操作系统与组原

1. 虚拟内存和物理内存的区别

1）物理内存

以前，还没有虚拟内存概念的时候，程序寻址都用的是物理寻址。而物理寻址的范围是十分有限的，这取决于 CPU 的地址线条数。比如在 32 位的机器上，寻址范围是 2^32，也就是 4G。并且，这是固定的，如果每开一个进程都给它们分配 4G 物理内存，那资源消耗就太大了。

况且，资源的利用率也是一个巨大的问题。没有分配到资源的进程就只能等待，当一个进程结束以后再把等待的进程装入内存，而这种频繁地装入内存操作效率也很低。

并且，由于指令都是可以访问物理内存的，那么任何进程都可以修改内存中其它进程的数据，甚至修改内核地址空间的数据，这是非常不安全的。

2）虚拟内存

由于物理内存使用时，资源消耗大、利用率低及不安全的问题。因此，引入了虚拟内存。

虚拟内存是计算机系统内存管理的一种技术，通过分配虚拟的逻辑内存地址，让每个应用程序都认为自己拥有连续可用的内存空间。而实际上，这些内存空间通常是被分隔开的多个物理内存碎片，还有部分暂时存储在外部磁盘存储器上，在需要时进行数据交换。

2. 逻辑地址如何形成物理地址（内存管理）

Linux 操作系统采用的是段页式内存管理方式：

- 页式存储能有效地解决内存碎片，提高内存利用率
- 分段式存储管理能反映程序的逻辑结构，并有利于段的共享

而段页式存储管理方式，就是将这两种存储管理方法结合起来所形成的，它是先把用户程序分成若干个段，为每一个段分配一个段名，再把每个段分成若干个页。

在段页式系统中，为了实现从逻辑地址到物理地址的转换，系统中需要同时配置段表和页表，利用段表和页表进行从用户地址到物理内存空间的映射。系统为每个进程创建一张段表，每个分段上有一个页表。段表包括段号、页表长度和页表始址，页表包含页号和块号。

在地址转换时，首先通过段表查到页表地址，再通过页表获取页帧号，最终形成物理地址。

3. 进程调度

1）进程的五种状态

进程分为**创建态，就绪态，运行态，阻塞态，结束态**。其中，由就绪态转化为运行态就是进程的调度。操作系统管理了系统的有限资源（比如处理机的个数），当多个进程要使用这些资源时，必须按照一定的原则选择某个进程来执行任务，占用资源。

进程调度是操作系统的核心，完成进程的状态转换，并设置进程的状态参数，是由交通控制程序和进程调度程序来完成的。分为三个步骤：

1. 记录系统中所有进程的状态，优先数和资源的请求情况；
2. 确定调度算法；
3. 为进程分配处理机。

2）进程的调度方式

进程的调度方式分为抢占式和不可抢占式：

- 抢占式：在某些条件下，系统可以强制剥夺正在运行的进程使用处理机的权利，并将处理机分配给另外一个合适的进程；
- 不可抢占式：一个进程在获得处理机以后，除非运行结束或者进入阻塞状态等原因主动放弃CPU，否则可以一直运行下去。

3）进程调度算法

调度算法解决了如何进行进程调度的问题：比如调度次序，对处理机占有的时间和比例。常见的进程调度算法有：

1. 先来先服务（FCFS）算法
2. 最短作业优先算法
3. 时间片轮转调度算法
4. 高响应比优先
5. 最高优先级调度法
6. 多级反馈调度法

1.先来先服务算法

思想：按照进程进入就绪队列的时间顺序来分配 CPU。

特点：

- 不可抢占式，一旦进程占用了 CPU，除非运行结束或阻塞主动放弃 CPU，否则其它进程就只能等待；
- 处在就绪队列头部的进程首先获得 CPU，一旦主动释放，要么进入阻塞状态，要么就直接挂在就绪队列的尾部。

缺点：

- 当运行大作业时，会让后面的小作业等待很久，会增加作业的平均等待时间；
- 对于 IO 繁忙的进程，每进行一次 IO 都有等待其它进程运行一个周期后才能再次获取 CPU 资源，大大延长了这类作业的运行时间，也不能有效地利用各种外部资源；
- 不能为紧急进程优先分配 CPU。

