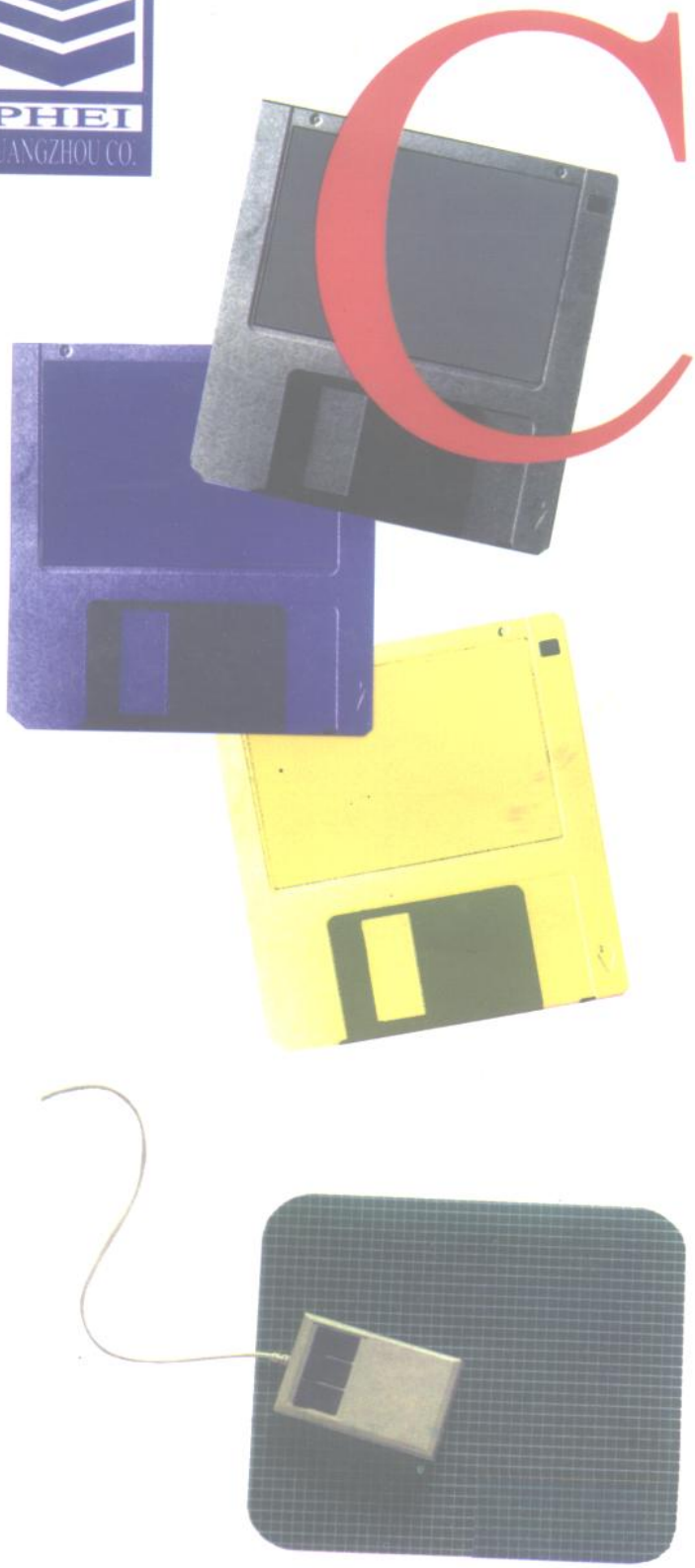
汤岳清
范　雄　编著
刘超胜

C语言的工程应用



電子工業出版社.

C语言的工程应用

C 语言的工程应用

汤岳清　范雄　刘超胜　编著
李智渊　主审

电　子　工　业　出　版　社

(京)新登字 055 号

内 容 简 介

本书的青年作者们近年来在C语言的工程应用方面，如网络与通信，数据库和多媒体软件开发方面积累了一些经验。

本书整理发表了几个在网络、通信、数据库、汉字处理、加密/解密，压缩/反压缩等实例，并给出完整的源程序(这些程序绝大多数都在有效地运行着)供有关读者借鉴。

C语言的工程应用
汤岳清　范雄　刘超胜　编著
李智渊　主审
责任编辑：穗　民
★
电子工业出版社出版(北京万寿路)
电子工业出版社发行　　各地新华书店经售
电子工业出版社广州科技公司排版
电子部情报所印刷厂印刷
★
开本：787×1090 毫米 1/16 印张：14.75　字数：300 千字
1995 年 6 月第一版　　1995 年 6 月第一次印刷
印数 1～5000 册　　定价：20.00 元
ISBN7—5053—2647—3/TP.810

