第3章 进程控制和进程调度

《计算机操作系统实验指导》

王红玲褚晓敏

内容

- Linux进程介绍
- Linux进程控制函数介绍
- 实验3.1 进程的创建
- Linux进程调度
- 实验3.2 进程的调度

Linux进程介绍

Linux进程

（1）**交互进程：**由一个shell启动的进程。交互进程既可在前台运行，也可以在后台运行，前者称为前台进程，后者称为后台进程。

（2）**批处理进程：**这种进程和终端没有联系，是一个进程系列，由多个进程按照指定的方式执行。

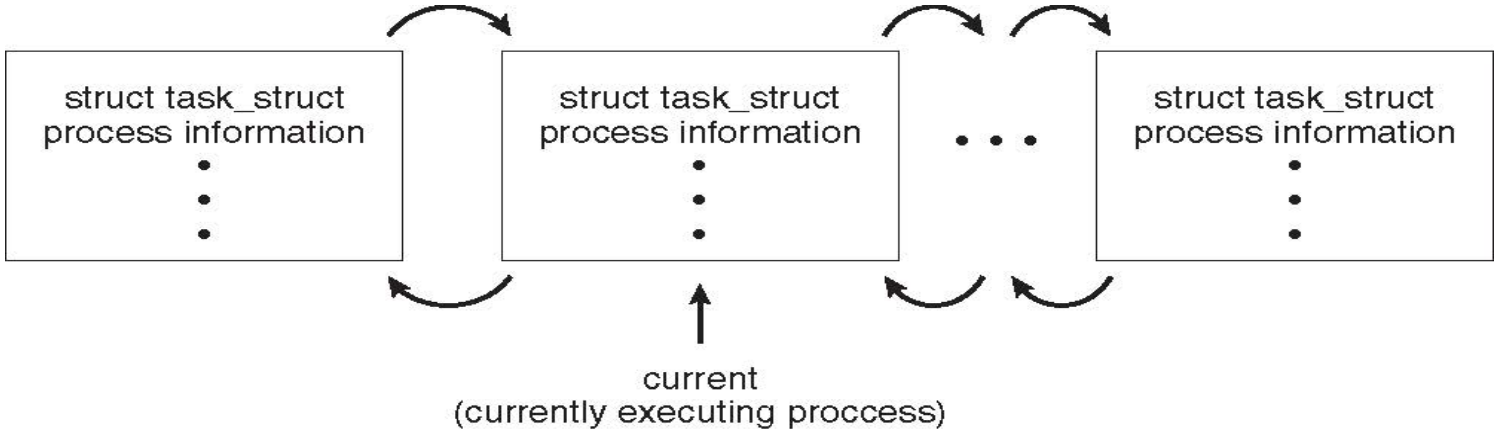
（3）**守护进程（Daemon）：**运行在后台的一种特殊进程，它在系统启动时启动，并在后台运行。

Linux PCB

C结构：task_struct

```
pid t_pid; /* process identifier */
long state; /* state of the process */
unsigned int time_slice /* scheduling information */
struct task_struct *parent; /* this process's parent */
struct list_head children; /* this process's children */
struct files_struct *files; /* list of open files */
struct mm_struct *mm; /* address space of this process
```

……



Linux进程状态

- 运行态（TASK_RUNNING）：进程准备运行，或正在运行
- 可中断等待态（TASK_INTERRUPTIBLE）：进程等待特定事件
- 可中断等待态（TASK_UNINTERRUPTIBLE）：进程处于等待状态，但是此刻进程是不可中断的
- 僵尸态（TASK_ZOMBIE）：进程已经停止运行，但在内存仍有结构(task_struct)
- 停止态（TASK_STOPPED/ TASK_TRACED ）：进程暂停状态

Linux三种资源拷贝方式

✓共享

共享同一资源，如虚存空间、文件等。仅增加有关描述符的用户计数器。

✓直接拷贝

相同的结构，原样复制。

✓COW(Copy On Write，写时拷贝)

在需要的时候才复制。

创建进程的系统调用

Linux提供了三个创建进程的系统调用：

✓fork()

用于普通进程的创建，采用COW方式。

✓vfork()

完全共享的创建，共享同一资源。

✓clone()