2.最短作业优先调度

思想：优先选择运行时间最短的进程来运行，有助于提高系统的吞吐量。

缺点：对长作业的进程太不利了，容易造成一种极端现象，比如一个长作业在就绪队列等待很久，但由于短作业进程很多，所以长作业迟迟没法运行，会增大进程的平均等待时间。

3.时间片轮转策略

思想：

- 各进程运行一小段时间，这段时间被称为 **时间片；**
- 在时间片内，如果进程运行完任务，或者 IO 等原因进入阻塞状态，该进程就提前让出 CPU；

- 当进程耗费完一个时间片后任务未执行完毕，也需要主动放弃处理机，并重新排列到就绪队列的尾部。

特点：

- 剥夺式调度，当时间片用完后，即便当前进程未完成任务，也会被剥夺 CPU；
- 时间片轮转适用于交互式分时系统；
- 系统的效率与时间片大小的设置有关：若时间片过大，系统和用户间的交互性就差，用户响应长；时间片过小，进程切换频繁，系统开销增大；

优化：

- 将时间片分成多个规格，比如 10ms，20ms，50ms 等；
- 按时间片大小，将就绪进程排成多个队列；
- 把交互性强和调度频率高的进程排入小时间片队列，计算性强和需要连续占用处理机的进程排在长时间队列，可提高系统的响应速度，并减少进程切换的开销；
- 根据运行状况，进程可以从小时间片队列转入长时间片队列。

4.高响应比优先调度

思想：通过响应比的公式【优先权重 = （等待时间+作业的时间）/ 作业的时间】，充分考虑短作业和长作业进程，高响应比的进程优先执行。

特点：

- 当两个进程的等待时间相同时，作业的时间越短，响应比就越高，这样短作业的进程容易被选中运行；
- 当两个进程的作业时间相同时，等待时间越长，响应比也越高，这样就兼顾到了长作业进程。

5.最高优先级调度算法

思想：

- 根据进程的重要和紧迫程度，系统为每个进程设置一个优先级；
- 调度程序总是从就绪队列中挑选出一个优先级最高的进程，让它占用 CPU 并运行。

优先级分为：

- 静态优先级，创建进程的时候就确定好，运行的整个过程优先级都不会发生变化；
- 动态优先级，根据进程的动态变化调整优先级，比如进程运行的时间增加，就降低其优先级；如果进程的等待时间增加，就升高其优先级。

缺点：可能导致低优先级的进程永远不会执行。

6.多级反馈队列调度

思想：结合了时间片轮转和最高优先级调度算法形成。

- **多级**代表有多个队列，每个队列的优先级从高到低，而优先级越高的时间片越短；
- **反馈**表示若有新的进程加入高优先级的队列时，先停止运行当前进程，转而去执行优先级高的进程；