前 言

不管是开发系统程序还是应用程序，C 语言都成了人们乐意采用的高级语言。国内已经有多本优秀书籍对 C 语言本身作了较详细的介绍。但是具体指导开发产品的书籍并不多见。近年来，作者有幸参加了多个国内外大型软件开发，特别是在网络与通信，数据库和多媒体方面积累了一些经验。在电子工业出版社广州科技公司的支持下，经过一年多时间的整理，终于完成了此书。

本书没有对 C 语言本身作介绍，重点放在具体开发应用上。并完全以实例为主，而这些程序绝大多数都在有效地运行着。另外，本书所选专题在许多实际应用场合都用得到，例如，C 语言操作 FoxBASE$^+$/Dbase IV 数据库(.DBF)文件的函数，对工业控制，证券系统，实时系统开发都很有用。PC 与 Unix/Xenix 通信，以及 Unix 与 Unix 通信的程序设计在实际应用中都具有极大的实用价值。已经建立起 Novell 网络的用户，在进一步应用时，如何加锁，解锁，如何利用事务跟踪系统(TTS)的功能，本书都有十分重要的指导意义。特别是如何利用 IPX/SPX 通信协议在两个工作站或工作站和服务器间实现通信，已经有许多应用场合都提出了这一要求。例如，无线电传呼系统就要求以迅速可靠的方法将接收到的寻呼从接收台送到无线发射工作站，这时 IPX/SPX 是一个相当好的选择。在网络应用与通信系统设计时，加密，解密，压缩和反压缩是要作相当考虑的，我们都给出了完整的源程序。另外，在不运行汉字操作系统的情况下利用汉字改善用户接口方面，我们提出了两种思路，并以 BIG—5 内码为例，给出了实现方法和程序实例。

本书共分为十二章，各章节内容概括如下：

第一章讨论 Novell NetWare 提供的事务跟踪系统(TTS)的概念，工作原理；详细列出了各 TTS 功能的用法。并以实例说明了如何在 C 应用程序以及如何在不支持 TTS 功能的数据库管理系统(如 FoxBASE$^+$或 DbaseIV)上加入 TTS 功能。

第二章讨论多用户环境下的共享问题，包括简单的数据覆盖到严重的死锁。程序员应当利用对一个文件或一组文件进行物理加锁(文件加锁)或对文件的一部分加锁(记录加锁)，或者对文件或记录进行逻辑加锁，以及利用信号灯(semaphore)等方式实现对多个用户同时存取进行控制。要注意的是，不管采取哪一种方式，程序员应当明白每一种锁定方式对网络性能带来的影响。一个总的原则是：尽可能锁最少的记录，占最短的时间。本章讨论 NetWare 环境下的 6 种类型的锁和 6 个主要的功能调用。我们将给出一个小型数据库实例来说明记录加锁是如何工作的。另外，我们还开发了一个实例来展示如何实现对软件的用户数进行控制。

第三章介绍了 Novell NetWare 可装载模块(NLM)的基本概念，分类，及开发方法。列出了 NLM 对 NetWare 服务器上的 DOS 分区的功能调用；最后给出了一个可以成为产品的 NLM 源程序，该程序实现了在 NetWare 控制台访问服务器上的 DOS 分区(包括 A:，B:，C:等)。

第四章详细分析了 IPX/SPX 协议格式，采用的数据结构，讨论了利用 IPX/SPX 通信协议进行程序设计的要点，并给出了用汇编语言构造的 C 函数。

第五章列出了每一个 IPX/SPX 调用的详细资料，并给出了利用 IPX/SPX 协议实现在 Novell 网络上两个工作站间传送文件的源程序。

第六章有两个目的，之一是给出一个系统需求规范(SRS)的实例；之二是给出有关终端仿真程序，PC 与 Unix/Xenix 主机间传送文件及实现 PC DOS 与 Unix/Xenix 共享资源的 SRS。

