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

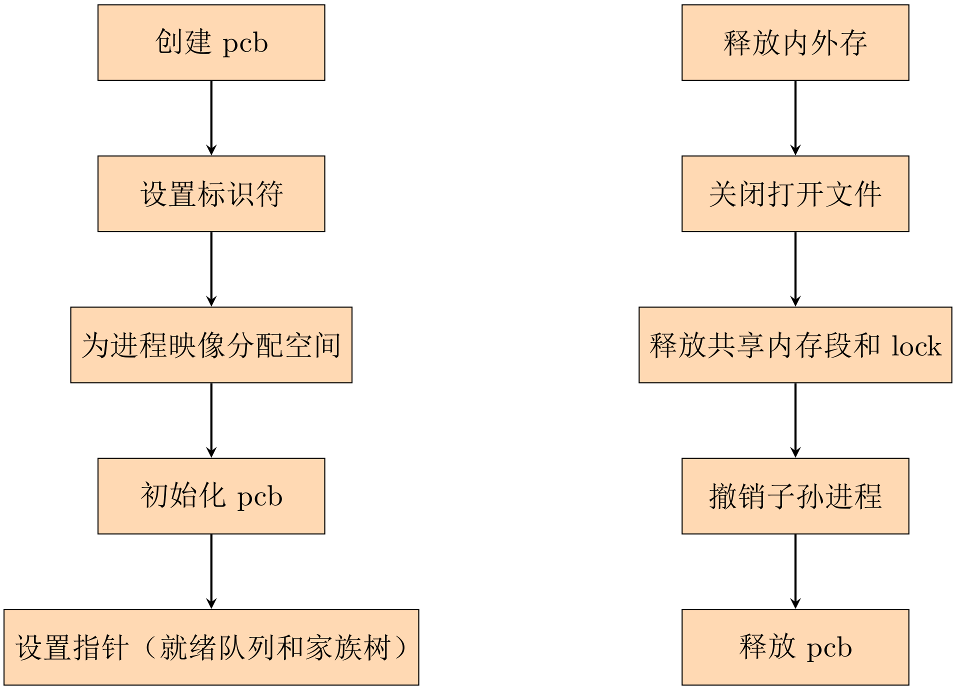
自行编制模拟程序，通过形象化的状态显示，使学生理解进程的概念、进程之间的状态转换及其所带来的 PCB 内容、组织的变化，理解进程与其 PCB 间的一一对应关系。

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



(a) 创建原语

(b) 撤销原语

Figure 1: 创建原语和撤销原语

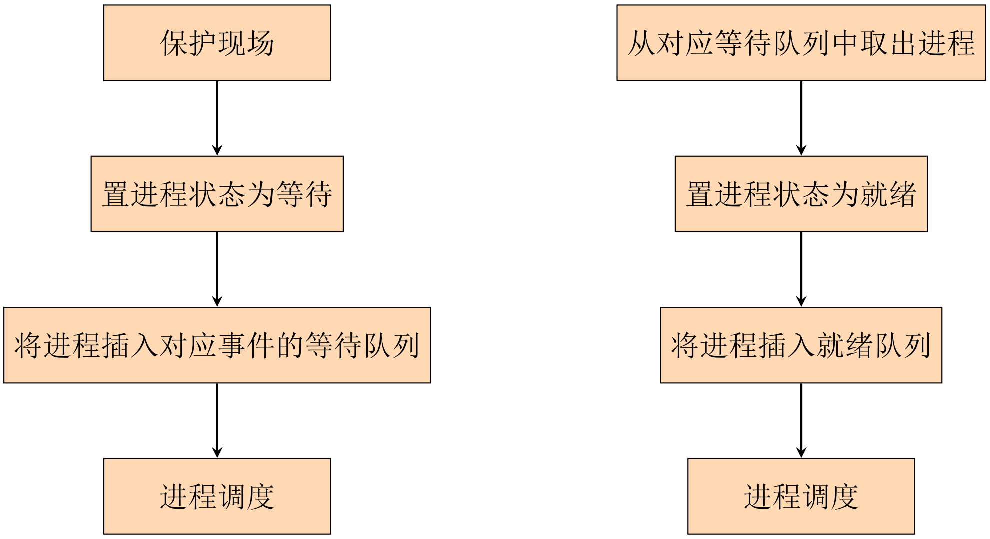
5 使用的数据结构及其说明

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：

```
#include <algorithm>
#include <cassert>
#include <cmath>
```