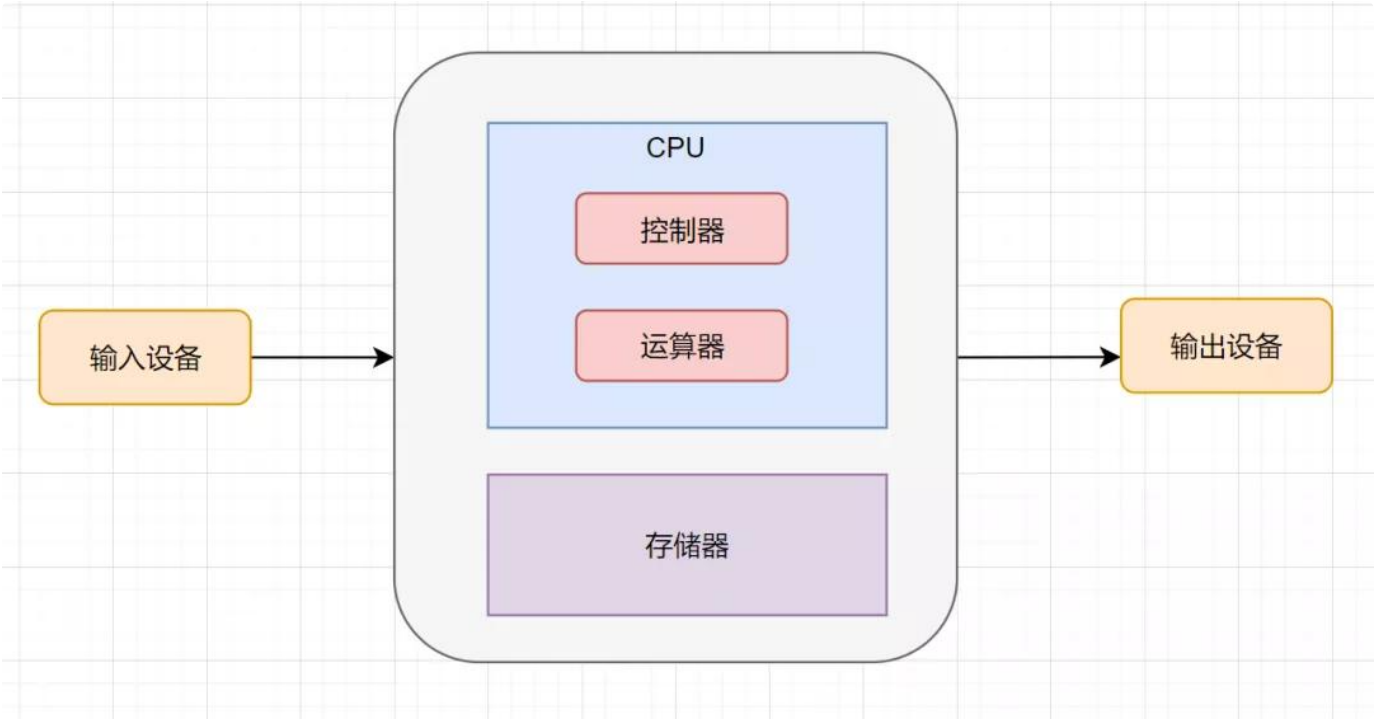
特点：

- 就绪进程分成多个级别队列【多级】，新进程放入第一级队列末尾，按先来先服务的原则排队等待调度。如果在第一级队列规定的时间片没运行完成，则将其转入第二级队列的末尾，以此类推，直至运行完成；
- 当较高优先级的队列为空，才调度低优先级队列中的进程。如果进程运行时，有新进程进入到较高优先级队列，则【反馈】。
- 此算法兼顾短作业和长作业，对于短作业来说，可能在第一级队列就很快处理完了；长作业虽然在高优先级队列没法处理完，但可以移入较低优先级中等待调度，虽然等待时间变长了，但是进程运行的时间也会更长。

4. 网络 IO 模型

1）IO简介

我们常说的 IO，比较直观的意思就是计算机的输入输出，计算机就是主体。大家是否还记得，大学学计算机组成原理的时候，有个冯.诺依曼结构，它将计算机分成分为5个部分：运算器、控制器、存储器、输入设备、输出设备。



输入设备是向计算机输入数据和信息的设备，键盘，鼠标都属于输入设备;输出设备是计算机硬件系统的终端设备，用于接收计算机数据的输出显示，一般显示器、打印机属于输出设备。

例如你在鼠标键盘敲几下，它就会把你的指令数据，传给主机，主机通过运算后，把返回的数据信息，输出到显示器。

鼠标、显示器这只是直观表面的输入输出，回到计算机架构来说，涉及计算机核心与其他设备间数据迁移的过程，就是 IO。如磁盘 IO，就是从磁盘读取数据到内存，这算一次输入，对应的，将内存中的数据写入磁盘，就算输出。这就是 IO 的本质。

操作系统负责计算机的资源管理和进程的调度。我们电脑上跑着的应用程序，其实是需要经过操作系统，才能做一些特殊操作，如磁盘文件读写、内存的读写等等。

因为这些都是比较危险的操作，不可以由应用程序乱来，只能交给底层操作系统来。也就是说，你的应用程序要把数据写入磁盘，只能通过调用操作系统开放出来的 API 来操作。

2）用户空间和内核空间

以 32 位操作系统为例，它为每一个进程都分配了 4G(2的32次方) 的内存空间。这 4G 可访问的内存空间分为二部分，一部分是用户空间，一部分是内核空间。

内核空间是操作系统内核访问的区域，是受保护的内存空间，而用户空间是用户应用程序访问的内存区域。

我们应用程序是跑在用户空间的，它不存在实质的 IO 过程，真正的 IO 是在操作系统执行的。即应用程序的 IO 操作分为两种动作：IO 调用和 IO 执行。IO 调用是由进程(应用程序的运行态)发

