操作系统篇

进程及其实现

进程的定义和属性

进程

进程是资源调度和分配的基本单位，线程是CPU调度和分配的基本单位。

进程提供两种虚拟机制：虚拟处理器和虚拟内存。

线程之间可以共享虚拟内存，但是每一个都有各自的虚拟处理器。

可以调用fork()来进行线程的复制，从而创建一个新的进程。调用fork()的进程为进程，新产生的进程为子进程。

内核通过一个为一个进程标识值或者PID来标识每一个进程，PID是一个数，实际上是一个int类型。

进程之间存在一个明显的继承关系。所有的进程都是PID为1的init进程的后代。

进程描述符之间有一个指向父进程的指针，和子进程的链表。

进程创建

linux创建进程的两个函数：fork()和exec()。首先fork函数通过拷贝当前进程创建一个子进程。父进程与子进程的区别仅仅在于PID、PPID和资源、统计量。

exec函数负责读取可执行文件将其载入地址空间运行。

fork函数使用写时拷贝页实现，写时拷贝是一种推迟甚至免拷贝数据的技术。内核并不复制这个那个进程地址空间，而是让父进程和子进程共享同一个拷贝。也就是只读共享，写入拷贝。

fork实际开销时复制父进程的页表以及给子进程分配唯一的进程描述符。

fork函数返回值:

负数：如果出错，则fork()返回-1,此时没有创建新的进程。最初的进程仍然运行。

零：在子进程中，fork()返回0

正数：在父进程中，fork()返回正的子进程的PID

fork函数调用一次却返回两次；向父进程返回子进程的ID，向子进程中返回0，

这是因为父进程可能存在很多过子进程，所以必须通过这个返回的子进程ID来跟踪子进程，

而子进程只有一个父进程，他的ID可以通过getppid取得。

进程的状态和转换

三状态模型

进程至少有3种状态：

1.运行态：进程占用处理器运行的状态。

2.就绪状态：进程具备运行状态，等待系统分配处理器以便其运行的状态。

3.等待态：也是阻塞态或睡眠态，也是在等待某个状态完成的状态。

进程状态转化

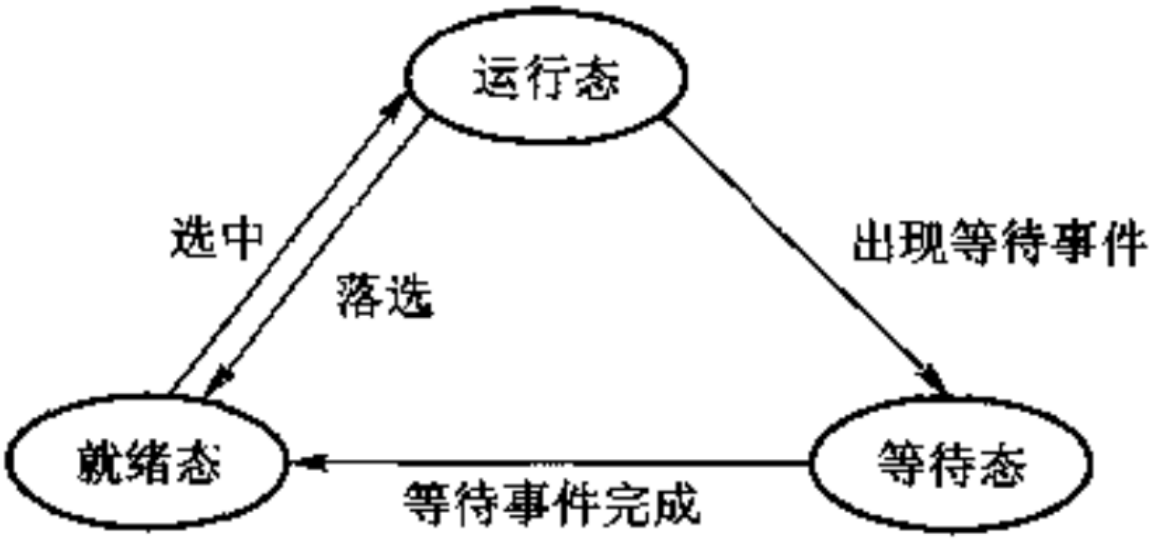


图 2.12　进程三态模型及其状态转换

进程状态转化的原因:

1.运行态—>就绪态:运行进程等待使用某种资源或某事件发生。

2.等待态——>就绪态:所需资源得到满足。

3.运行态———>就绪态:运行时间片到了，或者更高优先级的进程。

4.就绪态———>运行态：CPU空闲，选择一个就绪进程。

五状态模型

新增两个状态：新建态和终止态。

新建态：进程被创建时的状态，进程尚未进入就绪队列，创建进程需要两步：1，为新进程分配所需资源，建立表的管理信息；然后设置此进程为就绪态，等待被调度执行。

终止态：进程完成任务，到达正常结束点。

七状态模型

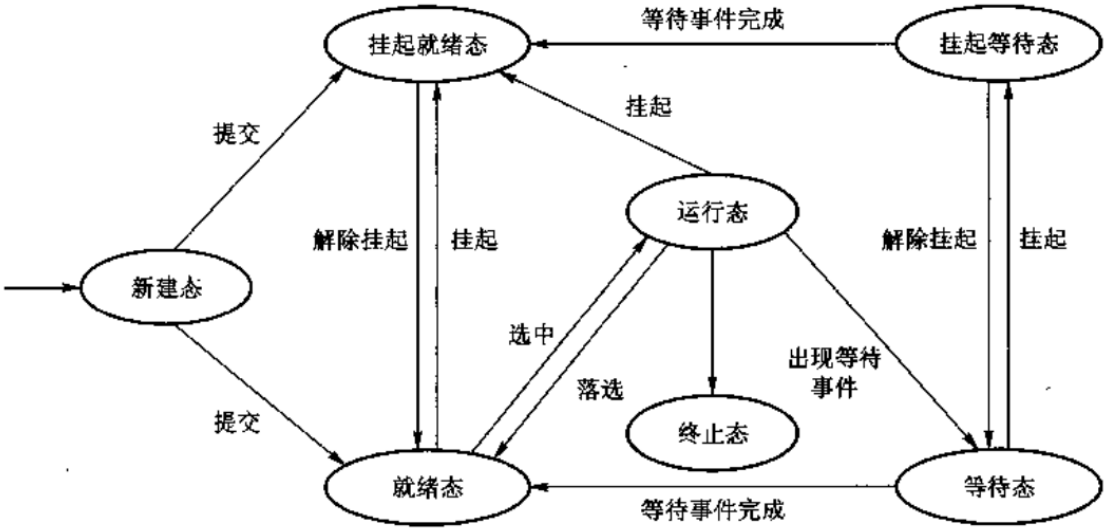


图 2.13　具有挂起进程功能的系统的进程状态及其状态转换

状态转化具体原因：

引起进程状态转换的具体原因如下。

(1) 等待态→挂起等待态：如果当前不存在就绪进程，系统根据资源分配状况和性能要求，选择等待态进程对换出去，使之处于挂起等待态。

(2) 挂起等待态→挂起就绪态：导致进程等待的事件完成后，相应的处于挂起等待态的进程将转换为挂起就绪态。

(3) 挂起就绪态→就绪态：当主存中不存在就绪态进程，或者挂起就绪态进程具有比就绪态进程更高的优先级，系统将把挂起就绪态进程换回主存并转换成就绪态。

(4) 就绪态→挂起就绪态：系统根据当前资源分配状况和性能要求，决定把就绪态进程对换出去，使之处于挂起就绪态。

(5) 挂起等待态→等待态：进程等待事件发生时，原则上无须将其调入主存，但当某些进程撤销后，主存拥有足够的自由空间，而某个挂起等待态进程具有较高的优先级，且系统得知导致它阻塞的事件即将结束，便可能发生这一类状态变化。

(6) 运行态→挂起就绪态：当一个具有较高优先级的挂起等待态进程所等待的事件完成后，它需要抢占 CPU，而此时主存空间不够，可能会导致正在运行的进程转换为挂起就绪态。另外，运行态进程也可自我挂起。

