第 38 卷 第 8 期
Vol.38 No.8

计 算 机 工 程
Computer Engineering

2012 年 4 月
April 2012

• 工程应用技术与实现 • 文章编号：1000—3428(2012)08—0228—03 文献标识码：A 中图分类号：TP302.7

一种高性能环形缓冲区的研究与实现

姚章俊[a]，陈蜀宇[b]，卢 尧[a]

(重庆大学 a. 计算机学院；b. 软件学院，重庆 400030)

摘 要：基于单生产者多消费者模型，剖析传统环形缓冲区写入和读出进程并发操作的缺陷，提出一种带有缓冲区单元状态标记的算法，解决环形缓冲区写入和读出进程的同步问题。定量分析产生环形缓冲区性能瓶颈的条件，在不满足该条件的情况下，环形缓冲区的性能会有大幅提升。对比实验和数学分析验证了该环形缓冲区处理数据包的性能较好。

关键词：环形缓冲区；进程同步；生产者；消费者；单元状态

Research and Implementation of High-performance Ring Buffer

YAO Zhang-jun[a], CHEN Shu-yu[b], LU Yao[a]

(a. College of Computer Science; b. College of Software Engineering, Chongqing University, Chongqing 400030, China)

【Abstract】Based on the model of single producer multiple consumers, the drawback of the writing concurrency and reading processes in traditional ring buffer is analysed. An algorithm with tagged buffer unit status is presented to solve the synchronization problem of writing and reading processes in the ring buffer. And the condition resulted in the bottleneck of the ring buffer performance is analysed quantificationally. The performance of the ring buffer is promoted greatly if the condition does not be met. Contrastive experiment and mathematical analysis verify that the performance of data packet processing in the ring buffer is better.

【Key words】ring buffer; process synchronization; producer; consumer; unit status

DOI: 10.3969/j.issn.1000-3428.2012.08.074

1 概述

环形缓冲区是一种先进先出的循环缓冲区，相对于队列减少了对地址的反复操作，增加了稳定性[1]，被广泛应用在不同领域中，如嵌入式操作系统[2]、数据采集[3]、网关设计[4]等。环形缓冲区采用生产者消费者模型同步数据写入和读出操作，降低生产者消费者间的耦合程度，并有效地解决忙闲不均的问题。文献[5-7]创造性地将其使用在 PF_RING 模块中。在通信程序中也使用环形缓冲区存放通信中发送和接收的数据[8]。

本文研究内容来源于对网络数据包捕获模块的设计工作。该工作的难点是降低丢包率，丢包产生的原因在于网卡接收数据速度快，应用程序读取数据速度慢。为解决这个问题，采用单生产者多消费者模型的环形缓冲区提高读出速度，提升捕包效率。本文以此为思路对环形缓冲区进行研究。

2 高性能环形缓冲区模型

图 1 描述的是环形缓冲区的结构，本文以该模型为基础。

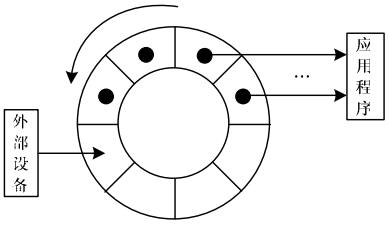

图 1 环形缓冲区模型

已有文献对于采用单生产者多消费者模型的环形缓冲区有一个基本要求：只有当所有消费者都读取了该缓冲区单元的数据后，生产者才能将数据写入该缓冲区单元。文献[9]在对数据缓存机制的研究中，保证了数据的时效性，但在处理海量数据时，会出现丢包现象。文献[10]设计的环形循环滑动窗口支持多线程并发，但在流速可变的情况下，其稳定性和可靠性都有缺陷。本文利用锁的思想，提出缓冲区单元标记算法，不仅提高了缓冲区的效率，而且降低了计算资源的占用率。

在环形缓冲区中，写入进程和读出进程相互协作，共享逻辑地址空间，但是因并发访问可能会引起数据不一致[11]。针对这种情况需要采取一整套方案以确保协作进程的有序执行并维护数据的一致性。

