



第2章 进程与线程

2.6 进程控制管理和组织





2.6 进程控制管理和组织

- 进程控制的职能是对系统中的所有进程实施有效的管理。
- 常见的进程控制功能有进程创建、终止、阻塞与唤醒等。
- 这些功能一般由操作系统内核原语来实现。





原语

- **原语**是由若干条机器指令构成的，用以完成特定功能的一段程序，这段程序在执行期间不可分割。

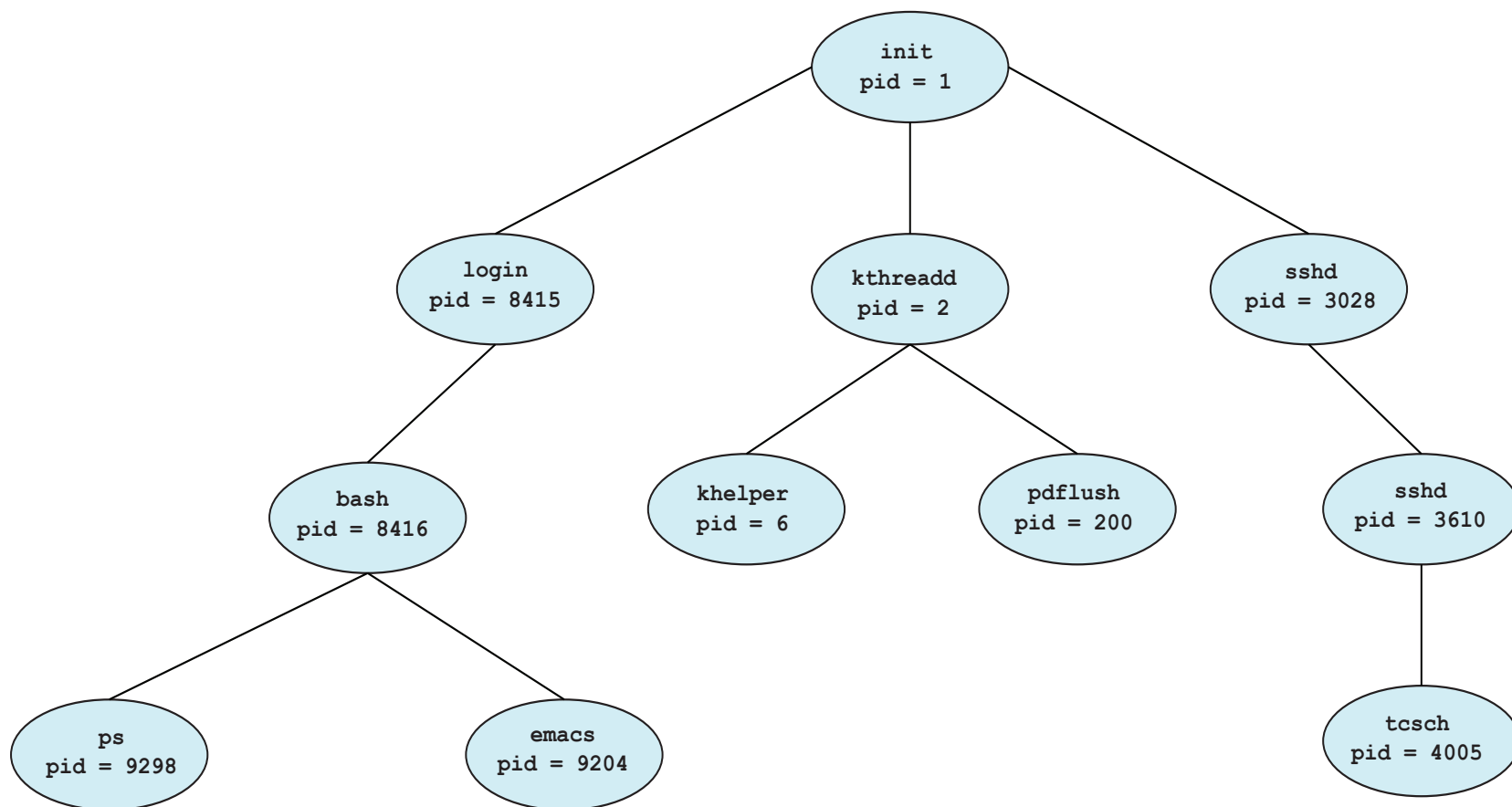


2.6.1 进程创建

- 为描述进程之间的创建关系，引入了进程树。
- **进程树** 又称进程图或进程家族树，是描述进程家族关系的一棵有向树。
- 进程树中的结点表示进程，若进程**A**创建了进程**B**，则从结点**A**有一条边指向结点**B**，说明进程**A**是进程**B**的父进程，进程**B**是进程**A**的子进程。



进程树例子





进程创建原语

■ 导致进程创建的原因有：


- 用户登录：用户登录后，若合法则为用户创建一个进程。
- 作业调度：为调度到的作业分配资源并创建进程。
- OS服务：创建服务进程。
- 应用需要：应用程序根据需要创建子进程。



创建原语的主要功能

- 进程创建原语的功能是创建一个新进程，其主要操作过程如下：
 - ① 向系统申请一个空闲PCB。
 - ② 为新进程分配资源。如分配内存空间。
 - ③ 初始化新进程的PCB。在其PCB中填入进程名、家族信息、程序和数据地址、进程优先级、资源清单及进程状态等。
 - ④ 将新进程的PCB插入就绪队列。





进程创建中的问题

1. 父子进程执行方式

- 父进程和子进程并发执行
- 父进程等待，直到某个子进程或全部子进程执行完毕

2. 父子资源共享原则

- 共享全部资源
