

**《操作系统》**

**处理器调度算法模拟实现与比较**

|  |  |
| --- | --- |
|  |  |
| 学 号： |  |
| 姓 名： |  |
| 学 院： |  |
| 日 期： | 2022年 10 月 25 日 |

# 一、实验内容

分析和探索处理器实施进程调度的前提条件，理解并掌握处理器调度算法的设计原理和实现机制，随机发生和模拟进程创建及相关事件，编程实现基于特定处理器调度算法（三种以上，譬如先来先服务调度算法、短进程优先调度算法、高优先权优先调度算法、高响应比优先调度算法、时间片轮转调度算法、多级反馈队列调度算法、等等）的系统调度处理过程，并加以测试验证。

# 二、开发环境、运行环境、测试环境

开发环境：Visual Studio 2022

运行环境：ubuntu虚拟机

测试环境：ubuntu虚拟机

# 三、实验步骤

## 一、关键数据结构

PCB结构体，用于存储进程的信息

typedef struct PCB{

char name[20];

// 运行时间

int running\_time;

// 到达时间

int enter\_time;

// 优先级

int priority;

// 完成时间

int done\_time; //用于时间片轮转

int copyRunning\_time; //用于时间片轮转

// 进程开始运行的时间

int start\_time;

struct PCB\* next;

} PCB;

PCB队列，用于实现时间片轮转算法

typedef struct PCBQueue{

PCB\* firstProg;

PCB\* LastProg;

int size;

} PCBQueue;

先来先服务算法：

void FCFS(PCB pro[],int num)

短进程优先算法：

void SJF(PCB pro[],int num)

高优先级优先算法：

void HPF(PCB pro[],int num)

