

## 操作系统实验 2 进程状态

计算机 1202 张艺瀚  
学号:20123852

June 23, 2015

### 1 题目

模拟进程状态转换及其 PCB 的变化

### 2 目的

自行编制模拟程序，通过形象化的状态显示，使学生理解进程的概念、进程之间的状态转换及其所带来的 PCB 内容、组织的变化，理解进程与其 PCB 间的一一对应关系。

### 3 要求

1. 设计并实现一个模拟进程状态转换及其相应 PCB 内容、组织结构变化的程序。
2. 独立编写、调试程序。进程的数目、进程的状态模型（三状态、五状态、七状态或其它）以及 PCB 的组织形式可自行选择。
3. 合理设计与进程 PCB 相对应的数据结构。PCB 的内容要涵盖进程的基本信息、控制信息、资源需求及现场信息。
4. 设计出可视性较好的界面，应能反映出进程状态的变化引起的对应 PCB 内容、组织结构的变化。
5. 代码书写要规范，要适当地加入注释。
6. 鼓励在实验中加入新的观点或想法，并加以实现。
7. 认真进行预习，完成预习报告。
8. 实验完成后，要认真总结，完成实验报告。

## 4 程序流程图

见图 1-4

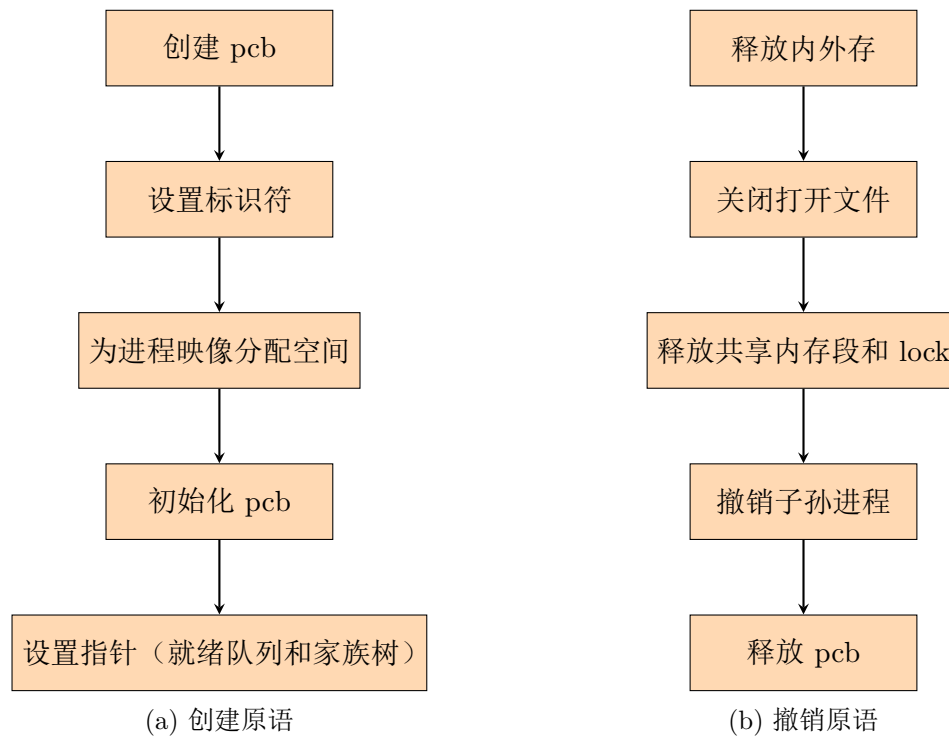


Figure 1: 创建原语和撤销原语

## 5 使用的数据结构及其说明

见表 2

## 6 程序源代码、文档注释及文字说明

`thread.cpp` 中实现了 6 种原语, `pcb.h` 中定义了 `pcb` 数据结构, `group_tree.h` 中实现了进程树。具体见代码清单 1-3:

```
1 #include <algorithm>
2 #include <cassert>
3 #include <cmath>
```