- 子进程共享父进程的部分资源
- 父子进程不共享资源





进程创建

3. 地址空间

- 子进程复制父进程空间内容（复制品）
- 子进程装入另一个程序运行

4. Linux例子

- **fork()**: 创造的子进程是父进程的完整副本，复制了父亲进程的所有资源
- **exec():fork()**函数会调用此系统调用，使用指定的新程序填充进程空间
- **vfork()**: 创建的子进程与父进程共享数据段,而且由**vfork()**创建的子进程将先于父进程运行
- **clone()**: 允许子进程有选择的共享父进程资源





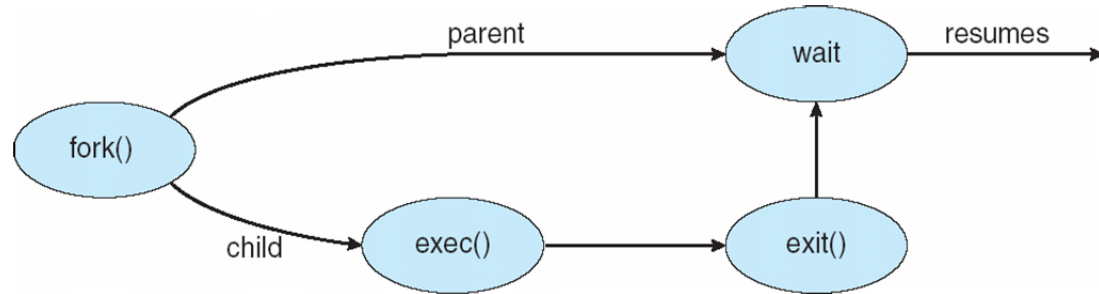
创建进程的例子1

```
void main( ) {  
    int i, p_id;  
    if ( (p_id = fork( )) == 0 ) {  
        /* 子进程程序 */  
        for (i = 1; i < 3; i ++ )  
            printf("This is child process\n");  
    } else if (p_id < 0){  
        printf("fork new process error!\n");  
        exit(-1);  
    } else {  
        /* 父进程程序*/  
        /*p_id 为新创建子进程的进程ID*/  
        for ( i = 1; i < 3; i ++ )  
            printf("This is parent process\n");  
    }  
}
```