(a) 阻塞原语

(b) 唤醒原语

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

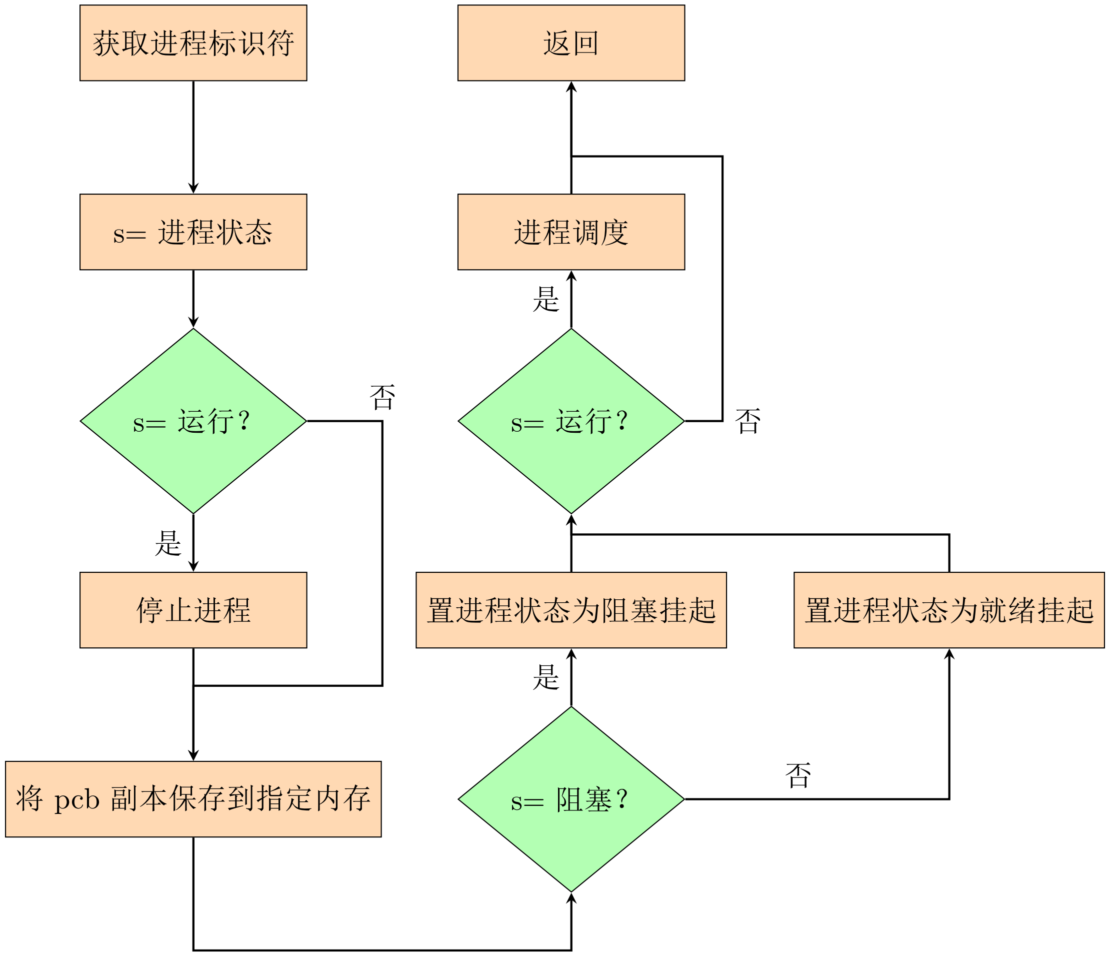


Figure 3: 挂起原语

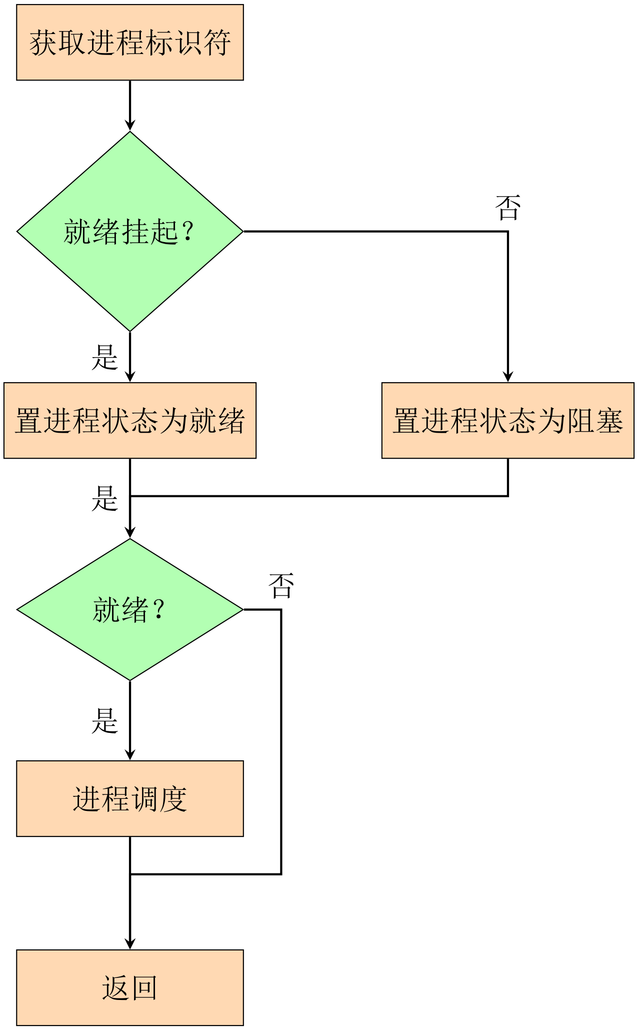


Figure 4: 激活原语

```
#include <initializer_list>
#include <iostream>
#include <fstream>
#include <functional>
#include <set>
#include <sstream>
#include <vector>

#include "pcb.h"
#include "group_tree.h"

using namespace std;

const int concurrency = 256;

vector<Pcb> pcb_table;
vector<int> ready_queue;
vector<int> blocked_queue;
int id = 0;
int exe_p;
bool schedule = false;
GroupTree group_tree;

bool get_pcb(string name, int& i){
  // return false: exists; true: not exists
  // i: index (-1, allocate memory failed)
  auto p = find_if(pcb_table.begin(), pcb_table.end(),
    [name](const Pcb& pcb){
      return pcb.name == name;
    });
  if(p == pcb_table.end()){
    if(pcb_table.size() + 1 > concurrency){
      i = -1;
      cerr << "Outstrip capacity of concurrency, apply for pcb failed!\n";
      return false;
    }else{
      pcb_table.push_back(Pcb());
      i = pcb_table.size() - 1;
      cerr << "Process named \"" << name << "\" not found, create successfully!\n";
      return false;
    }
  }else{
    i = p - pcb_table.begin();
    cerr << "Procdss named\"" << name << "\" already exists.\n";
    return true;
  }
}

bool create(string name, CpuState cpu_state, int priority,
   MainStore main_store, Resources resources){
  int i;
  if(!get_pcb(name, i) && i != -1){
    pcb_table[i].id = ++id;
    pcb_table[i].priority = priority;
    pcb_table[i].cpu_state = cpu_state;
    pcb_table[i].main_store = main_store;
    pcb_table[i].resources = resources;
    ready_queue.push_back(i);
    // group_tree
    cerr << "Apply pcb successfully.\n";
    return true;
  }else{
    cerr << "Error: create process named \"" << name << "\"
   failed!\n";
    return false;
  }
}

bool remove(int i){
  if(pcb_table[i].status == ready){
    ready_queue.erase(find_if(ready_queue.begin(), ready_queue.
   end(),
      [i](const int& idx){
        return i == idx;
      }));
    cerr << "Remove pcb[" << i << "] from ready_queue
   successfully.\n";
    return true;
  }else if(pcb_table[i].status == blocked){
    blocked_queue.erase(find_if(blocked_queue.begin(),
   blocked_queue.end(),
      [i](const int& idx){
        return i == idx;
      }));
    cerr << "Remove pcb[" << i << "] from blocked_queue successfully.\n";
    return true;
  }else{
    return false;
  }
}

bool kill(int i){
  if(i >= 0){
    if(pcb_table[i].status == running){
      // stop(i)
      schedule = true;
    }
    remove(i);
    // kill subtree in group_tree
    // release main_store and resources
    pcb_table.erase(pcb_table.begin() + i);
    cerr << "Kill pcb[" << i << "] successfully.\n";
    return true;
  }else{
    cerr << "Error: kill pcb[" << i << "] failed!\n";
    return false;
  }
}

bool destroy(string name){
  schedule = false;
  int i;
  if(get_pcb(name, i)){
    kill(i);
    if(schedule){
      // scheduler dispatches
    }
    cerr << "Destroy process named \"" << name << "\" successfully.\n";
    return true;
  }else{
    cerr << "Error: destroy process named \"" << name << "\" failed!\n";
    return false;
  }
}

bool block(){
  int i = exe_p;
  if(exe_p >= 0)
  {
    // stop(i)
    pcb_table[i].status = blocked;
    blocked_queue.push_back(i);
    // scheduler dispatches
    cerr << "Block successfully.\n";
    return true;
  }else{
    cerr << "Error: block failed!\n";
    return false;
  }
}

bool wake_up(string name){
  int i;
  if(get_pcb(name, i)){
    remove(i);
    pcb_table[i].status = ready;
    ready_queue.push_back(i);
    // scheduler dispatches
    cerr << "Wake up process named \"" << name << "\" successfully.\n";
    return true;
  }else{
    cerr << "Error: wake up process named \"" << name << "\" failed!\n";
    return false;
  }
}

bool suspend(string name){
  int i;
  if(get_pcb(name, i)){
    Status s = pcb_table[i].status;
    if(s == running){
      // stop(i)
    }
    // main_store <- pcb_table[i]
    if(s == blocked){
      pcb_table[i].status == blocked_suspend;
    }else{// ready / running
      pcb_table[i].status = ready_suspend;
    }
    if(s == running){
      // scheduler dispatches
    }
    cerr << "Suspend process named \"" << name << "\" successfully.\n";
    return true;
  }else{
    cerr << "Error: suspend process named \"" << name << "\" failed!\n";
    return false;
  }
}

bool activate(string name){
  int i;
  if(get_pcb(name, i)){
    pcb_table[i].status = pcb_table[i].status == ready_suspend? ready: blocked;
    if(pcb_table[i].status == ready){
      // scheduler dispatches
    }
    cerr << "Activate process named \"" << name << "\" successfully.\n";
    return true;
  }else{
    cerr << "Error: activate process named \"" << name << "\" failed!\n";
    return false;
  }
```

```
}

int main(int argc, char** argv){
  group_tree.create(-1, 0);
  group_tree.create(-1, 1);
  group_tree.create(-1, 2);
  group_tree.create(0, 4);
  group_tree.create(4, 5);
  group_tree.create(4, 6);
  group_tree.create(6, 7);
  group_tree.create(1, 3);
  group_tree.preorder_travers_display();
  group_tree.destroy(4);
  group_tree.preorder_travers_display();
  return 0;
}
```

