文章编号:1001 - 9081(2007)01 - 0038 - 02

一种新型有效的 KeepAlive 算法在通讯系统中的应用

王 鹏,许 毅
(电子科技大学 计算机科学与工程学院, 四川 成都 610054)
(wotseb@ hotmail. com)

摘 要:KeepAlive 机制在通信中使用极其广泛。大量使用 KeepAlive 机制来保活的一个主要原因是物理链路的不可靠性。利用它可以避免不可预料的异常的发生而引发的进程或者资源吊死。文中提出了一种新型有效的 KeepAlive 算法,并结合 3G BSS 嵌入式通讯系统进行分析计算,证明了它的高效性和实用性。

关键词:KeepAlive; 资源吊死; 3G BSS; 嵌入式系统
中图分类号: TP316.2 **文献标识码**:A

A new effective KeepAlive arithmetic used in embedded application

WANG Peng, XU Yi
(*College of Computer Science and Engineering,*
University of Electronic Science and Technology of China, Chengdu Sichuan 610054, *China*)

Abstract: KeepAlive strategy is widely used in communications system. One important reason of using the strategy is that the physical link is unreliable. With KeepAlive strategy, many unexpected exceptions can be avoided. A new effective KeepAlive arithmetic was presented in the paper. The performance of the arithmetic was analyzed and computed, being integrated with 3G Base Station System (3G BSS). The result shows it is of high efficiency and practicability.

Key words: keepalive; resource hang; 3G BSS(3G Base Station System); embeded system

1 KeepAlive 算法

KeepAlive(保活),是通信双方检测对方是否存在、链路是否正常的一种非超时失败检测策略[1]。使用 KeepAlive 机制最常见的就是面向可靠连接的 TCP 协议:建立 TCP 连接的两端定时会向对端发起探测消息,判断对方是否存在。如果不存在,就认为本端的套接口资源是处于吊死状态,于是释放本端的套接口资源。所谓资源吊死[2],是指资源没有正确释放,导致以后有新的需要时申请不到资源。

本文将结合 3G BSS 系统提出一种新型保活算法,并且进行分析、计算,体现算法的高效性和实用性。

3G BSS 系统是按照 SubSystem、Module、SubModule 等由上到下划分的。各个子系统是相对独立,每个模块在子系统完成不同的功能。每个模块都有一个 CallHandler 进程,用于处理呼叫。当有用户起呼时,每个模块的 CallHandler 进程都会创建一个实例来处理本次呼叫。现在我们假设有两个模块 ModuleA 和 ModuleB,并且假设系统能支持的最大 Handler 实例数为 MAX_HANDLER_INSTANCE_NUM。对于一路呼叫,ModuleA 和 ModuleB 分别创建实例 Handler_I、Handler_J(I,J 标记 ModuleA 和 ModuleB 创建的实例的下标,要求:1≤I,J≤MAX_HANDLER_INSTANCE_NUM)。当呼叫释放的时候,按照第三代合作伙伴计划 2(3rd Generation Partnership Project 2,3GPP2)中协议规定,一个进程的实例释放会发消息通知对等进程的实例(例如这里的 Handler_I 和 Handler_J)进行释放,这样能保证资源的合理释放和重复使用。

为避免进程吊死,相关联的进程之间采用 KeepAlive 机制进行实例保活。下面分别给出传统的算法和我们的高效算法。

1.1 传统的 KeepAlive 算法

由于 Instance_I 和 Instance_J 的对等关系,这里我们只分析 Instance_I 发起的保活流程。假设系统定义的发送 KeepAlive 探测消息最大次数:

#define MAX_REPEAT_KEEPALIVE_COUNT 3

正常流程:

(1) 有语音用户接入时,ModuleA 和 ModuleB 各自创建一个呼叫实例 Instance_I、Instance_J 都成功,Instance_I 设定一个定时器 TKeepAlive 定时器,同时,Instance_I 初始化超时次数计数器 KeepAliveCounter 为 0;

