|  |
| --- |
| トライテック |
| PaxosCIFS |
| [文書のサブタイトルを入力] |
|  |
| **Noritaka WATANABE** |
| **[日付を選択]** |

|  |
| --- |
| [文書の要約をここに入力してください。要約は一般に、文書の内容を短くまとめたものです。文書の要約をここに入力してください。要約は一般に、文書の内容を短くまとめたものです。] |

内容

[はじめに 0](#_Toc468192511)

[1. 概要 0](#_Toc468192512)

[**1.1.** **構成** 0](#_Toc468192513)

[**1.2.** **仮応答と仮要求** 0](#_Toc468192514)

[**1.3.** **サーバーからの要求とクライアントからの応答** 1](#_Toc468192515)

[**1.4.** **メタデータと個別データ** 1](#_Toc468192516)

[**1.5.** **再参入時のflush** 2](#_Toc468192517)

[2. CIFS(SMB) 2](#_Toc468192518)

[2.1. VC 2](#_Toc468192519)

[2.2. メタ識別子 2](#_Toc468192520)

[2.2.1. SessionKey 2](#_Toc468192521)

[2.2.2. PID/MID 2](#_Toc468192522)

[2.2.3. FID 3](#_Toc468192523)

[2.2.4. SID 3](#_Toc468192524)

[2.2.5. TID 3](#_Toc468192525)

[2.2.6. UID 3](#_Toc468192526)

[2.3. IDマップ 3](#_Toc468192527)

[2.4. 署名 3](#_Toc468192528)

[2.5. 通信プロトコル 4](#_Toc468192529)

[2.5.1. Batch要求 4](#_Toc468192530)

[2.5.2. Transaction要求 4](#_Toc468192531)

[2.5.3. Notify要求 6](#_Toc468192532)

[2.6. データ構造 7](#_Toc468192533)

[2.6.1. CoC 7](#_Toc468192534)

[2.6.2. CoS 8](#_Toc468192535)

[3. Windows環境の設定 8](#_Toc468192536)

[**3.1.** **Visual　StudioとWindows　Kits** 8](#_Toc468192537)

[3.2. Socket 8](#_Toc468192538)

# はじめに

　C/Sモデルは、要求・応答型であり、クライアント側とサーバー側にモジュールを挿入し、Paxos化を行う。

# 概要

## **構成**

CoC

CoS

CoC

CoS

CoS

CoS

I/F

　要求応答型のC/Sモデルに、クライアント側にCoC(ConverterOfClient)、サーバー側にCoS(ConverterOfServer)を設定する。

　CoCには、プロキシーSVアドレスとSVセル名の対を与える。

CoCは、CLからの接続要求時にCoSに対するセッションを確立する。CoCの接続構造体とCoSのアダプターは１：１対応とする。

　再参入のとき、SVのキャッシュをFlushするかレプリケーションを作成する必要がある。

## **仮応答と仮要求**

　連続的な要求、連続的な応答がCL-SV間で行われる場合がある。この場合でも、CoC-CoS間ではアトミック（速やかに終了する）な要求―応答を実現する必要がある。

このため、通信コンテキストが既知であることを前提として、仮応答、仮要求を導入する。

CL

CoC

CoS

SV

CL

SV

仮応答

仮要求

連続する要求では、CoCに要求が滞留され、CoSが仮応答を返し、CoCが滞留された次の要求を処理する。これにより、CoCとCoSが並列動作する。

　連続する応答では、CoCは通信コンテキストから連続応答を認識できるとし、仮要求を発行する。CoSは滞留している応答を順次に返す。これにより、SVからの連続応答とCoCへの応答を平行動作することができる。

## **サーバーからの要求とクライアントからの応答**

　CIFSのSMB\_COM\_LOCKING\_ANDXはサーバーが要求を出しクライアントが応答を返す場合にも使われる。

　先行するクライアントからの要求を起因として、すべてのサーバーが要求を発行し、クライアントサイドはイベントセッションでリーダから受け取り、応答の別セッションで応答を返す。この応答にはACK応答を返す。。

## **メタデータと個別データ**

　一般に、各サーバー(SV)は個別の識別子等を払い出す。これらを統合するメタデータを払いだし、クライアントに通知する必要がある。

　CoCが外部インターフェースであるので、メタデータ管理を主導する。

例えば、SMBのファイルオープン時のFIDは個別サーバー毎接続毎にに払い出されるのでメタ化を行わなければならない。CoSはリーダーのみがCoCに応答を返すので、CoCは全ての個別データは取得できない。そこで、メタ化が必要なデータはCoCが前もってメタデータを払い出し、CoSで個別データと対応付けする。マルチクライアントでもCoC毎にメタを払い出す。したがって、次図のように複数のCoCのメタと個別の対応関係を管理することになる。なお、　CoSはCoC毎にサーバと接続管理を行う。