Listing 1: `thread.cpp` 代码清单

```
#ifndef PCB_H
#define PCB_H

#include <algorithm>
#include <cassert>
#include <cmath>
#include <initializer_list>
#include <iostream>
#include <iterator>
#include <fstream>
#include <functional>
#include <regex>
#include <set>
#include <sstream>
#include <vector>

using namespace std;

enum CpuState{usr_state, kernel_state};
enum Status{creat, ready, ready_suspend, blocked, blocked_suspend
    , running, ex};
struct MainStore{
  int start_addr;
  int mem_size;
  bool usable;
};
struct Resources{
  vector<string> opened_files;
};

class Pcb{
public:
  Pcb(){}

// description
  int id;
  string name;
  string usr_id;
  // group_tree

// ctrl_infl
  CpuState cpu_state;
  Status status;
  int priority;
  // int entry_addr;
  // int disk_addr;
  // StatInfo stat_info;

// resources
  // main_store
  MainStore main_store;
  Resources resources;

// cpu_scene

private:
};

#endif // PCB_H
```

Listing 2: pcb.h 代码清单

```
#ifndef GROUP_TREE_H
#define GROUP_TREE_H

#include <algorithm>
#include <cassert>
#include <cmath>
#include <initializer_list>
#include <iostream>
#include <iterator>
#include <fstream>
#include <functional>
#include <regex>
#include <set>
#include <sstream>
#include <vector>

#include "pcb.h"

using namespace std;

struct Node{
  Node(){}
  Node(int n, Node* pp, vector<Node*> cpl):
  pcb_id(n), parent_ptr(pp), child_ptr_list(cpl){}

  int pcb_id;
  Node* parent_ptr;
  vector<Node*> child_ptr_list;
};

class GroupTree{
public:
  GroupTree(){
    root = new Node();
    root->pcb_id = -1;
  }

  void destruct(Node* p){
    if(p != nullptr){
      for(size_t i = 0; i < p->child_ptr_list.size(); ++i){
        destruct(p->child_ptr_list[i]);
      }
      delete p;
    }
  }

  ~GroupTree(){
    destruct(root);
    // cout << "destruct\n";
  }

  void find(int i, Node* p, bool& found, Node*& pp){
    if(p != nullptr){
      if(p->pcb_id == i){
        found = true;
        pp = p;
        return;
      }else{
        for(size_t j = 0; j < p->child_ptr_list.size(); ++j){
          find(i, p->child_ptr_list[j], found, pp);
        }
      }
    }
  }

  bool create(int i, int pcb_id){
    Node* pp;
    bool found = false;
    find(i, root, found, pp);
    // if(pp != nullptr){
    //  cout << "find " << i << " " << pp->pcb_id << endl;
    // }else{
    //  cout << "find null\n";
    // }
    if(found){
      Node* np = new Node();
      np->pcb_id = pcb_id;
      np->parent_ptr = pp;
      pp->child_ptr_list.push_back(np);
    }
  }