创建进程的例子2

```
void main( ) {  
    pid_t pid;  
    pid = fork();  
    if (pid < 0){  
        fprintf(".....");  
        exit(-1);  
    }  
    else if (pid == 0){  
        execlp("/bin/ls", "ls", NULL);  
    }  
    else{  
        wait(NULL);  
        printf("Child Complete");  
        exit(0);  
    }  
}
```





2.6.2 进程终止

- 引起进程终止的原因有：

- 正常结束

- 异常结束：超时、内存不足、地址越界、算术错、I/O故障、非法指令等。

- 外界干预：包括操作员或系统干预，父进程请求。





终止原语采用的两种策略

- 终止原语采用的两种策略：
 - 终止指定标识符的进程
 - 终止指定进程及其所有子孙进程
- 后一种终止策略的功能描述
 - 从系统的PCB表中找到被终止进程的PCB。
 - 检查被终止进程的状态是否为执行状态，若是则立即停止该进程的执行，设置重新调度标志。
 - 检查被终止进程是否有子孙进程，若有子孙进程还应终止该进程的子孙进程。
 - 回收该进程占有的全部资源并回收其PCB。



进程终止的方法

■ 方法一：正常退出时

- 进程执行到最后一条语句，然后申请OS删除自己，**exit()**


- 进程返回状态值给父进程，父进程通过**wait()**获知
- 进程资源被OS回收

■ 方法二：非正常退出时

- 父进程可终止子进程，**abort()**，当：

- 子进程使用了超过的分配资源
- 分配给子进程的任务不再需要继续执行
- 父进程正在退出，且不允许无父进程的子进程继续执行，级联终止





进程终止的方法

- 父进程可通过**wait()**，等待子进程的结束
`pid = wait(&status);`
- 当子进程已退出，且父进程没有调用**wait()**，则子进程成为僵尸进程**zombie process**
- 如果子进程继续执行，而父进程已经终止，则子进程成为孤儿进程**orphan process**



2.6.3 进程阻塞与唤醒

■ 引起进程阻塞及唤醒的事件：

- 请求系统服务。如请求分配打印机，但无空闲打印机则进程阻塞；当打印机重又空闲时应唤醒进程。
- 启动某种操作并等待操作完成。如启动I/O操作，进程阻塞；I/O完成则唤醒进程。
- 等待合作进程的协同配合。如计算进程尚未将数据送到缓冲区，则打印进程阻塞；当缓冲区中有数据时应唤醒进程。
- 系统进程无新工作可做。如没有信息可供发送，则发送请求阻塞；当收到新的发送请求时，应将阻塞进程唤醒。





阻塞原语的主要功能

- 阻塞原语的主要功能是将进程由执行状态转为阻塞状态。其主要操作过程如下：
 - ① 停止当前进程的执行；
 - ② 保存该进程的CPU现场信息；
 - ③ 将进程状态改为阻塞，并插入到相应事件的等待队列中；
 - ④ 转进程调度程序，从就绪队列中选择一个新的进程投入运行。





唤醒原语的主要功能

- 当进程等待的事件发生时，由发现者进程将其唤醒。
- 唤醒原语的主要功能是将进程唤醒，其主要操作过程如下：
 - ① 将被唤醒进程从相应的等待队列中移出；
 - ② 将进程状态改为就绪，并将该进程插入就绪队列；
 - ③ 转进程调度或返回。





阻塞与唤醒的关系

- 一个进程由执行状态转变为阻塞状态，是这个进程**自己调用阻塞原语**去完成的。
- 进程由阻塞状态转变为就绪状态，是另一个发现者进程**调用唤醒原语实现**的。
- 一般发现者进程与被唤醒进程是**合作的并发进程**。





2.6.4 进程的挂起与激活

- 挂起原语和激活原语都有多种实现方式如：
 - 把发出挂起原语的进程自身挂起
 - 挂起具有指定标识符的进程
 - 把某进程及其子孙进程挂起
 - 激活一个具有指定标识名的进程
 - 激活某进程及其子孙进程
- 下面以挂起或激活具有指定标识符的进程为例，说明这两种原语的主要功能。



