链滴

socket 发送缓冲区控制

作者：PJunQuey

原文链接：https://ld246.com/article/1679127731462

来源网站：链滴

假设应用程序需要发送40kB数据，但是TCP发送缓冲区只有25kB剩余空间，那么剩下的15kB数据怎办?

如果等待OS缓冲区可用，会阻塞当前线程，因为不知道对方什么时候收到并读取数据.

为了避免这种阻塞，可以引入应用层发送缓冲区，将要发送的数据暂时缓存起来。当TCP连接的发送冲区有足够的空间时，应用层发送缓冲区中的数据就可以被立即发送，从而避免阻塞

TCP连接发送数据时，内核维护了一个发送缓冲区，用于存储待发送的数据，而<u>发送缓冲区的大是有限的</u>。

当调用send()发送数据时，数据从用户态缓冲区复制到内核缓冲区，然后通过网络发送出去

如果发送缓冲区已满，则send()会被阻塞，直到发送缓冲区中有空间可用

在这里我们可以使用快速用户空间套接字[1],减少内核复制开销

当发送大量数据时，如果一次性全部发送到内核缓冲区中，则可能会因为缓冲区空间不足而导致阻塞阻塞时间可能很长。

为提高数据传输的效率和响应速度，将数据分批发送，每次发送一部分数据，并通过应用层发送缓冲进行缓存

等待发送缓冲区有空间时再发送下一批数据。使得发送操作不会因为缓冲区不足而阻塞，从而提高传效率和响应速度

相当于设置消息帧协议[22]

注意socket发送缓冲区可以不装满发送, 而且可以由程序员手动设置缓冲区大小

解决办法

1. 使用非阻塞IO操作

就是将socket设置成非阻塞模式, 发送数据时不会阻塞线程

2. 使用多线程

也就说类似于sylar项目中那样, 通过hook把socket修改为异步的

3. 使用快速用户空间套接字 [1]

减少内核复制开销

4. 设置缓冲区大小

一般不建议把缓冲区设置的太大, 在高性能场景下, 设置太大会导致利用率低, 没办法有效利用

而且socket也不止一个, 每个都设置的较大就会导致缓冲区非常大而且利用率非常低

□

1.

快速用户空间套接字

Linux内核提供的高性能IPC[2]机制, 目的是降低状态切换[19]的开销^（上下文切换和内存复制开销）^

传统IPC[21]涉及多次上下文切换和内存复制导致性能损失, FUS 是零拷贝,无系统调用,无状态切换[19]

FUS 完全可以取代 普通socket, 但是在一些对性能没那么高要求的地方可以使用普通的, 因为相对简单

基于内存映射和事件驱动机制的

首先, 两个进程都创建 FUS 套接字，并使用内存映射将其映射到共享的内存区域中。
然后，进程可以直接在内存区域中读写数据，而不需要进行状态切换[19]
当数据准备好时，通过事件驱动机制通知另一个进程进行读取

```
#define FUS_SOCKET_SIZE (1 << 12)
#define FUS_PAYLOAD_SIZE (1 << 12)

struct fus_socket {
    int evtfd;
    uint32_t head, tail, wrap_counter;
    char pad1[4], pad2[4], payload[FUS_PAYLOAD_SIZE];
};

int main() {
    void* map = mmap(nullptr,
        FUS_SOCKET_SIZE, PROT_READ | PROT_WRITE, MAP_SHARED | MAP_ANONYMOUS,
        -1, 0);
    if (map == MAP_FAILED) {
        std::cerr << "Failed to mmap: " << strerror(errno) << std::endl;
        return 1;
    }

    fus_socket* sock = reinterpret_cast<fus_socket*>(map);

    sock->evtfd = eventfd(0, EFD_NONBLOCK);
    if (sock->evtfd == -1) {
        std::cerr << "Failed to create eventfd: " << strerror(errno) << std::endl;
        return 1;
    }

    sock->head = sock->tail = sock->wrap_counter = 0;

    // Send some data
    const char* message = "Hello, FUS!";
    uint32_t len = strlen(message);
    memcpy(sock->payload + sock->tail, message, len);
    sock->tail += len;

    // Notify the receiver
    uint64_t event_val = 1;
```

```
    int res = write(sock->evtfd, &event_val, sizeof(event_val));
    if (res == -1) {
        std::cerr << "Failed to write to eventfd: " << strerror(errno) << std::endl;
        return 1;
    }

    // Wait for the receiver to read the data
    int futex_res = syscall(SYS_futex, &sock->tail, FUTEX_WAIT, sock->tail, nullptr, nullptr, 0);
    if (futex_res == -1) {
        std::cerr << "Failed to wait on futex: " << strerror(errno) << std::endl;
        return 1;
    }

    std::cout << "Data received by receiver." << std::endl;

    // Cleanup
    close(sock->evtfd);
    munmap(map, FUS_SOCKET_SIZE);
    return 0;
}
```

