



上海大学

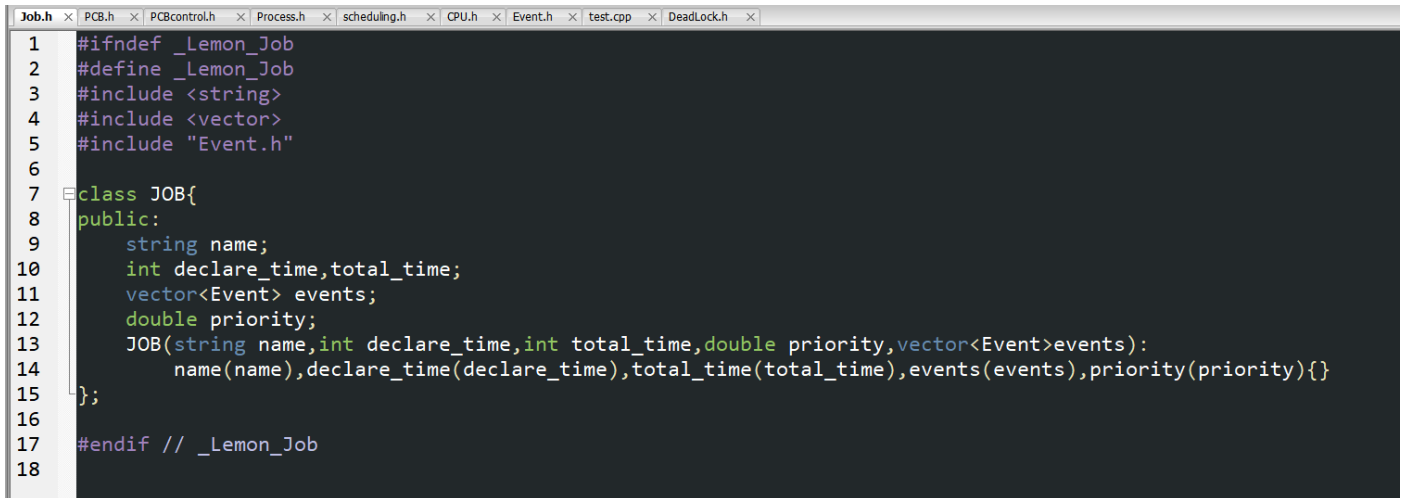
SHANGHAI UNIVERSITY

操作系统（二）实验一、二报告

组	号	第 8 组
学	号	17122490
姓	名	秦敏浩

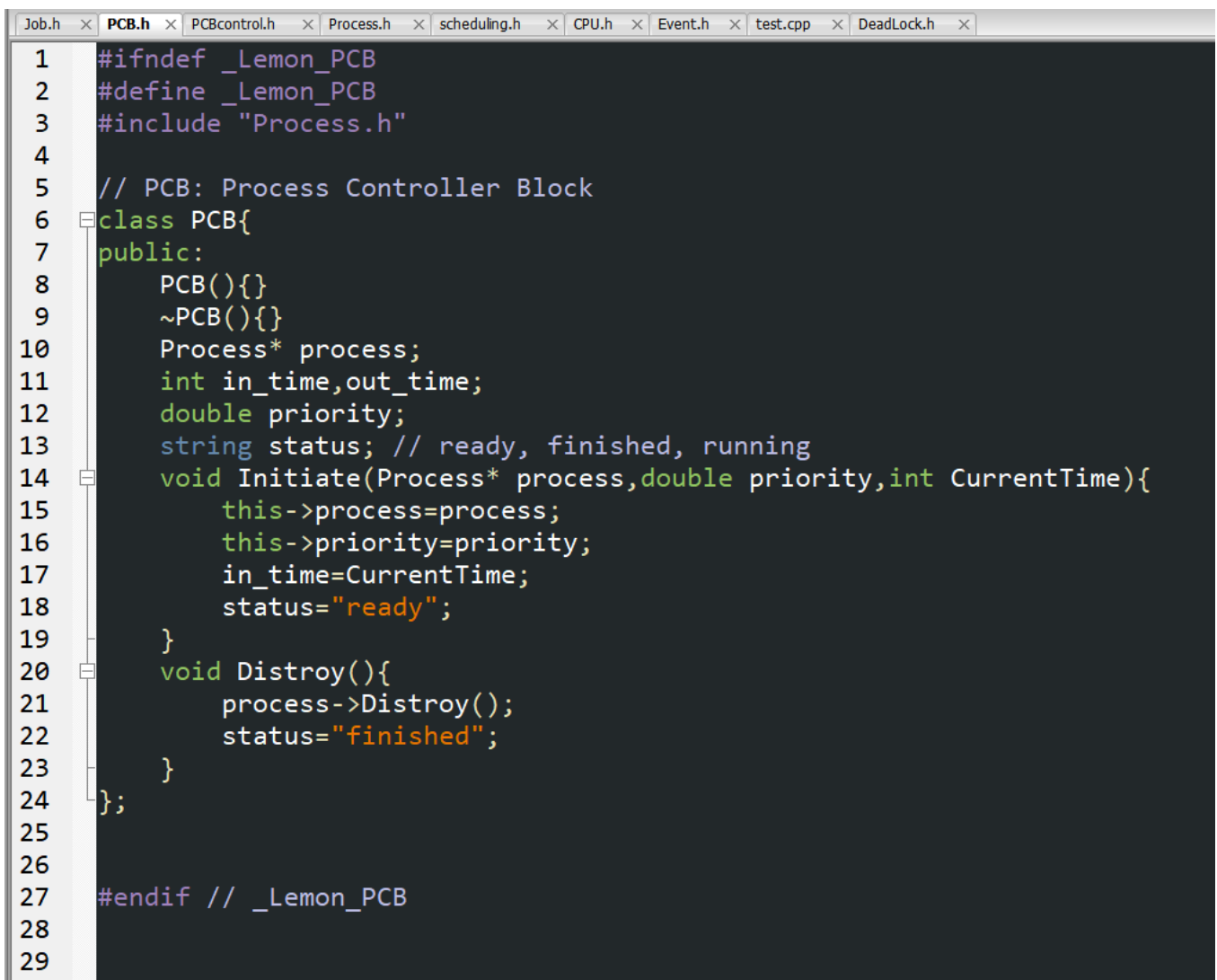
一、 代码截屏图片

Job.h: 作业类



```
1 #ifndef _Lemon_Job
2 #define _Lemon_Job
3 #include <string>
4 #include <vector>
5 #include "Event.h"
6
7 class JOB{
8 public:
9     string name;
10    int declare_time,total_time;
11    vector<Event> events;
12    double priority;
13    JOB(string name,int declare_time,int total_time,double priority,vector<Event>events):
14        name(name),declare_time(declare_time),total_time(total_time),events(events),priority(priority){}
15 };
16
17 #endif // _Lemon_Job
18
```

PCB.h: PCB 类（记录进程所在地址和重要信息）



```
1 #ifndef _Lemon_PCB
2 #define _Lemon_PCB
3 #include "Process.h"
4
5 // PCB: Process Controller Block
6 class PCB{
7 public:
8     PCB(){}
9     ~PCB(){}
10    Process* process;
11    int in_time,out_time;
12    double priority;
13    string status; // ready, finished, running
14    void Initiate(Process* process,double priority,int CurrentTime){
15        this->process=process;
16        this->priority=priority;
17        in_time=CurrentTime;
18        status="ready";
19    }
20    void Destroy(){
21        process->Destroy();
22        status="finished";
23    }
24 };
25
26
27 #endif // _Lemon_PCB
28
29
```