(7) 新建态→挂起就绪态：考虑系统当前资源分配状况和性能要求，决定将新建进程对换出去，使之处于挂起就绪态。

进程的描述和组成

进程映像

进程的状态在不断发生变化，某时刻进程的内容及其状态集合称为进程映像。主要包含以下要素：

1.进程控制块

存储进程标志信息、现场信息和控制信息

2.进程程序块

被执行的程序。

3.进程核心栈

保存中断 / 异常现场。

4.进程数据块

保存私有数据。

在操作系统中，进程物理试题和支持进程运行的环境合成为进程上下文。进程上下文有三个部分组成：

1.用户上下文

由正文、数据、共享存储区、用户栈组成，占用进程的虚拟地址空间。

2.寄存器上下文

指令计数器、栈指针、控制寄存器等组成。

3.系统上下文

由进程控制块、主存管理信息、核心栈组成。

线程及其实现

引入多线程的动机

为了减少程序并发执行时所付出的时空开销，是的并发力度更细、并发性更好。

线程的组成

线程时进程中能够并发执行的实体，是进程的组成部分，也是处理器调度和分配的基本单位。

进程的组成部分：

1.线程的唯一标识符以及线程状态信息

2.为运行所保存的线程上下；可以把线程堪称进程中一个独立的程序计数器。

3.核心栈，在核心态工作时保存参数，在函数调用时的返回地址

4.用于存放线程局部变量和用户栈的私有存储区。

线程的状态

运行、就绪和等待状态，没有挂起状态。

多线程的优点是：提高系统性能，具体表现在快速线程切换，节省主存空间，减少管理开销；通信易于实现；并发程度高。

线程的实现

多线程的实现分为三类：用户级线程、内核级线程和混合方式。

内核级线程

线程的管理的工作由内核完成，内核为其创建进程和一个基线程。

优点：同时调度同一进程中的多个线程并发执行；运行速度快，执行效率高。

缺点：模式切换系统开销较大。

用户级线程

线程的管理是应用程序完成。

优点：线程切换无须使用内核特权方式。

缺点：一个用户线程的阻塞引起整个进程的组塞。

混合式线程

即支持用户级线程，有支持内核级线程。

处理器调度

处理器调度的层次

1.高级调度

作业调度、长程调度，在多道批处理操作系统中，从输入系统的一批作业中按照预定的调度策略挑选若干的作业进入主存，为其分配所需的资源，并创建作恶的相应用户进程后便完成启动阶段的高级调度任务。

2.中级调度

平衡调度、中程调度，根据贮存资源决定主存中所能容纳的进程数量，并根据进程的当前状态来决定辅助存储器的主存中的进程的兑换。

3.低级调度

进程调度、线程调度、短程调度。主要的功能是根据某种原则决定就绪队列中的那些进程 / 内核级线程获得处理器，并将处理器出让给它使用。

处理器调度算法

低级调度的最初对象时进程。

低级调度的基本类型

剥夺式算法

非剥夺式

低级调度算法

先来先服务(FCFS)

非剥夺式调度算法，易于实现，但效率不高。

最短作业优先算法(SJF)

非剥夺式调度算法，易于实现，但执行效率不高。

优先级调度算法

根据线程的优先级来运行。

轮转调度算法

也是时间片调度。

CFS公平调度算法

Linux的CFS调度器并没有直接分配时间片到进程，它是将处理器的使用比划分给进程。这样以来，进程所获取的处理器时间其实是和系统负载密切相关的。这个比例进一步还会收到进程nice值的影响，nice值作为权重将调整进程所使用的处理器时间比，具有更高nice值(更低优先权)的进程将被赋予低权重，从而丧失一部分的处理器使用比，而具有更小nice值(更高优先级)的进程则会被赋予高权重，从而抢占更多的处理器使用比。

内核同步介绍

临界区和竞争条件

临界区就是访问和操作共享数据的代码段。

避免并发和竞争条件称为同步。

管程:把分散在各个进程中的临界区集中起来管理，并把共享资源用数据结构抽象地表达出来，由于临界区是访问共享资源的代码段，建立一个秘书来进行管理。这个秘书就是管程。

Linux同步机制

1.原子操作

原子操作在执行的过程中不会被打断，以防止简单的竞争条件发生，确保操作结果的正确性，复杂的锁机制能够在原子操作的基础上构建。

2.内核信号量

在Linux中使用等待队列来实现内核信号量。

3.等待队列

并发进程同步时，只要等待条件不满足时，就必须挂起，放入相应的等待队列。

4.关中断

关中断是把内核态执行的程序作为一个临界区来保护的一种手段，主要保护中断处理程序也要访问的数据结构。

5.自旋锁

自旋锁是最简单的一种锁原语，锁的取值是0表示资源可用，锁的取值是1表示资源加锁。

加锁

Linux自身实现几种不同的锁机制，各种锁之间的区别:当锁已经被其他线程持有，一些锁被争用时简单的执行忙等待，而另一些锁会使当前任务睡眠直到锁可用为止。

造成并发执行的原因

中断:中断几乎可以在任何时刻异步发生，也就可能随时打断当前正在执行的代码。

软中断和tasklet:内核能在任何时刻唤醒或调度软中断和tasklet,打断当前正在执行的代码.

