

# 河南工业大学 操作系统原理 实验报告

班级: 软件 1305 班    学号: 201316920311    姓名: 田劲锋    指导老师: 刘扬    日期: 2015 年 6 月 4 日

## 实验3 高(动态)优先权优先的进程调度算法模拟

### 1. 实验步骤

1. 以下是priority.c的源代码, 注释已详细给出:

Listing 1: parent\_child.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4  #include <errno.h>
5
6  typedef enum STATE { /* 状态的枚举类型 */
7      READY,
8      RUNNING,
9      BLOCK,
10     FINISH
11 } state;
12
13 const char* STATES[] = { /* 状态字符串 */
14     "READY",
15     "RUNNING",
16     "BLOCK",
17     "FINISH"
18 };
19
20 typedef struct PCB { /* 进程控制块 */
21     int id; /* 进程标识 */
22     int priority; /* 进程优先级 */
23     int cputime; /* 进程已占用时间 */
24     int needtime; /* 进程还需占用时间 */
25     int startblock; /* 进程开始阻塞的时刻 */
26     int blocktime; /* 进程需要阻塞的时长 */
27     state state; /* 进程状态 */
28     // struct PCB* next; /* 因为用了数组所以不再需要 */
29 } pcb;
30
31 typedef struct TASKLIST { /* 任务列表 */
32     pcb* at[1024]; /* 任务列表指针数组 */
33     size_t length; /* 任务列表长度 */
34     int finished; /* 已完成任务数 */
35 } tasklist;
36
37 /* 创建任务列表 */
38 tasklist* new_tasklist()
39 {
40     tasklist* tl = (tasklist*)malloc(sizeof(tasklist));
41     if (tl == NULL) {
42         perror("创建任务列表失败");
43         exit(-1);
44     }
45     tl->length = 0;
46     tl->finished = 0;
47     return tl;
48 }
49
50 /* 将进程控制块加入任务列表 */
51 int create_process(tasklist* tl, pcb* proc)
52 {
```

```

53     if (proc == NULL) {
54         return -1;
55     }
56     if (tl->length >= 1024) {
57         return -1;
58     }
59     tl->at[tl->length++] = proc;
60     return 0;
61 }
62
63 /* 打印一个进程控制块信息 */
64 void print_pcb(const pcb* proc)
65 {
66     printf("%2d    %8d %7d %8d %7s %10d %10d\n",
67         proc->id, proc->priority, proc->cputime, proc->needtime,
68         STATES[proc->state], proc->startblock, proc->blocktime);
69 }
70
71 /* 打印指定状态的任务队列 */
72 void print_queue(const tasklist* tl, state st)
73 {
74     static int id[1024];
75     int i, l = 0;
76     for (i = 0; i < tl->length; i++) {
77         pcb* p = tl->at[i];
78         if (p->state == st) {
79             id[l++] = p->id;
80         }
81     }
82     if (l > 0) {
83         printf("%d", id[0]);
84     }
85     for (i = 1; i < l; i++) {
86         printf("->%d", id[i]);
87     }
88 }
89
90 /* 打印任务列表 */
91 void print_tasklist(const tasklist* tl)
92 {
93     printf("    RUNNING PROCESS: ");
94     print_queue(tl, RUNNING);
95     printf("\n    READY QUEUE    : ");
96     print_queue(tl, READY);
97     printf("\n    BLOCK QUEUE    : ");
98     print_queue(tl, BLOCK);
99     printf("\n    FINISH QUEUE    : ");
100    print_queue(tl, FINISH);
101    printf("\n===== \n");
102    printf("ID    PRIORITY  CPUTIME  NEEDTIME  STATE  STARTBLOCK  BLOCKTIME\n");
103    int i;
104    for (i = 0; i < tl->length; i++) {
105        print_pcb(tl->at[i]);
106    }
107    printf("===== \n");
108 }
109
110 /* 读入任务列表 */
111 tasklist* read_table(const char* filename)
112 {
113     /* 打开文件 */
114     FILE* fin = fopen(filename, "r");
115     if (fin == NULL) {
116         fprintf(stderr, "打开文件 '%s' 失败: %s\n", filename, strerror(errno));
117         exit(-1);
118     }

```