PCBcontrol.h: 将 PCB 链表封装成类（用于产生就绪队列、阻塞队列等）

```
Job.h x PCB.h x PCBcontrol.h x Process.h x scheduling.h x CPU.h x Event.h x test.cpp x DeadLock.h x
1 #ifndef Lemon_PCB_Control
2 #define Lemon_PCB_Control
3 #include <cassert>
4 #include "PCB.h"
5
6 // A PCB Queue
7 class PCBControl{
8 public:
9     vector<PCB*> PCB_List;
10    void initiate(){
11        PCB_List.clear();
12    }
13    void insert_process(string name,int total_time,double priority,vector<Event> events,int CurrentTime,int resource_n){
14        PCB *pcb=new PCB;
15        Process *p=new Process;
16        p->Initiate(total_time,name,events,resource_n);
17        pcb->Initiate(p,priority,CurrentTime);
18        PCB_List.push_back(pcb);
19    }
20    void insert_process(PCB *pcb){
21        PCB_List.push_back(pcb);
22    }
23    void free_process(PCB* p){
24        for (vector<PCB*>::iterator it=PCB_List.begin();it!=PCB_List.end();++it){
25            if (*it==p){
26                PCB_List.erase(it);
27                return;
28            }
29        }
30        assert(false);
31    }
32 };
33
34 #endif // Lemon_PCB_Control
35
36
```

Process.h: 进程类

```
Job.h x PCB.h x PCBcontrol.h x Process.h x scheduling.h x CPU.h x Event.h x test.cpp x DeadLock.h x
1 #ifndef _Lemon_Process
2 #define _Lemon_Process
3 #include <iostream>
4 #include "Event.h"
5
6 class Process{
7 public:
8     Process(){}
9     ~Process(){}
10    int total_time;
11    int run_time;
12    int stuck_time;
13    string name;
14    vector<Event> events;
15    vector<int> resources;
16
17    void Initiate(int total_time,string name,vector<Event> events,int resources_n){
18        this->name=name;
19        this->total_time=total_time;
20        this->run_time=0;
21        this->events=events;
22        resources.clear();
23        resources.resize(resources_n,0);
24        stuck_time=0;
25    }
26
27    void Destroy(){
28        name="";
29        total_time=0;
30        events.clear();
31    }
32 };
33
34 #endif // _Lemon_Process
35
36
37
```

Scheduling.h: 作业调度算法（FCFS（非抢占）、SJF（抢占）、RR（时间片））

```
Job.h x PCB.h x PCBcontrol.h x Process.h x scheduling.h x CPU.h x Ev
1  #ifndef _Lemon_scheduling
2  #define _Lemon_scheduling
3  #include "PCBcontrol.h"
4
5  // process scheduling algorithm: first come first serve.
6  // FCFS selects the process that comes earlier.
7  namespace FCFS{
8      PCB* select_next_process(vector<PCB*>& PCB_List,PCB* cur){
9          if (cur) return cur;
10         PCB* ret=NULL;
11         int best_in=0x3f3f3f3f;
12         for(auto pcb:PCB_List){
13             if (pcb->in_time<best_in){
14                 best_in=pcb->in_time;
15                 ret=pcb;
16             }
17         }
18         return ret;
19     }
20 }
21
22 // process scheduling algorithm: shortest job first.
23 // SJF selects the process that has the lowest runtime (priority).
24 namespace SJF{
25     PCB* select_next_process(vector<PCB*>& PCB_List,PCB* cur){
26         PCB* ret=NULL;
27         double best_priority=1e10;
28         for (auto pcb:PCB_List){
29             if (pcb->priority<best_priority){
30                 best_priority=pcb->priority;
31                 ret=pcb;
32             }
33         }
34         if (ret){
35             ret->priority+=1.0;
36         }
37         return ret;
38     }
39 }
40
41 // process scheduling algorithm: round robin
42 // RR selects the next process of current process.
43 namespace RR{
44     PCB* select_next_process(vector<PCB*>& PCB_List,PCB* cur){
45         if (PCB_List.size()==0) return NULL;
46         if (cur==NULL) return PCB_List[0];
47         vector<PCB*>::iterator it;
48         for (it=PCB_List.begin();it!=PCB_List.end();++it){
49             if (*it==cur) break;
50         }
51         assert(it!=PCB_List.end());
52         ++it;
53         if (it==PCB_List.end()) return PCB_List[0];
54         return *it;
55     }
56 }
57
58 #endif // _Lemon_scheduling
59
60
```