第七章是上一章的继续，给出了 Xenix 终端仿真的系统设计规范(SDS)，对该系统的终端仿真，PC 与 Unix/Xenix 间文件传送以及 DOS 用户共享 Unix/Xenix 文件系统都有详细讨论。

第八章讨论 DOS 和 Xenix 文件传送程序中 DOS 上的基本通信函数和 DOS，Xenix 利用通信函数进行文件传送的程序实例。基本通信函数适合于 DOS 与 DOS 间，DOS 与 Unix/Xenix 系统间通信的任何一种应用。

第九章讨论了有关 Unix 通信的基础，重点讨论了 Unix/Xenix 的终端特性，并给出了上一章的 DOS 与 Unix/Xenix 通信程序有关接收，发送的 Unix 主机部分的源程序。

第十章讨论如何利用 C 语言存取 FoxBase＋或 Dbase IV 数据库，存取包括读，写，加锁，解锁，以及索引操作。本章详细分析了.dbf 文件的格式，存取方法。列出了用 C 语言实现的所有函数源程序，并以实例说明了调用方法。

第十一章讨论用 C 语言实现中文输出的两种思路；分析了两种汉字系统：BIG—5 和 GB2312 的字库结构及访问算法；并以 BIG—5 内码系统为例列出了实用 16 点阵和 24 点阵中文显示源程序。

第十二章简要地讨论了加密，解密和压缩，反压缩的原理；给出了加密，解密源程序和两个压缩，反压缩源程序。

李智渊教授在百忙中审阅了全书，在此谨表敬意和谢意。

编审者

1994.3 于深圳

本书全部程序，均可以软磁盘方式提供。

邮购地址：广东省广州市五山路华南师大科技大楼 215 室

电子工业出版社广州科技公司邮购部

邮　编：510631

定价(含邮挂费)：55.00 元

目　录

第一章　如何调用 NetWare 的 TTS

1.1　概　述

1.1.1　TTS 基本概念

NetWare 的事务跟踪系统 TTS(Transaction Tracking System)，最早在 SFT LevelII NetWare 上实现，在后续的 SFT NetWare 和 NetWare 386 以及最新推出的 NetWare 4 中都支持。引入 TTS 的目的在于处理类似如下应用的情况：一个银行系统在转账时，要求完成三个操作：(1) 从支出账号减去所转金额；(2) 在接收账号加上所转金额；(3)作必要的记载。因为转账这一事务由以上三步组成，任何一步未完成都要求放弃所有操作，所以要有一种办法保证这一事务要么完全成功，要么完全失败。

有了 TTS，应用程序便可以将多个逻辑相关的磁盘写操作组当成事务看待，以便对所要写入磁盘的数据进行跟踪，即在磁盘上保存其备份直到所有写操作都完成。如果因为某种原因文件服务器垮台或其它原因使得系统不能正常工作，则事务可以恢复到执行前的状态，即把保存在磁盘上的数据备份写回磁盘上。这样就保证了数据的完整性。

Novell 公司提供的 TTS 功能尽管十分简单，但因为它与记录锁定有关，所以如何使用却并不那么明了，有时甚至使程序员感到困惑。例如，如果一个应用程序利用信号灯(semaphores)来完成数据同步，则事务不能被正确地跟踪。只有记录加锁(包括 NetWare 的物理或逻辑加锁调用)，或 DOS 功能调用 5Ch，或控制记录存取(Control Record Access)时，NetWare 才会隐含地对事务进行跟踪。

1.1.2　TTS 工作原理

一个文件服务器在下面情况之一发生时开始跟踪一个事务：运行在工作站上的软件锁定一个记录；或者调用 TTS Begin Transaction 系统调用。然后该工作站向这个文件服务器写数据。这时，所要写入的数据放在文件服务器的 Cache 里。然后该服务器找到磁盘上的目标文件，定位要写入的位置，读出将被覆盖的原始数据和该目标文件的名字，目录以及这些数据所处的位置等信息，并将其送到 Cache 内，此时，目标文件仍没有被修改。

下一步，服务器将读出的数据写到该事务工作卷的事务工作文件(Transaction Work File)上。最后新的数据从 Cache 里写入目标文件。直到这时才对目标文件作了物理修改。图 1—1 列出了 TTS 跟踪事务时的 5 个步骤。