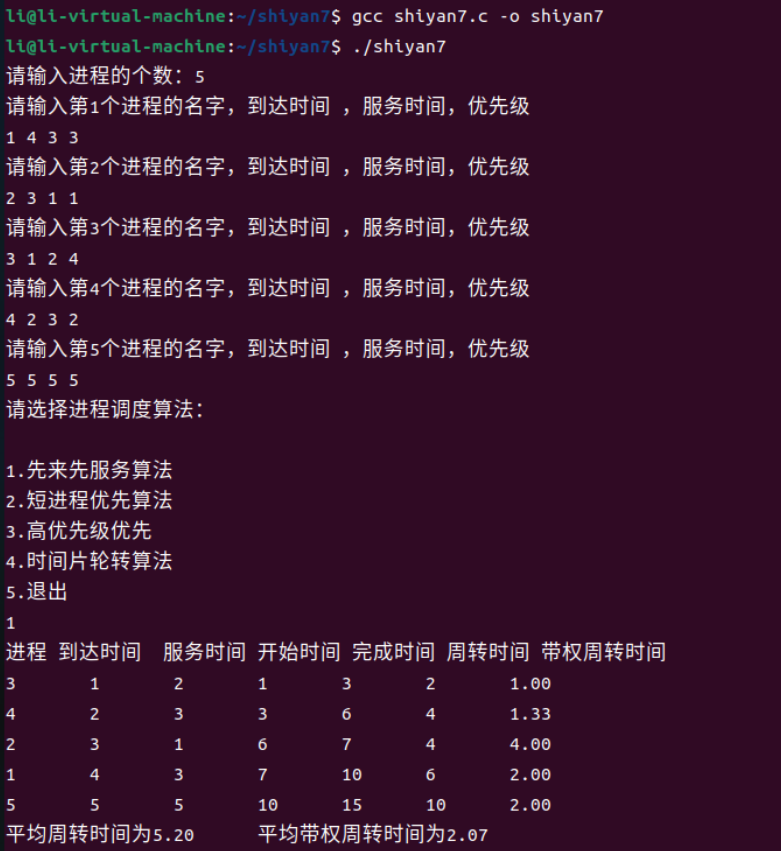
时间片轮转算法：

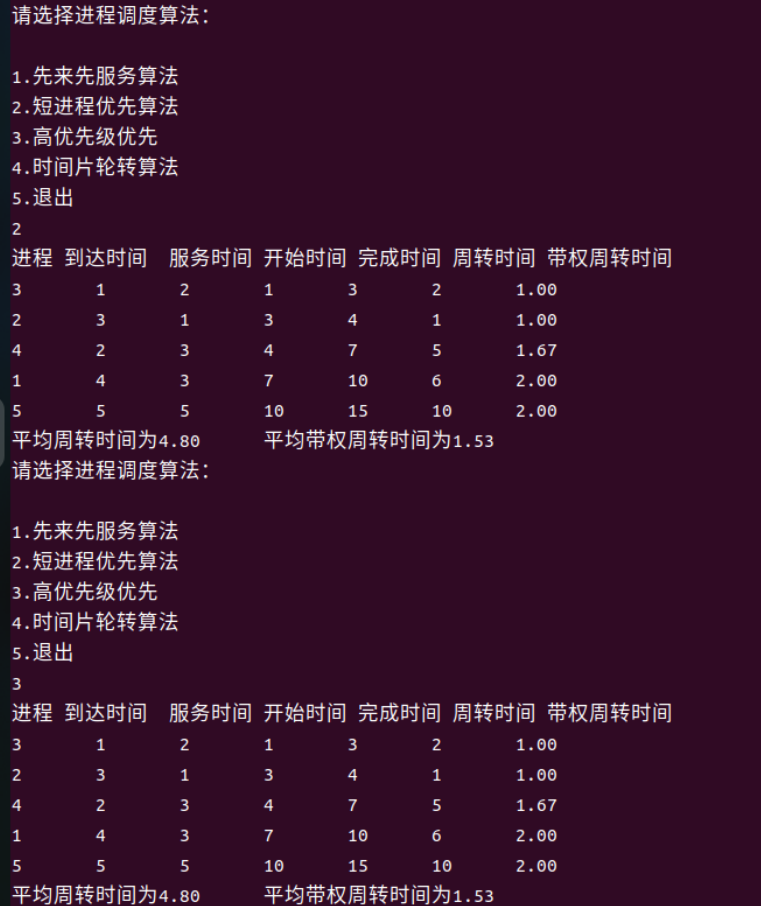
void RR(PCB pro[],int num)

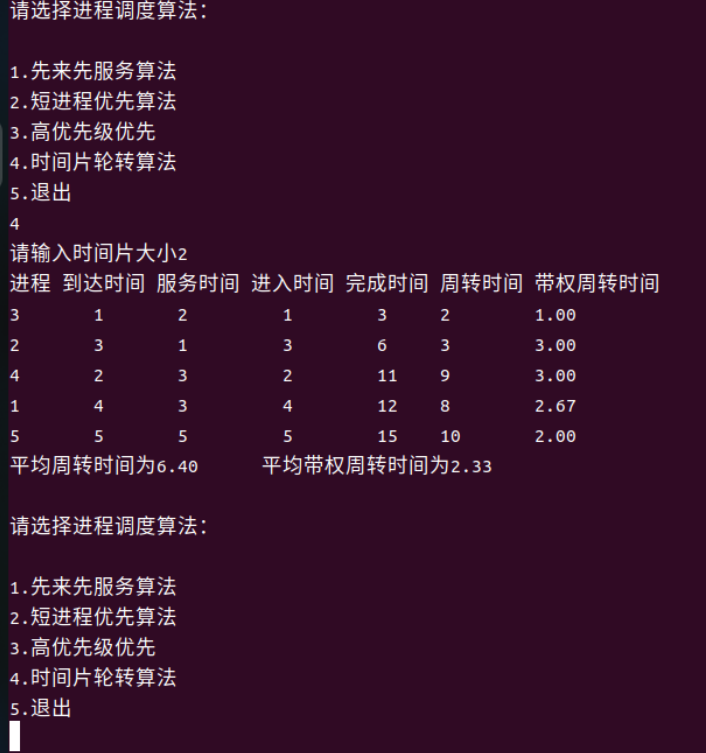
## 二、算法流程

* 先来先服务算法：先到达的进程先运行，后到达的进程进入等待队列。
* 短进程优先算法： 第一个到达的进程先运行，后到达的进程进入等待队列。第一个进程运行完毕之后，从等待队列选择运行时间最短的进程运行。
* 高优先级优先算法: 给每一个进程设置一个优先级。 第一个到达的进程先运行，后到达的进程进入等待队列。第一个进程运行完毕之后，从等待队列选择优先级高的进程运行。
* 时间片轮转算法：设置一段时间。每个进程运行完这段时间之后，进入等待队列队尾，下一个进程运行。如果只有一个进程，那么这个进程将一直运行，直到结束。