```

119     int i, n;
120     /* 申请任务列表内存 */
121     tasklist* tl = new_tasklist();
122     /* 读入任务数 */
123     int x = fscanf(fin, "%d", &n);
124     if (x != 1) {
125         fprintf(stderr, "读入任务列表失败\n");
126         exit(-1);
127     }
128     for (i = 0; i < n; i++) {
129         /* 申请进程控制块内存 */
130         pcb* p = (pcb*)malloc(sizeof(pcb));
131         if (p == NULL) {
132             perror("创建进程控制块失败");
133             exit(-1);
134         }
135         /* 读入一个进程信息 */
136         x = fscanf(fin, "%d %d %d %d %d %d", &p->id, &p->priority, &p->cpu_time,
137             &p->need_time, &p->start_block, &p->block_time, &p->state);
138         if (x != 7) {
139             fprintf(stderr, "读入任务列表失败\n");
140             exit(-1);
141         }
142         /* 创建进程到任务列表中 */
143         if (create_process(tl, p) != 0) {
144             fprintf(stderr, "创建进程 '%d' 失败\n", i);
145             fclose(fin);
146             exit(-1);
147         }
148     }
149     fclose(fin);
150     /* 安全检查 */
151     if (tl->length != n) {
152         fprintf(stderr, "创建进程表失败\n");
153         exit(-1);
154     }
155     return tl;
156 }
157
158 /* 进程列表的比较函数 */
159 int cmp(const void* x, const void* y)
160 {
161     pcb* a = *(pcb**)x; /* 指向指针的指针 */
162     pcb* b = *(pcb**)y;
163
164     /* 只有一个运行中的任务，一定排在最前 */
165     if (a->state == RUNNING) {
166         return -1;
167     }
168     if (b->state == RUNNING) {
169         return 1;
170     }
171
172     /* 相同的状态比较优先级大小 */
173     if (a->state == b->state) {
174         return b->priority - a->priority;
175     }
176
177     /* 阻塞进程和已完成进程放在最后 */
178     if (a->state == BLOCK || a->state == FINISH) {
179         return 1;
180     }
181     if (b->state == BLOCK || b->state == FINISH) {
182         return -1;
183     }
184 }

```

```

185     /* 其他情况不排序 */
186     return 0;
187 }
188
189 /* 进程列表的排序函数 */
190 void sort_tasklist(tasklist* tl)
191 {
192     qsort(tl->at, tl->length, sizeof(tl->at[0]), cmp);
193 }
194
195 /* 运行任务列表 */
196 void run_tasklist(tasklist* tl)
197 {
198     int i, now = 0;
199     while (tl->finished < tl->length) {
200         /* 首先将优先级最高的就绪任务设为运行态 */
201         sort_tasklist(tl);
202         if (tl->at[0]->state != RUNNING) {
203             for (i = 0; i < tl->length; i++) {
204                 pcb* p = tl->at[i];
205                 if (p->state == READY) {
206                     p->state = RUNNING;
207                     break;
208                 }
209             }
210         }
211         /* 打印任务列表 */
212         printf("时间片 %d:\n", now++);
213         print_tasklist(tl);
214         /* 对每个任务 */
215         int finished = 0;
216         for (i = 0; i < tl->length; i++) {
217             pcb* p = tl->at[i];
218             if (p->state == READY) {
219                 /* 就绪 */
220                 p->priority++; /* 优先级加1 */
221             } else if (p->state == RUNNING) {
222                 /* 运行 */
223                 if (p->needtime > 0) {
224                     p->needtime--;
225                     p->cputime++; /* 运行了一个时间片 */
226                 }
227                 if (p->needtime == 0) {
228                     p->state = FINISH; /* 运行完 */
229                 }
230                 p->priority -= 3; /* 优先级减3 */
231                 if (p->startblock >= 0) {
232                     p->startblock--;
233                 }
234                 if (p->startblock == 0) {
235                     p->state = BLOCK; /* 进入阻塞状态 */
236                 }
237             } else if (p->state == BLOCK) {
238                 /* 阻塞 */
239                 if (p->blocktime > 0) {
240                     p->blocktime--;
241                 }
242                 if (p->blocktime == 0) {
243                     p->state = READY; /* 进入就绪状态 */
244                 }
245             } else if (p->state == FINISH) {
246                 /* 完成 */
247                 finished++; /* 记录已完成的任务数 */
248             }
249         }
250         tl->finished = finished;

```

