime / a[i].servetime;
  }

 }
 for (int i = 0; i<n; i++)
 {
  printf("\n-----------------------------------------------------\n");
  cout << setw(2) << "进程名：" << a[i].name << " ";
  cout << setw(2) << "到达时间：" << a[i].arrivetime << " ";
  cout << setw(2) << "服务时间/优先级：" << a[i].servetime << endl;
  cout << setw(2) << "完成时间：" << a[i].finishtime << endl;
  cout << setw(2) << "周转时间：" << a[i].roundtime << endl;
  cout << setw(2) << "带权周转时间" << a[i].daiquantime << endl;
  t1 += a[i].roundtime;
  t2 += a[i].daiquantime;
 }
}
class RR1//轮转调度算法的结构体
{   public:
 char name;  //进程名
 float arrive; //进程到达时间
 float run;  //进程运行时间
 float rest;  //运行进程剩余时间
 float finish;   //完成时间
 float zhouzhuan = 0.0f; //周转时间，
    float daiquan = 0.0f;//带权周转时间
 string state; //进程状态
};
void RR(int &a)//时间片轮转算法
{
 RR1 b[100];
 queue<RR1*>buffer;
 int i, j, flag;
 RR1 t;
 float temp = 0.0f; //缓存最后一个正数rest
 float nowtime = 0.0f;
 float sum_zhouzhuan = 0.0f, sum_daiquan = 0.0f; //所有进程总周转时间，所有进程总带权周转时间
 float avr_zhouzhuan = 0.0f, avr_daiquan = 0.0f;
 for (i = 0; i < a; i++)
 {
  printf("%d 进程名: ", i + 1);
  scanf("%s", &b[i].name);
  printf("到达时间：");
  scanf("%f", &b[i].arrive);
  printf("服务时间：");
  scanf("%f",&b[i].run);
  b[i].rest = b[i].run;
  b[i].state = "wait for coming";
 }
 float slice = 0.0f;
 printf("请输入时间块大小: ");
 scanf("%f", &slice);
 flag = 0;
 for (i = 1; i < a; i++)//按到达时间排序
 {
  for (j = 0; j < a - i; j++)
  for (j = 0; j < a - 1; j++)
  {
   if (b[j].arrive > b[j + 1].arrive) //将到达时间短的交换到前边
   {
    t = b[j];
    b[j] = b[j + 1];
    b[j + 1] = t;
    flag ++;//交换
   }
  }
  if (flag == 0)//如果一趟排序中没发生任何交换，则排序结束
  {
   break;
  }

