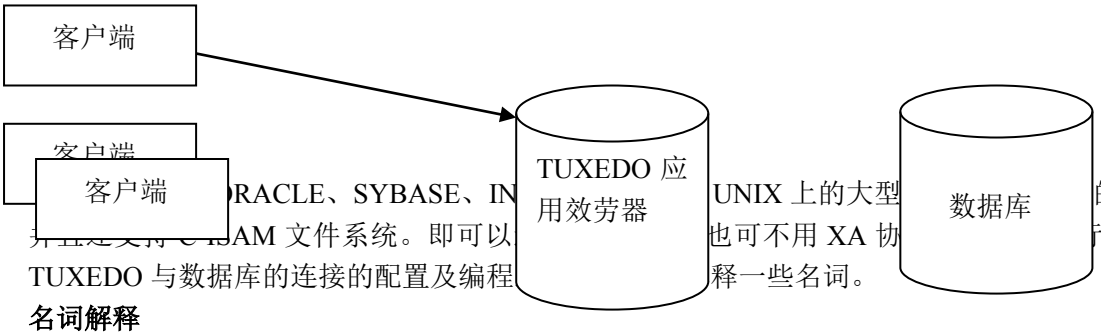
9.1 TUXEDO 如何处理分布式事务

在两层的 C/S 结构中，客户端直接访问数据库，当采用 TUXEDO 中间件后，形成三层结构。这时，客户端不直接访问数据库，而是改为调用中间件 TUXEDO 效劳端上的效劳，由 TUXEDO 效劳端访问数据库，并把结果返回给客户端。如下图。



RACLE、SYBASE、IN UNIX 上的大型 的 SQL Server，AM 文件系统。即可以 也可不用 XA 协 连接。在说明 TUXEDO 与数据库的连接的配置及编程 释一些名词。

名词解释

资源管理器(RESOURCE MANAGER)：

最常见的是数据库，可以是其他的，如 TUXEDO 的 QUEUE，EJB 的 JMS 等,它们对数据进行管理和维护.

事务(TRANSACTION)：

事务的定义很多，简单地说,事务是对资源管理器的一组操作，它使所涉及的资源管理器从一个状态转变到另一个状态，这些操作要么全部成功，要么全部失败。事务具有以下的 4 个特征(一般称为 ACID):

原子性(ATOMICITY): 指事务中的所有操作作为一个整体单元要么成功要么失败.

一致性(CONSISTENCY):一致性意味着不管事务提交或放弃,参与事务的所以资源管理器在事务结束后都保持一种合法的状态.一致性也意味着,当一个事务结束时,所有的参与者都要释放它所锁住的资源.

隔离性(ISOLATION):隔离性意味着事务正在处理过程中,在事务外面无法看到事务处理的中间结果.

持久性(DURABILITY):使事务的最终结果已被真正写到磁盘系统中.

本地事务(LOCAL TRANSACTION)：

如果一个事务只涉及到一个资源管理器，那么该事务称为本地事务。在 TUXEDO 中,不通过 XA 接口的事务都是本地事务,如:在 ORACLE 中,用 EXEC CONNECT 建立与数据库的连接,并用 EXEC COMMIT 提交一个事务,那么该事务就是本地事务.

全局事务(GLOBAL TRANSACTION)：

全局事务涉及到一个或多个资源管理器，它也称为分布式事务(DISTRBUTED TRANSACTION)，对所有涉及的资源管理器的操作必须被看作单个工作单元。它们必须被同步，并在所有效劳器上圆满完成，否那么，就必须被彻底取消。例如:一个效劳器在写过程中被关闭，那么事务处理中其他系统上的所有写的东西就必须被取消。在 TUXEDO 中，采用 XA 接口的事务都是全局事务。全局事务是相对 LT 而言的，它也有 AICD 四个特性，所不同的是它可以跨越多个资源管理器，这些资源管理器可能在不同的平台上。在 TUXEDO 中，一个全局事务最多可跨越 16 个不同的资源管理器.