(2) 等到 TKeepAlive 保活定时器超时,Instance_I 判断是否满足 KeepAliveCounter < MAX_REPEAT_KEEPALIVE_COUNT(主要用于异常情况),如果满足条件,Instance_I 发送 EV_S_KEEPALIVE 消息,并且 KeepAliverCounter ++;否则如果 KeepAliveCounter < MAX_REPEAT_KEEPALIVE_COUNT 不成立,就认为自身(Instance_I)是吊死了,执行实例 Instance_I 的释放;

(3) Instance_J 收到来自 Instance_I 的 EV_S_KEEPALIVE,给 Instance_I 回 EV_S_KEEPALIVE_ACK 消息,以示自己存在,宣告链路正常;

(4) Instance_I 收到 EV_S_KEEPALIVE_ACK 消息后,将超时次数定时器 KeepAliveCounter 清零,跳到(2)执行;否则,直接跳到(2)执行。

1.2 高效的 KeepAlive 算法

传统的 KeepAlive 算法中,对于两个相关联的处理模块的 Handler 进程实例要相互发送 KeepAlive 消息来保活。现在我们再引入一个 Main 进程概念。从功能角度上说,引入 Main

收稿日期:2006 - 07 - 18;**修订日期**:2006 - 10 - 08

作者简介:王鹏(1982 -),男,湖北黄冈人,硕士研究生,主要研究方向:网络通信、嵌入式系统; 许毅(1972 -),男,四川新都人,博士,主要研究方向: 计算机软件.

进程是为了更好的集中控制,创建、释放、维护 Handler 进程;从系统性能方面讲,在系统控制模块和呼叫处理模块中引入 Main 进程,可以均匀各个分布式模块的负荷;最后,也是最重要的就是能大大节省进程间 KeepAlive 消息交互占用的带宽。

为了保证模块 ModuleA 和 ModuleB 中进程实例的对应关系,ModuleA 中的 Handler 进程实例和 ModuleB 中的 handler 进程实例要相互保存对方的信息(这里我们只用记住对方的 Handler 实例的 Index 就可以)。我们在两个模块的 Main 进程中,分别定义一个数组存放各自 Handler 实例的保活信息,定义如下:

ModuleA Main 进程:

WORD32 adwAKeepAlive[MAX_HANDLER_INSTANCE_NUM / 32 +1]

ModuleB Main 进程:

WORD32 adwBKeepAlive[MAX_HANDLER_INSTANCE_NUM / 32 +1]

数组元素都初始化为 0。ModuleA Main 进程设定一个保活定时器 TShakeHand,定时时长为 MIN_SHAKEHAND_TIMEOUT,同时增加一个发送次数计数器 SendKeepAliveCounter。

保活算法如下:

(1) 有用户起呼,ModuleA Main 每创建一个 Handler 进程实例 i,就会将数组元素 adwAKeepAlive[i/32] 的第 i%32 位置为 1;当 ModuleA Main 释放一个进程 i 时,就会将数组元素 adwAKeepAlive[i/32] 的第 i%32 位置为 0;ModuleB 也是如此。

(2) 等到保活定时器 TShakeHand 超时,ModuleA Main 向 ModuleB Main 发送 EV_S_KEEPALIVE 保活消息,消息中携带了 adwAKeepAlive[ ]数组,里面存放了模块 ModuleA 的所有 Handler 实例的状态(active or dead)。