CPU.h: 运行逻辑 可见附件或访问 <https://paste.ubuntu.com/p/cvfkQskFdZ/>

[illegible]

Event.h: 事件类（如请求系统资源等）

```

1  #ifndef _Lemon_Event
2  #define _Lemon_Event
3  #include <string>
4  using namespace std;
5
6  class Event{
7  public:
8      string reason;
9      int start_time;
10     int total_time;
11     vector<int> requirements;
12
13     Event(string reason,int start_time,int total_time,vector<int> requirements):
14         reason(reason),start_time(start_time),total_time(total_time),requirements(requirements){}
15
16     void debug(){
17         cout<<reason<<"", "<<total_time<<" ";
18         if (reason=="Require res"){
19             cout<<"[ ";
20             for (auto x:requirements){
21                 cout<<x<<" ";
22             }
23             cout<<"]";
24         }
25         else assert(requirements.empty());
26         cout<<endl;
27     }
28 };
29
30 #endif // _Lemon_Event
31

```

Deadlock.h: 死锁避免算法（不避免死锁、预防死锁-按优先级释放资源）

```
Job.h x PCB.h x PCBcontrol.h x Process.h x scheduling.h x CPU.h x Event.h x test.cpp x DeadLock.h x
1  #ifndef _Lemon_Dealock
2  #define _Lemon_Dealock
3
4  // no deadlock protection
5  namespace NoProtection{
6      vector<int> apply_for_allocation(int resource_n,vector<int>& own,vector<int>& apply){
7          vector<int> release(resource_n,0);
8          return release;
9      }
10
11      bool check_allocation(int resource_n,vector<int>& requirements,vector<int>& available){
12          return true;
13      }
14  }
15
16  // deadlock protection: prevention
17  // wreck the condition of circular wait: [textbook p109, 3.6.3]
18  // 1. set priority for resources.
19  // 2. If a process applies for resources of lower priority,
20  //    it should release all the resources of higher priority.
21  namespace DeadLockPrevention{
22      vector<int> apply_for_allocation(int resource_n,vector<int>& own,vector<int>& apply){
23          vector<int> release(resource_n);
24          int split=resource_n;
25          for (int i=0;i<resource_n;++i){
26              if (apply[i]!=0){
27                  split=i;
28                  break;
29              }
30          }
31          for (int i=0;i<resource_n;++i){
32              release[i]=(i>=split?0:own[i]);
33          }
34          return release;
35      }
36
37      bool check_allocation(int resource_n,vector<int>& requirements,vector<int>& available){
38          for (int i=0;i<resource_n;++i){
39              if (requirements[i]>available[i]){
40                  return false;
41              }
42          }
43          return true;
44      }
45  }
46
47  #endif // _Lemon_Dealock
48
```

项目 github 仓库:

<https://github.com/Lemon-412/OS-project>

二、 代码实现的简要说明

目前实现的功能

- Process、Job 与 Process Controller Block 的实现
- CPU：基于 PCB*链表的进程控制（进程队列）
- CPU：时钟控制，空闲时 busy waiting
- 中断：支持进程主动中断，进行上下文切换。支持调度中断。
- 调度算法：FCFS（非抢占）、修改的 SJF（抢占）、RR（按时间片）
- 死锁检测：不检测死锁、预防死锁（按优先级释放资源）

自定目标

- 不使用复杂的数据结构，尽可能只是用链表、数组、队列、栈等基本数据结构，最多使用 `std::vector`
- 不使用不可实现的算法（对比用除外），如各类预知未来的算法（银行家，SJF 等）
- 尽可能真实的模拟操作系统的一些方面，从最实际的角度思考问题