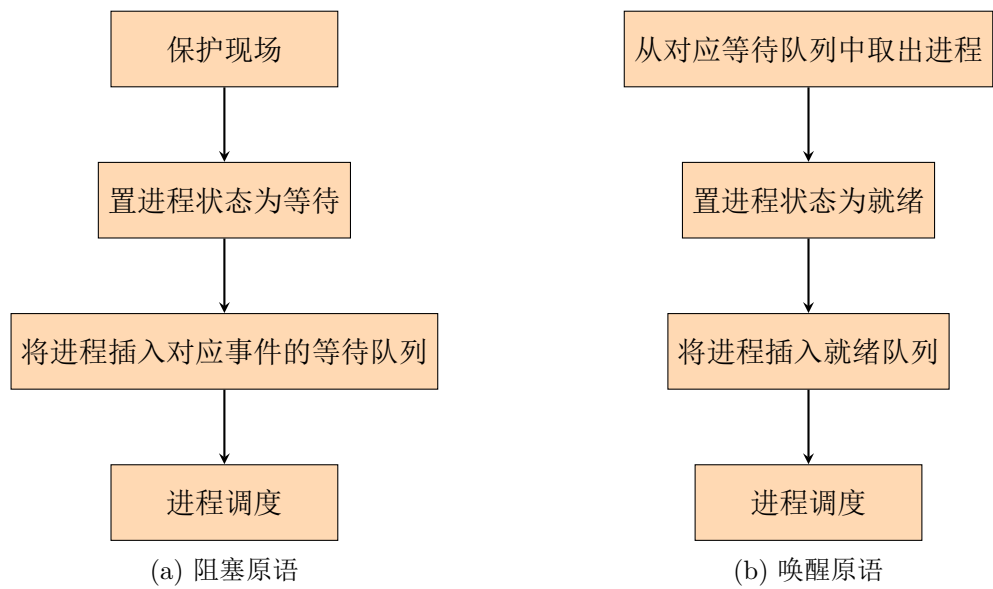


Figure 2: 阻塞原语和唤醒原语

名称	数据结构	说明
pcb 表	Pcb 型（类）数组	/
Pcb	类	属性：进程 id，进程名，用户 id，cpu 状态，进程状态，优先级，内存，资源
cpu 状态	枚举类型	/
进程状态	枚举类型	/
内存	结构类型	域：起始地址，内存大小，占用标识位
资源	结构	域：打开文件数组
等待队列	数组	/
阻塞队列	数组	/
进程树	树	类属性：根节点指针
进程树节点	类	属性：进程 id，双亲指针，子女指针数组