```

251     }
252 }
253
254 int main(int argc, const char* argv[])
255 {
256     if (argc < 2) {
257         printf("用法: %s <初始进程表>\n", argv[0]);
258         return 0;
259     }
260
261     tasklist* t1 = read_table(argv[1]);
262     printf("初始进程表:\n");
263     print_tasklist(t1);
264     run_tasklist(t1);
265
266     return 0;
267 }

```

我们为该程序准备了一个输入文件:

```

5
0 9 0 3 2 3 0
1 38 0 3 -1 0 0
2 30 0 6 -1 0 0
3 29 0 3 -1 0 0
4 0 0 4 -1 0 0

```

编译并执行该程序:

```

$ cc -Wall priority.c -o priority
$ ./priority pros.in > 1

```

得到输出结果如下, 可以看到这个模拟程序按照既定的规则, 共执行了20个时间片。

初始进程表:

```

RUNNING PROCESS:
READY QUEUE    : 0->1->2->3->4
BLOCK QUEUE    :
FINISH QUEUE   :

```

```

=====
ID   PRIORITY  CPUTIME  NEEDTIME  STATE  STARTBLOCK  BLOCKTIME
0       9       0         3  READY         2         3
1      38       0         3  READY        -1         0
2      30       0         6  READY        -1         0
3      29       0         3  READY        -1         0
4       0       0         4  READY        -1         0
=====

```

时间片 0:

