电子科技大学 计算机科学与工程学院

实验指导书

（实验）课程名称： 计算机操作系统

电子科技大学教务处制表

进程与资源管理器设计

- 实验所属系列：操作系统课程实验
- 实验对象：本科
- 相关课程及专业：计算机操作系统，计算机专业
- 实验类型：配套上机
- 实验学时数： 6 学时

1. 实验目的

设计和实现进程与资源管理，并完成 Test shell 的编写，以建立系统的进程管理、调度、资源管理和分配的知识体系，从而加深对操作系统进程调度和资源管理功能的宏观理解和微观实现技术的掌握。

2. 实验内容

在实验室提供的软硬件环境中，设计并实现一个基本的进程与资源管理器。该管理器能够完成进程的控制，如进程创建与撤销、进程的状态转换；能够基于优先级调度算法完成进程的调度，模拟时钟中断，在同优先级进程中采用时间片轮转调度算法进行调度；能够完成资源的分配与释放，并完成进程之间的同步。该管理器同时也能完成从用户终端或者指定文件读取用户命令，通过 Test shell 模块完成对用户命令的解释，将用户命令转化为对进程与资源控制的具体操作，并将执行结果输出到终端或指定文件中。

3. 实验环境

设计平台和语言不限，平台可为 Linux 或 Windows, 语言可为 C，C++，Java, python 等。

4. 实验要求

要求学生熟悉掌握计算机操作系统进程管理和资源管理的基本原理和关键技术，包含进程控制、进程调度、进程同步及资源管理，在此基础上设计和实现进程和资源管理器。具体步骤包括：了解功能需求、完成系统总统设计、相关的数据结构定义和具体函数设计。

同时要求设计与实现驱动程序（test shell）：驱动该管理器工作，即将命令语言（即用户要求）转换成对与内核函数（如 create, request 等）的调用。

5. 实验过程

5.1 总体设计

系统总体架构如图 1 所示，最右边部分为进程与资源管理器，属于操作系统内核的功能。该管理器具有如下功能：完成进程创建、撤销和进程调度；完成多单元 (multi_unit)资源的管理；完成资源的申请和释放；完成错误检测和定时器中断功能。

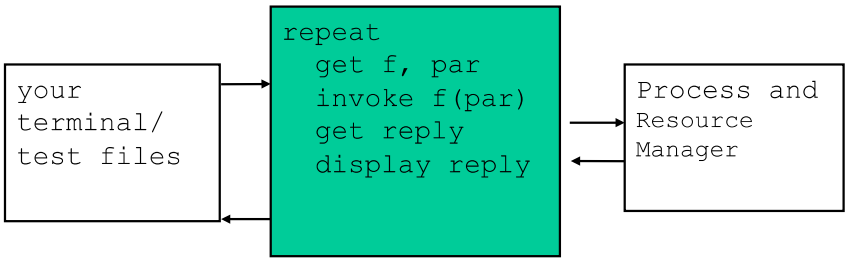

图 1 系统总体结构

图 1 中间绿色部分为驱动程序 test shell，设计与实现 test shell，该 test shell 将调度所设计的进程与资源管理器来完成测试。Test shell 的应具有的功能：

- 从终端或者测试文件读取命令；
- 将用户需求转换成调度内核函数（即调度进程和资源管理器）；
- 在终端或输出文件中显示结果：如当前运行的进程、错误信息等。

图 1 最左端部分为：通过终端（如键盘输入）或者测试文件来给出相应的用户命令，以及模拟硬件引起的中断

5.2 Test shell 设计

Test_shell 的功能如 4.1 所述，代码示例如图 1 中绿色部分。

Test shell 要求完成的命令（Mandatory Commands）

-init

-cr <name> <priority>(=1 or 2) // create process

-de <name> // delete process

-req <resource name> <# of units> // request resource

-rel <resource name> <# of units> // release resource

-to // time out

查看进程状态和资源状态的命令

-list ready //list all processes in the ready queue

-list block // list all processes in the block queue

-list res //list all available resources

可选实现命令

-pr <name> //print pcb information about a given process.