起，而 IO 执行是操作系统内核的工作。此时所说的 IO 是应用程序对操作系统 IO 功能的一次触发，即 IO 调用。

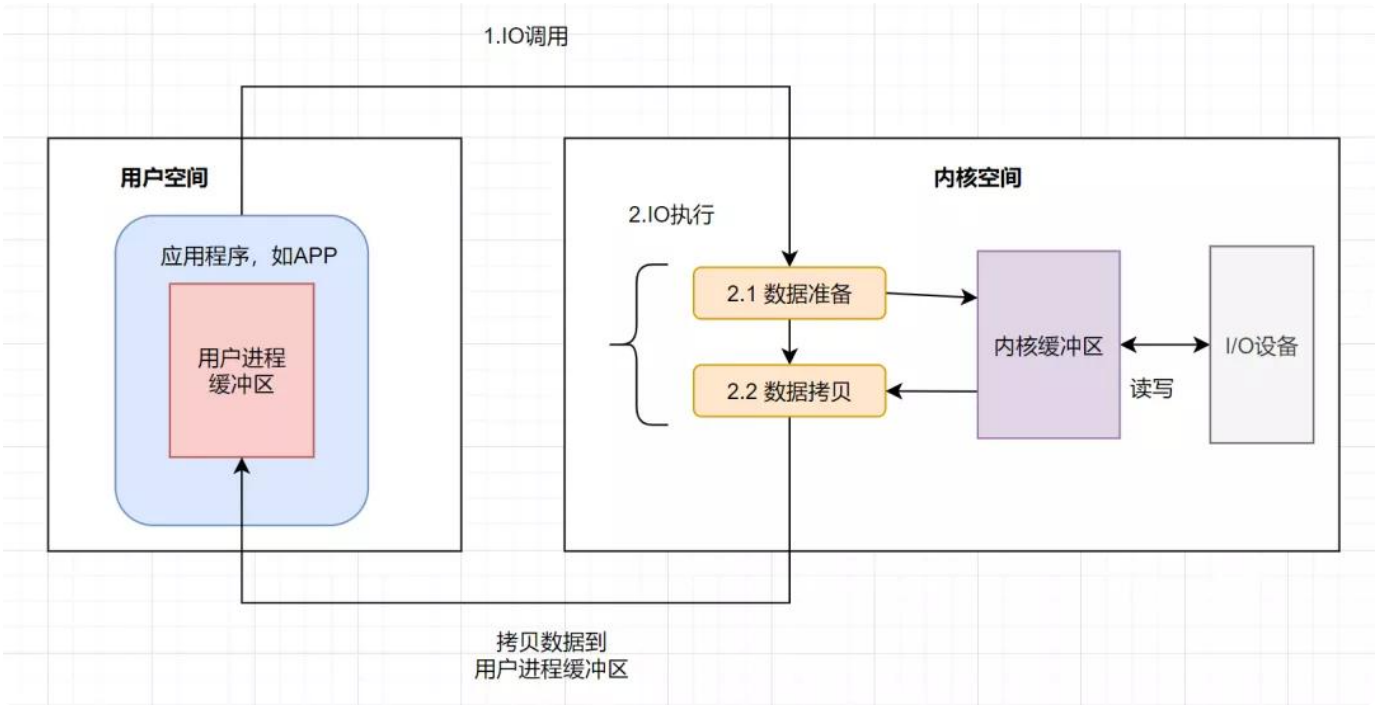
3）一次IO过程

应用程序发起的一次 IO 操作包含两个阶段：

- IO 调用：应用程序进程向操作系统内核发起调用。
- IO 执行：操作系统内核完成 IO 操作。

操作系统内核完成 IO 操作还包括两个过程：

- 准备数据阶段：内核等待 I/O 设备准备好数据
- 拷贝数据阶段：将数据从内核缓冲区拷贝到用户进程缓冲区



其实 IO 就是把进程的内部数据转移到外部设备，或者把外部设备的数据迁移到进程内部。外部设备一般指硬盘、socket 通讯的网卡。一个完整的 IO 过程包括以下几个步骤：

- 应用程序进程向操作系统发起 IO 调用请求
- 操作系统准备数据，把 IO 外部设备的数据，加载到内核缓冲区
- 操作系统拷贝数据，即将内核缓冲区的数据，拷贝到用户进程缓冲区