```

RUNNING PROCESS: 1
READY QUEUE      : 2->3->0->4
BLOCK QUEUE      :
FINISH QUEUE     :

```

```

=====
ID   PRIORITY  CPUTIME  NEEDTIME  STATE  STARTBLOCK  BLOCKTIME
1      38       0         3  RUNNING        -1         0
2      30       0         6  READY        -1         0
3      29       0         3  READY        -1         0
0       9       0         3  READY         2         3
4       0       0         4  READY        -1         0
=====

```

时间片 1:

RUNNING PROCESS: 1  
 READY QUEUE : 2->3->0->4  
 BLOCK QUEUE :  
 FINISH QUEUE :

ID	PRIORITY	CPUTIME	NEEDTIME	STATE	STARTBLOCK	BLOCKTIME
1	35	1	2	RUNNING	-1	0
2	31	0	6	READY	-1	0
3	30	0	3	READY	-1	0
0	10	0	3	READY	2	3
4	1	0	4	READY	-1	0

时间片 2:

RUNNING PROCESS: 1  
 READY QUEUE : 2->3->0->4  
 BLOCK QUEUE :  
 FINISH QUEUE :

ID	PRIORITY	CPUTIME	NEEDTIME	STATE	STARTBLOCK	BLOCKTIME
1	32	2	1	RUNNING	-1	0
2	32	0	6	READY	-1	0
3	31	0	3	READY	-1	0
0	11	0	3	READY	2	3
4	2	0	4	READY	-1	0

时间片 3:

RUNNING PROCESS: 2  
 READY QUEUE : 3->0->4  
 BLOCK QUEUE :  
 FINISH QUEUE : 1

ID	PRIORITY	CPUTIME	NEEDTIME	STATE	STARTBLOCK	BLOCKTIME
2	33	0	6	RUNNING	-1	0
3	32	0	3	READY	-1	0
0	12	0	3	READY	2	3
4	3	0	4	READY	-1	0
1	29	3	0	FINISH	-1	0

时间片 4:

RUNNING PROCESS: 2  
 READY QUEUE : 3->0->4  
 BLOCK QUEUE :  
 FINISH QUEUE : 1

ID	PRIORITY	CPUTIME	NEEDTIME	STATE	STARTBLOCK	BLOCKTIME
2	30	1	5	RUNNING	-1	0
3	33	0	3	READY	-1	0
0	13	0	3	READY	2	3
4	4	0	4	READY	-1	0
1	29	3	0	FINISH	-1	0

时间片 5:

RUNNING PROCESS: 2  
 READY QUEUE : 3->0->4  
 BLOCK QUEUE :  
 FINISH QUEUE : 1

ID	PRIORITY	CPUTIME	NEEDTIME	STATE	STARTBLOCK	BLOCKTIME
2	27	2	4	RUNNING	-1	0
3	34	0	3	READY	-1	0

0	14	0	3	READY	2	3
4	5	0	4	READY	-1	0
1	29	3	0	FINISH	-1	0

时间片 6:

RUNNING PROCESS: 2  
 READY QUEUE : 3->0->4  
 BLOCK QUEUE :  
 FINISH QUEUE : 1

ID	PRIORITY	CPUTIME	NEEDTIME	STATE	STARTBLOCK	BLOCKTIME
2	24	3	3	RUNNING	-1	0
3	35	0	3	READY	-1	0
0	15	0	3	READY	2	3
4	6	0	4	READY	-1	0
1	29	3	0	FINISH	-1	0

时间片 7:

RUNNING PROCESS: 2  
 READY QUEUE : 3->0->4  
 BLOCK QUEUE :  
 FINISH QUEUE : 1

ID	PRIORITY	CPUTIME	NEEDTIME	STATE	STARTBLOCK	BLOCKTIME
2	21	4	2	RUNNING	-1	0
3	36	0	3	READY	-1	0
0	16	0	3	READY	2	3
4	7	0	4	READY	-1	0
1	29	3	0	FINISH	-1	0

时间片 8:

RUNNING PROCESS: 2  
 READY QUEUE : 3->0->4  
 BLOCK QUEUE :  
 FINISH QUEUE : 1

ID	PRIORITY	CPUTIME	NEEDTIME	STATE	STARTBLOCK	BLOCKTIME
2	18	5	1	RUNNING	-1	0
3	37	0	3	READY	-1	0
0	17	0	3	READY	2	3
4	8	0	4	READY	-1	0
1	29	3	0	FINISH	-1	0

时间片 9:

RUNNING PROCESS: 3  
 READY QUEUE : 0->4  
 BLOCK QUEUE :  
 FINISH QUEUE : 1->2

ID	PRIORITY	CPUTIME	NEEDTIME	STATE	STARTBLOCK	BLOCKTIME
3	38	0	3	RUNNING	-1	0
0	18	0	3	READY	2	3
4	9	0	4	READY	-1	0
1	29	3	0	FINISH	-1	0
2	15	6	0	FINISH	-1	0

时间片 10:

RUNNING PROCESS: 3  
 READY QUEUE : 0->4  
 BLOCK QUEUE :

FINISH QUEUE : 1->2

ID	PRIORITY	CPUTIME	NEEDTIME	STATE	STARTBLOCK	BLOCKTIME
3	35	1	2	RUNNING	-1	0
0	19	0	3	READY	2	3
4	10	0	4	READY	-1	0
1	29	3	0	FINISH	-1	0
2	15	6	0	FINISH	-1	0

时间片 11:

RUNNING PROCESS: 3  
READY QUEUE : 0->4  
BLOCK QUEUE :  
FINISH QUEUE : 1->2

ID	PRIORITY	CPUTIME	NEEDTIME	STATE	STARTBLOCK	BLOCKTIME
3	32	2	1	RUNNING	-1	0
0	20	0	3	READY	2	3
4	11	0	4	READY	-1	0
1	29	3	0	FINISH	-1	0
2	15	6	0	FINISH	-1	0

时间片 12:

RUNNING PROCESS: 0  
READY QUEUE : 4  