2.

进程通信

● 共享存储^（通信进程之间存在一块可直接访问的共享空间）^

需要同步互斥工具

- 低级通信方式^（基于数据结构的共享）^ 只能存放固定的数据结构, 速度慢, 限制多
- 高级通信方式^（基于存储区的共享）^ 在内存划分共享存储区, 不限制数据结构和存放位置, 速快, 限制少

● 消息传递^（进程间直接交换数据, 数据以 格式化的消息 为单位交换）^

通过发送/接受原语进行数据交换

格式(消息头, 消息体)

当前最广泛的进程间通信方式, 网络的主要通信工具(报文[3])

- 直接通信方式^（直接将消息交给进程(将消息放置接收进程的消息队列)）^
- 间接通信方式^（消息发送给中间实体(信箱), 网络中常用）^

● 管道通信^（一个特殊的共享文件）^

在内存中开辟一个固定大小的缓冲区(与内存页[4]大小相同)

采用半双工通信[17], 进程访问管道需要互斥访问

没写满不允许读, 没读完不允许写

只能读一次, 读出后管道抛弃数据

IPC问题[18]

实际通信方式

- 管道^（半双工的通信方式，分为匿名管道和命名管道）^
- 信号^（异步的通信方式，用于通知接收进程发生了某种事件）^
- 共享内存^（进程间数据共享的方式，允许多个进程访问同一块内存区域）^
- 消息队列^（消息传递机制，消息队列存放在内核中，允许一个进程向另一个进程发送消息）^
- 信号量^（进程间同步的机制，用于协调多个进程对共享资源的访问）^
- 套接字^（进程间网络通信的机制，可以实现不同主机上的进程间通信）^

3. - 报文交换
- 优点
    - 无须建立连接
    - 动态分配路线
    - 提高线路可靠性
    - 提供多目标服务
- 缺点
    - 产生了转发时延
    - 存储管理困难

4. - 页式虚拟存储器 :: 基本分页存储管理 [5]
- 页表 :: 是一种存储结构
    - 有效位被称为装入位
    - 页表项 = (有效位^（是否被使用, 是否在主存中）^, 脏位^（是否被修改过, 是否需要使用回写略）^, 引用位^（也称访问位, 用于记录一段时间内的访问次数或者其他, 供置换算法参考）^, 物理页磁盘地址)
        - 在没有使用虚拟技术时, 那么就没有有效位这些辅助信息, 因为作业全装入内存[^7]
    - 优点 :: 长度固定, 页表简单, 调入方便
    - 缺点 :: 会有部分空间浪费, 页逻辑上不独立, 处理, 保护和共享不方便
    - 页表是地址变化机构的一部分(数据), 另一部分是判断(判断逻辑地址越界和是否在页表)
- 内存页面 > 内存块(Cache块)
- 快表 :: 将页表放入高速缓冲存储器, 遵循局部性原理 [15]
    - 放入主存的页表称为慢表
    - 通常采用全相联或组相联方式, 是相联存储器, 可以按内容寻址
    - 命中率一般在 90% 以上
    - 快表加速的是将逻辑地址转化成物理地址的速度, 类似与ARP [16]

5. - 快表 :: 将页表放入高速缓冲存储器, 遵循((20220825171041-2qo8g4p "局部性原理"))
- 放入主存的页表称为慢表
- 通常采用全相联或组相联方式, 是相联存储器, 可以按内容寻址