由用户指定创建的方式。

Linux进程控制函数

进程创建——fork ()（1）

- fork ()函数通过系统调用创建一个与原来进程几乎完全相同的进程
  - 两个进程可以做完全相同的事，
  - 根据初始参数或者传入的变量不同，两个进程也可以做不同的事。

- 一个进程调用fork ()函数后，系统先给新的进程分配资源，例如存储数据和代码的空间。然后把原来的进程的所有值都复制到新的进程中，只有少数值与原来的进程的值不同。相当于克隆了一个自己。

fork ()函数（2）

- **fork()的一个奇妙之处就是它仅仅被调用一次，却能够返回两次，它可能有三种不同的返回值：**
  1）（>0）在父进程中，fork返回新创建子进程的进程ID；
  2）（=0）在子进程中，fork返回0；
  3）（<0）如果出现错误，fork返回一个负值；

- **fork ()出错可能有两种原因**
  ① 当前的进程数已经达到了系统规定的上限，这时errno的值被设置为EAGAIN。
  ② 系统内存不足，这时errno的值被设置为ENOMEM。

进程标识符管理

- int getpid();　　//取得当前进程的标识符（进程ID）。
- int getppid();　　//取得当前进程的父进程ID。
- int getpgrp();　　//取得当前进程的进程组标识符。
- int getpgid(int pid); //将当前进程的进程组标识符改为当前进程的进程ID，使其成为进程组首进程，并返回这一新的进程组标识符。

加载新的进程映像——exec函数族（1）

- 创建的进程往往希望它能执行新的程序，在Linux中，进程创建和加载新进程映像是分离操作的。
- 在**Linux**中，当创建一个进程后，通常使用exec系列函数将子进程替换成新的进程映像。
- 注意：Linux中并不存在一个exec()的函数形式，exec指的是一组函数，一共有6个，分别是：

```
#include <unistd.h>
int execl(const char *path, const char *arg, ...);
int execlp(const char *file, const char *arg, ...);
int execle(const char *path, const char *arg, ..., char *const envp[]);
int execv(const char *path, char *const argv[]);
int execvp(const char *file, char *const argv[]);
int execve(const char *path, char *const argv[], char *const envp[]);
```

其中，只有execve()是真正意义上的系统调用，其它都是在此基础上经过包装的**库函数**

exec函数族（2）

- exec函数族的作用是**根据指定的文件名找到可执行文件，并用它来取代调用进程的内容**，换句话说，就是在调用进程内部执行一个可执行文件。
  - 这里的可执行文件既可以是二进制文件，也可以是任何Linux下可执行的脚本文件。

- 与一般情况不同，**exec函数族的函数执行成功后不会返回，因为调用进程的实体，包括代码段，数据段和堆栈等都已经被新的内容取代，**只留下进程ID等一些表面上的信息仍保持原样。

wait()/waitpid()函数

- 作用：父进程查询子进程状态

```
#include <sys/types.h>
#include <sys/wait.h>
pid_t wait(int *status);
pid_t waitpid(pid_t pid, int *status, int options);
```

- 进程一旦调用了wait()，就立即阻塞自己，由wait()自动分析是否当前进程的某个子进程已经退出，如果让它找到了这样一个已经变成僵尸态的子进程，wait ()就会收集这个子进程的信息，并把它彻底销毁后返回；如果没有找到这样一个子进程，wait ()就会一直阻塞在这里，直到有一个出现为止。
- wait()要与fork()配套出现,如果在使用fork()之前调用wait(),wait()的返回值则为-1,正常情况下wait()的返回值为子进程的PID
- 当父进程没有使用wait()函数等待已终止的子进程时,子进程就会进入一种无父进程清理自己尸体的状态，此时的子进程就是僵尸进程，不能在内核中清理尸体的情况