Table 1: 数据结构说明

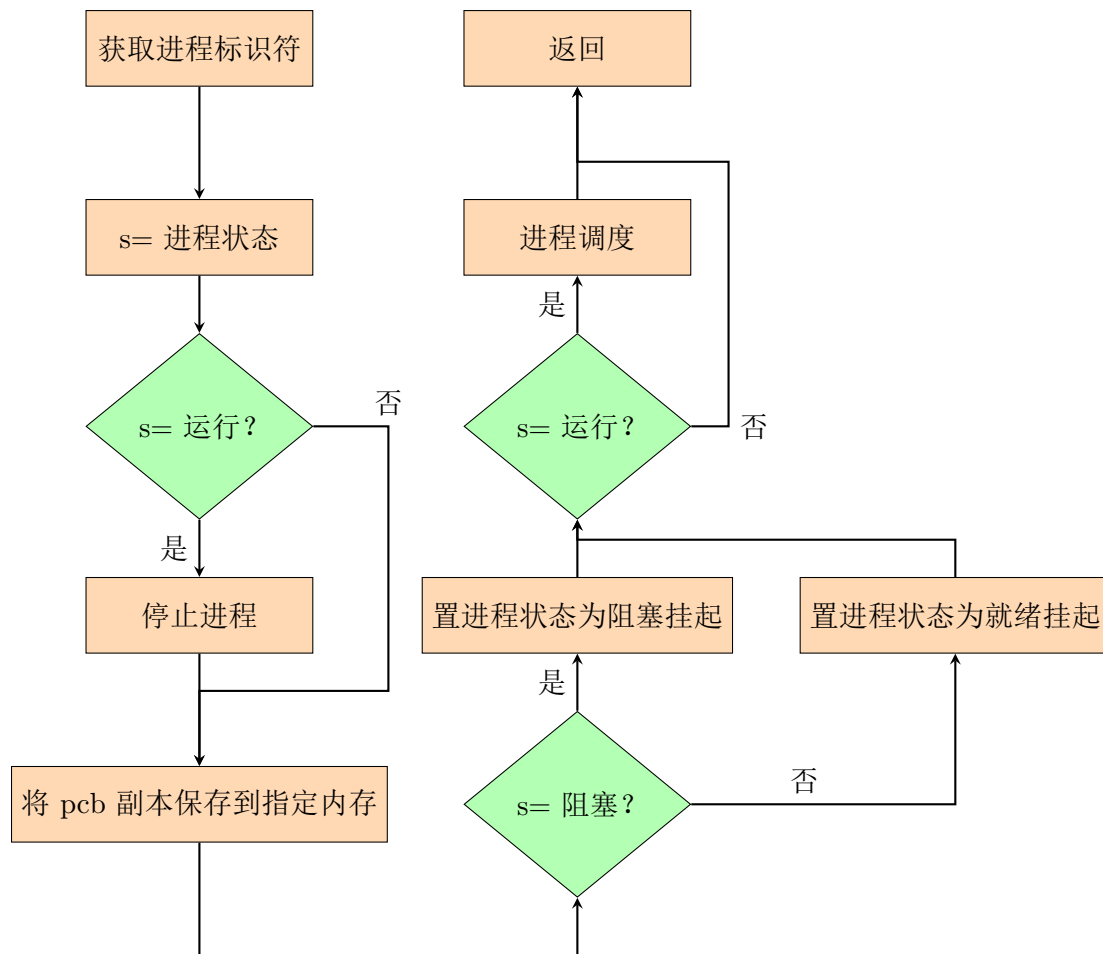


Figure 3: 挂起原语

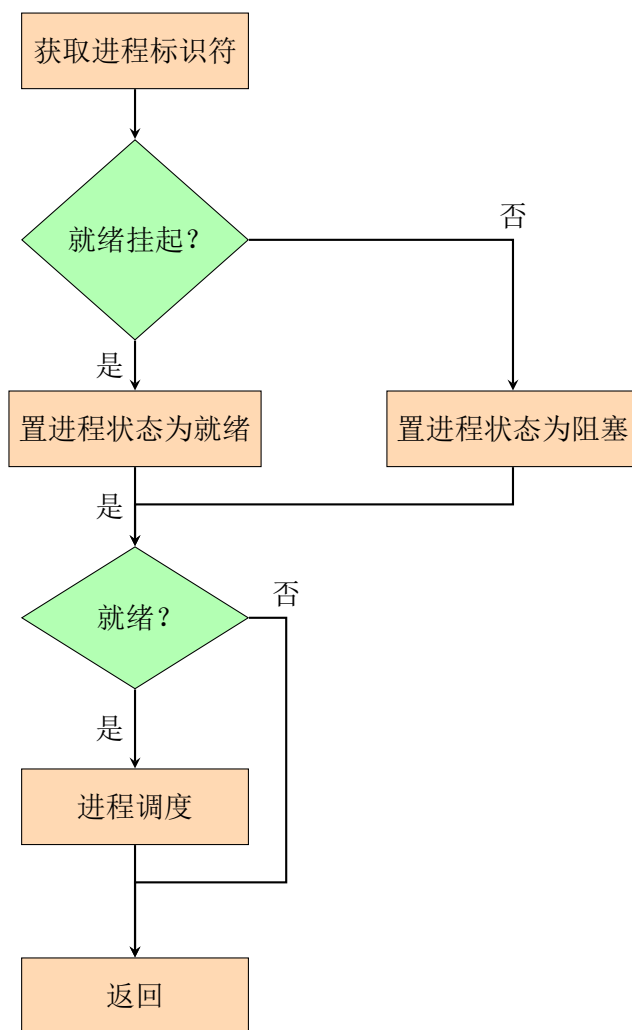


Figure 4: 激活原语

```
4 #include <initializer_list>
5 #include <iostream>
6 #include <fstream>
7 #include <functional>
8 #include <set>
9 #include <sstream>
10 #include <vector>
11
12 #include "pcb.h"
13 #include "group_tree.h"
14
15 using namespace std;
16
17 const int concurrency = 256;
18
19 vector<Pcb> pcb_table;
20 vector<int> ready_queue;
21 vector<int> blocked_queue;
22 int id = 0;
23 int exe_p;
24 bool schedule = false;
25 GroupTree group_tree;
26
27 bool get_pcb(string name, int& i){
28     // return false: exists; true: not exists
29     // i: index (-1, allocate memory failed)
30     auto p = find_if(pcb_table.begin(), pcb_table.end(),
31         [name](const Pcb& pcb){
32             return pcb.name == name;
33         });
34     if(p == pcb_table.end()){
35         if(pcb_table.size() + 1 > concurrency){
36             i = -1;
37             cerr << "Outstrip capacity of concurrency, apply for pcb
38 failed!\n";
39             return false;
40         }else{
41             pcb_table.push_back(Pcb());
42             i = pcb_table.size() - 1;
43             cerr << "Process named \" " << name << "\" not found, create
44 successfully!\n";
```

```
43     return false;
44 }
45 }else{
46     i = p - pcb_table.begin();
47     cerr << "Procdss named\" << name << "\" already exists.\n";
48     return true;
49 }
50 }
51
52 bool create(string name, CpuState cpu_state, int priority,
53             MainStore main_store, Resources resources){
54     int i;
55     if(!get_pcb(name, i) && i != -1){
56         pcb_table[i].id = ++id;
57         pcb_table[i].priority = priority;
58         pcb_table[i].cpu_state = cpu_state;
59         pcb_table[i].main_store = main_store;
60         pcb_table[i].resources = resources;
61         ready_queue.push_back(i);
62         // group_tree
63         cerr << "Apply pcb successfully.\n";
64         return true;
65     }else{
66         cerr << "Error: create process named \" << name << "\"
67             failed!\n";
68         return false;
69     }
70 }
71
72 bool remove(int i){
73     if(pcb_table[i].status == ready){
74         ready_queue.erase(find_if(ready_queue.begin(), ready_queue.
75             end(),
76             [i](const int& idx){
77                 return i == idx;
78             }));
79         cerr << "Remove pcb[" << i << "] from ready_queue
80             successfully.\n";
81         return true;
82     }else if(pcb_table[i].status == blocked){
83         blocked_queue.erase(find_if(blocked_queue.begin(),
```

```
80     blocked_queue.end(),
81     [i](const int& idx){
82         return i == idx;
83     }));
84     cerr << "Remove pcb[" << i << "] from blocked_queue
85     successfully.\n";
86     return true;
87 }else{
88     return false;
89 }
90 }
91
92 bool kill(int i){
93     if(i >= 0){
94         if(pcb_table[i].status == running){
95             // stop(i)
96             schedule = true;
97         }
98         remove(i);
99         // kill subtree in group_tree
100         // release main_store and resources
101         pcb_table.erase(pcb_table.begin() + i);
102         cerr << "Kill pcb[" << i << "] successfully.\n";
103         return true;
104     }else{
105         cerr << "Error: kill pcb[" << i << "] failed!\n";
106         return false;
107     }
108 }
109
110 bool destroy(string name){
111     schedule = false;
112     int i;
113     if(get_pcb(name, i)){
114         kill(i);
115         if(schedule){
116             // scheduler dispatches
117         }
118         cerr << "Destroy process named \"" << name << "\"
119         successfully.\n";
120         return true;
121     }
```



```
118 }else{
119     cerr << "Error: destroy process named \"" << name << "\"
120     failed!\n";
