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 5 月 27 日

实验3　高(动态)优先权优先的进程调度算法模拟

1. 实验内容

(1) 用C语言来实现对N个进程采用动态优先权优先算法的进程调度。
(2) 每个用来标识进程的进程控制块PCB用结构来描述，包括以下字段：
- 进程标识数ID；
- 进程优先数PRIORITY，并规定优先数越大的进程，其优先权越高；
- 进程已占用的CPU时间CPUTIME；
- 进程还需占用的CPU时间NEEDTIME。当进程运行完毕时，NEEDTIME变为0；
- 进程的阻塞时间STARTBLOCK，表示当进程再运行STARTBLOCK个时间片后，进程将进入阻塞状态；
- 进程被阻塞的时间BLOCKTIME，表示已阻塞的进程再等待BLOCKTIME个时间片后，进程将转换成就绪状态；
- 进程状态STATE；(READY, RUNNING, BLOCK, FINISH)
- 队列指针NEXT，用来将PCB排成队列。

(3) 优先数改变的原则：
- 进程在就绪队列中呆一个时间片，优先数增加1；
- 进程每运行一个时间片，优先数减3。

(4) 假设在调度前，系统中有5个进程，它们的初始状态如下：

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

(5) 为了清楚地观察进程的调度过程，程序应将每个时间片内的进程的情况显示出来，参照的具体格式如下：

```
            RUNNING PROCESS: $id0
            READY QUEUE:  $id1->$id2
            BLOCK QUEUE:  $id3->$id4
    FINISH QUEUE:  $id0->$id1->$id2->$id3->$id4
=================================================================
ID    PRIORITY  CPUTIME   NEEDTIME   STATE   STARTBLOCK   BLOCKTIME
0       XX        XX        XX         XX       XX          XX
1       XX        XX        XX         XX       XX          XX
2       XX        XX        XX         XX       XX          XX
3       XX        XX        XX         XX       XX          XX
4       XX        XX        XX         XX       XX          XX
=================================================================
```

2. 实验要求

(1) 将源程序(`priority.c`)和程序运行结果写入实验报告。
(2) 将该算法执行过程与高响应比优先调度算法的执行过程进行比较。

3. 实验步骤

1. 以下是priority.c的源代码：

Listing 1: parent_child.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>

typedef enum STATE {
    READY,
    RUNNING,
    BLOCK,
    FINISH
} state_t;

const char* state_str[] = {
    "READY",
    "RUNNING",
    "BLOCK",
    "FINISH"
};

typedef struct PCB {
    int id; // 进程标识数ID;
    int priority; // 进程优先数PRIORITY,并规定优先数越大的进程,其优先权越高;
    int cputime; // 进程已占用的CPU时间CPUTIME;
    int needtime; // 进程还需占用的CPU时间NEEDTIME。当进程运行完毕时,NEEDTIME变为0;
    int startblock; // 进程的阻塞时间STARTBLOCK,表示当进程再运行STARTBLOCK个时间片后,进程将进入
    int blocktime; // 进程被阻塞的时间BLOCKTIME,表示已阻塞的进程再等待BLOCKTIME个时间片后,进程
    state_t state; // 进程状态STATE;(READY, RUNNING, BLOCK, FINISH)
    struct PCB* next; // 队列指针NEXT,用来将PCB排成队列。
} pcb;

typedef struct TASKLIST {
    pcb* head;
    pcb* tail;
    size_t length;
} tasklist;

tasklist* new_tasklist()
{
    tasklist* tl = (tasklist*)malloc(sizeof(tasklist));
    tl->head = NULL;
    tl->tail = NULL;
    tl->length = 0;
    return tl;
}

int create_process(tasklist* tl, pcb* proc)
{
    if (proc == NULL) {
        return -1;
    }
    if (tl->head == NULL) {
        proc->next = NULL;
        tl->head = proc;
        tl->tail = proc;
        tl->length = 1;
    }
    else {
        proc->next = NULL;
        tl->tail->next = proc;
        tl->tail = proc;
        tl->length++;
```

```
62      }
63      return 0;
64  }
65
66  pcb* process_at(tasklist* tl, int index)
67  {
68      pcb* p;
69      for (p = tl->head; p; p = p->next) {
70          if (p->id == index) {
71              return p;
72          }
73      }
74      return NULL;
75  }
76
77  void print_pcb(const pcb* proc)
78  {
79      printf("%2d    %8d  %7d   %8d %7s   %10d   %10d\n",
80          proc->id, proc->priority, proc->cputime, proc->needtime,
81          state_str[proc->state], proc->startblock, proc->blocktime);
82  }
83
84  void print_tasklist(const tasklist* tl)
85  {
86      printf("================================================================================\n");
87      printf("ID    PRIORITY  CPUTIME   NEEDTIME   STATE   STARTBLOCK   BLOCKTIME\n");
88      pcb* p;
89      for (p = tl->head; p; p = p->next) {
90          print_pcb(p);
91      }
92      printf("================================================================================\n");
93  }
94
95  tasklist* read_table(const char* filename)
96  {
97      FILE* fin = fopen(filename, "r");
98      if (fin == NULL) {
99          fprintf(stderr, "打开文件 '%s' 失败: %s\n", filename, strerror(errno));
100         exit(-1);
101     }
102     int i, j, n;
103     tasklist* tl = new_tasklist();
104     fscanf(fin, "%d", &n);
105     for (i = 0; i < n; i++) {
106         pcb* p = (pcb*)malloc(sizeof(pcb));
107         fscanf(fin, "%d %d %d %d %d %d %d", &p->id, &p->priority, &p->cputime,
108             &p->needtime, &p->startblock, &p->blocktime, &p->state);
109         // for (j = 0; j < 4; j++) {
110         //     if (strcmp(s, state_str[j]) == 0) {
111         //         p->state = j;
112         //         break;
113         //     }
114         // }
115         p->next = NULL;
116         if (create_process(tl, p) != 0) {
117             fprintf(stderr, "创建进程 '%d' 失败\n", i);
118             fclose(fin);
119             exit(-1);
120         }
121     }
122     fclose(fin);
123     if (tl->length != n) {
124         fprintf(stderr, "创建进程表失败\n");
125         exit(-1);
126     }
127     return tl;
```

```
128 }
129
130 void run_tasklist(tasklist* tl)
131 {
132     while (tl->length) {
133
134     }
135 }
136
137 int main(int argc, const char* argv[])
138 {
139     if (argc < 2) {
140         printf("用法: %s <初始进程表>\n", argv[0]);
141         return 0;
142     }
143
144     tasklist* tl = read_table(argv[1]);
145     run_tasklist(tl);
146
147     return 0;
148 }
```