第一章 操作系统

操作系统基本特征

并发 资源共享

1、中断

由外部事件（时钟中断、硬件中断）导致，并且发生时间不可预测。

- 时间中断：如果某一个程序运行了足够长用完了分配给它的时间片，CPU决定切换到另一个进程运行，就会产生一个时钟中断，切换到下一个进程运行。
- 硬件中断：由硬件引起的中断，比如一个程序需要用户输入一个数据，但用户一直没有输入，操作系统决定是一直等待用户输入还是转而运行别的进程。

2、陷阱

陷阱指令可以使执行流程从用户态陷入内核，并把控制权转移给操作系统，使得用户程序可以调用内核函数和使用硬件从而获得操作系统所提供的服务。

3、异常

程序执行过程中的异常行为。比如除零异常，缓冲区溢出异常

中断和异常的区别

CPU处理中断的过程中会屏蔽中断，不接受新的中断直到此次中断处理结束。而陷阱的发生并不屏蔽中断，可以接受新的中断。

4、直接内存访问（DMA)

允许外围设备和主内存之间直接传输它们的I/O数据，而不需要系统处理器的参与。

中断和DMA的区别

1. 中断方式是在数据缓冲寄存器满之后，要求CPU进行中断处理。
2. DMA方式是在所要求的数据块全部结束后要求CPU进行中断，有DMA控制器进行控制，不需要CPU的太多的资源，大大减少了CPU进行中断的次数。

5、多核和多CPU的区别

1. 多核CPU是指1个CPU里面有多个处理核心。
2. 多个CPU是指在一块主板上存在多个处理器芯片。
3. 相对来说，多个CPU的系统比单CPU的系统拥有更多的处理核心数量，能应付更大的数量处理工作。
4.

6、多道程序设计

- **定义**：多道程序设计是在计算机内存中同时存放几道相互独立的程序，使它们在管理程序控制之下，相互穿插的运行。 两个或两个以上程序在计算机系统中同处于开始到结束之间的状态。
- **特征**：间断性、共享性、制约性。

分时系统

分时系统要求计算机系统是可交互（interactive）的，以便用户与系统直接通信。用户通过输入设备，如键盘、鼠标、触摸板、触摸屏等向操作系统或程序发出指令，并等待输出设备的即时结果。相应地，响应时间（response time）应当较短，通常小于 1 秒。

7、操作系统的两种模式

1）用户模式

2）内核模式

第二章 操作系统概述

OS提供的界面

-GUI、图形用户界面

图形用户界面是一种人与计算机通信的界面显示格式，允许用户使用鼠标等输入设备操纵屏幕上的图标或菜单选项，以选择命令、调用文件、启动程序或执行其它一些日常任务。

-批处理

批处理(Batch)，也称为批处理脚本。顾名思义，批处理就是对某对象进行批量的处理，通常被认为是一种简化的脚本语言，它应用于DOS和Windows系统中

-命令行

命令解释器 shell

shell是一个命令行解释器。它为用户提供了一个向Linux内核发送请求以便运行程序的界面系统级程序，用户可以用shell来启动、挂起、停止和编写一些程序。shell就是用户操作Linux系统的界面。

系统调用

系统调用是通向操作系统本身的接口，是面向底层硬件的。通过系统调用，可以使得用户态运行的进程与硬件设备(如CPU、磁盘、打印机等)进行交互，是操作系统留给应用程序的一个接口。

POSIX

POSIX表示可移植操作系统接口（Portable Operating System Interface of UNIX，缩写为 POSIX ），POSIX标准定义了操作系统应该为应用程序提供的接口标准，是IEEE为要在各种UNIX操作系统上运行的软件而定义的一系列API标准的总称，其正式称呼为IEEE 1003，而国际标准名称为ISO/IEC 9945。

与库函数的区别

库函数

把函数放到库里，供别人使用的一种方式。.方法是把一些常用到的函数编完放到一个文件里，供不同的人进行调用。分两类，一类是C语言标准规定的库函数，一类是编译器特定的库函数。