121     return false;
122 }
123
124 bool block(){
125     int i = exe_p;
126     if(exe_p >= 0)
127     {
128         // stop(i)
129         pcb_table[i].status = blocked;
130         blocked_queue.push_back(i);
131         // scheduler dispatches
132         cerr << "Block successfully.\n";
133         return true;
134     }else{
135         cerr << "Error: block failed!\n";
136         return false;
137     }
138 }
139
140 bool wake_up(string name){
141     int i;
142     if(get_pcb(name, i)){
143         remove(i);
144         pcb_table[i].status = ready;
145         ready_queue.push_back(i);
146         // scheduler dispatches
147         cerr << "Wake up process named \"" << name << "\"
148         successfully.\n";
149         return true;
150     }else{
151         cerr << "Error: wake up process named \"" << name << "\"
152         failed!\n";
153         return false;
154     }
155 }
156
157 bool suspend(string name){
```

```
156 int i;
157 if(get_pcb(name, i)){
158     Status s = pcb_table[i].status;
159     if(s == running){
160         // stop(i)
161     }
162     // main_store <- pcb_table[i]
163     if(s == blocked){
164         pcb_table[i].status == blocked_suspend;
165     }else{// ready / running
166         pcb_table[i].status = ready_suspend;
167     }
168     if(s == running){
169         // scheduler dispatches
170     }
171     cerr << "Suspend process named \"" << name << "\"
172     successfully.\n";
173     return true;
174 }else{
175     cerr << "Error: suspend process named \"" << name << "\"
176     failed!\n";
177     return false;
178 }
179 }
180
181 bool activate(string name){
182     int i;
183     if(get_pcb(name, i)){
184         pcb_table[i].status = pcb_table[i].status == ready_suspend?
185         ready: blocked;
186         if(pcb_table[i].status == ready){
187             // scheduler dispatches
188         }
189         cerr << "Activate process named \"" << name << "\"
190         successfully.\n";
191         return true;
192     }else{
193         cerr << "Error: activate process named \"" << name << "\"
194         failed!\n";
195         return false;
196     }
197 }
```

```
192 }
193
194 int main(int argc, char** argv){
195     group_tree.create(-1, 0);
196     group_tree.create(-1, 1);
197     group_tree.create(-1, 2);
198     group_tree.create(0, 4);
199     group_tree.create(4, 5);
200     group_tree.create(4, 6);
201     group_tree.create(6, 7);
202     group_tree.create(1, 3);
203     group_tree.preorder_travers_display();
204     group_tree.destroy(4);
205     group_tree.preorder_travers_display();
206     return 0;
207 }
```

Listing 1: thread.cpp 代码清单

```
208 #ifndef PCB_H
209 #define PCB_H
210
211 #include <algorithm>
212 #include <cassert>
213 #include <cmath>
214 #include <initializer_list>
215 #include <iostream>
216 #include <iterator>
217 #include <fstream>
218 #include <functional>
219 #include <regex>
220 #include <set>
221 #include <sstream>
222 #include <vector>
223
224 using namespace std;
225
226 enum CpuState{usr_state, kernel_state};
227 enum Status{creat, ready, ready_suspend, blocked, blocked_suspend,
             , running, ex};
228 struct MainStore{
```

```
229     int start_addr;
230     int mem_size;
231     bool usable;
232 };
233 struct Resources{
234     vector<string> opened_files;
235 };
236
237 class Pcb{
238 public:
239     Pcb(){}
240
241     // description
242     int id;
243     string name;
244     string usr_id;
245     // group_tree
246
247     // ctrl_infl
248     CpuState cpu_state;
249     Status status;
250     int priority;
251     // int entry_addr;
252     // int disk_addr;
253     // StatInfo stat_info;
254
255     // resources
256     // main_store
257     MainStore main_store;
258     Resources resources;
259
260     // cpu_scene
261
262 private:
263 };
264
265 #endif // PCB_H
```