Test shell 输出示例：shell> 后为输入命令，*后为输出内容

* Process init is running //the running process create new process

*....

shell> cr A 1 // this command is from terminal and indicates creating process A with priority 1.Test shell will invoke kernel function: create

* Process A is running //display

shell> cr B 2 // A create process B with priority 2

* Process B is running

shell> cr C 2 // B create process C with priority 2, C is at the end of ready list

* Process B is running //Because C's priority is same as B's

shell>to //time out invokes process switch

* Process C is running, Process B is ready

shell>list ready //list all ready processes with different priorities

* 2

* 1: A-C-B

* 0: init

shell> list res //list all available resources

* R1 1

* R2 2

* R3 3

* R4 4

shell>list block //list all block processes for different resources

* R1 Q-P // Q and P are waiting for R1

* R3 A // A is waiting for R3

5.3 进程管理设计

5.3.1 进程状态与操作

进程状态： ready/running/blocked

进程操作：

- 创建(create)： (none) -> ready
- 撤销(destroy)： running/ready/blocked -> (none)
- 请求资源(Request): running -> blocked (当资源没有时，进程阻塞)
- 释放资源(Release): blocked -> ready (因申请资源而阻塞的进程被唤醒)
- 时钟中断(Time_out): running -> ready
- 调度：ready -> running / running ->ready

5.3.2 进程控制块结构（PCB）

- **PID（name）**
- CPU state — not used
- Memory — not used
- Open_Files — not used
- **Other_resources //: resource which is occupied**
- **Status: Type & List// type: ready, block, running…., //List: RL(Ready list) or BL(block list)**
- **Creation_tree: Parent/Children**
- **Priority: 0, 1, 2 (Init, User, System)**

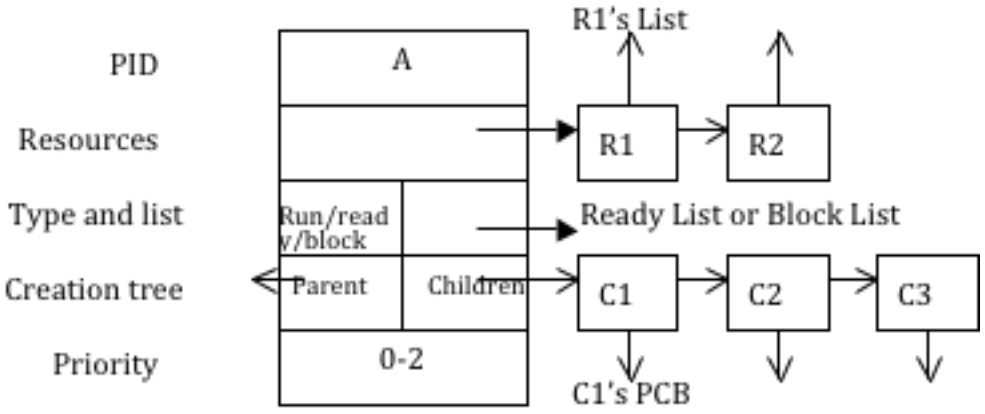

图 2 PCB 结构示意

就绪进程队列：Ready list (RL)

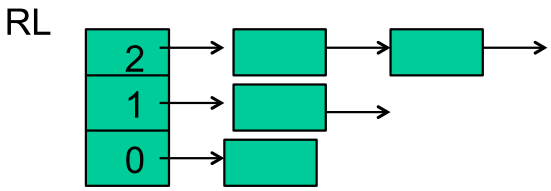

图 3 Ready list 数据结构

3 个级别的优先级，且优先级固定无变化

2 = “system”

1 = “user”

0 = “init”

每个 PCB 要么在 RL 中，要么在 block list 中 。当前正在运行的进程，根据优先级，可以将其放在 RL 中相应优先级队列的首部。

5.3.3 主要函数

- **创建进程：**

Create(initialization parameters)// initialization parameters 可以为进程的

ID 和优先级，优先级：初始进程 0、用户进程 1 和系统进程 2。

{

create PCB data structure

initialize PCB using parameters //包括进程的 ID，优先级、状态等

link PCB to creation tree /*连接父亲节点和兄弟节点，当前进程为父亲节点，父亲节点中的子节点为兄弟节点*/

insert(RL, PCB)//插入就绪相应优先级队列的尾部

Scheduler()