4）网络IO模型

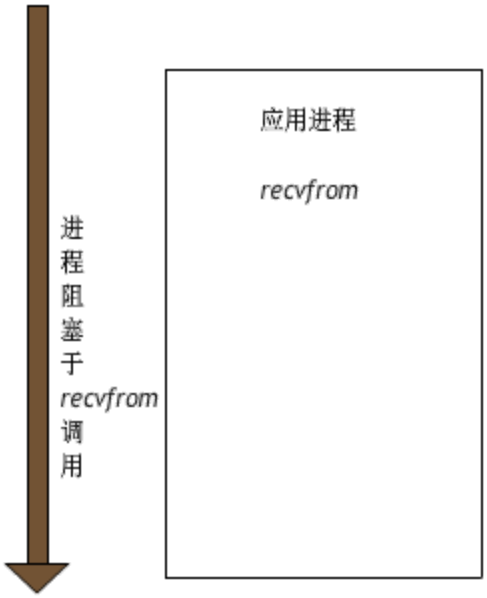
IO 有两种操作，同步 IO 和异步 IO。同步 IO 是指必须等待 IO 操作完成后，控制权才返回给用户进程。异步 IO 是，无须等待 IO 操作完成，就将控制权返回给用户进程。

常见的 4 种 IO 模型有：

- 阻塞 IO 模型
- 非阻塞 IO 模型
- IO 多路复用
- 异步 IO 模型

1 阻塞 IO 模型

在 Linux，默认情况下所有的 socket 都是阻塞的，一个典型的读操作流程如图：



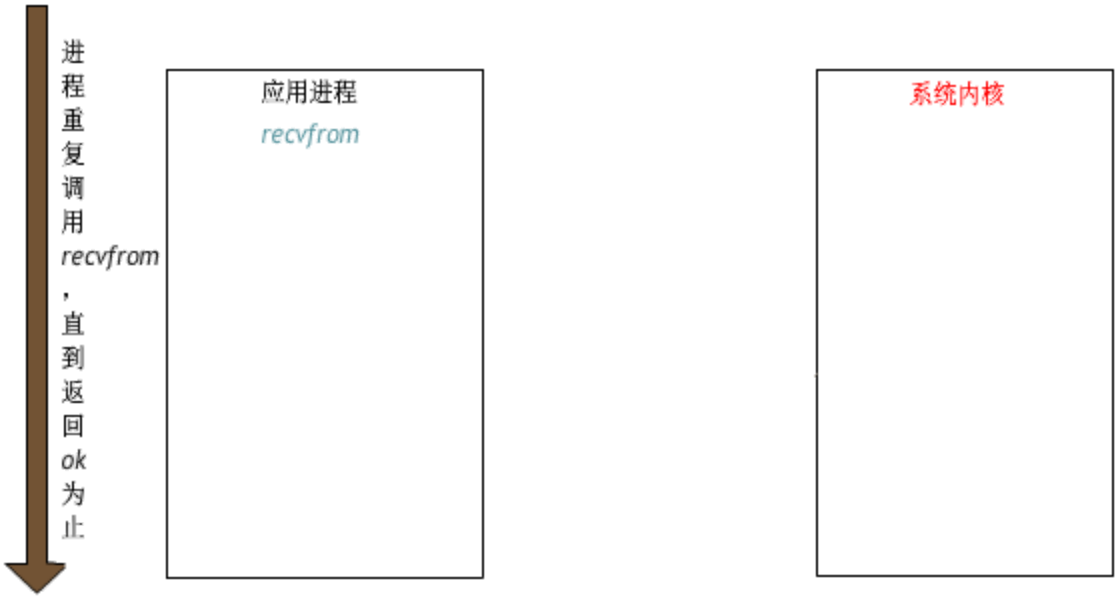
阻塞和非阻塞描述的是用户线程调用内核 IO 操作的方式：阻塞是指 IO 操作彻底完成后才返回用户空间，非阻塞是指 IO 操作被调用后立即返回给用户一个状态值，不需要等待 IO 操作彻底完成。

对于网络 IO 来说，很多时候在用户线程请求 IO 时，系统内核就要等待足够数据的到来。而在用户进程这边，整个进程会被阻塞。当系统内核一直等到数据准备好了，它就会**将数据从系统内核拷贝到用户内存中**，然后系统内核返回结果，用户进程才解除阻塞的状态，重新运行起来。