1.1.3　记录锁定和隐式，显式跟踪

事务跟踪有两种方式：显式事务跟踪和隐式事务跟踪。隐式事务跟踪主要针对那些不是利用 TTS 功能开发的程序；而显式跟踪则是针对直接调用 TTS API 的应用。但不管是那一种方式，记录加锁和 TTS 都结合在一起。

当一个程序的数据文件被用户设置为属性 Transactional(利用 FLAG filename /T 命令设置)或者利用记录加锁(物理或逻辑锁)来同步输入/输出时，隐式事务跟踪便开始工作。

当具有一定的加锁时(用户或者程序设置锁阀值),一个事务跟踪开始;当并发的锁个数低于这个阀值时,事务结束。在这个期间,所有写操作都是该事务的一部分。

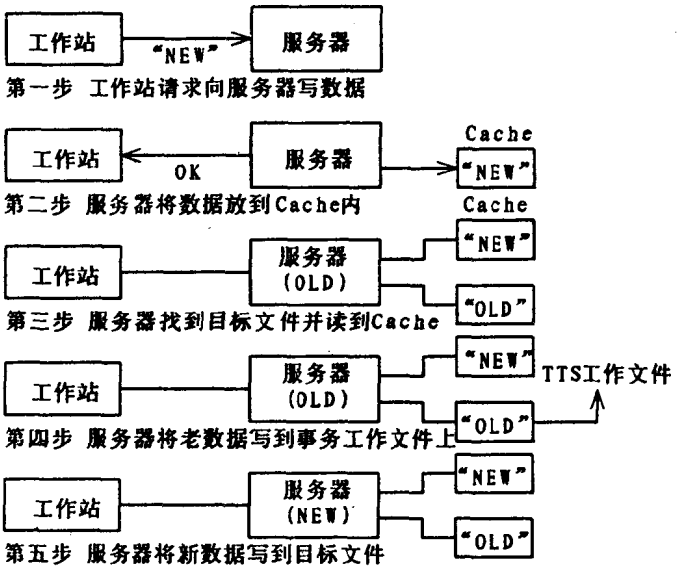


图 1—1 TTS 工作过程

因为一个隐式事务是基于第一个加锁的位置以及最后锁释放情况,程序员必须当心,不要过早地释放一个锁。如果在锁定另一个记录之前解锁了最后一个锁,则本事务便结束,一个新的事务重新开始。这会导致事务分割不正确。通过确保在整个事务完成之前保持所有的锁有效可以避免这一情况发生。但这种方法的不利之处是产生的事务记载文件可能太大,以致不能存放于网络卷上。

这种情况不会在显式跟踪中出现,显式跟踪允许开发者调用 NetWare 系统调用来开始和结束一个事务。这样给程序员带来了极大的灵活性,程序员可以终止一个事务,并监视事务状态以确保其写入到磁盘上。

但是,显式跟踪并非没有危险,这主要是加锁与跟踪交织在一起的原因。因为隐含地,当一个程序加锁时,一个事务便产生了。显式地,当一个事务开始时,加锁便产生了。不管是哪一种情况,TTS 都物理地锁定成为事务一部分的所有记录,锁定在事务期间要写的所有文件,这些锁一直有效,直到事务结束。这样可以保护作为事务的一部分的文件,但当要求在事务结束之前释放某一个记录锁时便会发生问题。

如果这样做,文件服务器仍会返回成功(00)完成码,但它并不会真正解锁该文件,直到该事务结束后才会解锁。

程序员一定要当心,当文件逻辑锁定后,成为事务的一部分时,它们同时被物理锁定了。如果不注意,有可能发生死锁。这是因为两个工作站都解锁(unlocked)它们的记录,但事务还没有结束,然后它们各自去锁对方认为解锁了的文件而实际上文件服务器仍对该文件加锁。这一情况可以通过加锁记录集(record set)而不是单个记录来避免。

在与 TTS 打交道时,死锁只是引起麻烦的一个方面,另外程序员还应当明白为什么会复原(backout),以及用户和系统管理员应当如何干预这种情况。

1.1.4 其它方面

1. 事务复原(backout)

所谓事务复原，是指在一个事务处理完成之前，程序因某种原因而终止，造成这一次事务处理无效。许多事务复原是由于文件服务器或者工作站硬件失效，或操作员误的操作引起的。但是与软件有关的复原也会发生。下面给出事务复原的几种可能情况：