挂起原语的主要功能

- 挂起原语的主要功能是将指定进程挂起，算法思想如下：
 - ① 到PCB表中查找该进程的PCB；
 - ② 检查该进程的状态，若为执行则停止执行并保护CPU现场信息，将该进程状态改为挂起就绪；
 - ③ 若为活动阻塞，则将该进程状态改为挂起阻塞；
 - ④ 若为活动就绪，则将该进程状态改为挂起就绪；
 - ⑤ 若进程挂起前为执行状态，则转进程调度，从就绪队列中选择一个进程投入运行。



激活原语的主要功能

- 激活原语的主要功能是将指定进程激活。其算法思想如下：

- ① 到PCB表中查找该进程的PCB。
- ② 检查该进程的状态。若状态为挂起阻塞，则将该进程状态改为活动阻塞。
- ③ 若状态为挂起就绪，则将该进程状态改为活动就绪。
- ④ 若进程激活后为活动就绪状态，可能需要转进程调度。





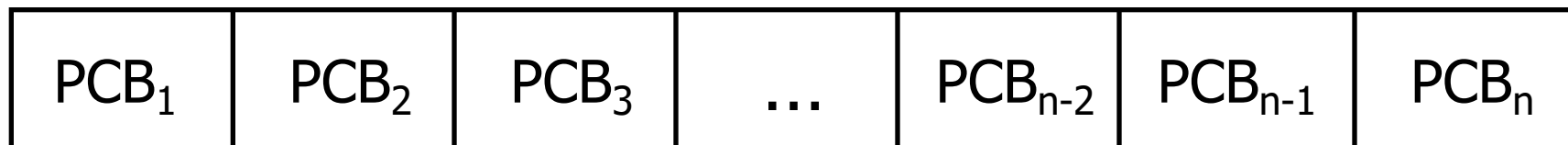
2.6.5 进程的组织

- 系统中有许多进程，为了能对它们进行有效的管理，应将PCB组织起来。
- 常用的组织方式有：
 - 线性方式
 - 链表方式
 - 索引方式



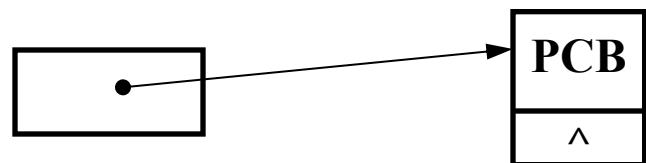
线性方式

- 线性方式：将PCB顺序存放在一片连续内存中。

