

Tmax ゲートウェイガイド (TCP/IP)

Tmax v6.0



Copyright © 2016 TmaxSoft Co., Ltd. All Rights Reserved.

Copyright Notice

Copyright © 2016 TmaxSoft Co., Ltd. All Rights Reserved.

45, Jeongjail-ro, Bundang-gu, Seongnam-si, Gyeonggi-do, 13613, South Korea

Restricted Rights Legend

All TmaxSoft Software (Tmax®) and documents are protected by copyright laws and international convention. TmaxSoft software and documents are made available under the terms of the TmaxSoft License Agreement and this document may only be distributed or copied in accordance with the terms of this agreement. No part of this document may be transmitted, copied, deployed, or reproduced in any form or by any means, electronic, mechanical, or optical, without the prior written consent of TmaxSoft Co., Ltd.

Nothing in this software document and agreement constitutes a transfer of intellectual property rights regardless of whether or not such rights are registered) or any rights to TmaxSoft trademarks, logos, or any other brand features. This document is for information purposes only. The company assumes no direct or indirect responsibilities for the contents of this document, and does not guarantee that the information contained in this document satisfies certain legal or commercial conditions. The information contained in this document is subject to change without prior notice due to product upgrades or updates. The company assumes no liability for any errors in this document.

このソフトウェア(Tmax®)マニュアルの内容とプログラムは、日本国の著作権法および国際条約によって保護されています。マニュアルの内容とプログラムは、TmaxSoft Co., Ltd.との使用許諾契約書の下でのみ使用することができ、マニュアルは使用許諾契約で許可されている範囲を除いては、配布または複製することができません。TmaxSoftの書面による事前の承諾を得ることなく、このマニュアルの全部または一部を電子的または機械的な方法を問わず、転送、複製、配布したり、または二次的著作物を作成する等の行為を一切禁じます。

このソフトウェアのマニュアルとプログラムの使用許諾契約は、いかなる場合においても、マニュアル及びプログラムと関連する知的財産権(登録の有無を問わず)を譲渡するものと解釈されず、TmaxSoftのブランド、ロゴ、商標等の使用権限を与えるものではありません。マニュアルは、情報を提供する目的でのみ提供しており、これに伴う契約上の直接的ないしは間接的な責任を負わず、マニュアルの内容は法律上もしくは商業的な特定の条件が満たされることを保証しません。マニュアルの内容は、製品のアップグレード及び修正により、その内容が予告なく変更されることがあり、内容上の誤りがないことを保証しません。

Trademarks

Tmax®, Tmax WebtoB® and JEUS® are registered trademark of TmaxSoft Co., Ltd. Other products, titles or services may be registered trademarks of their respective companies.

Tmax®, Tmax WebtoB®, JEUS® は、TmaxSoft Co., Ltd.の登録商標です。その他、記載されている会社名、製品名などは、各社の商標または登録商標です。

Open Source Software Notice

Some modules or files of this product are subject to the terms of the following licenses. : openssl-0.9.7.m, zlib-1.1.4, expat-2.0.0, net-snmp, DCE1.0, pthread, google-diff-match-patch, libevent, getopt

Detailed Information related to the license can be found in the following directory :
\${INSTALL_PATH}/license/oss_licenses

この製品の一部ファイルまたはモジュールは、openssl-0.9.7.m、zlib-1.1.4、expat-2.0.0、net-snmp、DCE1.0、pthread、google-diff-match-patch、libevent、getoptライセンスを遵守します。

詳細情報については、製品ディレクトリーの\${INSTALL_PATH}/license/oss_licensesに記載されている事項を参照してください。

文書情報

文書名: Tmax ゲートウェイガイド (TCP/IP)

発行日: 2016年8月5日

ソフトウェアバージョン: Tmax v6.0

ガイドバージョン: v2.1.1

目次

このガイドについて	ix
第1章 紹介	1
1.1. 概要	1
1.2. サービスのタイプ	2
1.2.1. 同期型TCPGW	3
1.2.2. 非同期型TCPGW	6
1.2.3. サーバーTCPGW	8
1.2.4. クライアントTCPGW	9
1.3. ゲートウェイのその他機能	9
1.3.1. ゲートウェイ・ヘッダー	9
1.3.2. 応答の多重処理	10
1.3.3. サービス名の検索順	10
1.3.4. ユーザーが任意のチャンネルを指定	11
1.3.5. コード変換	11
第2章 環境設定	15
2.1. 概要	15
2.2. Tmax環境構成	15
2.2.1. サーバー/クライアントTCPGW	26
2.2.2. サービス・ブロック型TCPGW	27
2.2.3. サービス非ブロック型TCPGW	28
2.2.4. リモート同期型と非同期型TCPGW	29
2.2.5. 再接続TCPGW	29
2.2.6. 複数のリモート・ノードと接続するクライアントTCPGW	30
2.3. ユーザー・ヘッダー環境設定および使用方法	31
2.4. チャンネル・バックアップの設定	33
2.5. Pingメッセージ・チェック時のデータ転送	33
第3章 ユーザー・プログラムと関数	35
3.1. 概要	35
3.2. custom.h	36
3.3. custom.c	37
3.3.1. init_remote_info	39
3.3.2. remote_connected	40
3.3.3. remote_connected_ipv6	40
3.3.4. remote_closed	42
3.3.5. allow_connection	42
3.3.6. allow_connection_ipv6	43
3.3.7. get_msg_length	45
3.3.8. get_msg_info	46
3.3.9. get_channel_num	46

3.3.10.	put_msg_info	47
3.3.11.	put_msg_complete	48
3.3.12.	get_service_name	48
3.3.13.	prepare_shutdown	48
3.3.14.	set_service_timeout	49
3.3.15.	chk_end_msg	49
3.3.16.	inmsg_recovery	50
3.3.17.	outmsg_recovery	50
3.3.18.	get_extmsg_info	51
3.3.19.	put_extmsg_info	52
3.3.20.	set_ping_msg	53
3.3.21.	chk_pong_msg	54
3.3.22.	set_extping_msg	54
3.3.23.	chk_extpong_msg	56
3.3.24.	reset_ping_msg	57
3.3.25.	reset_extping_msg	58
3.3.26.	set_error_msg	59
3.3.27.	get_msg_security	60
3.3.28.	put_msg_security	61
3.4.	register.c	62
第4章 例	67
4.1.	アウトバウンドTCPGW	67
4.1.1.	環境ファイル	67
4.1.2.	TCPGW	68
4.1.3.	リモート・ノード	72
4.1.4.	サーバー	74
4.2.	同期インバウンドTCPGW	75
4.2.1.	環境ファイル	76
4.2.2.	TCPGW	76
4.2.3.	リモート・ノード	81
4.2.4.	クライアント	86
4.3.	非ブロックTCPGW	87
4.3.1.	環境ファイル	88
4.3.2.	TCPGW	89
4.3.3.	リモート・ノード	92
4.3.4.	サーバー	94
4.3.5.	クライアント	95
付録 A. TCGWのエラーコード	97
索引	99

図目次

[図 1.1]	TCPGWの動作構造	1
[図 1.2]	TCPGWの動作構造(同期/非同期)	2
[図 1.3]	同期型TCPGWの動作構造 - サービス・ブロック型方式	4
[図 1.4]	同期型TCPGWの動作構造 - サービス非ブロック型方式	5
[図 1.5]	同期型TCPGWの動作構造 - リモート方式	6
[図 1.6]	非同期型TCPGWの動作構造 - Tmaxからのサービス要求方式	7
[図 1.7]	非同期型TCPGWの動作構造 - リモートからのサービス要求方式	8
[図 1.8]	サーバーTCPGWの動作構造	8
[図 1.9]	クライアントTCPGWの動作構造	9
[図 4.1]	アウトバウンドTCPGWプログラムの動作	67
[図 4.2]	同期インバウンドTCPGWプログラムの動作	75
[図 4.3]	非ブロックTCPGW方式	87

このガイドについて

対象読者

本書は、Tmax[®](以下、Tmax) TCP/IPゲートウェイ(以下、TCPGW)を使用して開発するユーザーを対象としています。同書では、Tmaxサーバーと非Tmaxサーバーのインターフェースを担当する、Tmaxが提供するTCP/IP ゲートウェイについて記述しています。

前提知識

本書は、Tmaxシステムに関する全般的な内容と、Tmaxシステムが提供している各種機能および特性を習得するためのガイドです。

本書を理解するには、事前に下記の内容を把握しておく必要があります。

- ミドルウェアおよびUNIXシステムに関する知識
- Tmaxの基本概念に関する知識
- Java、Cプログラミングに関する知識

制限事項

本書を読む前に、Tmaxの基本概念について熟知している必要があります。実務における詳細な使用方法および管理、運用に関する内容については、各製品のガイドを参照してください。

参考

Tmaxシステム開発に関する基本的な内容は、『Tmax 運用ガイド』、または『Tmax アプリケーション開発ガイド』を参照してください。Tmaxが提供しているコマンドとC APIに関する説明については、『Tmax リファレンスガイド』を参照してください。

本書の構成

Tmax ゲートウェイガイド(TCP/IP)は、計4章と付録で構成されています。

各章の主な内容は下記のとおりです。

- 第1章: 紹介

TCP/IPゲートウェイ・システムの概要および動作方式、サービスのタイプについて記述します。

- 第2章: 環境設定

TCP/IPゲートウェイの環境設定方法について記述します。

- 第3章: ユーザープログラムと関数

TCP/IPゲートウェイを使用するため、custom.h、custom.c、regidster.cを修正する方法について記述します。

- 第4章: 例

TCP/IPゲートウェイ・サービスタイプに関する例について記述します。

- 付録A: TCPGWのエラーコード

エラーコードについて記述します。

表記上の規則

表記	意味
<AaBbCc123>	プログラム・ソースコードのファイル名、ディレクトリー
<Ctrl>+C	CtrlキーとCキーを同時に押す
[Button]	GUIのボタン、メニュー名
太字	強調
「」、『』（鍵カッコ）	関連文書、あるいはガイド内の他の章および節の表示
「入力項目」	画面UI上の入力項目
<ハイパーリンク>	メール・アカウント、Webサイト
>	メニューの実行順
+----	下位ディレクトリー/ファイル有り
----	下位ディレクトリー/ファイル無し
参考	参照/注意事項
[図 1.1]	図の名称
[表 1.1]	表の名称
AaBbCc123	コマンド、コマンド実行結果の画面出力、サンプル・コード
[]	オプション・パラメータ値
	選択・パラメータ値

システム要件

	要求事項
プラットフォーム	IBM AIX 5.x / 6.1 / 7.1
	HP-UX 11.xx
	SunOS 5.7~5.9 / SunOS 5.10 / SunOS 5.11
ハードウェア	1GB以上のハードディスク空き容量
	512MB以上のメモリー空き容量
データベース	Oracle 9~12
	Tibero 4~5
	DB2
	Informix

関連ガイド

ガイド	説明
Tmax 運用ガイド	Tmaxを利用するための環境設定ファイルとシステム運用方法について説明しています
Tmax アプリケーション開発ガイド	Tmaxアプリケーション・プログラムの開発で使用するAPIの概念と使用方法および例について説明しています
Tmax リファレンスガイド	Tmaxアプリケーションの開発に使用するコマンドおよびクライアントとサーバーの接続、通信に使用する関数の使用方法と例について説明しています

お問合せ先

Korea

TmaxSoft Co., Ltd.
45, Jeongjail-ro, Bundang-gu,
Seongnam-si, Gyeonggi-do, 13613
South Korea
Tel: +82-31-8018-1000
Fax: +82-31-8018-1115
Email: info@tmax.co.kr
Web (Korean): <http://www.tmaxsoft.com>
TechNet: <http://technet.tmaxsoft.com>

USA

TmaxSoft Inc.
101 North Wacker Drive, Suite 2014,
Chicago, IL 60606
U.S.A
Tel: +1-312-525-8330
Email: info@tmaxsoft.com
Web (English): http://www.tmaxsoft.com/us_en/home

Japan

TmaxSoft Japan Co., Ltd.
5F Sanko Bldg, 3-12-16 Mita, Minato-Ku, Tokyo, 108-0073
Japan
Tel: +81-3-5765-2550
Fax: +81-3-5765-2567
Email: info@tmaxsoft.co.jp
Web (Japanese): <http://www.tmaxsoft.co.jp>

China

Beijing TmaxSoft System Software Co., Ltd.
Room103, No.2 Huizhong Building, Seven Street Shangdi,
Haidian District, Beijing, 100085
P.R.China
Tel: +86-10-6298-8827
Email: info@tmaxsoft.com.cn
Web (Chinese): http://www.tmaxsoft.com/cn_en/home_cn_en

Brazil

Tmax Brasil Sistemas e Serviços Ltda.
Av. Copacabana, 177, sala 32~35 Empresarial 18 do Fortel
Alphaville Barueri, Sao Paulo, 06472-001
Brazil
Tel: +55-11-4191-3100
Fax: +55(11) 4191-3705 (extension#112)
Email: info.bra@tmaxsoft.com
Web (Portuguese): http://www.tmaxsoft.com/br_en/home_br_en

Russia

Tmax Rus L.L.C.
Leninsky prospekt, 113/1 (Park Place Moscow),
Office 318e, Moscow, 117198
Russia
Tel: +7(495)970-01-35
Email: info.rus@tmaxsoft.com
Web (Russian): http://www.tmaxsoft.com/ru_ru/home_ru_ru

Singapore

Tmax Singapore Pte. Ltd.
430 Lorong 6, Toa Payoh #10-02,
OrangeTee Building, 319402
Singapore
Tel: +65-6259-7223
Fax: +65-6258-7112
Email: info.sg@tmaxsoft.com
Web (English): http://www.tmaxsoft.com/sg_en/home_sg_en

United Kingdom

TmaxSoft UK Ltd.
215 Knyvett House, Watermans Business Park,
The Causeway, Staines TW18 3BAB
United Kingdom
Tel: +44-1784-895005
Email: info.uk@tmaxsoft.com
Web (English): http://www.tmaxsoft.com/gb_en/home_gb_en

Canada

TmaxSoft Canada, Inc.
2425 Matheson Blvd East, 8th floor,
Unit 824 Mississauga, ON, L4W 5K4
Canada
Tel: +1-905-361-2888
Email: info.canada@tmaxsoft.com
Web (English): http://www.tmaxsoft.com/ca_en/home_ca_en

Australia

TmaxSoft Proprietary Limited
L32, 101 Miller Street, North Sydney 2060
Australia
Tel: +91-9845-330-704
Email: info.aus@tmaxsoft.com
Web (English): http://www.tmaxsoft.com/au_en/home_au_en

India

TmaxSoft Technologies Private Limited
Sobha Alexander Plaza, 3rd Floor,
16/2 Commissariat Road, Bangalore-560025
India
Tel: +91-9845-330-704
Email: info.india@tmaxsoft.com
Web (English): http://www.tmaxsoft.com/in_en/home_in_en

Turkey

TmaxSoft Co., Ltd. Turkey Liaison Office
Windowist Tower. Eski Buyukdere Cad. No:26,
Maslak 34467 Istanbul
Turkey
Tel: +90-544-553-6045
Email: cslee@tmaxsoft.com
Web (English): http://www.tmaxsoft.com/tr_en/home_tr_en

第1章 紹介

本章では、TCP/IPゲートウェイ・システムの概要および動作方式、サービスのタイプについて説明します。

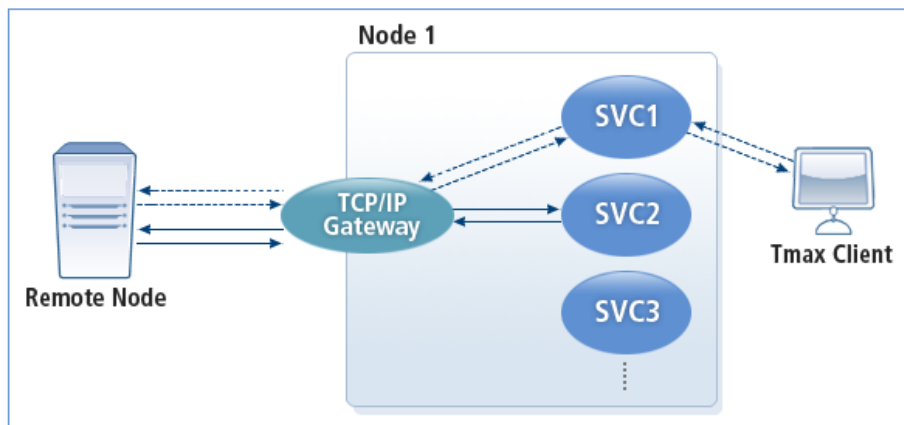
1.1. 概要

TCP/IPゲートウェイ(TCP/IP Gateway、以下TCPGW)はTmaxが提供するゲートウェイのうち、Tmaxサーバーと非Tmaxサーバー(以下、リモート・ノード)間のインターフェースを担当します。TCPGWはTmaxサーバーの一種として、TCP/IPによって接続されているUNIX/Windowsサーバーなどのゲートウェイの役割をします。

TCPGWは、リモート・ノードから送信されたメッセージを該当のサービスにtpacallし、サービスの結果は最初に要求したリモート・ノード・サーバーに転送します。一方、Tmaxサービスでtpcallまたはその他の方式でサービスを要求すると、TCPGWはリモート・ノードに要求を転送した後、応答が届いたら自身を呼び出したサービスにtpreturnします。ソケットを開いてメッセージを送受信など、他システムをTCP/IPで接続するために必要な複雑な作業はすべてTCPGWで処理されるため、開発者は業務ロジックのみ作成します。

下記は、TCPGWの動作構造です。

[図 1.1] TCGWの動作構造



TCPGWの基本動作方式は次の2つに分けられます。

- Tmaxでサービス要求

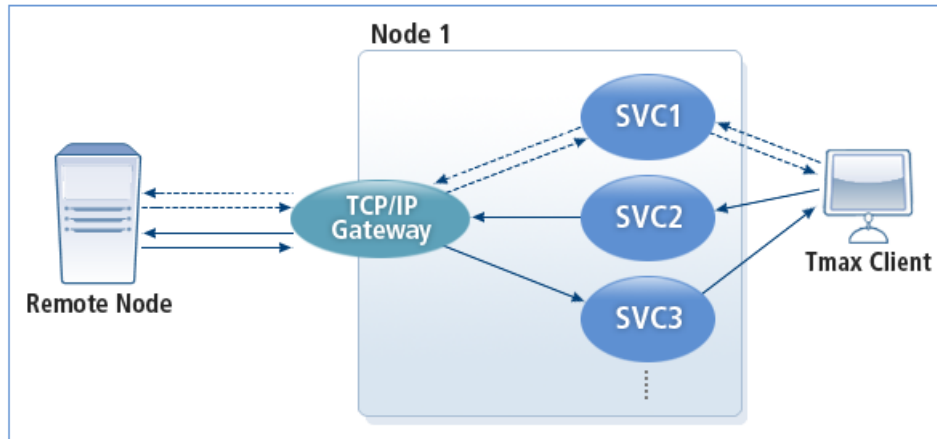
[図 1.1]で点線が表示された部分です。Tmaxクライアントはサービスからサービス要求を受け、リモート・ノードにサービスを要求することができます。このようなサービスをインバウンド・サービスといいます。

- リモート・ノードからのサービス要求

[図 1.1]で実線で表示された部分です。TCPGWはリモート・ノードからサービス要求を受け、処理することができます。このようなサービスをアウトバウンド・サービスといいます。

さらに、TCPGWは同期呼び出し方式と非同期呼び出し方式で動作します。

[図 1.2] TCPGWの動作構造(同期/非同期)



- 同期呼び出し方式

[図 1.1]で点線で表示された部分です。TmaxクライアントまたはサービスでTCPGWを直接呼び出し、応答が届くまで待機する方式です。

- 非同期呼び出し方式

[図 1.1]で実線で表示された部分です。TmaxクライアントがTmaxサービスを呼び出すと、そのサービスからTCPGWに制御権が移るため、当該サービスは他のサービス要求を受けることができます。TCPGWがリモート・ノードから応答を受け、応答を処理するサービスに要求を渡す方式です。

参考

TmaxにインストールされるTCPGWはTmaxサーバーの1つであり、これを使用するにはTmax環境ファイルにサーバーとして登録する必要があります。一般サーバーはTCSまたはUCS用サーバー・ライブラリーを用いて作成しますが、TCPGWは外部との通信を担当するライブラリー(libtcpgw.a、libtcpgw.so)とユーザーが作成したプログラム(custom.c)をリンクして作成します。Tmax環境設定の詳細内容については、「第2章 環境設定」と「第3章 ユーザー・プログラムと関数」を参照してください。

1.2. サービスのタイプ

TCPGWは、Tmaxが提供するライブラリー(libtcpgw.a、libtcpgw.so)と開発者が作成するcustom.c、custom.hをリンクして作成されます。このように作成されたTCPGWはリモート・ノードと通信し、Tmaxクライアントの要

求をリモート・ノードに渡したり、リモート・ノードの要求をTmaxサービスで処理できるような役割をしたりします。

TCPGWはその動作方式によって**同期型TCPGW**と**非同期型TCPGW**に分けられ、起動方法によって**サーバーTCPGW**と**クライアントTCPGW**に分けられます。サーバーとクライアント・モードは、リモート・ノードとTCPGWの接続時に、どちらから先に接続を試みるかによって決まります。一旦相手と接続されたら、どちらからでも先にサービスを要求することができるため、サーバーとクライアント・モードは特に意味がありません。

1.2.1. 同期型TCPGW

同期型方式は、Tmaxのクライアントまたはサーバーでサービスを要求し、そのサービスを要求したクライアントまたはサーバーに応答が受信される方式です。一方、リモート・ノードでサービスを要求すると、TCPGWはTmaxのサービスを要求し、サービスを要求したリモート・ノードにその結果を返す方式です。

前者の場合、リモート・ノードにサービスを要求したTmaxサービスがブロックされるかどうかによってTCPGWの動作は異なります。後者の場合は、リモート・ノードが要求した処理結果を返す際、それを要求したチャンネルに返す方式です。

参考

Tmaxのクライアントまたはサーバーが同期型方式でリモート・ノードにサービスを要求する場合は、TCPGWとリモート・ノード間に相互UID(Unique Id.)を共有する必要があります。

