* 并发概念

进程交替占用CPU执行片刻。

* 为什么要引入并发执行？

1. 按时间片并发执行，即分时操作系统，满足多用户多任务。
2. 进程等待I/O时不使用CPU，让出CPU给其他进程使用，提高CPU的利用率。

* 如何并发?

每个进程都有PCB(进程控制块),由PCB控制进程的状态和并发.

* 并发带来的新问题

临界资源,进程同步

进程1从CPU撤下来，进程2上去，但是进程2未必能执行，如进程2使用的临界资源被进程1正在使用，如打印机，或进程2和进程1会修改同一个变量，如银行存钱取钱时你的账户余额。进程1需要等进程2的资源才能启动.相关进程执行要分先后.

并发执行的条件：为每个进程设置PCB。

程序能够并发执行的条件：

自由并发：能够满足程序并发执行条件的多进程之间的并发。【p1从处理机上被撤下来，p2上去就能执行。因为p2不会使用p1使用的临界资源，也不用等p1的消息】

制约并发：能够满足程序并发执行条件的多进程之间的并发。【p1从处理机上被撤下来，p2上未必就能执行。因为p2可能要使用p1使用的临界资源，要等p1的消息】

* 简述程序和进程的区别

程序是静态的，永存的。进程是动态的，暂存的。程序和进程的关系相当于剧本和场幕。

进程的描述

* 简单介绍进程。

1，进程实体 =  程序 + 数据 + PCB

2，进程运行除程序和数据外，还需资源和CPU

3，进程是分配资源和调度运行的基本单元

4，PCB是进程存在的唯一标志，用于控制和管理进程

* 简述进程控制块PCB的主要信息。

答：程序和数据的地址，资源清单，进程标识符，家族联系，进程状态，进程优先级，CPU现场保护区，进程队列指针，消息队列指针，信号量，进程等待时间和已执行时间等信息。

* 进程有哪几种状态？简述各状态的特征。

①创建②静止就绪③就绪④执行⑤阻塞⑥终止【⑦挂起】任何状态都可转换成挂起状态。

创建：过程：1，申请空白PCB。2，填写PCB。3，申请资源

申请到除内存外的所有资源进入静止就绪状态，申请到包括内存在内的所有资源进入就绪状态。

静止就绪：进程申请到除内存和CPU以外的所有资源。进程在内存外，进入内存立刻转为就绪状态。

就绪：进程申请到除CPU以外的所有资源。在内存中的进程，能被调度的进程状态。

执行：单处理机最多一个进程处于执行状态。【执行状态只能由就绪状态装换】

阻塞：根据阻塞原因设置多个阻塞队列，能提高系统效率。

终止：过程：1，中止执行。2，终止子进程。3，归还资源给父进程或系统。4，等待其他进程收集信息 5，清空并归还PCB。

挂起：

* 为什么要引入进程挂起状态？

1，终端用户发现程序异常，挂起它修改。

2，父进程挂起子进程来协调各子进程。

3，操作系统挂起进程统计资源。

4，挂起不重要的进程从而削弱系统负荷，有利于实时任务。

* 简述进程终止的原因。

答：1，正常结束 2，异常错误 3，父进程请求结束 4，父进程被终止 5，由于死锁被操作系统终止。

* 进程各状态转换流程。

创建->静止就绪/就绪，内存足够时直接转入就绪状态，不够时转入静止就绪状态。

静止就绪->就绪。【内存足够时】

就绪->执行。【调度】

执行->就绪。【时间片用完】

执行->阻塞。【临界资源，消息，等待I/O，等待任务】

阻塞->就绪。

就绪->静止就绪。【挂起原语】

执行->静止就绪。【挂起原语】

阻塞->挂起。

挂起原语造成的状态被激活成就绪或阻塞。

每当由激活的进程进入就绪队列，立刻检测该进程与当前正在执行的进程的优先级，若较高，立即切换。

额外注意点：

1，当进程处于阻塞状态，即使把处理器分配给进程，它也无法运行。

2，在时间片轮转调度中，时间片用完后，进程转换为就绪状态而非阻塞状态。

1. 进程由执行状态转换成阻塞状态，是进程调用阻塞原语主动阻塞自己。阻塞状态转换成就绪状态，一般是由一个发现者进程调用激活原语激活，是被动行为。发现者进程一般是与阻塞进程相互合作的进程。

调度算法

评价一个调度算法的指标？

1.CPU利用率 2.系统吞吐量 3.响应时间 4.周转时间

1，先来先服务

按照进入就绪队列的先后次序分配处理机

特点：

①有利于计算繁忙型进程，不利于I/O繁忙型进程

②对短进程不公平（周转时间远远大于执行时间）

2，短进程优先

对占用CPU时间最短的进程优先分配处理机

特点：

①长作业饥饿

②平均周转时间最短

3，优先级调度

特点：

①静态优先级和动态优先级

②抢占式优先级调度（等当前次优先级主动停止执行再占有CPU ）和非抢占式优先级调     度（强制剥夺CPU）

③相同优先级队列一般采用先来先服务或短进程优先调度算法

4，时间片轮转法

特点：

①时间片太长，退化成先来先服务；时间片太短，进程切换频繁开销大

②响应时间T = 就绪队列进程数N × 时间片q

③

5，多级反馈队列

①设置多个就绪队列，队列的优先级逐个降低，队列的时间片逐级增大

②Q1队头进程运行到时间片结束，进入Q2队尾……Qn-1队头进程运行到时间片结           束，进入Qn队尾，Qn队头进程运行到时间片结束进入自身队列队尾，即最后一级队     列采用时间片轮转调度算法