## 三、测试结果及分析







编译并运行shiyan7.c程序，对四种调度算法分别进行测试，实验结果与自我推演结果相同，可见调度算法的正确性得到了证明。

# 四、实验总结

在这次实验中，我对于四种处理器调度算法的实现思路和算法流程理解的更加深刻，并成功进行了代码实现。在实验过程中，由于对linux多线程编程不是很熟悉导致编写的代码多次通不过编译，运行时也有很多的BUG，通过自己DUBUG和参考网络上经典的代码最终成功完成了实验，并在这个过程中学会了很多新的知识和技能。

# 五、完整源代码

1. #include<stdio.h>
2. #include<malloc.h>
3. #include<string.h>
4. #include<stdlib.h>
6. **typedef** **struct** PCB{
7. **char** name[20];
8. //  运行时间
9. **int** running\_time;
10. //   到达时间
11. **int** enter\_time;
12. //    优先级
13. **int** priority;
14. //   完成时间
15. **int** done\_time;    //用于时间片轮转
16. **int** copyRunning\_time;  //用于时间片轮转
17. //  进程开始运行的时间
18. **int** start\_time;
20. **struct** PCB\* next;
21. } PCB;
23. **typedef** **struct** PCBQueue{
24. PCB\* firstProg;
25. PCB\* LastProg;
26. **int** size;
27. } PCBQueue;
29. **void** Queueinit(PCBQueue\* queue){
30. **if**(queue==NULL){
31. **return**;
32. }
33. queue->size = 0;
34. queue->LastProg = (PCB\*)malloc(**sizeof**(PCB));
35. queue->firstProg = queue->LastProg;
36. }


