**实验四 进程调度**

**1 实验目的**

1. 理解 Linux 管理进程所用到的数据结构。

2. 理解 Linux 的进程调度算法的处理逻辑及其实现所用到的数据结构。

**2 实验内容**

1. 通过查阅参考书或者上网找资料，熟悉/usr/src/linux（注意：这里最后一级目录名可能是个含有具体内核版本号和“linux”字符串的名字）下各子目录的内容，即所含 Linux源代码的情况。

2. 分析 Linux 进程调度有关函数的源代码，主要是 schedule()函数，并且要对它们引用的头文件等一并分析。

3. 实现 Linux 的进程调度算法及理解其实现所用的主要数据结构。

**3 实验准备**

**3.1.Linux进程状态的描述**

Linux将进程状态描述为如下五种：

TASK\_RUNNING：可运行状态。处于该状态的进程可以被调度执行而成为当前进程。

TASK\_INTERRUPTIBLE：可中断的睡眠状态。处于该状态的进程在所需资源有效时被唤醒，也可以通过信号或定时中断唤醒。

TASK\_UNINTERRUPTIBLE：不可中断的睡眠状态。处于该状态的进程仅当所需资源有效时被唤醒。

TASK\_ZOMBIE：僵死状态。表示进程结束且已释放资源，但其 task\_struct 仍未释放。

TASK\_STOPPED：暂停状态。处于该状态的进程通过其他进程的信号才能被唤醒。

**3.2. 进程的虚拟地址空间**

调度方式Linux中的每个进程都分配有一个相对独立的虚拟地址空间。该虚存空间分为两部分：用户空间包含了进程本身的代码和数据；内核空间包含了操作系统的代码和数据。Linux采用“有条件的可剥夺”调度方式。对于普通进程，当其时间片结束时，调度程序挑选出下一个处于TASK\_RUNNING状态的进程作为当前进程（自愿调度）。对于实时进程，若其优先级足够高，则会从当前的运行进程中抢占CPU 成为新的当前进程（强制调度）。发生强制调度时，若进程在用户空间中运行，就会直接被剥夺CPU；若进程在内核空间中运行，即使迫切需要其放弃CPU，也仍要等到从它系统空间返回的前夕才被剥夺CPU。

**3.3. 调度策略**

1）SCHED\_OTHER

SCHED\_OTHER 是面向普通进程的时间片轮转策略。采用该策略时，系统为处于TASK\_RUNNING状态的每个进程分配一个时间片。当时间片用完时，进程调度程序再选择下一个优先级相对较高的进程，并授予CPU 使用权。

2）SCHED\_FIFO

SCHED\_FIFO 策略适用于对响应时间要求比较高，运行所需时间比较短的实时进程。采用该策略时，各实时进程按其进入可运行队列的顺序依次获得 CPU。除了因等待某个事件主动放弃CPU，或者出现优先级更高的进程而剥夺其CPU 之外，该进程将一直占用CPU运行。

3）SCHED\_RR

SCHED\_RR 策略适用于对响应时间要求比较高，运行所需时间比较长的实时进程。采用该策略时，各实时进程按时间片轮流使用 CPU。当一个运行进程的时间片用完后，进程调度程序停止其运行并将其置于可运行队列的末尾。

**3.4. 相关函数**

1）Schedule()函数

Schedule()函数首先对所有任务（进程）进行检测，唤醒任何一个得到信号的任务。具 体方法是针对任务数组中的每个任务，检查其报警定时值alarm。如果任务的alarm时间已经 过期（alarm<jiffies），则在它的信号位图中设置SIGALRM信号，然后清alarm值。jiffies是 系统从开机开始算起的滴答数（10ms/滴答），在sched..h 中定义。如果进程的信号位图中 除去被阻塞的信号外还有其他信号，并且任务处于可中断睡眠状态 （TASK\_INTERRUPTIBLE），则置任务为就绪状态（TASK\_RUNNING）。

2）sleep\_on()函数

sleep\_on()函数的主要功能是当一个进程（或任务）所请求的资源正忙或不在内存中时 暂时切换出去，放在等待队列中等待一段时间，当切换回来后再继续运行。放入等待队列的 方式是利用了函数中的tmp指针作为各个正在等待任务的联系。

3）wake\_up()函数 唤醒操作函数wake\_up()把正在等待可用资源的指定任务置为就绪状态。该函数是一个通用唤醒函数。在有些情况下，例如读取磁盘上的数据块，由于等待队列中的任何一个任务 都可能被先唤醒，因此还需要把被唤醒任务结构的指针置空。这样，在其后进入睡眠的进程 被唤醒而又重新执行sleep\_on()时，就无需唤醒该进程了。