对于本文提出的环形缓冲区模型，从缓冲区单元状态标记、算法实现和高效性制约条件 3 个方面阐述。

2.1 缓冲区单元状态标记

在传统有限缓冲条件下的生产者消费者模型中，缓冲区单元的状态被设置为“Free”和“Full”2 种状态。如图 2 所示，状态为“Free”的缓冲区单元被生产者写入数据后变为“Full”，状态为“Full”的缓冲区单元被消费者读出数据后变为“Free”[12]。

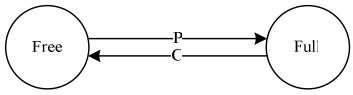

图 2 传统同步算法下单元状态转换图

在一个固定空间容量的环形缓冲区中，对环形缓冲区并发执行写入和读出操作时，效率表现为对单个缓冲区的读写

基金项目：科技部国际科技合作基金资助项目(2007DFR10420)；重庆市自然科学基金资助项目(2008BB2307)

作者简介：姚章俊(1987－)，男，硕士研究生，主研方向：网络体系结构；陈蜀宇，教授、博士生导师；卢 尧，硕士研究生

收稿日期：2011-07-20 **E-mail：**yzj19870824@126.com

时间比值的大小。本文从生产者消费者对缓冲区单元进行读写操作的微观层面，分析传统单生产者多消费者模型的缺陷以及对这个模型的改进。

图 3 描述的是单个生产者和多个消费者对一个缓冲区单元的写入和读出过程。生产者进程使用 T_t 的时间传输数据到缓冲区单元，经过 T_w 的时间将数据放入缓冲区单元，经过 T_t 的时间找到下一个数据存放处；消费者进程使用 T_r 的时间从缓冲区单元中读出数据，并花费 T_t(为简化模型，将生产者和消费者传输数据的时间都设为 T_t)的时间将数据传输到目的地。设定对一个缓冲区单元的读写时间比：

$$\Psi = \frac{T_C}{T_P}$$

为保护对缓冲区单元的互斥访问，在消费者进行下次读写前，生产者必须找到下一次要写入的数据，即表达式：

$$2T_t + T_w \leqslant T_t + T_r \tag{1}$$

成立。

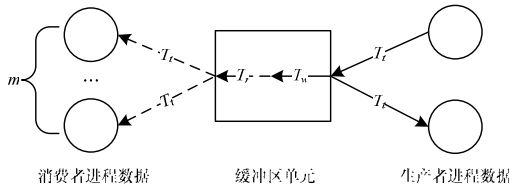

图 3　缓冲区读写操作微观图

在传统单生产者多消费者算法中，假设有 m 个消费者进程，对一个存储单元的读取时间为 $m(T_t + T_r)$，时间比：

$$\Psi_{\text{old}} = \frac{m(T_r + T_t)}{2T_t + T_w}$$

由式(1)可知：

$$\Psi_{\text{old}} \geqslant m \tag{2}$$

如果在消费者读取数据时，将当前占用的缓冲区单元状态设为“Booked”，其他消费者在获取该单元的状态为“Booked”后便不能再次读取其中的数据，此时的时间比：

$$\Psi_{\text{new}} = \frac{T_r + T_t}{2T_t + T_w} \geqslant 1 \tag{3}$$

由此可知：当 $m>1$ 时，$\Psi_{\text{old}} > \Psi_{\text{new}}$，这意味着带有“Booked”标记的算法读写效率比传统算法的读写效率更高。

综上，得出缓冲区单元状态及其对应描述：Free——单元为空，可被生产者写入，不可被消费者读出；Full——单元包含有效数据，可被消费者读出，不可被生产者写入；Booked——单元在消费者读出之前被预定，不可被生产者写入，亦不可被其他消费者读出。