ファイル名

メタ

メタ

個別

　また、これらのメタデータはマルチクライアントを考慮すると、本来の要求とあわせてatomicでなければならない。したがって、CoCからは、「メタ情報＋要求」としてCoSにパケット送信する。

　CoCでは「ファイル名とメタの対応」、CoSではセッションアダプター毎に「メタと個別の対応」を管理する。CoSの再参入時には、メタを引き継ぎ、個別を再構築する。

## **再参入時のflush**

　再参入のために、オリジナルプロトコルにflushが装備されている必要がある。

# CIFS(SMB)

## VC

　SMB\_COM\_NEGOTIATE(0x72)で返されたSessionKeyに関連してVCが作られる。

VCは、SMB\_COM\_SESSION\_SETUP\_ANDX(0x73)でVcNumberを指定して作成する。最初のVCのVcNumberは0であり、VC関連情報を初期化する。VCの最大数はSMB\_COM\_NEGOTIATEのMaxNumberVcsで指定される。

　SessionKeyの同じVCが同一connectionとなる。VcNumberは多重転送、マルチクライアントで用いられる。

## メタ識別子

　CIFSでは、connection毎に各種IDが払い出される。

### SessionKey

　サーバーが払い出すのでメタ化する必要がある。

### PID/MID

　PID(ProcessID)、MID(MultiplexID)はクライアントが払いだすのでメタ化する必要はない。PIDとMIDでクライアントの論理threadに対応する。

### FID

　FID(FileID)は、Openあるいはcreateされたファイルのハンドルでconnection毎にサーバーで払い出されるのでメタ化する必要がある。

### SID

　SID(SearchID)は、openされたディレクトリのハンドルでconnection毎にサーバーで払い出されるのでメタ化する必要がある。

### TID

　TID(TreeID)は、shareに対する木接続ハンドルでconnection毎にサーバーで払い出されるのでメタ化する必要がある。

### UID

UID(UserID)は、認証ユーザでconnection毎にサーバーで払い出されるのでメタ化する必要がある。

## IDマップ

メタFIDは１６ビットであり、fd\_setのビットマップを接続毎に管理する。ビットマップは１バイト８ビットであるので、2\*\*13バイト＝2\*\*10x2\*\*3＝８KBが必要である。

## 署名

## 通信プロトコル

### Batch要求

SMB\_Header

SMB\_Parameter

SMB\_Data

AndXCommand

AndXOffset

SMB\_Parameter

SMB\_Data

AndXCommand

AndXOffset

Batch（AndX）要求は、複数のコマンドの連続したコンパウンド要求である。

本コマンドは、サーバーの受信最大サイズを超えない。複数の応答もサイズを超えない。

### Transaction要求

大容量データ(64KB以上)の転送に使う。

最初の要求でパラメータ数とデータ数をサーバーに通知する。

バッファーサイズに収まらないときには、サーバーはOKあるいはエラー応答を行う。

クライアントは、残りを送信する。

サーバーは、処理を行い、複数の応答を返す。

The flow for the transaction protocol when the request parameters and

data do not all fit in a single buffer is:

Client <-> Server

=============================== ==== ==============================

Primary TRANSACTION request ->

<- Interim Server Response

仮応答

Secondary TRANSACTION request 1 ->

仮応答

Secondary TRANSACTION request 2 ->

Secondary TRANSACTION request N ->

<- TRANSACTION response 1

仮要求

<- TRANSACTION response 2

<- TRANSACTION response m

仮要求

The flow for the transaction protocol when the request parameters and

data does all fit in a single buffer is:

Client <-> Server

=============================== ==== ==============================

Primary TRANSACTION request ->

<- TRANSACTION response 1

仮要求

<- TRANSACTION response 2

仮要求

<- TRANSACTION response m

　ワーカー方式の中継では、ワーカーはクライアントの最初のトランザクションがサイズ内に収まらない時にはサーバーの中間応答を待ち、引き続くクライアントからの要求の最後を認識してサーバーの応答を待つ。

　「Secondary　TRANSACTION　request　ｎ」では、仮応答を待ち、終了でなければ次の要求を送信する。

　「TRANSACTION　response　ｎ」では、CoCが仮要求を発行し、応答をCLに返す。終了でなければ、仮要求を再度発行する。

1. 仮応答の開始

SMB\_COM\_TRANSACTION\_SECONDARY

SMB\_COM\_TRANSACTION2\_SECONDARY

SMB\_COM\_NT\_TRANSACT\_SECONDARY

　で仮応答が発生する。

　CoSでは上記受信時に直ちに仮応答を返す。CoCは仮応答であれば、CLからの次の要求を取り出しCoSに送る。

1. 仮応答の終了

SMB\_COM\_TRANSACTION

SMB\_COM\_TRANSACTION2

SMB\_COM\_NT\_TRANSACT

　の最初の要求で、TotalParameterCount、TotalDataCountが通知され、２番目以降のパケットのParameterCount、DataCountをデクリメントし、TotalParameterCountとTotalDataCountがゼロとなったときが最後の要求パケットである。これにより、仮応答の終了を知る。