Listing 2: pcb.h 代码清单

```
266 #ifndef GROUP_TREE_H
267 #define GROUP_TREE_H
268
269 #include <algorithm>
270 #include <cassert>
271 #include <cmath>
272 #include <initializer_list>
273 #include <iostream>
274 #include <iterator>
275 #include <fstream>
276 #include <functional>
277 #include <regex>
278 #include <set>
279 #include <sstream>
280 #include <vector>
281
282 #include "pcb.h"
283
284 using namespace std;
285
286 struct Node{
287     Node(){}
288     Node(int n, Node* pp, vector<Node*> cpl):
289         pcb_id(n), parent_ptr(pp), child_ptr_list(cpl){}
290
291     int pcb_id;
292     Node* parent_ptr;
293     vector<Node*> child_ptr_list;
294 };
295
296 class GroupTree{
297 public:
298     GroupTree(){
299         root = new Node();
300         root->pcb_id = -1;
301     }
302
303     void destruct(Node* p){
304         if(p != nullptr){
305             for(size_t i = 0; i < p->child_ptr_list.size(); ++i){
306                 destruct(p->child_ptr_list[i]);
```

```
307     }
308     delete p;
309 }
310 }
311
312 ~GroupTree(){
313     destruct(root);
314     // cout << "destruct\n";
315 }
316
317 void find(int i, Node* p, bool& found, Node*& pp){
318     if(p != nullptr){
319         if(p->pcb_id == i){
320             found = true;
321             pp = p;
322             return;
323         }else{
324             for(size_t j = 0; j < p->child_ptr_list.size(); ++j){
325                 find(i, p->child_ptr_list[j], found, pp);
326             }
327         }
328     }
329 }
330
331 bool create(int i, int pcb_id){
332     Node* pp;
333     bool found = false;
334     find(i, root, found, pp);
335     // if(pp != nullptr){
336     //     cout << "find " << i << " " << pp->pcb_id << endl;
337     // }else{
338     //     cout << "find null\n";
339     // }
340     if(found){
341         Node* np = new Node();
342         np->pcb_id = pcb_id;
343         np->parent_ptr = pp;
344         pp->child_ptr_list.push_back(np);
345     }
346 }
347
```

```
348 bool destroy(int i){
349     Node* p;
350     bool found = false;
351     find(i, root, found, p);
352     if(found){
353         p->parent_ptr->child_ptr_list.erase(
354             find_if(
355                 p->parent_ptr->child_ptr_list.begin(), p->parent_ptr->
child_ptr_list.end(),
356                 [i](Node* cp){
357                     return cp->pcb_id == i;
358                 }));
359         destruct(p);
360     }
361 }
362
363 void predispc(vector<int>& stk, Node* p, bool b1){
364     if(p != nullptr){
365         for(int i = 0; i < stk.size(); ++i){
366             cout << (stk[i] == 0? " ": "|") << "\t";
367         }
368         cout << "\\____ ";
369         // display node
370         cout << "id: " << p->pcb_id;
371         cout << "\n";
372         for(size_t i = 0; i < p->child_ptr_list.size(); ++i){
373             if(b1){
374                 stk.push_back(0);
375             }else{
376                 stk.push_back(1);
377             }
378             predispc(stk, p->child_ptr_list[i], i == p->child_ptr_list
.size() - 1);
379             stk.erase(stk.end() - 1);
380         }
381     }
382 }
383
384 void preorder_travers_display(){
385     vector<int> stk;
386     bool b = true; // show whether my parent is the last child of
```

```

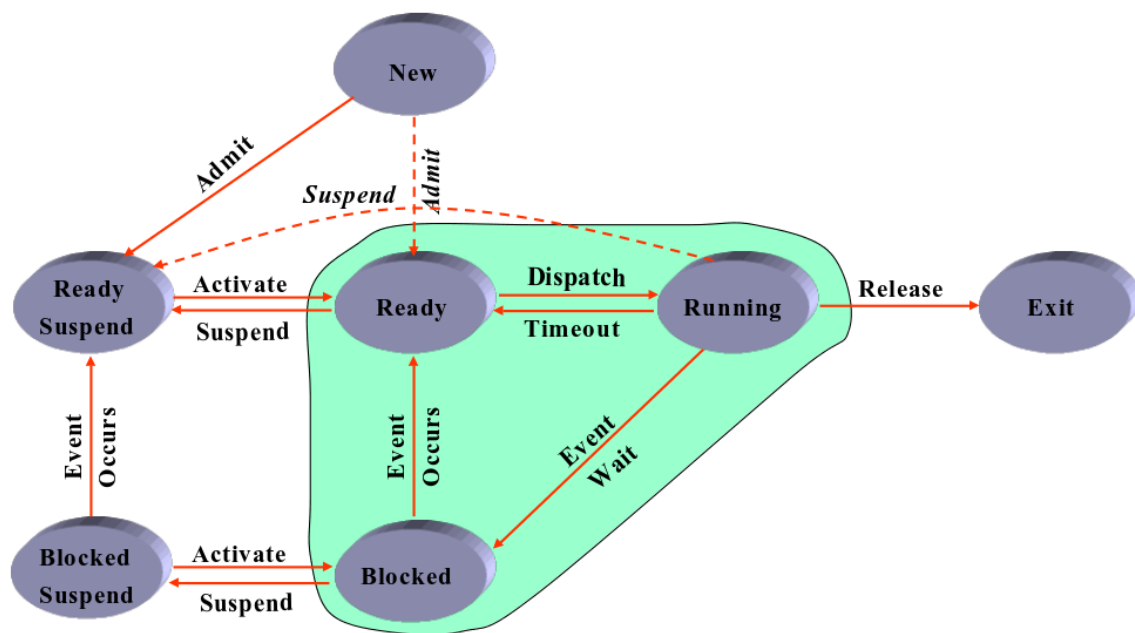
387     my grand parent
388     predisb(stk, root, b);
389 }
390 Node* root;
391 private:
392 };
393
394 #endif // GROUP_TREE_H