③高优先级队列存在进程，队列外调度算法就不会调度次高优先级队列的进程

④正在运行的次高优先级队列的进程，会被突然进入空高优先级的队列抢占式中断，       此时，被中断的次高优先级进程进入自身队列队尾

线程

* 为什么说线程是为了使操作系统具有更高的并发性？

1.线程时CPU调度的基本单元,进程是分配资源的基本单元

2.线程基本不拥有系统资源仅有少量寄存器和栈，相较于进程的“重资源”，创建，撤销，切换状态时有更小的时空开销。

3.线程共享进程的内存地址和资源，线程之间的同步和通信简单。

4.进程之间可以并发，同一进程的线程之间以及不同进程的线程之间也可并发。

5.可以把同一进程的不同线程放到不同处理机上执行，可支持多处理机系统。

内核支持线程：  
在内核空间为每一个内核线程开辟线程控制块，系统通过线程控制块感知线程的存在。

|  |  |
| --- | --- |
| 内核支持线程 | 用户级线程 |
|  |  |
| 内核空间设置TCB，内核感知存在，内核控制 | 用户空间线程库控制，内核无法感知存在 |
| OS调度单位是线程 | OS调度单位是进程，线程可在线程库自选调度算法 |
| 资源由内核管理 | 资源由内核管理 |
| 因为线程的控制和管理是在内核实现的，用户的线程切换状态需要从用户态转到核心态，系统开销大。 | 所有线程由线程库控制和管理，无需进入内核，线程切换速度快，即线程管理与内核无关。 |
| 直接通过系统调用取得内核服务 | 借助中间系统通过系统调用取得内核服务 |
| 一个线程被“系统调用”阻塞，该进程的其他线程不受影响 | 一个线程被“系统调用”阻塞，该进程的其他进程一起阻塞 |
| 可为一个进程分配多个CPU，可支持多处理机 | 一个进程只能分配一个CPU，同一进程只能一个线程占用CPU |

* 进程通信的几种方式

管道( pipe )：管道是一种半双工的通信方式，数据只能单向流动，而且只能在具有亲缘关系的进程间使用。进程的亲缘关系通常是指父子进程关系。

有名管道 (named pipe) ： 有名管道也是半双工的通信方式，但是它允许无亲缘关系进程间的通信。

信号量( semophore ) ：信号量是一个计数器，可以用来控制多个进程对共享资源的访问。它常作为一种锁机制，防止某进程正在访问共享资源时，其他进程也访问该资源。因此，主要作为进程间以及同一进程内不同线程之间的同步手段。

消息队列( message queue ) ： 消息队列是由消息的链表，存放在内核中并由消息队列标识符标识。消息队列克服了信号传递信息少、管道只能承载无格式字节流以及缓冲区大小受限等缺点。

信号 ( signal ) ： 信号是一种比较复杂的通信方式，用于通知接收进程某个事件已经发生。

共享内存( shared memory ) ：共享内存就是映射一段能被其他进程所访问的内存，这段共享内存由一个进程创建，但多个进程都可以访问。共享内存是最快的 IPC 方式，它是针对其他进程间通信方式运行效率低而专门设计的。它往往与其他通信机制，如信号量，配合使用，来实现进程间的同步和通信。

套接字( socket ) ： 套解字也是一种进程间通信机制，与其他通信机制不同的是，它可用于不同及其间的进程通信。

进程的同步

进程同步：相互合作的多个进程在某一个关键点需要互相等待或互相交换信息。

临界资源：任一时刻，仅容许一个进程访问，多个进程必须互斥访问，一旦开始使用，必须使用完才能释放给其他进程使用。

临界区：进程中访问临界资源的代码。

访问临界资源方法：

进入区：检查能否访问

临界区：访问

退出区：恢复可访问

互斥访问临界资源的要求：闲则让进，忙则等待，有限等待，让权等待。

系统资源的分类：

临界资源，非临界资源；

可抢占资源，不可抢占资源；

可重用资源：重复利用，进程用了自己释放。P，V操作在同一进程内

可消耗资源：用完就没了，使用进程仅申请不必释放，需要其他进程释放，P，V操作不在同一进程中。

只有不可抢占的临界资源才能引起死锁！

信号量机制

什么是信号量？

typedef struct

{

    int count;

    pcb Queue;

}sem;

资源和信号量映射，信号量阐述资源的两种属性：资源数目||等待资源的进程数目，等待资源的进程。

任何与该资源相关的进程都可以通过仅且通过初始化，wait和signal原语改变它的count和Queue.