定义状态间转换的生产者消费者行为：P1——生产者向空单元中写入数据；C1——消费者预定有数据的单元；C2——消费者读取预定单元中数据。

为了清晰地描述新算法，使用有限状态机(FSM)表示缓冲区单元状态及生产者和消费者的行为，如图 4 所示。

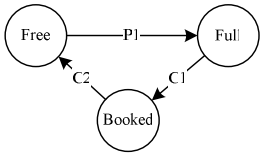

图 4　缓冲区单元标记算法下单元状态转换图

2.2　算法实现

本节采用伪代码实现缓冲区单元状态标记算法，在算法的适当位置注释。

```
struct buffer                    //缓冲区单元结构体
{
  FLAGTYPE flag;                 //缓冲区单元状态
  DATATYPE date;                 //缓冲区单元中数据
}
producer()
{
 lock(global_lock);              /加锁同步写入缓冲区单元状态
 if(buffer[P].flag == FREE )     //缓冲区单元可写不可读
 {
   buffer[P].flag = FULL;        //改变缓冲区单元为“Full”
   produce(buffer[P]);           //写入数据
   myP=P;                        //读取写入位置
   P=(P+1)%BUF_SIZE;             //下一次写入位置
 }
 else                            //缓冲区单元不可写
   myP=－1;
   unlock(global_lock);
   if(myP == -1)       //如果缓冲区单元不可写返回错误状态
       return FAILURE_BUFFER_FULL;
   return SUCCESS;
}

consumer()
{
 lock(global_lock);              //加锁读取缓冲区单元状态
 myflag = buffer[C].flag;
 unlock(global_lock);
 if(myflag == FREE)    //缓冲区状态为“Free”，返回错误状态
     return FAILURE_EMPTY;
 if(myflag == BOOKED)
 //缓冲区状态为“Booked”，读取缓冲区中数据
     consume(buffer[C].data);
 lock(global_lock);
 buffer[C].flag = FREE;
 //读取数据后，缓冲区单元状态设为“Free”
 C=(C+1)%BUF_SIZE;               //下一次读取数据位置
 unlock(global_lock);
 return SUCCESS;
}
```

生产者的动作都被限制在锁 global_lock 中，确保数据写入安全。消费者进程互斥读取单元的当前状态，当单元状态为“Booked”时，读取单元中的数据。该算法完善环形缓冲区进程同步的机制，保证缓冲区的高效性能。

2.3　制约数据包处理性能的条件

尽管缓冲区单元标记算法提升了环形缓冲区的性能，但当对环形缓冲区的写入速度和读出速度差距悬殊时，它不能自适应处理这一极端情况，因此有必要寻找写入速度和读出速度间的关系，只有在满足这个关系的基础上，新环形缓冲区才能发挥其最大性能。设定变量如表 1 所示。

表 1　环形缓冲区变量

变量	说明	变量	说明
m	消费者个数	N_c	每个消费者读出环形缓冲区单元个数
V_{in}	数据写入环形缓冲区速度	V_p	生产者写入环形缓冲区速度
V_{out}	数据读出环形缓冲区速度	V_c	每个消费者读出环形缓冲区速度
N_p	生产者写入缓冲区单元个数	V_C	消费者读出环形缓冲区总速度
N_C	消费者读出环形缓冲区单元个数	R	环形缓冲区单元个数

根据速度的微分定义，得到：

$$V_{\text{in}}=\frac{\mathrm{d}N_p}{\mathrm{d}t} \tag{4}$$

$$V_{\text{out}}=\frac{\mathrm{d}N_C}{\mathrm{d}t} \tag{5}$$

从微观方面考虑，在$(0,t]$时间内，生产者写入缓冲区单元的个数是关于生产者进程的定积分，即：

$$N_p=\int_0^t V_p(\tau)\mathrm{d}\tau \tag{6}$$

同理：

$$N_c=\int_0^t V_c(\tau)\mathrm{d}\tau \tag{7}$$

消费者读出环形缓冲区单元个数为每一个消费者读出环形缓冲区单元个数之和：

$$N_C=\sum_{i=1}^{m}N_{c_i} \tag{8}$$

由式(7)、式(8)可得：

$$N_C=\sum_{i=1}^{m}\int_0^t V_{c_i}(\tau)\mathrm{d}\tau=\int_0^t\sum_{i=1}^{m}V_{c_i}(\tau)\mathrm{d}\tau \tag{9}$$