```

Listing 3: group\_tree.h 代码清单

## 7 运行结果及其说明

通过 6 种原语实现进程在如图 5 的 7 种状态间的转换和相应 pcb 的变化。





运行结果	说明
图 6	打印程序使用说明（所有命令格式）
图 7	执行 <code>admit 0</code> ，收容 0 进程（并就绪）
图 8	执行 <code>admit 1</code> ，收容 1 进程（并就绪）
图 9	执行 <code>dispatch 0</code> ，调度 0 进程执行（从就绪队列切上 CPU）
图 10	执行 <code>fork 2</code> ，为 0 进程创建子进程 2（子进程就绪）
图 11	执行 <code>eventwaits 0</code> ，0 进程等待某事件发生，被阻塞（从 CPU 切下进入阻塞队列）
图 12	执行 <code>eventoccurs 0</code> ，0 进程等待的事件发生，被唤醒（由阻塞队列进入就绪队列）
图 13	执行 <code>suspend 0</code> ，挂起 0 进程
图 14	执行 <code>activate 0</code> ，激活 0 进程
图 15	执行 <code>timeout 0</code> ，0 进程超时（进入就绪队列）
图 16	执行 <code>dispatch 0</code> ，再次调度 0 进程执行（从就绪队列切上 CPU）
图 17	执行 <code>rebase 0</code> ，杀死 0 进程（及其所有子孙进程）
图 18	执行 <code>quit 1</code> ，退出

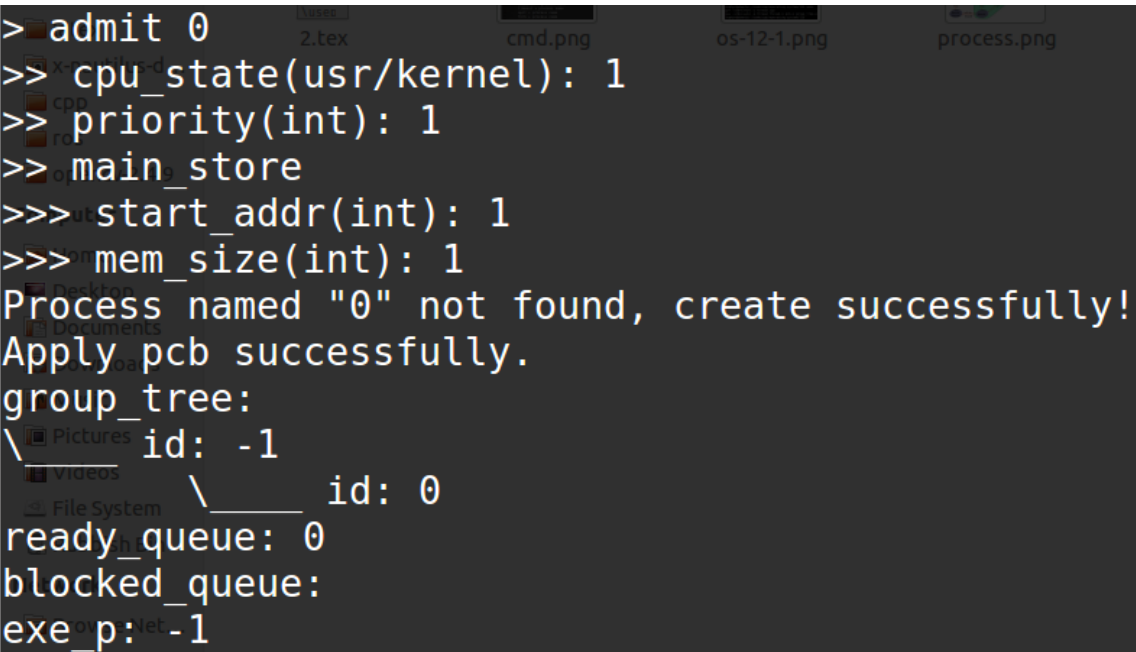
Table 2: 数据结构说明

```

zephyr@ubuntu: ~/code/cpp/operating-system/12
% ./thread
command list
admit + process name
rebase + process name
activate + process name
suspend + process name
eventoccurs + process name
eventwaits + process name
dispatch + process name
timeout + process name
quit + process name
fork + process name

```

Figure 6: cmd



```
> admit 0
>> cpu_state(usr/kernel): 1
>> priority(int): 1
>> main_store
>>> start_addr(int): 1
>>> mem_size(int): 1
Process named "0" not found, create successfully!
Apply pcb successfully.
group_tree:
\_____ id: -1
\_____ id: 0
ready_queue: 0
blocked_queue:
exe_p: Net-1
```