事务管理器(TRANSACTIN MORNITOR)：

管理协调参与全局事务的各个资源管理器的准备，提交及回滚等操作，事务管理器还在出现场地故障、网络故障或全局资源死锁时协调全局事务的恢复。TUXEDO 在全局事务中就充当事务管理器的作用。在一个全局事务中有一个事务协调器，有一个以上的资源管理器。事务协调器与资源管理器之间采用 XA 协议进行通讯在 TUXEDO 中一个 GROUP 只能有以个资源管理器，所以一个全局事务会跨越多个 GROUP

XA 协议:

XA 协议由 TUXEDO 首先提出,并交给 X/Open 组织,作为资源管理器(数据库)与事务管理器的接口标准。Informix 是最早宣布支持 XA 协议的数据库厂家，Informix5.0 以上的版本都提供 XA 接口，以实现与 TUXEDO 的连接。目前，Oracle、Informix、DB2、Sybase 等各大数据库厂家都提供对 XA 的支持.XA 协议采用两阶段提交方式来管理分布式事务.XA 接口提供资源管理器与事务管理器之间的进行通讯的标准接口,TUXEDO 支持根本的 XA 标准

(PRELIMINARY XA SPECIFICATION),及最终的 XA 标准(THE FINAL SPECIFICATION).XA 协议包括两套函数,以 xa_开头的及以 ax_开头的.
以下的函数使事务管理器可对 RM 进行操作
xa_open, xa_close:建立,关闭与 RM 的连接
xa_start,xa_end:开始,结束一个本地事务
xa_prepare,xa_commit,xa_rollback:预提交,提交,回滚一个本地事务
xa_recover:回滚一个已进行预提交的事务
ax_开头的函数使 RM 可以动态在事务管理器中进行注册,并可以对 XID(TRANSACTION IDS)进行操作.
说明: 在 FINAL XA SPECIFICATION 中,用 XID 代替全局事务 RID
ax_reg,ax_unreg:允许一个 RM 在一个 TMS(TRANSACTOIN MANAGER SERVER)中动态注册或撤消注册.
全局事务 rid_开头的函数在 PRELIMINARY XA SPECIFICATION 中有,在 FINAL XA SPECIFICATION 中没有定义.
全局事务 rid_cmp:比拟两个全局事务 RID
全局事务 rid_fmt:格式化一个全局事务 RID,以便打印
全局事务 rid_hash:根据全局事务 RID,生成一个 HASH 值.
现在主要的数据库都支持 FINAL XA SPECIFICATION

TUXEDO 中的全局事务有以下特点：

1. 可以在客户端或效劳端开始一个全局事务
2. 在 TUXEDO 中全局事务能跨越多个进程
3. 每个全局事务有一个唯一的 ID 号〔全局事务 RID〕标识，它可在 TUXEDO 的进程间传递。
4. 全局事务可以跨越 DOMAIN

一个全局事务从发起到提交的过程：

全局事务的提交采用两阶段提交方式.
在两阶段提交过程中,应用程序是事务提交得发起者,应用程序通过调用 TPCOMMIT()开始一个事务的提交,该应用所在 GROUP 的 TMS 是这个事务的协调者(COORDINATOR),我们称之为 COORDINATOR TMS,其他参与本领务的 GROUP 所对应的 TMS,我们称之为:SUBORDINATE TMS,CORDINATOR TMS 负责与所有参与本全局事务的 RM 的通讯.完成该事务的提交,整个过程如下:

1. 应用程序通过调用 TPCOMMIT()开始一个事务的提交
2. CORDINATOR TMS 给参与该全局事务的每个 GROUP 中的 TMS 发送请求,每个 TMS 要求 RM 进行预提交操作
3. 每个 RM 各自进行预提交,预提交指把要更改的数据写到磁盘上,以便在失败时可以进行恢复.但没有真正更新 RM
4. 当 RM 预提交完毕,每个 SUBORDINATE TMS 都把预提交操作结果(成功或失败)告诉给 COORDINATOR TMS,
5. 如果有任何一个 RM 预提交失败,或 COORDINATOR TMS 得不到它的应答,那么 COORDINATOR TMS 告诉所有的 RM 回滚它们各自的本地事务.
   如果所有的 RM 预提交都成功,那么 COORDINATOR TMS 在它所在机器的 TLOG 文件中写一条记录,内容包括该全局事务的全局事务 RID，参与该全局事务的所以 RM 的列表及

其他信息.以便在第二阶段提交失败时回滚.

6.下一步做什么取决于在 UBBCONFIG 中 TP_COMMIT_CONTROL 得设置

TP_CMT_LOGGED

TPCOMMIT()调用返回,程序继续往下走,各个 RM 各自进行真正得提交操作,如果在第二阶段提交失败,TMS 时是不知道的.而这是 TPCOMMIT()已返回成功.

TP_CMT_COMPLETE

TPCOMMIT()会等到第二阶段提交完成才返回,如果在第二阶段提交失败,TMS 是知道的.