(3) ModuleB Main 进程收到 EV_S_KEEPALIVE 保活消息后,取出消息中的 adwAKeepAlive[ ]数组,分析,判断是否有进程吊死(如果 adwKeepAlive[m] 的第 n 位等于 0,且 ModuleB 中的与 ModuleA 中 Index = 32 * m + n 相对应的 Handler 实例存在,表明 ModuleB 上的该 Handler 进程实例吊死了,就释放该进程;如果发现 adwKeepAlive[m] 的第 n 位等于 1,并且 ModuleB 中的与 ModuleA 中 Index = 32 * m + n 相对应的 Handler 进程不存在,表明 ModuleA 上有进程吊死)。如果有进程吊死,回一个 EV_S_KEEPALIVE_ACK 告诉 M1 Main,参数中携带 adwBKeepAlive[ ],将 M2 这边所有 Handler 进程的状态告知 M1 Main;否则应答消息中不带任何内容;如果 ModuleB Main 进程在定时器超时仍然没有收到来自 ModuleA Main 进程的 EV_S_KEEPALIVE 保活消息,也给 ModuleA Main 发送应答消息,告知本方 Handler 进程的状态信息;如果 ModuelB Main 进程超时未收到来自 ModuleA Main 进程的 EV_S_KEEPALIVE 消息,超时未收到消息次数加 1;当超过未收到消息次数超过 MAX_REPEAT_KEEPALIVE_COUNT,释放本方所有的 Handler 进程。

(4) ModuleA 的 Main 进程如果收到 EV_S_KEEPALIVE_ACK,判断消息头信息,如果发现应答消息中的消息体长度为 0,表明两边进程同步正常,设置 SendKeepAliveCounter = 0,等待下一次 TShakeHand 超时;否则取出消息体内容,释放吊死的进程,设置定时器定时时长为 MIN_SHAKEHAND_TIMEOUT,等待下一次 TShakeHand 超时。如果 ModuleA Main 进程超时未收到来自 ModuleB Main 进程的 EV_S_KEEPALIVE_ACK 消息,超时未收到消息次数加 1;当超过未收到消息次数超过 MAX_REPEAT_KEEPALIVE_COUNT,释放本方所有的 Handler 进程。

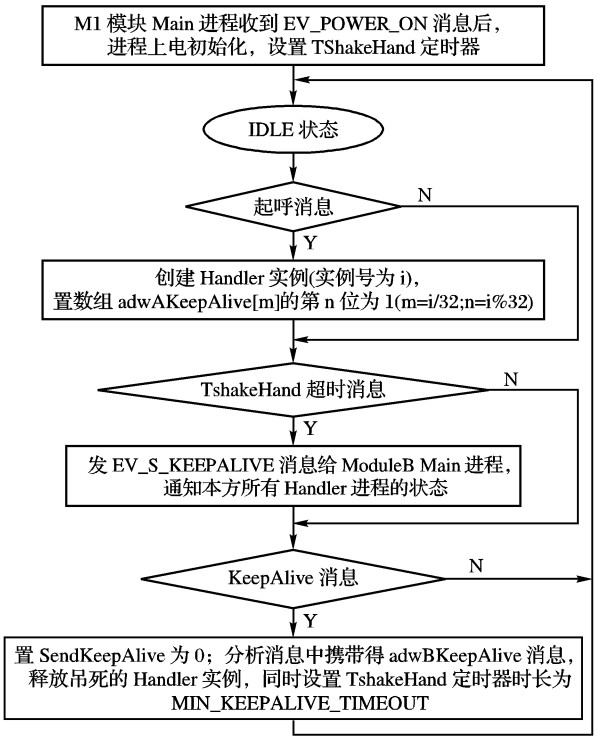

图 1　新型 KeepAlive 算法流程

2　结语

在通讯系统中,系统的容量和带宽是运营商非常关注的两个指标。下面我们以保活机制需要占用的带宽来比较两种保活算法(以一天 24h 两算法使用保活机制发送消息的信息量),通过比较的结果来证明本文提出的新型保活算法的实用性和高效性。

假设系统设定的 MAX_HANDLER_INSTANCE_NUM 为 5000,TKeepAlive 定时器定时时长:MAX_KEEPALIVE_TIMEOUT = 3min,MIN_KEEPALIVE_TIMEOUT = 1min。