由式(4)~式(6)、式(9)，可得：

$$\begin{cases}V_{\text{in}}=\dfrac{\mathrm{d}(\int_0^t V_p(\tau)\mathrm{d}\tau)}{\mathrm{d}t}=V_p\\ V_{\text{out}}=\dfrac{\mathrm{d}(\int_0^t\sum_{i=1}^{m}V_{c_i}(\tau)\mathrm{d}\tau)}{\mathrm{d}t}=\sum_{i=1}^{m}V_{c_i}\end{cases} \tag{10}$$

在缓冲区单元标记算法中，生产者和消费者必然满足：

$$\begin{cases}N_p-N_C\geqslant 0\\ N_p-N_C\leqslant tR\end{cases}$$

其中，t为程序运行时间。

$$\begin{cases}\int_0^t[V_p(\tau)-\sum_{i=1}^{m}V_{c_i}(\tau)]\mathrm{d}\tau\geqslant\int_0^t 0\mathrm{d}\tau\\ \int_0^t[V_p(\tau)-\sum_{i=1}^{m}V_{c_i}(\tau)]\mathrm{d}\tau\leqslant\int_0^t R\mathrm{d}\tau\end{cases}$$

$$\begin{cases}V_p(t)-\sum_{i=1}^{m}V_{c_i}(t)\geqslant 0\\ V_p(t)-\sum_{i=1}^{m}V_{c_i}(t)\leqslant R\end{cases} \tag{11}$$

由式(10)、式(11)得：

$$0\leqslant V_{\text{in}}-V_{\text{out}}\leqslant R \tag{12}$$

当$V_{\text{in}}-V_{\text{out}}$的取值范围为[0,*R*]时，环形缓冲区才能发挥其最大效率。根据式(12)分析：

(1)当$V_{\text{in}}-V_{\text{out}}<0$，即$V_{\text{in}}<V_{\text{out}}$时，写入数据速度小于读取数据速度。则消费者可读取所有数据，之后将会被阻塞，直到有新数据写入；

(2)当$V_{\text{in}}-V_{\text{out}}>R$时，极端情况发生：写入速度过快，消费者未能及时读取所有数据。需采用某种方式降低生产者的写入速度以保持缓冲区的性能。

3 实验分析

实验环境为 Lenovo T100 服务器(单核 Xeon E5500 CPU，1.4 GHz 主频，1 GB 内存)。实验时用本文设计的环形缓冲区替换 PF_RING 的环形缓冲区，将新 PF_RING 加载到 Linux 2.6.27.2 内核，标记该内核为 A 组；B 组为加载原始 PF_RING 的同版本内核。

在 2 组内核做同样实验：stream.c 模拟产生的网络数据流速作为缓冲区的写入速度，网络数据在终端上的显示速度作为缓冲区的读出速度。

定义环形缓冲区的性能：

$$\Phi=\frac{V_{\text{out}}}{V_{\text{in}}} \tag{13}$$

实验初期写入速度逐渐从 0 增加到 90 Mb/s，之后在满足式(12)的前提下，写入速度升至 110 Mb/s。图 5 为不同时间段里 2 个实验对照组性能Φ的实验结果(为方便作图将实际时间化为单位时间)。

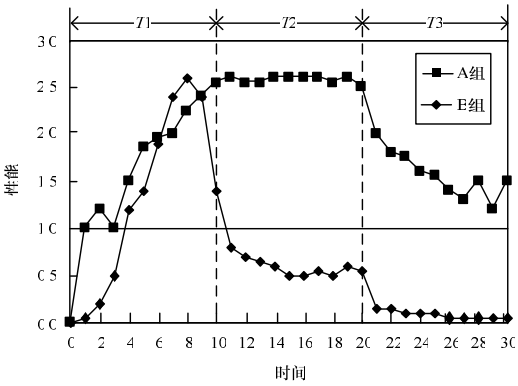

图 5 2 个实验的对比结果

从图 5 可知，采用缓冲区单元状态标记算法的环形缓冲区性能整体优于采用传统同步算法实现的环形缓冲区，尤其在应对巨量数据时性能的差别更是显而易见。根据性能表现，将实验时间分为 *T*1、*T*2 和 *T*3 共 3 个阶段，用数学解释不同阶段的性能差异。