代码实现的简要说明

- 设计 Event.h 事件类。包含事件名称，事件起始时间（即进程运行到第几个时间片触发），需求得不到相应的中断时间长度。如果事件为请求系统资源，需要包含请求资源量。
- 设计 JOB.h 作业类。包含作业名，创建时间，作业完成需要的时长，作业运行中产生的事件(Event)，优先级（如果是优先级相关的调度算法需要暂存优先级）
- 设计 Process.h 进程类。包含进程名，创建时间，进程完成需要的时长，进程运行中产生的事件(Event)，进程已经运行的时间，进程阻塞的时长，进程已经获得的资源。其中进程完成需要的时长对 CPU 透明（不预知未来）。
- 设计 PCBControl.h 进程队列类。包含一个 PCB*的链表。封装了在进程队列中插入进程和释放进程的方法。
- 设计 scheduling.h 作业调度算法。根据现有进程 PCB*和就绪进程选取下一个进程。
 - FCFS（非抢占）：如果现有进程非空，则继续选择现有进程，否则选取最早创建的进程
 - 修改的 SJF（抢占）：选取运行时间最少的进程
 - RR：如果现有进程非空，则轮转调度选取下一个进程，否则选取第一个进程（如果有）

- 设计 **DeadLock.h** 死锁避免算法。根据系统资源向量（总量和当前），决策是否允许分配、返回进程的资源释放向量
 - 无保护：进程永远不释放已有资源；永远允许资源分配
 - 预防死锁（按优先级释放资源）：为资源设计优先级，如果进程在拥有高优先级资源的时候申请低优先级资源，则它需要先释放所有更高优先级的资源。
- 设计 **CPU.h** 中央处理器类。包含 **CPU** 的基本信息（资源向量、调度算法逻辑、死锁避免逻辑、作业队列、进程就绪队列、进程阻塞队列、进程完成队列、一个存放当前运行进程地址的寄存器）。在每个时钟信号中：
 - 检查是否有新的作业进入，如果有则为其创建进程和 **PCB**，并插入就绪队列
 - 根据作业调度算法提供的接口，获取下一个运行的进程，如果没有这样的进程（就绪队列为空）则忙等；否则，运行该进程一个时间片，期间：
 - 如果该进程提出了资源分配请求，则根据死锁检测算法提供的接口判断是否分配资源
 - 如果许可分配资源，则先根据死锁检测算法提供的接口释放资源（如有必要），再分配资源并修改相应资源分配信息
 - 如果驳回，则将进程放入阻塞队列
 - 如果进程提出其他主动中断，则将进程放入阻塞队列
 - 如果该时间片正常完成（没有请求或申请资源受到许可），则将其运行时间加一。若进程运行完成，释放相应资源，将其放入完成队列；否则放入就绪队列
 - 检查阻塞队列，将完成阻塞的进程重新放回就绪队列
 - 时钟信号加一，运行下一个时间片

三、 代码运行结果及结果说明

设计测试数据：

- 作业 A：在第 3 个时间片产生，运行 9 个时间片，优先级 0.0
 - 在运行到第 1 个时间片时请求 IO 中断（输入），中断时长为 5 个时间片
 - 在运行到第 2 个时间片时请求资源向量{2,0,1}，如得不到满足中断 2 个时间片
 - 在运行到第 4 个时间片时请求系统资源向量{0,1,0}，如得不到满足中断 2 个时间片
 - 在运行带第 7 个时间片石请求 IO 中断（输出），中断时长为 3 个时间片
- 作业 B：在第 12 个时间片产生，运行 15 个时间片，优先级 0.0
 - 在运行到第 10 个时间片时请求资源向量{0,2,0}，如得不到满足中断 5 个时间片
 - 在运行到第 13 个时间片时请求资源向量{1,0,1}，如得不到满足中断 5 个时间片
- 作业 C：在第 15 个时间片产生，运行 6 个时间片
 - 在运行到第 3 个时间片时请求使用外部设备（打印机），中断时长为 3 个时间片