```

```
bool destroy(int i){
  Node* p;
  bool found = false;
  find(i, root, found, p);
  if(found){
    p->parent_ptr->child_ptr_list.erase(
      find_if(
        p->parent_ptr->child_ptr_list.begin(), p->parent_ptr->child_ptr_list.end(),
        [i](Node* cp){
          return cp->pcb_id == i;
        }));
    destruct(p);
  }
}

void predisp(vector<int>& stk, Node* p, bool b1){
  if(p != nullptr){
    for(int i = 0; i < stk.size(); ++i){
      cout << (stk[i] == 0? " ": "|") << "\t";
    }
    cout << "\\____ ";
    // display node
    cout << "id: " << p->pcb_id;
    cout << "\n";
    for(size_t i = 0; i < p->child_ptr_list.size(); ++i){
      if(b1){
        stk.push_back(0);
      }else{
        stk.push_back(1);
      }
      predisp(stk, p->child_ptr_list[i], i == p->child_ptr_list.size() - 1);
      stk.erase(stk.end() - 1);
    }
  }
}

void preorder_travers_display(){
  vector<int> stk;
  bool b = true; // show whether my parent is the last child of
```

```
    my grand parent
    predisp(stk, root, b);
  }

  Node* root;
private:
};

#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。

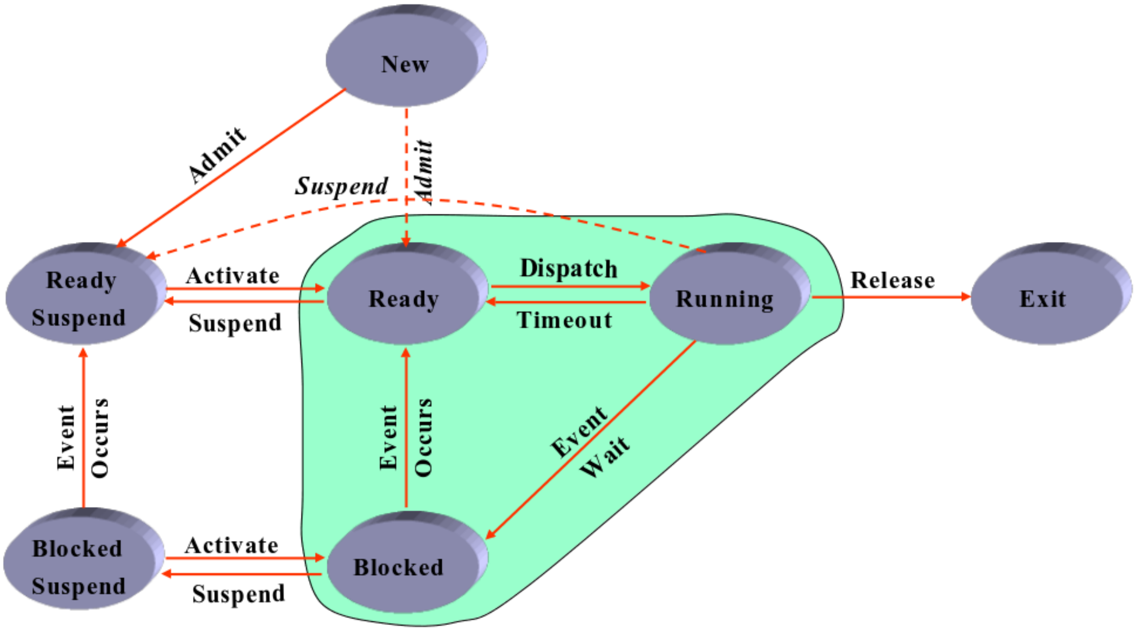
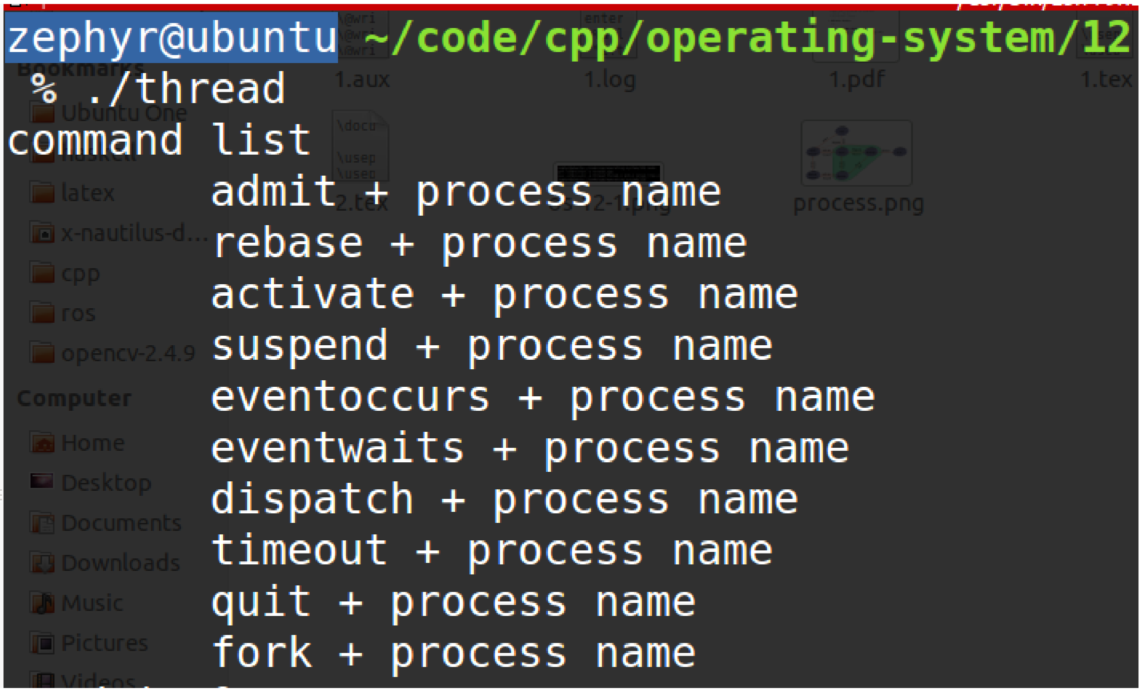


Figure 5: 进程状态转换的 7 状态模型

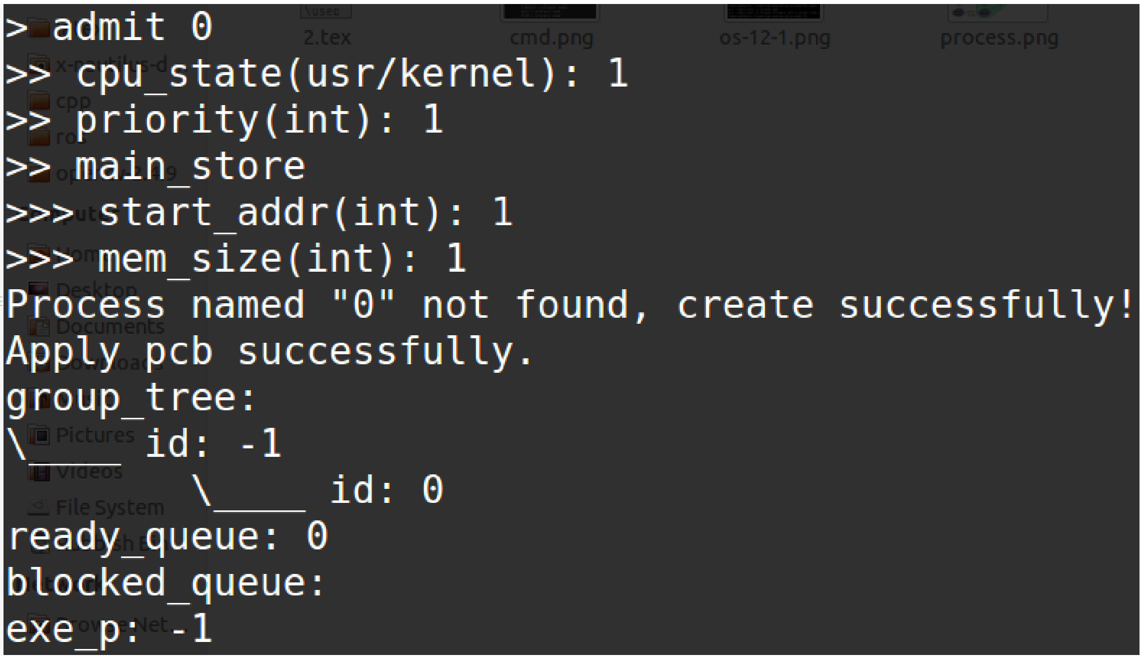
运行结果	说明
图 6	打印程序使用说明（所有命令格式）
图 7	执行 `admit 0`，收容 0 进程（并就绪）
图 8	执行 `admit 1`，收容 1 进程（并就绪）
图 9	执行 `dispatch 0`，调度 0 进程执行（从就绪队列切上 CPU）
图 10	执行 `fork 2`，为 0 进程创建子进程 2（子进程就绪）
图 11	执行 `eventwaits 0`，0 进程等待某事件发生，被阻塞（从 CPU 切下进入阻塞队列）
图 12	执行 `eventoccurs 0`，0 进程等待的事件发生，被唤醒（由阻塞队列进入就绪队列）
图 13	执行 `suspend 0`，挂起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

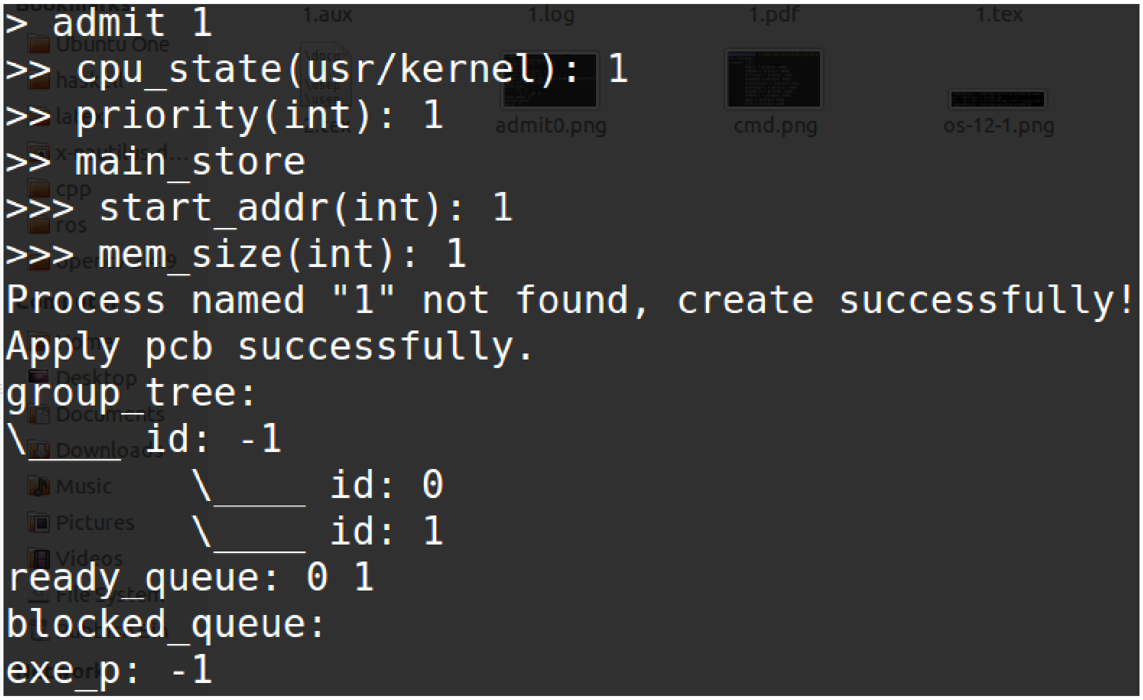