サービス・ブロック型方式

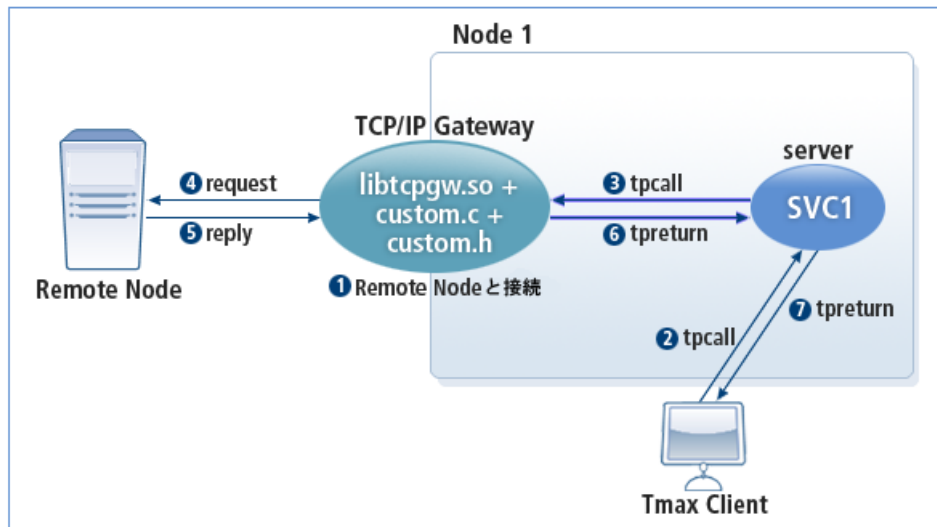
TmaxのサーバーまたはクライアントがTCPGWにサービスを要求し、その結果が受信されるまで待機する最も一般的な方式です。Tmaxクライアントの要求を受けたサービスがTCPGWサービスにtpcallすると、リモート・ノードが処理結果を送信するまでTmaxサービス(図のSVC1)がブロックされます。

TCPGWを動作させると、Tmaxクライアントが呼び出したTmaxサービスはTCPGWを呼び出した後、結果を受けるまでブロックされます。tpacallした場合もtpgetrplyで応答を受けるところでブロックされるため、tpcallと同様にブロックされます。多くの要求を受けるには、ブロックされる時間まで考慮し、多数のサーバーを実行する必要があります。

外部通信では、様々な障害(マシン、ネットワーク)に備えて多数のサーバーを必要となるため、サービス・ブロック型方式が適しています。

下記は、ブロック型同期方式でサービスを呼び出す場合のTCPGWの動作構造です。

[図 1.3] 同期型TCPGWの動作構造 - サービス・ブロック型方式



下記は、フェーズ別の処理内容です。

1. TCGWとリモート・ノードが接続されている状態です。
2. TmaxクライアントはTmaxサービスをtpcallします。
3. Tmaxサービスではクライアントの要求を受け、TCPGWにサービスをtpcallします。
4. TCGWは接続されているリモート・ノードにサービスを要求します。
5. リモート・ノードから結果が受信されたら、エラー有無を判断します。
6. TCGWサービスを呼び出したTmaxサービスにtpreturnします。
7. 結果を受けたTmaxサービスは、Tmaxクライアントにtpreturnします。

サービス非ブロック型方式

サービス非ブロック型方式は、Tmaxクライアントで直接TCPGWを要求する方式では使用できません。TCPGWにサービスを要求し、結果を受信するサーバーをその間に設けて処理する方式です。すなわち、TCPGWの前に送信サービスと受信サービスをおき、Tmaxクライアントは送信サービスを呼び出し、送信サービスはTCPGWにサービスを渡してサービスを終了します。

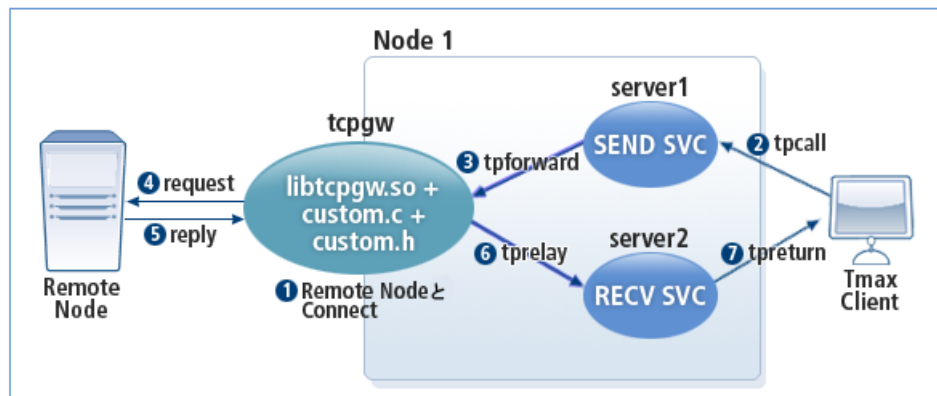
TCPGWがリモート・ノードにサービスを要求し、その結果が受信されたら該当の結果を受信サービスに渡します。受信サービスがTCPGWから結果を取得してクライアントに渡したら、サービスのサイクルが終了される方式です。結果的に、クライアントはサービスを要求して結果を取得する同期方式ですが、サーバーは要求を渡して終了する非同期方式のように動作します。

サービス非ブロック型方式は、ブロック型方式で実行する方式より少ないサーバーで多くの処理ができます。ブロック型方式は、サービスがTCPGWを呼び出してブロックされるため、同時に多くの処理を行うには多数のサーバープロセスを実行する必要があります。しかし、非ブロック型方式は自身のジョブのみ処理してサービスを終了するため、少ないサーバープロセスでも多くのジョブが処理できます。

外部通信を処理する場合は、非ブロック型方式が効率的です。

下記は、サービス非ブロック型方式でサービスを呼び出す場合のTCPGWの動作構造です。

【図 1.4】 同期型TCPGWの動作構造 - サービス非ブロック型方式



1. TCGWとリモート・ノードが接続されている状態です。
2. TmaxクライアントはTmaxサービスをtpcallします。
3. Tmaxサービスではクライアントの要求を受け、TCPGWサービスをtpforwardします。
4. TCGWは接続されているリモート・ノードにメッセージを転送します。
5. リモート・ノードから結果が受信されたら、エラー有無を判断します。
6. リモート・ノードの結果をサービスにtprelayします。
7. TCGWサービスから結果を取得したサービス(Tmaxクライアントがtprelayするサービスとして指定したサービス)は、Tmaxクライアントにtpreturnします。

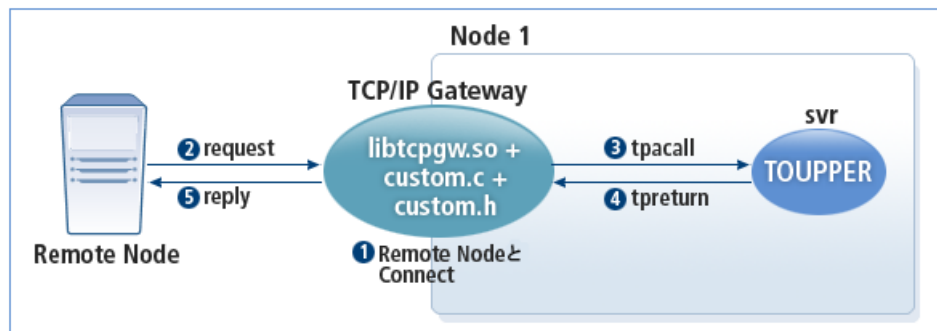
リモート同期型呼び出し方式

リモート・ノードがTCPGWに先にサービスを要求する方式です。TCPGWは、リモート・ノードが要求したサービスを読み出し、その結果を取得して該当のサービスを要求したチャンネルに転送します。リモート・ノードは、TCPGWに対してTmax環境ファイルに定義されているMAXSACALL数を超過して同時に呼び出すことができません。

リモート同期型呼び出し方式は、リモート・ノードでTmaxのサービス呼び出す最も一般的な方式です。TCPGWはリモート・ノードのチャンネル情報を保存しており、サービスから結果が受信されたら保存していたチャンネルの中から該当のチャンネルを見つけ出し結果を転送します。そのとき、当該チャンネルに結果を返す前に他の要求を受けることもできます。TCPGWはリモート・ノードが要求したチャンネルはブロックせず、次の要求が受けられるように処理するため、TCPGWの運用方式によって処理方式が変わります。

下記は、リモート同期型方式でサービス呼び出す場合のTCPGWの動作方式です。

[図 1.5] 同期型TCPGWの動作構造 - リモート方式



1. TCGWとリモート・ノードが接続されている状態です。
2. リモート・ノードは、TCGWと接続されているチャンネルにメッセージを転送します。
3. TCGWはtpacall()を利用してTmaxサービス呼び出します。
4. TCGWはサービスの処理結果を受け、メッセージを要求したチャンネルを検索します。
5. 該当のチャンネルが正常に接続されていたら結果を転送します。

1.2.2. 非同期型TCPGW

非同期型呼び出し方式は、Tmaxのクライアント、サーバー、リモート・ノードがTCGWサービスを要求するだけで、その結果を受けなかったり、あるいはサービスを要求したプログラムではなく他のプログラムで処理したりする方式です。TmaxのサービスでTCGWにサービスを要求し、その結果は他のサービスで受けることができます。

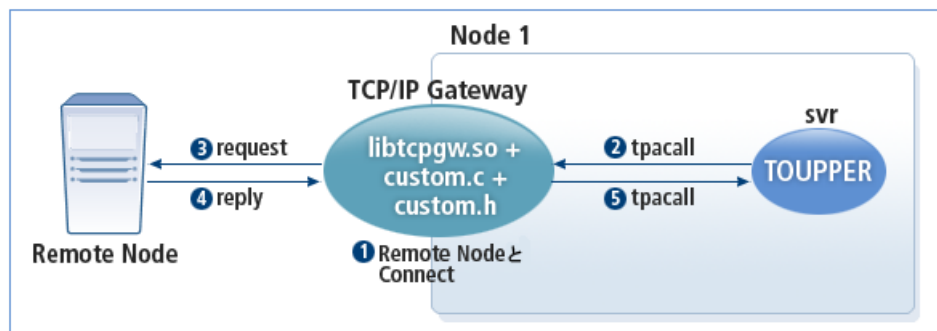
クライアントの場合、非同期型で処理するときは、tpacallで応答を受けない場合のみ可能です。一方、リモート・ノードでサービスを先に要求する場合は、上述のようにサービスを要求するだけで結果は受けないか、あるいは他のチャンネルに転送するのが非同期型方式です。

Tmaxからのサービス要求方式

Tmaxのサービスで、TCPGWにtpacallのTPNOREPLYで要求し、当該サービスは終了します。TCPGWはリモート・ノードに要求を転送し、結果が受信されたらTmaxの他のサービスをtpacallのTPNOREPLYで呼び出し、結果を処理する方式です。

下記は、Tmaxで非同期型方式でサービスを要求する場合のTCPGWの動作構造です。

[図 1.6] 非同期型TCPGWの動作構造 - Tmaxからのサービス要求方式



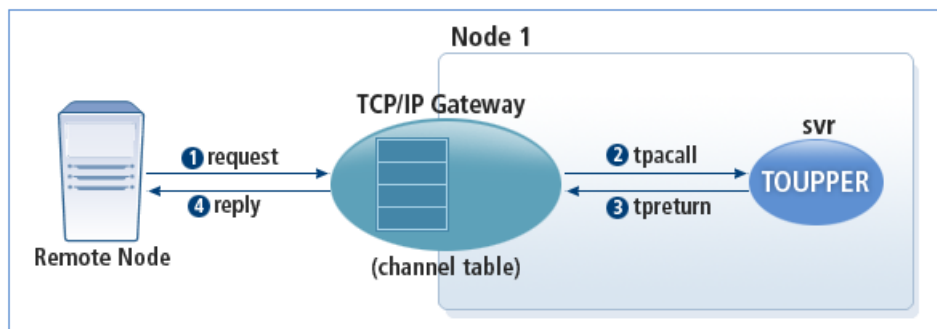
1. TCPGWとリモート・ノードが接続されている状態です。
2. TmaxサービスでTCPGWにサービスを要求します。(tpacallのTPNOREPLY)
3. TCPGWはリモート・ノードにデータを転送します。
4. TCPGWはリモート・ノードからデータを受信します。
5. TCPGWはTmaxの他のサービスを呼び出します。(tpacallのTPNOREPLY)

リモートからのサービス要求方式

リモート・ノードでTCPGWにサービスを要求すると、TCPGWはtpacallでTmaxサービスを要求します。Tmaxサービス処理が完了したら、TCPGWはリモート・ノードに接続されて使用可能なチャンネルのうちいずれかに処理結果を転送します。

下記は、リモート・ノードが非同期型方式でサービスを要求する場合のTCPGWの動作構造です。

[図 1.7] 非同期型TCPGWの動作構造 - リモートからのサービス要求方式



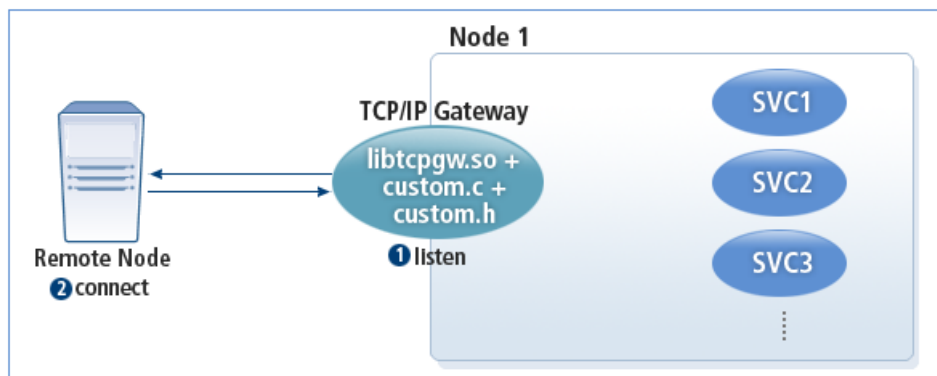
1. リモート・ノードでTCPGWにサービスを要求します。
2. TCGWはTmaxのサービスをtpacallして要求します。
3. Tmaxのサービス処理結果をTCPGWに転送します。
4. TCGWは、リモート・ノードと接続されているチャンネルの中から、チャンネル表でチャンネルを検索して結果を転送します。

1.2.3. サーバーTCPGW

サーバーTCPGWとクライアントTCPGWは、接続を確立する際は違いがありますが、接続された後は機能上の相違点はありません。サーバーTCPGWは、TCPGWで待機していればリモート・ノードから接続することになります。サーバーTCPGWは、クライアントTCPGWと同様、リモート・ノードとTmaxクライアントのどちらからでもサービスを要求することができます。

下記は、サーバーTCPGWの動作構造です。

[図 1.8] サーバーTCPGWの動作構造



1. TCGWは、Tmaxの起動時にリモート・ノードからの接続を待機します。

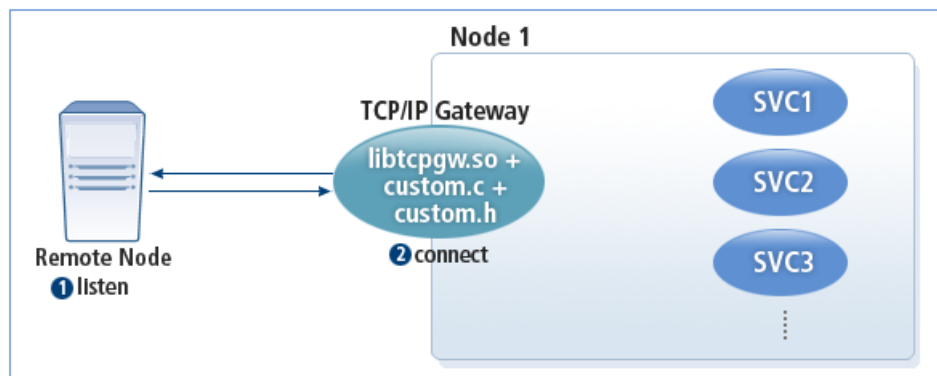
2. リモート・ノードからTCPGWに接続します。

1.2.4. クライアントTCPGW

クライアントTCPGWは、リモート・ノードで待機しているポートに接続すると同時に起動されます。クライアントTCPGWは、リモート・ノードとTmaxクライアントのどちらからでもサービスを要求することができます。

下記は、クライアントTCPGWの動作構造です。

[図 1.9] クライアントTCPGWの動作構造



1. リモート・ノードはTCPGWからの接続を待機します。
2. TCGWはTmaxの起動時にリモート・ノードと接続します。

1.3. ゲートウェイのその他機能

本節では、ゲートウェイのその他機能について説明します。

1.3.1. ゲートウェイ・ヘッダー

TCPGWは、Tmaxクライアントまたはサーバーでサービスを呼び出す際にゲートウェイ・ヘッダーを使用することができます。ゲートウェイ・ヘッダーを使用するには、TCPGWの別のライブラリーを使用する必要があります。通常はlibtcpgw.a、libtcpgw.soを使用しますが、ゲートウェイ・ヘッダーを使用する場合は、libtcpgw.gwh.aまたはlibtcpgw.gwh.soを使用します。ゲートウェイ・ヘッダーは、すべてのデータ・バッファの最初のオフセットに位置する必要があります。すなわち、ユーザー・ヘッダーを使用する場合は、ゲートウェイ・ヘッダーの次にユーザー・ヘッダーが位置することになります。

ゲートウェイ・ヘッダーは様々な目的で使用されます。TCPGWでは、様々な項目の中からsvcのみ使用可能です。この項目は、非ブロック・モードで使用する場合または非同期方式で使用する時、応答データに対し

て処理するサービス名を指定する場合に使用します。ゲートウェイ・ヘッダーは、通常のユーザー・ヘッダーを使用しない場合、応答を処理するサービスをメッセージ別に処理するときに使用します。

TCPGWが応答を処理するサービスを検索する順序は、以下のとおりです。

1. ゲートウェイ・ヘッダー
2. ユーザー・ヘッダー
3. [-S]オプション

1.3.2. 応答の多重処理

一般的なTCPGWの処理方式は、1つの要求に対して1つの応答を受ける形式ですが、場合によっては複数の応答が受信されることもあります。そのときTCPGWは、custom.cのget_msg_info関数の呼び出しを利用して対応します。数回にかけて応答が受信される場合、そのたびget_msg_info関数が呼び出されますが、ユーザーは受信されたデータが最後のデータなのかを判断し、適切な値を返します。

連続したデータの場合、戻り値として3を返すとTCPGWは次のデータが存在すると判断し、サービスを要求したクライアントまたはサービスに応答を渡さずに次のデータを待ちます。ユーザーは最後のデータが受信されたら、REPLY値(1)を返します。そうするとTCPGWは、以前保存していたデータと最後のデータを、サービスを要求したクライアントまたはサービスに返します。

1.3.3. サービス名の検索順

非ブロック型のTCPGWまたは非同同期型のTCPGWの場合、リモート・ノードからの要求や応答を処理するTmaxのサービスが必要となります。TCPGWはこのサービス名が分からないため、ユーザーが適当なサービス名を指定する必要があります。TCPGWは3つの方式でサービスを検索します。

下記で説明するサービス名は、Tmax設定ファイルに登録されている必要があります。

- TmaxクライアントまたはサービスでTCPGWを要求するとき、ゲートウェイ・ヘッダーにサービス名を入力してTCPGWを呼び出すと、TCPGWは優先的にこのサービス名を使用します。ゲートウェイ・ヘッダーを使用する場合のみ可能です。
- ユーザー・ヘッダーでサービス名を検索します。TmaxクライアントまたはサービスでTCPGWを要求するとき、ユーザー・ヘッダーにサービス名を入力した後、ユーザー関数のget_service_nameからサービス名を取得することができます。
- CLOPTの[-S]オプションに指定したサービスを利用します。この場合は、すべてのメッセージに対して同様なサービスが適用されます。

参考

リモート・ノードで最初のサービスを要求する場合は、上記の方式に従わず、ユーザーが`get_msg_info`でサービス名を指定する必要があります。

1.3.4. ユーザーが任意のチャンネルを指定

完全な非同期方式でTCPGWを構成した場合、ユーザーはリモート・ノードに転送するチャンネルを指定することができます。CLOPTセクションに`[-a]`を使用します。完全な非同期方式の場合、相互に応答はありません。応答データもサービス要求形式で転送される必要があります。

例えば、TmaxのクライアントまたはサービスでTCPGWに`tpacall`の`TPNOREPLY`で呼び出すと、TCPGWはサービス要求に関するUIDやいかなる情報データも保存せず、リモート・ノードにデータを転送した後、当該サービスを終了します。その後リモート・ノードから応答に関するメッセージが受信されてもこれをサービス要求として処理します。逆の場合も(リモート・ノードで先に要求)同様に処理します。

上記の場合、ユーザーはCLOPTセクションに`[-u]`オプションを使用してリモート・ノードに転送するチャンネル番号を指定することができます。

チャンネルの指定には、下記の2つの場合があります

- Tmaxで先にサービスをリモート・ノードに要求した場合です。TCPGWは`get_channel_num`関数を呼び出しますが、ユーザーは転送データを分析してチャンネルを指定することが可能です。
- リモート・ノードで先に要求した場合です。リモート・ノードでサービスを要求するとき、`get_msg_info`関数でリモート・ノードが要求したチャンネルをデータに保存し、Tmaxサービス呼び出します。Tmaxサービスは、リモート・ノードに結果を転送するため、`tpacall`の`TPNOREPLY`で再びTCPGWにサービスを転送します。ユーザーは、`get_channel_num`に保存されているチャンネルを使用してリモート・ノードに応答を転送することができます。この方式を使用することによって、TCPGWとTmaxエンジン間のチャンネルがブロックされず、サービスを要求したチャンネルに応答を転送することができます。

1.3.5. コード変換

TCPGWが実行されるマシンは、すべてASCIIコードを使用します。リモート・ノードは、使用するコードが異なるマシン同士でメッセージをやり取りするためにコードを一致させる必要があります。このようなコード変換はリモート、TCPGW両方とも可能です。TCPGWは全コードを変換せず、ASCII⇄EBCDIC、組合せ型⇄完成型コードのみ相互変換します。TCPGWでリモート・ノードにメッセージを転送する場合は、ASCIIからEBCDIC、完成型から組合せ型にコードを変換します。リモートで受信する場合は、EBCDICからASCII、組合せ型から完成型にコードを変換します。

TCPGWは、サービス処理の単位別にコードを変換します。TCPGWはサービス単位別に送受信フォーマットを登録(マップ・ファイル)して変換します。TCPGWとリモート・ノード間にコードを変換するには、すべてのサービスに対してマップ・ファイルを登録する必要があります。