所以，阻塞 IO 模型的特点是 **IO 执行的两个阶段都被阻塞了**。

2 非阻塞 IO 模型

在 Linux 中，可以通过设置 socket IO 变为非阻塞状态，流程如图：



当用户进程发出 read 数据的操作时，如果内核中的数据还没准备好，那么它不会阻塞用户进程，而是立刻返回一个错误。

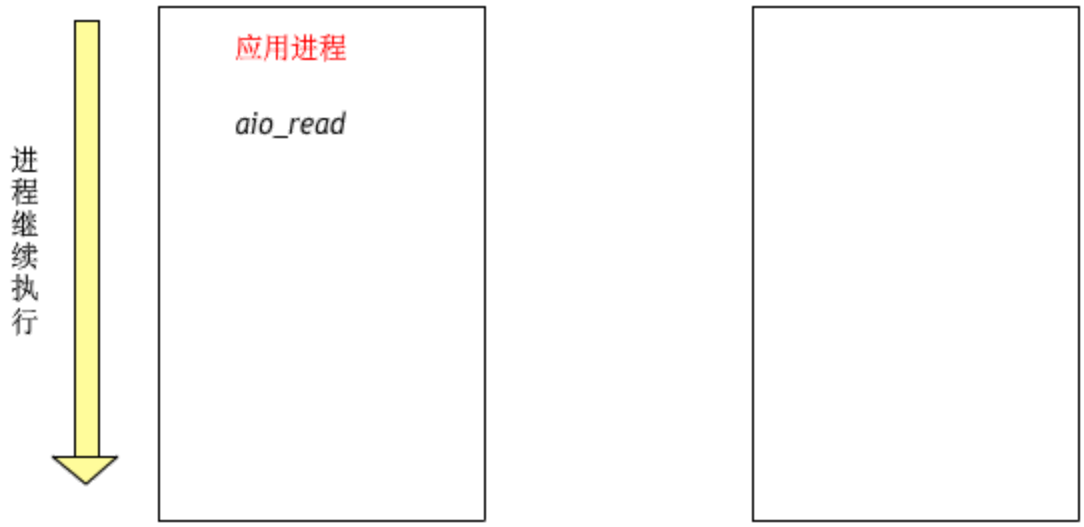
用户进程收到了错误结果，知道数据还没准备好，于是再发起一次 read 操作。

一旦内核中的数据准备好了，并且又再次收到了用户进程的系统调用，则立即将数据拷贝到用户内存中，然后返回正确的返回值。

所以，在非阻塞 IO 中，用户进程需要**不断地主动询问系统内核数据是否准备好**。这将大幅度占用 CPU 资源，因此这个模型在实际应用中很少使用。

3 IO 多路复用模型

IO 多路复用，也称为**事件驱动 IO**。它的原理是有个函数会不断轮询所负责的所有 socket，当某个 socket 有数据到达时，就通知用户进程。流程如图：