并且 TPCOMMIT()会返回失败. 默认值是 TP_CMT_COMPLETE

事务模式和非事务模式:

TUXEDO 的应用程序可分为两种,事务模式和非事务模式,有以下两种方式可以使一个 TUXEDO 应用处于事务模式下:

1.显式: 通过调用 tpbegin(),显式开始一个全局事务

2.隐式: 在 UBBCONFIG 中对一个 SERVICE 设置了 AUTOTRAN=Y,当该 SERVICE 被调用时,TUXEDO 会自动启动一个全局事务

注意：AUTOTRAN=Y 只对全局事务起作用，对本地事务不起作用。

例如：下面的程序中 SERVICE A 中调用了 SERVICE B

```
A(TPSRCINFO *RQST)
{
    tpcall(B,.....,flags)
}
```

那么：

- 如果 A 当前不处于事务模式中,A 在 UBBCONFIG 中设置了 AUTOTRAN=Y,那么系统自动起一个全局事务
- 如果 A 当前已处于事务模式中,A 在 UBBCONFIG 中设置了 AUTOTRAN=Y,那么:
  1. 如果 tpcall()中的 flags 没有设置 TPNOTRAN〔默认方式是不设置〕那么 SERVICE：B 参与当前的事务。
  2. 如果 tpcall()中的 flags 为 TPNOTRAN,那么 SERVICE：B 不参与当前的事务。

注意：如果在 UBBCONFIG 中对 SERVICE B 设置了 AUTOTRAN=Y,那么当前的事务被自动挂起,系统为 B 自动起一个新的事务,如果 B 中的事务失败了，对 A 中的事务没有影响。

TUXEDO 中与全局事务有关的设置:

与全局事务有关的设置包括:RESOURCE,MACHINE,GROUPS,SERVICE 4 个节

在 RESOURCE 中设置主要有:

MAXGTT〔MAX Global Transaction〕

在任一时刻在某一台效劳器上最多可以有多少个全局事务存在,也就是可以最多有多少个未提交的全局事务,范围:0-32767。0 意味着该系统不支持事务,默认值是 100.该值也可在 MACHIENS 中设置。在 MACHINES 中设置的值回覆盖在 RESOURCE 中的设置值。

GTT(Global Transaction Table)

TUXEDO 在 BULLETIN BOARD 中维护的一张表,用于记录全局事务的状态信息,在该台效劳器上发起的或该台效劳器参与的每个全局事务,在 GTT 中都对应一条记录.每台效劳器的 GTT 中最多可以有多少条全局事务的记录是由 MAXGTT 决定的,当一个全局事务成功提交时,它在 GTT 中的记录将被删除.如果该全局事务提交失败,它在 GTT 中的记录还会保持一段时间,一般是 5 分钟.所以 GTT 应该大于 TLOGSIZE。GTT 的范围为 0-64,000,默认值为 100,可以在 MACHINE 中覆盖该设置.

CMTRET:

设置 TP_COMMIT_CONTROL 的初始值,该值可以由在 tpscmt()中重新设置.

与 MACHINES 有关的设置:

在 MACHINES 中设置每台机器中的全局事务 日志文件的位置和大小.

TLOGDEVICE:

指定包含 TLOG 文件的 TUXEDO 文件系统名,长度不能超个 64 个字符

TLOGNAME:
TLOG 文件的名称
TLOGSIZE:
TLOG 文件的大小，它的大小为 0 <= TLOGSIZE <= 2048,默认值为 100
如果一个全局事务跨越多台效劳器,在每台效劳器上都应该有一个 TLOG 文件,用于记录全局事务的信息,在全局事务要回滚时要用到记录在 TLOG 中的信息.TLOG 可以创立在裸设备上.如果一个全局事务在该台机器上发起但还没有提交,那么它在 TLOG 文件种占有一页的空间,在一般的平台上一页的大小使 512 字节,,当该全局事务成功提交或回滚后,它在 TLOG 中的记
录将被删除。
与 GROUPS 有关的设置:
在 TUXEDO 中一个 GROUP 中只能定义一个资源管理器,如果一个 TUXEDO 系统有多个资源管理器,就要定义多个 GROUP 才行.在 GROUP 中定义 TMS 的名称及个数,翻开,关闭该资源管理器的参数,不同的资源管理器的设置都不一样.
SERVICES 中的设置:
AUTOTRAN:
如果一个客户端调用了设置 AUTOTRAN=Y 的 SERVICE,并且该客户端当前不在事务模式下,那么 TUXEDO 自动为该 SERVICE 启动一个全局事务.
TRANSTIME:
对因设置了 AUTOTRAN 的而起动的事务设置超时时间.默认为 30 秒,0 表示没有超时。
全局事务中用到的函数
在 TUXEDO 中有两套操作全局事务的函数
ATMI
TUXEDO 自己提供的函数
TX
由 X/OPEN 定义的函数
它们根本差不多,因为 TX 是以 ATMI 为根底定义的.区别
1. TX 提供 CHAINED TRANSACTIONS,ATMI 不提供
2. TX 以 tx_开始,而 ATMI 以 tp 开始,如 tx_begin(),tpbegin()
3. TX 在 tx.h 中定义,ATMI 在 atmi.h 中定义