区别

1. 库函数是语言或应用程序的一部分，而系统调用是内核提供给应用程序的接口，属于系统的一部分
2. 库函数在用户地址空间执行，系统调用是在内核地址空间执行，库函数运行时间属于用户时间，系统调用属于系统时间，库函数开销较小，系统调用开销较大
3. 库函数是有缓冲的，系统调用是无缓冲的
4. 系统调用依赖于平台，库函数并不依赖

操作系统的四种结构

简单结构

1. MS-DOS：利用最小的空间提供更多的功能，没有仔细划分模块，没有很好地区分接口和功能层次,应用程序可以直接访问硬件。没有用户态和内核态。
2. UNIX：UNIX系统由内核和系统程序两个独立部分组成。内核进一步分为一系列接口和驱动程序。物理硬件之上和系统调用接口之下的所有部分作为内核，内核通过系统调用以提供文件系统、CPU调用、内存管理和其他操作系统功能。难扩展。

分层结构

操作系统分成若干层（级）。最底层（层0）为硬件，最高层（层N）为用户接口。分层法的主要优点在于构造和调试的简单化。每层只能利用较低层的功能和服务。这种方法简化了调试和系统验证。每层都是利用较低层提供的功能来实现的。该层不必知道如何实现这些功能，它只需要知道这些操作能做什么。因此，每层为较高层隐藏了一定的数据结构、操作和硬件的存在,扩展性强。

微内核

将所有非基本部分从内核中移走，并将它们实现为系统程序或用户程序。微内核通常包括最小的进程和内存管理以及通信功能。微内核的主要功能是使客户程序和运行在用户空间的各种服务进行通信。通信以消息传递形式提供。

移不出去的功能

CPU调度 内存管理 进程交流

微内核系统举例

微内核系统有WindowNT,Minix,Mac等等

宏内核系统举例

宏内核的系统有Unix，Linux

模块化结构

将整个操作系统**按功能**划分为若干个**模块**，每个模块实现一个特定的功能。模块之间的通信只能通过预先定义的接口进行。或者说模块之间的相互关系仅限于接口参数的传递。模块间联系少，模块内联系紧密。这种方法使用动态加载模块，并在现代的UNIX，如Solaris、Linux和Max OSX中很常见。

第三章 进程

进程定义

1. 运行中的程序（区别于未运行的程序）
2. 进程是资源分配的最小单位
3. CPU调度的一个单位（**CPU调度的最小单位是线程**）

进程特征

1. **动态性**：进程是程序的一次执行，它有着创建、活动、暂停、终止等过程，具有一定的生命周期，是动态地产生、变化和消亡的。动态性是进程最基本的特征。
2. **并发性**：指多个进程实体，同存于内存中，能在一段时间内同时运行，并发性是进程的重要特征，同时也是操作系统的重要特征。引入进程的目的就是为了使程序能与其他进程的程序并发执行，以提高资源利用率。
3. **独立性**：指进程实体是一个能独立运行、独立获得资源和独立接受调度的基本单位。凡未建立PCB的程序都不能作为一个独立的单位参与运行。
4. **异步性**：由于进程的相互制约，使进程具有执行的间断性，即进程按各自独立的、 不可预知的速度向前推进。异步性会导致执行结果的不可再现性，为此，在操作系统中必须配置相应的进程同步机制。
5. **结构性**：每个进程都配置一个PCB对其进行描述。从结构上看，进程实体是由程序段、数据段和进程控制段三部分组成的。