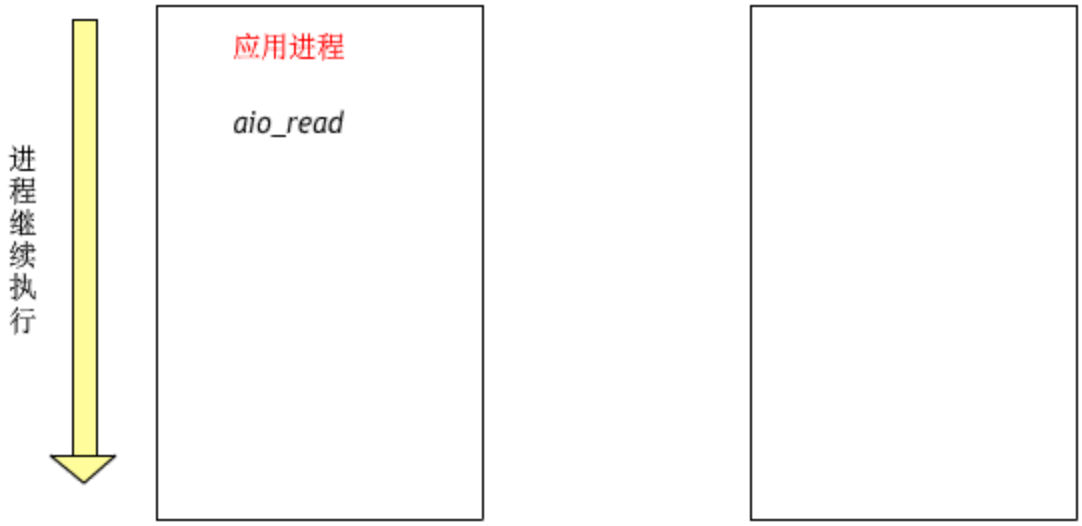


当用户进程调用了 select，那么整个进程就会阻塞。而同时，**内核会监听所有 select 负责的 socket**，当任何一个 socket 数据准备好了，select 就会返回。这时用户进程再调用 read 操作，将数据从内核拷贝到用户进程。

这个模型和阻塞 IO 的模型性能上其实并没有太大的不同，事实上还更差一些，因为这里需要使用两次系统调用，而阻塞 IO 只调用了一次。

用 select/epoll 这些 IO 多路复用的优势在于它可以同时处理多个连接，如果连接数比较高的话，用 IO 多路复用比较合适。

4 异步 IO 模型



用户进程发起 read 操作，内核收到这个异步的 read 请求操作以后，会立刻返回，所以不会对用户进程产生任何阻塞。

然后，内核会等待数据准备完成，再将数据拷贝到用户内存，当**这一切都完成以后，内核会给用户进程发送一个信号**，返回 read 操作已完成的信息。

它与阻塞 IO 的区别是，用户进程调用后不会阻塞进程，而是去开始别的任务；与非阻塞 IO 的区别是，用户进程不需要主动询问系统内核。而是当用户请求的数据拷贝完成后，直接给用户进程一个信号。

5. 进程间通讯方式

每个进程都有各自不同的地址空间，任何一个进程的全局变量在其它进程中都看不到，所以进程之间交换数据必须要通过内核。

进程 A 和 B 进行跨进程数据读写时，首先需要在内核中开辟一块缓冲区，进程 A 把数据先拷贝进缓冲区，然后进程 B 来内核缓存区中把数据读走。内核提供的这种机制称为进程间通信（Inter Process Communication，IPC）。

1）管道

管道分为匿名管道、有名管道和高级管道。

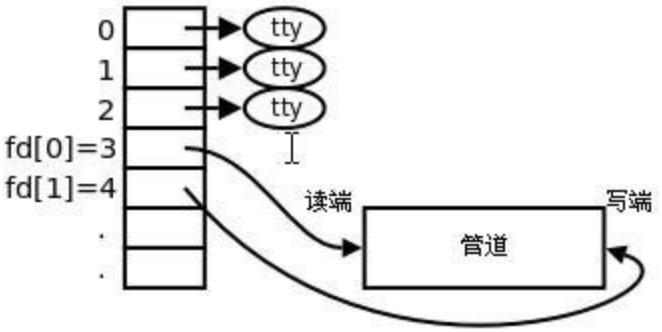
匿名管道

匿名管道（pipe）是一种半双工通信方式，即数据只能在通信双方单向流动。而且，只能在具有亲缘关系的进程间使用，通常是父子进程之间。

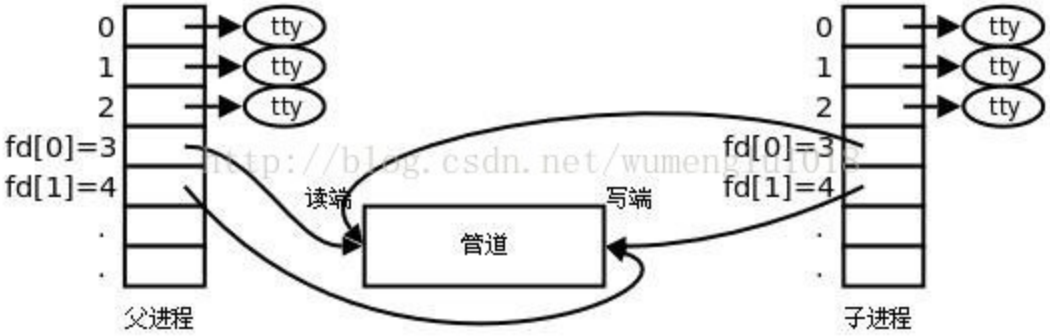
```
// 需要的头文件
#include <unistd.h>

// 通过pipe()函数来创建匿名管道
// 返回值：成功返回0，失败返回-1
// fd参数返回两个文件描述符
// fd[0]指向管道的读端，fd[1]指向管道的写端
// fd[1]的输出是fd[0]的输入。
int pipe (int fd[2]);
```