由式(12)得：

$$V_{\text{in}}\leqslant V_{\text{out}}\leqslant R+V_{\text{in}}$$

将不等式两边同时除以V_{in}得：

$$1-\frac{R}{V_{\text{in}}}\leqslant\frac{V_{\text{out}}}{V_{\text{in}}}\leqslant 1 \tag{14}$$

将式(13)代入式(14)得：

$$1-\frac{R}{V_{\text{in}}}\leqslant\Phi\leqslant 1 \tag{15}$$

在 *T*1 阶段随着写入速度的不断加大，性能不断提升，但是性能增长率会不断降低，在图中的表现是折线斜率的减小；在 *T*2 阶段环形缓冲区的写入速度基本稳定；在 *T*3 阶段的起始时间写入大量数据，根据式(15)可知，其性能必然会下降，但不会低于$1-\frac{R}{V_{\text{in}}}$。对于采用传统同步算法的环形缓冲区，运行初期性能上升极快，但从式(2)可知，其性能会迅速下降，特别是写入大量数据后的性能极低。

实验数据和数学分析相辅相成，证明新环形缓冲区算法具有高效性。

4 结束语

本文介绍一种新型环形缓冲区模型。首先从微观层面上分析需对缓冲区单元进行状态标记的原因，根据生产者和消费者的行为制作有限状态机。结合有限状态机与进程互斥的方法，用伪代码实现新环形缓冲区的同步算法。之后计算出发挥新环形缓冲区最优性能的条件。最后根据实验结果和数学分析验证了新模型的高效性。新环形缓冲区可作为独立模块被其他需使用环形缓冲区的系统自由安装和拆卸，提升所在系统的性能。

后续研究的重点在于如何控制环形缓冲区的写入速度，当出现$V_{\text{in}}-V_{\text{out}}>R$后，通过调整写入速度，最大化利用环形缓冲区。

参考文献

[1] Lemos M, Lifschitz S. A Study of a Multi-ring Buffer Management for BLAST[C]//Proc. of the 14th International Workshop on Database and Expert System Applications. Washington D. C., USA: IEEE Computer Society, 2003: 247-252.

[2] 王亚军, 李建文, 吉 方. 基于环形缓冲区的实时系统负载平衡技术[J]. 计算机应用与软件, 2005, 22(4): 38-39.

[3] 庄哲民, 蔡清福. 基于环形缓冲区的高速数据采集系统[J]. 数据采集与处理, 1998, 25(3): 41-43.

[4] 程安宇, 何 川, 冯辉宗, 等. 基于 SAEJ1939 协议的双缓冲区网关设计[J]. 计算机应用, 2010, 30(z1): 101-103.

[5] Deri L. Improving Passive Packet Capture: Beyond Device Polling[C]//Proc. of the 4th International Conference on System Administration and Network Engineering. Amsterdam, Holland: IEEE Press, 2004: 814-826.

[6] Fusco F, Deri L, Gasparakis J. Towards Monitoring Programmability in Future Internet: Challenges and Solutions[C]//Proc. of the 21st Tyrrhenian Workshop on Digital Communications. Ponza, Italy: IEEE Press, 2006: 514-528.

[7] Deri L. Effective Traffic Measurement Using Ntop[J]. Network Traffic Measurements and Experiments, 2000, 6(8): 138-143.

[8] 杨 武, 方滨兴, 云晓春, 等. 基于 Linux 系统的报文捕获技术研究[J]. 计算机工程与应用, 2003, 39(26): 28-30.

[9] 楮蓬飞, 张焕强, 方贵明. 视频会议系统中数据缓冲机制的研究[J]. 计算机应用研究, 2006, 34(5): 67-69.

[10] 詹 英, 吴春明, 王宝军. 一种与缓冲区紧耦合的环形循环滑动窗口的数据流抽取算法[J]. 电子学报, 2011, 39(4): 1-5.

[11] 刘晓建, 吴庆波, 戴 华, 等. 一种用于并行系统的非阻塞消息队列机制[J]. 计算机工程与科学, 2011, 33(4): 75-80.