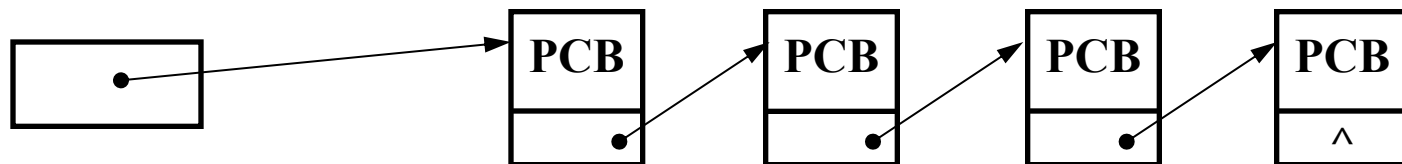


链接方式

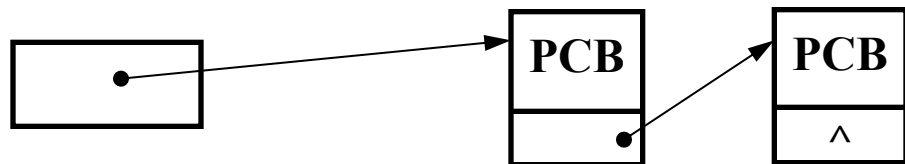
- 链接方式：将同一状态的PCB组成一个链表。



运行指针



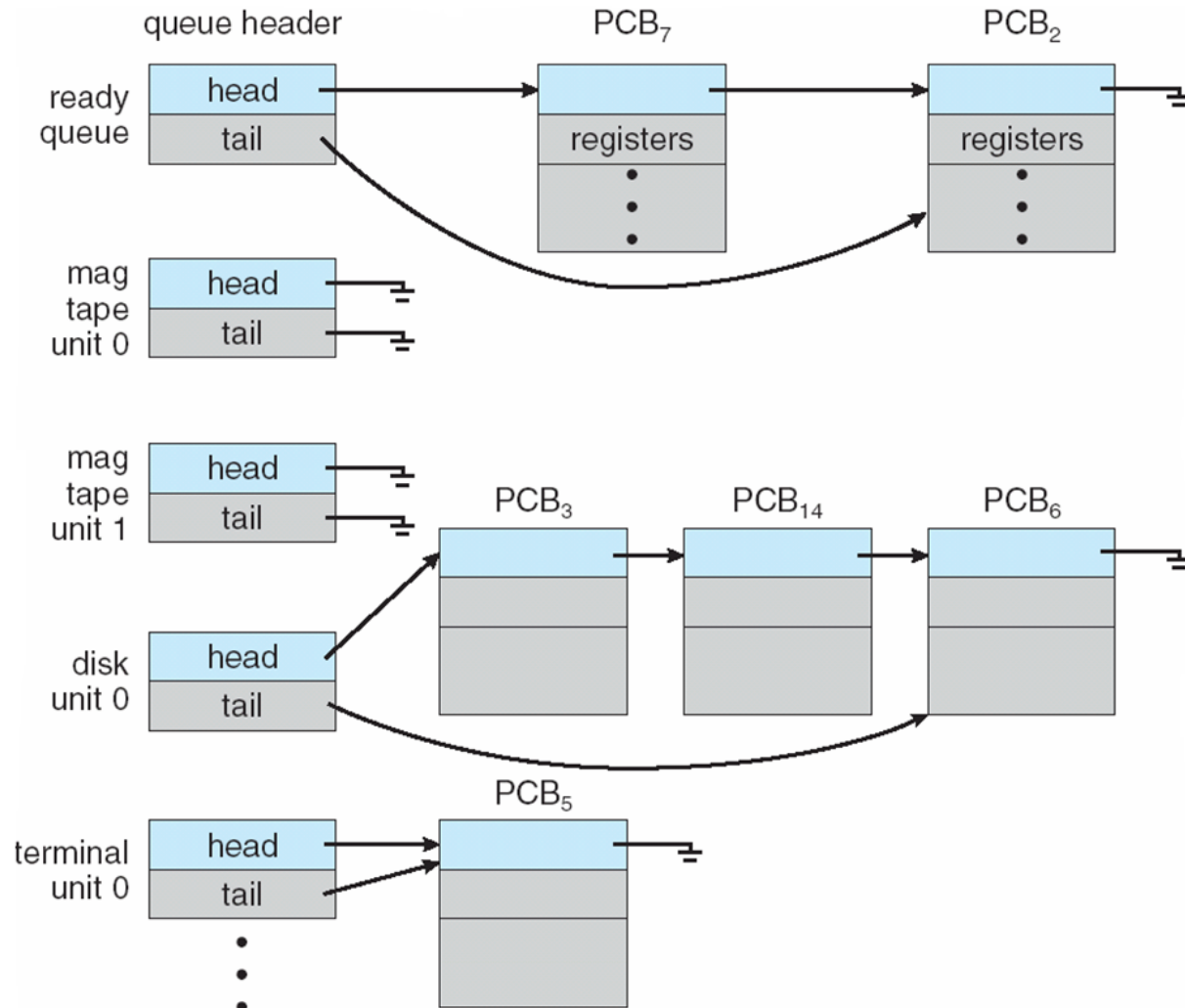
就绪队列指针



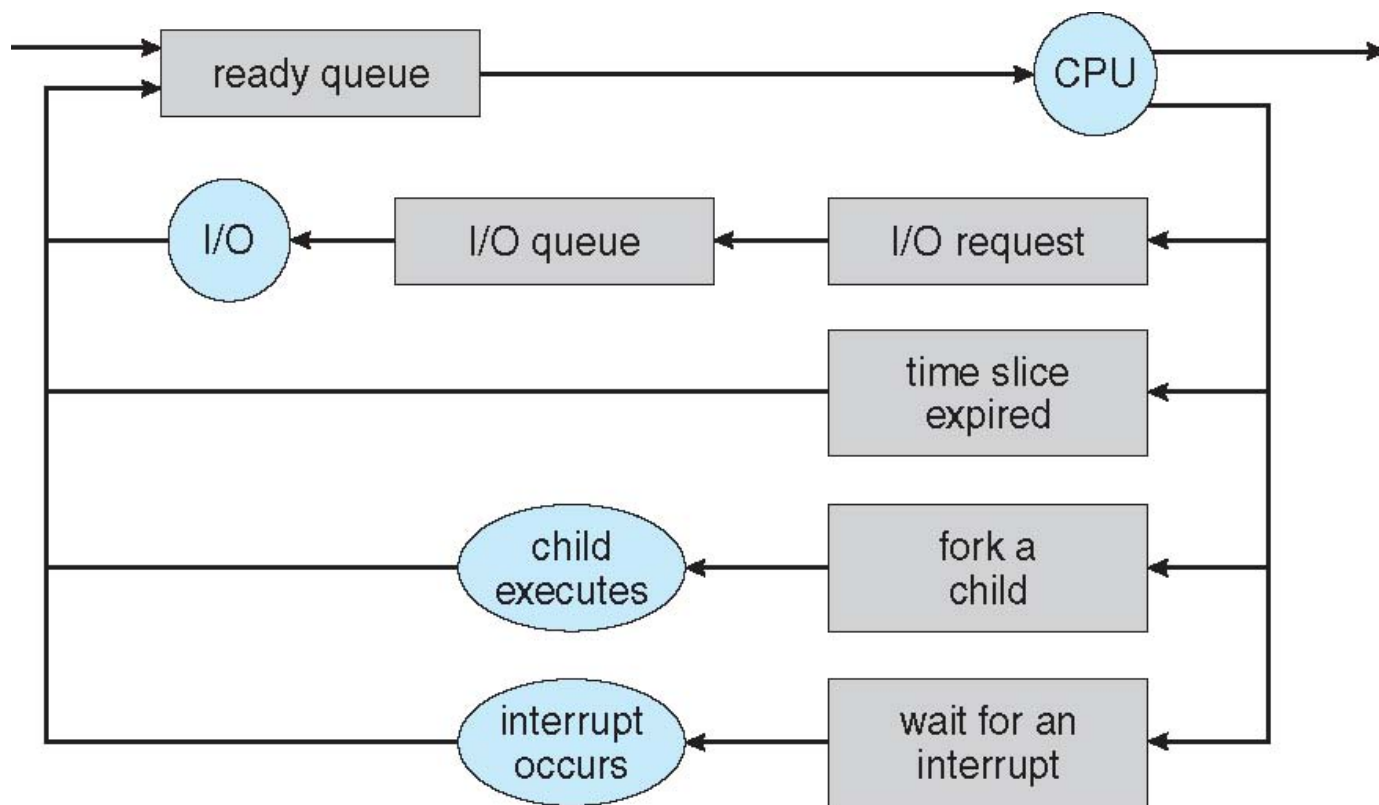
阻塞队列指针



就绪队列和各种I/O设备队列

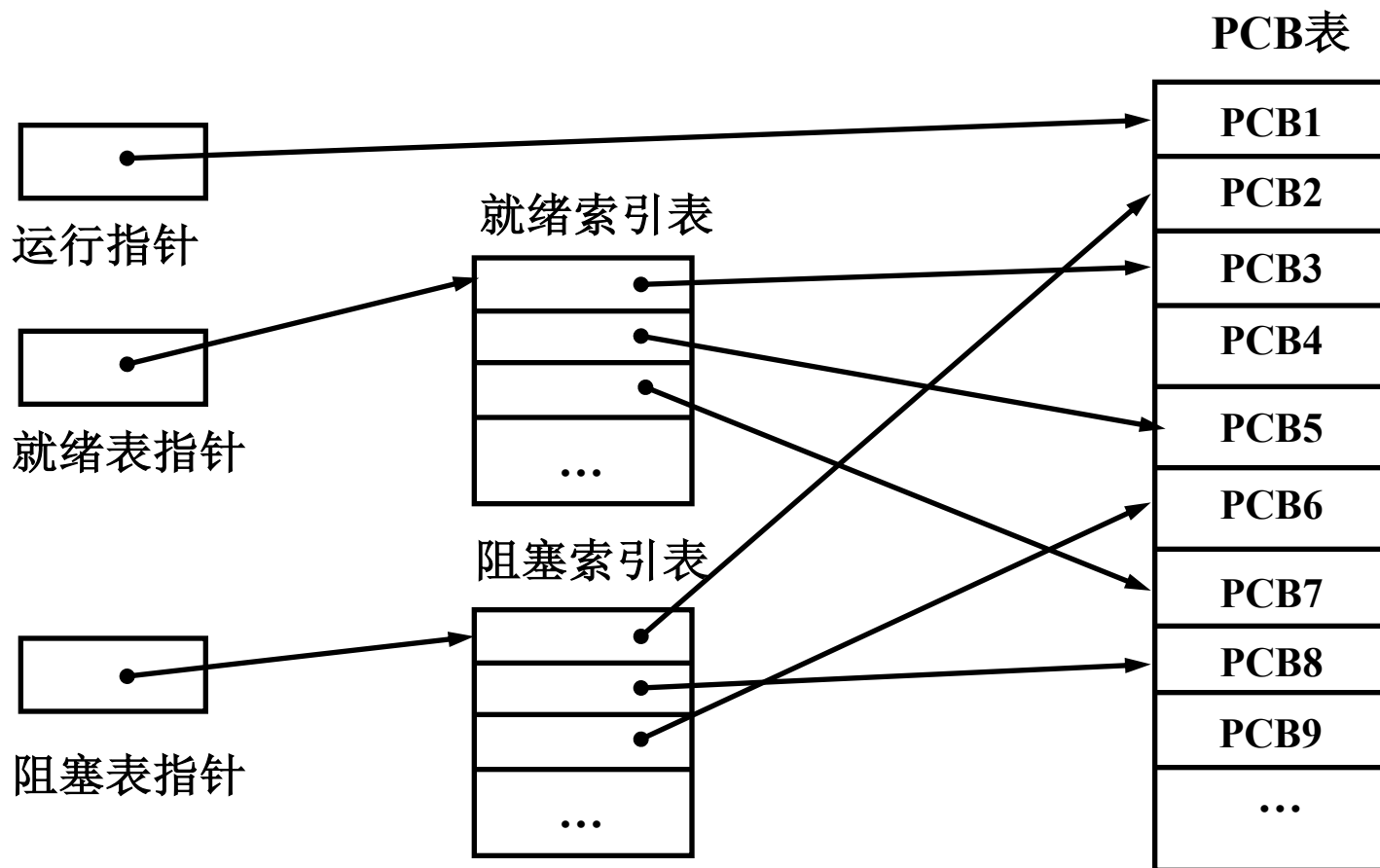


进程调度的队列图



索引方式

- 索引方式：将同一状态的进程归入一个索引表，再由索引指向相应的PCB





课堂讨论

在Linux系统中运行以下程序

```
main(){  
    printf("fork1 is:%d\n", fork());  
    printf("fork2 is:%d\n", fork());  
    printf("fork3 is:%d\n", fork());  
}
```

请问这个程序最多可以输出多少行打印信息？
产生多少个进程（含main函数本身）？请绘制进程家族树来说明。

解答:

- 包括自身，可共创建8个进程，输出信息14行，进程创建的过程图如下