40. **void** EnterQueue(PCBQueue\* queue,PCB\* pro){   //加入进程队列
41. queue->LastProg->next = (PCB\*)malloc(**sizeof**(PCB));
42. queue->LastProg = queue->LastProg->next;
43. queue->LastProg->enter\_time = pro->enter\_time;
44. //    将name复制给 LastProg
45. memcpy(queue->LastProg->name,pro->name,**sizeof**(pro->name));
46. queue->LastProg->priority = pro->priority;
47. queue->LastProg->running\_time = pro->running\_time;
48. queue->LastProg->copyRunning\_time = pro->copyRunning\_time;
49. queue->LastProg->start\_time = pro->start\_time;
50. queue->size++;
51. }
52. PCB\* poll(PCBQueue\* queue){
53. PCB\* temp = queue->firstProg->next;
54. **if**(temp == queue->LastProg){
55. queue->LastProg=queue->firstProg;
56. queue->size--;
57. **return** temp;
58. }
59. queue->firstProg->next = queue->firstProg->next->next;
60. queue->size--;
61. **return** temp;
62. }
64. **void** inputPCB(PCB pro[],**int** num){
65. **for**(**int** i=0;i<num;i++){
66. PCB prog ;
67. printf("请输入第%d个进程的名字，到达时间 ，服务时间，优先级\n",i+1);
68. scanf("%s",prog.name);
69. scanf("%d",&prog.enter\_time) ;
70. scanf("%d",&prog.running\_time);
71. prog.copyRunning\_time = prog.running\_time;
72. scanf("%d",&prog.priority);
73. pro[i]=prog;
74. }
75. }
77. //冒泡排序算法（每次找到最大的放在末尾）
78. **void** sortWithEnterTime(PCB pro[],**int** num){
79. **for**(**int** i=1;i<num;i++){
80. **for**(**int** j= 0;j<num-i;j++){
81. **if**(pro[j].enter\_time>pro[j+1].enter\_time){
82. PCB temp = pro[j];
83. pro[j] = pro[j+1];
84. pro[j+1] = temp;
85. }
86. }
87. }
88. }
90. **void** FCFS(PCB pro[],**int** num){
91. printf("进程 到达时间  服务时间 开始时间 完成时间 周转时间 带权周转时间\n");
92. sortWithEnterTime(pro,num);    //按照进入顺序排序
93. PCBQueue\* queue = (PCBQueue\*)malloc(**sizeof**(PCBQueue));
94. Queueinit(queue);
95. EnterQueue(queue,&pro[0]);
96. **int** time = pro[0].enter\_time;
97. **int** pronum=1;    //记录当前的进程
98. //平均周转时间
99. **float** sum\_T\_time = 0;
100. //    带权平均周转时间
101. **float** sum\_QT\_time = 0 ;
103. **while**(queue->size>0){
104. PCB\* curpro = poll(queue);   //从进程队列中取出进程
105. **if**(time < curpro->enter\_time)
106. time =  curpro->enter\_time;
107. //完成时间
108. **int** done\_time = time+curpro->running\_time;
109. //周转时间（作业完成的时间-作业到达的时间）
110. **int** T\_time = done\_time - curpro->enter\_time;
111. sum\_T\_time += T\_time;
112. // 带权周转时间（（作业完成的时间-作业到达的时间）/ 作业运行时间）
113. **float** QT\_time = T\_time / (curpro->running\_time+0.0) ;
114. sum\_QT\_time += QT\_time;
115. **for**(**int** tt = time;tt<=done\_time && pronum<num;tt++){    //模拟进程的执行过程
116. **if**(tt >= pro[pronum].enter\_time){
117. EnterQueue(queue,&pro[pronum]);
118. pronum++;
119. }
120. }
121. printf("%s\t%d\t%d\t%d\t%d\t%d\t%.2f\n",curpro->name,curpro->enter\_time,curpro->running\_time,time,done\_time
122. ,T\_time,QT\_time);
123. time += curpro->running\_time;
124. **if**(queue->size==0 && pronum < num){   //防止出现前一个进程执行完到下一个进程到达之间无进程进入
125. EnterQueue(queue,&pro[pronum]);
126. pronum++;
127. }
128. }
129. printf("平均周转时间为%.2f\t平均带权周转时间为%.2f\n",sum\_T\_time/(num+0.0),sum\_QT\_time/(num+0.0));
130. }
132. //按照运行时间排序
133. **void** sortWithLongth(PCB pro[],**int** start,**int** end){
134. **int** len = end - start;
135. **if**(len == 1)
136. **return**;
137. **for**(**int** i=1;i<len;i++){
138. **for**(**int** j= start;j<end-i;j++){
139. **if**(pro[j].running\_time>pro[j+1].running\_time){
