一、目录

1、进程基础知识

进程前言：进程知识总结

1、进程的引入
2、进程的概念
3、进程的特征
4、进程的内容
5、进程的切换
6、进程的基本状态
7、进程和程序的关系
8、进程和线程的关系
9、进程的控制

总结：
1、1-8条，是让我们了解进程概念，为了更好让我们更好的实现进程的控制。
2、嵌入式程序中，进程较少，但是应用的工具比较多，了解进程的关系，可以帮 助我们更好了解大型程序的执行过程。（实现同步，异步，协程等关系）
3、进程之间的资源分配等关系，也帮助我们在开发过程中，对嵌入式资源掌控更加精细。减少嵌入式程序的成本问题。

进程其他概念：
守护进程，交互式进程等 这些都是进程宏观的表现方式。
交互式进程：例如：shell。

一、进程：

进程是操作系统结构的基础；是一个正在执行的程序；计算机中正在运行的程序实例；

可以分配给处理器并由处理器执行的一个实体；

由单一顺序的执行显示，一个当前状态和一组相关的系统资源所描述的活动单元。

二、进程简述

windows的进程管理器

对应用程序来说，进程就像一个大容器。
在应用程序被运行后，就相当于将应用程序装进容器里了，你可以往容器里加其他东西
(如:应用程序在运行时所需的变量数据、需要引用的DLL文件等)，当应用程序被运行两次时，容器里的东西并不会被倒掉，系统会找一个新的进程容器来容纳它。

1.进程的引入:

多道程序在执行时，需要共享系统资源，从而导致各程序在执行过程中出现相互制约的关系，程序的执行表现出间断性的特征。

这些特征都是在程序的执行过程中发生的，是动态的过程，而传统的程序本身是一组指令的集合，是一个静态的概念，无法描述程序在内存中的执行情况，

即我们无法从程序的字面上看出它何时执行，何时停顿，也无法看出它与其它执行程序的关系，因此，程序这个静态概念已不能如实反映程序并发执行过程的特征。

为了深刻描述程序动态执行过程的性质，人们引入“进程（Process）”概念。

2.进程的概念:

进程的定义

进程是一个具有一定独立功能的程序关于某个数据集合的一次运行活动（简单说是程序在并发环境中的执行过程。）。

它是操作系统动态执行的基本单元，在传统的操作系统中，进程既是基本的分配单元，也是基本的执行单元。

主要两点

进程的概念主要有两点：

第一，进程是一个实体。每一个进程都有它自己的地址空间，一般情况下，包括文本区域（text region）、数据区域（data region）和堆栈（stack region）。

文本区域存储处理器执行的代码；数据区域存储变量和进程执行期间使用的动态分配的内存；堆栈区域存储着活动过程调用的指令和本地变量。

> 第二，进程是一个“执行中的程序”。程序是一个没有生命的实体，只有处理器赋予程序生命时，它才能成为一个活动的实体，我们称其为进程。

进程是操作系统中最基本、重要的概念。是多道程序系统出现后，为了刻画系统内部出现的动态情况，描述系统内部各道程序的活动规律引进的一个概念,

所有多道程序设计操作系统都建立在进程的基础上。 操作系统引入进程的概念的原因:

3.进程的特征

动态性：进程的实质是程序在多道程序系统中的一次执行过程，进程是动态产生，动态消亡的。

并发性：任何进程都可以同其他进程一起并发执行

独立性：进程是一个能独立运行的基本单位，同时也是系统分配资源和调度的独立单位；

异步性：由于进程间的相互制约，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

结构特征：进程由程序、数据和进程控制块三部分组成。

多个不同的进程可以包含相同的程序：一个程序在不同的数据集里就构成不同的进程，能得到不同的结果；但是执行过程中，程序不能发生改变

4.进程的内容

一个计算机系统进程包括（或者说“拥有”）拥有下列数据：

那个程序的可运行机器码的一个在存储器的图像。 分配到的存储器（通常包括虚拟内存的一个区域）。

存储器的内容包括可运行代码、特定于进程的数据（输入、输出）、调用堆栈、堆栈（用于保存运行时运数中途产生的数据）。

分配给该进程的资源的操作系统描述子，诸如文件描述子（Unix 术语）或文件句柄（Windows）、数据源和数据终端。

安全特性，诸如进程拥有者和进程的权限集（可以容许的操作）。 处理器状态（内文），诸如寄存器内容、物理存储器寻址等。当进程正在运行时，状态通常储存在寄存器，其他情况在存储器。

5.进程切换

Windows 和Windows Vista 体系结构

进行进程切换就是从正在运行的进程中收回处理器，然后再使待运行进程来占用处理器。

这里所说的从某个进程收回处理器，实质上就是把进程存放在处理器的寄存器中的中间数据找个地方存起来，从而把处理器的寄存器腾出来让其他进程使用。

那么被中止运行进程的中间数据存在何处好呢？当然这个地方应该是进程的私有堆栈。

让进程来占用处理器，实质上是把某个进程存放在私有堆栈中寄存器的数据（前一次本进程被中止时的中间数据）再恢复到处理器的寄存器中去，

并把待运行进程的断点送入处理器的程序指针PC，于是待运行进程就开始被处理器运行了，也就是这个进程已经占有处理器的使用权了。

这就像多个同学要分时使用同一张课桌一样，所谓要收回正在使用课桌同学的课桌使用权，实质上就是让他把属于他的东西拿走；

而赋予某个同学课桌使用权，只不过就是让他把他的东西放到课桌上罢了。

在切换时，一个进程存储在处理器各寄存器中的中间数据叫做进程的上下

文，所以进程的 切换实质上就是被中止运行进程与待运行进程上下文的切换。

在进程未占用处理器时，进程 的上下文是存储在进程的私有堆栈中的。

6.进程的基本状态

进程的三个基本状态

进程执行时的间断性，决定了进程可能具有多种状态。事实上，运行中的进程可能具有以下三种基本状态。

1）就绪状态（Ready）：

进程已获得除处理器外的所需资源，等待分配处理器资源；只要分配了处理器进程就可执行。就绪进程可以按多个优先级来划分队列。

例如，当一个进程由于时间片用完而进入就绪状态时，排人低优先级队列；当进程由I/O操作完成而进入就绪状态时，排入高优先级队列。

2）运行状态(Running)：

进程占用处理器资源；处于此状态的进程的数目小于等于处理器的数目。在没有其他进程可以执行时(如所有进程都在阻塞状态)，通常会自动执行系统的空闲进程。

3）阻塞状态(Blocked)：

由于进程等待某种条件（如I/O操作或进程同步），在条件满足之前无法继续执行。该事件发生前即使把处理机分配给该进程，也无法运行。

在很多系统中，又增加了两个基本进程状态：新建状态和终止状态；

7.进程与程序的关系

程序是指令的有序集合，其本身没有任何运行的含义，是一个静态的概念。而进程是程序在处理机上的一次执行过程，它是一个动态的概念。

程序可以作为一种软件资料长期存在，而进程是有一定生命期的。程序是永久的，进程是暂时的。

进程更能真实地描述并发，而程序不能；

进程是由进程控制块、程序段、数据段三部分组成;