TX_开头的函数一般很少用，在此不作介绍。
int tpopen(void)
描述：建立与一个资源管理器得连接,该资源管理器是该 SERVER 所在得组中定义的.并且在 buildserver 时要用 -r 参数指定该资源管理器.如果采用默认的 TPSVRINIT()该函数会自动被调用。
参数：无
返回值：失败返回-1, 错误号保存在全局变量 tperrno 中。
int tpclose(void)
描述：关闭与一个资源管理器得连接,如果采用默认的 TPDONE()该函数会自动被调用,
参数：无
返回值：失败返回-1，错误号保存在全局变量 tperrno 中。
int tpbegin(unsigned long timeout, long flags)
描述：开始一个全局事务
参数：
timeout：全局事务的超时时间，从 tpbegin()开始,过了 TIMEOUT 秒该全局事务还没有完成,
该全局事务将因超时而回滚，范围为 0 to $2^{31}-1$ 秒， 0 表示可以为无穷长。注意:TIMEOUT 应该大于 SCANUNIT
FLAGS：保存值，现在没有用到，要设为 0

返回值：失败返回-1，错误号保存在全局变量 tperrno 中。

int tpcommit(long flags)

描述：对一个全局事务进行提交操作

参数：FLAG 应该设为 0

返回值：失败返回-1，错误号保存在全局变量 tperrno 中。

int tpabort(long flags)

描述：对当前的全局事务进行回滚

参数：FLAG 应该设为 0

返回值：失败返回-1，错误号保存在全局变量 tperrno 中。

int tpsuspend(TPTRANID *tranid, long flags)

描述：挂起当前的全局事务,一个全局事务的发起进程 A 可以调用它,并把得到的句柄传给在同一台机器上另一个进程 B,在 B 中可以调研 TPRESUME()重起该全局事务.如果不是在一个全局事务的发起进程中调用 TPSUSPEND(),那么只能在进程中调用 TPRESUME().

注意:一个全局事务被挂起之后,超时仍然起作用.

参数：

tranid：如果调用成功，返回一个当前全局事务的句柄

FLAG 应该设为 0

返回值：失败返回-1, 错误号保存在全局变量 tperrno 中。

成功返回一个当前全局事务的句柄,

int tpresume(TPTRANID *tranid, long flags)

描述：重新开始一个全局事务

参数：

tranid：要重新开始的全局事务的句柄，为 tpsuspend〔〕中参数 tranid 返回值

flags： 应该设为 0

返回值：失败返回-1，错误号保存在全局变量 tperrno 中。

int tpscmt(long flags) ()

描述：设置 TP_COMMIT_CONTROL 得值

参数：FLAG 有两个值

TP_CMT_LOGGED: TPCOMMIT〔〕在第一阶段完成之后就返回.

TP_CMT_COMPLETE:TPCOMMIT〔〕在第二阶段完成之后才返回.

返回值：失败返回-1，错误号保存在全局变量 tperrno 中。

int tpgetlev()

描述：返回当前所处得全局事务状态.

参数：

返回值：1:在一个全局事务中,0:不在一个全局事务中,

失败返回-1，错误号保存在全局变量 tperrno 中。

9.2 ORACLE 数据库 XA 的配置

TUXEDO 效劳器可以和 ORACLE 数据库在同一台效劳器上，也可以在不同的机器上，如果在不同的机器上，在 TUXEDO 的效劳器上安装一个 ORACLE 的客户端。从上一节的内容我们知道 TUXEDO 效劳器与 ORACLE 数据库连接有两种方式：

1、不通过 XA 接口直接互连。适用于整个系统只有一个数据库的情况。不需要作任何的配置。

2、通过 XA 接口互连，对整个系统有一个数据库或多个数据库都适用，要进行一些配置工作才行。下面介绍采用这种互连方式的配置方法。

系统说明:

TUXEDO 版本:7.1 安装目录 d:\tuxedo71

ORACLE 版本:8.1.5 安装目录 d:\ora81

操作系统： win2000

配置的步骤：

一、ORACLE 的的配置

1. 用 internal 用户〔缺省的口令是 oracle〕进入 SQLPLUS

   C:\>sqlplus internal/oracle

2. 运行 ORACLE 的安装路径下的/rdbms/admin/xaview.sql

   SQL> @d:\ora81\rdbms\admin\xaview.sql

3. 授权

   SQL>grant select on v$xatrans$ to public with grant option;

   SQL>grant select on v$pending_xatrans$ to public with grant option;

4. 用 system 用户〔缺省的口令是 manager〕连接并授权

SQL>connect system/manager

SQL>grant select any table to public;

二、TUXEDO 的配置

1. 修改 TUXEDO 安装路径的 udataobj 目录下的 RM 文件，把以 Oracle_XA:xaosw:开头的一行用#注释掉，并参加一行：

   Oracle_XA;xaosw;d:\ora81\rdbms\xa\oraxa8.lib d:\ora81\precomp\lib\msvc\orasql8.lib

   如果是在 UNIX 环境下，那么为：

   Oracle_XA:xaosw:-L${ORACLE_HOME}/lib -lclntsh

2. 在 TUXEDO 用户下创立 TMS 文件:TMS_ORA8i，TUXEDO 通过 TMS_ORA8i 与 ORACLE 数据库采用 XA 协议进行通讯

buildtms -o d:\tuxedo71\bin\TMS_ORA8i -r Oracle_XA

注意:如果 TUXEDO 效劳端与 ORACLE 数据库不在同一台效劳器上,可能会提示找不到库文件 oraxa8.lib 和 orasql8.lib,可到 ORACLE 数据库的效劳端相应目录下把这两个文件拷到当前机器 ORACLE 的客户端下的对应目录下.

3. 配置 UBBCONFIG

   1. 在*MACHINES 节中增加：

      TLOGDEVICE = "/home/oracle/temp/simpdb/TLOG"

      TLOGNAME=TLOG

      TLOGSIZE=200

   2. 改*GROUPS 节的配置为：(scott/tiger 为本数据库所采用的用户及口令,可根据需要更改)

      *GROUPS

      GROUP1 LMID=simple GRPNO=1

      OPENINFO="Oracle_XA:Oracle_XA+Acc=P/scott/tiger+SesTm=600+MaxCur=5+LogDir=."

TMSNAME="TMS_ORA8i" TMSCOUNT=2

修改后的配置文件 ubb 内容如下

```
IPCKEY      123456
DOMAINID    simpapp
MASTER      simple
MAXACCESSERS    100
MAXSERVERS  50
MAXSERVICES 100
MODEL       SHM
LDBAL       N
```

```
*MACHINES
server      LMID=simple
            APPDIR="d:\test"
            TUXCONFIG="d:\test\tuxconfig"
            TUXDIR="d:\tux71"
            TLOGDEVICE = "d:\test\TLOG"
            TLOGNAME=TLOG
            TLOGSIZE=100
*GROUPS
        GROUP1  LMID=simple
        GRPNO=1OPENINFO="Oracle_XA:Oracle_XA+Acc=P/scott/tiger+SqlNet=DEMO+SesTm=600+M
        axCur=5+LogDir=."
        TMSNAME="TMS_ORA8i" TMSCOUNT=2
*SERVERS
        DEFAULT:
            CLOPT="-A"
        test    SRVGRP=GROUP1 SRVID=1
*SERVICES
```

说明：OPENINFO 的含义

P: 指定用户名与口令,通过该用户名与口令与数据库建立连接,上面的配置中用户为 scott,口令为 tiger

SqlNet: 如果 TUXEDO 效劳器与数据库效劳器不在同一台机器上，TUXEDO 上的 ORACLE 客户端通过网络方式与数据库效劳器建立连接，

在 OPENINFO 中的 SqlNet 指定该数据库的 SID，如在上面的配置中该数据库的 SID=DEMO，如果不 SqlNet 指定，在生成的 TRC 文件中会有如下错误:

```
xaolgn_help:XAER_RMERR:OCIServerAttach failer. ORA-12154
```

ORA-12154: TNS: 无法处理效劳名

```
SesTm(Session time limit): maximum time a transaction can be inactive
```

MaxCur: 最多可以同时翻开多少个 CURSOR

4.重命名以下文件,因为以下文件名与 ORACLE 带的文件名有冲突，所以要改名。

1.TUXEDO 安装路径 include 目录下的下面文件

2.重命名 TUXEDO 安装路径 lib 目录下的下面文件

5.用 TMADMIN 创立 TLOG 文件，TUXEDO 用一个文件 TLOG 记录对数据库操作的日志。用于协调分布式数据库的提交与回滚.

```
D:\>tmadmin
>crdl -b 500 -z d:\test\TLOG
>crlog -m simple
>q
```