さらに、TCPGWでコード変換機能を利用するには、必ずゲートウェイ・ヘッダーを使用します。TCPGWはリモートに転送するサービスのサービス名が分からないため、ゲートウェイ・ヘッダーを使用してTCPGWにリモート・サービス名を転送する必要があります。ゲートウェイ・ヘッダーのpgmname項目にリモート・サービス名を設定し、TCPGWはこの項目の値でマップ・ファイルをロードし、コードを変換します。

TCPGWは、Tmaxクライアントまたはサービスでリモートにサービスを要求する場合、ゲートウェイ・ヘッダーのpgmname項目の値でマップ・ファイルをロードします。さらに、リモート・ノードでTmaxにサービスを要求する場合は、get_msg_info関数で指定したサービス名でマップ・ファイルをロードします。

ユーザーがマップ・ファイルを登録する方法について説明します。マップ・ファイルは、ユーザーが指定したテキストファイル形式に合わせて登録します。

```
# COMMSIZEは、INPUT/OUTPUTで同時に使用されるため、INPUT/OUTPUTのうち
# 大きいサイズの値を入力します。
*COMMSIZE
1024

*INPUT
#-----
# item-name  length  data-type
# data-type: CHAR, NUMERIC(UNPACK), KOREAN, BINARY, USER-TYPE
#-----
  a             10      CHAR
  b             10     KOREAN
  c              5     NUMERIC
  d             10     BINARY

*OUTPUT
  a             10      CHAR
  b             10     KOREAN
  c              5     NUMERIC
  d             10     BINARY
  e              3     USER1
  e1            6      CHAR
  e2            10     KOREAN
  e3            8      BINARY
```

項目	説明
COMMSIZE	リモート・ノードと送受信するデータ長を指定する項目です。 すべてのサービス・メッセージ(送受信を含む)の中から最大メッセージの長さを 入力します
INPUT	TCPGWからリモート・ノードに転送するメッセージを登録する項目です。

項目	説明
	TCPGWでリモート・ノードにサービスを要求する場合、リモート・ノードに要求するメッセージのフォーマットをこの項目に登録します。一方、リモート・ノードからTmaxにサービスを要求する場合は、Tmaxサービスの処理結果のフォーマットに登録します
item-name(項目名)	単純にユーザーが参照するために使用する名前です。 TCPGWは項目名で認識しますが、内部では使用しません
length(項目の長さ)	送受信データの各項目の実長です。 各項目の長さを指定したら、TCPGW内部でオフセットに変換して使用します。オフセットは常に0から始まります
data-type(データ型)	送受信の各項目のデータ型を指定します <ul style="list-style-type: none"> – CHAR : データがすべての文字(英文字、数字)を含むことができる項目を示します – NUMERIC : データが数字のみ示す場合です。CHAR型と同様にコードを変換します。NUMERIC型とはいえ、整数型(Integer)を表すものではありません – BINARY : データがコードを変換しません。データのコード変換が不要な場合にこのデータ型を指定します – USER-TYPE : 配列(Array)型のデータ型を指定する場合に使用します。例えば、画面に複数の項目が繰り返し表示される場合、各項目をその回数の分だけ登録する手間がかかりました。また、繰り返す回数をデータで表す場合、その回数が把握できずコードの変換が難しいため、ユーザー定義タイプを指定して配列型のデータを簡単に変換できるようにします
OUTPUT	リモート・ノードからTCPGWに受信されたメッセージを登録する項目です。 TCPGWでリモート・ノードにサービスを要求する場合、リモート・ノードで処理した結果が受信されるメッセージのフォーマットを登録します。一方、リモート・ノードからTmaxにサービスを要求する場合は、Tmaxにサービスを要求するメッセージのフォーマットを登録します

上記のマップ・ファイルでUSER1は、ユーザーが指定したユーザー定義タイプです。タイプ名はユーザーが任意で指定します。ユーザー定義のデータ型は、配列データの繰り返しを表す項目なので、数字で構成される必要があります。TCPGWは、ユーザー定義タイプを配列の回数として認識します。

配列のエンドは、最後の項目名をスラッシュ(/)で始めます。上記の場合は、e1、e2、e3を配列項目として認識し、e項目で指定した回数の分だけコードを変換します。

注

配列回数はマップ・ファイルに登録せず、TCPGWとリモート・ノード間のデータ構文に表示します。

第2章 環境設定

本章では、TCP/IPゲートウェイ(TCPGW)の環境設定方法について説明します。

2.1. 概要

TCPGWは、TCPGWライブラリーと適切に実装したcustom.c、custom.hをコンパイルして作成します。TCPGWサーバーが構成されるためには、オペレーティング・システムによってディレクトリーごとに下記のようなファイルが必要となります。

- UNIX

ディレクトリー	ファイル名
lib	libtcpgw.a、libtcpgw.so、libtmaxgw.a、libtmaxgw.so
lib64	libtcpgw.a、libtcpgw.so、libtmaxgw.a、libtmaxgw.so

- Windows

ディレクトリー	ファイル名
bin	tcpgw.dll、tmaxgw.dll
lib	tcpgw.lib、tmaxgw.lib

参考

TCPGWライブラリーは、Tmaxのインストール時に各ディレクトリーの下位に作成されますが、custom.cまたはcustom.hは別途提供されるため、必要な場合は技術サポートの担当者にリクエストします。

2.2. Tmax環境構成

TCPGWを使用するためには、Tmax環境ファイルにTCPGWをサーバーとして登録します。

Tmaxサーバーの中でUCS方式のサーバーと登録方法が似ています。SVRTYPEが「UCS」ではなく「CUSTOM_GATEWAY」であるということ以外は相違点がありません。TCPGWを使用するためにTmax環境ファイルを修正するときは、**SERVER**セクションと**SERVICE**セクションのみ設定します。

下記は、Tmax環境ファイルの例です。

```

*DOMAIN
tmax      SHMKEY=88000,
          MINCLH=1,
          MAXCLH=1,
          TPORTNO=8800

*NODE
tmax1     TMAXDIR="/home/tmax",
          APPDIR="/home/tmax/appbin"

*SVRGROUP
svg1      NODENAME=tmax1

*SERVER
tcpgw     SVGNAME=svg1,
          MIN=1,
          MAX=1,
          CPC=10,
          SVRTYPE=CUSTOM_GATEWAY,
          CLOPT="-- -P 5050 -N 10 -k 71673"

*SERVICE
TCPGW1
TCPGW2    SVRNAME=tcpgw,
          SVCTIME=20,
          SVRNAME=tcpgw

```

● MIN

TCPGWのプロセス数を指定する項目です。この項目の値が1より大きい場合、TCPGWのモードによって使用するポート番号が異なります。

区分	説明
サーバー・モード	<p>MIN値が2以上の場合、2つ以上のプロセスで同じポートをListen(待ち受け)することができないため、TCPGWはCLOPT項目(TCPGWオプション)のオプションのうち[-P]オプションを利用して、入力するポート番号に自身のサーバー番号を使用します。</p> <p>例えば、ポート番号が5050の場合、1番目のプロセスは5050ポート番号、2番目のプロセスは5051ポート番号を使用します。このように1ずつ増加され、プロセス数の分だけポートを使用します。そのため、プロセス数が2つ以上の場合は、指定したプロセス数の分だけポート番号の使用が可能なのかを確認する必要があります</p>
クライアント・モード	<p>リモート・ノードで1つのポート番号を待機し、すべてのTCPGWプロセスがリモート・ノードで待機するポートに接続する方式です。</p> <p>クライアントで動作する場合は、下記の3つの方式で使用できます。</p>

区分	説明
	<ul style="list-style-type: none"> - [-C]オプションを使用してサーバー・モードのようにTCPGWプロセスごとに異なるポート番号を使用することができます。 - [-p]オプションを使って指定したポート番号にサーバー番号を足してリモート・ノードに接続を試みます。この場合、リモート・ノードは複数のポート番号をListenしている必要があります。また、プロセス数の分だけ指定したポートが使用できる必要があります。 - [-f]オプションを使用して複数のリモートに接続する方式です。接続するリモート・ノードの情報を別途のファイルに登録し、[-f]オプションを使用して該当のファイル名を指定すると、TCPGWは当該ファイルの情報をロードしてリモート・クライアントと接続します

● CPC

TmaxエンジンとTCPGW間のチャンネル数を指定する項目です。Tmaxのクライアントまたはサービスでリモート・ノードにサービスを要求する場合、同時に要求する数の分だけCPC数を指定する必要があります。しかし、非ブロック型の場合は、その数の分だけ指定する必要はなく適切に指定します。逆の場合、TCPGWはTmaxエンジンのtpacallを利用して要求するため、多くのチャネルを使用する必要はありません。

● CLOPT

[CLOPT項目\(TCPGWオプション\)](#)の説明を参照してください。

● SVCTIME

TCPGWは、1つのサーバー・プロセスに複数のサービスを指定し、サービスごとに異なるSVCTIME値を適用することができます。SERVICEセクションに複数のサービスを指定してサービスごとに異なるSVCTIME値を指定すると、TCPGWは要求したサービス別に、指定したSVCTIME値を使用します。

参考

SERVERセクションと**SERVICE**セクションの詳細設定項目の説明については『*Tmax 運用ガイド*』の「第3章 環境ファイルの設定」を参照してください。

CLOPT項目(TCPGWオプション)

TCPGWでは、Tmax環境ファイルに登録できる項目が制限されているため、CLOPT項目に多くのオプションを設定することができます。オプションによってTCPGWの動作方式が異なるため、下記の内容を十分に理解する必要があります。

下記は、TCPGWの使用オプションについての説明です。

- [-R]

- TCPGWがサーバー・モードで動作する場合、ListenしようとするIPアドレスを指定するオプションです。このオプションは指定したIPアドレスに対してのみ接続を許可する場合に使用します。
- TCPGWがクライアント・モードで動作する場合、[-M]オプションと一緒に使用して動作モードを変更し、TCPGWが接続するリモート・ノードのIPを指定することができます。
- TCPGWは入力チャンネルに対して、「1.1.1.1」IPアドレスに8080ポートで10個のチャンネルを接続します。さらに、出力チャンネルに対しては、「2.2.2.2」IPアドレスに8081ポートで10個のチャンネルを接続します。下記はその例です。このようにクライアント・モードで動作する場合は、リモート・ノードのIPアドレスを別途指定することができます。

```
-R 1.1.1.1 -P 8080 -r 2.2.2.2 -p 8081 -M 1 -N 10 -n 10
```

- IPv6プロトコルまたはInfiniBandのSDP(Socket Direct Protocol)を使用してリスンするには、-XSERVER_IPV6を「IPV6」または「SDP」に設定する必要があります。

```
-R 2011::100:100 -P 8080 -XSERVER_IPV6=IPV6 -r 2.2.2.2 -p 8081 -M 1 -N 10 -n 10
```

- [-r]

- TCPGWがクライアント・モードで動作する場合、リモート・ノードのIPアドレスを指定するオプションです。
- クライアント・モードでバックアップ機能およびマルチ・サーバー・チャンネル情報を設定するため、下記のように文法が強化されました。

```
-r mainaddr[:mainport][/backupaddr[:backupport]][(type)][,...]mainaddr
```

- backupaddrはIPアドレスであり、mainport、backupportはポート番号です。

IPv6プロトコルまたはInfiniBandのSDP(Socket Direct Protocol)を使用する場合、-XSERVER_IPV6オプションを「IPV6」または「SDP」に設定する必要があります。IPv6プロトコル環境ではmainaddr、backupaddrを設定するとき、ホスト・アドレスとポート番号を区分するために角括弧([])でアドレス部分を囲みます。

```
-r [2011::100:100]:8080(A)/[2011::100:200]:9090(A) -X CLIENT_IPV6=IPV6
```

- タイプは下記のように設定できます。

設定項目	説明
I(IN CHANNEL)	IN CHANNELが[-N]オプションで指定された数の分だけ作成されます
O(OUT CHANNEL)	OUT CHANNELが[-n]オプションで指定された数の分だけ作成されます
A(ANY CHANNEL)	IN CHANNELが[-N]オプション、OUTCHANNELが[-n]オプションで指定された数の分だけ作成されます

- [-P]

- サーバー・モードではリスンするポート番号です。当該ポート番号は、他のプロセスで使われていないポート番号である必要があります。
- クライアント・モードでは、[-R]オプションで説明したようにクライアント・ポート番号としても使用できます。

- [-p]

- リモート・ノードが待機しているポート番号を設定します。当該ポート番号は、他のプロセスで待機していないポート番号である必要があります。

- [-M]

- サーバー・モードで入力したIPアドレスとポート番号をクライアント・モードに変更するオプションです。
- [-R]、[-P]オプションを利用して入力した値は、サーバー・モードのIPアドレスとポート番号です。これを接続するリモート・ノードのIPアドレスとポート番号に変更するオプションです。
- 「-M0」は基本的なTCPGWのモード値です。しかし、「-M1」に設定するとサーバー・モードからクライアント・モードに変更されます。(デフォルト値: 0)

- [-m]

- TCGWと通信するリモート・ノードのコード・システムが異なる場合にデータを変換するオプションです。
- 送受信データをASCII EBCDIC、組合せ型、完成型に変換します。オプションを使用する場合、ユーザーは送受信データのマップ・ファイルを登録します。
- コード変換については、[「1.3.5. コード変換」](#)を参照してください。(デフォルト値: 変換しない)

- [-N]

- リモート・ノードからTCPGWに受信されるチャンネル数を指定するオプションです。
 - サーバー・モード、クライアント・モードの両方とも使用されるオプションです。
- [-n]
 - TCGWからリモート・ノードに送信されるチャンネル数を指定するオプションです。
 - TCGWがクライアント・モードで動作する場合のみ有効です。
- [-k]
 - ユーザーが共有メモリーを確保して、上述した[-N]、[-n]オプションを利用して指定されたチャンネルの状態が保存できるように、共有メモリーのキー値を入力するオプションです。
 - TCGWは、このオプションで共有メモリーを事前に確保するのではなく、ユーザーが任意で使えるように渡すだけの役割をします。一般的にこのオプションは、TCGWが最初に行われた時にユーザー・ルーチン(init_remote_info)を呼び出すときに渡し、ユーザーが共有メモリーを確保するために使われます。
- [-F]
 - 特殊な用途で使われるCLOPT項目のオプションです。上述したIPアドレスやポート番号を入力せず、環境ファイルに登録して使います。
 - 特殊な用途で使われるため、一般ユーザーは使えません。
- [-f]
 - TCGWをクライアントとして使う場合に使われるオプションです。[-r]、[-p]が1つのリモート・ノードのIPと接続するためのオプションなら、[-f]は複数のリモート・ノードと接続するためのオプションです。この場合、接続するリモート情報を別々のファイルに登録し、[-f]オプションを使用して該当のファイル名を指定すると、TCGWは当該ファイルの情報をロードしてリモート・ノードと接続します。
 - [-r]、[-p]オプションと同時に使えず、TCGWをサーバーとして使う場合は使えません。クライアント・モードでのみ使用可能です。
 - [-f]オプションを使用するには、常にリモート・ノードと接続を確立する必要があります。要求するたびにリモート・ノードに接続する[-E]オプションは使えません。
 - リモートがIPv6プロトコルを使う場合、ipaddr入力フィールドにIPv6形式のアドレスを入力し、ipv6入力フィールドに「IPV6」を設定すると、IPv6プロトコルでソケットを作成して接続します。このファイルに

指定したipv6の設定は、CLOPTセクションに指定した「-X CLIENT_IPV6」の設定より優先して適用されます。

- InfiniBandのSDP(Socket Direct Protocol)を使用する場合は、ipv6入力フィールドに「SDP」を設定すると、SDPプロトコルでソケットを作成して接続します。このファイルに指定したipv6の設定は、CLOPTセクションに指定した「-X CLIENT_IPV6」の設定より優先して適用されます。

- [-C]

- TCPGWがクライアント・モードで動作すると同時に、同じTCPGWプロセスが2つ以上(MIN=2)実行されるときにリモート・ノードと接続するポート番号を指定するオプションです。このオプションを指定すると、TCPGWは[-p]オプションで指定したポート番号にサーバー番号を足したポート番号でリモート・ノードに接続を試みます。
- 例えば、「-p 8080 -C」オプションを使用しMIN=3と設定した場合、1つ目のTCPGWプロセスは8080ポート番号、2つ目のTCPGWプロセスは8081ポート番号、3つ目は8082ポート番号を使用することになります。

- [-c]

- リモート・ノードと通信するとき、TCPGWに入力されるチャンネルと出力されるチャンネルを区別する場合に使用します。Tmaxからリモート・ノードに要求した場合のみ有効です。
- TCPGWはオプションが指定されると、Tmaxからの要求を出力チャンネルに転送します。さらに、リモート・ノードから出力されるチャンネルにデータが受信された場合は、これを以前転送したデータのACK応答として処理し、要求に対する応答は入力チャンネルで受信します。すなわち、TCPGWとリモート・ノードの間で、要求したデータが正常に受信されたというACK情報をやり取りする場合に使用するオプションです。一方、要求に対するACKは送信しません。

- [-S]

- TCPGWを非ブロック・モードまたは非同期方式で使用する際、TCPGWが要求するサービス名を登録するオプションです。
- 非ブロック・モードで使用するとき、Tmaxの送信サービスはTCPGWにtpforwardし、TCPGWはリモート・ノードから受信した応答を受信サービスにtprelayするために、リレーされるサービス名を登録します。また、非同期通信方式でのTmaxクライアントまたはサービスは、TCPGWにサービスを要求(tpacall with TPNOREPLY)して終了します。以降、リモート・ノードから受信された応答は、通常TPNOREPLYで呼び出したため、データを捨てることになります。しかし、サービス名を指定すると、TCPGWは指定したサービスにtpacallで応答データを転送します。オプションで指定されたサービスは、登録されているサービス名である必要があります。

- [-H]

- TmaxクライアントまたはサービスとTCPGWの間で、ユーザー情報データをやり取りする場合に使用します。オプションで指定したデータはリモート・ノードに転送せず、TCPGWで一時保存し、応答が受信されたらユーザー情報データに応答データを加え、呼び出したサービスに渡すときに使用します。
- 通常、非ブロック・モードで使用する際、送受信サービスが区分されており、送信サービスから受信サービスにデータを転送する場合に多く使われます。(デフォルト値: 0)

- [-h]

- [-H]オプションと同じオプションです。ただし、[-H]は同期方式と非同期方式の両方とも適用できるオプションですが、[-h]はtpforwardのtprelayで使用する呼び出しに対してのみ適用されます。(デフォルト値: 0)

- [-d]

- リモート・ノードでTmaxのサービスを呼び出し、応答データをリモート・ノードに転送する際、転送するチャンネルを選択する方法を指定するオプションです。このオプションを指定すると、リモート・ノードで要求した応答は必ず要求したチャンネルに転送されます。オプションを指定しない場合は、リモート・ノードと接続されているチャンネルのうち、使用可能な任意のチャンネルを選択して転送します。今後、拡張するためには任意の値を設定する必要があります。
- 例えば、「-d 0」のように使用し、現在設定されている値は無視されます。

- [-E]

- TCGWがクライアント・モードで動作するとき、要求ごとにチャンネルを接続および解除する場合に使用します。
- このオプションを使用すると、リモート・ノードと接続されているチャンネルが存在しないため、リモート・ノードから先にサービスを要求することができません。また、毎回チャンネルが接続/解除されるため、処理速度が遅くなります。アウトバウンド要求による接続が失敗した場合、[-X OUTBOUND_REQ]オプションを使用するとoutmsg_recoveryが呼び出され、TPESYSTEMエラーが返されます。

- [-e]

- get_msg_info()で受信したデータが異常な場合は-1を返し、環境設定ファイルのCLOPTセクションに[-e]オプションを設定すると、当該チャンネルが終了します。(TCPGW 3.9バージョン以上)

- [-a]

- TCGWを非同期通信方式で使用するためのオプションです。

- 非同期通信方式は、サービスを要求してから終了される形式なので、TCPGWはサービスに対する応答を他のサービスに対する要求として認識します。
- [-u]
 - Tmaxクライアントまたはサービスが、TCPGWでリモート・ノードにサービスを要求する際、ユーザーがチャンネルを指定できるようにするオプションです。
 - このオプションを使用するには、[-k]オプションを使用してチャンネルの状態情報を保有している必要があります。ユーザーが指定したチャンネルが使用できない場合は、TCPGWがエラーを返します。
- [-w]
 - リモート・ノードとTCPGW間の通信時、その間には多くの通信装備やファイアウォールが存在することがあります。このとき、特定の装備やファイアウォールで、両者間のチャンネルが一定時間の間使用されない場合は、一方的にチャンネルを切断することができます。
 - チャンネルが切断された場合、通常はTCPGWまたはリモート・ノードのプロセスでそれを検知し、チャンネルを再接続することが可能です。しかし、切断されたことが検知できない場合が度々発生します。そのため、TCPGWでこのオプションを使用して指定した時間(秒)の間、該当のチャンネルが使用されない場合は、無条件でチャンネルを切断し、再接続されるように設定するオプションです。
- [-Y]
 - リモート・ノードでTmaxのサービスを要求した後、サービスは正常に処理されたものの、結果をリモート・ノードに転送できない場合、以前処理したサービスに対して取り消し処理ができるように復旧サービスを指定するオプションです。
 - リモート・ノードでTmaxのサービスを呼び出した後、リモート・ノードとセッションがすべて切断されると、サービス処理結果はリモート・ノードに転送されず捨てられます。この場合、サービスは正常に処理されたが、リモート・ノードからはエラーとして判断するため、問題が発生する可能性があります。
 - ユーザーがこのオプションにサービス名を指定すると、ゲートウェイがリモートに結果を転送できない場合、リモート・ノードに転送するデータを指定したサービスで呼び出すことになります。
- [-t]
 - TCGWがクライアント・モードで動作する場合、TCPGWは先にリモート・ノードと接続を確立します。このとき、リモート・ノードとの接続が切断されるか、または接続できない場合にこのオプションが設定されていないと、一定時間(3秒)が経つか、サービスが再要求されたら再接続が行われます。この際、リモート・ノードとの接続の試行が頻繁に行われるため、システムに負荷がかかります。

- システムの負荷問題を解決するには、リモート・ノードとの接続を試行する時間間隔(秒)を設定した後、
[-i]オプションを使って時間を指定します。TCPGWは指定した時間間隔でリモート・ノードとの接続を試
みます。

- [-i]

- リモート・ノードからデータを受信するとき、データのエンドを示す特定の文字またはビットストリームが存
在する場合に必要なオプションです。
- このオプションを指定すると、データを受信時にユーザーのchk_end_msgが呼び出されます。