Figure 7: admit 0

## 8 程序使用说明

见表 3

```
> admit 1
>> cpu_state(usr/kernel): 1
>> priority(int): 1
>> main_store
>>> start_addr(int): 1
>>> mem_size(int): 1
Process named "1" not found, create successfully!
Apply pcb successfully.
group tree:
\ id: -1
  \ id: 0
  \ id: 1
ready_queue: 0 1
blocked_queue:
exe_p: -1
```

Figure 8: admit 1

admit（收容）	cpu_state（CPU 状态）	
	priority（优先级）	
	main_store（主存）	start_addr（起始地址） mem_size（占用内存大小）
dispatch（调度执行）		
suspend（挂起）		
activate（激活）		
eventwaits（等待事件发生，阻塞）		
eventoccurs（等待的事件发生，唤醒）		
timeout（超时，就绪）		
rebase（杀死进程及其所有子孙进程）		
fork（创建子进程）		
quit（退出）		

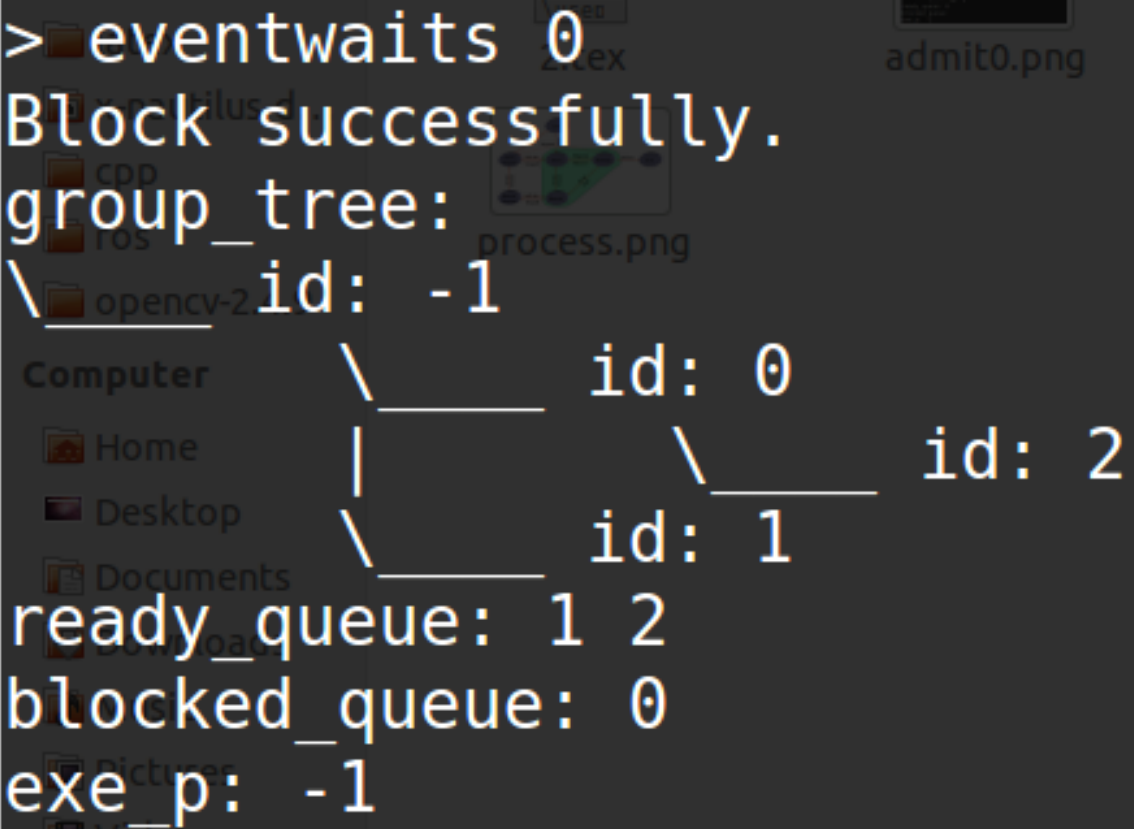
Table 3: 程序使用说明

```
> dispatch 0
Procdss named "0" already exists.
Procdss named "0" already exists.
Running process "0" successfully.
group_tree:
\computer id: -1
  Home \_____ id: 0
  Desktop \_____ id: 1
  Documents
ready_queue: 1
blocked_queue:
exe_p: 0
```

Figure 9: dispatch 0

```
> fork 2
Process named "2" not found, create successfully!
Procdss named "0" already exists.
Fork process "0" successfully.
group_tree:
\ Home id: -1
  Desktop \_____ id: 0
  Documents | \_____ id: 2
  Downloads | \_____ id: 1
  Music
ready_queue: 1 2
blocked_queue:
exe_p: 0
```