初始化满足：count>=0 & Queue=NULL

sem S; 【注释】S是全局变量

//S->count = 1;初始化>=0

wait(S)

{

    S->count --;

    if(S->count<0)

    p阻塞自己进入阻塞队列Queue；

}

【wait】申请资源！

资源不够，主动进入阻塞队列；资源够用，继续执行。

signal(S)

{

    S->count ++;

  if(S->count<=0)

  唤醒阻塞队列的队头进程;

}

【signal】释放资源！或生产资源！

使用完毕，归还资源。if(有因该资源阻塞的进程){唤醒阻塞进程出队，自身继续执行}；else{自身继续}。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

|  |  |  |
| --- | --- | --- |
|  | 互斥 | 不互斥 |
| p1类 | PV | 无 |
| p2类 | PV | 无 |
| 类间 |  |  |

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

1，P1执行完才能执行P2[前s后w]

sem S;

S.count = 0; S.Queue = NULL;

P1{

    signal(S);

    P1代码内容;

}

P2{

    wait(S);

    P2代码内容;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

2，多个进程使用1个可重用临界资源

sem S;

S.count = 1; S.Queue = NULL;

P1{

    wait(S);

    P1的临界区代码;

    signal(S);

}

P2{

    wait(S);

    P2的临界区代码;

    signal(S);

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

3，生产者-消费者问题

生产者生产产品放进一个有界缓冲区，消费者从此缓冲区取走产品，放进取拿走不能同时进行，放满了不能再放，取空了不能再取。

Semaphore mutex = 1;

Semaphore  full = 0;

Semaphore  empty = n;

生产者

{

    while (true)

    {

        p(empty);

        p(mutex);

        产品放入缓冲池

            v(mutex);

        v(full);

    }

}

消费者

{

    while (true)

    {

        p(full);

        p(mutex);

        从缓冲池取出产品

            v(mutex);

        v(empty);

    }

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

4，读者-写者问题

如何实现多个进程互斥使用临界资源？【多个进程可以包含若干完全相同的进程，不同进程的临界区代码可以相同】

实现方法：信号量机制

每个进程的临界区前加wait，临界区后加signal

p

{

    其他代码;

    wait(S); //判断临界资源是否可用，不可用进入阻塞队列，可用执行临界区代码，并禁止其他后来进程执行临界区代码

    临界区代码;

    signal(S); //执行完临界区代码，允许其他进程进入自己的临界区，若阻塞队列含有PCB，唤醒队头PCB进入就绪队列，等待调度占有处理机执行临界区代码

    其他代码;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

资源信号量的原理

生产者-消费者问题

n个单元的缓冲区，生产者生产的数据在缓冲区未满的情况下放入缓冲区，消费者在缓冲区有数据的情况下拿走数据，放入数据和拿走数据不能交叉执行。

【分析】生产者是一个进程，该进程的临界区是在缓冲区放数据；消费者是一个进程，该进程的临界区是在缓冲区拿走数据，缓冲区是共享的临界资源，

故把进程阻塞进入Queue或唤醒Queue中的一个阻塞进程。

2、什么是缓冲区溢出？有什么危害？其原因是什么？

　　缓冲区溢出是指当计算机向缓冲区内填充数据时超过了缓冲区本身的容量，溢出的数据覆盖在合法数据上。

　　危害：在当前网络与分布式系统安全中，被广泛利用的50%以上都是缓冲区溢出，其中最著名的例子是1988年利用fingerd漏洞的蠕虫。而缓冲区溢出中，最为危险的是堆栈溢出，因为入侵者可以利用堆栈溢出，在函数返回时改变返回程序的地址，让其跳转到任意地址，带来的危害一种是程序崩溃导致拒绝服务，另外一种就是跳转并且执行一段恶意代码，比如得到shell，然后为所欲为。通过往程序的缓冲区写超出其长度的内容，造成缓冲区的溢出，从而破坏程序的堆栈，使程序转而执行其它指令，以达到攻击的目的。

造成缓冲区溢出的主原因是程序中没有仔细检查用户输入的参数

# 内存管理

编译：源文件编译成多个目标模块，后缀是.o，每个模块的地址从0开始。

链接：把多个目标文件和库函数链接成一个装入模块，地址从0开始。

装入：将可执行代码装入内存，逻辑地址转换成物理地址。

链接方式：

（1）静态链接，在程序运行之前，把所有模块和库函数链接成一个完整的可执行程序，以后不能再拆开。

（2）装入时链接，将目标模块，边装入边链接。

（3）运行时动态链接，在执行过程中，发现需要调用的模块未装入内存，就去寻找装入内存，链接到调用者模块。

装入后，进行重定位。

静态重定位：一个连续的存储区，连续存储区不够，整个程序就不能分配；程序所在的内存地址，后续不能再改变；不需要硬件地址变换机构。

动态重定位：程序部分装入时即可运行；程序可随便装入任何一块内存，取指令时临时用硬件转换机构把逻辑地址转换成物理地址。

现代操作系统一般采用，运行时动态链接和动态重定位。