- [-x]

- チャンネルの障害有無の検知(TCPLレベルのPING/PONG)機能を有効にします。
- 下記は、オプションの各引数の処理内容についての説明です。

設定値	説明
0	メイン・チャンネルの障害復旧が検知されると、直ちにメイン・チャンネルに切り替わりま す(デフォルト値)
1	メイン・チャンネルの障害復旧が検知されても、安全なタイミング(チャンネルの未使用時) にメイン・チャンネルに切り替わります

- オプションは、set_ping_msg関数およびchk_pong_msg関数と同時に使用されます。

- [-b]

- [-x]オプションでチャンネルの障害検知機能が有効になった場合に、set_extping_msgが呼び出される
ときに使用可能なメッセージ本体の最大サイズを指定します。(デフォルト値：0、最大値：268435455)
- Pingメッセージを転送するとき、reset_ping_msg関数により転送するメッセージの内容を修正すること
ができます。メッセージ本体のサイズは、このオプションで指定した最大サイズ内で自由に指定できま
す。

- [-y]

- UIDとして利用できるメッセージ番号(sequence number)の最大値を設定するオプションです。
(デフォルト値：50000、最大268435455まで指定可能)
- [-y]オプションで指定された値に+1して適用されます。したがって、デフォルト値として1~500001まで設
定されます。

- [-X CHANNEL_IRT]
 - TCPGWをCOUSINで構成した場合、Channel IRT機能(IRT COUSIN)が動作するように設定するオプションです。
 - TCPGWがリモートと接続されていない場合は、COUSINグループに設定されている別のTCPGWに要求を渡します。
- [-X CONN_RETRY=n]
 - TCPGWがクライアント・モードで動作するとき、リモートに接続が失敗した際に再試行する回数を設定するオプションです。
 - このオプションを指定しない場合は、デフォルトで1回のみ接続を試みます。
 - メイン・アドレスに再試行が失敗した場合、指定した回数だけバックアップ・アドレスに再試行します。
- [-X CONN_TIMEOUT=n]
 - TCPGWがクライアント・モードで動作するとき、リモートに接続を試行する際に待機する最大のタイムアウト時間を設定するオプションです。(デフォルト値：3、単位：秒)
- [-X CONN_NONBLOCK]
 - TCPGWがクライアント・モードで動作するとき、リモートに非ブロック・モードで接続を試みます。接続の試行中にTmaxからリモートへの要求を受信すると、接続が完了したり失敗するまで内部で待機してから適切に処理します。
- [-X SERVER_IPV6=["IPV4"]|"IPV6"]|"SDP"|Y|N]]
 - TCPGWがサーバー・モードで動作するとき、リスン・ソケットを作成時に使用するプロトコルを指定するオプションです。重複設定が可能な場合は、区切り子のコンマ(,)を使って複数の項目を設定できます。
 - 下記は、設定値についての説明です。

設定値	説明
"IPV4"またはN	IPv4プロトコルを使用します。オプションを設定しない場合も同じです
"IPV6"またはY	IPv6プロトコルを使用します
"SDP"	InfiniBandのSDPを使用します。"IPV4"または"IPV6"と一緒に使用できます

- [-X CLIENT_IPV6=["IPV4"]|"IPV6"]|"SDP"|Y|N]]

- TCPGWがクライアント・モードで動作するとき、リモートに接続するためのソケットを作成時に使用するプロトコルを指定するオプションです。重複設定が可能な場合は、区切り子のコンマ(,)を使って複数の項目を設定できます。
- 下記は、設定値についての説明です。

設定値	説明
"IPV4"またはN	IPv4プロトコルを使用します。オプションを設定しない場合も同じです
"IPV6"またはY	IPv6プロトコルを使用します
"SDP"	InfiniBandのSDPを使用します。"IPV4"または"IPV6"と一緒に使用できます

2.2.1. サーバー/クライアントTCPGW

下記は、サーバー/クライアントTCPGW環境設定の例です。

```
*DOMAIN
...
*NODE
...
*SVRGROUP
...
*SERVER
testtcpgw      SVGNAME = svg1,
                MIN = 1,
                MAX = 1,
                CPC = 10,
                SVRTYPE = CUSTOM_GATEWAY,
                # クライアントTCPGWの場合
                CLOPT = "-- -r 168.126.185.131 -p 5050 -n 5 -k 71673 "
                # サーバーTCPGWの場合
                #CLOPT = "-- -P 5050 -N 5 -k 71673 "

*SERVICE
TESTTCPGW      SVRNAME = testtcpgw
```

上記のように環境ファイルを作成した場合、Tmaxのブート時にtesttcpgwという名前の1つのTCPGWが起動されます。testtcpgwのサービス名はTESTTCPGWであり、タイムアウト値は設定されていないため無限に待機します。

クライアントTCPGWの場合、リモート・ノードのIPアドレスは「168.126.185.131」です。リモート・ノードのListenポート番号は「5050」になります。testtcpgwは起動時にリモート・ノードと5つの接続を確立します。サーバーTCPGWの場合、環境設定はサーバーで動作します。Listenポート番号は「5050」で、testtcpgwはリモート・ノードと5つの接続を確立します。

いずれの場合でも[-k]オプションで指定した値は、custom.cの関数(init_remote_info())の引数として渡され、ユーザーは必要な情報をこのキーを利用して共有メモリーに保存することができます。

2.2.2. サービス・ブロック型TCPGW

TCPGWはリモート・ノードと接続された後は、サーバー/クライアントを問わず同期型/非同期型の両方とも使用できるため、サーバー/クライアント・モードの説明は省略します。

```
*DOMAIN
...
*NODE
...
*SVRGROUP
...
*SERVER
testtcpgw      SVGNAME = svg1,
                MIN = 1,
                MAX = 1,
                CPC = 10,
                SVRTYPE = CUSTOM_GATEWAY,
                #クライアントTCPGWの場合
                CLOPT = "-- -r 168.126.185.131 -p 5050 -n 5 -k 71673 -d 0"
                #サーバーTCPGWの場合
                #CLOPT = "-- -P 5050 -N 5 -k 71673 -d 0"

*SERVICE
TESTTCPGW      SVRNAME = testtcpgw, SVCTIME = 30
```

上記のように環境ファイルを作成した場合、Tmaxのブート時にtesttcpgwという名前の1つのTCPGWが起動されます。

testtcpgwのサービス名はTESTTCPGWです。タイムアウトは30秒になっており、30秒以内に応答がない場合はタイムアウトを返します。タイムアウトは、Tmaxのクライアントまたはサービスからリモート・ノードへの要求に対してのみタイムアウトをチェックします。一方、リモート・ノードで要求したサービスに対しては、要求した該当のサービスでタイムアウトを設定します。

Tmaxのクライアントまたはサーバーからリモート・ノードにサービスを要求したとき、TCPGWはUIDを使用します。UIDはメッセージごとに一意の値であり、要求に対する応答を返す時にTCPGWに渡す必要があります。そうすることで該当のサービスを要求したサービスまたはクライアントに応答を返すことができます。

リモート・ノードからサービスを要求したとき、TCPGWは応答を要求したチャンネルに応答を転送します。ただし、要求したチャンネルが切断された場合、データは消滅されます。

2.2.3. サービス非ブロック型TCPGW

TCPGWは、通常Tmaxのサービスからリモート・ノードにサービスを要求する場合に使用されます。一般のTmaxクライアントでは、この方式を利用して直接サービスを呼び出すことができません。Tmaxクライアントで直接使用できる方式は、ブロック型方式または非同期方式です。

```
*DOMAIN
...
*NODE
...
*SVRGROUP
...
*SERVER
testtcpgw      SVGNAME = svg1,
                MIN = 1,
                MAX = 1,
                CPC = 5,
                SVRTYPE = CUSTOM_GATEWAY,
                #クライアントTCPGWの場合
                CLOPT = "-- -r 168.126.185.131 -p 5050 -n 5 -k 71673 -S RECVSVC
                        -H 20 -d 0"
                #サーバーTCPGWの場合
                #CLOPT = "-- -P 5050 -N 5 -k 71673 -S RECVSVC -H 20 -d 0"
sendsvr        MIN = 1, MAX = 1
recvsvr        MIN = 1, MAX = 1

*SERVICE
TESTTCPGW      SVRNAME = testtcpgw, SVCTIME = 30
SENDSVC        SVRNAME = sendsvr
RECVSVC        SVRNAME = recvsvr
```

上記のように環境ファイルを作成した場合、Tmaxのブート時にtesttcpgwという名前の1つのTCPGWが起動されます。

testtcpgwのサービス名はTESTTCPGWです。タイムアウトは30秒になっており、30秒以内に応答がない場合はタイムアウトを返します。リモート・ノードの応答が、指定した時間(30秒)を超過すると、TCPGWはタイムアウトをRECVSVCに渡します(tprelay)。RECVSVCサービスでは必ずtpurcode値を確認し、値が0より大きい場合はエラーが発生した場合なので、適切な処理を行います。タイムアウトのみならずすべてのエラーに対して、tpurcodeに値が渡されます。エラーについての詳細内容は、「[付録 A. TCGWのエラーコード](#)」を参照してください。

Tmaxのクライアントまたはサーバーで先にSENDSVCサービスを呼び出すと、SENDSVCで事前作業を実行した後、TESTTCPGWサービスに制御を渡します(tpforward)。TCPGWは、Tmaxエンジンにチャンネルの解除メッセージを転送した後、リモート・ノードにサービスを要求し、応答を受信する際に[-S]オプションに指定したサービスに結果を渡します(tprelay)。RECVSVCが処理結果に対する作業を実行した後返すと、SENDSVCを呼び出したクライアントまたはサービスに応答が渡されます。この方式を使用すると、小数の

サービス(SENDSVC)でより多くの処理ができます。ブロック型と同様に必ずUIDを使用する必要があります。

2.2.4. リモート同期型と非同期型TCPGW

リモート・ノードで要求するサービスに対する同期型/非同期型の区分は、[-d]オプションを使用するかどうかによって決められます。また、ユーザー定義関数(get_msg_info)で、flagsにTPNOREPLYを設定する方法もあります。

前者の場合、[-d]オプションを使用しないとサービスに対する応答は他のチャンネルに転送される可能性があります。後者の場合は、応答をリモート・ノードに転送しません。[-d]オプションを指定した場合にも、ユーザー定義関数(get_msg_info)でflagsにTPNOREPLYを設定するとサービスの応答はリモート・ノードに転送されません。

```
*DOMAIN
...
*NODE
...
*SVRGROUP
...
*SERVER
testtcpgw      SVGNAME =  svg1,
                MIN = 1,
                MAX = 1,
                CPC = 5,
                SVRTYPE = CUSTOM_GATEWAY,
                #非同期型
                CLOPT = " -- -P 5050 -N 5 -k 71673"
                #同期型
                #CLOPT = "-- -P 5050 -N 5 -k 71673 -d 0"

*SERVICE
TESTTCPGW      SVRNAME = testtcpgw
```

リモート・ノードでTmaxのサービスを要求する場合、TCPGWはtpacallを使用するため、応答を受けずにtpacallを同時に使用できる数が制限されています(デフォルト値:8個)。同時要求が8個以上の場合は、DOMAINセクションのMAXSACALLの数を適切に指定する必要があります。

2.2.5. 再接続TCPGW

通常、リモート・ノードと通信する際、両チャンネル間の通信は、複雑な通信装備を経由して行われるか、ファイアウォールを間に設けることが大いにあります。この場合、両者間のチャンネルが一定時間の間使用されないと、通信装備またはファイアウォールで該当のチャンネルが切断され、通信ができなくなる問題が発生します。これの対策として、両者間に一定の間隔でシグナルをやり取りすることも可能ですが、そのためには、

TCPGWのみならず相手のプログラムまで考慮しなければなりません。したがって、この方法は使用せず、一定時間の間チャンネルを使用しない場合は、チャンネルを切断して再接続する方法を使用します。

下記のように設定すると、TCPGWは600秒(10分)間隔で、600秒間一度も使用していないチャンネルを切断し、再接続を行います。

```
*DOMAIN
...
*NODE
...
*SVRGROUP
...
*SERVER
testtcpgw      SVGNAME = svg1,
                MIN = 1,
                MAX = 1,
                CPC = 5,
                SVRTYPE = CUSTOM_GATEWAY,
                #非同期型
                CLOPT = " -- -P 5050 -N 5 -k 71673 -w 600"
                #同期型
                #CLOPT = "-- -P 5050 -N 5 -k 71673 -w 600"

*SERVICE
TESTTCPGW      SVRNAME = testtcpgw
```

2.2.6. 複数のリモート・ノードと接続するクライアントTCPGW

下記は、複数ノードのIPと接続するクライアントTCPGWの環境設定の例です。

```
*DOMAIN
...
*NODE
...
*SVRGROUP
...
*SERVER
testtcpgw      SVGNAME = svg1,
                MIN = 4,      #下記の設定ファイルの環境と同じ数(addr)に変更
                MAX = 4,
                CPC = 10,
                SVRTYPE = CUSTOM_GATEWAY,
                CLOPT = "-- -f config -n 5 -k 71673"

*SERVICE
TESTTCPGW      SVRNAME=testtcpgw
```


下記は、設定ファイルの例です。

#	tcpgw_no	ip_addr	port_no	in_channel	out_channel
0		192.168.1.1	7717	5	5
1		192.168.1.2	7727	5	5
2		192.168.1.3	7737	5	5
3		192.168.1.4	7747	5	5

下記は、IPv6プロトコル環境での設定ファイルの例です。

#	tcpgw_no	ip_addr	port_no	in_channel	out_channel	ipv6
1		192.168.1.1	7717	5	5	
2		www.tmax.co.kr	7727	5	5	IPV4
3		2011::100:100	7737	5	5	IPV6
4		2011::100:200	7747	5	5	IPV6

設定ファイルの各項目についての説明は下記のとおりです。

設定項目	説明
tcpgw_no	TCPGWプロセス番号で、0から開始します
ip_addr	remote hostname or ip address
port_no	remote port number
in_channel	input channel number
out_channel	output channel number

2.3. ユーザー・ヘッダー環境設定および使用方法

TCPGWは、サーバー/クライアント・モード、同期型/非同期型方式のすべてにおいて、ユーザー・ヘッダーを設定して使用することができます。ユーザー・ヘッダーはTmaxクライアントまたはサービスでリモート・ノードにサービスを要求する場合のみ使用可能で、逆の場合には使用できません。

ユーザー・ヘッダーに指定したデータはリモート・ノードに転送されず、一時のTCPGWでUID別に保存されており、応答が受信されると応答データからUIDを見つけ、該当するUIDのユーザー・ヘッダー・データと応答データを渡すことになります。ユーザー・ヘッダー・データの長さは、最大256 Bytesまで使用できます。

下記は、ユーザー・ヘッダー環境設定の例です。

*DOMAIN	
...	
*NODE	
...	
*SVRGROUP	
...	
*SERVER	
testtcpgw	SVGNAME = svg1,

```

MIN = 1,
MAX = 1,
CPC = 10,
SVRTYPE = CUSTOM_GATEWAY,
#クライアントTCPGWの場合
CLOPT = "-- -r 168.126.185.131 -p 5050 -n 5 -H 9 -h 10"
#サーバーTCPGWの場合
#CLOPT = "-- -P 5050 -N 5 -H 9 -h 10"

*SERVICE
TESTTCPGW          SVRNAME = testtcpgw

```

ユーザー・ヘッダーはすべてのモードで使用可能なオプションです。ユーザー・ヘッダーは、2つに分けて指定できます。tpforward方式で使用する時(非ブロック型)と、それ以外のサービス(ブロック型、非同期型)を使用する場合で、ユーザー・ヘッダーの長さは異なる値を指定することができます。

[-H]オプションは、全タイプのサービス要求に対してユーザー・ヘッダーを指定するときに使用するオプションです。一方、[-h]オプションは、tpforward方式(非ブロック型)でのみ使用できるオプションです。

上記の環境設定では、TCPGWはユーザー・ヘッダーが指定されている場合、リモート・ノードに転送するデータのうち、一般サービスは"9Bytes"、tpforwardの場合は"10Bytes"を保存し、それ以外のデータはリモート・ノードに転送します。リモート・ノードから応答が受信されると、保存されているユーザー・ヘッダーと応答データを一緒に返します。

ユーザー・ヘッダーは、ユーザーが任意で利用できるデータなので、様々な方法で使うことができます。

下記は、ユーザー・ヘッダーを使用する一般的な方法です。

- tprelayおよび非同期サービス名を指定

非ブロック型の方式で使用する場合、ユーザー・ヘッダーにtprelayされるサービスを指定することができます。この場合、[-S]オプションで指定したサービスより優先されます。ユーザー・ヘッダーにもサービスを指定し、[-S]オプションでサービスを指定した場合、TCPGWは先にユーザー・ヘッダーでユーザー関数(get_service_name)を呼び出してサービスを検索します。サービスが見つからなかった場合は、[-S]オプションで指定したサービスを使用します。

非同期型方式(tpacallのTPNOREPLAY)で呼び出した場合も、応答データに対して上記と同様に処理します。

- キー・データの保存

非ブロック・モードで使用する場合、送信と受信サービスで区分されます。送信サービスでデータベースに適切な作業を処理し、リモート・ノードにデータを転送してから送信サービスは終了されます。その後、受信サービスで、送信サービスが処理したデータベースのキー情報を確認したい場合や、リモート・ノードでエラーが発生した場合、受信サービスにデータベースをリカバリーするため、重要データを保存する場合に使用されます。

2.4. チャンネル・バックアップの設定

[-E]オプションを使用して、要求ごとにチャンネルを接続し解除することができます。この場合も、バックアップ・チャンネルが指定できます。CLOPT項目に[-r]オプションを使用して、メイン・サーバーおよびバックアップ・サーバーのアドレスとポート番号を指定し、バックアップ・チャンネルを設定します。

下記は、環境設定の例です。

```
サーバー名      CLOPT = literal
```

下記は、CLOPTの使用方法的例です。

```
CLOPT = "-- -r [ip]:[port]/[backup ip]:[backup port] -E"
```

参考

NODEセクションにCRYPTPORTを指定する場合、CRYPTPORTに指定されたポートで接続したクライアントのデータのみ暗号化されます。NODEセクションの設定項目の詳細については『*Tmax 運用ガイド*』の「第3章 環境ファイルの設定」を参照してください。

2.5. Pingメッセージ・チェック時のデータ転送

set_ping_msg()関数とchk_pong_msg()関数を使用してPingメッセージを転送し、チャンネルの障害有無を周期的に監視することができます。この場合、Pingメッセージはヘッダーのみ存在するため、データ部分を転送することができません。

set_ping_msg()、chk_pong_msg()関数の代わりに、set_extping_msg()、chk_extpong_msg()、reset_extping_msg()関数を使用します。この関数についての詳細説明は、[「3.3. custom.c」](#)を参照してください。

下記は、環境設定の例です。

```
[DEFAULT :]  
SERVER名      [,CLOPT = -b len]
```

項目	説明
CLOPT = -b len	選択項目としてPingメッセージ・データの最大長を指定します

下記のように、CLOPT項目に[-r]オプションを設定し、Pingメッセージのデータを追加して転送することができます。

```
CLOPT = "-- -r 192.168.1.31 -p 5060 -n 1 -x 1 -b 64"
```


第3章 ユーザー・プログラムと関数

本章では、TCP/IPゲートウェイを使用するため、`custom.h`、`custom.c`、`register.c`を修正する方法について説明します。

3.1. 概要

Tmaxの環境を設定してTCPGWを登録した後、TCPGWを使用するために**`custom.h`**、**`custom.c`**を修正します。

1. TCPGWの使用目的に合わせて、`custom.h`で**`msg_header_t`**構造体を修正します。

TCP/IPはデータのエンドが分からず、データ受信終了のタイミングが把握できないため、可変長データの処理が困難です。そのためTCPGWは、`msg_header_t`構造体を使用してリモート・ノードとメッセージ・ヘッダーを設定しており、実データの長さが把握できる構造になっています。構造体の情報は、TCPGWとリモート・ノード間にメッセージ・ヘッダーを指定する構造体なので、該当のリモート・ノードと通信時に使用するメッセージ・ヘッダーを登録します。

TCPGWは、リモート・ノードからデータを受信する際、まず**`msg_header_t`**の長さの分だけデータを受信し、`msg_header_t`の構造体から実データの長さを取得した後、実データを受信します。両者の通信を円滑に行うには、`msg_header_t`を正確に登録する必要があります。

ユーザーが定義した**`msg_header_t`**の情報は、TCPGWとリモート・ノードの通信でのみ使用し、サービスを呼び出すときは渡されません。

2. `custom.h`を修正した後は、`custom.c`を適切に修正します。`custom.cl`には、様々なユーザー定義関数が存在します。その詳細内容については「[3.3. custom.c](#)」で説明します。
3. 修正した**`custom.h`**、**`custom.c`**とTCPGWライブラリーおよび**`register.c`**をリンクしてコンパイルすると、TCPGWが完成されます。

参考

このとき作成された実行ファイル名は、Tmax環境ファイルのSERVERセクションに登録された名前と同じである必要があります。

3.2. custom.h

custom.hは、TCPGWで使用するmsg_info_t構造体が含まれるファイルです。ユーザーは、リモート・ノードとTCPGW間に通信されるその他のメッセージ構造体、またはヘッダーを修正することができます。

msg_info_t構造体は、TCPGWライブラリー内部とユーザー関数の間に使用されるため、メンバー変数、タイプ、長さなどを修正してはいけません。

- msg_info_t

TCPGWライブラリー内部とユーザーの修正が必要なcustom.cファイルの関数で使用する構造体です。custom.cファイルでは、リモート・ノードからメッセージを受信した直後に呼び出されるget_msg_info()関数で、この構造体に適切な値を設定します。

また、リモート・ノードにメッセージを送信する直前に呼び出されるput_msg_info()関数で、構造体の値でリモート・ノードに送信したデータを構成する必要があります。get_msg_info()とput_msg_info()の修正事項については、[「3.3. custom.c」](#)を参照してください。

TCPGWで使用するmsg_info_t構造体は下記のとおりです。

```
typedef struct msg_info {
    char   svc[20];
    int    err;
    int    len;
    int    uid;
    int    flags;                /* フラグを設定します。(TPNOREPLYなど) */
    int    msgtype;
    int    channel_id;
} msg_info_t;
```