- 命中率一般在 90% 以上
- 快表加速的是将逻辑地址转化成物理地址的速度, 类似与((20220819165406-gosrf2n "ARP"))
- 基本分页存储管理^（将主存分为大小相等相对较小的块作为分配的基本单位）^

页^（内存分配单元(存储单元是字/字节)）^

页框^（进程的最大页数）^

注意页框号和页号的区别^（页框号是作业在内存的实际物理地址,页号一般指逻辑地址 选择题往往考给你个虚地址,先让你判断是否在内存,然后置换算法, 等找到页框号后加页内偏移量组成实际物理地址^

页表 :: 是一种存储结构[6], 数组结构导致页号可以用地址隐含表示不需要存储

- 注意区分Cache映射方式[8]与页表 :: 是一种存储结构[6]的关系

页表负责将逻辑地址转换为主存地址, 然后在拿主存地址再进行一次页表查询是否再Cache中

- 地址变换机构[14]的任务是逻辑地址转内存物理地址
-
- 多级页表

各级页表不能超过一个页面的最大页表项数目(页面大小/页表项大小)

n = log(虚地址位数 - 页内偏移量)/log(页面大小/页表项大小)

n级页表访存次数 = n + 1 次, 才能获得物理地址

6. - 页表 :: 是一种存储结构
- 有效位被称为装入位
- 页表项 = (有效位^（是否被使用, 是否在主存中）^, 脏位^（是否被修改过, 是否需要使用回写策）^, 引用位^（也称访问位, 用于记录一段时间内的访问次数或者其他, 供置换算法参考）^, 物理页或盘地址)
    - 在没有使用虚拟技术时, 那么就没有有效位这些辅助信息, 因为作业全装入内存[^7]
- 优点 :: 长度固定, 页表简单, 调入方便
- 缺点 :: 会有部分空间浪费, 页逻辑上不独立, 处理, 保护和共享不方便
- 页表是地址变化机构的一部分(数据), 另一部分是判断(判断逻辑地址越界和是否在页表)

7. - 映射方式 :: 直接映射, 全相联映射, 组相联映射[9]
- 直接映射
    - 主存块只能放在Cache的唯一位置, 若已有内容则直接替换, 哪怕有空闲的Cache块
    - 冲突概率最高, 空间利用率最低
    - 地址结构 :: 标记 - 行号 - 块内地址
    - 标记被多个地址块共享, 相当于标记被复用了, 在主存的位置用行号进行区分

相对于全相联映射[11]

- Cache 行号 = 主存块号 mod (Cache 行数)

- 全相联映射
    - 主存块可以放在Cache的任意位置

● 冲突概率低, 空间利用率最高, 命中率高, 标记比较速度慢, 成本高

● 地址结构 :: 标记 - 块内地址

● 组相联映射

● 直接映射[10]和全相联映射的折中, 设置组号, 地址块先判断组号, 如果组号对应的Cache内有数据, 在通过遍历直接检索是否有相应的标志, 地址块可以放在Cache组里的任意位置

● 组的大小适当时, 成本直逼直接映射, 性能接近全相联映射

● 地址结构 :: 标志 - 组号 - 块内地址

● 当一组由两个时称为 二路组相联, 以此类推, n路组相联

● 除此之外Cache的标记项内还有一个 有效位, 用来判断当前标志是否有效

原因是最初Cache默认标签都指向0, 但是并不是真的是0地址块的副本, 只是初始化

所以需要一个有效位来判断是否真的有效

● 错误理解[12]

标记, **组号**, **行号**, **脏位**, **有效位** 是存放在页表 :: 是一种存储结构[6]中的, 错误理解[13]

Cache不仅仅只是用来存放数据的, 同时还将存储相应的**地址映射表**

只是我们常认为的容量是指数据部分

只要看到求Cache总容量有多少位时就是求数据位 + 地址映射表

● 计算Cache命中率时 要留意读写指令!