Multiple parts

栈 stack

由编译器自动分配和释放，存放：

1. 函数内部的**局部变量**
2. 函数的**参数值**

堆 heap

由程序员分配和释放，如果程序员不释放，程序结束时可能由OS释放，存放：new, malloc产生的动态数据

全局数据库 data

全局变量和**静态数据**，即使是函数内部的静态局部变量，均存放于全局数据区。 程序结束后由OS释放

常量区 data

所有常量，程序结束后由OS释放

文本段 text

存放程序的二进制代码

```
//main.cpp
int a = 0;                    //全局初始化区
char *p1;                    //全局未初始化区
main()
{
    int b;                          //栈
    char s[] = “abc”;        //栈
    char *p2;                     //栈
    char *p3 = “123456”;          //123456/0在常量区，p3栈
    static int c =0;                    //全局（静态）初始化区
        p1 = (char *)malloc(10);       //堆区
    p2 = (char *)malloc(20);       //堆区
    strcpy(p1, "123456");          //123456/0放在常量区，编译器可能会将它与p3所指向的"123456"优化成一个地方。

}
```

进程的状态

new(新建)

进程在创建时需要申请一个空白PCB，向其中填写控制和管理进程的信息，完成资源分配。如果创建工作无法完成，比如资源无法满足，就无法被调度运行，把此时进程所处状态称为创建状态

running(运行)

进程处于就绪状态被调度后，进程进入执行状态

waiting(等待或被阻塞)

正在执行的进程由于某些事件（I/O请求，申请缓存区失败）而暂时无法运行，进程受到阻塞。在满足请求时进入就绪状态等待系统调用

ready(就绪)

进程已经准备好，已分配到所需资源，只要分配到CPU就能够立即运行

terminated(终止)

进程结束，或出现错误，或被系统终止，进入终止状态。无法再执行

进程状态转换图

进程与线程的对比

概念

进程

进程是具有独立功能的程序关于某个数据集合上的一次运行活动，是资源分配的基本单位。一个进程包括多个线程。

线程

是程序执行流的最小单元，是系统独立调度和分配CPU（独立运行）的基本单位。

不同之处

1. 调度：线程作为调度的基本单位，同进程中线程切换不引起进程，当不同进程的线程切换才引起进程切换；进程作为拥有资源的基本单位。
2. 并发性：一个进程间的多个线程可并发。
3. 拥有资源：线程仅拥有隶属进程的资源；进程是拥有资源的独立单位。
4. 系统开销：进程大；线程小。

多线程与多进程

多线程：同一时刻执行多个线程。用浏览器一边下载，一边听歌，一边看视频，一边看网页。

多进程：同时执行多个程序。如，同时运行YY，QQ，以及各种浏览器。

进程控制块 PCB

进程在内存中的位置是离散的，是有操作系统的内存管理单元分配的

进程控制块(PCB)是系统为了管理进程设置的一个专门的数据结构。系统用它来记录进程的外部特征，描述进程的运动变化过程。同时，系统可以利用PCB来控制和管理进程，所以说，PCB（进程控制块）是系统感知进程存在的唯一标志

PCB记录进程的所有信息，一个进程对应一个PCB。

进程切换

定义

操作系统为了控制进程的执行，必须有能力挂起正在CPU上运行的进程，并恢复以前挂起的某个进程的执行，这种行为被称为进程切换，任务切换或上下文切换。

过程

1. 决定是否作上下文切换以及是否允许作切换
2. 保存当前执行进程的上下文
3. 使用进程调度算法，选择一处于就绪状态的进程
4. 恢复进程上下文，将CPU控制权交到所选进程中。

对CPU的影响

每次上下文切换都需要几十纳秒到数微妙的CPU时间，特别是在进程上下文切换次数较多的情况下，很容易导致CPU将大量时间耗费在寄存器、内核栈以及虚拟内存等资源的保存和恢复上，进而大大缩短了真正运行进程的时间，是导致平均负载升高的一个重要因素。

进程调度

在许多进程或线程都准备使用CPU进行任务处理时，就会存在资源竞争和分配的问题。一般都会将进程或线程先放在一个缓冲池中，等待合适的时机调度程序从中选择一个进程或线程进行交给CPU进行处理。

1、长期调度

长期调度，又称为**作业调度**或**高级调度**，这种调度将已进入系统并处于后备状态的作业按某种算法选择一个或一批，为其建立进程，并进入主机，当该作业执行完毕时，还负责回收系统资源。