下記は、msg_info構造体のメンバーについての説明です。

メンバー	説明
char svc[20]	サービス名を指定します。リモート・ノードでサービスを要求するとき、項目にサービス名を指定すると、TCPGWはこれを利用してサービスを呼び出します
int err	サービスを要求して応答データを受信した場合のエラー有無を示します
int len	送受信データの長さを示します
int uid	同期通信時に、TCPGWが作成したUIDを有する項目です
int flags	リモート・ノードでTmaxにサービスを要求する場合に使用します <ul style="list-style-type: none">– NOFLAGS : リモート・ノードで要求したサービスの応答を受ける場合– TPNOREPLY : リモート・ノードで要求したサービスの応答を受けず、サービス要求のみする場合(TmaxのtpacallのTPNOREPLYと同様)
int msgtype	現在は使用しません

メンバー	説明
int channel_id	送受信チャンネル番号です

- msg_header_t

custom.hに、リモート・ノードとTCPGW間のメッセージ・ヘッダー・データ構造を定義する必要があります。下記は、TCPGWで使用するメッセージ・ヘッダーの構造体です。

```
typedef struct msg_header {
    int    length;
} msg_header_t;
```

3.3. custom.c

custom.cは、リモート・ノードとTCPGWが通信するために開発者が実装するもので、TCPGWライブラリー(libtcpgw.a、libtcpgw.so)と一緒にコンパイルして使用します。custom.cは定義された形式に合わせて実装します。形式は下記のとおりです。

- 変数

custom.cには、下記の変数を定義します。

- msg_header_size

ライブラリー内部でユーザーが定義したメッセージ・ヘッダーのサイズとして認識され使用される変数です。下記のとおり、この変数をユーザー定義メッセージ・ヘッダーのサイズに設定します。

```
int msg_header_size = sizeof(msg_header_t);
```

- comm_header_size

msg_header_sizeが0に設定された場合、この変数がmsg_header_sizeと同じ用途で使われます。msg_header_size変数と一緒に設定された場合は、msg_header_size値を使用します。

```
int comm_header_size = 0;
```

- 関数

下記は、custom.cで実装する関数の一覧です。

関数	説明
init_remote_info	リモート・ノードと接続を確立する前に呼び出される関数です
remote_connected	リモート・ノードと接続を確立した後に呼び出される関数です

関数	説明
remote_connected_ipv6	IPv6プロトコル環境でリモート・ノードと接続を確立した後に呼び出される関数です
remote_closed	リモート・ノードと接続を終了した後に呼び出される関数です
allow_connection	リモート・ノードと新しい接続を確立する前に呼び出される関数です。当該接続要求の許可可否を戻り値で決めることができます
allow_connection_ipv6	IPv6プロトコル環境でリモート・ノードと新しい接続を確立する前に呼び出される関数です。当該接続要求の許可可否を戻り値で決めることができます
get_msg_length	リモート・ノードから要求または応答が受信される場合に呼び出す関数です。戻り値の分だけ実データを再読み取りする関数です
get_msg_info	リモート・ノードから要求または応答が受信されデータを読み取った後、TCPGWライブラリーとcustom.cのインターフェースの役割をするinfoを参照あるいは加工する関数です
get_channel_num	Tmaxサービスまたはクライアントから要求したデータをリモート・ノードに転送する際、ユーザーがチャンネルを選択できるようにする関数です
put_msg_info	リモート・ノードにメッセージを転送しようとする際に呼ぶ出される関数です
put_msg_complete	リモート・ノードにメッセージを転送した後に呼び出される関数です
get_service_name	tpreply()またはtpacall()するサービス名のエラーコードに従って設定する関数です
prepare_shutdown	TCPGWが終了する直前に呼び出される関数です
set_service_timeout	サービス・タイムアウトが発生した場合、ユーザーが呼び出せる関数です
chk_end_msg	リモート・ノードからデータを受信するとき、データのエンドを示す特定の文字あるいはビットストリームが存在する場合、ユーザーが呼び出せる関数です
inmsg_recovery	リモート・ノードからの要求を処理したとき、エラーが発生する場合に呼び出す関数です
outmsg_recovery	リモート・ノードに要求を送信するとき、エラーが発生する場合に呼び出されます
get_extmsg_info	get_msg_infoと一部機能を除いて基本的に同じ機能を持ちます
put_extmsg_info	put_msg_infoと一部機能を除いて基本的に同じ機能を持ちます
set_ping_msg	チャンネル障害監視のために送信するメッセージ設定および周期、タイムアウトなどが設定できるユーザー関数です
chk_pong_msg	チャンネル障害監視の応答メッセージ有無をチェックするユーザー関数です
set_extping_msg	チャンネル障害を検知したとき、サーバーに送信するメッセージを設定する関数です
chk_extpong_msg	チャンネル障害を検知したとき、サーバーから受信されたメッセージを確認する関数です

関数	説明
reset_ping_msg	TCP/IP Ping(チャンネル障害の検知)メッセージの転送可否と転送メッセージを再設定するため、周期的に呼び出される関数です
reset_extping_msg	reset_ping_msgと一部機能を除いて基本的に同じ機能を持ちます
set_error_msg	リモート・ノードとのやり取り中にエラーが発生した場合、自動的に呼び出される関数です
get_msg_security	get_msg_infoが呼び出された後にデータを加工できる関数です
put_msg_security	put_msg_infoが呼び出される前にデータを加工できる関数です

参考

本節で説明した関数の使用例については「[第4章 例](#)」で説明します。

3.3.1. init_remote_info

リモート・ノードと接続を確立する前に呼び出される関数です。TCPGWが自身の初期化作業を終了した後に即時呼び出される関数であり、一度だけ呼び出されます。Tmax環境ファイルのCLOPTに[?k]オプションで共有メモリー・キーを設定した場合、接続情報などを保存するために共有メモリーを作成するロジックを実装することができます。場合によって、内部ロジックを実装しなくても構いません。

● 使用方法

```
int init_remote_info(char *myname, int mynumber, int num_channel, int key)
```

● パラメータ

パラメータ	説明
myname	TCPGWサーバー名です
mynumber	同じTCPGWが同時に複数実行される場合、それぞれのプロセスを識別するためのTCPGWプロセス番号であり、0から開始します
num_channel	TCPGWが接続している最大のチャンネル数です。Tmax環境ファイルの[-n]、[-N]オプションに設定した値の合計です
key	Tmax環境ファイルで[?k]オプションで設定した共有メモリー・キー値です

3.3.2. remote_connected

リモート・ノードと接続を確立したときに呼び出される関数です。リモート・ノードと接続を確立した後に必要な作業があれば、この関数で実装します。チャンネルと同数の関数が呼び出され、途中チャンネルが解除された後再接続されるときにも呼び出されます。

- 使用方法

```
int remote_connected(int index, int addr, int portno, int type, int fd)
```

- パラメータ

パラメータ	説明
index	TCPGWが[-n]または[-N]オプションで複数のチャンネルを接続している場合、それぞれのチャンネルに対する自身の索引値です
addr	リモート・ノードのアドレスです
portno	接続しているサーバーのポート番号です。TCPGWがクライアント・モードで動作する場合はリモート・ノードのポート番号であり、TCPGWがサーバー・モードで動作する場合は、リスンしているソケットのポート番号です
type	リモート・ノードと接続されたチャンネルのタイプです。IN_CHANNELかOUT_CHANNELかを表します
fd	リモート・ノードと接続されたソケット番号です

3.3.3. remote_connected_ipv6

IPv6プロトコル環境でリモート・ノードと接続を確立したときに呼び出される関数です。リモート・ノードと接続を確立した後に必要な作業があれば、この関数で実装します。チャンネルと同数の関数が呼び出され、途中チャンネルが解除された後再接続されるときにも呼び出されます。

IPv6環境ではremote_connectedコールバック関数ではなく、remote_connected_ipv6コールバック関数を設定する必要があります。このコールバック関数を設定せずにIPv6環境で使用すると、remote_connected関数が呼び出されますが、ipaddrの値に任意の値が設定され、slogに警告メッセージが記録されます。

- 使用方法

```
#include <arpa/inet.h>
int remote_connected_ipv6(int index, struct sockaddr *saddr, int portno, int type, int fd)
```

- パラメータ

パラメータ	説明
index	TCPGWが[-n]または[-N]オプションで複数のチャンネルを接続している場合、それぞれのチャンネルに対する自身の索引値です
saddr	リモート・ノードのアドレスおよびポート番号が保存されます。IPv4またはIPv6プロトコル環境に適切なsockaddr型の構造体でキャストして使用します
portno	接続されたサーバーのポート番号です。TCPGWがクライアント・モードで動作する場合はリモート・ノードのポート番号であり、TCPGWがサーバー・モードで動作する場合はリスンしているソケットのポート番号です
type	リモート・ノードと接続されたチャンネルのタイプです。IN_CHANNELかOUT_CHANNELかを表します
fd	リモート・ノードと接続されたソケット番号です

● 例

```
int
remote_connected_ipv6(int index, struct sockaddr *saddr, int portno, int type,
int fd)
{
    char buf[INET6_ADDRSTRLEN];
    char *ipaddr = NULL;
    int cli_portno;

    if (index < 0)
        return -1;

    if (saddr->sa_family == AF_INET6) {
        ipaddr = (char *)inet_ntop(AF_INET6, &(((struct sockaddr_in6
*)saddr)->sin6_addr), buf, sizeof(buf));
        cli_portno = ntohs(((struct sockaddr_in6 *)saddr)->sin6_port);
    } else if (saddr->sa_family == AF_INET) {
        ipaddr = (char *)inet_ntop(AF_INET, &(((struct sockaddr_in
*)saddr)->sin_addr), buf, sizeof(buf));
        cli_portno = ntohs(((struct sockaddr_in *)saddr)->sin_port);
    }
    if (ipaddr == NULL)
        ipaddr = "unknown";

    printf("remote_connected_ipv6, REMOTE[%d] IPADDR[%s] CLIPORT[%d] SVRPORT[%d]
TYPE[%d] fd[%d] connected\n",
        index, ipaddr, cli_portno, portno, type, fd);

    return 1;
}
```

3.3.4. remote_closed

リモート・ノードとの接続を終了したときに呼び出される関数です。リモート・ノードとの接続が切断された後に必要な作業があれば、この関数で実装します。init_remote_info関数で共有メモリーを作成するロジックを実装した場合、この関数で解除ロジックを実装します。チャンネルと同数の関数が呼び出されます。

- 使用方法

```
int remote_closed(int index, int type)
```

- パラメータ

パラメータ	説明
index	TCPGWが[-n]または[-N]オプションで複数のチャンネルを接続している場合、それぞれのチャンネルに対する自身の索引値です
type	リモート・ノードと接続されたチャンネルのタイプです。IN_CHANNELかOUT_CHANNELかを表します

3.3.5. allow_connection

リモート・ノードがTCPGWで接続を試みたり、あるいはTCPGWがリモート・ノードに接続を試みて接続に成功した場合、remote_connected()コールバック関数が呼び出される前に呼び出されます。当該リモートとの接続を許容するかどうかを指定します。

- 使用方法

```
int allow_connection(int addr, int portno, int type, int fd)
```

- パラメータ

パラメータ	説明
addr	リモート・ノードのアドレスです
portno	接続されたサーバーのポート番号です。TCPGWがクライアント・モードで動作する場合はリモート・ノードのポート番号であり、TCPGWがサーバー・モードで動作する場合はリスンしているソケットのポート番号です
type	リモート・ノードと接続されたチャンネルのタイプです。IN_CHANNELかOUT_CHANNELかを表します
fd	リモート・ノードと接続されたソケット番号です

- 戻り値

戻り値	説明
1	当該リモート・ノードとの接続要求を許容します
負数	当該リモート・ノードとの接続を許容しません

- 例

```
int allow_connection(int addr, int portno, int type, int fd)
{
    int i, n, len;
    struct in_addr in;
    char *ipaddr, *endptr;
    char *deny[] = {"10.10.30.1",
                    "121.100.100.*",
                    NULL};

    if (addr == -1)
        return -1;
    in.s_addr = addr;
    ipaddr = inet_ntoa(in);

    for (i = 0; deny[i] != NULL; i++) {
        if ((endptr = strpbrk(deny[i], "*")) != NULL) {
            if (strncmp(deny[i], ipaddr, (endptr - deny[i])) == 0)
                return -1;
        } else {
            if (strcmp(deny[i], ipaddr) == 0)
                return -1;
        }
    }
    return 1;
}
```

3.3.6. allow_connection_ipv6

IPv6プロトコル環境でリモート・ノードがTCPGWに接続を試みたり、あるいはTCPGWがリモート・ノードに接続を試みて接続に成功した場合、remote_connected()コールバック関数が呼び出される前に呼び出されます。当該リモートとの接続を許容するかどうかを指定します。

IPv6環境ではallow_connectionコールバック関数ではなく、allow_connection_ipv6コールバック関数を設定する必要があります。このコールバック関数を設定せずにIPv6環境で使用すると、allow_connection関数が呼び出されますが、ipaddrの値に任意の値が設定され、slogに警告メッセージが記録されます。

- 使用方法

```
#include <arpa/inet.h>

int allow_connection(struct sockaddr *saddr, int portno, int type, int fd)
```

● パラメータ

パラメータ	説明
saddr	リモート・ノードのアドレスおよびポート番号です。IPv4またはIPv6プロトコル環境に適切なsockaddr型の構造体でキャストして使用します
portno	接続されたサーバーのポート番号です。TCPGWがクライアント・モードで動作する場合はリモート・ノードのポート番号であり、TCPGWがサーバー・モードで動作する場合はリスンしているソケットのポート番号です
type	リモート・ノードと接続されたチャンネルのタイプです。IN_CHANNELかOUT_CHANNELかを表します
fd	リモート・ノードと接続されたソケット番号です

● 戻り値

戻り値	説明
1	当該リモート・ノードとの接続要求を許容します
負数	当該リモート・ノードとの接続を許容しません

● 例

```
int
allow_connection_ipv6(struct sockaddr *saddr, int svr_portno, int type, int fd)
{
    char buf[INET6_ADDRSTRLEN];
    char *ipaddr;
    int cli_portno;

    int i, n, len;
    char *endptr;
    char *allow[] = {"10.10.30.1",
                    "121.100.100.*",
                    NULL};

    if (saddr->sa_family == AF_INET6) {
        ipaddr = (char *)inet_ntop(AF_INET6, &(((struct sockaddr_in6
*)saddr)->sin6_addr), buf, sizeof(buf));
        cli_portno = ntohs(((struct sockaddr_in6 *)saddr)->sin6_port);
    } else if (saddr->sa_family == AF_INET) {
        ipaddr = (char *)inet_ntop(AF_INET, &(((struct sockaddr_in
```

```

*)saddr)->sin_addr), buf, sizeof(buf));
    cli_portno = ntohs(((struct sockaddr_in *)saddr)->sin_port);
}
if (ipaddr == NULL)
    ipaddr = "unknown";

for (i = 0; allow[i] != NULL; i++) {
    if ((endptr = strpbrk(allow[i], "*")) != NULL) {
        if (strncmp(allow[i], ipaddr, (endptr - deny[i])) == 0)
            return 1;
    } else {
        if (strcmp(allow[i], ipaddr) == 0)
            return 1;
    }
}
return -1;
}

```

3.3.7. get_msg_length

リモート・ノードから要求や応答が到着し、当該チャンネルでmsg_header_t部分だけをrecvした後(msg_header_sizeまたはcomm_header_sizeで指定した値を読み込む)呼び出す関数で、戻された値の実データを再度読み込みます。

- 使用方法

```
int get_msg_length(msg_header_t *hp)
```

- パラメータ

パラメータ	説明
hp	<p>開発者がcustom.hに定義できるmsg_header_t構造体のポインターです。</p> <p>一般的に当該チャンネルからデータを読み込む場合、決まったヘッダー・サイズのデータを読み込んだ後、ヘッダーに設定されているデータ・サイズの値を取得して、それを基に次の実データを読み込めるようになっています。hpはTCPGWがリモート・ノードから読み込んだヘッダー・データを渡します</p>

- 戻り値

リモート・ノードから実データのサイズを返します。関数の戻り値としてTCPGWは実データをリモート・ノードから読み込みます。ヘッダーを確認して問題がある場合に-2を戻すと、当該チャンネルが終了します。

3.3.8. get_msg_info

リモート・ノードから要求や応答が到着してデータを読み込んだ後、Tmaxサービス・プログラムに再度要求や応答を送信する前に当該データ値を加工したり、情報送信のための様々な情報(uid、len、flags、サービス名など)をTCPGWライブラリーとcustom.cとのインターフェースの役割をするinfoで参照または加工する関数です。

- 使用方法

```
int get_msg_info(msg_header_t *hp, char *data, msg_info_t *info)
```

- パラメータ

パラメータ	説明
hp	リモート・ノードから読み込んだメッセージ・ヘッダー・データのポインターです。 get_msg_lengthで使用した構造体と同じです
data	リモート・ノードから読み込んだデータです
info	TCPGWライブラリー(libtcpgw.a、libtcpgw.so)とcustom.cとのインターフェースの 役割をする構造体です。ユーザーが受信したデータを基にしてinfo構造体の項目 に各種の情報を関数で設定します

- 戻り値

Tmaxサービスに送信するタイプを定義します。TCPGWは値を基にTmaxでどんな処理を行うかを判断します。たとえば、REMOTE_REQUEST値は、リモート・ノードから要求が発生したと判断します。REMOTE_REPLYは、Tmaxサーバーから要求が発生してリモート・ノードから応答が返される場合に返す値です。REMOTE_REPLY_CONTは、応答メッセージが引き続いて返される場合に返す値であり、REMOTE_SENDTOCLIは非要求メッセージの場合に返す値です。

-1を返し、環境設定ファイルのCLOPTセクションに-eオプションを設定すると、当該チャンネルが終了します。

リモート・ノードから応答を受信した場合には、UID値をinfo構造体のuid項目に指定する必要があります。その他の値は状況に合わせて指定してください。

3.3.9. get_channel_num

Tmaxサービスやクライアントから要求したデータをリモートに送信するときに、ユーザーがチャンネルを選択できるようにする関数です。ユーザーは与えられたデータの特性によって送信するチャンネルを指定できます。ただし、ここで指定するのはリモート・ノードと接続されたソケット番号ではなく、単純なチャンネル番号です。TCPGWはユーザーが指定したチャンネルを使用できない場合は、エラーを返します。

- 使用方法


```
int get_channel_num(char *data)
```

- パラメータ

パラメータ	説明
data	リモート・ノードに送信するデータです

- 戻り値

チャンネル番号を戻します。

3.3.10. put_msg_info

リモート・ノードにメッセージを送信する場合に呼び出す関数です。同期型通信の場合は、ユーザーが関数でUIDをメッセージに保存する必要があります。UIDはinfo構造体のuid項目の値を使用してもいいし、またはユーザーが任意のUIDを作成して使用した後、infoのuid項目に入力します。

ユーザーはmsg_header_t構造体の各項目に適切な値を保存する必要があります。構造体はユーザーが任意に設定できる項目であるため、TCPGWではmsg_header_tの構造体項目に値を保存しません。

msg_header_t構造体の長さ以上に設定する場合、dataに影響を与えるので、値を注意して使用する必要があります。

- 使用方法

```
int put_msg_info(msg_header_t *hp, char *data, msg_info_t *info)
```

- パラメータ

パラメータ	説明
hp	リモート・ノードに送信するメッセージのヘッダーです
data	リモート・ノードに送信するデータです
info	リモート・ノードに送信するデータの情報です

- 戻り値

ユーザーは実際にリモート・ノードに送信するデータの全体サイズを戻す必要があります。メッセージ・ヘッダーと実データを足したサイズを戻します。

3.3.11. put_msg_complete

リモート・ノードにメッセージを送信した後呼び出される関数で、データが完全にリモート・ノードに送信されたことを通知するために呼び出されます。

- 使用方法

```
int put_msg_complete(msg_header_t *hp, char *data, msg_info_t *info)
```

- パラメータ

パラメータ	説明
hp	リモート・ノードに送信したメッセージのヘッダーです
data	リモート・ノードに送信したデータです
info	リモート・ノードに送信したデータの情報です

3.3.12. get_service_name

Tmaxでリモート・ノードに要求を送信するときに、要求を送信するサーバーと結果を受信するサーバーが異なる非ブロック型や非同期型のTCPGWを構成する場合、tpreply()またはtpacall()を実行するサービスの名前をエラーコードによって設定します。

- 使用方法

```
int get_service_name(char *header, int err, char *svc)
```

- パラメータ

パラメータ	説明
header	[-H]または[-h]オプションで設定したTCPGWで保存しているユーザー・ヘッダーのポインターです
err	エラーコードです
svc	tpreply()またはtpacall()を受信するサービスの名前を設定します

3.3.13. prepare_shutdown

TCPGWが終了する直前に呼び出される関数で、一般的にinit_remote_info()関数で作成した共有メモリーを解除します。

- 使用方法

```
int prepare_shutdown(int code)
```

- パラメータ

パラメータ	説明
code	終了コードで、現在は使用されていません

3.3.14. set_service_timeout

サービス・タイムアウトが発生したときに呼び出される関数です。サービス・タイムアウトが発生した後に必要な作業があれば、この関数で実装します。

- 使用方法

```
int set_service_time_out(int uid, char *header)
```

- パラメータ

パラメータ	説明
uid	サービス・タイムアウトが発生したトランザクションのユーザーIDです
header	サービス・タイムアウトが発生したトランザクションのヘッダーです

3.3.15. chk_end_msg

リモート・ノードからデータを受信するとき、データの最後を表す特定の文字またはビット・ストリームが存在する場合にユーザーが呼び出せる関数です。

- 使用方法

```
int chk_end_msg(int len, char *data)
```

- パラメータ

パラメータ	説明
len	リモート・ノードから読み込んだデータのサイズです
data	リモート・ノードから読み込んだデータです

3.3.16. inmsg_recovery

ノードから要求をtpacall(..., TPBLOCK)で処理した場合、サーバーが起動されていないとエラーが返されますが、そのようなときに呼び出される関数です。ユーザーは関数内で適切に新しいデータを作成し、データのサイズを返します。

- 使用方法

```
int inmsg_recovery(char *data, msg_info_t *info)
```

- パラメータ

パラメータ	説明
data	リモート・ノードから読み込んだデータです
info	リモート・ノードから読み込んだデータの情報で、意味のある値は以下のとおりです <ul style="list-style-type: none">– info→svc : tpacall()したサービス名– info→len : tpacall()したデータ長– info→err : エラーが発生した場合– tperno info→uid : 以前get_msg_info()したときに作成されたUID値