进程具有创建其他进程的功能，而程序没有。

同一程序同时运行于若干个数据集合上，它将属于若干个不同的进程。也就是说同一程序可以对应多个进程。

在传统的操作系统中，程序并不能独立运行，作为资源分配和独立运行的基本单元都是进程。

8.进程与线程的关系

进程和线程关系

通常在一个进程中可以包含若干个线程，它们可以利用进程所拥有的资源。

在引入线程的操作系统中，通常都是把进程作为分配资源的基本单位，而把线程作为独立运行和独立调度的基本单位。

由于线程比进程更小，基本上不拥有系统资源，故对它的调度所付出的开销就会小得多，能更高效的提高系统内多个程序间并发执行的程度。

因而近年来推出的通用操作系统都引入了线程，以便进一步提高系统的并发性，并把它视为现代操作系统的一个重要指标。

9.进程控制

进程控制是进程管理中最基本的功能。它用于创建一个新进程，终止一个已完成的进程，或者去终止一个因出现某事件而使其无法运行下去的进程，还可负责进程运行中的状态转换。

进程的创建

1.引起创建进程的事件

在多道程序环境中，只有（作为）进程（时）才能在系统中运行。因此，为使程序能运行，就必须为它创建进程。导致一个进程去创建另一个进程的典型事件，可以有以下四类：

1）用户登录。

在分时系统中，用户在终端键入登录命令后，如果是合法用户，系统将为该终端建立一个进程，并把它插入到就绪队列中。

2）作业调度。

在批处理系统中，当作业调度程序按照一定的算法调度到某作业时，便将该作业装入到内存，为它分配必要的资源，并立即为它创建进程，再插入到就绪队列中。

3）提供服务。

当运行中的用户程序提出某种请求后，系统将专门创建一个进程来提供用户所需要的服务，

例如，用户程序要求进行文件打印，操作系统将为它创建一个打印进程，这样，不仅可以使打印进程与该用户进程并发执行，而且还便于计算出为完成打印任务所花费的时间。

4）应用请求。

在上述三种情况中，都是由系统内核为它创建一个新进程，而这一类事件则是基于应用进程的需求，由它创建一个新的进程，以便使新进程以并发的运行方式完成特定任务。

2.进程的创建过程

一旦操作系统发现了要求创建新进程的事件后，便调用进程创建原语Creat（）按下述步骤创建一个新进程。

1）申请空白PCB。为新进程申请获得唯一的数字标识符，并从PCB集合中索取一个空白PCB。

2）为新进程分配资源。为新进程的程序和数据以及用户栈分配必要的内存空间。显然，此时操作系统必须知道新进程所需要的内存大小。

3）初始化进程控制块。PCB的初始化包括：

①初始化标识信息。将系统分配的标识符和父进程标识符，填入新的PCB中；

②初始化处理机状态信息。使程序计数器指向程序的入口地址，使栈指针指向栈顶；

③初始化处理机控制信息。将进程的状态设置为就绪状态或静止就绪状态，对于优先级，通常是将它设置为最低优先级，除非用户以显式的方式提出高优先级要求。

4）将新进程插入就绪队列。如果进程就绪队列能够接纳新进程，便将新进程插入到就绪队列中。

进程终止

1.引起进程终止的事件

1）正常结束。

在任何计算机系统中，都应该有一个表示进程已经运行完成的指示。

例如，在批处理系统中，通常在程序的最后安排一条Hold指令或终止的系

统调用。当程序运行到Hold指令时，将产生一个中断，去通知OS本进程已经完成。

2）异常结束。

在进程运行期间，由于出现某些错误和故障而迫使进程终止。这类异常事件很多，常见的有：越界错误，保护错，非法指令，特权指令错，运行超时，等待超时，算术运算错，I/O故障。