 }
 cout << "按到达时间排序共交换" << flag << "次"<<endl;
 buffer.push(&b[0]);
 b[0].state = "ready";
 int round = 1;
 nowtime = b[0].arrive;//第一次开始时间即为到达时间排名第一的进程的到达时间
 int n = 1;
 while (!buffer.empty())
 {
  for (int m = n; m< a; m++)//考虑排名第一的进程和后面同时到达的情况
  {
   if (nowtime >= b[m].arrive)
   {
    b[m].state = "ready";
    buffer.push(&b[m]);
    n++;
   }
  }
  temp = buffer.front()->rest;
  nowtime += slice;
  buffer.front()->rest = buffer.front()->rest - slice;
  buffer.front()->state = "running";
  cout << "\n-----------------------------------------------\n";
  cout << "round:" << round<<" "<<buffer.front()->name<<" runing" << endl;
  if (buffer.front()->rest <= 0)
  {
   buffer.front()->finish = nowtime - slice + temp;
   nowtime = buffer.front()->finish;
   buffer.front()->zhouzhuan = nowtime - buffer.front()->arrive;
   buffer.front()->daiquan = buffer.front()->zhouzhuan / buffer.front()->run;
   sum_zhouzhuan += buffer.front()->zhouzhuan;
   sum_daiquan += buffer.front()->daiquan;
   buffer.front()->rest = 0;
   buffer.front()->state = "run and end";
   cout << "note:process  " << buffer.front()->name << "  " << "end at  " << buffer.front()->finish << "  zhouzhuantime=" << buffer.front()->zhouzhuan <<"  daiquanzhouzhuan="
   cout << "\n-----------------------------------------------\n";
   buffer.pop();
   for (int m = n; m< a; m++)//考虑进程还未到的情况，以及先于完成前到达的情况，还有在完成时到达的情况
   {
    if (nowtime >= b[m].arrive)
    {
     b[m].state = "ready";
     buffer.push(&b[m]);
     n++;
    }
    if ((nowtime < b[m].arrive) && (buffer.empty()))
    {
     cout << "\n---------------bad status-------------\n";
     cout << "name\t";
     cout << setw(2) << "arrive\t";
     cout << setw(2) << "run\t";
     cout << setw(2) << "rest\t";
     cout << setw(2) << "state\t" << endl;
     for (int p = 0; p < a; p++)
     {
      cout << b[p].name << "\t";
      cout << b[p].arrive << "\t";
      cout << b[p].run << "\t";
      cout << b[p].rest << "\t";
      cout << b[p].state;
      cout << endl;
     }
     cout << "there is a bad status" << endl;
     cout << "there is(are)  " << b[m].arrive - nowtime << " free time(times)" << endl;
     cout << "\n---------------bad status-------------\n";
     cout << "now when " << b[m].name << " is coming." << endl << "time is " << b[m].arrive << endl;
     b[m].state = "ready";
     nowtime = b[m].arrive;
     buffer.push(&b[m]);
     n++;
    }
   }
  }
  else
  {
   for (int m = n; m< a; m++)//考虑进程在时间片结束前到达的情况
   {
    if (nowtime >= b[m].arrive)
    {
     b[m].state = "ready";
     buffer.push(&b[m]);
```

```
     buffer.push(&b[m]);
     n++;
    }
    /*if ((nowtime < b[m].arrive) && (buffer.empty()))//永远不会走到这里
    {
     cout << "\n---------------bad status--------------\n";
     cout << "name\t";
     cout << setw(2) << "arrive\t";
     cout << setw(2) << "run\t";
     cout << setw(2) << "rest\t";
     cout << setw(2) << "state\t" << endl;
     for (int p = 0; p < a; p++)
     {
      cout << b[p].name << "\t";
      cout << b[p].arrive << "\t";
      cout << b[p].run << "\t";
      cout << b[p].rest << "\t";
      cout << b[p].state;
      cout << endl;
     }
     cout << "there is a bad status" << endl;
     cout << "there is(are)  " << b[m].arrive - nowtime << " free time(times)" << endl;
     cout << "\n---------------bad status--------------\n";
     cout << "now when " << b[m].name << " is coming."<<endl<<"time is " << b[m].arrive << endl;
     b[m].state = "ready";
     nowtime = b[m].arrive;
     buffer.push(&b[m]);
     n++;
    }*/
   }
   buffer.front()->state = "ran and waiting for next time";
   buffer.push(buffer.front());
   buffer.pop();
  }
  cout << "name\t";
  cout << setw(2) << "arrive\t";
  cout << setw(2) << "run\t";
  cout << setw(2) << "rest\t";
  cout << setw(2) << "state\t" << endl;
  for (int p = 0; p < a; p++)
  {
   cout << b[p].name << "\t";
   cout << b[p].arrive << "\t";
   cout << b[p].run << "\t";
   cout << b[p].rest << "\t";
   cout << b[p].state;
   cout << endl;
  }
  round++;
 }
 printf("\n总周转时间： ");
 printf("%f", sum_zhouzhuan);
 printf("\n总带权周转时间： ");
 printf("%f\n", sum_daiquan);
 avr_zhouzhuan = sum_zhouzhuan / a;
 avr_daiquan = sum_daiquan / a;
 printf("\n平均周转时间： ");
 printf("%f", avr_zhouzhuan);
 printf("\n平均带权周转时间： ");
 printf("%f\n", avr_daiquan);
}