- 戻り値

ユーザーは新しいデータの長さを返す必要があります。

戻り値	説明
正数	<ul style="list-style-type: none">– info→svcに値が存在する場合 : 当該サービスにtpacall(..., TPNOREPLY)します– その他の場合 : リモート・ノードに応答を送ります
負数	データを捨てます

3.3.17. outmsg_recovery

リモート・ノードに要求を送信した時にエラーが発生した場合に呼び出される関数です。ユーザーは関数内で適切に新しいデータを作成し、希望するサービス名およびUIDなどを設定し、データのサイズを返します。

- 使用方法

```
int inmsg_recovery(char *data, msg_info_t *info)
```

- パラメータ

パラメータ	説明
data	リモート・ノードから読み込んだデータです
info	リモート・ノードから読み込んだデータの情報で、意味のある値は以下のとおりです <ul style="list-style-type: none"> – info→svc : tpacall()したサービス名 – info→len : tpacall()したデータ長 – info→err : エラーが発生した場合 – tperrno info→uid : 以前get_msg_info()したとき作成されたUID値

- 戻り値

戻り値	説明
正数	<ul style="list-style-type: none"> – info→svcに値が存在する場合 : 当該サービスにtpacall(..., TPNOREPLY)します – info→msgtypeが1000未満の場合 : データを呼び出し元に戻します – info→msgtypeが1000以上の場合 : データを捨てます

3.3.18. get_extmsg_info

get_msg_infoと基本的に機能は同じです。ただし、データ・バッファをダブル・ポインター型で渡して、ユーザーがバッファのメモリを再割り当て(realloc)することができます。この関数はget_msg_infoと同時に使用できません。この関数を使用するには、TCPGWをコンパイル時に-D_TCPGW_USE_EXTMSGフラグを設定する必要があります。

- 使用方法

```
int get_extmsg_info(msg_header_t *hp, char **data, int asize, msg_info_t *info)
```

- パラメータ

パラメータ	説明
hp	リモート・ノードから読み込んだメッセージ・ヘッダー・データのポインターです。get_msg_length()で使用した構造体と同じです

パラメータ	説明
data	リモート・ノードから読み込んだデータ・バッファのアドレス値です
asize	リモート・ノードから読み込んだデータ・バッファに割り当てられたメモリのサイズです
info	TCPGWライブラリー(libtcpgw.a、libtcpgw.so)とcustom.cとのインターフェースの役割をする構造体です。ユーザーが受信したデータを基にしてinfo構造体の項目に各種の情報を設定します

- 戻り値

Tmaxサービスに送信するタイプを定義します。TCPGWは値を基にTmaxでどんな処理を行うかを判断します。たとえば、REMOTE_REQUEST値はリモート・ノードから要求が発生したと判断します。REMOTE_REPLYは、Tmaxサービスから要求が発生してリモート・ノードから応答が返される場合に戻す値です。REMOTE_REPLY_CONTは、応答メッセージが引き続いて返される場合に戻す値であり、REMOTE_SENDOCLIは非要求メッセージの場合に戻す値です。リモート・ノードから応答を受信した場合には、UID値をinfo構造体のuid項目に指定する必要があります。その他の値は状況に合わせて指定してください。

3.3.19. put_extmsg_info

put_msg_infoと基本的に機能は同じです。ただし、データ・バッファをダブル・ポインター型で渡して、ユーザーがバッファのメモリを再割り当て(realloc)することができます。この関数はput_msg_infoと同時に使用できません。この関数を使用するには、TCPGWをコンパイル時に-D_TCPGW_USE_EXTMSGフラグを設定する必要があります。

- 使用方法

```
int put_extmsg_info(msg_header_t *hp, char **data, int asize, msg_info_t *info)
```

- パラメータ

パラメータ	説明
hp	リモート・ノードに送信するメッセージのヘッダーです
data	リモート・ノードに送信するデータ・バッファのアドレス値です
asize	リモート・ノードに送信するデータ・バッファに割り当てられたメモリ・サイズです
info	リモート・ノードに送信するデータの情報です

- 戻り値

ユーザーは関数で実際にリモート・ノードに送信するデータの全体サイズを戻す必要があります。メッセージ・ヘッダーと実データを足したサイズを戻します。

3.3.20. set_ping_msg

チャンネル障害を監視するために送信するメッセージやメッセージの周期、タイムアウトなどを設定できるユーザー関数で、[?x]オプションを指定する場合は必ず設定する必要があります。この関数はTCPGWプログラムが起動するときに一度呼び出されます。

- 使用方法

```
int set_ping_msg(msg_header_t *hp, int *interval, int *binterval, int *timeout,  
  
int *mode)
```

- パラメータ

パラメータ	説明
hp	チャンネル障害の監視のために周期的に送信されるメッセージです。ユーザーはこのメッセージを設定する必要があります
interval	チャンネル障害の監視周期です。 0を指定する場合、チャンネル障害検知機能は無効になります(単位: 秒)
binterval	バックアップ・チャンネル・モードのときにメイン・チャンネルの状態をチェックする周期です。 0を指定する場合、メイン・チャンネルの状態チェック機能は無効になります(単位: 秒)
timeout	チャンネル障害の監視タイムアウトで、時間内に応答がなければ、チャンネルの接続は切断されます(単位: 秒)。 実際にチャンネルの接続が終了される時点は、interval値に従ってpingメッセージを送信したとき、reset_ping_msg()が呼び出される前です。0を指定する場合は、pingメッセージだけ送り、pongメッセージは気にしません(半二重通信(Half Duplex)障害の検知)
mode	障害を監視するチャンネルの種類を指定します – 0 : OUTチャンネル – 1 : INチャンネル – 2 : OUT/IN両チャンネル

- 戻り値

エラーが発生したときは負数値を返す必要があり、その場合、監視機能は無効になります。

3.3.21. chk_pong_msg

リモート・サーバーから受信したメッセージがチャンネル障害検知のための応答メッセージであるかどうかを確認する関数です。

- 使用方法

```
int chk_pong_msg(msg_header_t *hp)
```

- パラメータ

パラメータ	説明
hp	リモート・サーバーから受信したメッセージです

- 戻り値

戻り値	説明
正数	リモート・サーバーから受信したメッセージが正常なチャンネル障害監視の応答メッセージである場合です
0	リモート・サーバーから受信したメッセージがチャンネル障害監視の応答メッセージではなく、一般的なメッセージである場合です。当該メッセージはget_msg_info関数で処理されます
負数	リモート・サーバーから受信したメッセージが異常なチャンネル障害監視の応答メッセージである場合です。当該チャンネルを終了させます

3.3.22. set_extping_msg

チャンネルの障害監視のために送信するメッセージやメッセージの周期、タイムアウトなどを設定できるユーザー関数で、[?x]オプションを指定する場合は必ず設定する必要があります。この関数はTCPGWプログラムが起動するときに一度呼び出されます。

- 使用方法

```
int set_extping_msg(msg_header_t *hp, char *data, int len, int *interval,  
int *binterval, int *timeout, int *mode)
```

- パラメータ

パラメータ	説明
hp	チャンネル障害の監視のために周期的に送信するメッセージです。ユーザーはこのメッセージを設定する必要があります
data	チャンネル障害を監視するために周期的に送信するメッセージの後に一緒に送信されるメッセージ本体です。必要な場合に設定します
len	チャンネル障害の監視メッセージ本体の最大長です。[-b]オプションで最大長を指定します。指定しない場合、サイズは0になります
interval	チャンネル障害の監視周期です。 0を指定する場合、チャンネルの障害監視機能は無効になります(単位: 秒)
binterval	バックアップ・チャンネル・モードのときにメイン・チャンネルの状態をチェックする周期です。 0を指定する場合、メイン・チャンネルの状態チェック機能は無効になります(単位: 秒)
timeout	チャンネル障害の監視タイムアウトで、時間内に応答がなければ、チャンネルの接続は切断されます(単位: 秒)。 0を指定する場合は、pingメッセージだけ送り、pongメッセージは気にしません(半二重通信(Half Duplex)障害の検知)
mode	障害を監視するチャンネルの種類を指定します <ul style="list-style-type: none"> – 0 : OUTチャンネル – 1 : INチャンネル – 2 : OUT/IN両チャンネル

- 戻り値

戻り値	説明
0	関数の実行に成功した場合です
負数	関数の実行に失敗した場合です

- 例

```
int set_extping_msg(msg_header_t *hp, char *data, int len, int *interval,
                   int *binterval, int *timeout, int *mode)
{
    msg_body_t *body;
```

```

body = (msg_body_t *)data;
memset(body->data, 0x00, 52);

memcpy(body->data, "tmax50", 7);
body->data[51] = 0;
printf("set_extping_msg : data = %s\n", body->data);

hp->len = 7;
*interval = 10;
*binterval = 20;
*timeout = 100;
*mode = 0; /* OUTBOUND CHANNEL */

return 1;
}

```

3.3.23. chk_extpong_msg

リモート・サーバーから受信したメッセージがチャンネル障害検知のための応答メッセージであるかどうかを確認する関数です。

- 使用方法

```
int chk_extpong_msg(msg_header_t *hp, char *data, int len)
```

- パラメータ

パラメータ	説明
hp	チャンネルの障害が検知されたときに、リモート・サーバーから受信したメッセージのヘッダーです
data	チャンネルの障害が検知されたときに、リモート・サーバーから受信したメッセージの本体です
len	受信したメッセージ本体の長さです

- 戻り値

戻り値	説明
正数	リモート・サーバーから受信したメッセージが正常なチャンネル障害監視の応答メッセージである場合です
0	リモート・サーバーから受信したメッセージがチャンネル障害監視の応答メッセージではなく、一般的なメッセージである場合です。当該メッセージはget_msg_info関数で処理されます

戻り値	説明
負数	リモート・サーバーから受信したメッセージが異常なチャンネル障害監視の応答メッセージである場合です。当該チャンネルを終了させます

- 例

```
int chk_extpong_msg(msg_header_t *hp, char *data, int len)
{
    msg_body_t *body;
    char data2[15];

    body = (msg_body_t *)data;
    printf("chk_extpong_msg : data = %s\n", body->data);

    if (strcmp(body->data, "ping_reply")
        return 1;
    return 0;
}
```

3.3.24. reset_ping_msg

チャンネル障害検知(TCP/IP ping)メッセージの送信の可否を決定し、送信メッセージを再設定するために周期的に呼び出される関数です。

set_ping_msg関数で設定したinterval値に従ってpingメッセージをリモート・サーバーに送信する前に呼び出される関数で、送信メッセージを確認し、必要場合は修正することができます。また戻り値によってpingメッセージをリモート・サーバーに送信するかどうかを決定します。

- 使用方法

```
int reset_ping_msg(msg_header_t *hp)
```

- パラメータ

パラメータ	説明
hp	チャンネル障害検知のために送信するメッセージのヘッダーです。デフォルト値としてset_ping_msg関数でユーザーが定義したヘッダーが設定されます

- 戻り値

戻り値	説明
正数	pingメッセージをリモート・サーバーに送信します

戻り値	説明
負数	pingメッセージをリモート・サーバーに送信しません

- 例

```
int reset_ping_msg(msg_header_t *hp)
{
    hp->len = 0;
    hp->msgtype = HEALTH_CHECK;
    return 1;
}
```

3.3.25. reset_extping_msg

チャンネル障害検知(TCP/IP ping)メッセージの送信の可否を決定し、送信メッセージを再設定するために周期的に呼び出される関数です。

set_ping_msg関数で設定したinterval値に従ってpingメッセージをリモート・サーバーに送信する前に呼び出される関数で、送信メッセージを確認し、必要な場合は修正することができます。また戻り値によってpingメッセージをリモート・サーバーに送信するかどうかを決定します。

- 使用方法

```
int reset_extping_msg(msg_header_t *hp, char *data, int len)
```

- パラメータ

パラメータ	説明
hp	チャンネル障害の検知のために送信するメッセージのヘッダーです。デフォルト値としてset_ping_msg関数でユーザーが定義したヘッダーが設定されます
data	再設定するメッセージ本体のポインターです。バッファの最大サイズはlenに設定されたサイズです。実際に送信するバッファのサイズは戻り値に指定します
len	使用可能なメッセージ本体の最大サイズです

- 戻り値

戻り値	説明
正数	pingメッセージをリモート・サーバーに送信します。指定されたサイズのメッセージ本体を送信します
0	pingメッセージをリモート・サーバーに送信します。メッセージ本体は送信しません
負数	pingメッセージをリモート・サーバーに送信しません

- 例

```
int reset_extping_msg(msg_header_t *hp, char *data, int len)
{
    int body_len;
    char *message = "reset_msg";

    body_len = min(len, strlen(message));
    strncpy(data, message, (body_len - 1));
    data[(body_len - 1)] = '\0';
    hp->len = body_len;
    printf("reset_extping_msg : data = %s\n", data);

    return body_len;
}
```

3.3.26. set_error_msg

TCPGWを介して行うリモート・ノードとのトランザクションの途中にタイムアウトまたはネットワーク切断などのエラーが発生したときに自動で呼び出される関数です。当該関数内でユーザーはユーザー・ヘッダーまたはユーザー・データを修正できます。

- 使用方法

```
int set_error_msg(msg_header_t *hp, int err, char *data, int len)
```

- パラメータ

パラメータ	説明
hp	エラーの発生時に送信されたメッセージのヘッダーで、ユーザーが修正できます
data	エラーの発生時に送信されたデータで、ユーザーが修正できます
len	メッセージ本体の長さです

- 戻り値

戻り値	説明
正数	戻された長さのデータを送信します。ただし、この場合、CLOPT="-I"オプションを設定した場合にのみ適用されます
0	ユーザー・ヘッダー部分まで送信します
負数	当該メッセージをCLHIに送信しません

- エラー

エラーコード	説明
[TPECLOSE]	リモート・ノードにサービスを要求した後、リモート・ノードとの接続が切断された場合に発生します
[TPENOENT]	[-E]オプションを使用する場合、サービス要求ごとにリモート・ノードと接続してデータを送信しますが、同時に呼び出した数が[-n]オプションで指定したチャンネルの数を超えた場合に発生します
[TPENOREADY]	リモート・ノードとの接続が切断され使用できるチャンネルがない場合に発生します
[TPEOS]	TCPGW内部でメモリーを確保できない場合に発生します
[TPEPROTO]	tpforwardでTCPGWを呼び出したとき、TCPGWが同期型モードではなく非同期型モードである場合に発生します。tpforwardとtprelayの方式は必ず同期型である必要があります
[TPESVCERR]	put_msg_infoで0または負数を返す場合に発生します
[TPESYSTEM]	TCPGWでコード変換を使用するとき、要求したデータのmapファイルがロードされない場合に発生します。またはコード変換エラーが発生してユーザー関数のput_msg_infoで負数を返す場合、リモート・ノードにデータを送信したときにエラーが発生した場合です
[TPETIME]	リモート・ノードにサービスを要求し、指定した時間内に応答がない場合に発生します

- 例

```
int set_error_msg(msg_header_t *hp, int err, char *data, int len)
{
    msg_body_t * body;
    body = (msg_body_t *)data;
    strcpy(body->data, "changed hello data");
    /* エラー・メッセージにはデータは含まれないため、
       データまで渡すためには、-Iオプションを使用する必要があります。
       このオプションを使用しない場合、ユーザー・ヘッダー部分までCLHに渡されます。 */
    /* ユーザー・ヘッダーがない場合、hpとdataの値が同じであることがあります。 */
    strcpy(hp->retsvcname, "RECVSVC_CHANGE");
    return len;
}
```

3.3.27. get_msg_security

get_msg_info()またはget_extmsg_info()関数が実行された後に呼び出される関数です。ユーザー・データを加工する必要がある場合に使用します。get_extmsg_info()関数はこの関数を代替することができます。

- 使用方法

```
int get_msg_security(char **data, int asize, int len)
```

- パラメータ

パラメータ	説明
data	リモート・ノードから読み込んだデータ・バッファのアドレス値です。バッファのサイズが不足する場合は、realloc()によりバッファを拡張することを許容します。realloc()を呼び出してバッファのポインターが変更された場合には、このアドレス値に変更されたポインター・アドレス値を保存する必要があります
asize	リモート・ノードから読み込んだデータ・バッファに割り当てられたメモリのサイズです
len	メッセージ本体の長さです

- 戻り値

戻り値	説明
正数	加工されたユーザー・データの全体長を返します
0	ユーザー・データの変更事項を反映しません

- 例

```
int get_msg_security(char **data, int asize, int len)
{
    ...
    new_size = len * 2;
    if (new_size > asize) {
        tmpbuf = (char *)realloc(*data, new_size);
        if (tmpbuf == NULL) {
            /* error processing */
        }
        *data = tmpbuf;
    }
    ...
    return new_size;
}
```

3.3.28. put_msg_security

put_msg_info()またはput_extmsg_info()関数が実行される前に呼び出される関数です。ユーザー・データを加工する必要がある場合に使用します。put_extmsg_info()関数はこの関数を代替することができます。

- 使用方法

```
int put_msg_security(char **data, int asize, int len)
```

- パラメータ

パラメータ	説明
data	リモート・ノードに送信するデータ・バッファのアドレス値です。バッファのサイズが不足する場合は、realloc()によりバッファを拡張することを許容します。realloc()を呼び出してバッファのポインターが変更された場合には、このアドレス値に変更されたポインター・アドレス値を保存する必要があります
asize	リモート・ノードに送信するデータ・バッファに割り当てられたメモリのサイズです
len	メッセージ本体の長さです

- 戻り値

戻り値	説明
正数	加工されたユーザー・データの全体長を返します
0	ユーザー・データの変更事項を反映しません

- 例

```
int put_msg_security(char **data, int asize, int len)
{
    ...
    new_size = len * 2;
    if (new_size > asize) {
        tmpbuf = (char *)realloc(*data, new_size);
        if (tmpbuf == NULL) {
            /* error processing */
        }
        *data = tmpbuf;
    }
    ...
    return new_size;
}
```

3.4. register.c

ユーザー関数の登録ファイルです。ユーザーが登録した関数のみゲートウェイ・ライブラリーで呼び出されます。ゲートウェイをコンパイルするとき、必ずregister.cを含めます。使用しない関数はNULL登録します。

下記は、register.cの例です。


```

/* ----- tcpgw register.c ----- */
#include <stdio.h>
#include <arpa/inet.h>
#include "custom.h"

extern int _tcpgw_regfn_init();
extern int _tcpgw_regfn(int type, void *regFn);

extern int init_remote_info(char *myname, int mynumber, int num_channel, int key);
extern int prepare_shutdown(int code);
extern int remote_connected(int index, int addr, int portno, int type, int fd);
extern int remote_connected_ipv6(int index, struct sockaddr *saddr, int portno, int
type, int fd);
extern int remote_closed(int index, int type);
extern int get_msg_length(msg_header_t * hp);
extern int get_msg_info(msg_header_t * hp, char *data, msg_info_t * info);
extern int put_msg_info(msg_header_t * hp, char *data, msg_info_t * info);
extern int put_msg_complete(msg_header_t * hp, char *data, msg_info_t * info);
extern int get_channel_num(char *data);
extern int get_service_name(char *header, int err, char *svc);
extern int set_service_timeout(int uid, char *header);
extern int get_msg_security(char **data, int asize, int len);
extern int put_msg_security(char **data, int asize, int len);
extern int chk_end_msg(int len, char *data);
extern int get_extmsg_info(msg_header_t * hp, char **data, int asize, msg_info_t *
info);
extern int put_extmsg_info(msg_header_t * hp, char **data, int asize, msg_info_t *
info);
extern int inmsg_recovery(char *data, msg_info_t * info);
extern int outmsg_recovery(char *data, msg_info_t * info);
extern int set_ping_msg(msg_header_t * hp, int *interval, int *bintrval, int *timeout,
int *mode);
extern int chk_pong_msg(msg_header_t * hp);
extern int set_extping_msg(msg_header_t * hp, char *data, int len, int *interval, int
*bintrval, int *timeout, int *mode);
extern int chk_extpong_msg(msg_header_t * hp, char *data, int len);
extern int reset_ping_msg(msg_header_t * hp);
extern int reset_extping_msg(msg_header_t * hp, char *data, int len);
extern int set_error_msg(msg_header_t * hp, int err, char *data, int len);
extern int allow_connection(int addr, int portno, int type, int fd);
extern int allow_connection_ipv6(struct sockaddr *saddr, int portno, int type, int fd);

/*****
* int
* _register_custom()
*
* returns no used
* [function number]
* 1. init_remote_info
* 2. prepare_shutdown
*****/

```

```

*      3.  remote_connected
*      4.  remote_closed
*      5.  get_msg_length
*      6.  get_msg_info
*      7.  put_msg_info
*      8.  put_msg_complete
*      9.  get_channel_num
*     10.  get_service_name
*     11.  set_service_timeout
*     12.  get_msg_security
*     13.  put_msg_security
*     14.  chk_end_msg
*     15.  get_extmsg_info
*     16.  put_extmsg_info
*     17.  inmsg_recovery
*     18.  outmsg_recovery
*     19.  set_ping_msg
*     20.  chk_pong_msg
*     21.  set_extping_msg
*     22.  chk_extpong_msg
*     23.  reset_ping_msg
*     24.  reset_extping_msg
*     25.  set_error_msg
*     26.  allow_connection
*     27.  allow_connection_ipv6
*     28.  remote_connected_ipv6
*****/

int
_register_custom()
{
    _tcpgw_regfn_init();
    _tcpgw_regfn(1, init_remote_info);
    _tcpgw_regfn(2, prepare_shutdown);
    _tcpgw_regfn(3, remote_connected);
    _tcpgw_regfn(4, remote_closed);
    _tcpgw_regfn(5, get_msg_length);

#ifdef _TCPGW_USE_EXTMSG
    _tcpgw_regfn(15, get_extmsg_info);
    _tcpgw_regfn(16, put_extmsg_info);
#else
    _tcpgw_regfn(6, get_msg_info);
    _tcpgw_regfn(7, put_msg_info);
#endif
    _tcpgw_regfn(8, put_msg_complete);
    _tcpgw_regfn(9, get_channel_num);
    _tcpgw_regfn(10, get_service_name);
    _tcpgw_regfn(11, set_service_timeout);

```

```

#if defined(_TCPGW_USE_EXTPING)
    _tcpgw_regfn(21, set_extping_msg);
    _tcpgw_regfn(22, chk_extpong_msg);
#else
    _tcpgw_regfn(19, set_ping_msg);
    _tcpgw_regfn(20, chk_pong_msg);
#endif

#if defined(_TCPGW_VERSION_OLD)
#elif defined(_TCPGW_VERSION_1)
    _tcpgw_regfn(12, get_msg_security);
    _tcpgw_regfn(14, chk_end_msg);
#elif defined(_TCPGW_VERSION_2)
    _tcpgw_regfn(12, get_msg_security);
    _tcpgw_regfn(14, chk_end_msg);
    _tcpgw_regfn(17, inmsg_recovery);
    _tcpgw_regfn(18, outmsg_recovery);
#else
    _tcpgw_regfn(14, chk_end_msg);
    _tcpgw_regfn(17, inmsg_recovery);
    _tcpgw_regfn(18, outmsg_recovery);
#endif

    return 1;
}