在**批处理系统**中，需要有作业调度的过程，以便将它们分批地装入内存，在分时系统和实时系统中，通常不需要长期调度。它的频率比较低，主要用来控制内存中进程的数量。

2、中期调度

中期调度，又称为**交换调度**。 它的核心思想是能将进程从内存或从CPU竞争中移出，从而降低多道程序设计的程度，之后进程能被重新调入内存，并从中断处继续执行，这种交换的操作可以调整进程在内存中的存在数量和时机。

其主要任务是按照给定的原则和策略，将处于外存交换区中的就绪状态或等待状态的进程调入内存，或把处于内存就绪状态或内存等待状态的进程交换到外存交换区——即**换入换出**。

3、短期调度

短期调度，又称为**进程调度**、低级调度或微观调度。 这也是通常所说的调度，一般情况下使用最多的就是短期调度。 它的主要任务是按照某种策略和算法将CPU分配给一个处于就绪状态的进程，分为**抢占式**和**非抢占式**。

实例：**实时系统、分时系统**

调度算法

先进先出算法、短作业优先、轮转法、多级反馈队列（第五章详述）

进程创建（fork）

进程的创建

新进程的创建，首先在内存中为新进程创建一个task_struct结构，然后将父进程的task_struct内容复制其中，再修改部分数据。分配新的内核堆栈、新的PID、再将task_struct 这个node添加到链表中。所谓创建，实际上是“复制”。子进程刚开始，内核并没有为它分配物理内存，而是以只读的方式共享父进程内存，**只有当子进程写时，才复制。**

fork（）函数

fork函数时调用一次，返回两次。在父进程和子进程中各调用一次。子进程中返回值为0，父进程中返回值为子进程的PID。程序员可以根据返回值的不同让父进程和子进程执行不同的代码。

代码举例，凹处代表执行

子进程和父进程之间并不会对各自的变量产生影响。

一般来说，fork之后父、子进程执行顺序是**不确定**的，这取决于内核调度算法。进程之间实现同步需要进行进程通信。

vfork函数以及对比

vfork与fork对比：

相同：返回值相同

不同：fork创建子进程，把父进程数据空间、堆和栈**复制**一份；vfork创建子进程，与父进程内存数据**共享**；

vfork先保证子进程先执行，当子进程调用exit()或者**exec**后，父进程才往下执行

exec函数

fork()函数通过系统调用创建一个与原来进程（父进程）几乎完全相同的进程（子进程是父进程的副本，它将获得父进程数据空间、堆、栈等资源的副本。注意，子进程持有的是上述存储空间的“副本”，这意味着父子进程不共享这些存储空间。linux将复制父进程的地址空间内容给子进程，因此，子进程由了独立的地址空间。），也就是这两个进程做完全相同的事。在fork后的子进程中使用exec函数族，可以装入和运行其它程序（子进程替换原有进程，和父进程做不同的事）。

在fork后的子进程中使用exec函数族，可以装入和运行其它程序（子进程替换原有进程，和父进程做不同的事）。

缓冲区问题

printf是一个行缓冲函数，先写到缓冲区，满足条件后才将缓冲区刷新到对应文件中，刷新缓冲区的条件如下：

1）缓冲区填满；

2）写入的字符中有‘\n’、'\r'；

3）调用fflush手动刷新缓冲区；

4）调用scanf要从缓冲区读取数据时，也会将缓冲区内的数据刷新。

5）另外，当执行printf的进程或线程结束的时候，也会主动调用flush来刷新缓冲区。

子进程复制父进程时会复制缓冲区，所以要注意printf函数没有“\n”的题目。

题目链接

进程通信

消息传递

定义：在协作进程间交换消息来实现通信

1. 消息传递对于交换较少数量的数据很有用，因为不需要避免冲突。
2. 对于计算机间的通信，消息传递也比共享内存更易于实现。
3. 通常用系统调用来实现，因此需要更多的内核介入的时间消耗。