三、效劳端的程序：test.pc

功能:根据客户端传的 EMPNO 到表 EMP 中取 ENAME 的值,并把它返回给客户端

```
#include <stdio.h>
#include <atmi.h>
#include <userlog.h>
EXEC SQL INCLUDE sqlca;
EXEC SQL BEGIN DECLARE SECTION;
    long al_empno=0;
```

```
    char ac_ename[11]="";
    EXEC SQL VAR  ac_ename      IS STRING(11);
EXEC SQL END DECLARE SECTION;
TEST(TPSVCINFO *rqst)
{
    /*接收客户端来的数据*/
    al_empno = (FBFR32 *)rqst->data;
    EXEC SQL select ename into :ac_ename from EMP where empno=:al_empno;
    if(sqlca.sqlcode!=0)
    {
        userlog("select  from  EMP  failure,sqlcode=%ld,  sqlerr=%s\n",sqlca.sqlcode,(char
*)sqlca.sqlerrm.sqlerrmc);
        strcpy(rqst->data,sqlca.sqlerrm.sqlerrmc);
        tpreturn( TPFAIL, 0, rqst->data, 0, 0 );
    }
    /*把取出的结果返回给客户端*/
    strcpy(rqst->data,ac_ename);
    tpreturn( TPSUCCESS, 0, rqst->data, 0, 0 );
}
```

四、编写客户端程序: testcli.c

功能:调用 TUXEDO 效劳端的效劳 TEST,取 EMPNO=1000 所对应的 ENAME 的值,并显示出来

```
#include <stdio.h>
#include "atmi.h"
main(argc, argv)
{
    long reqlen=1024;
    char *reqbuf;
    /* 与 TUXEDO 效劳端建立连接 */
    if (tpinit((TPINIT *) NULL) == -1)
    {
        (void) fprintf(stderr, "Tpinit failed\n");
        exit(1);
    }
    /* 分配发送缓冲区*/
    reqbuf = (char *)tpalloc("STRING",NULL,reqlen);
    if ( reqbuf  == (char *)NULL)
     {
        printf("tpalloc failed\n");
        tpterm();
    }
    strcpy(reqbuf,"1000");
    /*调用 TUXEDO 的效劳 TEST*/
    if (tpcall("TEST", (char *)reqbuf, 0L, (char **)&reqbuf, (long *)&reqlen, 0< 0 )
    {
        printf("tpcall failed,tperrno=%ld,tperrtext=%s\n",tperrno,tpstrerror(tperrno));
        tpfree(reqbuf);
```

```
        tpterm();
        exit(1);
    }
    printf("name=%s\n",reqbuf);
    tpfree(reqbuf);
    tpterm();
    return(0);
}
```

五、编译效劳端程序

1.用 ORACLE 的 PROC 把 test.pc 文件预编译成 test.c 文件

```
d:\test> proc test.pc include=%TUXDIR%/include
```

2.用 buildserver 把 test.c 编译成可执行文件,注意-r 后带的 Oracle_XA 与 RM 文件中的一致。

```
d:\test> buildserver -o test -f test.c -r Oracle_XA -s TEST
```

六、编译客户端程序

```
d:\test> buildclient -o testcli -f testcli.c
```

七、启动 TUXEDO 应用系统

应能看到所有的 SERVER 都启动成功.这时,我们的效劳端程序 test会自动与 ORACLE 数据库建立连接,并一直保持这个连接,直到 TUXEDO 系统或 ORACLE 数据库关闭.所以在我们的程序 test.pc 中看不到与数据库连接的语句,因为现在与

数据库的连接由 TUXEDO 自动管理.如果 TMS_ORA8i 启动失败会在当前目录生成一个*.trc 文件,记录失败的原因,同时 TUXEDO 的 ULOG

文件中也会有一些错误信息.可参考这些错误信息.进行错误分析.

```
d:\test> tmboot -y
exec TMS_ORA8i-A :
    process id=1072 ... Started.
exec TMS_ORA8i-A :
    process id=528 ... Started.
exec test -A :
    process id=876 ... Started.
```

八、运行客户端程序,应能看到效劳端返回的结果

```
d:\test> testcli
name=bill
```

到此,整个配置过程就大功告成了.ORACLE 的其他版本的配置及在其他操作系统上的配置根本与本文所述差不多,差异主要在 RM 文件中所连的库文件可能会不样.