```


第4章 例

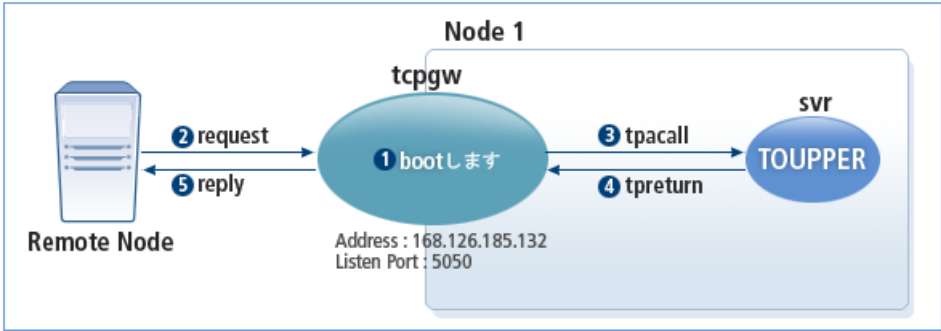
本章では、TCPGWのサービス・タイプ別の例について説明します。

4.1. アウトバウンドTCPGW

Tmaxのブート時にTCPGWが起動され、リモート・ノードの要求が受信されたらユーザーが指定したサービス呼び出した後、リモート・ノードに処理結果を再転送する例です。TCPGWのクライアントおよびサーバー方式を問わず、リモート・ノードの状況に合わせてcustom.cを修正してTCPGWを構成します。

下記は、アウトバウンドTCPGWプログラムの動作です。

[図 4.1] アウトバウンドTCPGWプログラムの動作



プログラムの構成は下記のとおりです。

区分	ファイル名
環境ファイル	tcpgw.m
TCPGW	custom.h, custom.c
リモート・ノード	rclient.c, custom.h
サーバー	svr.c

4.1.1. 環境ファイル

<tcpgw.m>

```
*DOMAIN
res          SHMKEY = 88000 ,
              MINCLH = 1 ,
```

```

        MAXCLH = 1,
        TPORTNO = 8888

*NODE
node1      TMAXDIR = "/home/tmaxqas/tmax",
           APPDIR  = "/home/tmaxqas/tmax/appbin",
           PATHDIR = "/home/tmaxqas/tmax/path",
           TLOGDIR = "/home/tmaxqas/tmax/log/tlog",
           ULOGDIR = "/home/tmaxqas/tmax/log/ulog",
           SLOGDIR = "/home/tmaxqas/tmax/log/slog",

*SVRGROUP
svg1       NODENAME = node1

*SERVER
tcpgw      SVGNAME = svg1,
           MIN = 1, MAX = 1,
           CPC = 5,
           SVRTYPE = CUSTOM_GATEWAY,
           CLOPT = "-- -P 5050 -N 2 -d 0"

svr        SVGNAME = svg1,
           MIN = 1, MAX = 1

*SERVICE
TOUPPER    SVRNAME = svr

```

4.1.2. TCGW

<custom.h>

```

#ifndef _CUSTOM_H_
#define _CUSTOM_H_

/* ----- */
/*          Fixed structures and macros          */

#define MSG_MAGIC          "Tmax"
#define REMOTE_REQUEST    0
#define REMOTE_REPLY      1
#define SVC_NAME_LENGTH   20

/* このmsg_info_tは、開発者が再定義してはいけない構造体です。*/
typedef struct msg_info {
    char  svc[SVC_NAME_LENGTH];
    int   err;

```

```

    int    len;
    int    uid;
    int    flags;
    int    msgtype;
    int    channel_id;
} msg_info_t;

/* ----- */
/*      Modifiable structures and macros      */
/* このmsg_header_tとmsg_body_tは、開発者が再定義できる構造体です。*/
typedef struct msg_header {
    int    len;
} msg_header_t;

typedef struct msg_body {
    char    name[16];
    char    data[100];
} msg_body_t;

#endif

```

<custom.c>

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <sys/types.h>
#include <sys/timeb.h>
#include "custom.h"

/* msg_header_sizeとcomm_header_sizeは、TCPGWライブラリー内で使用されるため、
定義する必要があります。*/
int    msg_header_size    = sizeof(msg_header_t);
int    comm_header_size = 0;

/* 下記の関数が不要な場合は、内部ロジックを実装する必要はありません。*/
/* しかし、TCPGWライブラリー内で使用されるため、定義する必要があります。*/
int init_remote_info(char *myname, int mynumber, int num_channel, int key)
{
    return 1;
}

int remote_connected(int index, int addr, int type, int fd)
{
    return 1;
}

```

```

int get_msg_length(msg_header_t *hp)
{
    if (hp == NULL || hp->len <= 0)
        return -1;
    /* 実データの長さを返します。*/
    /* こちらで返す値でリモート・ノードからのデータを読み取ります。*/
    return ntohl(hp->len);
}

int get_msg_info(msg_header_t *hp, char *data, msg_info_t *info)
{
    msg_body_t *body;

    if ((info == NULL) || (hp == NULL) || (data == NULL))
        return -1;

    body = (msg_body_t *)data;

    info->len = ntohl(hp->len);
    info->err = 0;
    info->flags = 0;

    memset(info->svc, 0x00, SVC_NAME_LENGTH);
    strncpy(info->svc, body->name, 8);

    /* リモート・ノードからの要求なので、REMOTE_REQUESTを返します。 */
    return REMOTE_REQUEST;
}

int get_service_name(char *header, int err, char *svc)
{
    return -1;
}

int put_msg_info(msg_header_t *hp, char *data, msg_info_t *info)
{
    msg_body_t *body;

    if ((info == NULL) || (hp == NULL) || (data == NULL))
        return -1;

    hp->len = htonl(info->len);
    body = (msg_body_t *)data;

    /* body->nameを利用してエラー有無を転送 */
    if (info->err) /* error */

```



```

        strcpy(body->name, "Fail");
    else
        strcpy(body->name, "Success");

    /* リモート・ノードに、要求に対する結果を転送するためのデータ長を返します。 */
    return (info->len + sizeof(msg_header_t));
}

int get_channel_num(char *data)
{
    return -1;
}

int put_msg_complete(char *hp, char *data, msg_info_t *info)
{
    return 1;
}

int remote_closed(int index, int type)
{
    return 1;
}

int prepare_shutdown(int code)
{
    return 1;
}

int set_service_timeout(int uid, char *header)
{
    return 1;
}

int chk_end_msg(int len, char *data)
{
    return len;
}

int outmsg_recovery(char *data, msg_info_t *info)
{
    return -1;
}

int inmsg_recovery(char *data, msg_info_t *info)
{
    return -1;
}

```

<Makefile>

```
#このMakefileはIBM 32bit用です。
#TARGETは、Tmax環境ファイルで定義したサーバー名と同じである必要があります。
TARGET    = tcpgw
APOBJS    = $(TARGET).o

#TCPGWを作成するため、libtcpgw.aまたはlibtcpgw.soをリンクさせます。
LIBS      = -ltcpgw -ltmaxgw
OBSJS     = custom.o register.o

CFLAGS    = -q32 -O -I$(TMAXDIR) -D_DBG
LDFLAGS   = -brtl

APPDIR    = $(TMAXDIR)/appbin
LIBDIR    = $(TMAXDIR)/lib

.SUFFIXES : .c

.c.o:
    $(CC) $(CFLAGS) $(LDFLAGS) -c $<

$(TARGET): $(OBSJS)
    $(CC) $(CFLAGS) $(LDFLAGS) -L$(LIBDIR) -o $(TARGET) $(OBSJS) $(LIBS)
    mv $(TARGET) $(APPDIR)/.
    rm -f $(OBSJS)

$(APOBJS): $(TARGET).c
    $(CC) $(CFLAGS) $(LDFLAGS) -c $(TARGET).c
```

4.1.3. リモート・ノード

<rclient.c>

下記の例題プログラムは、テストのためのリモート・ノードのクライアント・プログラムです。テストを行うためにTCPGWが起動された後、プログラムを起動します。

```
#include <stdio.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include "custom.h"

#define SERV_ADDR  "168.126.185.132"
#define SERV_PORT  5050
```

```

int main(int argc, char *argv[])
{
    int    n, i, fd, ilen, olen;
    msg_header_t *header;
    msg_body_t  *body, *result;

    /* ① TCPGWに接続します。 */
    if ( (fd = network_connect()) < 0 ) {
        perror("network connect error:");
        return -1;
    }

    /* ② 構文を構成します。 */
    /* Message Header */
    header = (msg_header_t *)malloc(sizeof(msg_header_t));
    memset(header, 0x00, sizeof(msg_header_t));
    header->len = sizeof(msg_body_t);

    /* メッセージ本体 */
    body = (msg_body_t *)malloc(sizeof(msg_body_t));
    memset(body, 0x00, sizeof(msg_body_t));
    strncpy(body->name, "TOUPPER", 16);

    for (i = 0; i < 1; i++) {
        /* ③ TCPGWにデータを送信します。 */
        if ( (n = send(fd, (char *)header, sizeof(msg_header_t), 0)) < 0 ) {
            perror("network write error:");
            close(fd);
            return -1;
        }
        printf("Msg header send !\n");

        /* Message Body */
        strncpy(body->data, argv[1], 100);

        if ( (n = send(fd, (char *)body, sizeof(msg_body_t), 0)) <= 0 ) {
            perror("network write error:");
            close(fd);
            return -1;
        }

        /* ④ TCPGWでデータを受信します。 */
        if ( (n = recv(fd, (char *)&olen, 4, 0)) < 0 ) {
            perror("network read error:");
            close(fd);
            return -1;
        }
        result = (msg_body_t *)malloc(olen);
    }
}

```

```

        memset((char *)result, 0x00, olen);
        if ( (n = recv(fd, (char *)result, olen, 0)) <= 0 ) {
            perror("network read error:");
            close(fd);
            return -1;
        }
    }
    /* ⑤ TCPGWとの接続を切断します。*/
    close(fd);
    return 1;
}

int network_connect()
{
    struct sockaddr_in serv_addr;
    int fd;

    memset((char *)&serv_addr, 0, sizeof(serv_addr));
    serv_addr.sin_family      = AF_INET;
    serv_addr.sin_addr.s_addr = inet_addr(SERV_ADDR);
    serv_addr.sin_port        = htons((unsigned short)SERV_PORT);

    if ((fd = socket(AF_INET, SOCK_STREAM, 0)) < 0) {
        return -1;
    }
    if (connect(fd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) >= 0)
        return fd;

    close(fd);
    return -1;
}

```

4.1.4. サーバー

<SVR.C>

```

#include <stdio.h>
#include <string.h>
#include <usrinc/atmi.h>

TOUPPER(TPSVCINFO *msg)
{
    int i;

    printf("TOUPPER service is started!\n");
    printf("INPUT : len=%d, data='%s'\n", msg->len, msg->data);
}

```

```

for (i = 0; i < msg->len; i++)
    msg->data[i] = toupper(msg->data[i]);

printf("OUTPUT: len=%d, data='%s'\n", strlen(msg->data), msg->data);

tpreturn(TPSUCCESS,0,(char *)msg->data, msg->len, 0);
}

```

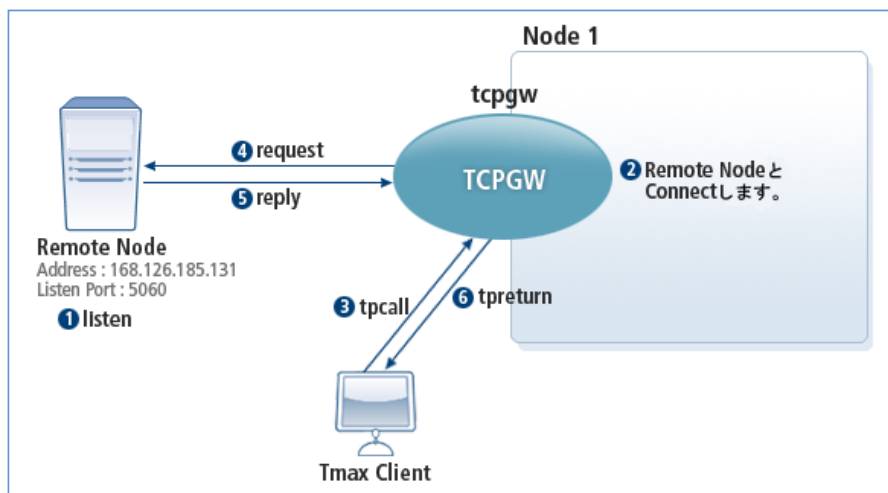
4.2. 同期インバウンドTCPGW

Tmaxクライアントで(またはTmaxサービスで)TCPGWのサービスをtpcallすると、TCPGWはリモート・ノードにデータを転送します。リモート・ノードで処理結果が受信されたら、該当のクライアントに結果を返します。

Tmaxクライアントでリモート・ノードにサービスを要求する際の重要点は、UIDを設定することです。UIDはTCPGWライブラリ内部で指定する値(info->uid)を、put_msg_infoで構文内容に保存するか、またはユーザーがUIDを作成して構文内容に保存した後、UID値をuid項目に書き込みます。このようにUIDを構文に保存し、リモート・ノードに要求を送信したら、リモート・ノードでは当該UIDを変更せず、そのまま返す必要があります。TCPGWは応答を受信した後、get_msg_info関数を呼び出してUID値を取得し、TCPGWの呼び出し元を判断して応答を返します。

下記の例は、同期インバウンドTCPGWプログラムの動作です。クライアントで起動されるため、Tmax環境ファイルのCLOPTIに[-r]、[-p]、[-n]オプションを設定し、リモート・ノードの状況に合わせてcustom.cを修正し、TCPGWを構成します。

[図 4.2] 同期インバウンドTCPGWプログラムの動作



プログラムの構成は下記のとおりです。

区分	ファイル名
環境ファイル	tcpgw.m

区分	ファイル名
TCPGW	custom.h, custom.c
リモート・ノード	rserver.c, network.c
クライアント	cli_tcpgw.c

4.2.1. 環境ファイル

<tcpgw.m>

```

*DOMAIN
res          SHMKEY = 88000,
              MINCLH = 1,
              MAXCLH = 1,
              TPORTNO = 8888

*NODE
node1        TMAXDIR = "/home/tmaxqas/tmax",
              APPDIR  = "/home/tmaxqas/tmax/appbin",
              PATHDIR = "/home/tmaxqas/tmax/path",
              TLOGDIR = "/home/tmaxqas/tmax/log/tlog",
              ULOGDIR = "/home/tmaxqas/tmax/log/ulog",
              SLOGDIR = "/home/tmaxqas/tmax/log/slog",

*SVRGROUP
svg1         NODENAME = node1

*SERVER
tcpgw        SVGNAME = svg1,
              MIN = 1, MAX = 1,
              CPC = 5,
              SVRTYPE = CUSTOM_GATEWAY,
              CLOPT = "-- -r 168.126.185.131 -p 5060 -n 2"

*SERVICE
TCPGW        SVRNAME = tcpgw

```

4.2.2. TCPGW

<custom.h>

```

#ifndef _CUSTOM_H_
#define _CUSTOM_H_

```

```

/* ----- */
/*          Fixed structures and macros          */

#define MSG_MAGIC          "Tmax"
#define REMOTE_REQUEST     0
#define REMOTE_REPLY       1
#define SVC_NAME_LENGTH    20

/* このmsg_info_tは、開発者が再定義してはいけない構造体です。 */
typedef struct msg_info {
    char    svc[SVC_NAME_LENGTH];
    int     err;
    int     len;
    int     uid;
    int     flags;
    int     msgtype;
    int     channel_id;
} msg_info_t;

/* ----- */
/*          Modifiable structures and macros          */
/* このmsg_header_tとmsg_body_tは、開発者が再定義できる構造体です。 */

#define UID_FIELD 98
#define SVC_NAME_FIELD 92
#define UID_LENGTH 4
#define SVC_LENGTH 6

typedef struct msg_header {
    int     len;
} msg_header_t;

typedef struct msg_body {
    char     data[92];
    char     name[6];
    char     uid[4];
} msg_body_t;

#endif /* _CUSTOM_H_ */

```

<custom.c>

```

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

```

```

#include <sys/types.h>
#include <sys/timeb.h>
#include "custom.h"

/* msg_header_sizeとcomm_header_sizeは、TCPGWライブラリー内で使用されるため、
   定義する必要があります。 */
int      msg_header_size    = sizeof(msg_header_t);
int      comm_header_size = 0;

/* 下記の関数が不要な場合は、内部ロジックを実装する必要はありません。 */
/* しかし、TCPGWライブラリー内で使用されるため、定義する必要があります。 */
int get_channel_num(char *data)
{
    return -1;
}

int put_msg_complete(char *hp, char *data, msg_info_t *info)
{
    return 1;
}

int get_service_name(char *header, int err, char *svc)
{
    return -1;
}

int init_remote_info(char *myname, int mynumber, int num_channel, int key)
{
    return 1;
}

int remote_connected(int index, int addr, int type, int fd)
{
    return 1;
}

int get_msg_length(msg_header_t *hp)
{
    if (hp == NULL || hp->len == 0){
        return -1;
    }
    return ntohl(hp->len);
}

int get_msg_info(msg_header_t *hp, char *data, msg_info_t *info)
{
    info->flags = 0;

```



```

if ((info == NULL) || (hp == NULL) || (data == NULL))
    return -1;

info->len = ntohl(hp->len);
info->err = 0;
info->flags = 0;
/* info->uidを設定します。 */
memcpy((char *)&(info->uid), (char *)&data[UID_FIELD], UID_LENGTH);

strncpy(info->svc, (char *)&data[SVC_NAME_FIELD], SVC_LENGTH);
info->svc[SVC_LENGTH] = 0;

if (info->uid == 0) {
    return REMOTE_REQUEST;
}
else {
    return REMOTE_REPLY;
}
}

int put_msg_info(msg_header_t *hp, char *data, msg_info_t *info)
{
    int nSize = 0;

    if ((info == NULL) || (hp == NULL) || (data == NULL))
        return -1;

    /* hp->lenに入力される値は実データの長さです。 */
    hp->len = htonl(info->len);

    if (info->err) {
        printf("info->err = %d\n", info->err);
        return -1;
    }
    else
        /* リモート・ノードに要求する場合、必ずこのuidを */
        /* ライブラリー内部で設定した値で設定します。 */
        memcpy((char *)&data[UID_FIELD], (char *)&(info->uid),
            UID_LENGTH);

    memcpy((char *)&data[SVC_NAME_FIELD], info->svc, SVC_LENGTH);

    return (info->len + msg_header_size);
}

int remote_closed(int index, int type)

```

```

{
    return 1;
}

int prepare_shutdown(int code)
{
    return 1;
}

int set_service_timeout(int uid, char *header)
{
    return 1;
}

int chk_end_msg(int len, char *data)
{
    return len;
}

int outmsg_recovery(char *data, msg_info_t *info)
{
    return -1;
}

int inmsg_recovery(char *data, msg_info_t *info)
{
    return -1;
}

```

<Makefile>

```

#このMakefileはIBM 32bit用です。
#TARGETは、Tmax環境ファイルで定義したサーバー名と同じである必要があります。
TARGET    = tcpgw
APOBJS    = $(TARGET).o

#TCPGWを作成するため、libtcpgw.aまたはlibtcpgw.soをリンクさせます。
LIBS      = -ltcpgw -ltmaxgw
OBJS      = custom.o register.o

CFLAGS    = -q32 -O -I$(TMAXDIR) -D_DBG
LDFLAGS   = -brtl

APPDIR    = $(TMAXDIR)/appbin
LIBDIR    = $(TMAXDIR)/lib

.SUFFIXES : .c

```

```
.c.o:
    $(CC) $(CFLAGS) $(LDFLAGS) -c $<

$(TARGET): $(OBJS)
    $(CC) $(CFLAGS) $(LDFLAGS) -L$(LIBDIR) -o $(TARGET) $(OBJS) $(LIBS)
    mv $(TARGET) $(APPDIR)/.
    rm -f $(OBJS)

$(APOBJS): $(TARGET).c
    $(CC) $(CFLAGS) $(LDFLAGS) -c $(TARGET).c
```

4.2.3. リモート・ノード

<rserver.c>

下記の例題プログラムは、TCPGWの接続を受け、構文を送受信する機能のデーモン形式のプログラムです。テストを行うには、事前にプログラムが起動された状態でTCPGWが起動される必要があります。

```
#include <stdio.h>
#include <unistd.h>
#include <sys/time.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <errno.h>
#include "custom.h"

#define SERV_PORT 5060

void doit(int fd);

int main(int argc, char *argv[])
{
    int ret, i, len;
    int fd, childfd;
    pid_t childpid;

    ret = network_listen(SERV_PORT);
    if (ret < 0){
        return -1;
    }
}
```

```

    }else fd = ret;

    for (;;) {
        childfd = network_accept(fd);
        if (childfd < 0) {
            return -1;
        }
        if ( (childpid = fork()) == 0) {
            close(fd);
            doit(childfd);
            exit(0);
        }
        close(childfd);
    }
}

void doit(int fd)
{
    ssize_t n;
    int ret, i;
    msg_header_t *header;
    msg_body_t *body;

    header = (msg_header_t *)malloc(sizeof(msg_header_t));
    memset((char *)header, 0x00, sizeof(msg_header_t));
    body = (msg_body_t *)malloc(sizeof(msg_body_t));
    memset((char *)body, 0x00, sizeof(msg_body_t));

    readn2(fd, (char *)header, sizeof(msg_header_t));

    readn2(fd, (char *)body, ntohl(header->len));

    if (body->uid == 0) {
        printf("It's reply message\n");
    } else printf("It's request message\n");

    for (i = 0; i < sizeof(body->data); i++)
        body->data[i] = toupper(body->data[i]);

    header->len = htonl(sizeof(msg_body_t));

    writen2(fd, (char *)header, sizeof(msg_header_t));

    writen2(fd, (char *)body, sizeof(msg_body_t));
}

```

<network.c>

```
#include <stdio.h>
#include <unistd.h>
#include <sys/time.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <errno.h>