```
> admit 0
>> cpu_state(usr/kernel): 1
>> priority(int): 1
>> main_store
>>> start_addr(int): 1
>>> mem_size(int): 1
Process named "0" not found, create successfully!
Apply pcb successfully.
group_tree:
\_____ id: -1
        \_____ id: 0
ready_queue: 0
blocked_queue:
exe_p: -1
```

Figure 7: `admit 0`

8 程序使用说明

见表 3

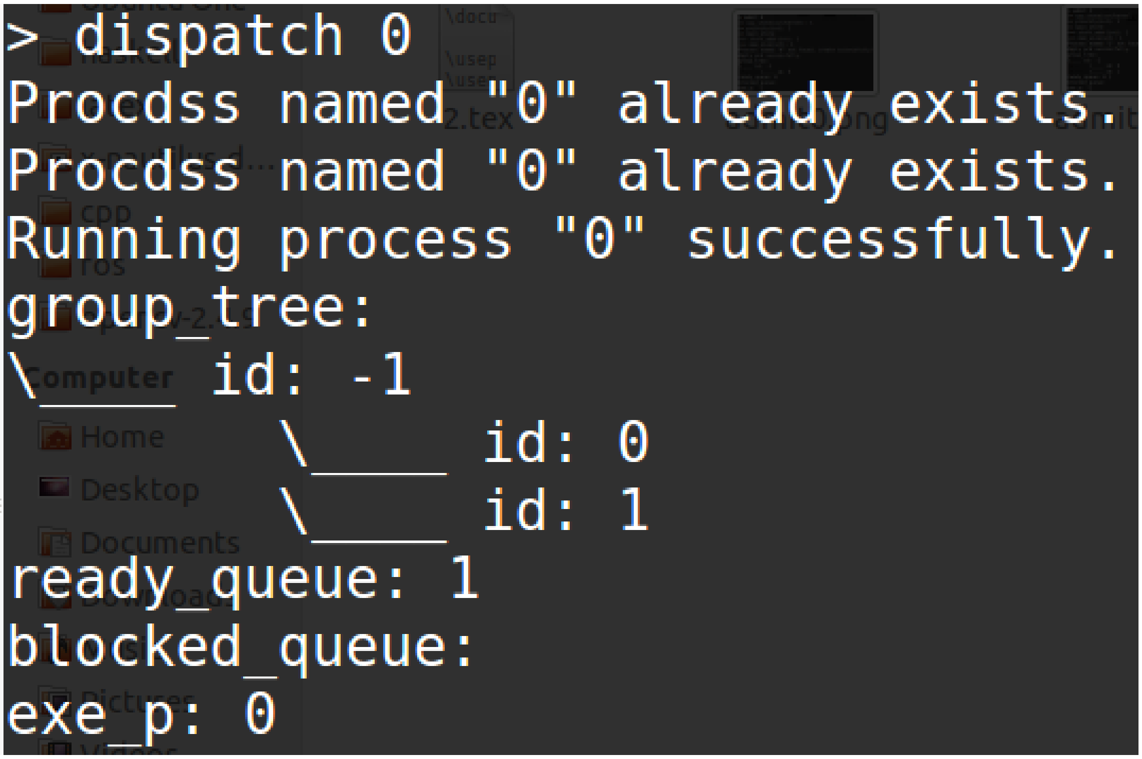


```
> admit 1
>> cpu_state(usr/kernel): 1
>> priority(int): 1
>> main_store
>>> start_addr(int): 1
>>> mem_size(int): 1
Process named "1" not found, create successfully!
Apply pcb successfully.
group_tree:
\_____ id: -1
        \_____ id: 0
        \_____ id: 1
ready_queue: 0 1
blocked_queue:
exe_p: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

Figure 8: `admit 1`

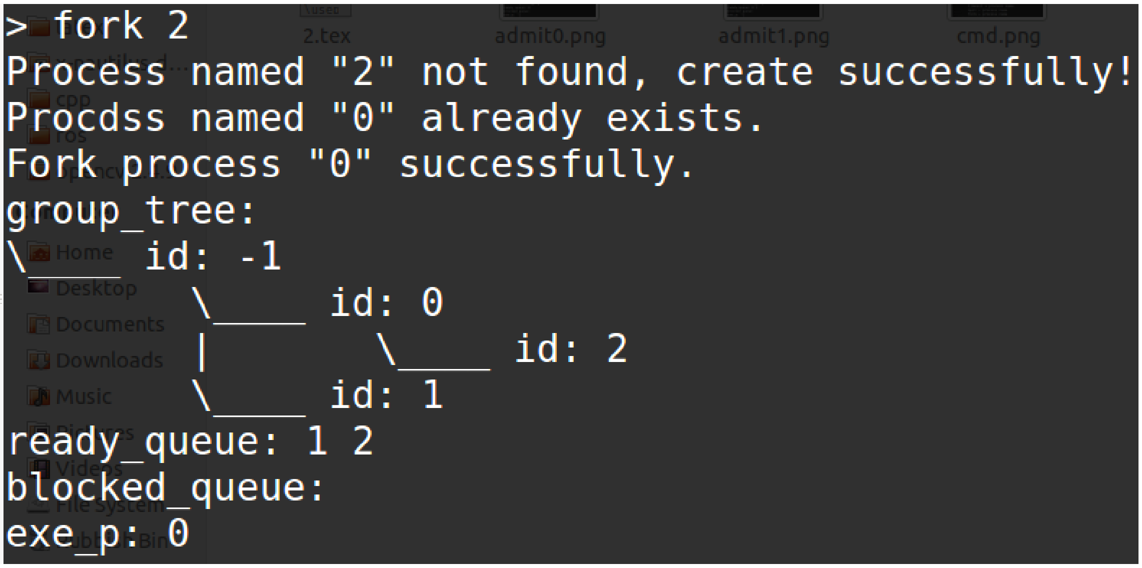
admit（收容）	cpu_state（CPU 状态）	
	priority（优先级）	
	main_store（主存）	start_addr（起始地址） mem_size（占用内存大小）
dispatch（调度执行）		
suspend（挂起）		
activate（激活）		
eventwaits（等待事件发生，阻塞）		
eventoccurs（等待的事件发生，唤醒）		
timeout（超时，就绪）		
rebase（杀死进程及其所有子孙进程）		
fork（创建子进程）		
quit（退出）		