实验3.1 进程的创建

- 实验目的

（1）加深对进程概念的理解，进一步认识并发执行的实质。

（2）掌握Linux 操作系统中进程的创建和终止操作。

（3）掌握在Linux 操作系统中创建子进程并加载新映像的操作。

实验3.1 实验内容

（1）编写一个C 程序，并使用系统调用fork()创建一个子进程。要求如下：

①在子进程中分别输出当前进程为子进程的提示、当前进程的PID 和父进程的PID、根据用户输入确定当前进程的返回值、退出提示等信息。

②在父进程中分别输出当前进程为父进程的提示、当前进程的PID 和子进程的PID、等待子进程退出后获得的返回值、退出提示等信息。

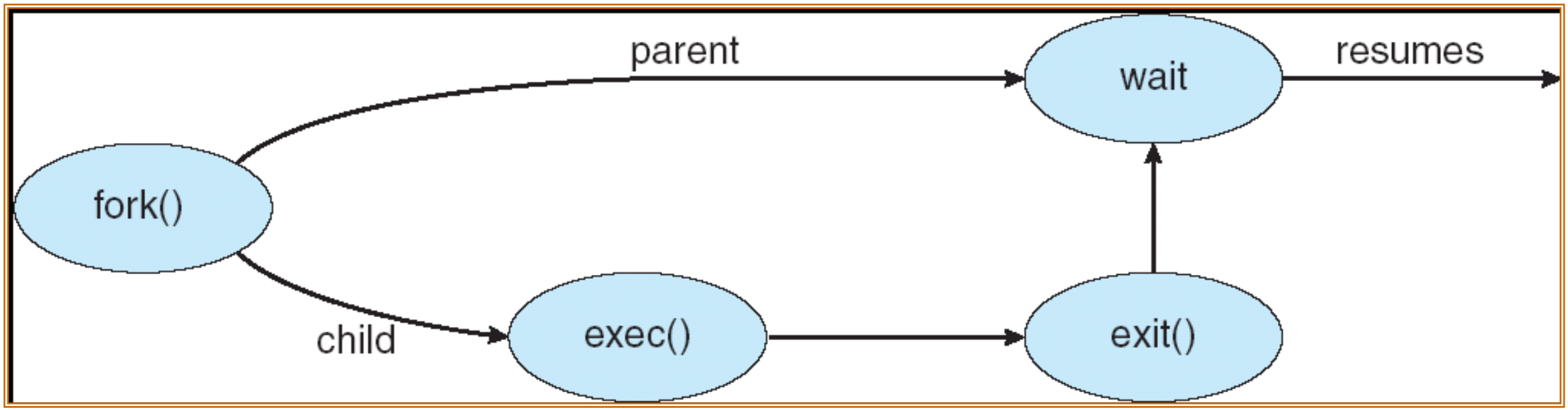
（2）编写另一个C 程序，使用系统调用fork()以创建一个子进程，并使用这个子进程调用exec 函数族以执行系统命令ls。

实验3.1 实验指导

- 本实验主要目的是学会在Linux下使用fork()创建进程，并验证fork()的返回值。首先在主程序中通过fork()创建子进程，并根据fork()的返回值确定所处的进程是子进程还是父进程，然后分别在子进程和当前进程（父进程）中调用getpid()、getppid()、wait()等函数以完成实验内容（1）
- 实验内容（2）的主要目的是学会在Linux下使用fork()创建进程、并使用exec族函数来加载新进程的映像。同时，也可以试验wait()函数的作用。

实验3.1 实验内容（2）示例代码

```
#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>
int main()
{
  pid_t  pid;
     pid = fork(); /* 创建子进程 */
     if (pid < 0) { /* 创建失败 */
                fprintf(stderr, "Fork Failed");
                return 1;
     }
     else if (pid == 0) { /* 子进程*/
            execlp(“/bin/ls”,“ls”,NULL);  /* 装载子进程映像 ls 命令*/
     }
     else { /* 父进程*/
                wait (NULL); /* 父进程等待子进程运行完毕 */
                printf ("Child Complete");
     }
       return 0;
}
```


parent
wait
resumes
fork()
child
exec()
exit()