BLOCK QUEUE :  
FINISH QUEUE : 3->1->2

ID	PRIORITY	CPUTIME	NEEDTIME	STATE	STARTBLOCK	BLOCKTIME
0	21	0	3	RUNNING	2	3
4	12	0	4	READY	-1	0
3	29	3	0	FINISH	-1	0
1	29	3	0	FINISH	-1	0
2	15	6	0	FINISH	-1	0

时间片 13:

RUNNING PROCESS: 0  
READY QUEUE : 4  
BLOCK QUEUE :  
FINISH QUEUE : 3->1->2

ID	PRIORITY	CPUTIME	NEEDTIME	STATE	STARTBLOCK	BLOCKTIME
0	18	1	2	RUNNING	1	3
4	13	0	4	READY	-1	0
3	29	3	0	FINISH	-1	0
1	29	3	0	FINISH	-1	0
2	15	6	0	FINISH	-1	0

时间片 14:

RUNNING PROCESS: 4  
READY QUEUE :  
BLOCK QUEUE : 0  
FINISH QUEUE : 3->1->2

ID	PRIORITY	CPUTIME	NEEDTIME	STATE	STARTBLOCK	BLOCKTIME
4	14	0	4	RUNNING	-1	0
3	29	3	0	FINISH	-1	0
1	29	3	0	FINISH	-1	0
2	15	6	0	FINISH	-1	0
0	15	2	1	BLOCK	0	3



=====  
时间片 15:

RUNNING PROCESS: 4  
READY QUEUE :  
BLOCK QUEUE : 0  
FINISH QUEUE : 3->1->2

=====  
ID    PRIORITY    CPUTIME    NEEDTIME    STATE    STARTBLOCK    BLOCKTIME  
4       11       1           3    RUNNING       -1           0  
0       15       2           1    BLOCK           0           2  
3       29       3           0    FINISH       -1           0  
1       29       3           0    FINISH       -1           0  
2       15       6           0    FINISH       -1           0  
=====

时间片 16:

RUNNING PROCESS: 4  
READY QUEUE :  
BLOCK QUEUE : 0  
FINISH QUEUE : 3->1->2

=====  
ID    PRIORITY    CPUTIME    NEEDTIME    STATE    STARTBLOCK    BLOCKTIME  
4       8       2           2    RUNNING       -1           0  
3       29       3           0    FINISH       -1           0  
1       29       3           0    FINISH       -1           0  
2       15       6           0    FINISH       -1           0  
0       15       2           1    BLOCK           0           1  
=====

时间片 17:

RUNNING PROCESS: 4  
READY QUEUE : 0  
BLOCK QUEUE :  
FINISH QUEUE : 3->1->2

=====  
ID    PRIORITY    CPUTIME    NEEDTIME    STATE    STARTBLOCK    BLOCKTIME  
4       5       3           1    RUNNING       -1           0  
0       15       2           1    READY           0           0  
3       29       3           0    FINISH       -1           0  
1       29       3           0    FINISH       -1           0  
2       15       6           0    FINISH       -1           0  
=====

时间片 18:

RUNNING PROCESS: 0  
READY QUEUE :  
BLOCK QUEUE :  
FINISH QUEUE : 3->1->2->4

=====  
ID    PRIORITY    CPUTIME    NEEDTIME    STATE    STARTBLOCK    BLOCKTIME  
0       16       2           1    RUNNING       0           0  
3       29       3           0    FINISH       -1           0  
1       29       3           0    FINISH       -1           0  
2       15       6           0    FINISH       -1           0  
4       2       4           0    FINISH       -1           0  
=====

时间片 19:

RUNNING PROCESS:  
READY QUEUE :  
BLOCK QUEUE :  
FINISH QUEUE : 3->1->2->0->4

=====  
ID    PRIORITY    CPUTIME    NEEDTIME    STATE    STARTBLOCK    BLOCKTIME

3	29	3	0	FINISH	-1	0
1	29	3	0	FINISH	-1	0
2	15	6	0	FINISH	-1	0
0	13	3	0	FINISH	-1	0
4	2	4	0	FINISH	-1	0

=====

2. 该算法即**高优先权优先调度算法**，每次执行一次排序，并执行优先级最高的可执行的任务，直到执行完毕或进入阻塞。这种方法要求给出进程的优先级，调度程序动态调整其优先级，按照其“重要程度”顺序执行任务。适用于实时系统。  
而**高响应比优先调度算法**的基本思想是把CPU分配给就绪队列中响应比（作业响应时间与作业执行时间的比值）最高的进程。这种方法兼顾了短作业与先后次序，且不会使长作业长期得不到服务。但是响应比计算用到了除法，增加了系统开销，所以更适合于批处理系统。