1. 仮要求の開始と終了

SVは応答にTotalParameterCount、TotalDataCount及びParameterCount、DataCountが設定されている。前者と後者が同じであれば終了である。

### Notify要求

　NT\_TRANSACTのNT\_TRANSACT\_NOTIFY\_CHANGEは、サーバーにディレクトリFIDで変更があったとき通知するように設定を要求し、サーバーは変更あったときにクライアントに通知の応答をする。

　要求と応答は非同期に行われる。一方、CoC－CoS間は、要求・応答の同期型である。

そこで、非同期応答を受け付けるイベントセッション（スレッド）を用意する。

CL

CoC

CoS

SV

CL

SV

仮応答

ｲﾍﾞﾝﾄ

設定

設定

その他

その他

ｲﾍﾞﾝﾄ

ｲﾍﾞﾝﾄ

通常ｾｯｼｮﾝ

ｲﾍﾞﾝﾄｾｯｼｮﾝ

イベントセッションは、CNV\_WAIT\_EVENTで要求を行い永遠に待つ。

　サーバーからの受信ハンドラーは、イベントであれば対応するイベントセッションに、そうでなければノーマル正常セッションにメッセージを送る。

なお、ノーマルセッションとイベントセッションの対応付けはCNV\_EVENT\_BINDによる。

＊PFSServerのeventにメモリ未返却あり？

## データ構造

### CoC

#### 接続構造体

##### 接続の識別子

　CLの接続毎に構造体を払い出し、CoSとのセッションを確立する。セッションのユーザ番号には適当なIDマップを使用する。

##### 制御

CLからの要求は受信ハンドラーがメッセージを接続構造体の要求キューに繋ぐ。接続構造体がワーカーに引き渡されていなければワーカーの待ちキューに繋ぐ。ワーカーはCoS(SV)とは同期型で通信する。ただし、連続要求の場合には仮応答を受け取り、ワーカーを終了することなく次の要求を処理する。また、連続応答を認識したとき（期待するサイズより応答が小さい）は、ワーカーを終了することなく仮要求を発行する。

##### PIDMID\_t（トランザクション用）

　CIFS仕様では、PIDMIDListのエントリ毎にTransactionState｛TransmittedPrimaryRequest、ReceivedInterimResponse、TransmittedAllRequests｝とある。そこで、本構造体をトランザクション管理用とし、

Typedef　enum　TransactionState　{

TransmittedPrimaryRequest、

ReceivedInterimResponse、

TransmittedAllRequests

｝　TransactionState\_t;

Typedef　struct　PIDMID {

List\_t pm\_Lnk;

Uint32\_t pm\_PID;

Uint16\_t pm\_MID;

TransactionState\_t pm\_TransactionState;

Uint16\_t pm\_TotalParameterCountReq;

Uint16\_t pm\_TotalDataCountReq;

Uint16\_t pm\_MaxParameterCountReq;

Uint16\_t pm\_MaxDataCountReq;

Uint16\_t pm\_TotalParameterCountRes;

Uint16\_t pm\_TotalDataCountRes;

}　PIDMID\_t;

### CoS

# Windows環境の設定

## **Visual　StudioとWindows　Kits**

　ビルド環境はVisual　Studio、socket等の実行環境はWindows　Kitsと分割されている。

さらに、バージョン、32ビットと64ビット、nativeとcrossがあり、組み合わせが複雑となっている。

　例えば、以下の環境設定を行う。

$ cat VC\_ENV.bat

set VS14=C:\Program Files (x86)\Microsoft Visual Studio 14.0\VC

set VS12=C:\Program Files (x86)\Microsoft Visual Studio 12.0\VC

set SDK=C:\Program Files (x86)\Windows Kits\10\

set SDK\_I=%SDK%\Include\10.0.10240.0

set SDK\_L=%SDK%\Lib\10.0.10240.0

set INCLUDE=%VS12%\Include;%VS14%\Include;%SDK\_I%\ucrt;%SDK\_I%\um;%SDK\_I%\shared

set LIBPATH=%VS14%\Lib\amd64;%SDK\_L%\ucrt\x64;%SDK\_L%\um\x64

set LIB=%LIBPATH%

set

　コンパイルは、例えば、

cl　z.c　ws2\_32.lib（あるいは、wsock32.lib）

とする。ライブラリは％LIB%から探し出されるので、名前は32ビットでも64ビットでも同じである。

## Socket

　Windowsのsocketの使用は、次のように初期化してから使用する。

//#include <stdafx.h>

#include <stdio.h>

#include <winsock2.h>

#include <windows.h>

int main()

{

intptr\_t fd;

WORD Ver;

WSADATA wsaData;

Ver = MAKEWORD(2,2);

(void)WSAStartup( Ver, &wsaData );

fd = socket( AF\_INET, SOCK\_STREAM, 0);

printf("This is a Native C program.fd=%d\n", (int)fd);

return 0;

}