#ifndef INADDR_NONE
#define INADDR_NONE 0xffffffff /* should be in <netinet/in.h> */
#endif

/* ----- extern global variable ----- */
extern int errno;
extern char _svrname[30];

int network_listen(int portno)
{
    int fd, on;
    struct sockaddr_in serv_addr;

    if ((fd = socket(AF_INET, SOCK_STREAM, 0)) < 0){

        return(-1);
    }

    on = 1;
    if (setsockopt(fd, SOL_SOCKET, SO_REUSEADDR, (char *) &on,
        sizeof(on)) < 0) {
        printf("setsockopt error for SO_REUSEADDR");
    }

#ifdef SO_KEEPAIVE
    on = 1;
    if (setsockopt(fd, SOL_SOCKET, SO_KEEPAIVE, (char *) &on,
        sizeof(on)) < 0) {
        printf("setsockopt error for SO_KEEPAIVE");
    }

```

```

    }
#endif

    memset((char *) &serv_addr, 0, sizeof(serv_addr));
    serv_addr.sin_family      = AF_INET;
    serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    serv_addr.sin_port        = htons(portno);

    if (bind(fd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0) {

        close(fd);
        return(-2);
    }

    if (listen(fd, 250) < 0) {          close(fd);
        return(-3);
    }

    return(fd);
}

/* ----- server : client accept ----- */
int network_accept(int listenfd)
{
    int    fd, len;
    struct sockaddr_in cli_addr;

    len = sizeof(cli_addr);
    fd = accept(listenfd, (struct sockaddr *)&cli_addr, &len);
    if (fd < 0){

        return(-1); /* often errno=EINTR, if signal caught */
    }

    return(fd);
}

int readn2(int fd, char *ptr, int nbytes)
{
    int    nleft, nread;

    nleft = nbytes;
    while (nleft > 0) {
        nread = recv(fd, ptr, nleft, 0);
        if (nread < 0) {

```

```

        if (errno == EINTR)
            continue;
        else if (errno == EWOULDBLOCK)
            return (nbytes - nleft);
        return(nread);          /* error, return < 0 */
    } else if (nread == 0)
        break;                  /* EOF */
    nleft -= nread;
    ptr   += nread;
}

return (nbytes - nleft);        /* return >= 0 */
}

int writen2(int fd, char *ptr, int nbytes)
{
    int    nleft, nwritten;

    nleft = nbytes;
    while (nleft > 0) {
        nwritten = send(fd, ptr, nleft, 0);
        if (nwritten <= 0) {
            return(nwritten);    /* error */
        }

        nleft -= nwritten;
        ptr    += nwritten;
    }

    return(nbytes - nleft);
}

```

<Makefile>(サーバー用)

```

#このMakefileはCompac用です。
TARGET    = rServer
APOBJS    = $(TARGET).o

OBJS      = $(APOBJS) network.o

CFLAGS    = -D_DBG
.c.o:
    $(CC) $(CFLAGS) -c $<

all       : $(TARGET)
$(TARGET):$(OBJS)

```

```
$(CC) $(CFLAGS) -o $(TARGET) $(OBJS) $(LIBS)
rm -f $(OBJS)
```

4.2.4. クライアント

<cli_tcpgw.c>

下記の例題プログラムは、TCPGWにサービスを要求するTmaxクライアントです。TCPGWとリモート・ノードの接続が完了した後、プログラムを実行します。

```
#include <stdio.h>
#include <usrinc/atmi.h>
#include "custom.h"

int main(int argc, char **argv)
{
    int ret;
    msg_body_t *body;
    char *buf;
    long rlen;

    ret = tmaxreadenv("tmax.env", "TMAX");
    if (ret < 0) {
        printf("tmaxreadenv fail...[%s]\n", tpstrerror(tperrno));
    }

    ret = tpstart((TPSTART_T *)NULL);
    if (ret < 0) {
        printf("tpstart fail...[%s]\n", tpstrerror(tperrno));
        return -1;
    }

    buf = tpalloc("STRING", 0, 0);
    if (buf == NULL){
        printf("buf tpalloc fail...[%s]\n", tpstrerror(tperrno));
        tpend();
        return -1;
    }

    body = (msg_body_t *)buf;
    memcpy(body->data, argv[1], strlen(argv[1]));
    body->data[51] = 0;

    /* TCPGWサービスを呼び出します。 */
    ret = tpcall("TCPGW", buf, sizeof(msg_body_t), &buf, &rlen, 0);
    if (ret < 0){
```



```

    printf("tpcall fail...[%s]\n", tpstrerror(tperrno));
    tpfree((char *)buf);
    tpend();
    return -1;
}

body = (msg_body_t *)buf;
printf("return value = %s\n", body->data);
tpfree((char *)buf);
tpend();
}

```

4.3. 非ブロックTCPGW

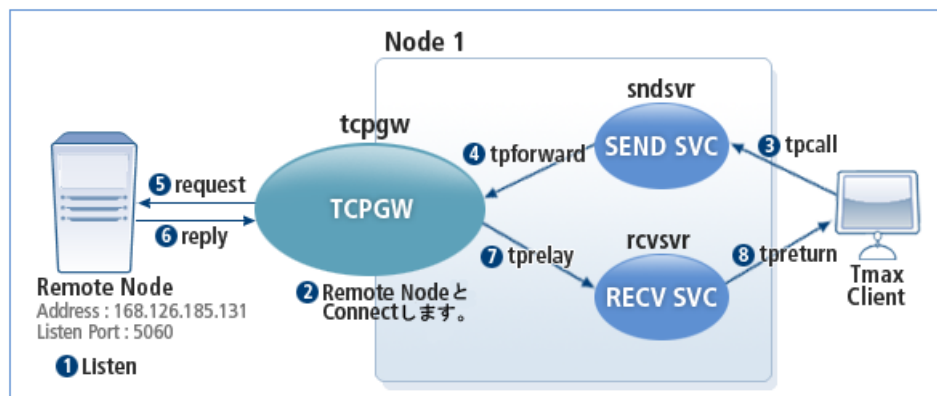
非ブロックTCPGW方式は、一般的に外部と通信する場合に多く使われます。その理由は、小数のプロセスでもシステム負荷が少なく、より多くのジョブが処理できるためです。ブロック型方式は、Tmaxクライアントまたはサービスで直接TCPGWを呼び出して応答を受けますが、この方式は、送信プロセスとTCPGWから応答を処理する受信プロセスを分離して処理します。

TCPGWを呼び出そうとするすべてのプロセスが先に送信プロセスを呼び出すと、送信プロセスはTCPGWを呼び出す前に事前作業を完了し、TCPGWにサービス制御を渡します(tpforward)。TCPGWは受け取ったサービスを処理する前に、Tmaxエンジンと接続されているチャンネルのブロック状態を解除し、リモート・ノードにサービスを送信します。リモート・ノードから応答が受信されたら、TCPGWは受信メッセージを処理する受信サービスを順序に従って検索し、受信プロセスにサービス制御を渡します。受信プロセスは、応答を処理した後、最初にサービス呼び出したプロセスに結果を渡します。

上記のようなプロセスでサービスされるため、実際にTCPGWを呼び出すサービスまたはtprelayを受けるサービスは非同期的に動作します。そのため、実行時間に対する負荷はほとんどありません。ただし、最初に送信サービスを呼び出すプロセスは応答が受信されるまで待機するため、ブロックされている状態になります。

下記の図は、リモート・ノードがサーバーになり、TCPGWがクライアントとして接続を確立する構造です。

[図 4.3] 非ブロックTCPGW方式



プログラムの構成は下記のとおりです。

区分	ファイル名
環境ファイル	tcpgw.m
TCPGW	custom.c, custom.h
リモート・ノード	rServer.c, network.c
サーバー	sndsvr.c, rcvsvr.c
クライアント	cli_tcpgw.c

4.3.1. 環境ファイル

<tcpgw.m>

```
*DOMAIN
res          SHMKEY = 88000,
              MINCLH = 1,
              MAXCLH = 1,
              TPORTNO = 8888

*NODE
node1        TMAXDIR="/home/tmax",
              APPDIR="/home/tmax/appbin"

*SVRGROUP
svg1         NODENAME = node1

*SERVER
tcpgw        SVGNAME = svg1,
              MIN = 1, MAX = 1,
              CPC = 5,
              SVRTYPE = CUSTOM_GATEWAY,
              CLOPT = "-- -r 168.126.185.131 -p 5060 -n 2 -H 9"
sndsvr       SVGNAME = svg1,
              MIN = 1, MAX = 1
rcvsvr       SVGNAME = svg1,
              MN = 1, MAX = 1
sndsvr       SVGNAME = svg1,
              MIN = 1, MAX = 1
rcvsvr       SVGNAME = svg1,
              MIN = 1, MAX = 1

*SERVICE
TCPGW        SVRNAME = tcpgw
```

```
SENDSVC      SVRNAME = sndsvr
RCVSVCSVC    SVRNAME = rcvsvr
```

4.3.2. TCPGW

<custom.c>

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <sys/types.h>
#include <sys/timeb.h>
#include "custom.h"

/* msg_header_sizeとcomm_header_sizeは、TCPGWライブラリー内で使用されるため、
  定義する必要があります。 */
int      msg_header_size    = sizeof(msg_header_t);
int      comm_header_size = 0;

/* 下記の関数が不要な場合は、内部ロジックを実装する必要はありません。 */
/* しかし、TCPGWライブラリー内で使用されるため、定義する必要があります。 */
int get_channel_num(char *data)
{
    return -1;
}

int put_msg_complete(char *hp, char *data, msg_info_t *info)
{
    return 1;
}

/* この関数でtprelayする関数名を設定します。 */
int get_service_name(char *header, int err, char *svc)
{
    /*内部的に保存しているユーザー・ヘッダーはretsvcnameなので、直ちにstrcpyします。*/
    strcpy(svc, header);
    svc[8] = 0;
    return 1;
}

int init_remote_info(char *myname, int mynumber, int num_channel, int key)
{
    return 1;
}
```

```

int remote_connected(int index, int addr, int type, int fd)
{
    return 1;
}

int get_msg_length(msg_header_t *hp)
{
    if (hp == NULL || hp->len == 0){
        return -1;
    }
    return ntohl(hp->len);
}

/* get_msg_infoとput_msg_infoは、同期クライアントTCPGWと同様に実装します。 */
int get_msg_info(msg_header_t *hp, char *data, msg_info_t *info)
{
    info->flags = 0;

    if ((info == NULL) || (hp == NULL) || (data == NULL))
        return -1;

    info->len = ntohl(hp->len);
    info->err = 0;
    info->flags = 0;
    /* info->uidを設定します。 */
    memcpy((char *)&(info->uid), (char *)&data[UID_FIELD], UID_LENGTH);

    strncpy(info->svc, (char *)&data[SVC_NAME_FIELD], SVC_LENGTH);
    info->svc[8] = 0;

    if (info->uid == 0) {
        return REMOTE_REQUEST;
    }
    else {
        return REMOTE_REPLY;
    }
}

int put_msg_info(msg_header_t *hp, char *data, msg_info_t *info)
{
    int nSize = 0;

    if ((info == NULL) || (hp == NULL) || (data == NULL))
        return -1;

    /* hp->lenに入力される値は、実データの長さです。 */

```

```

        hp->len = htonl(info->len);

        if (info->err) {
            printf("info->err = %d\n", info->err);
            return -1;
        }
        else
            /* リモート・ノードに要求する場合、必ずこのuidを */
            /* ライブラリー内部で設定した値で設定します。*/
            memcpy((char *)&data[UID_FIELD], (char *)&(info->uid), 4);

        memcpy((char *)&data[SVC_NAME_FIELD], info->svc, 6);

        return (info->len + 4);
    }

int remote_closed(int index, int type)
{
    return 1;
}

int prepare_shutdown(int code)
{
    return 1;
}

int set_service_timeout(int uid, char *header)
{
    return 1;
}

int chk_end_msg(int len, char *data)
{
    return len;
}

int outmsg_recovery(char *data, msg_info_t *info)
{
    return -1;
}

int inmsg_recovery(char *data, msg_info_t *info)
{
    return -1;
}

```

<Makefile>

「4.2. 同期インバウンドTCPGW」のMakefileの例と同じです。

4.3.3. リモート・ノード

<rServer.c>

下記の例題プログラムは、TCPGWの接続を受け、構文を送受信する機能のデーモン形式のプログラムです。テストを行うには、事前にプログラムが起動された状態でTCPGWが起動される必要があります。

```
#include <stdio.h>
#include <unistd.h>
#include <sys/time.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <errno.h>
#include "custom.h"

#define SERV_PORT 5060

void doit(int fd);

int main(int argc, char *argv[])
{
    int ret, i, len;
    int fd, childfd;
    pid_t childpid;

    ret = network_listen(SERV_PORT);
    if (ret < 0){
        return -1;
    }else fd = ret;

    for (;;) {
        childfd = network_accept(fd);
        if (childfd < 0){
            return -1;
        }
        if ( (childpid = fork()) == 0){
```

```

        close(fd);
        doit(childfd);
        exit(0);
    }
    close(childfd);
}

void doit(int fd)
{
    ssize_t n;
    int ret, i;
    msg_header_t *header;
    /* ユーザー・ヘッダー以外の部分のみ受けます。 */
    remote_body_t *body;
    int uid;

    header = (msg_header_t *)malloc(sizeof(msg_header_t));
    memset((char *)header, 0x00, sizeof(msg_header_t));
    body = (remote_body_t *)malloc(sizeof(remote_body_t));
    memset((char *)body, 0x00, sizeof(remote_body_t));

    readn2(fd, (char *)header, sizeof(msg_header_t));
    readn2(fd, (char *)body, ntohl(header->len));

    if (body->uid == 0){
        printf("It's reply message\n");
    }else printf("It's request message\n");

    for (i = 0; i < sizeof(body->data); i++)
        body->data[i] = toupper(body->data[i]);

    header->len = htonl(sizeof(remote_body_t));
    writen2(fd, (char *)header, sizeof(msg_header_t));
    writen2(fd, (char *)body, sizeof(remote_body_t));
}

```

<network.c>

「[4.2. 同期インバウンドTCPGW](#)」のnetwork.cの例と同じです。

4.3.4. サーバー

<sndsvr.c>

```
#include <string.h>
#include <stdio.h>
#include <usrinc/atmi.h>

SENDSVC(TPSVCINFO *msg)
{
    char *sndbuf;
    long len;

    printf("[%s] Service Started!\n", msg->name);

    len = (msg->len);
    printf("len is [%d]\n", len);

    if ((sndbuf = (char *)tpalloc("CARRAY", NULL, len)) == NULL) {
        printf("sndbuf alloc failed !\n");
        tpreturn(TPFAIL, -1, NULL, 0, 0);
    }

    memcpy(sndbuf, msg->data, msg->len);
    /* TPCGWのサービスをtpforwardで呼び出します。 */
    tpforward("TCPGW", (char *)sndbuf, len, TPNOREPLY);
}
```

<rcvsvr.c>

```
#include <stdio.h>
#include <string.h>
#include <usrinc/atmi.h>

/* TPCGWで動的にこのサービスにtprelayします。 */
RCVSVSVC(TPSVCINFO *msg)
{
    char *rcvbuf;
    long len;

    printf("[%s] Service Started!\n", msg->name);

    len = msg->len;
    rcvbuf = msg->data;
```



```

    if(tpurcode != 0) {
        printf("tpurcode is [%d] tperrmsg is [%s]\n", tpurcode,
            tpstrerror(tpurcode));
        tpreturn( TPFAIL, -1, (char *)rcvbuf, len, 0 );
    }
    tpreturn( TPSUCCESS, 0, (char *)rcvbuf, len, TPNOFLAGS );
}

```

4.3.5. クライアント

<cli_tcpgw.c>

下記の例題プログラムでは、TmaxクライアントとしてSENDSVCを呼び出すようになっています。SENDSVCを呼び出す前に、TCPGWでtprelayするサービスを構文ヘッダーに設定します。

```

#include <stdio.h>
#include <usrinc/atmi.h>
#include "../server/custom.h"

int main(int argc, char **argv)
{
    int ret;
    msg_body_t *body;
    char *buf;
    long rlen;

    ret = tmaxreadenv("tmax.env", "TMAX");
    if (ret < 0) {
        printf("tmaxreadenv fail...[%s]\n", tpstrerror(tperrno));
    }

    ret = tpstart((TPSTART_T *)NULL);
    if (ret < 0) {
        printf("tpstart fail...[%s]\n", tpstrerror(tperrno));
        return 0;
    }

    buf = tpalloc("STRING", 0, 0);
    if (buf == NULL){
        printf("buf tpalloc fail...[%s]\n", tpstrerror(tperrno));
        tpend();
        return 0;
    }
}

```

```

body = (msg_body_t *)buf;
memcpy(body->data, argv[1], strlen(argv[1]));
body->data[51] = 0;
memcpy(body->retsvcname, "RECVSVC", 9);
body->retsvcname[8] = 0;
memcpy(buf, (char *)body, sizeof(msg_body_t));

/* TCPGWを呼び出すサービスを呼び出します。*/
ret = tpcall("SENDSVC", buf, sizeof(msg_body_t), &buf, &rlen, 0);
if (ret < 0){
    printf("tpcall fail...[%s]\n", tpstrerror(tperrno));
    tpfree((char *)buf);
    tpend();
    return 0;
}

body = (msg_body_t *)buf;

printf("return value = %s\n", body->data);
tpfree((char *)buf);
tpend();
}

```

付録 A. TCPGWのエラーコード

TCPGWのエラーコードは下記のとおりです。

エラーコード	説明
TPECLOSE	リモート・ノードにサービスを要求した後、リモート・ノードとの接続が切断された場合に発生します
TPEINVAL	ゲートウェイ・ヘッダーを使用するとき、TCPGWを呼び出したデータ長がゲートウェイ・ヘッダーより小さい場合に発生します
TPENOENT	[-E]オプションを使用する場合、サービス要求ごとにリモート・ノードと接続してデータを転送しますが、同時に呼び出した数が[-n]オプションで指定したチャンネル数を超過した場合に発生します
TPENOREADY	リモート・ノードとの接続が切断され、使用可能なチャンネルが存在しない場合に発生します
TPEOS	TCPGW内部でメモリーが確保できない場合に発生します
TPEPROTO	tpforwardでTCPGWを呼び出すとき、TCPGWが同期型モードではなく、非同期型モードの場合に発生します。tpforwardとtprelayは、同期型方式である必要があります
[TPESVCERR]	put_msg_infoで0または負数を返す場合に発生します
TPESYSTEM	TCPGWでコード変換を使用するとき、要求したデータに対するマップ・ファイルがロードできない場合です。また、コード変換のエラー時にユーザー関数のput_msg_infoで負の数を返した場合、リモート・ノードにデータを転送するとエラーが発生します
TPETIME	リモート・ノードにサービスを要求した後、指定された時間内に応答がない場合です

索引

A

allow_connection(), 42
allow_connection_ipv6(), 43

C

chk_end_msg(), 49
chk_extpong_msg, 56
chk_pong_msg(), 54
custom.h
 msg_header_t, 27
 msg_info_t, 36

G

get_channel_num(), 46
get_extmsg_info(), 51
get_msg_info(), 46
get_msg_length(), 45
get_msg_security(), 60
get_service_name(), 48

I

init_remote_info(), 39
inmsg_recovery(), 50

O

outmsg_recovery(), 50

P

Pingメッセージ転送, 33
prepare_shutdown(), 48
put_extmsg_info(), 52
put_msg_complete(), 48
put_msg_info(), 47
put_msg_security(), 61

R

remote_closed(), 42
remote_connected(), 40
remote_connected_ipv6(), 40
reset_extping_msg, 58
reset_ping_msg(), 57

S

set_error_msg(), 59
set_extping_msg(), 54
set_ping_msg(), 53
set_service_timeout(), 49

T

TCP/IPゲートウェイ, 1
TCPGW, 1
TCPGWの動作構造, 1
Tmax環境構成
 CLOPT項目(TCPGWオプション), 18
 複数のリモート・ノードと接続クライアントTCPGW, 30
 再接続TCPGW, 29
 サーバー/クライアントTCPGW, 26
 サービス・ブロック型TCPGW, 27
 サービス非ブロック型TCPGW, 28
 リモート同期型と非同期型TCPGW, 29
Tmax環境構成(CLOPT項目)
 [-a], 22
 [-b], 24
 [-C], 21
 [-c], 21
 [-d], 22
 [-E], 22
 [-e], 22
 [-F], 20
 [-f], 20
 [-H], 22
 [-h], 22
 [-i], 24
 [-k], 20
 [-M], 19
 [-m], 19
 [-N], 19

[-n], 20
[-P], 19
[-p], 19
[-R], 18
[-r], 18
[-S], 21
[-t], 23
[-u], 23
[-w], 23
[-X CHANNEL_IRT], 25
[-X CLIENT_IPV6], 25
[-X CONN_NONBLOCK], 25
[-X CONN_RETRY], 25
[-X CONN_TIMEOUT], 25
[-X SERVER_IPV6], 25
[-x], 24
[-Y], 23
[-y], 24

あ

応答の多重処理, 10
アウトバウンドTCPGWの例, 67
アウトバウンド・サービス, 2
インバウンド・サービス, 1
エラーコード
 [TPESVCERR], 97
 TPECLOSE, 97
 TPEINVAL, 97
 TPENOENT, 97
 TPENOREADY, 97
 TPEOS, 97
 TPEPROTO, 97
 TPESYSTEM, 97
 TPETIME, 97

か

環境設定
 Tmax環境構成, 15
クライアントTCPGW, 9
ゲートウェイ・ヘッダー, 9
コード変換, 11

さ

再接続TCPGW, 29
サーバー/クライアントTCPGW, 26
サーバーTCPGW, 8
サービスブロック型TCPGWの例, 87
サービス・ブロック型TCPGW, 27
サービス非ブロック型TCPGW, 28

た

同期インバウンドTCPGWの例, 75
同期型TCPGW, 3
 サービス・ブロック型方式, 3
 サービス非ブロック型方式, 4
 リモート同期型呼び出し方式, 5
同期呼び出し方式, 2
チャンネル・バックアップの設定, 33

は

非同期型TCPGW, 6
 Tmaxからのサービス要求方式, 7
 リモートからのサービス要求方式, 7
非同期呼び出し方式, 2
複数のリモート・ノードと接続クライアントTCPGW, 30

や

ユーザー・ヘッダー, 31

ら

例
 同期インバウンドTCPGW, 75
 非ブロックTCPGW, 87
 アウトバウンドTCPGW, 67
リモート同期型と非同期型TCPGW, 29