Linux进程调度

Linux进程调度

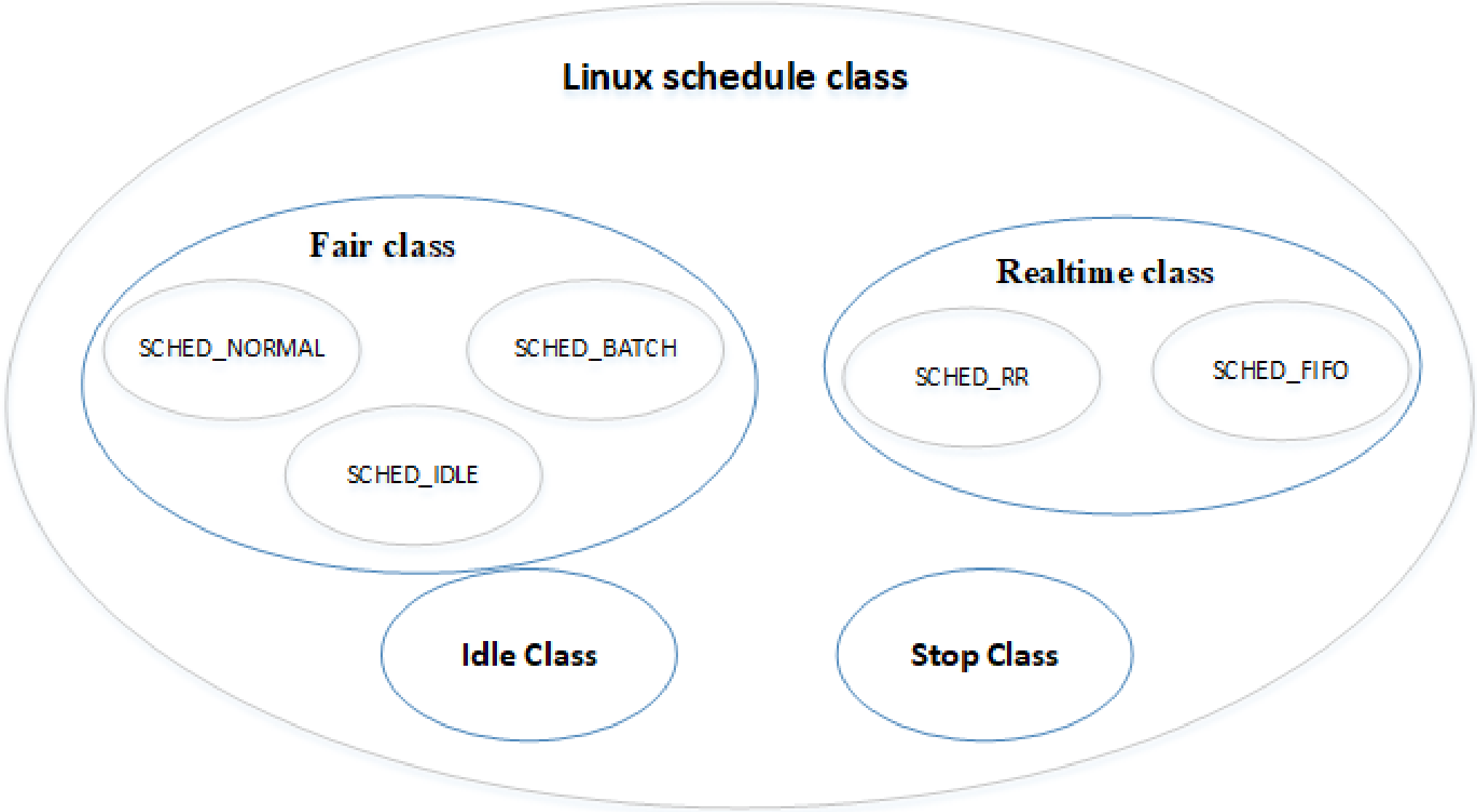
- Linux 中有一个总的调度结构，称之为调度器类（scheduler class）
  - 允许不同的可动态添加的调度算法并存，总调度器根据调度器类的优先顺序，依次挑选调度器类中的进程进行调度。
  - 确定调度器类后，再使用该调度器类的调度算法（调度策略）进行内部调度。

- 调度器类的优先级顺序为：

  Stop_Task > Real_Time > Fair > Idle_Task

  其中，Fair和Real_time最常用，分别采用CFS（完全公平调度算法）调度算法的默认调度类和实时调度类

Linux进程调度


Linux schedule class
Fair class
SCHED_NORMAL
SCHED_BATCH
SCHED_IDLE
Realtime class
SCHED_RR
SCHED_FIFO
Idle Class
Stop Class