向 a = a + 1; 这种就有读指令和写指令, 如果缺是a, 那么命中率是50%

● 从Cache中取数据首先获得物理地址(也就是在主存的真实物理地址)

将地址根据映射方式, 拆分出组号和标签等信息, 通过地址映射表比对

从而找到Cache对应的数据

8. ● 组相联映射

● 直接映射[10]和全相联映射的折中, 设置组号, 地址块先判断组号, 如果组号对应的Cache内有数据, 通过遍历直接检索是否有相应的标志, 地址块可以放在Cache组里的任意位置

● 组的大小适当时, 成本直逼直接映射, 性能接近全相联映射

● 地址结构 :: 标志 - 组号 - 块内地址

● 当一组由两个时称为 二路组相联, 以此类推, n路组相联

9. ● 直接映射

● 主存块只能放在Cache的唯一位置, 若已有内容则直接替换, 哪怕有空闲的Cache块

● 冲突概率最高, 空间利用率最低

● 地址结构 :: 标志 - 行号 - 块内地址

● 标记被多个地址块共享, 相当于标记被复用了, 在主存的位置用行号进行区分

相对于全相联映射[11]

● Cache 行号 = 主存块号 mod (Cache 行数)

10. ● 全相联映射

● 主存块可以放在Cache的任意位置

- 冲突概率低, 空间利用率最高, 命中率高, 标记比较速度慢, 成本高
- 地址结构 :: 标记 - 块内地址

11. - 这些映射方式需要用到的标记, 组号, 行号, 脏位等有专门的memory来存储~~~~换句话来说这标记的额外信息又是如何工作的不在映射方式的讨论范围内(现代计CPU)

标记, 组号, 行号, 脏位, 有效位 是存放在Cache块内的, 块中的数据只有部分是来自内存

12. - Cache内存放的是纯粹的内存数据

13. - 页表是地址变化机构的一部分(数据), 另一部分是判断(判断逻辑地址越界和是否在页表)

14. - 局部性原理: 时间局部性, 空间局部性
- 时间局部性 :: 现在使用的, 在未来大概率会被使用
- 空间局部性 :: 未来使用的, 在空间上靠近现在使用的
- 在程序执行过程中, 程序对主存的访问是不均匀的

15. - 地址解析协议
- ARP表 = (IP地址, MAC地址)
- 工作在网络层

16. -

单向通信(单工)	**只有一个方向的通信**
半双工通信(半双工)	双方都可以发送或接收数据
全双工通信(全双工)	通信双方可以同时发送和接数据

17.

IPC问题

由于进程的独立性和封闭性，需要特殊的技术和机制来完成不同进程间的通信和协作

1. 根据需求选择适合的进程间通信

2. 数据格式：通信的数据格式需要保证统一，否则无法正确解析数据

常用的数据格式包括二进制、XML、JSON等

3. 安全性：数据需要保证安全，防止被非法获取或篡改

需要考虑数据加密、身份验证等安全措施。

4. 性能：不同的IPC方式有不同的性能特点，需要根据需求选择

例如，共享内存可以实现高效的数据共享，但需要注意同步和互斥问题。

18.

内核态和用户态的切换

- 内核 -> 用户 :: 执行特权指令(修改PSW), 让出CPU使用权
- 用户 -> 内核 :: 由中断[20]引发, <u>硬件</u>自动完成转换状态过程, 系统强行夺取使用权

19. ● 中断^（来自CPU外部, 与执行指令的无关事件引起）^
● 可屏蔽中断^（CPU可以通过中断控制器的设置, 被屏蔽的信号将不被送到CPU）^
● 不可屏蔽中断^（通过专门的中断请求线发送中断信号）^

20. ## 实际通信方式

● 管道^（半双工的通信方式，分为匿名管道和命名管道）^
● 信号^（异步的通信方式，用于通知接收进程发生了某种事件）^
● 共享内存^（进程间数据共享的方式，允许多个进程访问同一块内存区域）^
● 消息队列^（消息传递机制，消息队列存放在内核中，允许一个进程向另一个进程发送消息）^
● 信号量^（进程间同步的机制，用于协调多个进程对共享资源的访问）^
● 套接字^（进程间网络通信的机制，可以实现不同主机上的进程间通信）^