测试使用 FCFS 调度算法，使用按优先级释放资源的死锁避免方法，系统资源{2,2,5}



```
1  #include <vector>
2  #include "CPU.h"
3
4  int main(){
5      vector<JOB> JOBS={
6          JOB(
7              "Process A",
8              3,9,0.0,
9              {
10                 Event("Input",1,5,{}),
11                 Event("Output",7,3,{}),
12                 Event("Require res",2,2,{2,0,1}),
13                 Event("Require res",4,2,{0,1,0})
14             }
15         ),
16         JOB(
17             "Process B",
18             12,15,0.0,
19             {
20                 Event("Require res",10,5,{0,2,0}),
21                 Event("Require res",13,5,{1,0,1})
22             }
23         ),
24         JOB(
25             "Process C",
26             15,6,0.0,
27             {
28                 Event("Printer",3,3,{})
29             }
30         )
31     };
32     CPU cpu(
33         "First Come First Serve",
34         "Deadlock Prevention",
35         {2,2,5},
36         JOBS
37     );
38     cpu.run();
39     return 0;
40 }
41
```

运行结果（可查看附件中的 gif 动图）:

初始状态:

```
"C:\Users\Lemon\Desktop\OS2\Experiments\exp1-2\OS Project\OS2\Main\bin\Debug\Main.exe"
Inner Time:0
===== Process Ready =====
NAME
CPUTIME
ALLTIME
PRIORITY
STATE
===== Process Stuck =====
NAME
CPUTIME
ALLTIME
PRIORITY
STATE
REMAIN
===== Process Done =====
NAME
CPUTIME
ALLTIME
PRIORITY
STATE
INTIME
OUTTIME
===== System Resources =====
SYSTEM      2    2    5
请按任意键继续. . .
```

进程 A 进入并运行:

```
"C:\Users\Lemon\Desktop\OS2\Experiments\exp1-2\OS Project\OS2\Main\bin\Debug\Main.exe"
Inner Time:3
===== Process Ready =====
NAME      Process A
CPUTIME   0
ALLTIME   9
PRIORITY  0.000
STATE     Running
===== Process Stuck =====
NAME
CPUTIME
ALLTIME
PRIORITY
STATE
REMAIN
===== Process Done =====
NAME
CPUTIME
ALLTIME
PRIORITY
STATE
INTIME
OUTTIME
===== System Resources =====
SYSTEM      2    2    5
Process A   0    0    0
*****
Process A: Input, 5
请按任意键继续. . . ■
```

进程 A 阻塞:

```
"C:\Users\Lemon\Desktop\OS2\Experiments\exp1-2\OS Project\OS2\Main\bin\Debug\Main.exe"
Inner Time:5
===== Process Ready =====
NAME
CPUTIME
ALLTIME
PRIORITY
STATE
===== Process Stuck =====
NAME      Process A
CPUTIME    1
ALLTIME    9
PRIORITY   0.000
STATE      Input
REMAIN     3
===== Process Done =====
NAME
CPUTIME
ALLTIME
PRIORITY
STATE
INTIME
OUTTIME
===== System Resources =====
SYSTEM      2   2   5
Process A   0   0   0
请按任意键继续. . .
```

进程 A 阻塞完成，运行并申请资源:

```
"C:\Users\Lemon\Desktop\OS2\Experiments\exp1-2\OS Project\OS2\Main\bin\Debug\Main.exe"
Inner Time:8
===== Process Ready =====
NAME      Process A
CPUTIME    1
ALLTIME    9
PRIORITY   0.000
STATE      Running
===== Process Stuck =====
NAME
CPUTIME
ALLTIME
PRIORITY
STATE
REMAIN
===== Process Done =====
NAME
CPUTIME
ALLTIME
PRIORITY
STATE
INTIME
OUTTIME
===== System Resources =====
SYSTEM      2   2   5
Process A   0   0   0
*****
Process A: Require res, 2 [ 2 0 1 ]
请按任意键继续. . . ■
```

系统许可申请资源:

```
"C:\Users\Lemon\Desktop\OS2\Experiments\exp1-2\OS Project\OS2\Main\bin\Debug\Main.exe"
Inner Time:9
===== Process Ready =====
NAME      Process A
CPUTIME    2
ALLTIME    9
PRIORITY   0.000
STATE      Running
===== Process Stuck =====
NAME
CPUTIME
ALLTIME
PRIORITY
STATE
REMAIN
===== Process Done =====
NAME
CPUTIME
ALLTIME
PRIORITY
STATE
INTIME
OUTTIME
===== System Resources =====
SYSTEM      0  2  4
Process A   2  0  1
请按任意键继续. . .
```