}

Init 进程在启动时创建，可以用来创建第一个系统进程或者用户进程。新创建的进程或者被唤醒的进程被插入到就绪队列（RL）的末尾。

示例：

图 4 中，虚线表示进程 A 为运行进程，在进程 A 运行过程中，创建用户进程 B：cr B 1，数据结构间关系图 4 所示

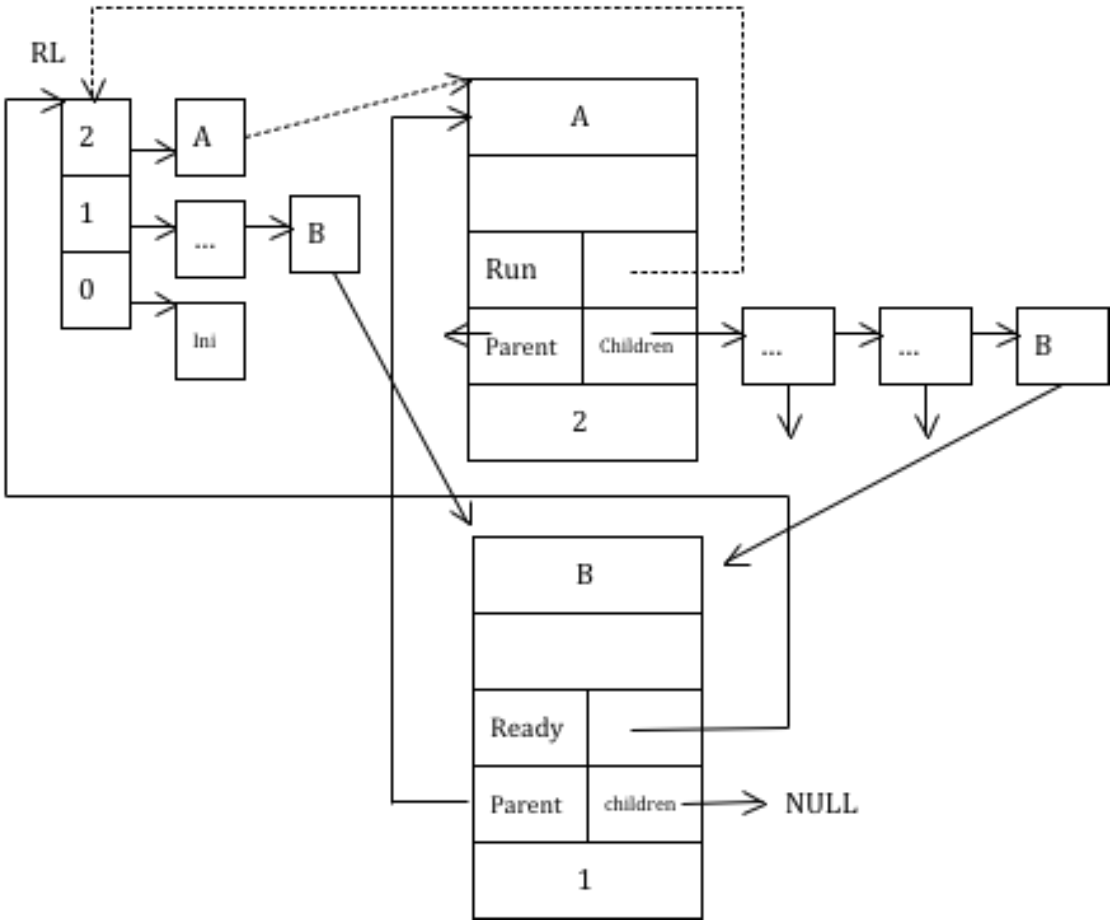

图 4 进程数据结构间关系

(为了简单起见，A 和 B 分别指向 RL 的链接可以不要)