21. # 消息帧协议

是一种通信协议，用于在通信系统中传输数据. 在协议中数据被分割成固定大小的帧
每个帧都包含了数据和控制信息，以确保数据的可靠传输和正确性通常

消息帧协议包含了帧的格式、帧类型、帧长度、校验和等内容

消息帧协议通常用于工业自动化、电力、交通、航空航天等领域的通信系统中，以实现数据的可靠传和控制

在实际应用中，具有数据传输可靠、数据量小、速度快、延迟低等优点，适用于实时性要求高的应用景。

但同时也存在一些缺点，例如带宽利用率低、灵活性不高等

因此，在设计和选择消息帧协议时，需要综合考虑应用场景、性能要求、可靠性要求和实际情况等因素

要注意, 消息帧协议是来自应用层的, 要注意区分链路层的帧

为什么还要在应用层[23]分帧?

链路层[24]设置的帧大小由底层协议^（以太网,wifi等）^所规定, 在开发层面上基本上是固定的, 没有法根据具体业务来进行划分

在应用层上再次进行分帧就可以将数据分成适当大小^（一般是分成更小）^来保证实时性和响应速度

现在主流的 http/1.x 是没有分帧能力的, 到2以上才有.

22. 应用层

23. # 数据链路层

链路层功能

- 为网络层提供服务 :: 无确认的无连接服务, 有确认的无连接服务, 有确认的面向连接服务
- 链路服务
  - 帧定界, 帧同步, 透明传输
    - 最大传输单元
  - 流量控制
  - 差错控制
    - 自动重传请求
    - 前向纠错

组帧

- 字符计数法 :: 在帧头部使用一个计数字段来表明帧内字符数
- 字符填充的首位定界符法 :: 特殊字符前用转移字符填充
- 零比特填充的首尾标志法 :: 连续五个 1 后 一定有个 0
- 违规编码法
  - 在物理层进行比特编码时, 通常采用违规编码法
  - 不采用任何填充技术, 只适用于采用冗余编码的特殊编码环境
  - 局域网 IEEE802 标准采用这种方法

差错控制

- 检测编码
  - 奇偶校验码
  - 循环冗余码
- 纠错编码

流量控制与可靠传输机制

- 停止-等待协议 :: 发送一帧后要接收到确认帧才能继续
- 滑动窗口流量控制
  - 发送窗口 :: 发送方维持一组连续的允许发送的帧序号
  - 接收窗口 :: 接收方维持一组连续的接收帧序号
  -

  - 单帧滑动窗口

- 多帧滑动窗口
    - 后退N帧协议
        - 发送窗口尺寸 [1, 2^n^ - 1]
    - 选择重传协议
        - 发送窗口 = 接收窗口 = 2^n-1^
    - 信道利用率 = $\displaystyle \frac{L}{CT}$
        - T 发送周期
        - C 数据发送速率
        - L 数据量
    - 信道吞吐率 = L / T
- 可靠流量机制 :: 实际 <u>有线网络</u>链路层很少采用可靠传输

介质访问控制

- 信道划分介质访问控制
    - 频分多路复用
    - 时分多路复用
    - 波分多路复用
    - 码分多路复用
        - 主要用于无线通信系统
- 随机访问介质访问控制
    - ALOHA协议
        - 纯ALOHA协议
        - 时隙ALOHA协议
    - CSMA协议
        - 1-坚持CSMA
        - 非坚持CSMA
        - p-坚持CSMA
    - CSMA/CD协议
    - CSMA/CA协议
- 轮询访问介质访问控制
    - 令牌传递协议
- 种类

- 静态 :: 信道划分介质访问控制 [25]
  - 动态 :: 随机访问介质访问控制 [26], 轮询访问介质访问控制 [27]

局域网

- 局域网的特点由三个要素决定 :: 拓扑结构, 传输介质, 介质访问控制方式
- 拓扑结构 :: 总线形网络, 星形网络, 环形网络, 网状网络 [28]
- 局域网中常使用双绞线 [29]
-