3）外界干预。

外界干预并非指在本进程运行中出现了异常事件，而是指进程应外界的请求而终止运行。这些干预有：操作员或操作系统干预，父进程请求，父进程终止。

2、进程的终止过程

如果系统发生了上述要求终止进程的某事件后，OS便调用进程终止原语，按下述过程去终止指定的进程。

1）根据被终止进程的标识符，从PCB集合中检索出该进程的PCB，从中读出该进程状态。

2）若被终止进程正处于执行状态，应立即终止该进程的执行，并置调度标志为真。用于指示该进程被终止后应重新进行调度。

3）若该进程还有子孙进程，还应将其所有子孙进程予以终止，以防他们成为不可控的进程。

4）将被终止的进程所拥有的全部资源，或者归还给其父进程，或者归还给系统。

5）将被终止进程（它的PCB）从所在队列（或链表）中移出，等待其它程序来搜集信息。

进程的阻塞和唤醒

1.引起进程阻塞和唤醒的事件

1）请求系统服务。

当正在执行的进程请求操作系统提供服务时，由于某种原因，操作系统并不立即满足该进程的要求时，该进程只能转变为阻塞状态来等待，一旦要求得到满足后，进程被唤醒。

2）启动某种操作。

当进程启动某种操作后，如果该进程必须在该操作完成之后才能继续执行，则必须先使该进程阻塞，以等待该操作完成，该操作完成后，将该进程唤醒。

3）新数据尚未到达。

对于相互合作的进程，如果其中一个进程需要先获得另一（合作）进程提供的数据才能运行以对数据进行处理，则是要其所需数据尚未到达，该进程只有（等待）阻塞，等到数据到达后，该进程被唤醒。

4）无新工作可做。

系统往往设置一些具有某特定功能的系统进程，每当这种进程完成任务后，便把自己阻塞起来以等待新任务到来，新任务到达后，该进程被唤醒。 2.进程阻塞过程

正在执行的进程，当发现上述某事件后，由于无法继续执行，于是进程便通过调用阻塞原语block把自己阻塞。

可见，进程的阻塞是进程自身的一种主动行为。进入block过程后，由于此时该进程还处于执行状态，所以应先立即停止执行，把进程控制块中的现行状态由执行改为阻塞，并将PCB插入阻塞队列。

如果系统中设置了因不同事件而阻塞的多个阻塞队列，则应将本进程插入到具有相同事件的阻塞（等待）队列。

最后，转调度程序进行重新调度，将处理机分配给另一就绪进程，并进行切换，

亦即，保留被阻塞进程的处理机状态（在PCB中），再按新进程的PCB中的处理机状态设置CPU环境。

进程唤醒过程
当被阻塞的进程所期待的事件出现时，如I/O完成或者其所期待的数据已经到达，则由有关进程（比如，用完并释放了该I/O设备的进程）调用唤醒原语wakeup（ ），将等待该事件的进程唤醒。
唤醒原语执行的过程是：首先把被阻塞的进程从等待该事件的阻塞队列中移出，将其PCB中的现行状态由阻塞改为就绪，然后再将该PCB插入到就绪队列中。
进程的调度算法
进程的调度算法包括：
FIFO（First Input First Output 先进先出法）、
RR（时间片轮转算法）、
（HPF）最高优先级算法。

三、进程状态描述

1、进程映像

进程的活动是通过在CPU上执行一系列程序和对相应数据操作来体现的，因此，程序和数据是组成进程的实体。
但这二者仅是静态的文本，没有反映其动态特性。为此，还需要有一个数据结构描述进程当前的状态、本身的特性、对资源的占有及调度信息等。
这种数据结构称为进程控制块（Process Control Block,PCB）。此外，程序的执行过程必须包括一个或多个栈，用来保存过程调用和相互传送参数的踪迹。
栈按“LIFO”的方式操作。所以，进程映像通常就由程序、数据集合、栈和PCB等4部分组成。

2、进程控制块的组成

进程控制块（PCB）有时也称进程描述块（Process Descriptor），它是进程组成中最关键的部分，其中含有进程的描述停息和控制停息，是进程动态特性的集中反映，是系统对进程旅行识别和控制的依据。
在不同的系统中，PCB的具体组成成分是不同的。在简单操作系统中它较小，在大型操作系统中它很复杂，设有很多信息项。
总起来说，进程控制块一般应包括如下内容：进程名、特征信息、进程状态信息和、现场保护区、资源需求、分配和控制方面的信息、进程实体信、族系关系、其他信息；
PCB是进程存在的唯一标识；