140. PCB temp = pro[j];
141. pro[j] = pro[j+1];
142. pro[j+1] = temp;
143. }
144. }
145. }
146. }
147. **void** SJF(PCB pro[],**int** num) {
148. printf("进程 到达时间  服务时间 开始时间 完成时间 周转时间 带权周转时间\n");
149. sortWithEnterTime(pro,num);
150. PCBQueue\* queue = (PCBQueue\*)malloc(**sizeof**(PCBQueue));;
151. Queueinit(queue);
152. EnterQueue(queue,&pro[0]);
153. **int** time = pro[0].enter\_time;
154. **int** pronum=1;    //记录当前的进程
155. **float** sum\_T\_time = 0,sum\_QT\_time = 0;
156. **while**(queue->size>0){
157. PCB\* curpro = poll(queue);   //从进程队列中取出进程
158. **if**(time <  curpro->enter\_time)
159. time =  curpro->enter\_time;
160. **int** done\_time = time+curpro->running\_time;
161. **int** T\_time = done\_time - curpro->enter\_time;
162. **float** QT\_time = T\_time / (curpro->running\_time+0.0) ;
163. sum\_T\_time += T\_time;
164. sum\_QT\_time += QT\_time;
165. **int** pre = pronum;
166. **for**(**int** tt = time;tt<=done\_time&&pronum<num;tt++){    //模拟进程的执行过程
167. **if**(tt>=pro[pronum].enter\_time){ // 统计从此任务开始到结束之间有几个进程到达
168. pronum++;
169. }
170. }
171. sortWithLongth(pro,pre,pronum);//将到达的进程按照服务时间排序
172. **for**(**int** i=pre;i<pronum;i++){    //将进程链入队列
173. EnterQueue(queue,&pro[i]);
174. }
175. pre = pronum;
176. printf("%s\t%d\t%d\t%d\t%d\t%d\t%.2f\n",curpro->name,curpro->enter\_time,curpro->running\_time,time,done\_time
177. ,T\_time,QT\_time);
178. time +=  curpro->running\_time;
179. **if**(queue->size==0&&pronum<num){   //防止出现前一个进程执行完到下一个进程到达之间无进程进入
180. EnterQueue(queue,&pro[pronum]);
181. pronum++;
182. }
183. }
184. printf("平均周转时间为%.2f\t平均带权周转时间为%.2f\n",sum\_T\_time/(num+0.0),sum\_QT\_time/num);
185. }
186. //按权重排序
187. **void** sortWithPriority(PCB pro[],**int** start,**int** end){
188. **int** len = end - start;
189. **if**(len == 1) **return** ;
190. **for**(**int** i=1;i<len;i++){
191. **for**(**int** j= start;j<end-i;j++){
192. **if**(pro[j].priority>pro[j+1].priority){
193. PCB temp = pro[j];
194. pro[j] = pro[j+1];
195. pro[j+1] = temp;
196. }
197. }
198. }
199. }
200. //优先级调度算法
201. **void** HPF(PCB pro[],**int** num){
202. printf("进程 到达时间  服务时间 开始时间 完成时间 周转时间 带权周转时间\n");
203. sortWithEnterTime(pro,num);
204. PCBQueue\* queue = (PCBQueue\*)malloc(**sizeof**(PCBQueue));;
205. Queueinit(queue);
206. EnterQueue(queue,&pro[0]);
207. **int** time = pro[0].enter\_time;
208. **int** pronum=1;    //记录当前的进程
209. **float** sum\_T\_time = 0,sum\_QT\_time = 0;
210. **while**(queue->size>0){
211. PCB\* curpro = poll(queue);   //从进程队列中取出进程
212. **if**(time<curpro->enter\_time)
213. time =  curpro->enter\_time;
214. **int** done\_time = time+curpro->running\_time;
215. **int** T\_time = done\_time - curpro->enter\_time;
216. **float** QT\_time = T\_time / (curpro->running\_time+0.0) ;
217. sum\_T\_time += T\_time;
218. sum\_QT\_time += QT\_time;
219. **int** pre = pronum;
220. **for**(**int** tt = time;tt<=done\_time&&pronum<num;tt++){    //模拟进程的执行过程
221. **if**(tt>=pro[pronum].enter\_time){ // 统计从此任务开始到结束之间有几个进程到达
222. pronum++;
223. }
224. }
225. sortWithPriority(pro,pre,pronum);//将到达的进程按照服务时间排序
226. **for**(**int** i=pre;i<pronum;i++){    //将进程链入队列
227. EnterQueue(queue,&pro[i]);
228. }
229. pre = pronum;
230. printf("%s\t%d\t%d\t%d\t%d\t%d\t%.2f\n",curpro->name,curpro->enter\_time,curpro->running\_time,time,done\_time
231. ,T\_time,QT\_time);
232. time +=  curpro->running\_time;