如果在 5 分钟时间内文件服务器还没有收到从一个工作站来的包，则文件服务器就会发一个"看家狗"包给该工作站，如果该工作站不响应，则文件服务器在 15 分钟内不断向该工作站发出"看家狗"包，如果在这段时间内仍得不到响应，则文件服务器便认为该工作站已经不再有效了，于是将它从服务器的连接表中清除出去。

当一个应用程序结束但还有记录没有解锁时，事务复原就会发生。这意谓着一个事务在结束前的任何时候，若程序终止时都会复原，即这一次事务处理无效。例如用户按 CTRL—BREAK，或程序自己终止。如果在一个事务跟踪时文件服务器控制台上发出 DOWN 或 CLEAR STATION 命令，则该事务也会被复原。

如果在一个事务跟踪期间，工作站用户重新引导机器，重新装载 Shell，或重新连入相同的文件服务器时，文件服务器都会做事务复原操作。如果该工作站重新引导后连入另一个文件服务器，则以前文件服务器的"看家狗"进程会侦测到并对相关事务进行复原。

当一个事务复原后，可能用户并不知道什么数据被复原了。文件服务器会给出如下提示：

"Transaction being backed out for station X"

X 代表工作站；或者当文件服务器在事务处理期间垮台后，又再启动时，会出现如下信息：

"Transactions need to be backed out。

Type a character to start the backout"

如果问题与服务器有关，则文件服务器复原可能要用几秒钟的时间。

如果出现事务被复原了，则用户应当查看最后一个记录是否都在。如果一个工作站不能运行下去了，则超级用户可以在文件服务器控制台上用 "CLEAR CONNECTION"命令将它清除掉。对程序员而言，则可以利用"TTS Transaction Status"调用来使应用程序或用户知道一个事务是否成功。

2. 文件服务器上允许或禁止 TTS 功能

利用如下的文件服务器控制台命令可以允许或禁止 TTS 功能：

- DISABLE TRANSACTIONS
- ENABLE TRANSACTIONS

要注意的是，即使当事务被禁止跟踪，"TTS Begin Transaction"和"TTS End Transaction"调用都仍能工作，完全可以标识一个事务的开始和结束，并对有关的记录加锁，同时" TTS Transaction Status" 调用会返回正确的状态。唯一不同的是，当系统出现故障时，无法将事务复原。另外还要注意，事务能否复原还与事务开始时文件服务器的状态有关，当事务开始时文件服务器 TTS 功能是允许的话，则即使在事务中途 TTS 功能被禁止，事务仍可以复原；但如果在事务开始时 TTS 功能是禁止的，则即使在事务中途 TTS 功能允许，事务也不能被复原。

3. 事务长度

尽管一个事务的长度可以任意，但要注意它受磁盘空间的约束。因此，对记录加锁时应当尽可能地短，事务也应当尽可能地短。另外还要注意当 TTS 功能允许时，写操作至少花两倍于禁止 TTS 状态下的时间。有些操作进行 TTS 时空间开销特别大，例如，一个文件从 20MB 截至 5MB 时，至少要求一个 15MB 空间来保存事务原数据。当文件服务器没有空间进行事务跟踪时，会产生如下信息：

"Transaction Tracking disabled because the volume is full."

如果出现这种情况，则事务跟踪被禁止。只有当网络超级用户腾出更多的服务器空间，并在文件服务器控制台上执行 "ENABLE TRANSACTIONS"命令时，TTS 才会重新有效。

4. 磁盘 Cache 缓冲区

当文件服务器的 Cache 缓冲区不够时，系统会变得相当缓慢。TTS 操作比正常操作使用更多的缓冲区，因此，系统至少要为每个工作站配备 8KB 缓冲区。

5. 将数据文件标明为 Transactional 属性

事务跟踪只对已经标明为 Transactional 属性的文件有效。这可以通过 NetWare 的 FLAG 命令使用 /T 参数实现。值得注意的是，Transactional 属性是扩充属性，大多数 BACKUP 在做备份时并不保存它，这样恢复时该属性已经清掉了，事务跟踪时要求重新设置该属性。

1.2 NetWare TTS 系统调用

下面列出的有关 TTS 的系统调用在 SFT Level II NetWare，SFT NetWare 286 和 NetWare 386 以及最新推出的 NetWare 4 中都支持。