2、守护进程创建过程

1、守护进程编程规则

在编写守护进程程序时需遵循一些基本规则，以防止产生不必要的交互作用。下面先说明这些规则， 然后给出一个按照这些规则编写的函数。

（1）调用umask将文件模式创建屏蔽字设置为一个已知值（通常是0）。

由继承得来的文件模式创建屏蔽套接字可能会被设置为拒绝某些权限。

如果守护进程要创建文件，那么它可能要设置特定的权限。

例如，若守护进程要创建组可读、组可写的文件，继承的文件模式创建屏蔽字可能会屏蔽上述两种权限中的一种，而使其无法发挥作用。

另一方面，如果守护进程调用的库函数创建了文件，那么将文件模式创建屏蔽字设置为一个限制性更强的值（如007）可能会更明智，

因为库函数可能不允许调用者通过一个显式地函数参数来设置权限。

（2）调用fork,然后使其父进程exit。这样做实现了以下几点。

第一，如果该守护进程是作为一条简单的shell命令启动的，那么父进程终止会让shell认为这条命令已经执行完毕。

第二，虽然子进程继承了父进程的进程组ID，但获得了一个新的进程ID，这就保证了子进程不是一个进程组的组长进程。

（3）调用setsid创建一个新会话。使调用进程：

（a）称为新会话的首进程

（b）称为新进程组的组长进程

（c）没有控制终端

（4）将当前目录更改为根目录。

从父进程处继承过来的当前工作目录可能在一个挂载的文件系统中。

因为守护进程通常在系统再引导之前是一直存在的，所以如果守护进程的当前工作目录在一个挂载文件系统中，那么该文件系统就不能被卸载。

（5）关闭不再需要的文件描述符。

这使守护进程不再持有从父进程继承来的任何文件描述符（父进程可能是shell进程，或某个其他进程）。

可以使用open_max函数或getrlimit函数来判定最高文件描述符值，并关闭直到该值的所有描述符。

（6）某些守护进程打开/dev/null使其具有文件描述符0,1,2,这样，任何一个试图读标准输入、写标准输出或标准错误的库例程都不会产生任何效果。

因为守护进程并不与终端设备相关联，所以其输出无处显示，也无处从交互式用户那里接收输入。

即使守护进程是从交互式会话启动的，但是守护进程是在后台运行的，所以登录会话的终止并不影响守护进程。

如果其他用户在同一终端设备上登录，我们不希望在该终端上见到守护进程的输出，用户也不期望他们在终端上的输入被守护进程读取。

2、相关操作解析：

1、将文件模式创建疲敝字设置为一个已知值（通常是0）。

设置允许当前进程创建文件或者目录最大可操作的权限，比如这里设置为0，它的意思就是0取反再创建文件时权限相与，

也就是：(~0) & mode 等于八进制的值0777 & mode了，这样给后面的代码调用函数mkdir给出最大的权限，避免了创建目录或文件的权限不确定性。

子进程继承了父进程的文件创建屏蔽字，同样也是给出mkdir最大的权限。

2、fork出子进程，然后父进程退出。

父进程终止，子进程运行。

虽然子进程继承了父进程的组ID，但是获得了一个新的进程ID，两者不可能相等，这就保证了子进程不是一个进程组的组长。这也是下面调用setsid函数的先决条件。

3、调用setsid创建一个新会话。

如果parent和child运行在同一个session里,而且parent是session头。

所以作为session头的parent如果exit结束执行的话,那么会话session组中的所有进程将都被杀死。

执行setsid()之后,parent将重新获得一个新的会话session组id,child将仍持有原有的会话session组，这时parent退出之后,将不会影响到child了

```
若成功，返回进程组ID，若失败，返回-1。
#include<unistd.h>
pid_t setsid(void);
调用setsid创建一个新会话。
使调用进程：a. 成为新会话的首进程；
            b. 成为一个新进程组的组长进程；
            c. 没有控制终端。到这里，如果创建成功，程序就基本脱离了终端的控制了，这也是我们的目的。
```

