IMG_256 简介

顺序程序设计特性：

1.程序执行的顺序性

程序指令执行严格按序

2.计算环境的封闭性

程序运行时如同独占受操作系统保护的资源

3.计算结果的确定性

程序执行结果与执行速度和执行时段无关

4.计算过程的可再现性

程序对相同数据集的执行轨迹是确定的

进程的并发执行

多道程序设计-》多个程序同时进入内存竞争处理器-》获得运行机会

OS允许计算机系统在一个时间段内存在多个正在运行的进程，即允许多个进程并发执行

OS保证按照“顺序程序设计“方法编制的程序在并发执行时不受影响，如同独占计算机

这些按照顺序程序设计思想编制的进程在OS中并发执行属于无关的并发进程

IMG_257 信号量

定义：可动态定义的软件资源

核心数据结构：等待进程队列

IMG_258 信号量声明： 资源报到，建立队列

申请资源的原语：若申请不得，调用进程入队等待

归还资源的原语：若队列中有等待进程，需释放

信号量撤销：资源注销，撤销队列

IMG_259 记录型信号量

定义：一种带数值的软资源

Typedef struct semaphore{

Int value;//信号量值

Struct pcb\* list；//信号量等待进程队列指针

}

每个信号量建立一个等待进程队列

每个信号量相关一个整数值

1.正值表示资源可复用次数

2.0值表示无资源且无进程等待

3.负值表示等待队列中进程个数

IMG_260 临界区

临界区指的是一个访问共用资源（例如：共用设备或是共用存储器）的程序片段，而这些共用资源又无法同时被多个线程访问的特性。当有线程进入临界区段时，其他线程或是进程必须等待（例如：bounded waiting 等待法），有一些同步的机制必须在临界区段的进入点与离开点实现，以确保这些共用资源是被互斥获得使用，例如：semaphore。

IMG_261 PV操作解决进程互斥问题框架

分为P操作原语和V操作原语

P操作原语用于申请一个资源

V操作原语用于释放一个资源

Procedure P(semaphore:s){

S = s - 1; //信号量减去1

If(s < 0) W(s); //若信号量小于0意味着原来没有资源可用，则调用进程被置成等待信号量s的状态

}

Procedure V(semaphore:s){

S:= s + 1; //信号量加1

If(s <= 0)R(s); //若信号量小于等于0，表明原来此记录型信号量原来在等待进程队列中有等待进程存在，所以释放一个等待进程

}

应用PV操作解决进程互斥问题

Semaphore s;

S = 1; //表示该临界资源只能为一个进程所私有

Cobegin

Process Pi{

….

P(s);//如果s初值为1，表明没有进程在临界区中则可以进入临界区

临界区

V(s);

}

Coend;

例：

PV操作解决机票问题

Int A[m];

Semaphore s;

S = 1;

Cobegin

Process Pi{

Int Xi;

Li:按旅客订票要求找到A[j]；

P(s);

Xi = A[j];

If(Xi >= 1){

Xi = Xi-1;

A[j] = Xi;

V(s);

输出一张票;

}

Else{

V(s);//P操作和V操作在执行路径上一一匹配

输出票已售光;

}

Goto Li;

}

Coend

改进

Int A[m];

Semaphore s[m];

For(int j = 0;j <m;j ++)

S[j] = 1;

Cobegin

Process Pi{

Int Xi;

Li:按旅客订票要求找到A[j]；

P(s[j]);

Xi = A[j];

If(Xi >= 1){

Xi = Xi-1;

A[j] = Xi;

V(s[j]);

输出一张票;

}

Else{

V(s[j]);//P操作和V操作在执行路径上一一匹配

输出票已售光;

}

Goto Li;

}

Coend

IMG_262 理论

在整型信号量机制中，信号量被定义为一个整型变量。除初始化外，仅能通过两个标准的原子操作wait(s)和signal(s)来访问。其通常分别被称为P、V操作。P意为通过，V意为释放。