1. TTS Is Available(C7h 02h)

功能：这个调用查询缺省文件服务器以确定它是否支持事务跟踪。

参数：输入寄存器包括：

AH　C7h

AL　2

输出寄存器包括：

AL　完成代码

00h　表示不支持事务跟踪；

01h　表示支持事务跟踪；

FDh　表示事务跟踪被禁止。

描述：该调用的 C 接口格式为：

```
int TTSIsAvailable(void);
```

2. TTS Begin Transaction(C7h 00h)

功能：这个调用开始一个显式事务。

参数：输入寄存器包括：

AH　C7h

AL　00h

输出寄存器包括：

AL　完成代码

进位清除　00h　表示成功；

进位设置　FEh　表示隐式事务已经被激活(此时隐式事务被转化为显式事务；

FFh　表示显式事务已经被激活(已存在的显式事务会继续进行)。

描述：该调用的 C 接口格式为：

```
int TTSBeginTransaction(void);
```

3. TTS End Transaction(C7h 01h)

功能：这个调用结束一个显式事务或隐式事务，然后返回，一个引用该事务的代码，该代码用于 TTS Transaction Status 调用中获取该事务的状态。

参数：输入寄存器包括：

AH　C7h

AL　01h

输出寄存器包括：

AL　完成代码

进位清除　00h　表示成功；

FDh　TTS 已经被禁止；

FEh　事务结束记录锁定；

进位设置　FFh　没有显式事务是激活的。

描述：值得注意的是在该调用返回时事务可能没有完全写到磁盘上。建议在使用该调用之前，先刷新磁盘缓冲区，保证事务写到磁盘上。该调用的 C 接口格式为：

```
int TTSBeginTransaction(void);
```

4. TTS Transaction Status(C7h 04h)

功能：这个调用返回事务是否写到了磁盘上，它要用到由 TTS End Transaction 调用返回的事务号码。

参数：输入寄存器包括：

AH　C7h

AL　04h

CX,DX　事务代码

输出寄存器包括：

AL　完成代码

进位清除　00h　表示成功

FFh　事务还没有写入磁盘上。

描述：值得注意的是：一旦“TTS End Transaction”调用执行，文件服务器可能要用 5 秒或更长时间将事务写到磁盘上，对隐式调用当最后一个文件解锁时也是如此，但文件服务器会保证按事务完成的顺序写盘。该调用的 C 格式为：

```
int TTSTransactionStatus(long transactionNumber);
```

5. TTS Abort Transaction(C7h 03h)

功能：这个调用中止显式或隐式事务。

参数：输入寄存器包括：

AH　C7h

AL 03h

输出寄存器包括：

AL 完成代码

进位清除 00h 表示成功；

进位设置 FDh TTS 已经被禁止；

FEh 事务结束记录锁定；

FFh 没有显式事务是激活的。

描述：当该调用返回时，该事务已经完全恢复成功。该调用的 C 格式为：

int TTSAbortTransaction(void)；

6. TTS Get Application Thresholds(C7h 05h)

功能：这个调用返回隐式事务的应用程序阀值。

参数：输入寄存器包括：

AH C7h

AL 05h

输出寄存器包括

AL 完成代码

00h 表示成功；

CL 逻辑记录锁定阀值(0—255)；

CH 物理记录锁定阀值(0—255)。

描述：利用这个调用，可以规定应用程序只有当第 n 个锁在一个文件中出现时才开始隐式锁定。这对于在程序执行过程中维护锁很有用，以便阻止所有的写磁盘操作构成大的事务。该调用可以与 TTS Set Application Threshold 一起使用来保存，设置和恢复到原来值。该调用的 C 格式为：

int TTSGetApplicationThresholds(BYTE * logicalRecordThreshold, BYTE * physicalRecordThreshold)；

7. TTS Set Application Thresholds(C7h 06h)

功能：这个调用为隐式事务设置应用程序阀值。

参数：输入寄存器包括：

AH C7h

AL 06h

CL 逻辑记录锁定阀值(0—255)；

CH 物理记录锁定阀值(0—255)。

输出寄存器包括：

AL 完成代码

00h 表示成功；

描述：阀值包含逻辑和物理锁。该调用的 C 格式为：

int TTSSetApplicationThresholds (BYTE logicalRecordThreshold, BYTE physicalRecordThreshold)

8. TTS Get Workstation Thresholds(C7h 07h)

功能：这个调用返回隐式事务工作站阀值。

参数:输入寄存器包括:

AH C7h

AL 07h

输出寄存器包括:

AL 完成代码

00h 表示成功;

CL 逻辑记录锁定阀值(0—255);

CH 物理记录锁定阀值(0—255)。

描述:利用这个调用可以得到工作站的逻辑和物理锁极限值。工作站阀值是永久的,当一个应用程序结束时它仍有效。该调用用"TTS Set Workstation Thresholds"来保存,设置阀值,然后恢复其原来的值。缺省阀值为 0,当设置为 FFh 时表示将隐式事务关掉。

该调用的 C 格式为:

int TTSGetWorkstationThresholds (BYTE * logicalRecordLockThreshold, BYTE *physicalRecordLockThreshold);

9. TTS Set Workstation Thresholds(C7h 08h)

功能:这个调用为隐式事务设置工作站阀值。

参数:输入寄存器包括:

AH C7h

AL 08h

CL 逻辑记录锁定阀值(0—255);

CH 物理记录锁定阀值(0—255)。

输出寄存器包括:

AL 完成代码

00h 表示成功;

描述:该调用的 C 格式为:

int TTSSetWorkstationThresholds (BYTE logicalRecordLockThreshold, BYTE physicalRecordLockThreshold);

1.3 实例分析

下面的程序(TestTTS.C)代码通过模拟一个复原事务来测试 TTS 系统。该程序执行一个显示事务来创建 2 个文件并将字符"1"写入这两个文件。第二个事务写入一个"2",然后中断。事务跟踪系统应当恢复第二个事务而使得第一个文件的 1 仍在。

该程序的工作过程大致如下:先调用 OpenOrCreate()创建两个文件。要注意的是,当文件的 Transactional 属性设置后,应当关闭该文件并重新打开它使得 TTS 能识别 Transactional 属性。

如果文件成功地打开或创建后,返回的指针存于 handle1 和 handle2 中,下一步是开始一个显式事务并将"1"写入这两个文件。该事务正常结束后程序进入一个循环,等待事务状态以便通知事务已经写入。然后程序开始另一个事务并写入"2"到第一个文件,这时程序等待用户输入键,一旦用户输入 Ctrl—Break 终止程序运行,则当用 TYPE 查看它们时会

发现两个文件中都没有“2”，而“1”在两个文件中都存在。下面是源程序清单。

```
/*************************************************************/
/* Name: TestTTS.C                                           */
/* Function:                                                 */
/*  This program tests TTS functions.                        */
/*************************************************************/
#include <stdio.h>
#include <dos.h>
#include <dir.h>
#include <string.h>
#include <fcntl.h>
#include <stdlib.h>
#include <conio.h>

#define TRANSACTIONAL 0x10
#define SHAREABLE 0x80

typedef unsigned char BYTE;
typedef unsigned int WORD;
typedef unsigned long LONG;

/* function prototypes */
int OpenOrCreate(char *fileName);
int GetExtFileAttributes(char *path,BYTE *attributes);
int SetExtFileAttributes(char *path,BYTE *attributes);
int TTSBeginTransaction(void);
int TTSEndTransaction(WORD *transactionNumber);
int TTSTransactionStatus(WORD *transactionNumber);

main()
{
    int handle1,handle2;
    LONG transactionNumber;

    clrscr();

    /* Show TTS Working */
    gotoxy(1,20);
    printf("Opening files.....");
    /* Attempt to open or create the files */
```

```
handle1 = OpenOrCreate("Test1.dat");
handle2 = OpenOrCreate("Test2.dat");

/* Write some text */
gotoxy(1,21);
printf("Writing header to file...");

write(handle1,"This is file1,value is : :\n",28);
write(handle2,"This is file2,value is : :\n",28);

/Start an explicit transaction */
TTSBeginTransaction();

/* Write to two files */
gotoxy(1,22);
printf("Started transaction,writing value to files...");
lseek(handle1,(long)25,0);
write(handle1,"1",1);

lseek(handle2,(long)25,0);
write(handle2,"1",1);

/* End transaction */
TTSEndTransaction(&transactionNumber);

/* Wait on transaction status */
gotoxy(1,23);
printf("Transaction complete. Waiting for server to write....");

while(TTSTransactionStatus(&transactionNumber));

printf("Written.");

gotoxy(1,24);
printf("Press any key to continue");
getch();

/* Show a backed out transaction */

/* Start an explicit transaction */
```