实验3.2 进程调度算法的模拟

实验3.2 进程调度实验

- 实验目的

（1）加深对进程概念的理解，明确进程和程序的区别

（2）深入理解系统如何组织进程

（3）理解常用进程调度算法的具体实现

- 实验内容

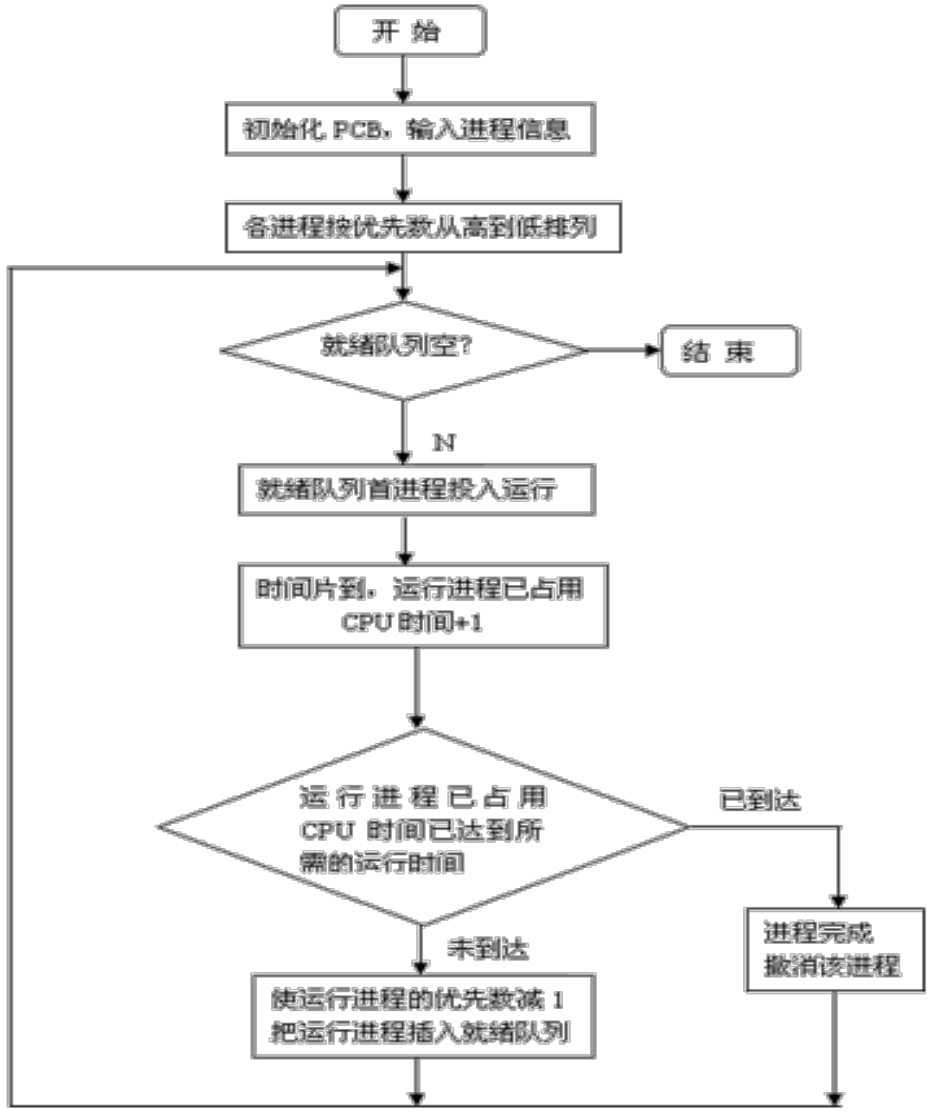
编写C程序模拟实现单处理机系统中的进程调度算法，实现对多个进程的调度模拟，要求采用常见进程调度算法（如先来先服务、时间片轮转和优先级调度等算法）进行模拟调度。