 	逻辑结构	物理结构
以太网	总线形	星形或拓展星形
令牌环	环形	星形
FDDI	环形	双环

- 以太网 :: 目前使用范围最广的局域网
- IEEE 802.3
  - 描述物理层和数据链路层MAC子层的实现方法
- 以太网简化通信措施
  - 采用无连接工作方式, 提供不可靠服务, 尽最大努力交付数据
  - 使用曼彻斯特的信号
-

参数	10BASE5	10BASE2	10BASE-T	0BASE-FL
传输媒体	基带同轴电缆(粗缆)	基带同轴电缆(细缆)		屏蔽双绞线
编码	曼彻斯特编码	曼彻斯特编码	曼彻	特编码
拓扑结构	总线形	总线形	星形	对点
最大段长	500m	185m	100m	000m
最多结点数目	100	30	2	2

- MAC帧
  - MAC地址 :: 6字节
  - 前导码 :: 8B
  - 以太网MAC帧的数据大小 64 ~ 1518 B
  - 最小长度为 64B, 由CSMA/CD的算法
- 高速以太网 :: 速率超过100Mb/s

- 100BASE-T以太网
    - 9.6us 变成 0.96us
  - 吉比特以太网
  - 10吉比特以太网
    - 只能工作在全双工方式
- 无线局域网
  - 由固定基础设施无线局域网
  - 无固定基础设施移动自组织网络
- 802.11局域网MAC帧 [30]
  - 类型 :: 数据帧, 控制帧, 管理帧
  - 数据帧
    - MAC首部 30字节
    - 帧主体 不超过2312字节
    - 帧检验序列FCS 4字节
    -

去往AP	来自AP	地址1	地址2	地址3
0	1	接收地址 = 目的地址	发送地址 = A	地址 源地址
1	0	接收地址 = AP地址	发送地址 = 源	址 目的地址

- VLAN
  - 可以隔离广播域和冲突域
  - 使用的是交换技术
  -

广域网

- 因特网的核心部分
- 广域网 != 互联网, 互联网可以连接不同类型的网络
- 广域网由节点交换机以及连接这些交换机的链路组成(结点交换机 != 路由器)
  - 结点交换机 = 三层交换机
-

区别	广域网	局域网

覆盖范围	大, 通常跨区域	小, 通常一个区域内
连接方式	点对点连接 为了提高网络可靠性,结点常连接多个点 	普遍采用多点接入技术
参考模型[31]	三层	两层
联系与相似	1. 都是互联网的重要组成构件, 二者平等 2. 连到两者上的主句在网内通信时, 只需要器物理地址	-
着重点	强调共享资源	强调数据输

- PPP协议
  - 串行线路面向字节的通信协议
  - 目的 :: 用来通过拨号或专线方式建立点对点连接发送数据
  - 组成部分
    - 链路控制协议
    - 网络控制协议
    - 一个将IP数据报封装到串行链路的方法
  - 点对点连接, 不采用CSMA/CD [32],没有最短最短帧[33]
  - 只提供检错, 不提供纠错, 是不可靠传输协议
  - 只支持全双工链路

链路层设备

- 局域网交换机
  - 本质是多端口网桥
  - 能划分冲突域
  - 总容量 = N × 10Mb/s (N是端口数, 10是例子)
  - 一般采用 全双工
  - 模式 :: 直通式交换机, 存储转发式交换机
  - MAC表 :: (MAC地址 - 接口)
    - 具有自学习功能
- 网桥

24. ● 信道划分介质访问控制
- 频分多路复用
- 时分多路复用
- 波分多路复用

- 码分多路复用
    - 主要用于无线通信系统

25. - 随机访问介质访问控制
- ALOHA协议
    - 纯ALOHA协议
    - 时隙ALOHA协议
- CSMA协议
    - 1-坚持CSMA
    - 非坚持CSMA
    - p-坚持CSMA
- CSMA/CD协议
- CSMA/CA协议

26. - 轮询访问介质访问控制
- 令牌传递协议

27. 拓扑结构 :: 总线形网络, 星形网络, 环形网络, 网状网络

28. - 在局域网和传统电话网中普遍使用

29. MAC帧

30. 参考模型

31. - CSMA/CD协议

32. - 最小长度为 64B, 由CSMA/CD的算法