信号量的物理含义是：当S>0时，其值表示要管理的某类资源的数量；当S<0，它的绝对值表示在相关队列中等待的进程个数。

进程的同步

一般来说，一个进程相对于另一个进程的运行速度是不确定的，也就是说，进程是在异步环境下运行的，每个进程都以各自独立的、不可预知的速度向前推进。但是，相互合作的进程需要在某些确定的点上协调它们的工作，当一个进程到达了这些点后，除非另一个进程已经完成了某些操作，否则就不得不停下来等待这些操作结束。

进程的互斥

在多道程序系统中，各进程可以共享各类资源，但有一些资源一次只能供一个进程使用，这种资源称为临界资源。

对临界区的管理原则：有空则进，无空则等。有限等待，让权等待。

进程互斥的情况是，信号量初值为1；而同步的情况是，初值为0

在解决问题时面对各种并发进程，首先应分析它们之间哪些是互斥关系，哪些是同步关系，由此确定应该设置哪些信息量及它们的初值。

如果所设置的信息量，每个相关进程既能对它施行P操作，也能对它施行V操作，则称其为共用信息量。用于互斥的都是公用信息量。

若只能有一个进程对其施行P操作，其他进程只能对其施行V操作，则称其为那一个进程的私有进程量。用于同步或资源分配管理的都是私有信号量。

IMG_263 <signal.h>

用途

**signal.h**是

[C标准函数库](https://baike.baidu.com/item/C%E6%A0%87%E5%87%86%E5%87%BD%E6%95%B0%E5%BA%93)中的信号处理部分， 定义了程序执行时如何处理不同的信号。信号用作进程间通信， 报告异常行为（如除零）、用户的一些按键组合（如同时按下Ctrl与C键，产生信号SIGINT）。

Unistd.h

对于类 Unix 系统，unistd.h 中所定义的接口通常都是大量针对[系统调用](https://baike.baidu.com/item/%E7%B3%BB%E7%BB%9F%E8%B0%83%E7%94%A8)的封装（英语：wrapper functions），如 fork、pipe 以及各种 [I/O](https://baike.baidu.com/item/I/O) 原语（read、write、close 等等）。

Sys/wait.h

使用wait和waitpid函数时需要include这个头文件

例

中山大学操作系统软件工程实验1

<https://blog.csdn.net/sinat_31790817/article/details/70149889?utm_source=blogxgwz2>

2.信号处理实验

A)编写一段程序，使用系统调用fork()创建两个子进程，再用系统调用signal()让父进程捕捉键盘上来的中断信号，当捕捉到中断信号后，父进程调用kill()向两个子进程发出信号，子进程捕捉到信号后，分别输出下面信息后终止：

Child process 1 is killed by parent!

Child process 2 is killed by parent!

父进程等待两个子进程终止后，输出以下信息后终止：

Parent process is killed!

关于unistd.h fork()的讲解：

<http://www.cnblogs.com/liulipeng/archive/2013/12/05/3460222.html>

代码如下：

#include <stdio.h>

#include <unistd.h>

#include <signal.h>

#include <stdlib.h>

#include <sys/wait.h>

Void waiting();

Void stop(int signum);

Int wait\_mark;

Int main(){

Int pid1,pid2;

Pid1 = fork();

If(pid1 > 0){

Pid2 = fork();

If(pid2 > 0){

Wait\_mark = 1;

Signal(SIGINT, stop);//设置收到信号ctrl c时执行stop函数

Waiting();

Kill(pid1, SIGINT);

Kill(pid2,SIGINT);

Waitpid(pid1,NULL,0);

Waitpid(pid2,NULL,0);

Printf("parent process is killed!\n");

Exit(0);

}

Else{

Wait\_mark = 1;

Signal(SIGINT, stop);

Waiting();

Printf("child process 2 is killed by parent!\n");

Exit(0);

}

}

Else{

Wait\_mark = 1;

Signal(SIGINT,stop);

Waiting();

Printf("child process 1 is killed by parent!\n");

Exit(0);

}

}

Void waiting(){

While(wait\_mark != 0);

}

Void stop(int signum){

Wait\_mark = 0;

}