int main()
{
 float t1 = 0.0f; //总周转时间
 float t2 = 0.0f; //总带权周转时间
 float avr_t1 = 0.0f; //平均周转时间
 float avr_t2 = 0.0f; //平均带权周转时间
 int n, i;
 char select = ' '; //选择算法变量标识
 while (select != 'e' && select != 'E') //不为退出标识，保持循环
 {
  system("cls");
  printf("请选择算法：\na.先来先服务算法\nb.短作业优先算法\nc.时间片轮转算法\nd.优先级算法\ne.退出程序\n输入选择字符标号： ");
  scanf("%c", &select);
  if (select == 'a' || select == 'A') //先来先服务算法
  {
   printf("\n\n=================先来先服务算法FCFS================\n\n");
   printf("请输入进程数：");
   scanf("%d", &n);
   for (i = 0; i<n; i++)
   {
    printf("%d 进程名:", i + 1);
    scanf("%s", &a[i].name);
    printf("到达时间：");
    scanf("%f", &a[i].arrivetime);
    printf("服务时间：");
    scanf("%f", &a[i].servetime);
   }
   sortarrivetime(a, n);//按到达时间先进行冒泡排序
   FCFS(a, n, t1, t2); //先来先服务算法
   avr_t1 = t1 / n;
   avr_t2 = t2 / n;
   printf("\n");
   printf("平均周转时间为：%f \n", avr_t1);
   printf("平均带权周转时间为：%f \n", avr_t2);
   system("pause");
  }
  else if (select == 'b' || select == 'B') //短作业优先算法
  {
   printf("\n\n=================短作业优先算法SJF================\n\n");
   printf("请输入进程数：");
   scanf("%d", &n);
   for (i = 0; i<n; i++)
   {
    printf("%d 进程名:", i + 1);
    scanf("%s", &a[i].name);
    printf("到达时间: ");
    scanf("%f", &a[i].arrivetime);
    printf("服务时间：");
    scanf("%f", &a[i].servetime);
   }
   sortservetime(a, n); //冒泡排序
   SJF(a, n, t1, t2); //短作业优先算法
   avr_t1 = t1 / n;
   avr_t2 = t2 / n;
   printf("\n");
   printf("平均周转时间为：%f \n", avr_t1);
   printf("平均带权周转时间为：%f \n", avr_t2);
   system("pause");
  }

  else if (select == 'c' || select == 'C') //时间片轮转算法
  {
   printf("\n\n=================时间片轮转算法RR================\n\n");
   int a;
   printf("请输入进程数: ");
   scanf("%d", &a);
   RR(a);
   system("pause");
  }
  else if (select == 'd' || select == 'D') //优先级算法r
  {
   printf("\n\n=================优先级算法r================\n\n");
   printf("请输入进程数：");
   scanf("%d", &n);
   for (i = 0; i<n; i++)
   {
    printf("%d 进程名:", i + 1);
    scanf("%s", &a[i].name);
    printf("到达时间: ");
    scanf("%f", &a[i].arrivetime);
    printf("优先级(服务时间)：");
    scanf("%f", &a[i].servetime);
   }
   sortservetime(a, n); //冒泡排序
   r(a, n, t1, t2); //优先级算法r
   avr_t1 = t1 / n;
   avr_t2 = t2 / n;
   printf("\n");
   printf("平均周转时间为：%f \n", avr_t1);
   printf("平均带权周转时间为：%f \n", avr_t2);
   system("pause");
  }
  else if (select == 'e' || select == 'E')
  {
   exit(0);
  }
  else {
   cout<< "please inter right choose!" << endl;
   system("pause");
  }

 }
 system("pause");
}
```