内核抢占:因为内核具有抢占性，所以内核的任务可能被其他任务抢占。

睡眠及用户空间的同步:在内核的进程可能随眠，这就会唤醒调度程序，从而导致调度一个新的用户进程执行。

对称多处理:两个和多个处理器可以同时执行代码。

在中断处理程序中能避免并发访问的安全代码称作为中断安全码，在对称多处理的机器中能避免并发访问的安全代码称为SMP安全代码，在内核抢占时能避免并发访问的安全代码称之为抢占安全代码。

了解需要保护什么

大多数内核数据结构需要加锁!

给数据加锁而不是给代码加锁

死锁

死锁产生的条件:

1.互斥条件

2.占用等待条件

3.不可剥夺

4.循环等待

防止死锁的策略

预防死锁、避免死锁、检测死锁、解除死锁

进程通信

进程之间互相交换信息的工作称为进程通信

进程通信的方式:

1.信号通信机制

2.管道通信通信机制

3.消息传递通信机制

4.信号量通信机制

5.共享内存通信机制

信号通信机制

信号是一种软中断，是传递短消息的简单通信机制，通过发送指定信号来通知进程某个异步事件发生，以迫使进程执行信号处理程序。

管道通信机制

管道是连续读写进程的一个特殊的文件，按照FCFS方式传递数据，也能使进程同步执行。管道是单向的，发送进程视管道为输出文件，以字符流的形式将数据发送到管道，接受进程视管道为输入文件，从管道接受数据，所以称之为管道通信。