Table 3: 程序使用说明



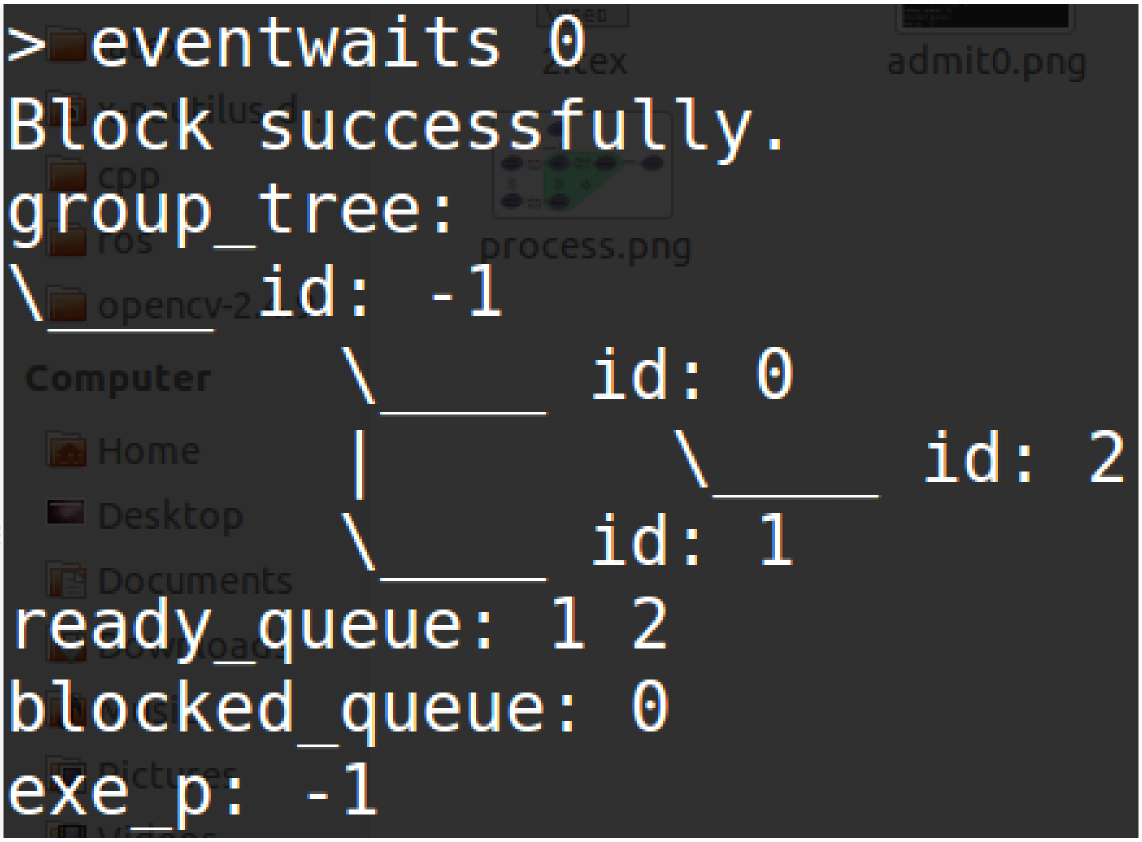
```
> dispatch 0
Procdss named "0" already exists.
Procdss named "0" already exists.
Running process "0" successfully.
group_tree:
\____ id: -1
        \____ id: 0
        \____ id: 1
ready_queue: 1
blocked_queue:
exe_p: 0
```

Figure 9: `dispatch 0`



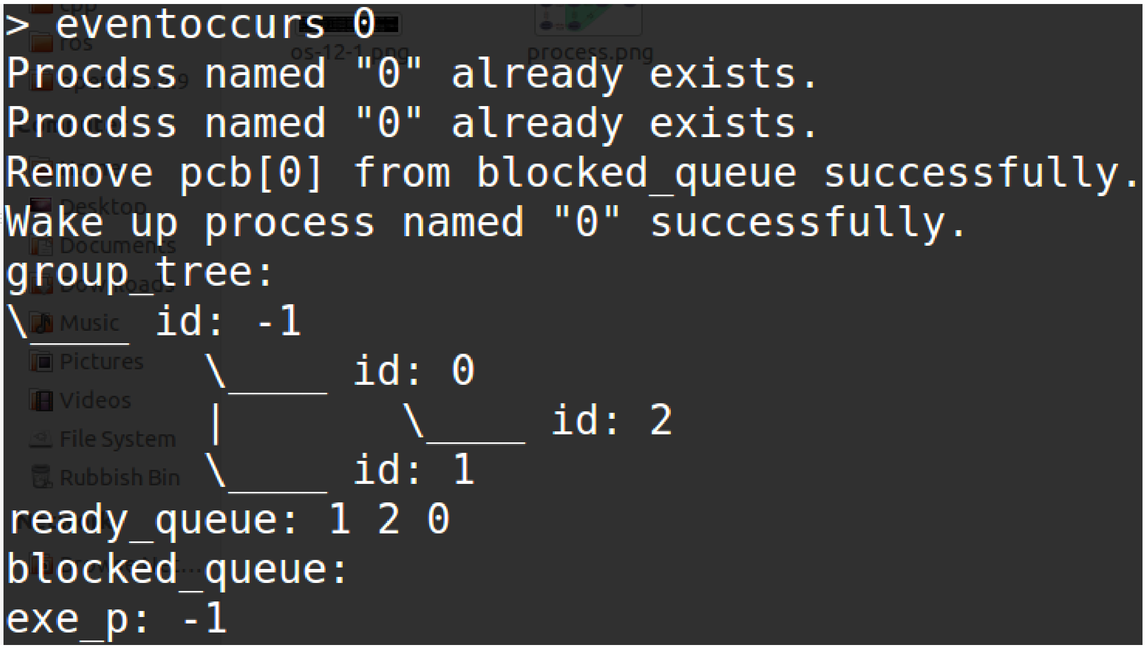
```
> fork 2
Process named "2" not found, create successfully!
Procdss named "0" already exists.
Fork process "0" successfully.
group_tree:
\____ id: -1
        \____ id: 0
        |       \____ id: 2
        \____ id: 1
ready_queue: 1 2
blocked_queue:
exe_p: 0
```