233. **if**(queue->size==0&&pronum<num){   //防止出现前一个进程执行完到下一个进程到达之间无进程进入
234. EnterQueue(queue,&pro[pronum]);
235. pronum++;
236. }
237. }
238. printf("平均周转时间为%.2f\t平均带权周转时间为%.2f\n",sum\_T\_time/(num+0.0),sum\_QT\_time/(num+0.0));
239. }
240. //时间片轮转调度
241. **void** RR(PCB pro[],**int** num){
242. printf("请输入时间片大小");
243. **int** timeslice;
244. scanf("%d",×lice);
245. printf("进程 到达时间 服务时间 进入时间 完成时间 周转时间 带权周转时间\n");
246. sortWithEnterTime(pro,num);
247. PCBQueue\* queue = (PCBQueue\*)malloc(**sizeof**(PCBQueue));;
248. Queueinit(queue);
249. //第一个进程开始运行的时间就是到达时间
250. pro[0].start\_time = pro[0].enter\_time;
251. EnterQueue(queue,&pro[0]);
252. **int** time = 0;
253. **int** pronum = 1;
254. **float** sum\_T\_time = 0,sum\_QT\_time = 0;
255. **while**(queue->size>0){
256. PCB\* curpro = poll(queue);    // 从队列中取出头节点
257. **if**(time<curpro->enter\_time)
258. time = curpro->enter\_time;
259. **if**(timeslice >= curpro->running\_time){   // 如果剩余时间小于时间片  则此任务完成
260. **for**(**int** tt = time;tt<=time+curpro->running\_time&&pronum<num;tt++){    // 模拟进程的执行过程
261. **if**(tt>=pro[pronum].enter\_time){ // 统计从此任务开始到结束之间有几个进程到达
262. pro[pronum].start\_time = tt;
263. EnterQueue(queue,&pro[pronum]);
264. pronum++;
265. }
266. }
267. time += curpro->running\_time;
268. curpro->running\_time = 0;
269. curpro->done\_time = time;
270. **int** T\_time = curpro->done\_time-curpro->start\_time;
271. **float** QT\_time = T\_time / (curpro->copyRunning\_time+0.0);
272. sum\_T\_time += T\_time;
273. sum\_QT\_time += QT\_time;
274. printf("%s\t%d\t%d\t  %d\t   %d\t %d\t  %.2f\n",curpro->name,curpro->enter\_time,curpro->copyRunning\_time,
275. curpro->start\_time,curpro->done\_time,T\_time,QT\_time);
276. **if**(queue->size==0&&pronum<num){   //防止出现前一个进程执行完到下一个进程到达之间无进程进入
277. pro[pronum].start\_time = pro[pronum].enter\_time;
278. EnterQueue(queue,&pro[pronum]);
279. pronum++;
280. }
281. **continue**;
282. }
283. // 运行时间大于时间片
284. **for**(**int** tt = time;tt<=time+timeslice&&pronum<num;tt++){    //模拟进程的执行过程
285. **if**(tt>=pro[pronum].enter\_time){ // 统计从此任务开始到结束之间有几个进程到达
286. pro[pronum].start\_time = tt;
287. EnterQueue(queue,&pro[pronum]);
288. pronum++;
289. }
290. }
291. time += timeslice;
292. curpro->running\_time -= timeslice;
293. //当前程序未完成  继续添加到队列中
294. EnterQueue(queue,curpro);
295. **if**(queue->size==0&&pronum<num){   //防止出现前一个进程执行完到下一个进程到达之间无进程进入
296. pro[pronum].start\_time = pro[pronum].enter\_time;
297. EnterQueue(queue,&pro[pronum]);
298. pronum++;
299. }
300. }
301. printf("平均周转时间为%.2f\t平均带权周转时间为%.2f\n\n",sum\_T\_time/(num+0.0),sum\_QT\_time/(num+0.0));
302. }
303. **void** choiceMenu(){
304. printf("请选择进程调度算法：\n\n");
305. printf("1.先来先服务算法\n2.短进程优先算法\n3.高优先级优先\n4.时间片轮转算法\n5.退出\n");
306. }
307. **void** menu(){
308. **int** proNum;
309. printf("请输入进程的个数：");
310. scanf("%d",&proNum);
311. PCB pro[proNum];
312. inputPCB(pro,proNum);
313. choiceMenu();
314. **int** choice;
315. **while**(1){
316. scanf("%d",&choice);
317. **switch**(choice){
318. **case** 1:FCFS(pro,proNum);choiceMenu();**break**;
319. **case** 2:SJF(pro,proNum);choiceMenu();**break**;
320. **case** 3:HPF(pro,proNum);choiceMenu();**break**;
321. **case** 4:RR(pro,proNum);choiceMenu();**break**;
322. **case** 5:**return**;
323. }
324. }
325. }
326. **int** main(){
327. menu();
328. **return** 0;
329. }