采用传统算法:

(1) 24h 内 M1 和 M2 之间交互的消息数目:

(24 * 60/1) * 2 * 4000 = 11 520 000

注:上式中的"1"是 TKeepAlive 定时器定时时长;"2"是表示消息成对,有发有应答;"4 000"是假定系统同时在线的呼叫数目,也就是同时运行的 Handler 数目。

(2) 24h 交互的消息流量:

44 * 11 520 000/(1 024 * 1 024) = 483.4 MB

注:"44"是我们 CDMA2000 系统中内部消息结构的消息头长度。这里计算的时候假设 KeepAlive 消息的长度为 0,应答消息也是如此。

采用高效算法:

(1) 24h 内 M1 和 M2 之间交互的消息数目:

(24 * 60/3) * 1 = 480

注:采用高效算法,正常情况下,TKeepAlive 定时器定时时长为 3min,只有在异常情况下为 1min,由于异常情况出现的几率很小,所以可忽略不计。

(2) 24h 交互的消息流量:　　(下转第 138 页)

在信噪比(SNR)等于 7 dB 时就可以使通信系统的误码率达到 10^{-3}。图 9 是用序列长度为 $N = 15$,用户个数为 $K = 2$ 时做出的系统误码率的数值仿真,从图中可以看出来在 $N = 15$ 时,系统的误码率在 SNR = 6 dB 时可以使通信系统的误码率达到 10^{-3},有着传统 PN 序列无法比拟的优势。

表 1 传统 PN 码性能与本文设计的系统产生的 PN 码性能比较

序列	长度	序列数	$\max\{\overline{C^2}\}$
传统PN序列	7	2	11.7857 (Gold Set)
	15	2	23.9 (Kasami Set)
优化的周期混沌序列	7	3	6.852502
	15	2	10.377765
本文设计的混沌序列	7	2	3.5263
	15	2	8.3352

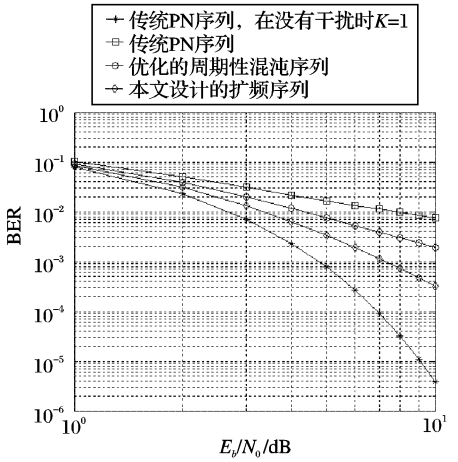

图 8 传统 PN 序列和本文混沌序列 BER 分析($N = 7, K = 2$)

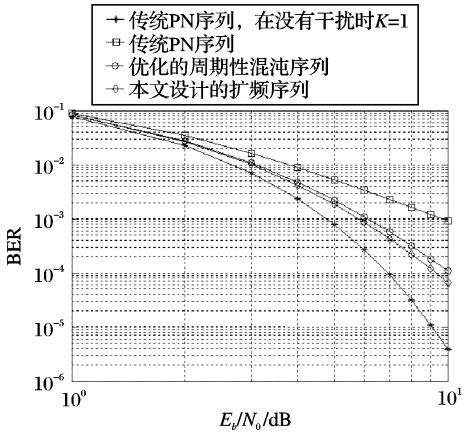

图 9 传统 PN 序列和本文混沌序列 BER 分析($N = 15$, $K = 2$)

5 结语

通过计算机数值仿真可以看出,本文设计的扩频混沌序列具有良好的平衡特性和相关特性,而且自相关旁瓣相当小,可有效地减少码间干扰。应用于 DS-CDMA 系统后,在高斯白噪声信道模型下,系统的误码率比传统的 PN 序列、周期性混沌序列更小。

参考文献:

[1] ZIEMER RE, PETERSON RL, BORTH DE. Introduction to Spread Spectrum Communications[M]. NewYork: Prentice-Hall, 1995.