- 撤销进程

```
Destroy (pid)
{
    get pointer p to PCB using pid
    Kill_Tree(p)
    Scheduler() //调度其他进程执行
}
Kill_Tree(p)
{
```

for all child processes q Kill_Tree(q) //嵌套调用，撤销所有子孙进程

free resources //和 release 调用类似的功能

delete PCB and update all pointers

}

Process can be destroyed by any of its ancestors or by itself (exit)

5.4 资源管理设计

5.4.1 主要数据结构

资源的表示：设置固定的资源数量，4 类资源，R1，R2，R3，R4，每类资源 Ri 有 i 个

资源控制块 Resource control block (RCB) 如图 5 所示

- RID: 资源的 ID
- Status: 空闲单元的数量
- Waiting_List: list of blocked process

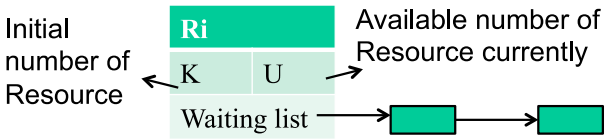

图 5 资源数据结构 RCB

5.4.2 请求资源

所有的资源申请请求按照 FIFO 的顺序进行

- **情况一：**当一类资源数量本身只有一个的情况

```
Request(rid)
{
    r = Get_RCB(rid);
    if (r->Status == 'free') //只有一个资源时可以用 free 和 allocated 来表示资源状态
    {
        r->Status = 'allocated‘;
        insert(self->Other_Resources, r);//self 为当前请求资源的进程 PCB，insert 以后 r 为 self 进程占有的资源, 参 PCB 结构图
    }
    else
    {
       self->Status.Type = 'blocked';
       self->Status.List = r;//point to block list, self is blocked by r
       remove(RL, self);// remove self from the ready list(self can be put at the head of RL when it is running).
       insert(r->Waiting_List, self);// 将进程 self 插入到资源 r 的等待队列尾部
    Scheduler();
    }
}
```

- 情况二：一类资源有多个的情况（multi_unit）

```
Request(rid, n) // n 为请求资源数量
{
    r = Get_RCB(rid);
    if (u ≥ n) // u 为 r->Status.u, 即可用资源数量，参资源数据结构图
    {
        u-n;
        insert(self->Other_Resources, r, n);//self 为当前请求资源的进程 PCB，insert 以后指明 n 个 r 为 self 进程占有的资源
    }
    else
    {
        if (n>k) exit;//k 为资源 r 的总数，申请量超过总数时，将打印错误信息并退出
        self->Status.Type = 'blocked'; //只要 u<n, 就不分配，进程阻塞
        self->Status.List = r;//point to block list，此处表明进程因 r 阻塞
        remove(RL, self);// remove self from the ready list，因为运行进程位于绪队列首部，所以此时将它从就绪队列移除
        insert(r->Waiting_List, self);// 将进程 self 插入到资源 r 的等待队列尾部
        Scheduler();
```

```
    }
}
```

5.4.3 释放资源

- 情况一：一类资源只有 1 个的情况

```
Release(rid)
{
    r = Get_RCB(rid);
    remove(self->Other_Resources, r);//将 r 从进程 self 占用的资源中移走
    if (r->Waiting_List == NIL) //没有进程在等待资源 r
    {
       r->Status = 'free';
    }
    else
    {
       remove(r->Waiting_List, q);//q 为 waiting_list 中第一个阻塞进程
       q->Status.Type = 'ready';
       q->Status.List = RL;//就绪队列
       insert(q->Other_Resources, r);
       insert(RL, q); //q 插入就绪队列中相应优先级队列的末尾
       Scheduler();
```

```
        }
}
```

- 情况二：一类资源有多个的情况

```
Release(rid,n) //rid 为资源 ID，n 为释放的资源数量
{
        r = Get_RCB(rid);

/*remove r from self->other_resources, and u= u + n，将资源 r 从当前进程占用的资源列表里移除，并且资源 r 的可用数量从 u 变为 u+n*/
        remove(self->Other_Resources, r, n);

/*如果阻塞队列不为空，且阻塞队列首部进程需求的资源数 req 小于等于可用资源数量 u，则唤醒这个阻塞进程，放入就绪队列*/
        while (r->Waiting_List != NIL && u>=req_num)
        {
                u=u- req_num; //可用资源数量减少
                remove(r->Waiting_List, q); //  从资源 r 的阻塞队列中移除
                q->Status.Type = 'ready';
                q->Status.List = RL;
                insert(q->Other_Resources, r); //插入 r 到 q 所占用的资源中
                insert(RL, q); //  插入 q 到就绪队列
```