管道的实质是一个共享文件，即利用辅助存储器来进行数据通信。

共享内存通信机制

共享内存是指在主机中开辟一个公用存储区、要通信的进程把自己的虚拟地址空间映射到共享内存区。

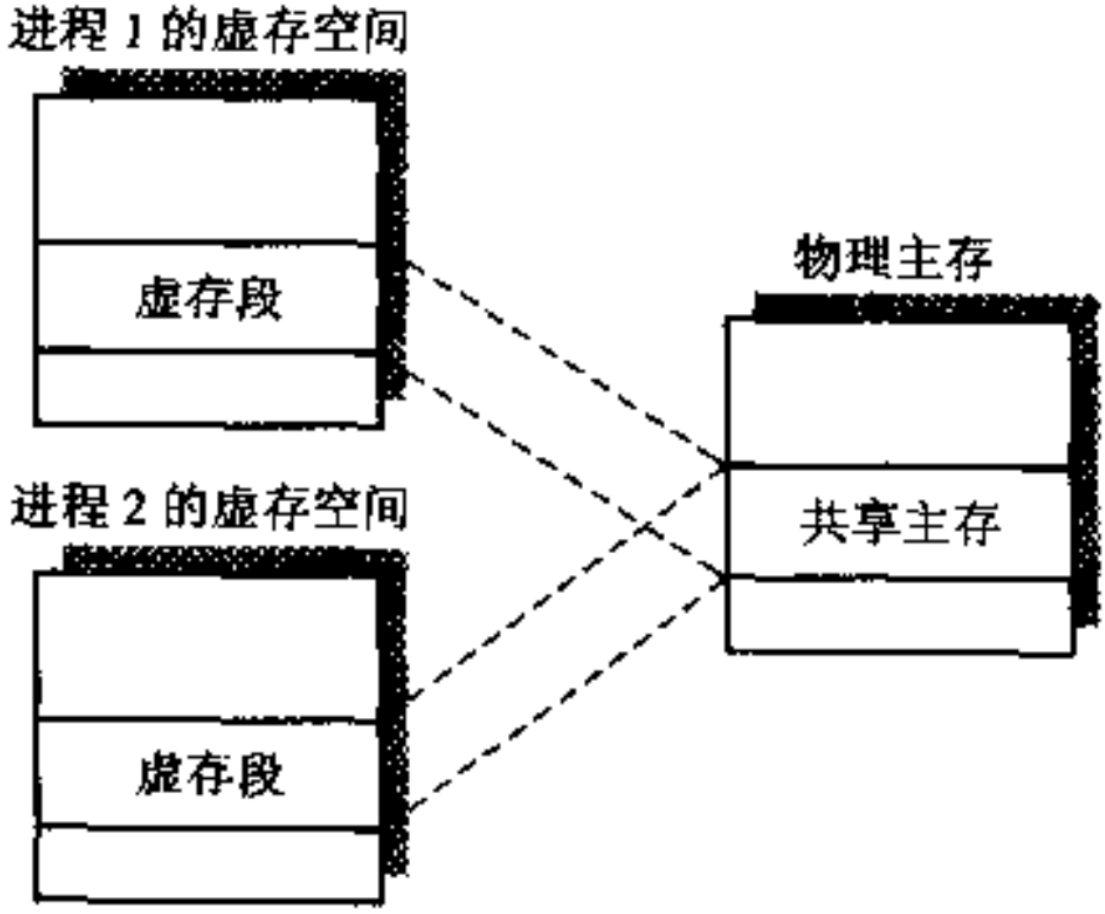


图 3.6　共享主存通信机制

消息传递通信机制

为了实现异步通信，必须采用间接通信方式，进程之间发生或者接受消息通过一个共享的数据结构——信箱进行，消息可被理解成信件，每一个信箱都有一个唯一标识符。

Linux同步机制和通信机制

Linux内核同步机制

1.原子操作

原子操作保证在执行过程中不被打断，以防止简单的竞争条件的发生，以确保操作的正确性，复杂的锁机制可以在原子操作的基础上构建。Linux内核定义两类原子操作 :

原子整型操作

原子位图操作:针对指针变量所指定的任意一块主存区域的位序列进行操作。

2.内核信号量

在Linux操作系统中，也使用等待队列中实现内核使用的信号量机制。

3.等待队列

Linux内核信号量采用非忙式等待来实现，当进程中执行DOWN操作而等待时，将放入等待信号量队列；事实上并发进程同步时，只要等待条件不满足时，就必须挂起，放入相应的等待队列。等待队列也是临界资源，对其进行修改时应加锁，以避免竞争条件。

4.关中断

关中断是把内核态执行的程序段作为一个临界区来保护的一种手段，主要保护中断处理程序也要访问的数据结构。

5.自旋锁

自旋锁是最简单的一种锁原语，锁的取值为0表示资源可用，锁的取值为1表示资源加锁。自旋锁很像二元信号量，但在具体实现还是有区别的。若一个资源得到自旋锁的保护，另一个试图取得此资源的内核例程将保护忙等待，直到资源解锁。

IPC机制

在Linux系统中，把信号量、消息队列和共享内存资源称为IPC(进程间通信)资源，用于进程间通信。Linux提供给用户的IPC资源是通过系统调用来实现的，它们为进程提供三种服务:

1.为信号量对进程所要访问临界资源进行保护

2.用消息队列在进程之间以异步方式发送和接收消息

3.预留共享内存段以供进程之间交换和共享数据

IPC和管道之间的区别 是，它允许同一个机器上的许多进程进行通信，而不是限于两个进程，且管道有一个限制，两个进程必须是相关的，他们必须有共同的祖先进程。

存储管理

主存空间一般分为两部分:一部分是系统区，用于存放操作系统的内核程序和数据结构等，另一部分是用户区，用于存放应用程序和数据。

存储管理包含一下功能:

1.分配与去配

2.抽象和映射

3.隔离与共享

4.存储与扩充

存储器

寄存器、高速缓存、主存储器、磁盘、磁带，按这个顺序速度越来越低，那个也越来越低

利用高速缓存来存放主存中经常访问的信息，以提高程序执行速度。

地址转换与存储保护

编译:源程序通过编译程序或汇编程序的处理来获得目标代码。

链接:把多个模块连接成为一个完整的可重定位程序，需要解析内部和外部符号表，把符号名的引用转化为数值引用，要将涉及名字地址的程序入口点和数据引用点转换为数值地址。

装入:在装载一个加载代码模块之前，存储管理程序总会分配一块实际主存区给进程，装入程序根据指定的主存区首地址，再次修改和调整被装载模块中的逻辑地址，将逻辑地址绑定为物理地址，使之成为可执行二进制代码。