通过匿名管道实现进程间通信过程如下：

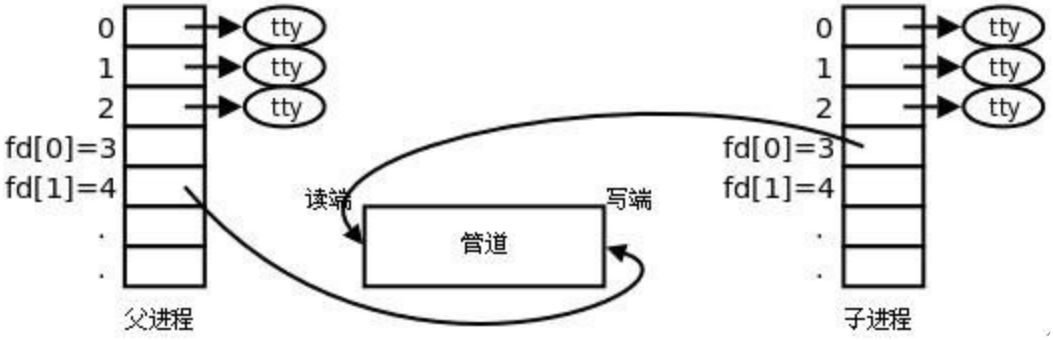
1. 父进程创建管道



2. 父进程fork出子进程



3. 父进程关闭fd[0]，子进程关闭fd[1]



- 父进程创建匿名管道，得到两个文件描述符（file descriptor，简称 fd）指向管道的两端；
- 父进程 fork 出子进程，子进程也有两个 fd 指向同一管道；
- 父进程关闭 fd[0]，子进程关闭 fd[1]，即父进程关闭管道读端，子进程关闭管道写端。父进程往管道里写，子进程从管道中读。管道用环形队列实现，数据从写端流入从读端流出，以实现进程间的通信。

有名管道

有名管道（named pipe）也是半双工的通信方式，但是它允许无亲缘关系的进程相互通信。

高级管道

将另一个程序当做一个新的进程在当前程序进程中启动，则它算是当前进程的子进程，这种通信方式称为高级管道（popen）方式。

2）信号、信号量

信号（sinal）是一种比较复杂的通信方式，用于通知进程的某个事件已经发生。

信号量（semophore）是一个计数器，用来控制多个进程对共享资源的访问。它经常作为一种锁机制，保证某进程更新共享资源时，其它进程无法访问该资源。因此，主要作为进程间或者同一进程不同线程之间的同步手段。

3）消息队列

消息队列（message queue）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。

4）共享内存

共享内存（shared memory）就是映射一段能被其它进程所访问的内存，这段内存由一个进程创建，但多个进程都可访问。共享内存是最快的 IPC（进程间通信） 方式，为了内存访问的安全性，它往往与其它通信机制，如信号量来配合使用，以实现进程间的同步和通信。

5）套接字

套接字（socket）也是一种进程通信机制，与其它通信机制不同的是，它可用于不同机器之间的进程通信。它的通信过程如下（具体可看计算机网络-套接字详解）：

- 创建 socket：通过套接字通信的双方需要具有 IP 地址和端口号；
- 绑定，将相应字段赋值，再用 bind 函数将其绑定在创建的服务器套接字上；
- 监听，服务器端套接字创建完毕并绑定后，需要调用 listen 函数进行监听，等待客户端连接；
- 接收调用，客户端连接后调用 accept 函数，处理客户端请求；
- 连接服务器，当接收调用完成后，需要客户端调用 connect 函数连接到服务器进行通信；
- 发送数据，无论是客户端和服务器端，进程的数据交互都是通过套接字来完成的，发送和接收数据时使用 write 和 read 函数；
- 交互完成后，调用 close 函数断开连接。

参考资料：

- 8种进程通讯：https://cloud.tencent.com/developer/article/1690556
- 管道：https://blog.csdn.net/wumenglu1018/article/details/54019755

6. 页面置换

1）缺页中断

https://www.cnblogs.com/xiaolincoding/p/13631224.html

2）置换算法