共享内存

定义：建立起一块供协作进程共享的内存区域，进程通过向此共享区域读或写数据来交换信息

1. 共享内存允许以最快的速度进行方便的通信，在计算机中它可以达到内存的速度。
2. 仅在建立共享内存区域时需要系统调用，一旦建立了共享内存，所有的访问都被处理为常规的内存访问，不需要来自内核的帮助

通常，操作系统不允许一个进程访问另一个进程的内存空间，共享内存要求两个或以上的进程打破这种限制。即，进程之间可以在共享内存通过读写交换数据。

方法1：System V共享内存

允许多个进程共享同一块物理内存 直接将共享的内存页面通过链接，映射到需要通信的进程各自的虚拟地址空间 进而使得多个进程可以直接访问同一个物理内存页面

方法2：POSIX

使用POSIX API，创建共享内存段。共享内存让一段内存可供多个进程访问。 用特殊的系统调用（即对 UNIX 内核的请求）分配和释放内存并设置权限；通过一般的读写操作读写内存段中的数据。 共享内存并不是从某一进程拥有的内存中划分出来的；进程的内存总是私有的。 共享内存是从系统的空闲内存池中分配的，希望访问它的每个进程连接它。这个连接过程称为映射，它给共享内存段分配每个进程的地址空间中的本地地址。

进程执行

顺序执行

定义

通常可以把一个应用程序分成若干个程序段，在各程序段之间，必须按照某种先后次序顺序执行，仅当前一操作(程序段)执行完后，才能执行后继操作。

特征

1. **顺序性**：处理机的操作严格按照程序所规定的顺序执行，即每一操作必须在上一个操作结束之后开始。
2. **封闭性**：程序是在封闭的环境下执行的，即程序运行时独占全机资源，资源的状态(除初始状态外)只有本程序才能改变它。程序一旦开始执行，其执行结果不受外界因素影响。

3. **可再现性**：只要程序执行时的环境和初始条件相同，当程序重复执行时，不论它是从头到尾不停顿地执行，还是"停停走走"地执行，都将获得相同的结果。

程序顺序执行时的特性，为程序员检测和校正程序的错误带来了很大的方便。

并发执行

1. **间断性**：程序在并发执行时，由于它们共享系统资源，以及为完成同一项任务而相互合作，致使在这些并发执行的程序之间，形成了相互制约的关系。例如，图2-3 中的I、C和P是三个相互合作的程序，当计算程序完成Ci - 1的计算后，如果输入程序I尚未完成Ii的处理，则计算程序就无法进行Ci的处理，致使计算程序必须暂停运行。又如，当打印程序完成Pi的打印后，若计算程序尚未完成Ci + 1的计算，则打印程序就无法对Ci + 1的计算结果进行打印。一旦使程序暂停的因素消失后(如Ii已处理完成)，计算程序便可恢复执行对Ci的处理。简而言之，相互制约将导致并发程序具有"执行—暂停—执行"这种间断性的活动规律。
2. **失去封闭性**：程序在并发执行时，是多个程序共享系统中的各种资源，因而这些资源的状态将由多个程序来改变，致使程序的运行失去了封闭性。这样，某程序在执行时，必然会受到其它程序的影响。例如，当处理机这一资源已被某个程序占有时，另一程序必须等待。
3. **不可再现性**：程序在并发执行时，由于失去了封闭性，也将导致其再失去可再现性。（多个部分同时进行，则某一变量会因为进行的顺序不同，过程不一样，但结果是一样的）

第四章 线程

线程定义

线程（英语：thread）是操作系统能够进行运算调度的最小单位。它被包含在进程之中，是进程中的实际运作单位。一条线程指的是进程中一个单一顺序的控制流，一个进程中可以并发多个线程，每条线程并行执行不同的任务。