物理主存储器从此那个统一的基地地址开始顺序编止的存储单元称之为物理地址或绝对地址，物理地址的总体构成物理地址空间。

把逻辑地址转化为物理地址的果醋鞲称之为地址重定位、地址映射或地址转换有以下两种方式。

静态地址重定位

由装入程序实现装载代码模块的加载和地址转换，把它装入分配给进程的主存指定区域，其中的所有逻辑地址修改成为主存地址，称静态重定位。

动态地址重定位

由装入程序实现装载代码模块的加载，把它装入分配给进程的主存指定区域，但对连接程序处理过的应用程序的逻辑地址则不做任何修改，程序主存起始地址被置入硬件专用寄存器---重定位寄存器。

连续存储空间管理

固定分配管理

固定分区存储管理的基本思想:主存空间被分成数量固定不变的分区，各分区的大小不等，每个分区只装入一个作业，若多个分区中都装有作业，则就可以并发执行，这就是支持多到程序设计的最简单的存储管理技术。

可变的分区存储管理

可变分区存储管理又称为动态分五模式，按照作业的大次奥来划分分区，但划分的时间、大小、位置都是动态。系统把作业装入主存时，根据起其需要的主存容量额查看是否又足够的空间，若有，则需要按需分配一个分区分配给作业；若无，则令作业等待主存资源。

可变分区分配算法:

1.最优适应分配算法

顺序查找未分配区表或链表，直到找到第一个满足长度要求的空闲为止，分割此分区，一部分分配给作业，另一部分仍为空闲区。此阿勇这一分配算法时，未分配区表或链表的空闲区通常按地址从小到大排序。

2.下次适应分配算法

总是从未分配区的上次扫描结束处顺序查找分配区表或链表，直到找到第一个能够满足长度要求的空闲区为止，分割这个未分配区，一部分分配给作业，另一部分仍为空闲区。

3.最优适应分配算法

扫面这个整个未分配区表或链表，从空闲区中挑选一个能满足用户进程要求的最小分区进程分配。

4.最坏适应分配算法

扫面整个未分配区表或链表，总是挑选一个最大的空闲分割给作业使用，其有显示使剩下的空闲区不至过小，对中小型走也有利。

5.快速适应分配算法

为那些经常用到的长度的空闲区独立设立单独的空闲区链表。

分页存储管理

采用分页存储管理允许程序存放到若干不相邻的空闲块中，即可免除移动信息动作，有可充分利用主存空间，消除动态分区算法中的"碎片"问题，从而提高主存空间的利用率。

1.页面

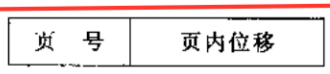
进程逻辑地址空间分成大小相等的区，每个区成为页面或者页，页号从0开始一次编号

2.页框

也称为页帧，把主存物理地址分配大小相等的区，其大小与页面大小相等，每个区是一个物理块或页框，快号从0开始依次编号

3.逻辑地址

分页存储器的逻辑地址由两部分组成:页号和页内位移



前面部分表示地址所在页面的编号，后面部分表示页内位移。计算机地址总线通常是 32 位，页面尺寸若规定为 12 位（页长 4 KB），那么，页号共 20 位，表示地址空间最多包含 2^{20} 个页面。

页号p和页内位移d，先从页表基址寄存器找到页面基地址，在用页号p作为索引查页表

物理地址 = 页框号 × 块长 + 页内位移

计算出欲访问的主存单元。因此，虽然进程存放在若干不连续的页框中，但在执行过程中总能按正确的物理地址进行存取。

如图 4.10 所示是分页存储管理的地址转换，在实际进行地址转换时，只要把逻辑地址中的页内位移 d 作为绝对地址中的低地址，根据页号 p 从页表中查得页框号 b 作为绝对地址中的高地址，就组成访问主存储器的绝对地址。