进程 B 进入，由于 FCFS 非抢占 B 为就绪状态

```
"C:\Users\Lemon\Desktop\OS2\Experiments\exp1-2\OS Project\OS2\Main\bin\Debug\Main.exe"
Inner Time:12
===== Process Ready =====
NAME      Process A  Process B
CPUTIME    5         0
ALLTIME    9         15
PRIORITY   0.000     0.000
STATE      Running  ready
===== Process Stuck =====
NAME
CPUTIME
ALLTIME
PRIORITY
STATE
REMAIN
===== Process Done =====
NAME
CPUTIME
ALLTIME
PRIORITY
STATE
INTIME
OUTTIME
===== System Resources =====
SYSTEM      0  1  4
Process A   2  1  1
Process B    0  0  0
请按任意键继续. . .
```

进程 A 申请 IO，进程 B 运行

```
"C:\Users\Lemon\Desktop\OS2\Experiments\exp1-2\OS Project\OS2\Main\bin\Debug\Main.exe"
Inner Time:14
===== Process Ready =====
NAME      Process B
CPUTIME    0
ALLTIME    15
PRIORITY   0.000
STATE      Running
===== Process Stuck =====
NAME      Process A
CPUTIME    7
ALLTIME    9
PRIORITY   0.000
STATE      Output
REMAIN     2
===== Process Done =====
NAME
CPUTIME
ALLTIME
PRIORITY
STATE
INTIME
OUTTIME
===== System Resources =====
SYSTEM      0   1   4
Process B   0   0   0
Process A   2   1   1
请按任意键继续. . .
```

进程 C 进入，进程 B 申请资源：

```
"C:\Users\Lemon\Desktop\OS2\Experiments\exp1-2\OS Project\OS2\Main\bin\Debug\Main.exe"
Inner Time:23
===== Process Ready =====
NAME      Process B   Process C   Process A
CPUTIME    9           0           7
ALLTIME    15          6           9
PRIORITY   0.000        0.000       0.000
STATE      Running     ready       Ready
===== Process Stuck =====
NAME
CPUTIME
ALLTIME
PRIORITY
STATE
REMAIN
===== Process Done =====
NAME
CPUTIME
ALLTIME
PRIORITY
STATE
INTIME
OUTTIME
===== System Resources =====
SYSTEM      0   1   4
Process B   0   0   0
Process C   0   0   0
Process A   2   1   1
*****
Process B: Require res, 5 [ 0 2 0 ]
请按任意键继续. . .
```

进程 B 申请资源被驳回进入阻塞，由于 A 进程进入时间早，进程 A 运行（FCFS）：

```
"C:\Users\Lemon\Desktop\OS2\Experiments\exp1-2\OS Project\OS2\Main\bin\Debug\Main.exe"
Inner Time:24
===== Process Ready =====
NAME      Process C   Process A
CPUTIME    0           7
ALLTIME    6           9
PRIORITY   0.000       0.000
STATE      ready      Running
===== Process Stuck =====
NAME      Process B
CPUTIME    9
ALLTIME    15
PRIORITY   0.000
STATE      Require res
REMAIN     4
===== Process Done =====
NAME
CPUTIME
ALLTIME
PRIORITY
STATE
INTIME
OUTTIME
===== System Resources =====
SYSTEM      0   1   4
Process C   0   0   0
Process A   2   1   1
Process B   0   0   0
请按任意键继续. . .
```

进程 A 完成，释放资源，进程 C 运行：

```
"C:\Users\Lemon\Desktop\OS2\Experiments\exp1-2\OS Project\OS2\Main\bin\Debug\Main.exe"
Inner Time:26
===== Process Ready =====
NAME      Process C
CPUTIME    0
ALLTIME    6
PRIORITY   0.000
STATE      Running
===== Process Stuck =====
NAME      Process B
CPUTIME    9
ALLTIME    15
PRIORITY   0.000
STATE      Require res
REMAIN     2
===== Process Done =====
NAME      Process A
CPUTIME    9
ALLTIME    9
PRIORITY   0.000
STATE      Finished
INTIME     3
OUTTIME    25
===== System Resources =====
SYSTEM      2   2   5
Process C   0   0   0
Process B   0   0   0
Process A   0   0   0
请按任意键继续. . . ■
```