[2] ABEL A , SCHWARZ W . Chaos communications principles , schemes and system analysis[J]. Proceeding of IEEE, 2002, 90(5): 679 - 710.

[3] GHOBAD HB , CLARE DM . A chaotic Direct Sequence Spread - Spectrum Communication System[J]. IEEE Transactions on Communications, 1994, 42(2/3/4): 1524 - 1527.

[4] HEIDARI - BATENI G , MCGILLEM CD . Chaotic sequences for spread spectrum: an alternative to PN-sequences[A]. Proceeding IEEE International Conference on Selected Topics in wireless communications[C]. 1992. 437 - 440.

[5] ZHANG Q, ZHENG J. Choice of chaotic spreading sequences for asynchronous DS-CDMA communication[A]. Proceeding of IEEE Asia-Pacific conference on circuits and systems[C]. 2000. 642 - 645.

[6] HU SG, ZOU Y, HU JD, BAO L. A synchronous CDMA system using discrete coupled-chaotic sequence[A]. Proceeding IEEE Conference Bringing Together Education, Science and Technology[C]. 1994, 44. 1524 - 1527.

[7] LING C, LI SQ. Chaotic spreading sequences with multiple access performance better than random sequences[A]. IEEE Transaction on circuits systems-I[C]. 2000. 394 - 397.

[8] VLADEANU C, BANICA I, ASSAD SE. Periodic chaotic spreading sequences with better correlation properties than conventional sequences-BER performances analysis[A]. International symposium on signals, circuits and systems 2[C]. 2003.

[9] GAN JC, XIAO XC. Characteristic of addition of chaos[J]. Chinese Physics, 2003, 52(5): 1085 - 1090.

[10] WANG H, HU JD. Improved Logistic-Map chaotic sequence[J]. Journal of Communications, 1997, 18(8): 71 - 77.

[11] BANICA I, VLADEANU C. Performance of iterative multistage detectors for DS-CDMA systems[A]. IEEE International Conference on Telecommunications[C]. 2001. 35 - 39.

[12] KURIAN AP, SADASIVAN PK, HTUT SM. Performance enhancement of DS/CDMA system using chaotic complex spreading sequence[J]. IEEE Transactions on wireless communications, 2005, 4(3): 984 - 989.

(上接第 39 页)

480 * (44 + (5000/32 + 1) * 4)/(1024 * 1024) = 0.31 MB

注:“44”的含义同上;(5000/32 + 1) * 4是adwAKeepAlive[ ]结构大小。

由此可见,无论是从两模块交互的 KeepAlive 消息的数目还是消息的流量来说,较之传统的保活算法,本文算法都有上千倍的提高。嵌入式通讯系统[3]对系统的容量和传输带宽要求特别高,如何将系统必要的开销降至最低是我们不断升级优化的目标。本文的 KeepAlive 算法实用性强、效率高、占用带宽小、灵活性强,为一高效可行算法。另外,本文中关于定时器时长动态控制的算法,只是简单的根据消息丢失情况来动态调整,实际中应该根据系统的吞吐量、负荷情况,同时结合无线环境的影响(信号强弱、外界干扰和自干扰情况、信噪比等)来动态调整。

参考文献:

[1] CHANDRA T, TOUEG S. Unreliable failure detectors for asynchronous systems[A]. Proceedings of the 10th ACM Symposium on Principles of Distributed Computing[C]. 1991. 325 - 340.

[2] CHANDRA T, TOUEG S. Unreliable failure detectors for reliable distributed systems[J]. Journal of the ACM, 1996, 43(2): 225 - 267.

[3] LI Q, YAO C. 嵌入式系统的实时概念[M]. 王安生, 译. 北京: 北京航空航天大学出版社, 2004. 76 - 153.