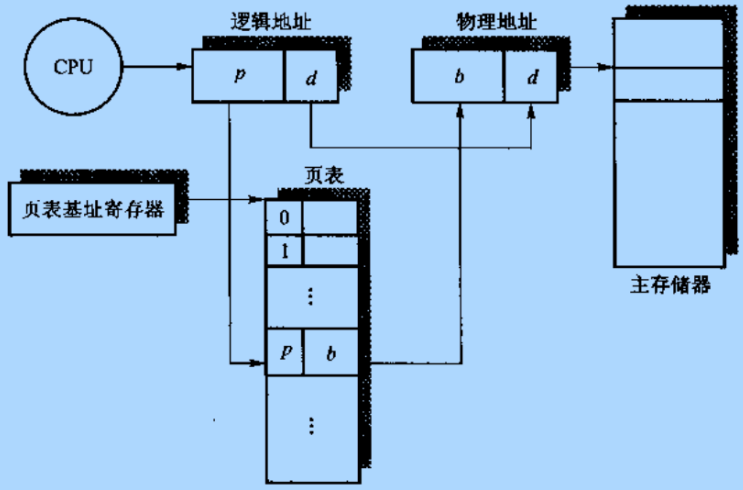


图 4.10 分页存储管理的地址转换

分段存储管理

分段存管理的引入主要是满足用户编程和使用上的要求，其他存储管理技术难以满足这些要求。

分段存储管理把进程的逻辑地址空间分成多段，提供如下形式的二维逻辑地址：

段 号	段内位移

在分页存储管理中，页的划分，即逻辑地址划分为页号和页内位移，是用户不可见的，连续的地址空间将根据页面的大小自动分页；而在分段存储管理中，地址结构是用户可见的，用户知道逻辑地址如何划分为段和段内位移，在设计程序时，段的最大长度由地址结构规定，程序中所允

分段和分页的比较

分段是信息的逻辑单位由源程序的逻辑结构及含义所决定，是用户可兼得，段长由用户根据要求确定，段起始地址可从主存地址开始。在分段方式中，源程序经链接装配后仍保持二维结构，引入目的是满足用户模块化程序的需求。

分页是信息的物理单位与源程序的逻辑结构分配，是用户不可见的，页长由系统确定，页面只能从也带下的整数倍地址开始。

页面置换算法

页面置换：在地址映射过程中，若在页面中发现所要访问的页面不在内存中，则产生缺页中断(page fault)。当发生缺页中断时操作系统必须在内存选择一个页面将其移出内存，以便为即将调入的页面让出空间。

OPT：最佳替换算法（optional replacement）。替换下次访问距当前时间最长的页。opt算法需要知道操作系统将来的事件，显然不可能实现，只作为一种衡量其他算法的标准。

LRU:最近最少使用(Least Recently Used).替换上次使用距离当前最远的页。根据局部性原理：替换最近最不可能 访问到的页。性能最接近OPT，但难以实现。可以维护一个关于访问页的栈或者给每个页添加最后访问的时间标签，但开销都很大。

FIFO:先进先出(First In First Out),将页面看做一个循环缓冲区，按循环方式替换。这是实现最为简单的算法，隐含的逻辑是替换驻留在内存时间最长的页。但由于一部分程序或数据在整个程序的生命周期中使用频率很高，所以会导致反复的换入换出。

常见面试题

（一）请分别简单说一说进程和线程以及它们的区别。

1.进程是具有一定功能的程序关于某个数据集合上的一次运行活动，进程是系统进行资源调度和分配的一个独立单位。

2.线程是进程的实体，是CPU调度和分派的基本单位，它是比进程更小的能独立运行的基本单位。

3.一个进程可以有多个线程，多个线程也可以并发执行

（二）线程同步的方式有哪些?

互斥量：采用互斥对象机制，只有拥有互斥对象的线程才有访问公共资源的权限。因为互斥对象只有一个，所以可以保证公共资源不会被多个线程同时访问。

信号量：它允许同一时刻多个线程访问同一资源，但是需要控制同一时刻访问此资源的最大线程数量。

事件（信号）：通过通知操作的方式来保持多线程同步，还可以方便的实现多线程优先级的比较操作。

（三）进程的通信方式有哪些?