Figure 10: `fork 2`



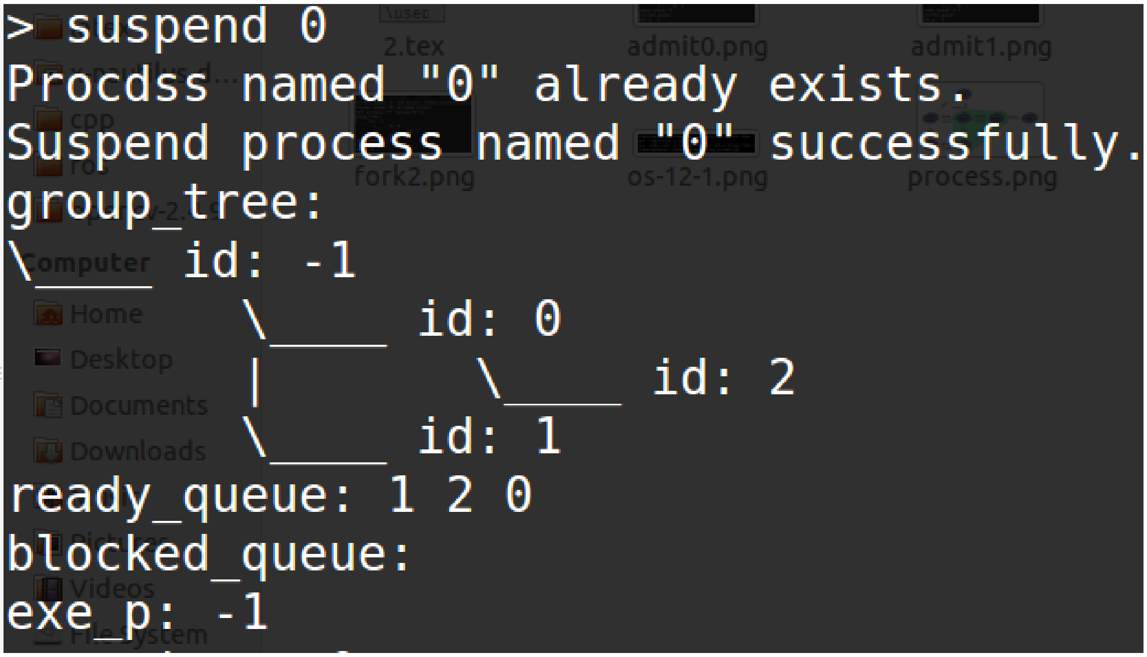
```
>eventwaits 0
Block successfully.
group_tree:
\____ id: -1
        \____ id: 0
        |       \____ id: 2
        \____ id: 1
ready_queue: 1 2
blocked_queue: 0
exe_p: -1
```