> 线程是CPU使用/调度的基本单元,由**线程ID、程序计数器、寄存器集合**和**栈**组成。

与属于同一进程的其他线程**共享**代码段、数据段和其他操作系统资源，如打开文件和信号。

多线程优势

1. 高响应度
2. 资源共享
3. 经济
4. 多处理器体系结构的利用

多线程模型

线程有**用户级**线程和**内核级**线程

有些系统同时支持用户线程和内核线程由此产生了不同的多线程模型，即实现用户级线程和内核级线程的连接方式。

1、多对一模型

将多个用户级线程映射到一个内核级线程，线程管理在用户空间完成。

此模式中，用户级线程对操作系统不可见（即透明）。

优点：线程管理是在用户空间进行的，因而效率比较高。

缺点：当一个线程在使用内核服务时被阻塞，那么整个进程都会被阻塞；多个线程不能并行地运行在多处理机上。

2、一对一模型

将每个用户级线程映射到一个内核级线程。

优点：当一个线程被阻塞后，允许另一个线程继续执行，所以并发能力较强。

缺点：每创建一个用户级线程都需要创建一个内核级线程与其对应，这样创建线程的开销比较大，会影响到应用程序的性能。

3、多对多模型

将 n 个用户级线程映射到 m 个内核级线程上，要求 m <= n。

特点：在多对一模型和一对一模型中取了个折中，克服了多对一模型的并发度不高的缺点，又克服了一对一模型的一个用户进程占用太多内核级线程，开销太大的缺点。又拥有多对一模型和一对一模型各自的优点，可谓集两者之所长。

线程库

定义

线程库为程序员提供创建和管理线程的 API。

线程库实现方法

1. 在用户空间中提供一个没有内核支持的库。这种库的所有代码和数据结构都位于用户空间。调用库内的一个函数只是导致了用户空间内的一个本地函数的调用，而不是系统调用。
2. 执行一个由操作系统直接支持的内核级的库。 此时，库的代码和数据结构存在于内核空间中。 调用库中的一个API函数通常会导致对内核的系统调用。

主流线程库

POSIX Pthread

Pthread作为POSIX标准的扩展，可以提供用户级或内核级的库

（在linux环境下当采用多线程编程时，需要在编译的时候加上-lpthread(或-pthread) 以显示链接该库。之所以这样是因为pthread并非Linux系统的默认库，而是POSIX线程库。）

Win32

Win32线程库是适用于Windows操作系统的内核级线程库

Java

Java语言都是用Java虚拟机解释和运行的，Java虚拟机本身运行在用户态，如果要支持线程，那只能支持到use-level状态 如果一个操作系统已经支持线程，则Java虚拟机可以调用内核关于线程的函数

一些知识点

1. 操作系统是对**计算机资源**进行管理的软件
2. 从用户的观点看，OS是**用户与计算机之间的接口**
3. OS提供给程序员的接口是**系统调用**（库函数是高级汇编语言提供的，不是OS提供的）
4. 批处理系统的主要缺点是**缺少交互性**（交互性是分时系统引入的目的，正是因为批处理系统缺少交互性）
5. **虚拟机**（Virtual Machine）指通过软件模拟的具有完整硬件系统功能的、运行在一个完全隔离环境中的完整计算机系统。虚拟机的一个本质特点是运行在虚拟机上的软件被局限在虚拟机提供的资源里——它不能超出虚拟世界。对于用户而言，它只是运行在物理计算机上的一个应用程序，但是对于在虚拟机中运行的应用程序而言，它就是一台真正计算机。
6. 进程是程序的一次执行过程，是动态的．它有生命周期。这些都是进程的基本特性，进程是由程序段，数据段和PCB三部分组成。程序才是指令的集合，而进程不是。
7. 并发进程是指**宏观上可同时执行**的进程
8. 保存在PCB中的有进程标识符信息，处理机状态信息（寄存器），进程调度信息（进程标识符、进程当前状态、代码段指针、通用寄存器值）
9. 一个进程P被唤醒后，P的状态变成就绪
10. 若把操作系统看作资源管理者，**中断**不属于操作系统所管理的资源。（中断不是资源）