主要分为：管道、系统**IPC**（包括消息队列、信号量、共享存储）、**SOCKET**

管道主要分为：普通管道**PIPE**、流管道（**s_pipe**）、命名管道（**name_pipe**）

管道是一种半双工的通信方式，数据只能单项流动，并且只能在具有亲缘关系的进程间流动，进程的亲缘关系通常是父子进程

命名管道也是半双工的通信方式，它允许无亲缘关系的进程间进行通信

信号量是一个计数器，用来控制多个进程对资源的访问，它通常作为一种锁机制。

消息队列是消息的链表，存放在内核中并由消息队列标识符标识。

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。

共享内存就是映射一段能被其它进程访问的内存，这段共享内存由一个进程创建，但是多个进程可以访问。

（四）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填充数据时超出了缓冲区本身的容量，溢出的数据覆盖在合法数据上。

危害有以下两点：

程序崩溃，导致拒绝额服务

跳转并且执行一段恶意代码

造成缓冲区溢出的主要原因是程序中没有仔细检查用户输入。

（五）什么是死锁？死锁产生的条件？

在两个或者多个并发进程中，如果每个进程持有某种资源而又等待其它进程释放它或它们现在保持着的资源，在未改变这种状态之前都不能向前推进，称这一组进程产生了死锁。通俗的讲就是两个或多个进程无限期的阻塞、相互等待的一种状态。

死锁产生的四个条件（有 一个条件不成立，则不会产生死锁）

互斥条件：一个资源一次只能被一个进程使用

请求与保持条件：一个进程因请求资源而阻塞时，对已获得资源保持不放

不剥夺条件：进程获得的资源，在未完全使用完之前，不能强行剥夺

循环等待条件：若干进程之间形成一种头尾相接的环形等待资源关系

（六）进程有哪几种状态？

就绪状态：进程已获得除处理机以外的所需资源，等待分配处理机资源

运行状态：占用处理机资源运行，处于此状态的进程数小于等于CPU数

阻塞状态： 进程等待某种条件，在条件满足之前无法执行

（七）分页和分段有什么区别？

段是信息的逻辑单位，它是根据用户的需要划分的，因此段对用户是可见的 ； 页是信息的物理单位，是为了管理主存的方便而划分的，对用户是透明的。

段的大小不固定，有它所完成的功能决定；页大大小固定，由系统决定

段向用户提供二维地址空间；页向用户提供的是一维地址空间

段是信息的逻辑单位，便于存储保护和信息的共享，页的保护和共享受到限制。

（八） 操作系统中进程调度策略有哪几种？

FCFS(先来先服务)，优先级，时间片轮转，多级反馈

（九）说一说进程同步有哪几种机制。

原子操作、信号量机制、自旋锁管程、会合、分布式系统

（十）说一说死锁的处理基本策略和常用方法。

解决死锁的基本方法如下：

预防死锁、避免死锁、检测死锁、解除死锁

解决四多的常用策略如下：

鸵鸟策略、预防策略、避免策略、检测与解除死锁

（十一）孤儿进程和僵尸进程

孤儿进程：一个父进程退出，而它的一个或多个子进程还在运行，那么那些子进程将成为孤儿进程。孤儿进程将被init进程(进程号为1)所收养，并由init进程对它们完成状态收集工作。

僵尸进程：一个进程使用fork创建子进程，如果子进程退出，而父进程并没有调用wait或waitpid获取子进程的状态信息，那么子进程的进程描述符仍然保存在系统中。这种进程称之为僵死进程。

（十二）fork函数的返回值

fork函数返回值:

负数：如果出错，则fork()返回-1,此时没有创建新的进程。最初的进程仍然运行。

零：在子进程中，fork()返回0

正数：在父进程中，fork()返回正的子进程的PID

fork函数调用一次却返回两次；向父进程返回子进程的ID，向子进程中返回0，

这是因为父进程可能存在很多过子进程，所以必须通过这个返回的子进程ID来跟踪子进程，

而子进程只有一个父进程，他的ID可以通过getppid取得。

（十三）软中断和硬中断

软中断和硬中断的区别?

从本质上来讲，中断是一种电信号，当设备有某种事件发生时，它就会产生中断，通过总线把电信号发送给中断控制器。如果中断的线是激活的，中断控制器就把电信号发送给处理器的某个特定引脚。处理器于是立即停止自己正在做的事，跳到中断处理程序的入口点，进行中断处理。

(1) 硬中断

由与系统相连的外设(比如网卡、硬盘)自动产生的。主要是用来通知操作系统系统外设状态的变化。比如当网卡收到数据包的时候，就会发出一个中断。我们通常所说的中断指的是硬中断(hardirq)。

(2) 软中断

为了满足实时系统的要求，中断处理应该是越快越好。linux为了实现这个特点，当中断发生的时候，硬中断处理那些短时间就可以完成的工作，而将那些处理事件比较长的工作，放到中断之后来完成，也就是软中断(softirq)来完成。