9.3 INFORMIX 数据库 XA 的配置

同采用 ORACLE 数据库一样，TUXEDO 效劳器可以和 INFORMIX 数据库在同一台效劳器上，也可以在不同的机器上，如果在不同的机器上，在 TUXEDO 的效劳器上安装一个 INFORMIX 的客户端。从上一节的内容我们知道 TUXEDO 效劳器与 INFORMIX 数据库连接有两种方式：

1、不通过 XA 接口直接互连。适用于整个系统只有一个数据库的情况。不需要作任何的配置。

2、通过 XA 接口互连，对整个系统有一个数据库或多个数据库都适用，要进行一些配置工作才行。下面介绍采用这种互连方式的配置方法。

系统说明:

TUXEDO: 版本 TUXEDO6.5(是 32 位的)安装在 HP-UX 11.0 64bit 上，
安装目录 /usr/tuxedo

TUXEDO 的例子: /usr/tuxedo/simpdb

INFORMIX: 版本 INFORMIX9.21(是 64 位的)安装在 SCO Unix 5.0.5 上，
安装目录 /INFORMIX

数据库名称: mydb

TUXEDO 用户名： TUXEDO

注意：如果 TUXEDO 系统是 32 位的，而 INFORMIX 数据库的效劳端是 64 是，在 TUXEDO 系统所在的机器上应安装 INFORMIX 数据库的 32 位的客户端才行。

配置的步骤：

一、INFORMIX 的的配置

1、 数据库一定要以 unbuffered log 方式创立，

create database databasename with log;

INFORMIX 数据库的 LOG 方式有 3 种:Buffered,Nobuffer,Unbuffered(under buffer)

用 onmonitor 命令可查看数据库是否是用 unbuffered log 方式创立的,log status 那一列为 U 的是 unbuffered log 方式。

用 ontape -s -L 0 -U databasename;可把一个其他方式创立的数据库改为 unbuffered log 方式的.

2、 tuxedo 用户应该有访问该数据库资源的权限。grant dba to tuxedo;

如果 TUXEDO 用户没有访问该数据库资源的权限，当 TUXEDO 启动时,TMS 启动会失败,在 ULOG 中会出现类似下面的错误信息：

```
145053.rs6000!BBL.17510: LIBTUX_CAT:262: INFO: Standard main starting
145053.rs6000!TMS_INFORMIX.20204: 020602: TUXEDO Version 6.5 AIX  2 4
007025954C00.
145053.rs6000!TMS_INFORMIX.20204: LIBTUX_CAT:262: INFO: Standard main
starting
145054.rs6000!TMS_INFORMIX.20204: LIBTUX_CAT:466: ERROR: tpopen TPERMERR
xa_open returned XAER_RMERR
145054.rs6000!TMS_INFORMIX.20204: LIBTUX_CAT:250: ERROR: tpsvrinit() failed
145054.rs6000!TMS_INFORMIX.20204: LIBTUX_CAT:300: ERROR: _tlog_open:
_gp_tblopen
145054.rs6000!TMS_INFORMIX.20204: LIBTUX_CAT:250: ERROR: tpsvrinit() failed
145054.rs6000!TMS_INFORMIX.20204: LIBTUX_CAT:300: ERROR: _tlog_open:
_gp_tblopen: UNIX sys call error - 2
145054.rs6000!tmboot.19178: 020602: TUXEDO Version 6.5 AIX  2 4 007025954C00.
145054.rs6000!tmboot.19178: CMDTUX_CAT:825: ERROR: Process TMS_INFORMIX at
simple failed with /T tperrno (TPERMERR -          resource manager error)
```

二、TUXEDO 的配置

1. 设置环境变量：〔在文件/usr/tuxedo/simpdb/setenv 中〕

```
. usr/tuxedo/tux.env
INFORMIXDIR=/tmp_mnt/informix/hc; export INFORMIXDIR
INFORMIXSERVER=dhc; export INFORMIXSERVER
PATH=$TUXDIR/bin:$INFORMIXDIR/bin:/bin:/usr/bin:/usr/ccs/bin:.; export
PATH
SHLIB_PATH=$SHLIB_PATH:$INFORMIXDIR/lib:$INFORMIXDIR/lib/esql:$INFORMIXDI
R/lib/cli:$INFORMIXDIR/lib/c++
:$INFORMIXDIR/lib/client:$INFORMIXDIR/lib/dmi:/usr/lib:/usr/lib/Motif1.2
INCLUDE=$INFORMIXDIR/incl/esql:$INFORMIXDIR/incl:/tuxedo/include:/usr/include;
```