4、将当前工作目录更改为根目录。

从父进程继承过来的当前目录可能是在一个挂载的文件系统中，

因为守护进程在引导之前是一直存在的，所以如果守护进程的当前工作目录在一个挂载文件系统中，将导致该文件系统不能被卸载。

5、关闭所有不需要的文件描述符。

子进程会从父进程继承文件描述符，因为不再需要，所以通常把所有继承过来的文件描述符关闭。

这里采用getdtablesize函数可以获取到这个进程打开的文件数目。

6、使标准输入、标准输出、标准错误重定向到/dev/null。

我们不希望后台进程往终端上打印任何消息，所以把标准输入、标准输出、标准错误重定向到/dev/null，使其输出无处显示。

7、处理SIGCHLD信号。

SIGCHLD信号需要处理，特别是在高并发服务器。

因为当子进程终止时，会向父进程发送一个SIGCHLD信号，

父进程默认忽略，就会导致子进程变成一个僵尸进程，僵尸进程会占用内核中的空间，耗费进程资源。

我们可以捕捉这个信号，然后处理它，也可用简单地设置成为S忽略这个信号，正如例子所示。

3、总结：

1、改变文件掩码，即为了创建文件的权限（根据需求安排相应权限）

2、创建进程，即守护进程

3、脱离终端控制，称为进程组组长，称为新进程

4、将当前工作目录更改为根目录。

5、关闭所有不需要的文件描述符。（即标准输入输出，/dev/null，黑洞）
6、退出处理，信号处理。

3、守护进程创建实例

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <time.h>
#include <fcntl.h>
#include <string.h>
#include <sys/stat.h>
#include <signal.h>


#define ERR_EXIT(m)     \
do                      \
{                       \
    perror(m);              \
    exit(EXIT_FAILURE);     \
}                       \
while(0);               \


void creat_deamon(void)
{
    pid_t pid;
    int DevNullfd,fd,fdtablesize;

    //设置权限
    umask(0);

    //创建线程
    pid = fork();
    if(-1 == pid)
    {
        ERR_EXIT("fork error");
    }

    //退出父进程
    if(pid > 0)
    {
        exit(EXIT_SUCCESS);
```

```
    }

    if(setsid() == -1)
    {
        ERR_EXIT("setsid,error");
    }

    //设置挂在文件
    chdir("/");


    //关闭其他文件描述符
    for(fd = 0, fdtablesize = getdtablesize();
fd<fdtablesize; fd++)
    {
            close(fd);
    }
    DevNullfd = open("/dev/null",0);
    if (DevNullfd == -1) {
        ERR_EXIT("can't open /dev/null");
    }

    if (dup2(DevNullfd, STDIN_FILENO) == -1) {
        ERR_EXIT("can't dup2 /dev/null to STDIN_FILENO");
    }
    if (dup2(DevNullfd, STDOUT_FILENO) == -1) {
        ERR_EXIT("can't dup2 /dev/null to STDOUT_FILENO");
    }
    if (dup2(DevNullfd, STDERR_FILENO) == -1) {
        ERR_EXIT("can't dup2 /dev/null to STDOUT_FILENO");
    }

    signal(SIGCHLD,SIG_IGN);

}


int main(void)
{
    time_t t;
    int fd;
    creat_deamon();
    fd =
open("deamon.log",O_WRONLY|O_CREAT|O_APPEND,0664);
    if(-1 == fd)
    {
        ERR_EXIT("open error");
        return -1;
```

```
    }
    while(1)
    {
        t = time(0);
        char *buf = asctime(localtime(&t));
        if(write(fd,buf,strlen(buf))<0)
        {
            ERR_EXIT("write error");
        }
        sleep(60);
    }

    return 0;
}
```