}

Scheduler(); //基于优先级的抢占式调度策略，因此当有进程获得资源时，需要查看当前的优先级情况并进行调度

}

5.5 进程调度与时钟中断设计

调度策略

- 基于 3 个优先级别的调度：2，1，0
- 使用基于优先级的抢占式调度策略，在同一优先级内使用时间片轮转（RR）
- 基于函数调用来模拟时间共享
- 初始进程(Init process)具有双重作用：

  虚设的进程：具有最低的优先级，永远不会被阻塞

  进程树的根

- Scheduler:

Called at the end of every kernel call

(1) Scheduler() {

(2) find highest priority process p

(3) if (self->priority < p->priority ||

(4) self->Status.Type != 'running' ||

(5) self == NIL)

(6) preempt(p, self)//在条件(3)(4)(5)下抢占当前进程

}

Condition (3): called from create or release, 即新创建进程的优先级或资源释放后唤醒进程的优先级高于当前进程优先级

Condition (4): called from request or time-out， 即请求资源使得当前运行进程阻塞或者时钟中断使得当前运行进程变成就绪

Condition (5): called from destroy，进程销毁

Preemption: //抢占，将 P 变为执行，输出当前运行进程的名称

- Change status of p to running (status of self already changed to ready/blocked)
- Context switch—output name of running process

● 时钟中断（Time out）：模拟时间片到或者外部硬件中断

Time_out()

{

find running process q; //当前运行进程 q

remove(RL, q);// remove from head? yes

q->Status.Type = 'ready';

insert(RL, q);// insert into tail? yes

```
    Scheduler();
}
```

5.6 系统初始化设计

启动时初始化管理器：

具有 3 个优先级的就绪队列 RL 初始化；

Init 进程；

4 类资源，R1，R2，R3，R4，每类资源 Ri 有 i 个

5.7 测试示例

终端输入测试命令，或将测试命令放在测试文件 input.txt 中，内容为图 6(a)所示.

对应的输出命令可以在屏幕上显示，也可以放入输出文件 output.txt 中，内容如图 6(b)所示。

```
init
cr x 1
cr p 1
cr q 1
cr r 1
list ready


to
req R2 1
to
req R3 3
to
req R4 3
list res



to
to
req R3 1
req R4 2
req R2 2
list block



to
de q
to
to
cr w 2
```

（a）输入

```
init process is running
process x is running
process x is running
process x is running
process x is running
2:
1:p-q-r
0:init
process p is running. process x is ready
process p requests 1 R2
process q is running. process p is ready
process q requests 3 R3
process r is running, process q is ready
process r requests 3 R4
R1 1
R2 1
R3 0
R4 1
process x is running. process r is ready
process p is running. process x is ready
process q is running. process p is blocked.
process r is running. process q is blocked.
process x is running. process r is blocked.
R1
R2 r
R3 p
R4 q
process x is running.
release R3. wake up process p
process p is running
process x is running
process w is running
```

（b）输出

图 6 输入与输出示例

6. 实验报告

要求按小组(1~3 人)完成一个实验，但是小组内分工，组员各自

有各自模块，按个人提交实验报告，组间及组内报告都不能雷同，报告应包括：

1） 实验原理

2） 实验目的

3） 实验内容

4） 实验环境

5） 实验步骤

a) 系统功能需求分析；

b) 总体框架设计：说明具体原理，用到的方法或者算法，模块之间的调用关系，总体工作流程

c) 具体模块的设计

对于组内其他人的模块，给出设计的思路和流程，理清模块间关系；对于自己模块的设计，需要给出详细的设计思想、设计流程和相应代码；

d) 测试：给出测试代码和测试结果，并作结果分析

6） 实验总结