export INCLUDE

CFLAGS="-I$INFORMIXDIR/incl -I$INFORMIXDIR/incl/esql" export CFLAGS

2. 重命名以下文件,因为以下文件名与 INFORMIX 中的文件名有冲突，所以要改名。

1.TUXEDO 安装路径 include 目录下的下面文件

2.重命名 TUXEDO 安装路径 lib 目录下的下面文件

3. 修改 TUXEDO 安装路径的 udataobj 目录下的 RM 文件，参加：

INFORMIX-DSHC:infx_xa_switch:-L/tuxedo/lib -L${INFORMIXDIR}/lib -L${INFORMIXDIR}/lib/esql -lifxa -lifsql -lifasf -lifgen -lifos -lifgls -lnsl -lm -lsec ${INFORMIXDIR}/lib/esql/checkapi.o -lifglx

4. 在 TUXEDO 用户下创立 TMS 文件:TMS_INFORMIX，TUXEDO 通过 TMS_INFORMIX 与 INFORMIX 数据库采用 XA 协议进行通讯

buildtms -r INFORMIX-DSHC -o /tuxedo/bin/TMS_INFORMIX

5. 配置 UBBCONFIG

1. 在*MACHINES 节中增加：

TLOGDEVICE = "/usr/tuxedo/simpdb/TLOG"

TLOGNAME=TLOG

TLOGSIZE=200

2. 改*GROUPS 节的配置为:

*GROUPS

GROUP1 LMID=simple GRPNO=1

TMSNAME="TMS_INFORMIX" TMSCOUNT=2

OPENINFO="INFORMIX-DSHC:mydb"

修改后的配置文件 ubb 内容如下

```
IPCKEY      123456
DOMAINID    simpapp
MASTER      simple
MAXACCESSERS    100
MAXSERVERS  50
MAXSERVICES 100
MODEL       SHM
LDBAL       N
*MACHINES
server      LMID=simple
        APPDIR="/usr/tuxedo/simpdb"
        TUXCONFIG="/usr/tuxedo/simpdb/tuxconfig"
        TUXDIR="/usr/tuxedo"
        TLOGDEVICE = "/usr/tuxedo/simpdb/TLOG"
        TLOGNAME=TLOG
        TLOGSIZE=100
*GROUPS
    GROUP1  LMID=simple GRPNO=1
        TMSNAME="TMS_INFORMIX" TMSCOUNT=2
        OPENINFO="INFORMIX-DSHC:mydb"
*SERVERS
    DEFAULT:
        CLOPT="-A"
```

```
        test    SRVGRP=GROUP1 SRVID=1
    *SERVICES
```

6. 用 TMADMIN 创立 TLOG 文件，TUXEDO 用一个文件 TLOG 记录对数据库操作的日志。用于协调分布式数据库的提交与回滚.

```
 D:\>tmadmin
 >crdl -b 500 -z /usr/tuxedo/simpdb/TLOG
 >crlog -m simple
 >q
```

三、效劳端的程序：test.cp

功能:根据客户端传的 EMPNO 到表 EMP 中取 ENAME 的值,并把它返回给客户端

```
#include <atmi.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sybhesql.h>
#include <sybtesql.h>
EXEC SQL INCLUDE sqlca;
EXEC SQL BEGIN DECLARE SECTION;
    long al_empno=0;
    char ac_ename[11]="";
EXEC SQL END DECLARE SECTION;
TEST(TPSVCINFO *rqst)
{
    /*接收客户端来的数据*/
    al_empno = (FBFR32 *)rqst->data;
    EXEC SQL select ename into :ac_ename from EMP where empno=:al_empno;
    if(sqlca.sqlcode!=0)
    {
        userlog("select from EMP failure,sqlca.sqlcode=%ld\n",sqlca.sqlcode);
        tpreturn( TPFAIL, 0, rqst->data, 0, 0 );
    }
    /*把取出的结果返回给客户端*/
    strcpy(rqst->data,ac_ename);
    tpreturn( TPSUCCESS, 0, rqst->data, 0, 0 );
}
```

四、编写客户端程序: testcli.c

功能:调用 TUXEDO 效劳端的效劳 TEST,取 EMPNO=1000 所对应的 ENAME 的值,并显示出来

```
#include <stdio.h>
#include "atmi.h"
main(argc, argv)
{
    long reqlen=1024;
    char *reqbuf;
    /* 与 TUXEDO 效劳端建立连接 */
    if (tpinit((TPINIT *) NULL) == -1)
    {
```