进程 B 阻塞完成，进程 C 请求使用外部设备阻塞（打印机），进程 B 运行并请求资源：

```
"C:\Users\Lemon\Desktop\OS2\Experiments\exp1-2\OS Project\OS2\Main\bin\Debug\Main.exe"
Inner Time:29
===== Process Ready =====
NAME      Process B
CPUTIME    9
ALLTIME    15
PRIORITY   0.000
STATE      Running
===== Process Stuck =====
NAME      Process C
CPUTIME    3
ALLTIME    6
PRIORITY   0.000
STATE      Printer
REMAIN     2
===== Process Done =====
NAME      Process A
CPUTIME    9
ALLTIME    9
PRIORITY   0.000
STATE      Finished
INTIME     3
OUTTIME    25
===== System Resources =====
SYSTEM      2  2  5
Process B   0  0  0
Process C   0  0  0
Process A   0  0  0
*****
Process B: Require res, 5 [ 0 2 0 ]
请按任意键继续. . .
```

系统许可资源申请吗，进程 B 第二次申请资源：

```
"C:\Users\Lemon\Desktop\OS2\Experiments\exp1-2\OS Project\OS2\Main\bin\Debug\Main.exe"
Inner Time:32
===== Process Ready =====
NAME      Process B   Process C
CPUTIME    12         3
ALLTIME    15         6
PRIORITY   0.000     0.000
STATE      Running    Ready
===== Process Stuck =====
NAME
CPUTIME
ALLTIME
PRIORITY
STATE
REMAIN
===== Process Done =====
NAME      Process A
CPUTIME    9
ALLTIME    9
PRIORITY   0.000
STATE      Finished
INTIME     3
OUTTIME    25
===== System Resources =====
SYSTEM      2  0  5
Process B   0  2  0
Process C   0  0  0
Process A   0  0  0
*****
Process B: Require res, 5 [ 1 0 1 ]
请按任意键继续. . .
```

系统许可（先释放{0,2,0}再申请{1,2,1}）:

```
"C:\Users\Lemon\Desktop\OS2\Experiments\exp1-2\OS Project\OS2\Main\bin\Debug\Main.exe"
Inner Time:33
===== Process Ready =====
NAME      Process B    Process C
CPUTIME   13           3
ALLTIME   15           6
PRIORITY  0.000        0.000
STATE     Running     Ready
===== Process Stuck =====
NAME
CPUTIME
ALLTIME
PRIORITY
STATE
REMAIN
===== Process Done =====
NAME      Process A
CPUTIME   9
ALLTIME   9
PRIORITY  0.000
STATE     Finished
INTIME    3
OUTTIME   25
===== System Resources =====
SYSTEM    1    0    4
Process B 1    2    1
Process C 0    0    0
Process A 0    0    0
请按任意键继续. . .
```

进程 ABC 陆续完成:

```
"C:\Users\Lemon\Desktop\OS2\Experiments\exp1-2\OS Project\OS2\Main\bin\Debug\Main.exe"
Inner Time:38
===== Process Ready =====
NAME
CPUTIME
ALLTIME
PRIORITY
STATE
===== Process Stuck =====
NAME
CPUTIME
ALLTIME
PRIORITY
STATE
REMAIN
===== Process Done =====
NAME      Process A    Process B    Process C
CPUTIME   9           15          6
ALLTIME   9           15          6
PRIORITY  0.000       0.000       0.000
STATE     Finished   Finished   Finished
INTIME    3           12          15
OUTTIME   25          34          37
===== System Resources =====
SYSTEM    2    2    5
Process A 0    0    0
Process B 0    0    0
Process C 0    0    0
请按任意键继续. . . ■
```