Figure 10: fork 2

A terminal window with a dark background and yellow text. The text shows the execution of 'eventwait(0)' and the resulting process tree. The process tree is a vertical chain: 'Computer' (id: 0) is the parent of 'Home' (id: 1), which is the parent of 'Desktop' (id: 2). Below this, the 'ready\_queue' is shown with processes 1 and 2, and the 'blocked\_queue' is empty. The 'exe\_p' is -1.

```
> eventwait 0
Block successfully.
group_tree:
\opencv-2 id: -1
Computer \_____ id: 0
      |      \_____ id: 2
      |      |
      |      \_____ id: 1
      |
      |
ready_queue: 1 2
blocked_queue: 0
exe_p: -1
```

Figure 11: eventwait(0)

```
> eventoccurs 0
Procdss named "0" already exists.
Procdss named "0" already exists.
Remove pcb[0] from blocked_queue successfully.
Wake up process named "0" successfully.
group_tree:
\Music id: -1
  Pictures \_____ id: 0
  Videos | \_____ id: 2
  File System \_____ id: 1
  Rubbish Bin \_____ id: 1
ready_queue: 1 2 0
blocked_queue:
exe_p: -1
```

Figure 12: eventoccurs0

```
> suspend 0
Procdss named "0" already exists.
Suspend process named "0" successfully.
group_tree:
\Computer id: -1
  Home \_____ id: 0
  Desktop | \_____ id: 2
  Documents \_____ id: 1
  Downloads \_____ id: 1
ready_queue: 1 2 0
blocked_queue:
exe_p: -1
```

Figure 13: suspend 0

```
> activate 0
Procdss named "0" already exists.
Activate process named "0" successfully.
group_tree:
\Home id: -1
  Desktop \ id: 0
  Documents | \ id: 2
  Downloads | \ id: 1
  Music \ id: 1
ready_queue: 1 2 0
blocked_queue:
exe_p: -1
```

Figure 14: activate 0

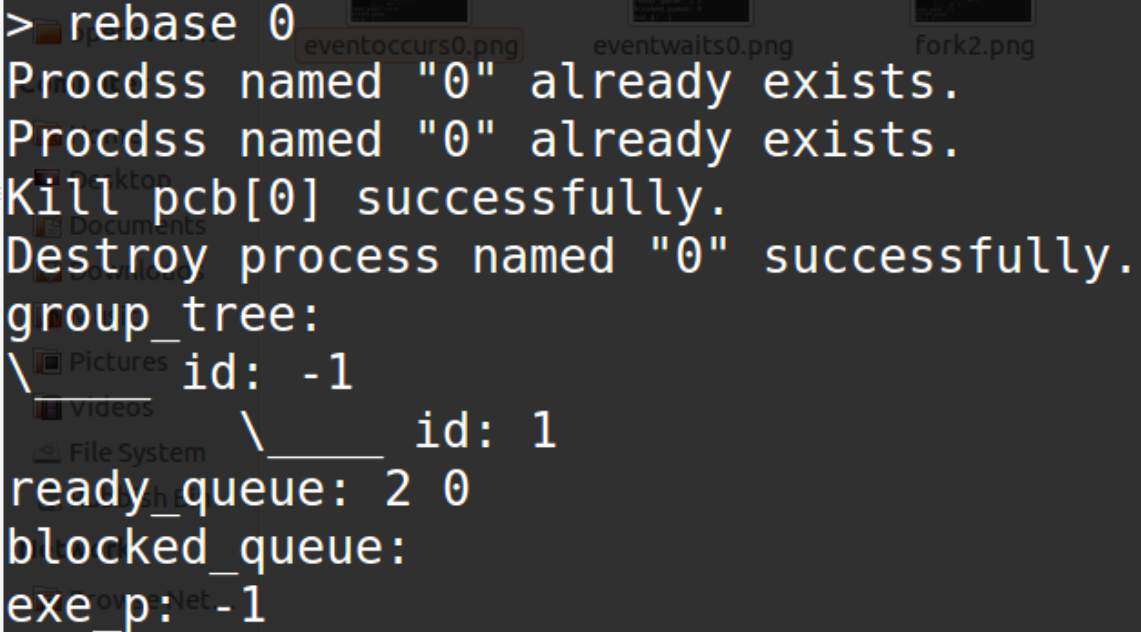
```
> timeout 0
Procdss named "0" already exists.
Schedule process "0" successfully when time out.
group_tree:
\Computer id: -1
  Home \ id: 0
  Desktop | \ id: 2
  Documents | \ id: 1
  Downloads \ id: 1
ready_queue: 1 2 0 0
blocked_queue:
exe_p: -1
```

Figure 15: timeout 0

```
> dispatch 0
Procdss named "0" already exists.
Procdss named "0" already exists.
Running process "0" successfully.
group_tree:
\_____ id: -1
  \_____ id: 0
    |_____ id: 1
    |_____ id: 2
ready_queue: 2 0
blocked_queue:
exe_p: 0
```


Figure 16: dispatch 0





```
> rebase 0
Procdss named "0" already exists.
Procdss named "0" already exists.
Kill pcb[0] successfully.
Destroy process named "0" successfully.
group_tree:
\ id: -1
  \ id: 1
ready_queue: 2 0
blocked_queue:
exe_p: -1
```

Figure 17: rebase 0



```
> quit 1
zephyr@ubuntu ~/code/cpp/operating-system/12
%
```

Figure 18: quit 1