layout: _post title: OS date: 2020-01-01 13:39:52 tags:

- os

4 进程及进程管理

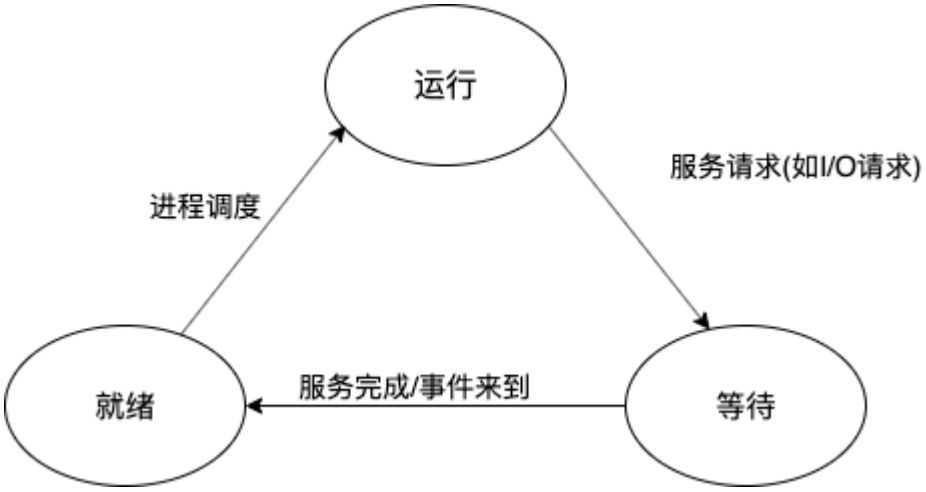
4.1 进程

> 简单地说：进程是计算机系统有限资源地基本单位，在**不支持线程**的处理机上是调度的基本单位(否则是线程)

进程的状态变化

- 运行
- 就绪
- 等待

状态变迁图:



进程的控制

> **功能**
>
> - 进程控制
> - 进程调度
> - 进程间的同步协调

进程创建

> 过程
>
> - **形成**进程控制块PCB，填充进程标识符，进程优先级(PCB块并不是创建时才生成，系统会维持一个PCB池，空闲时PCB进程标识符存放 '-1')

> - UNIX 或Linux系统中，父进程创建子进程时，该子进程继承父进程占用的系统资源，以及**除进程内部系统标示符以外**的其他特性

UNIX和Linux系统创建进程及应用实例(调用fork系统调用)

- 分配PCB
- 为子进程分配一个唯一的PID(进程标识号)
- **拷贝**父进程**数据段**(不会有多线程线程修改父进程的变量的危险操作)和堆栈段数据到新的主存区，正文段(代码段)引用计数+1(即共享代码段)
- 增加与进程相关联的文件表和索引节点表的引用数，即子进程可以使用父进程打开的文件
- 子进程返回0,父进程返回子进程进程号(PID)

进程和线程具体使用

- github仓库

进程撤销

- 归还PCB -> PCB资源池，占用资源 -> 父进程
- 转进调度程序

进程等待

- CPU现场保留至PCB现场保护区
- PCB插入到等待队列

进程唤醒

- 被唤醒，转为就绪态
- PCB插入就绪队列

4.4 进程之间的约束关系

- 互斥 -> 存在临界区
- 同步 -> 进程执行有顺序限制

4.5 同步机构

> 操作系统提供的同步机构
>
> - 锁
> - 信号灯(信号量) 和P,V操作

锁

- 可以把它想像成现实生活中的锁，机制是差不多的。

信号灯

- 一组确定的二元组(s,q)

- P操作，以P(s)为例
  - s减1
  - 如果结果 >= 0 ,则进程继续执行
  - 小于0，线程被封锁，并将它(前面提到过，这里插入的是进程的PCB)插入到该信号灯的等待队列中，然后进入调度程序
- V操作，以V(s)为例
  - s值加1
  - 如果结果大于0，进程**继续执行**
  - 如果结果小于或者等于零，则从该信号灯的等待队列中移除一个进程，解除它的等待状态，然后返回本进程**继续执行**
  - V操作不会使当前线程状态变化

4.6线程互斥和同步的实现

> 总的来说，常见的就那么几种，直接举几个用信号灯实现的例子

简单互斥

```
// 伪代码
// 这里定义cobegin 和coend之间的程序是同时执行的
//
main(){
    int mutex = 1 ;
    cobegin
        p_a();
        p_b();
    coend
}
p_a(){
    p(mutex)
    // do something
    v(mutex)
}
p_b(){
    p(mutex)
    // do something
    v(mutex)
}
```

简单的先后

```
    // 伪代码，一个看病的例子，先看病，再化验，最后诊断
    main(){
        int s1 = 0;
        int s2 = 0;
        cobegin
        labora();
        diagnosis();
```

```
        coend
    }
labora(){
    while(化验工作未完成){
        p(s1);
        化验工作;
        v(s2);
    }
}
diagnosis(){
    while(看病工作未完成){
        看病;
        v(s1);
        p(s2);
        诊断
    }
}
```

多级先后关系

> **实现思路**
>
> 和简单先后的思想差不多，如果要让线程-1在另外一个线程-2后面执行，只需要控制信号灯的值在线程-1 执行完之前都小于等于0即可

生成者消费者问题

```
    // 一个有界缓冲区 多个消费者度，多个生产者写问题
    //         _ _ _ _ _ _ _ _
    // 读 <-- | | | | | | | | <-- 写
    //    <-- |_|_|_|_|_|_|_| <--
    main(){
        int full = 0;   //缓冲区可读的缓冲块数目
        int empty = n;  // 缓冲区可写的缓冲区数目
        int mutex = 1;  // 同时只能有一个线程能够写/读缓冲区
        cobegin
        p1();p2();p3();pk();
        c1();c2();c3();ck();
        coend
    }
    producer(){
        while(/*生产未完成*/){
            //生产一个产品;
            p(empty);
            p(mutex);
           // 写一个缓冲块
            v(mutex);
            v(full);
        }
    }
```

```
consumer(){
    while(/*还要继续消费*/){
        p(full);
        p(mutex);
        // 读一个缓冲块;
        v(mutex);
        v(empty);

    }
}
```

4.7进程通信

进程通信的几种方式

- 消息缓冲通信
  - 在进程的主存空间设置一个接收区，然后用接收原语(原语可以理解为原子性操作，即操作不可分割)接收消息。
- 信箱通信
  - 用户空间信箱
  - 系统空间信箱

4.8线程

> 概念提出 -> 为了进一步提高系统的并行处理能力

线程描述

- 线程是进程的一条**执行路径**
- 私有堆栈和处理机环境(PC,CPU运行时间,寄存器)
- 共享父进程的主存(想一下和子进程的区别？)
- 任务调度的基本单位(一般地，进程是系统**资源分配**的基本单位，线程是**系统调度**的基本单位)

内核线程和用户线程

- 内核线程由操作系统支持，运行在内核空间
- 用户线程是在内核的支持下，在用户层通过线程库实现(即用户态线程和内核线程不一定是1:1，也有可能是多对一)。

Linux下进程和进程管理

> - Linux下面，线程被视为一个与其他进程共享某些资源的**特殊进程**
> - PCB -> task_struct
> - 进程状态基本和上述一致，但是等待状态有TASK_INTERUPTIBLE和TASK_UNINTERUPTIBLE，即是和否可以被中断改变其等待状态