实验3.2 实验指导（1）

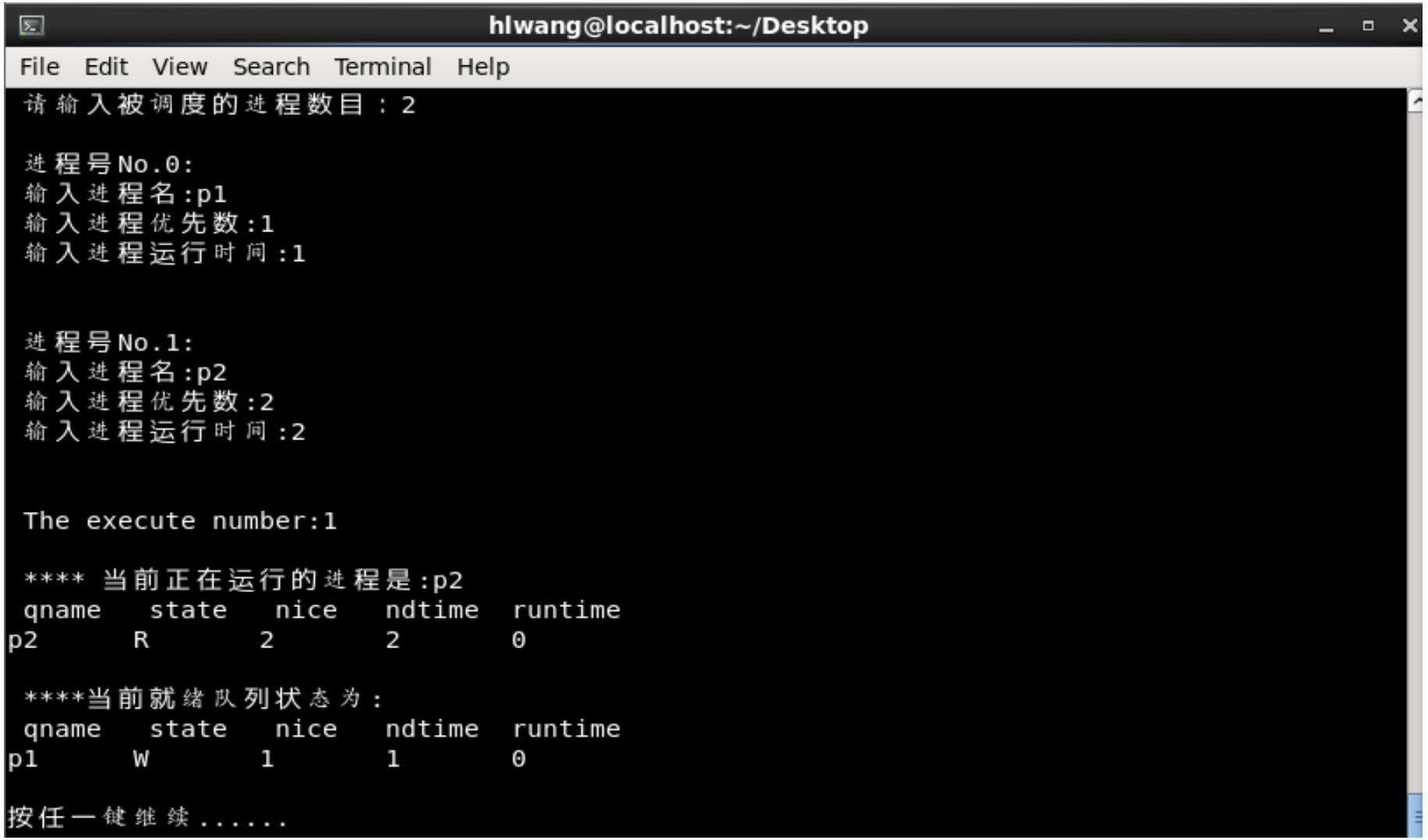
- 数据结构设计
  - PCB：结构体
  - 就绪队列：链表，每个节点为进程PCB
  - 进程状态
- 调度算法设计
  - 具体调度算法：FCFS、SJF、PR
  - 涉及多种操作：排序、链表操作
- 程序输出设计
  - 调度进程的顺序、每个进程的起始时间、终止时间等
  - CPU每次调度的过程

实验3.2 实验指导（2）

- 基于动态优先数的进程调度算法（示例代码）
  - 优先数大者优先，且优先数每运行一个时间单位降低一级（即优先数=优先数-1）
  - 进程的优先数及需要的运行时间事先人为地指定，以1个CPU时间单位进行计算
  - 进程状态：W（就绪态）、R（运行态）、F（完成态）
  - 输出：每进行一次调度程序都输出一次运行进程和就绪队列中的所有进程信息



实验3.2 实验结果


hlwang@localhost:~/Desktop
File Edit View Search Terminal Help
请输入被调度的进程数目：2
进程号No.0:
输入进程名:p1
输入进程优先数:1
输入进程运行时间:1
进程号No.1:
输入进程名:p2
输入进程优先数:2
输入进程运行时间:2
The execute number:1
**** 当前正在运行的进程是:p2
qname state nice ndtime runtime
p2 R 2 2 0
****当前就绪队列状态为:
qname state nice ndtime runtime
p1 W 1 1 0
按任一键继续.....