[12] 韩明峰. 环形缓冲区读写操作的分析与实现[J]. 单片机与嵌入式系统应用, 2003, 23(12): 23-25.

编辑 顾逸斐

(上接第 223 页)

参考文献

[1] 张佳琳. 基于 GPS 残迹的分布式导航算法[J]. 计算机工程, 2010, 36(20): 34-36.

[2] Jimenez A R, Seco F. A Short-range Ship Navigation System Based on Ladar Imaging and Target Tracking for Improved Safety and Efficiency[J]. IEEE Trans. on Intelligent Transportation Systems, 2009, 10(1): 186-197.

[3] Omerbashich M. Integrated INS/GPS Navigation from a Popular Perspective[J]. Journal of Air Transportation, 2002, 7(1): 103-118.

[4] 卢德兼. 多星座全球导航卫星系统完整性分析[J]. 计算机工程, 2010, 36(11): 238-240.

[5] 籍 颖, 刘兆祥, 刘 刚, 等. 基于 Kalman 滤波农用车辆导航定位方法[J]. 农业机械学报, 2009, 40(增刊): 13-17.

[6] 肖 鹏, 张彩友, 冯 华, 等. 变电站巡检机器人 GPS 导航研究[J]. 传感器与微系统, 2010, 29(8): 23-26.

[7] Shiotani S, Sasa K, Tarada D, et al. Numerical Navigation for a Ship in Simulation of Waves[C]//Proc. of the International Offshore and Polar Engineering Conference. Beijing, China: [s. n.], 2010: 663-670.

[8] 段海滨, 邵 山, 苏丙未, 等. 基于仿生智能的无人作战飞机控制技术发展新思路[J]. 中国科学: 技术科学, 2010, 40(8): 853-860.

[9] Capi G, Tod H. Evolution of Neural Controllers for Robot Navigation in Human Environments[J]. Journal of Computer Science, 2010, 6(8): 837-843.

编辑 顾逸斐

(上接第 227 页)

用户的使用需求，为基于全系统虚拟化技术的操作系统安全增强、进程控制、病毒防护等技术的研究成果更进一步地拓展了实用空间。

由于时间和能力所限，本文研究成果主要适用于当前主流的独立显存的显卡，对于一些特殊显卡或共享显存的集成显卡还会有些特殊处理，这里不再详述。通过在华硕 K40IP 笔记本电脑(支持 VT-x,不支持 VT-d)上验证，主机向客户机操作系统直接分配 NVIDIA G205M 显卡，并安装原厂驱动进行测试，测试证明虚拟机操作系统的显示效果得到质的提升，获得了和主机显示完全接近的效果。

参考文献

[1] 董耀祖, 周正伟. 基于 X86 架构的系统虚拟机技术与应用[J]. 计算机工程, 2006, 32(13): 71-73.

[2] 马文琦. 基于虚拟化的多域安全框架及其关键技术研究[D]. 长沙: 国防科学技术大学, 2008.

[3] 杜 海, 陈 榕. 基于完全虚拟化的进程监控方法[J]. 计算机工程, 2009, 35(8): 88-90.

[4] 英特尔开源软件技术中心, 复旦大学并行处理研究所. 系统虚拟化——原理与实现[M]. 北京: 清华大学出版社, 2009.

[5] Linux-KVM. KVM-kernel-based Virtual Machine[EB/OL]. (2010-01-15). http://www.linux-kvm.org/page/Main_Page.

[6] Linux-KVM. How to Assign Devices with VT-d in KVM[EB/OL]. (2010-01-15). http://www.linux-kvm.org/page/How_to_assign_devices_with_VT-d_in_KVM.

[7] 河 秦, 王洪涛. Linux2.6 内核标准教程[M]. 北京: 人民邮电出版社, 2008.

[8] Ng B H, Lau B, Prakash A. Direct Access to Graphics Card Leveraging VT-dt[EB/OL]. (2009-06-20). http://www.eecs.umich.edu/~bengheng/pubs/vgapt_techreport.pdf.

编辑 顾逸斐