Figure 11: `eventwaits 0`



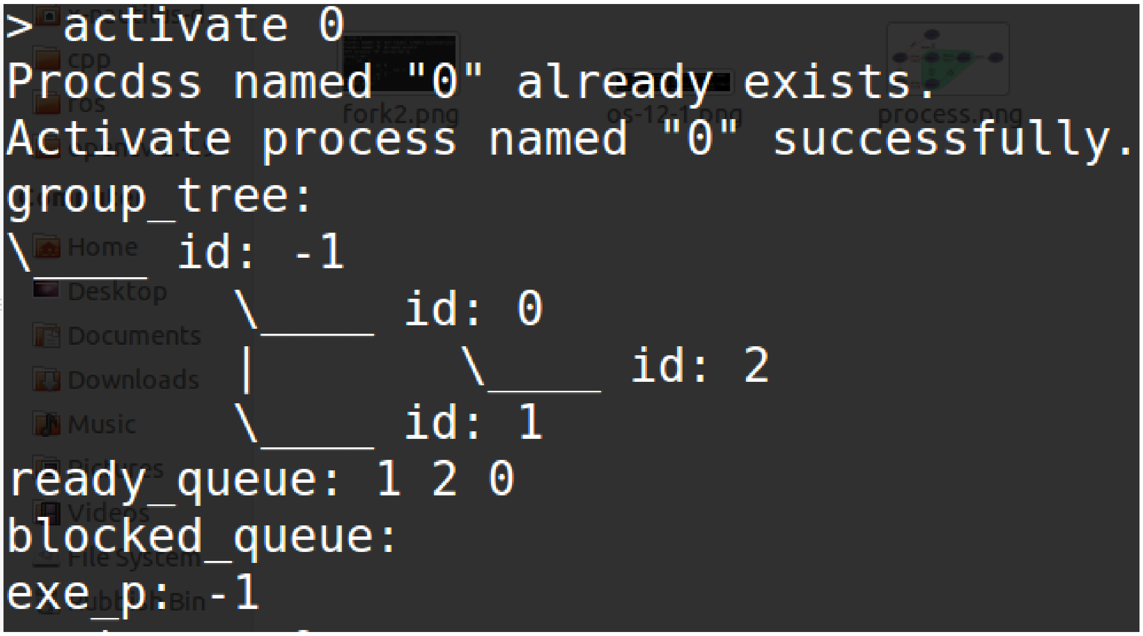
```
> eventoccurs 0
Procdss named "0" already exists.
Procdss named "0" already exists.
Remove pcb[0] from blocked_queue successfully.
Wake up process named "0" successfully.
group_tree:
\_____ id: -1
        \_____ id: 0
        |        \_____ id: 2
        \_____ id: 1
ready_queue: 1 2 0
blocked_queue: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

Figure 12: eventoccurs0



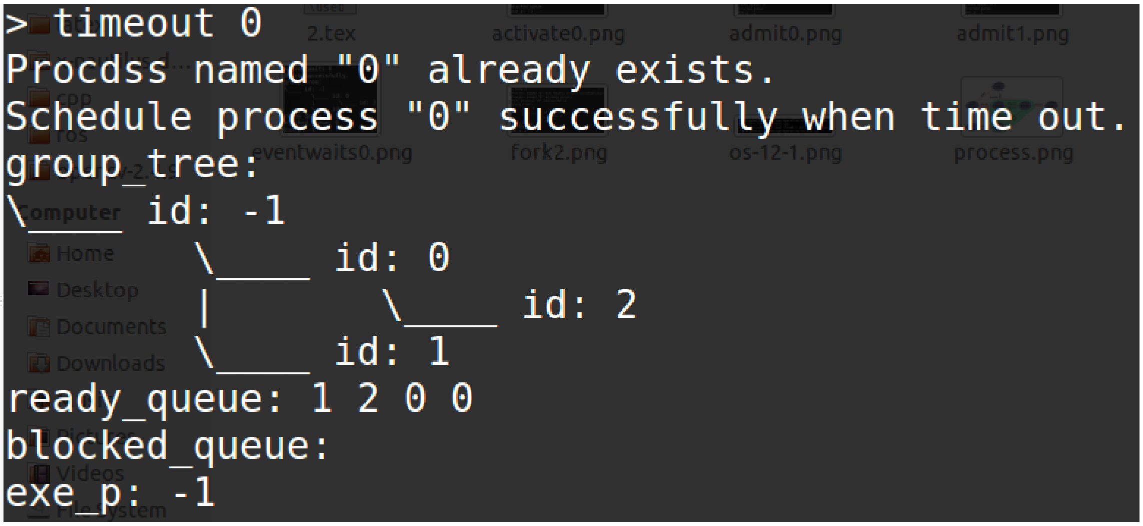
```
> suspend 0
Procdss named "0" already exists.
Suspend process named "0" successfully.
group_tree:
\_____ id: -1
        \_____ id: 0
        |        \_____ id: 2
        \_____ id: 1
ready_queue: 1 2 0
blocked_queue:
exe_p: -1
```

Figure 13: suspend 0



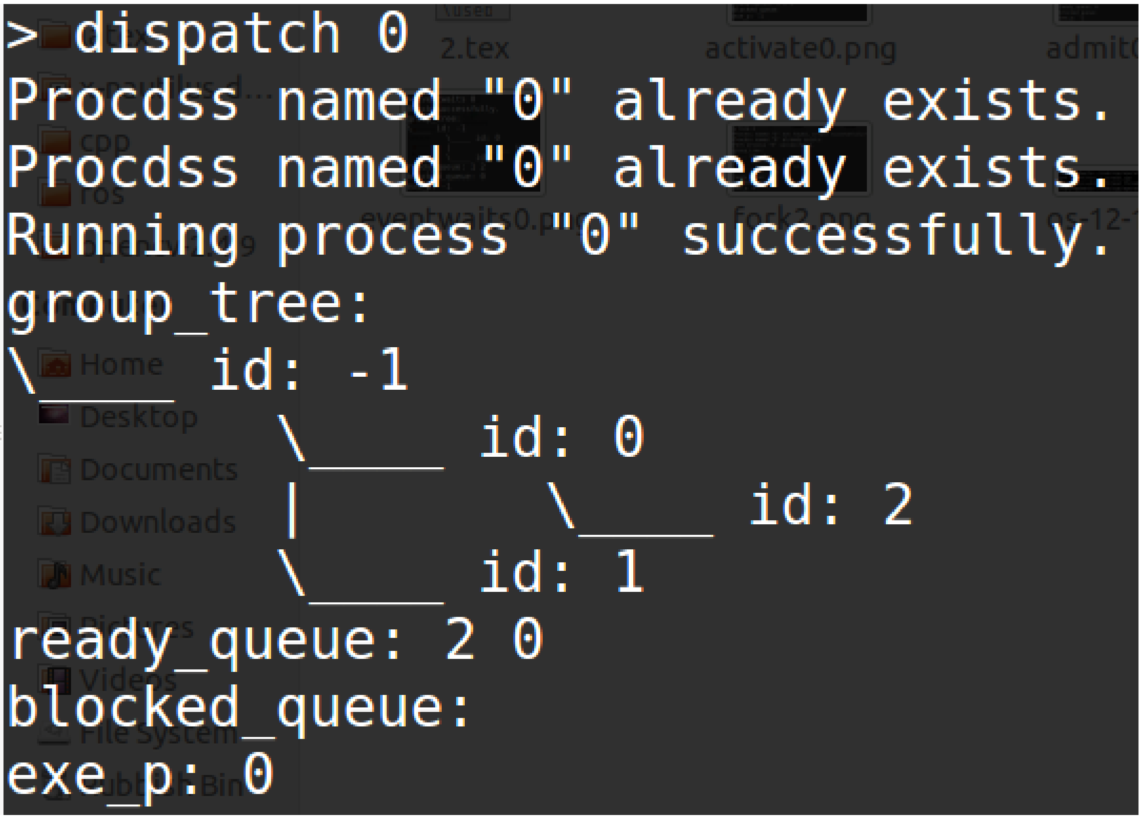
```
> activate 0
Procdss named "0" already exists.
Activate process named "0" successfully.
group_tree:
\____ id: -1
        \____ id: 0
        |       \____ id: 2
        \____ id: 1
ready_queue: 1 2 0
blocked_queue:
exe_p: -1
```

Figure 14: `activate 0`



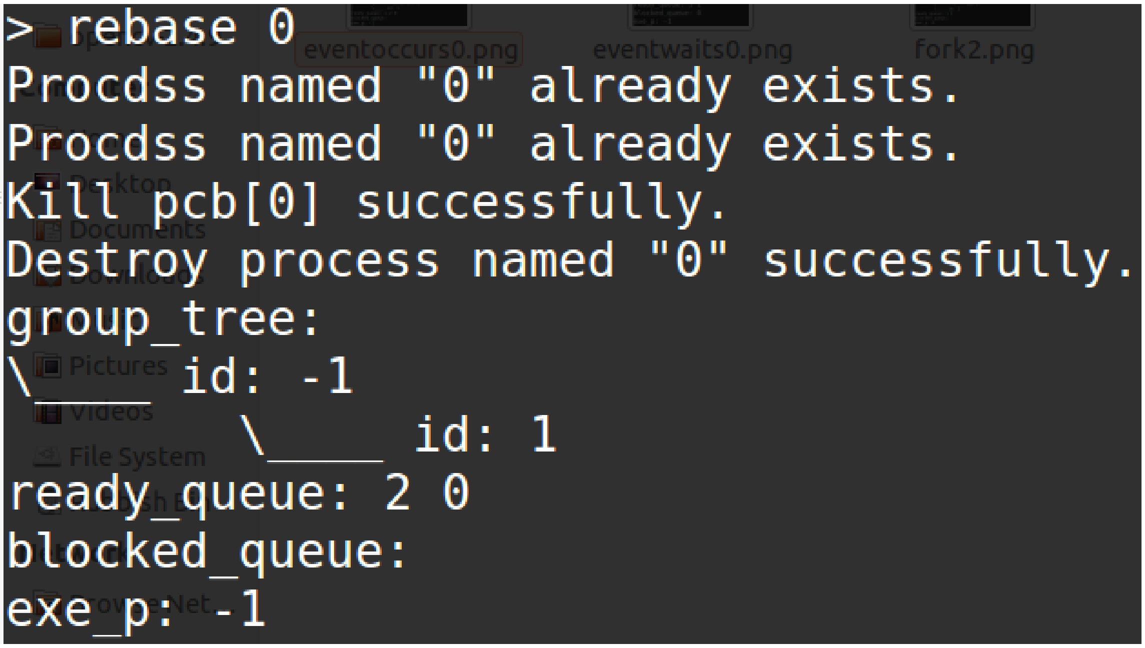
```
> timeout 0
Procdss named "0" already exists.
Schedule process "0" successfully when time out.
group_tree:
\____ id: -1
        \____ id: 0
        |       \____ id: 2
        \____ id: 1
ready_queue: 1 2 0 0
blocked_queue:
exe_p: -1
```

Figure 15: `timeout 0`



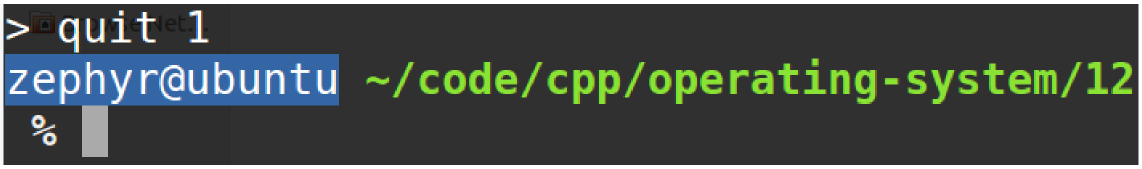
```
> dispatch 0
Procdss named "0" already exists.
Procdss named "0" already exists.
Running process "0" successfully.
group_tree:
\_____ id: -1
        \_____ id: 0
        |       \_____ id: 2
        \_____ id: 1
ready_queue: 2 0
blocked_queue:
exe_p: 0
```

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`